Vrije Universiteit Amsterdam
Period 1, 2024-2025

Computer Assignment 2

Stochastic Processes : The Fundamentals

Yunji Eo (2735445)
y.eo@student.vu.nl
Archit Murkunde (2845576)
a.u.murkunde@student.vu.nl

Instructors:
Eva Mynott and Mark-jan Boes

October 16, 2024

Econometrics and Operations Research

1 Monte Carlo Simulation

(a)

Our values for u and d are **1.05166** and **0.95698**, respectively.

By using these values, we calculate our risk-neutral probabilities and price our call option using the Binomial Tree Model. Our price comes out to be **238.242**.

We also calculate the price using Black Scholes formula which comes out to be **238.24**.

We see similarities in the prices which is as expected.

(b)

We calculate the price of the European Call option using Euler's method. We use the following formula to simulate the future values/paths of the stock price:

$$S_{n+1} = S_n + S_n \left(r\Delta t + \sigma\sqrt{\Delta t}\phi \right),$$

where S_n is the stock price at step n, r is the continuously compounded interest rate, Δt is the time step, σ is the volatility, and ϕ is a standard normal random variable.
We then calculate the price of the option as:

$$V(S_0) = e^{-rT}\frac{1}{M}\sum_{m=1}^{M} \text{payoff}_m(S_N),$$

where r is the continuously compounded interest rate, T is the total time period, N is the total number of steps, M is the number of simulated paths, and $\text{payoff}_m(S_N)$ is the payoff of the m-th simulated path at time N.

Choosing Δt as 1 month, below are the prices for each simulated path:
Price for M = 100 is **253.37**
Price for M = 500 is **218.92**
Price for M = 1000 is **252.05**
Price for M = 5000 is **238.75**
Price for M = 10000 is **238.6**

By using Euler method we see that as the number of simulated paths M increases, the option price converges toward the values obtained using the Binomial Tree and Black-Scholes models. For $M = 5000$ and $M = 10000$, the prices (238.75 and 238.6) are very close to the Black-Scholes price of 238.24, demonstrating the stability and accuracy of the method when more paths are used, reducing the simulation error.

(c)

We choose Δt as 1 trading day, which leads us to the below price for the same number of simulations:

Price for M = 100 is **208.58**
Price for M = 500 is **250.57**
Price for M = 1000 is **247.4**
Price for M = 5000 is **242.89**
Price for M = 10000 is **233.12**

The results are similar to what we have in the previous section where we used Δt as 1 month, we do not see any significant difference in the results. As M increases, we expect the option price to converge to the one obtained from Black-Scholes model.

In this case, using daily steps does not significantly improve the accuracy of the price. We can hence conclude that increasing the number of simulations has a more pronounced effect than just reducing the time step size.

(d)

We now use the following formula to simulate the stock prices for 100, 500, 1000, 5000, and 10000 paths:

$$S_T = S_0 e^{\left(r-0.5\sigma^2\right)T+\sigma\sqrt{T}\phi},$$

where S_T is the simulated stock price at time T, S_0 is the initial stock price, r is the continuously compounded interest rate, σ is the volatility, and ϕ is a standard normal random variable.

The price of the call option for each number of paths is:
Price for M = 100 is **218.99**
Price for M = 500 is **276.03**
Price for M = 1000 is **230.85**
Price for M = 5000 is **238.63**
Price for M = 10000 is **238.47**

(e)

Using the exact formula for S_T provides one of the most accurate approximations of the Black-Scholes price. The results converge closely to the Black-Scholes value of 238.24 as the number of paths M increases, with prices for $M = 5000$ and $M = 10000$ being 238.63 and 238.47, respectively. This indicates that using the exact solution for stock prices, combined with a large number of simulations, effectively captures the theoretical option price. Compared to other methods, it reduces discretization errors, leading to higher accuracy.

(f)

We now price the Asian call option using Euler method for stock price simulation with $M = 10000$ and $\Delta t = 1$ trading day. The payoff of the Asian Call option is given by:

$$\text{payoff}^m(S) = \max\left(\left(\frac{1}{n}\sum_{i=N-n}^{N} S_i^m\right) - K, 0\right),$$

We calculate the price to be **210.09**

The price of the Asian call option (210.09) compared to the European call option (233.12 for $M = 10,000$) is lower primarily due to differences in the payoff calculation between the two options.

