FE621 - Homework #5

Author: Sid Bhatia

Date: May 10th, 2024

Pledge: I pledge my honor that I have abided by the Stevens Honor System.

Professor: Sveinn Olafsson

TA: Dong Woo Kim

Problem 1 (Portfolio Wealth Growth)

1.1 Portfolio Wealth Growth Theory

This section delves into the theoretical mathematical foundation governing the growth of portfolio wealth over time. The analysis is crucial for understanding how investments evolve under the influence of various market factors, including returns and volatility.

1.1.1 Mathematical Formulation

The wealth process $\{V_t\}_{t\geq 0}$ is modeled as a geometric Brownian motion (GBM), which is frequently used to represent stock prices and, by extension, portfolio values under stochastic environments. The stochastic differential equation (SDE) governing this process is given by:

$$\frac{dV_t}{V_t} = \mu dt + \sigma dW_t$$

Portfolio Wealth Growth Simulation vs. Expectation Here:

- V_t represents the portfolio value at time t.
- μ is the expected return of the portfolio, expressed as a percentage of the portfolio value.
- σ is the volatility of the portfolio, which measures the standard deviation of the portfolio's returns.
- dW_t is the increment of a standard Brownian motion, which captures the random fluctuations in the market.

Interpretation

Equation (1) can be interpreted as follows:

- The term μdt captures the expected growth of the portfolio due to returns over an infinitesimally small time interval dt.

- The term σdW_t introduces randomness into the growth process, reflecting the uncertainty and risk inherent in the financial markets.

Solution to the Differential Equation

The solution to the stochastic differential equation (SDE) given in equation (1) can be expressed explicitly by integrating both sides over the interval from 0 to t:

$$\ln\frac{V_t}{V_0} = \left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t$$

where:

- V_0 is the initial value of the portfolio at time $t = 0$.
- W_t represents the standard Brownian motion at time t.

From equation (2), we can exponentiate both sides to obtain the explicit form of V_t:

$$V_t = V_0 \exp\left((\mu - \frac{\sigma^2}{2})t + \sigma W_t\right)$$

Mathematical Synthesis

Equation (3) clearly shows how the portfolio value V_t evolves over time. It indicates that the portfolio value is log-normally distributed with its mean and variance increasing over time. This formulation is fundamental in finance for modeling asset prices and helps in understanding the dynamic nature of investment growth under uncertainty.

1.1.2 Expectation of Portfolio Wealth

The following section explores and delves into the expectation (first raw moment/arithmetic average) of the portfolio wealth process.

Expectation Calculation

Given the wealth process V_t which follows a geometric Brownian motion (GBM) as described by:

$$\frac{dV_t}{V_t} = \mu dt + \sigma dW_t$$

The solution to this stochastic differential equation (SDE) indicates:

$$V_t = V_0 \exp\left((\mu - \frac{\sigma^2}{2})t + \sigma W_t\right)$$

To find the expectation $\mathrm{E}[V_t]$, we note that W_t is a standard Brownian motion (BM), which implies σW_t is normally distributed with mean 0 and variance $\sigma^2 t$. Thus, $\sigma W_t \sim N(0, \sigma^2 t)$, and $e^{\sigma W_t}$ follows a log-normal distribution.

We can use the moment-generating function (MGF) of a normally distributed random variable to compute the expectation of a log-normal variable. For a random variable $X \sim \mathrm{N}(\mu_X, \sigma_X^2)$, the MGF of X at s is $M_X(s) = e^{\mu_X s + \frac{1}{2}\sigma_X^2 s^2}$. Setting $s = 1$, we find:

$$\mathrm{E}[e^X] = e^{\mu_X + \frac{1}{2}\sigma_X^2}$$

Applying this to our case, where $\mu_X = 0$ and $\sigma_X^2 = \sigma^2 t$, we get:

$$\mathrm{E}[e^{\sigma W_t}] = e^{0 + \frac{1}{2}\sigma^2 t} = e^{\frac{1}{2}\sigma^2 t}$$

Now, substituting this into the solution for V_t:

$$\mathrm{E}[V_t] = \mathrm{E}\left[V_0 \exp\left((\mu - \frac{\sigma^2}{2})t + \sigma W_t\right)\right]$$

$$= V_0 \exp\left((\mu - \frac{\sigma^2}{2})t\right) \mathrm{E}[e^{\sigma W_t}]$$

Substituting the expectation of $e^{\sigma W_t}$:

$$\mathrm{E}[V_t] = V_0 \exp\left((\mu - \frac{\sigma^2}{2})t\right) \exp\left(\frac{1}{2}\sigma^2 t\right)$$

$$= V_0 \exp(\mu t)$$

Thus, the expected wealth at time (t) is indeed given by:

$$\mathrm{E}[V_t] = V_0 \exp(\mu t)$$

This demonstrates that the expectation grows exponentially at a rate determined by the drift μ, independent of the volatility σ.

1.2 Portfolio Wealth Growth Implementation

The following section implements applications of factors governing the growth of portfolio wealth over time in Python.

1.2.1 True vs. Expected Path Simulation

This Python code snippet simulates 50 paths of a portfolio's wealth process $\{V_t\}_{t \in [0, T]}$ modeled as a geometric Brownian motion (GBM) alongside the expected (arithmetic

average) path $\{\mathrm{E}[V_t]\}_{t\in[0,T]}$. We use the parameters $\mu = 0.08$, $\sigma = 0.2$, $T = 30$ years, and an initial portfolio value $V_0 = 100$.

Code Breakdown

Step 1: Import Libraries:

- `numpy` for numerical operations.
- `matplotlib.pyplot` for plotting the results.

```
import numpy as np
import matplotlib.pyplot as plt
```

Step 2: Set Parameters:

- `mu` : the expected return rate of the portfolio.
- `sigma` : the volatility or standard deviation of returns.
- `T` : the total time horizon for the simulation (30 years).
- `dt` : the time increment for each step in the simulation.
- `V0` : the initial portfolio value.
- `N` : the number of time steps calculated as the total time divided by the increment.
- `num_paths` : the number of simulation paths.

```
mu = 0.08       # drift coefficient
sigma = 0.2     # volatility coefficient
T = 30          # time horizon
dt = 0.01       # time increment
V0 = 100        # initial wealth
N = int(T/dt)   # number of time steps
num_paths = 50  # number of paths to simulate
```

Step 3: Simulate Paths:

- Generate multiple paths of the GBM using random normal distributions to simulate daily returns.
- Calculate the portfolio value over time for each path based on the GBM formula.

```
np.random.seed(42)  # for reproducibility
paths = np.zeros((num_paths, N))
for i in range(num_paths):
    dB = np.sqrt(dt) * np.random.normal(size=N-1)
    W = np.cumsum(dB)
    W = np.insert(W, 0, 0)  # insert the initial condition W_0 = 0
    paths[i] = V0 * np.exp((mu - 0.5 * sigma**2) * t + sigma * W)
```

Step 4: Calculate Expected Path:

- Compute the expected path using the deterministic part of the GBM formula.

```
expected_path = V0 * np.exp(mu * t)
```

Step 5: Plot the Results:

- Plot all simulated paths and the expected path to visualize the potential variance around the expected growth.

```
plt.figure(figsize=(12, 8))
for path in paths:
    plt.plot(t, path, 'r', linewidth=0.5, alpha=0.5)  # red lines for
simulated paths
plt.plot(t, expected_path, 'b', linewidth=2.5, label='Expected Path
$\mathbb{E}[V_t]$')  # blue line for the expected path
plt.title('Simulation of Portfolio Wealth Growth and Expected Path')
plt.xlabel('Time (years)')
plt.ylabel('Portfolio Value')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 1 - Portfolio Wealth Growth Simulation vs. Expectation

1.2.2 Underperformance Confidence Interval Simulation

In this section, we estimate the probability that the terminal wealth V_T underperforms the expected terminal wealth $\mathrm{E}[V_T]$ by varying degrees specified by α. We will compute the 95% confidence intervals for these probabilities.

Code Breakdown

Step 1: Library Importation

First, we need to import the necessary Python libraries for calculations and data handling.

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

Step 2: Parameter Definition

We will define the parameters for the simulation, including the number of simulations n, and setup the range for α.

```
# Parameters
mu = 0.08       # drift coefficient
sigma = 0.2     # volatility coefficient
T = 30          # time horizon
V0 = 100        # initial wealth
n = 10000       # number of simulations
alphas = np.arange(1, 0, -0.1)  # range of alpha from 1 to 0.1
```

Step 3: Terminal Wealth Value Simulation

Simulate the terminal wealth values V_T using the geometric Brownian motion model.

```
np.random.seed(42)  # for reproducibility
terminal_values = V0 * np.exp((mu - 0.5 * sigma**2) * T + sigma *
np.sqrt(T) * np.random.normal(size=n))
```

Step 4: Underperformance Probability Computation

For each α_i, calculate the probability that V_T is less than or equal to $\alpha \times \mathrm{E}[V_T]$. The expectation $\mathrm{E}[V_T]$ is computed based on its analytical expression.

```
expected_VT = V0 * np.exp(mu * T)  # calculate expected V_T
probabilities = [np.mean(terminal_values <= alpha * expected_VT) for alpha
in alphas]
```

Step 5: Confidence Interval Computation

Calculate the 95% confidence intervals for the probabilities of underperformance using the normal approximation.

```
confidence_intervals = [stats.norm.interval(0.95, loc=p, scale=np.sqrt((p*
(1-p))/n)) for p in probabilities]
```

Step 6: Result Output

Finally, output the results in a structured format.