The Asian option's payoff depends on the average stock price over the last month, which tends to smooth out fluctuations. This averaging effect generally results in a lower volatility compared to the European option, whose payoff is based on the final stock price. Lower volatility reduces the option's value because there is a smaller chance of the average price exceeding the strike price significantly. Thus, the nature of the payoff formula leads to a lower option price for the Asian call.

2 Dynamic Hedging

(a)

The trader receives the total value of the sold call options at the time of maturity using the Black-Scholes model. The trader sells 10 European call options with a strike price of \$5,600 on the S&P 500 index. The size of each contract is 100 units.
Using the Black-Scholes formula with the following parameters:

- $S_0 = 5600$ (initial value of the S&P 500 index),
- $K = 5600$ (strike price),
- $r = 0.04$ (annualized risk-free interest rate with quarterly compounding),
- $T = \frac{3}{12}$ (time to maturity, which is 3 months),
- $\sigma = 0.047 \times \sqrt{12}$ (annualized volatility based on the sample volatility from log returns).

The calculated Black-Scholes call option price was approximately $210.136 per option. Since the trader sells 10 contracts, each consisting of 100 options, the total amount received from the institutional investor is:

$$\text{Total amount} = \text{size} \times n \times \text{call price} = 100 \times 10 \times 210.136 = 210,136.312$$

Thus, the trader receives **$210,136.31** from the institutional investor for selling the 10 call option contracts.

(b)

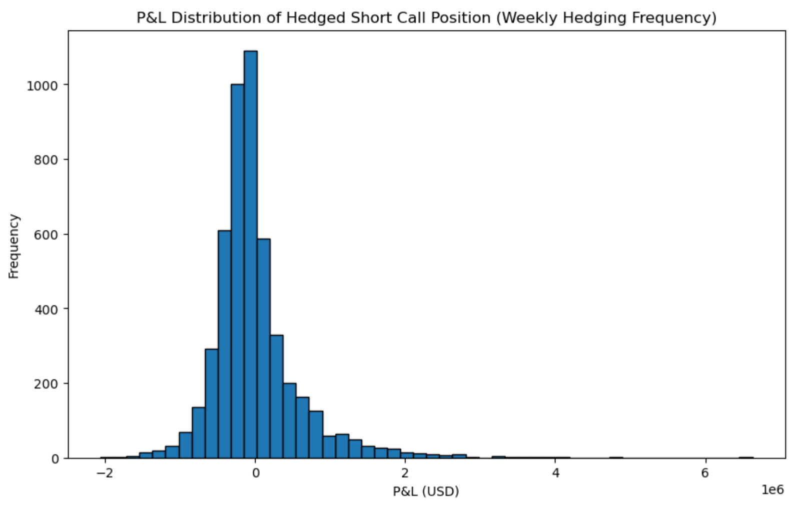

Figure 1: P&L Distribution (Weekly Hedging Frequency)

The P&L distribution 1 for the weekly hedging frequency is centered around zero, with most values ranging between -2 million and +2 million USD. The shape of the distribution is relatively narrow, indicating that the weekly adjustments in the hedging strategy manage to reduce the variance effectively, though there are still some outliers on both ends.

(c)

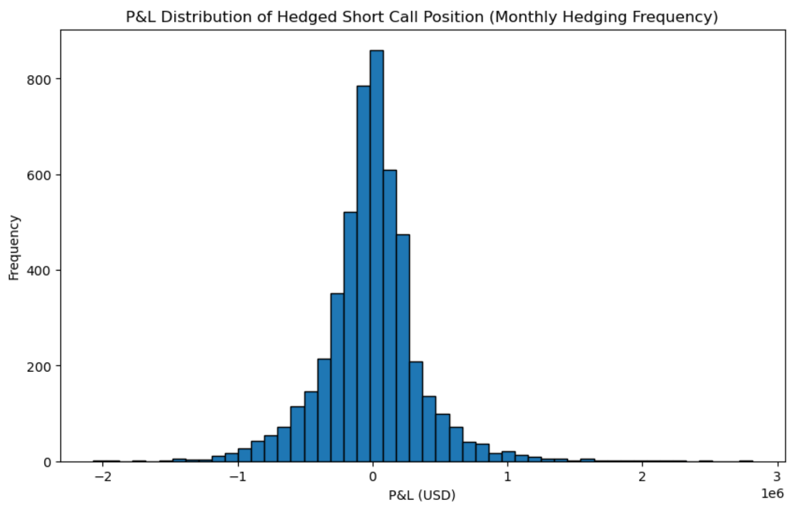

Figure 2: P&L Distribution (Monthly Hedging Frequency)