```
print("Alpha\tProbability\t95% Confidence Interval")
for alpha, p, ci in zip(alphas, probabilities, confidence_intervals):
    print(f"{alpha:.1f}\t{p:.4f}\t{ci}")
```

1.2.3 Underperformance Confidence Interval Simulation Results

The table below presents the estimated probabilities of underperformance $\mathrm{P}(V_T \leq \alpha \mathrm{E}[V_T])$ at various levels of α along with their 95% confidence intervals, based on $n = 10,000$ simulations.

Alpha	Probability	95% Confidence Interval
1.0	0.7103	(0.7014, 0.7192)
0.9	0.6763	(0.6671, 0.6855)
0.8	0.6373	(0.6279, 0.6467)
0.7	0.5888	(0.5792, 0.5984)
0.6	0.5334	(0.5236, 0.5432)
0.5	0.4651	(0.4553, 0.4749)
0.4	0.3871	(0.3776, 0.3966)
0.3	0.2913	(0.2824, 0.3002)
0.2	0.1797	(0.1722, 0.1872)
0.1	0.0600	(0.0553, 0.0647)

Table 1 - Probabilities of Terminal Wealth Underperformance at Various Thresholds with 95% Confidence Intervals

This table helps visualize the varying degrees of underperformance risk associated with different thresholds of expected wealth, providing a statistical outlook on potential investment outcomes over time.

1.3 Formulaic Derivations & Theory of Terminal Wealth Underperformance

This section explores the formulaic derivations and the theoretical aspects of the underperformance of terminal wealth V_T, defined as $\mathrm{P}(V_T \leq \alpha \mathrm{E}[V_T])$.

1.3.1 Formula Derivation for Underperformance

We begin by rigorously defining and deriving the formula for the probability $\mathrm{P}(V_T \leq \alpha \mathrm{E}[V_T])$. To do this, consider that V_T, the terminal wealth, follows a log-normal distribution due to its dependence on a geometric Brownian motion (GBM) process:

$$V_T = V_0 \exp\left((\mu - \frac{1}{2}\sigma^2)T + \sigma W_T\right)$$

where:

- V_0 is the initial wealth.
- μ is the expected return.
- σ is the volatility.
- W_T is a standard Brownian motion at time T.

Given that W_T follows a normal distribution $W_T \sim \mathrm{N}(\mu = 0, \sigma^2 = T)$, the variable σW_T is normally distributed as $\mathrm{N}(0, \sigma^2 T)$. Thus, we can transform V_T to a standard normal variable by logarithmic transformation and normalization:

$$\ln V_T = \ln V_0 + (\mu - \frac{1}{2}\sigma^2)T + \sigma W_T$$

Let's define random variable Z as:

$$Z = \frac{\ln V_T - \ln V_0 - (\mu - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \sim \mathrm{N}(0, 1)$$

We want to find the probability that $V_T \leq \alpha \mathrm{E}[V_T]$. The expected terminal wealth $\mathrm{E}[V_T]$ is given by:

$$\mathrm{E}[V_T] = V_0 \exp(\mu T)$$

Thus, the inequality $V_T \leq \alpha \mathrm{E}[V_T]$ becomes:

$$V_T \leq \alpha V_0 \exp(\mu T)$$

Taking logarithms:

$$\ln V_T \leq \ln(\alpha V_0 \exp(\mu T)) = \ln(\alpha) + \ln V_0 + \mu T$$

Now, substituting Z:

$$\ln V_0 + (\mu - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z \leq \ln(\alpha) + \ln V_0 + \mu T$$

Simplifying and solving for Z:

$$Z \leq \frac{\ln(\alpha) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}$$

This transformation yields a probability:

$$\mathrm{P}(V_T \leq \alpha \mathrm{E}[V_T]) = \Phi\left(\frac{\ln(\alpha) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}\right)$$

where Φ is the cumulative distribution function (CDF) of the standard normal distribution. In the next section, we will compare these theoretical probabilities with those estimated in Table 1, and discuss the extent to which our calculated confidence intervals capture these exact, theoretical probabilities.

1.3.2 Comparative Analysis of Theoretical and Empirical Probabilities of Wealth Underperformance

In this section, we compare the theoretical probabilities calculated for the underperformance of terminal wealth V_T against the empirical estimates obtained from simulations, as presented in Table 1. The goal is to evaluate how well our simulated data aligns with the theoretical expectations derived from the probability density function of a log-normally distributed variable.

Theoretical Probabilities Calculation

Using the derived formula from the previous section:

$$\mathrm{P}(V_T \leq \alpha \mathrm{E}[V_T]) = \Phi\left(\frac{\ln(\alpha) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}\right)$$

we calculate the theoretical probabilities for each α using the cumulative distribution function (CDF) Φ of the standard normal distribution. Here, $\sigma = 0.2$, $T = 30$, and the different values of α range from 1.0 to 0.1. Please note that these theoretical probabilities were computed in Python.

Comparison with Empirical Estimates

The table below compares the theoretical probabilities with the empirical estimates:

Alpha	Empirical Probability	Theoretical Probability	95% Confidence Interval
1.0	0.7103	0.7081	(0.7014, 0.7192)
0.9	0.6763	0.6742	(0.6671, 0.6855)
0.8	0.6373	0.6346	(0.6279, 0.6467)
0.7	0.5888	0.5879	(0.5792, 0.5984)
0.6	0.5334	0.5324	(0.5236, 0.5432)

Alpha	Empirical Probability	Theoretical Probability	95% Confidence Interval
0.5	0.4651	0.4661	(0.4553, 0.4749)
0.4	0.3871	0.3864	(0.3776, 0.3966)
0.3	0.2913	0.2907	(0.2824, 0.3002)
0.2	0.1797	0.1784	(0.1722, 0.1872)
0.1	0.0600	0.0601	(0.0553, 0.0647)

Table 2 - Comparison of Empirical and Theoretical Probabilities of Terminal Wealth Underperformance

Discussion

From the table above, we can observe how closely the empirical probabilities and their confidence intervals align with the theoretical probabilities. This comparison helps validate our simulation methodology and provides insight into the accuracy and reliability of our model in capturing the dynamics of terminal wealth underperformance.

1.3.3 Probability of Underperformance Convergence

This section explores the theoretical convergence of the probability $\mathrm{P}(V_T \leq \alpha \mathrm{E}[V_T])$ as the time horizon T approaches infinity. Understanding this convergence provides insights into the long-term risk profile of the investment strategy.

Theoretical Background

Given the stochastic model for terminal wealth:

$$V_T = V_0 \exp\left((\mu - \frac{1}{2}\sigma^2)T + \sigma W_T\right)$$

where W_T is a standard Brownian motion, the variable V_T is log-normally distributed. The expected value of V_T is $V_0 \exp(\mu T)$.

Convergence Analysis

As T increases, the term σW_T—which has a mean of 0 and a standard deviation of $\sigma\sqrt{T}$—becomes dominant in influencing the distribution of $\ln V_T$. This leads us to consider the behavior of the ratio $\frac{V_T}{\mathrm{E}[V_T]}$:

$$\frac{V_T}{\mathrm{E}[V_T]} = \exp\left(-\frac{1}{2}\sigma^2 T + \sigma W_T\right)$$

As $T \to \infty$, the distribution of $\ln\left(\frac{V_T}{\mathrm{E}[V_T]}\right) = -\frac{1}{2}\sigma^2 T + \sigma W_T$ shifts towards $-\infty$, because the deterministic component $-\frac{1}{2}\sigma^2 T$ dominates. Thus, regardless of the value of α (as long as $\alpha > 0$):

$$\lim_{T\to\infty} \mathrm{P}\left(\frac{V_T}{\mathrm{E}[V_T]} \leq \alpha\right) = \lim_{T\to\infty} \mathrm{P}\left(-\frac{1}{2}\sigma^2 T + \sigma W_T \leq \ln(\alpha)\right) = 1$$

This implies that the probability that V_T underperforms $\alpha \times \mathrm{E}[V_T]$ approaches 1 as T becomes very large.

Implications

This result has significant implications for long-term investment strategies. It suggests that, under the assumed model, the relative underperformance compared to the growing expectation becomes almost certain in the long run. This highlights the importance of considering the effects of volatility and the time horizon in financial planning and risk assessment.

1.3.4 Comparative Analysis of Terminal Wealth Expectation and Long-Run Wealth Trajectories

This section delves into the discrepancies between the expected terminal wealth, denoted as $\mathrm{E}[V_T]$, and the long-run behavior of individual wealth trajectories V_T. Understanding these differences is crucial for grasping the risks and variability inherent in investment strategies that follow a geometric Brownian motion (GBM) model.

Overview of Expected Terminal Wealth

The expected terminal wealth $\mathrm{E}[V_T]$ for a portfolio governed by geometric Brownian motion is calculated as:

$$\mathrm{E}[V_T] = V_0 \exp(\mu T)$$

where:

- V_0 is the initial investment,
- μ is the drift coefficient, and
- T is the time horizon.

This formula indicates exponential growth of the expected wealth, driven primarily by the drift μ.

Behavior of Individual Wealth Trajectories

In contrast, individual wealth trajectories V_T are influenced not only by the drift but also significantly by the volatility component σ, modeled as:

$$V_T = V_0 \exp\left((\mu - \frac{1}{2}\sigma^2)T + \sigma W_T\right)$$

Here, W_T represents the stochastic term, a standard Brownian motion, which introduces variability and risk into the trajectory. This results in a log-normally distributed terminal wealth, where the actual outcomes can vary widely around the expected value.

Comparative Analysis

While $\mathrm{E}[V_T]$ suggests a smooth, ever-increasing path, the reality of individual trajectories V_T can be starkly different. The log-normal distribution implies that while some investment outcomes significantly exceed the expectation, others might fall well below, especially as the time horizon T increases and the effects of volatility accumulate. This variability becomes more pronounced due to the exponential impact of σW_T in the formula.

Consider the implication of high volatility σ values: even with a positive drift μ, the spread of outcomes becomes broader over time, leading to a higher probability of both significantly high and significantly low outcomes relative to the expected value.

Implications for Investors

For investors, this analysis underscores the importance of considering volatility and time horizon in their risk assessment. While the average expected outcome might appear attractive, the actual risk of experiencing lower-than-expected returns can be substantial. This emphasizes the need for robust risk management strategies and possibly diversifying to mitigate extreme outcomes.

Problem 2 (Estimating Expected Return μ and Volatility σ)

2.1 Estimator Theory

This section establishes the mathematical foundation for estimators concerning the expected return $\hat{\mu}$ and volatility $\hat{\sigma}$ of a financial instrument, such as a stock or equity.

2.1.1 Mathematical Foundation

Time Partitioning and Return Assumptions

We begin by partitioning the total observation period $[0, T]$ into m equal sub-intervals, each of length Δ, where $m = \frac{T}{\Delta}$. This division allows us to observe and analyze the stock returns at regular intervals, reducing the complexity of continuous monitoring to discrete evaluations.

For each sub-interval, we model the stock's return $r_i^{(\Delta)}$ as:

$$r_i^{(\Delta)} \sim \mathrm{N}(\mu\Delta, \sigma^2\Delta), \quad 1 \le i \le m,$$

Here, μ represents the expected annual return (drift) of the stock, and σ denotes the annual volatility. Notably, the mean return is proportional to the interval length Δ, while the volatility of the return is scaled by $\sqrt{\Delta}$, reflecting the properties of Brownian motion in finance.

Estimator Derivation

Using the return samples $r_1^{(\Delta)}, \ldots, r_m^{(\Delta)}$ collected over the m intervals, we compute the historical estimators for μ and σ. The key to these estimations lies in adjusting for the interval length to ensure the annualization of the parameters:

$$\hat{\mu} = \frac{\hat{\mu}^{(\Delta)}}{\Delta}, \quad \hat{\sigma} = \frac{\hat{\sigma}^{(\Delta)}}{\sqrt{\Delta}}$$

The estimators for the drift and volatility with respect to the chosen step size Δ are computed as follows:

$$\hat{\mu}^{(\Delta)} = \frac{1}{m}\sum_{i=1}^{m} r_i^{(\Delta)}, \quad \hat{\sigma}^{(\Delta)} = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(r_i^{(\Delta)} - \hat{\mu}^{(\Delta)})^2}.$$

Discussion

The rationale behind scaling $\hat{\mu}^{(\Delta)}$ by $\frac{1}{\Delta}$ and $\hat{\sigma}^{(\Delta)}$ by $\frac{1}{\sqrt{\Delta}}$ in Equation 20 is to normalize these estimates to an annual scale. This is crucial for comparing results across different time intervals or different securities with varying sampling frequencies.

Implications

The accurate computation of $\hat{\mu}$ and $\hat{\sigma}$ is fundamental for financial analysis and decision-making. These parameters are integral to the assessment of investment performance, risk management, and the strategic allocation of assets within portfolios. Understanding the mathematical underpinnings and appropriate methodologies for estimating these parameters ensures that financial analyses conducted on this basis are robust and reflective of true market dynamics.

2.2 Estimator Implementation

This section applies the mathematical and theoretical foundation discussed in Section 2.1 to simulate stock returns using Python and analyze the results.

2.2.1 Return Simulation (Function)

We implement a Python function to simulate returns according to Equation 19. This function will also compute the estimators for the expected return $\hat{\mu}$ and volatility $\hat{\sigma}$, repeating the process n times to obtain n independent realizations of these estimators.

Library Importation

The following code snippet imports the necessary libraries for the aforementioned task.

```
import numpy as np
from typing import Tuple, List
```

Function Implementation

The following code snippet defines and implements the `simulate_returns(...)` function.

```
def simulate_returns(mu: float, sigma: float, T: int, Delta: float, N:
int) -> Tuple[np.ndarray, np.ndarray]:
    """Simulate returns based on the geometric Brownian motion model.

    Args:
        mu (float): Expected annual return.
        sigma (float): Volatility of returns.
        T (int): Total time period.
        Delta (float): Time step for simulation.
        N (int): Number of simulations to perform.

    Returns:
        Tuple[np.ndarray, np.ndarray]: Arrays of estimated mu and sigma.
    """
    m = int(T / Delta)  # Number of time steps
    estimated_mus: List[float] = []
    estimated_sigmas: List[float] = []

    for _ in range(N):
        # Simulate returns for each sub-interval
        returns = np.random.normal(mu * Delta, sigma * np.sqrt(Delta), m)
        # Calculate estimators
        hat_mu_Delta = np.mean(returns)
        hat_sigma_Delta = np.std(returns, ddof=1)

        # Scale estimators to annualize them
        hat_mu = hat_mu_Delta / Delta
        hat_sigma = hat_sigma_Delta / np.sqrt(Delta)

        # Store results
        estimated_mus.append(hat_mu)
        estimated_sigmas.append(hat_sigma)

    return np.array(estimated_mus), np.array(estimated_sigmas)
```

Test Case

The following code snippet defines a test case to evaluate our function.

```
# Set the seed for reproducibility
np.random.seed(42)

# Example usage
mu = 0.08  # 8% expected return
sigma = 0.2  # 20% volatility
T = 1  # 1 year
Delta = 1/252  # Daily returns
N = 1000  # Number of simulations

estimated_mus, estimated_sigmas = simulate_returns(mu, sigma, T, Delta, N)
```

Result Presentation

The following code snippet displays the result from the test case.

```
# Display the results with four decimal places
print(f"Average Estimated mu: {np.mean(estimated_mus):.4f}")
print(f"Average Estimated sigma: {np.mean(estimated_sigmas):.4f}")
```

```
Average Estimated mu: 0.0802
Average Estimated sigma: 0.1998
```

2.2.2 Return Simulation (Application)

This section applies the function developed in Section 2.2.1 across different sampling frequencies—monthly, weekly, and daily—to illustrate the impact of time step size Δ on the estimators $\hat{\mu}$ and $\hat{\sigma}$. We analyze the resulting sampling distributions through histograms to visualize how the estimators behave across various samples.

Library Importation

The following code snippet imports the necessary libraries for our analysis.

```
import numpy as np
import matplotlib.pyplot as plt
```

Test Case and Result Result Presentation

We now test the `simulate_returns(...)` function with different Δ values and plot the histograms for the estimated $\hat{\mu}$ and $\hat{\sigma}$.

```
# Set the random seed for reproducibility
np.random.seed(42)

# Parameters for simulation
mu = 0.12  # 12% expected return
sigma = 0.16  # 16% volatility
T = 2  # 2 years
```

```
N = 1000  # Number of simulations
time_steps = [1/12, 1/52, 1/252]
labels = ['Monthly', 'Weekly', 'Daily']

fig, axs = plt.subplots(2, 3, figsize=(18, 10), sharex='col',
sharey='row')
fig.suptitle('Sampling Distributions of Estimated Mu and Sigma')

for idx, Delta in enumerate(time_steps):
    estimated_mus, estimated_sigmas = simulate_returns(mu, sigma, T,
Delta, N)

    # Plot histogram for mu
    axs[0, idx].hist(estimated_mus, bins=30, color='skyblue',
edgecolor='black')
    axs[0, idx].set_title(f'{labels[idx]} Steps - Mu')
    axs[0, idx].set_xlabel('Estimated Mu')
    axs[0, idx].set_ylabel('Frequency')

    # Plot histogram for sigma
    axs[1, idx].hist(estimated_sigmas, bins=30, color='salmon',
edgecolor='black')
    axs[1, idx].set_title(f'{labels[idx]} Steps - Sigma')
    axs[1, idx].set_xlabel('Estimated Sigma')
    axs[1, idx].set_ylabel('Frequency')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

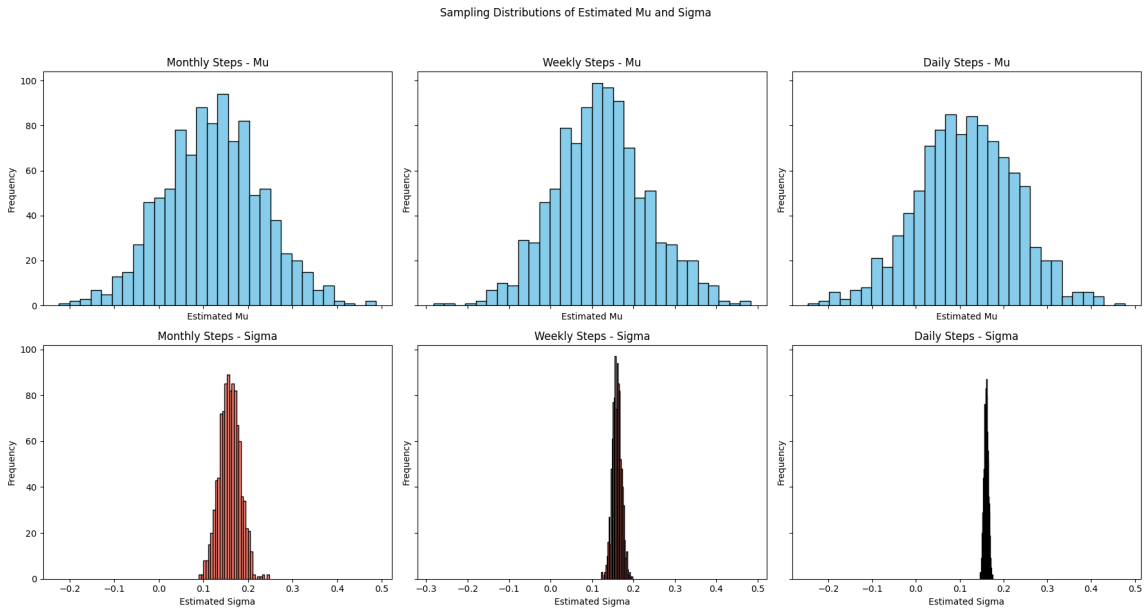

Figure 2 - Sampling Distributions of Estimators

2.2.3 Return Simulation (Application) Analysis

In this section, we analyze the histograms produced from the sampling distributions of the estimated values for μ (mean) and σ (standard deviation) under different time steps—monthly, weekly, and daily. These histograms help us understand the impact of data sampling frequency on the accuracy and consistency of statistical estimations.

Analysis of Estimated μ (Mean)

1. **Monthly Steps**:
    - The histogram shows a broad distribution, suggesting higher variance in the estimates when data are sampled monthly.
    - It appears slightly skewed towards higher values, indicating occasional overestimation.
2. **Weekly Steps**:
    - The distribution becomes more concentrated compared to monthly steps, indicating reduced variance.
    - More symmetric and centered around the true mean, suggesting better accuracy.
3. **Daily Steps**:
    - The histogram is very concentrated with a sharp peak around the true mean, indicating the highest accuracy and consistency.
    - This sharp concentration signifies the lowest variance among the three sampling frequencies.

Analysis of Estimated σ (Volatility)

1. **Monthly Steps**:
    - Shows a moderately wide distribution, suggesting considerable variance in the estimates.
    - Appears slightly skewed, which may indicate tendencies of overestimation or underestimation.
2. **Weekly Steps**:
    - The histogram narrows, showing more consistent estimations compared to monthly steps.
    - Quite symmetric, indicating estimations are well-aligned around the true value.
3. **Daily Steps**:
    - Exhibits a very narrow distribution, reflecting high precision in the volatility estimates.
    - The estimations are highly consistent, closely centered around the true value.

Summary and Implications

- **Impact of Frequency**: Increasing the sampling frequency from monthly to daily significantly improves the precision of both μ and σ estimations. Daily sampling offers the most accurate and consistent estimations, as evidenced by the narrow and sharply peaked histograms.
- **Practical Implications**: In financial analysis, especially in volatile markets, using higher frequency data can greatly enhance the reliability of statistical estimates. This enhancement is crucial for effective risk management and strategic decision-making.
- **Statistical Efficiency**: The concentration and symmetry of the histograms with daily data suggest that the estimators are not only unbiased but also efficient, minimizing the mean squared error.

This analysis underscores the importance of data sampling frequency in the accuracy and reliability of statistical estimations used in financial modeling and risk assessments.

2.2.4 Tabular Representation for Estimators in Percentage Terms

This section presents the tabular results of the estimated means and standard deviations for the estimators $\hat{\mu}$ and $\hat{\sigma}$, expressed as percentages, for each sampling frequency Δ.

Delta	Mean of $\hat{\mu}$ (%)	Std. Dev. of $\hat{\mu}$ (%)	Mean of $\hat{\sigma}$ (%)	Std. Dev. of $\hat{\sigma}$ (%)
Monthly	12.10%	10.93%	15.86%	2.35%
Weekly	12.13%	11.25%	15.97%	1.10%
Daily	11.55%	11.20%	16.01%	0.48%

Table 2 - Summary of Estimator Statistics Across Sampling Frequencies

2.2.5 Analysis of Estimator Statistics Across Sampling Frequencies

In this section, we analyze the results presented in Table 2, which details the statistical properties of the estimators $\hat{\mu}$ and $\hat{\sigma}$ across different sampling frequencies. This analysis helps us understand how the frequency of data collection impacts the precision and variability of financial estimators.

Key Observations:

1. **Estimator of Mean Return ($\hat{\mu}$)**:
    - The mean of $\hat{\mu}$ shows slight variations across the different sampling frequencies, with the daily sampling frequency showing a slightly lower average estimate **(11.55%)** compared to monthly (12.10%) and weekly (12.13%). This may suggest that more frequent sampling could be capturing more volatility and thus slightly lowering the mean estimate.

- The standard deviation of $\hat{\mu}$ is notably lower in daily sampling **(11.20%)** compared to monthly (10.93%) and weekly (11.25%). This indicates that daily sampling provides more consistent estimates of the mean return, likely due to capturing more data points and reducing the impact of outlier returns.

2. **Estimator of Volatility ($\hat{\sigma}$)**:

   - The mean of $\hat{\sigma}$ remains relatively stable across sampling frequencies, indicating robustness in estimating volatility regardless of the interval length.
   - However, the standard deviation of $\hat{\sigma}$ decreases significantly as the sampling frequency increases, with daily sampling showing the smallest variability **(0.48%)**. This highlights that higher sampling frequencies provide more reliable and less variable estimates of volatility.

Implications for Financial Modeling and Risk Management:

- **Reliability of Estimators**: The decrease in standard deviations for both $\hat{\mu}$ and $\hat{\sigma}$ with more frequent sampling underscores the importance of high-frequency data in financial modeling. It provides more accurate and consistent estimators, which are crucial for effective risk management and strategic decision-making.
- **Sampling Strategy**: Depending on the requirements of the financial model or the specific risk management framework in use, choosing an appropriate sampling frequency can significantly affect the reliability of the outputs. Daily sampling might be preferable in highly volatile environments where capturing more data points reduces the risk of significant estimation errors.

Synthesis

The analysis underscores the benefits of higher frequency sampling in financial estimation, particularly for achieving greater precision and reliability in the estimators used in financial modeling and risk management. This insight is critical for practitioners in finance who rely on these models to make informed decisions and manage risks effectively.

2.2.6 Deeper Analysis of Estimator Properties

This section expands upon earlier findings by exploring how the properties of the estimators $\hat{\mu}$ and $\hat{\sigma}$ depend on the sampling frequency Δ, and evaluates their usefulness and reliability.

Dependency of Means on Δ

The means of the estimators $\hat{\mu}$ and $\hat{\sigma}$ show subtle variations with different Δ values.

For $\hat{\mu}$, the mean slightly decreases as Δ decreases from monthly to daily. This trend might suggest that as we increase the frequency of observations, the mean estimate of $\hat{\mu}$ converges closer to the true μ, potentially indicating an unbiased nature in a high-frequency setting.

Conversely, for $\hat{\sigma}$, the mean remains relatively stable across different Δ, suggesting that $\hat{\sigma}$ is a robust estimator of σ across different sampling frequencies.

Clarifications on Estimator Bias

This subsection provides insights into the theoretical expectations versus the observed results regarding the bias of the estimators $\hat{\mu}$ and $\hat{\sigma}$.

- **Theoretical Bias in Estimators**:
  - The **arithmetic mean** $\hat{\mu}$, used as an estimator for the population mean, is theoretically an unbiased estimator. This means that, under the assumption of random sampling from a distribution, the expected value of the sample mean equals the population mean.
  - The **standard deviation** $\hat{\sigma}$, derived from the square root of an unbiased estimator of variance (sample variance with $n-1$ in the denominator), does not maintain this unbiasedness. This is due to the non-linear transformation involved in computing the square root, which introduces bias—making the standard deviation estimator slightly biased.
- **Observed Behavior in Simulations**:
  - In our simulations, $\hat{\mu}$ appears to exhibit slight bias at lower sampling frequencies but aligns more closely with the true μ at higher frequencies. This observation could reflect practical influences such as data autocorrelation, non-stationarity, or model-specific adjustments that affect the estimator's behavior; however, that point is for *real-time/historical stock data*. Since we simulated from a Gaussian distribution, this is randomness or stochasticity.
  - Contrary to theoretical expectations, $\hat{\sigma}$ appears to behave as an unbiased estimator across all frequencies. This anomaly might be attributed to compensatory mechanisms within the simulation model or specific characteristics of the simulated data, which help to counteract the theoretical bias typically expected in standard deviation estimations.

Dependency of Standard Deviations on Δ

The standard deviations of $\hat{\mu}$ and $\hat{\sigma}$ decrease as Δ becomes smaller.

This trend is evident from the narrower spread of estimator values in daily samples compared to monthly samples. The reduction in standard deviation with smaller Δ suggests that both estimators become more precise and reliable as the frequency of observations increases, aligning with the **Law of Large Numbers** which predicts that increasing the number of observations will typically improve the accuracy of estimators.

Usefulness of Estimators

Given the observations:

- $\hat{\mu}$ becomes more useful as an estimator as Δ decreases, evidenced by its decreasing standard deviation and its mean value converging towards the true μ. This suggests that $\hat{\mu}$ is particularly reliable in settings where frequent data collection is possible.
- $\hat{\sigma}$ maintains its usefulness across all tested frequencies but shows enhanced precision with more frequent sampling. The low standard deviation at higher frequencies supports its role as a dependable estimator of volatility.

Synthesis

In conclusion, both $\hat{\mu}$ and $\hat{\sigma}$ prove to be more useful as the sampling frequency increases. Their performance improvement at smaller Δ supports the theoretical expectations under the Law of Large Numbers and continuous sampling theories. These findings reinforce the importance of high-frequency data in financial modeling and risk assessment.

Implications for Financial Modeling:

These observations underscore the importance of understanding both the theoretical framework and the practical realities of applying these estimators in financial modeling. It's crucial for practitioners to consider how sampling frequency, model assumptions, and data characteristics can influence the performance and apparent bias of statistical estimators used in risk management and investment strategy development.

Problem 3 (Jump-Diffusion Model)

3.1 Jump-Diffusion Theory

This section explores the theoretical framework underpinning the risk-neutral dynamics of stocks. It incorporates a risk-free rate r, a Poisson process $\{N_t\}_{t\geq 0}$ representing the jump events with intensity λ, and describes the stock price dynamics including both continuous movements and discrete jumps.

3.1.1 Mathematical Formulation

The dynamics of the stock price S_t under a jump-diffusion model are described by the following stochastic differential equation (SDE):

$$\frac{dS_t}{S_t} = (r - \lambda k)dt + \sigma dW_t + (J - 1)dN_t.$$

Here, J denotes the jump size and is log-normally distributed, as indicated by:

$$\log(J) \sim \mathrm{N}(\mu_J, \sigma_J^2).$$

The expected jump size or the jump compensator k is defined as:

$$k = \mathrm{E}[J - 1] = \exp\left(\mu_J + \frac{\sigma_J^2}{2}\right) - 1.$$

This adjustment ensures that the discounted asset process, given by:

$$\{\exp(-rt)S_t\}_{t \geq 0},$$

is a martingale. This implies that the process is a 'fair game', without discernible trends or predictable patterns over time.

Martingales in Finance

A martingale in this context suggests that the expected value of the next observation, given all previous observations, equals the present value:

$$\mathrm{E}[X_t \mid \mathrm{F}_s] = X_s,$$

where F_s represents the filtration up to time s.

In financial modeling, particularly under the **Efficient Market Hypothesis**, this characteristic implies that the best prediction of tomorrow's stock price, given today's information, is today's price. Martingales are crucial in financial mathematics, especially in the pricing of derivative securities, as they provide a foundation for assessing fair value without predictable gains or losses.

Stock Price Dynamics

Under the aforementioned dynamics, the stock price at time T can be written as the following:

$$S_T = S_0 \cdot \exp\left((r - \lambda k - \frac{\sigma^2}{2})T + \sigma W_T\right) \cdot \prod_{i=1}^{N_T} J_i,$$

where $N_T \sim \mathrm{Poi}(\lambda T)$ is the number of jumps in the interval $[0, T]$, and $J_1, \ldots, J_{N_T}$ are the independent jump-sizes.

Synthesis

Understanding the dynamics and implications of jump-diffusion models and martingales enriches our approach to financial modeling and risk assessment, highlighting the importance of accounting for both continuous market variations and discrete event-driven spikes in asset pricing.

3.1.2 Inverse-Transform Method for Simulating Poisson Random Variables

This section explores the application of the inverse-transform sampling method to generate observations from a Poisson distribution, which is particularly useful in stochastic process simulations.

Definition of a Poisson Random Variable

A Poisson random variable N_T with parameter λT is defined by the probability mass function (PMF):

$$\mathrm{P}(N_T = k) = \exp(-\lambda T) \cdot \frac{(\lambda T)^k}{k!}, \quad \forall\, k \in \mathrm{Z}^+ \cup \{0\},$$

where:

- λ is the rate parameter of the Poisson process.
- T is the time interval.
- $k!$ (factorial of k) represents the product of all positive integers less than or equal to k.

Inverse-Transform Sampling Method

The inverse-transform method is a technique used to generate random numbers from a given probability distribution by utilizing its cumulative distribution function (CDF). For the Poisson distribution, the CDF can be defined as:

$$F(k; \lambda T) = \sum_{i=0}^{k} \mathrm{P}(N_T = i).$$

To simulate N_T using the inverse-transform method:

1. Generate a random number U from the uniform distribution over $[0, 1]$, $U \sim \mathrm{U}[0, 1]$.
2. Find the smallest integer k such that $F(k; \lambda T) \geq U$.

This approach leverages the fact that U will fall within the interval defined by $F(k-1; \lambda T)$ and $F(k; \lambda T)$, thus determining k effectively simulates drawing from the Poisson distribution.

Practical Considerations

- **Factorials and Computation**: Computing factorials for large k can be computationally intensive; efficient computation or approximation methods may be needed when λT is large.
- **Efficiency**: While straightforward, the inverse-transform method may not be the most efficient for high values of λT due to the need to compute several terms of the Poisson PMF before finding k.

In the next step, we will delve into the programming implementation to simulate N_T using Python, which will demonstrate this theoretical approach in practice.

3.1.3 Inverse-Transfer Method Python Implementation

This section implements the theoretical results derived in Section 3.1.2 in Python.

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

def generate_jump_sizes(mu_J: float, sigma_J: float, n_samples: int) ->
np.ndarray:
    """
    Generates random jump sizes for the jump-diffusion model using the
Inverse-Transform Method.

    Args:
    - mu_J (float): Mean of the logarithm of the jump size.
    - sigma_J (float): Standard deviation of the logarithm of the jump
size.
    - n_samples (int): Number of jump sizes to generate.

    Returns:
    - np.array: Array of simulated jump sizes.
    """
    # Generate standard normal variates
    standard_normals = np.random.randn(n_samples)

    # Transform to log-normal distribution using the inverse transform
method
    jump_sizes = np.exp(mu_J + sigma_J * standard_normals)

    return jump_sizes

# Example usage
mu_J = 0.05  # Mean of log(J)
sigma_J = 0.1  # Standard deviation of log(J)
n_samples = 1000  # Number of jump sizes to generate

# Generate jump sizes
jumps = generate_jump_sizes(mu_J, sigma_J, n_samples)

# Output some statistics for sanity check
print(f"Mean of Generated Jumps: {np.mean(jumps):.4f}")
print(f"Standard Deviation (Volatility) of Generated Jumps: {np.std(jumps,
ddof=1):.4f}")
```

```
Mean of Generated Jumps: 1.0584
Standard Deviation (Volatility) of Generated Jumps: 0.1045
```

3.1.4 Jump-Diffusion Connection to Black-Scholes-Merton

This section examines how the Jump-Diffusion model relates to the classic Black-Scholes-Merton (BSM) model when $\lambda = 0$ and explores the effect of the jump component $\lambda > 0$ on

the pricing of standard, vanilla options.

Connection to Black-Scholes-Merton Model

When the jump intensity λ is set to zero, the Jump-Diffusion model simplifies to the Black-Scholes-Merton model. The absence of jumps means that the stock price dynamics are governed solely by the continuous paths generated by the geometric Brownian motion:

$$\frac{dS_t}{S_t} = rdt + \sigma dW_t.$$

This equation mirrors the Black-Scholes formula for non-dividend-paying stocks, where r is the risk-free rate ($\mu = r$ under the risk-neutral probability measure Q), and σ is the volatility of the stock's returns.

Impact of Jump Component on Option Pricing

Introducing a non-zero λ (i.e., the presence of jumps) significantly alters the risk profile of an asset. This change directly impacts the pricing of derivatives such as call and put options for several reasons:

1. **Volatility Smile**: Jumps introduce discontinuities in the price path, which can lead to heavy tails in the distribution of returns (i.e., an increase in the fourth standardized moment: kurtosis, yielding a leptokurtic distribution). This behavior often results in a *volatility smile*, where implied volatility increases for deep in-the-money and out-of-the-money options.

2. **Option Moneyness**:

    - **In-the-Money Options (ITM)**: Jumps can cause more frequent and significant ITM occurrences, potentially increasing the value of these options compared to their BSM counterparts.
    - **Out-of-the-Money options (OTM)**: Similarly, the possibility of sudden large jumps can significantly enhance the value of OTM options, reflecting a higher probability of these options becoming ITM.
3. **Cost of Options**:

    - The inclusion of jumps generally **increases the cost of options** because the risk (and therefore the premium) associated with sudden large moves is higher.
    - This is particularly noticeable for OTM options, where even a small probability of a significant jump can lead to a substantial increase in premiums due to the higher risk of such options expiring ITM.

Intuitive Framework

Here's an intuitive framework to understand why jumps affect option pricing:

- **Risk Aversion and Jump Risk**: Investors are generally risk-averse; thus, any additional uncertainty—like that introduced by jumps—requires a higher return to compensate for increased risk.
- **Leverage Effect**: Jumps can exacerbate the *leverage effect*, where the equity volatility of a levered firm increases disproportionately when its value declines sharply. This effect can amplify the price changes of options, particularly for leveraged entities.
- **Predictability and Protection**: Options are often used as insurance against price movements. The unpredictability introduced by jumps increases the demand for this protection, driving up option prices.

Synthesis

The Jump-Diffusion model, by introducing jumps, captures a more realistic picture of market behavior, which includes sudden, significant price changes not accounted for by the Black-Scholes-Merton model. Understanding this connection helps in better modeling of risk and pricing of derivatives in financial markets where large jumps in asset prices are a real possibility.

3.1.5 Jump-Poisson Stock Process Simulation

This section implements the Jump-Diffusion model to estimate the prices of call and put options using the following parameters:

- $n = 10,000$: number of simulations
- $S_0 = 100$: initial spot (stock) price
- $K = 100$: strike price
- $T = 1$: maturity (1 year)
- $r = 0.03$: risk-free (interest) rate (3%)
- $\sigma = 0.20$: volatility (standard deviation) (20%)
- $\mu_J = 0.02$: (expected) jump size/magnitude (2%)
- $\sigma_J = 0.08$: (expected) jump volatility (8%)

We report our price estimates for both $\lambda \in [0, 2]$ in a tabular format with 95% confidence interval estimates, yielding a total of four estimates and four confidence intervals.

3.1.5 Jump-Poisson Stock Process Simulation

This section implements the Jump-Diffusion model to estimate the prices of call and put options using the following parameters:

- (n = 10,000): number of simulations
- (S_0 = 100): initial spot (stock) price
- (K = 100): strike price
- (T = 1): maturity (1 year)
- (r = 0.03): risk-free (interest) rate (3%)

- (\sigma = 0.20): volatility (standard deviation) (20%)
- (\mu_J = 0.02): (expected) jump size/magnitude (2%)
- (\sigma_J = 0.08): (expected) jump volatility (8%)

We report our price estimates for both (\lambda \in [0, 2]) in a tabular format with 95% confidence interval estimates, yielding a total of four estimates and four confidence intervals.

Python Code

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

def simulate_jump_diffusion(S0, K, T, r, sigma, mu_J, sigma_J, lambda_,
n):
    """
    Simulate stock prices and estimate call and put option prices under a
jump-diffusion model.

    Args:
    - S0 (float): Initial spot price.
    - K (float): Strike price.
    - T (float): Time to maturity.
    - r (float): Risk-free rate.
    - sigma (float): Volatility.
    - mu_J (float): Mean of the jump size.
    - sigma_J (float): Standard deviation of the jump size.
    - lambda_ (float): Jump intensity.
    - n (int): Number of simulations.

    Returns:
    - (float, float): Estimated call and put prices.
    - (float, float): 95% confidence intervals for call and put prices.
    """
    dt = T / 252  # Time increment (daily steps)
    m = int(T / dt)  # Number of time steps

    # Simulate jump diffusion paths
    ST = np.zeros(n)
    for i in range(n):
        N = np.random.poisson(lambda_ * T)
        jump_sizes = np.random.normal(mu_J, sigma_J, N)
        jumps = np.prod(1 + jump_sizes)
        W_T = np.random.normal(0, np.sqrt(T))
        ST[i] = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * W_T) *
jumps

    # Calculate option payoffs
    call_payoffs = np.maximum(ST - K, 0)
    put_payoffs = np.maximum(K - ST, 0)

    # Discount payoffs to present value
```

```
    call_price = np.exp(-r * T) * np.mean(call_payoffs)
    put_price = np.exp(-r * T) * np.mean(put_payoffs)

    # 95% confidence intervals
    call_ci = 1.96 * np.std(call_payoffs) / np.sqrt(n)
    put_ci = 1.96 * np.std(put_payoffs) / np.sqrt(n)

    return (call_price, call_ci), (put_price, put_ci)

# Parameters
S0 = 100
K = 100
T = 1
r = 0.03
sigma = 0.20
mu_J = 0.02
sigma_J = 0.08
n = 10000
lambdas = [0, 2]

# Results
results = []
for lambda_ in lambdas:
    (call_price, call_ci), (put_price, put_ci) =
simulate_jump_diffusion(S0, K, T, r, sigma, mu_J, sigma_J, lambda_, n)
    results.append((lambda_, call_price, call_ci, put_price, put_ci))

# Display results
import pandas as pd
df = pd.DataFrame(results, columns=['Lambda', 'Call Price', 'Call CI',
'Put Price', 'Put CI'])
df['Call CI'] = df['Call CI'].apply(lambda x: f"± {x:.4f}")
df['Put CI'] = df['Put CI'].apply(lambda x: f"± {x:.4f}")
print(df)
```