The P&L distribution 2 for monthly hedging is noticeably wider compared to that of weekly hedging. This outcome is expected because with monthly hedging, the portfolio is adjusted less often than the portfolio with weekly hedging. This increases the risk of larger discrepancies as the value of portfolio can accumulate over a longer period without correction. As a result, the distribution becomes wider, indicating more significant potential for both gains and losses

(d)

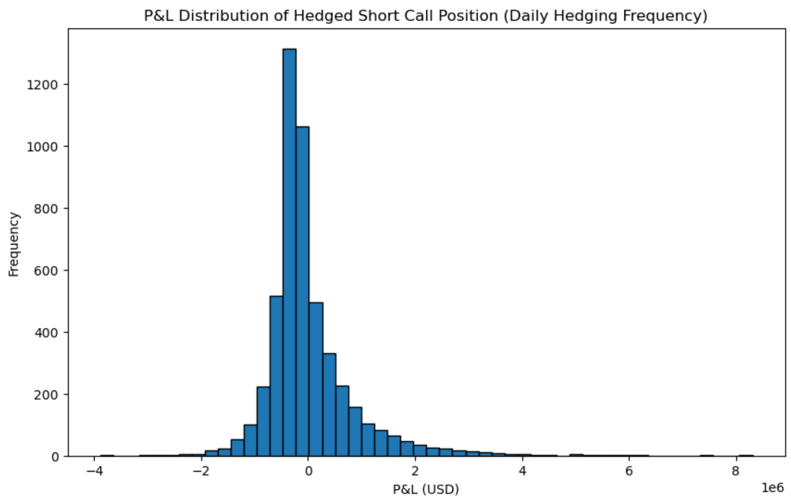

Figure 3: P&L Distribution (Daily Hedging Frequency)

The figure 3 shows a significantly narrower P&L distribution compared to both weekly and monthly hedging frequencies. With daily hedging, the portfolio is adjusted much more frequently, minimizing the discrepancy between the option's payoff and the value of the replicating portfolio at each interval. This results in a more precise replication of the option's value. This confirms that as the hedging frequency increases (in this case, daily), the P&L distribution becomes more concentrated around zero.

(e)

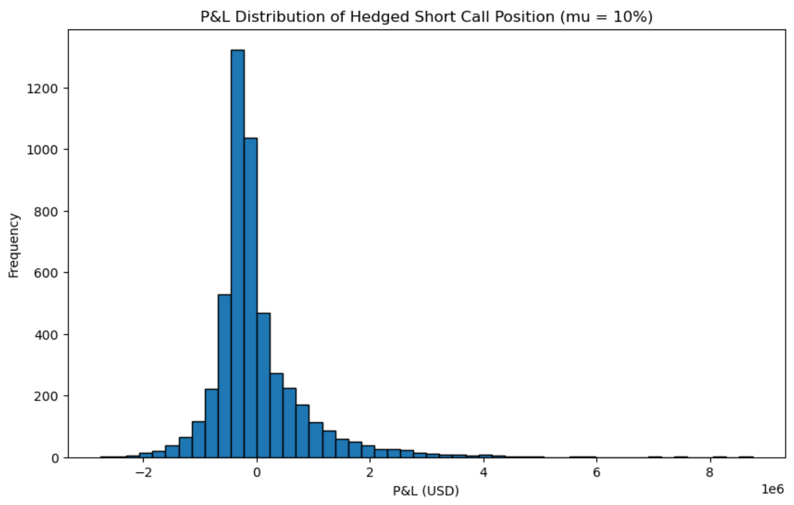

Figure 4: P&L Distribution ($\mu = 10\%$)

With $\mu = 10\%$, the P&L distribution for daily hedging is remaining centered around zero, which indicates that daily hedging still effectively minimizes risk. However, the distribution exhibits slightly narrower range of loss. This means that the likelihood of significant losses have decreased with the higher drift rate.

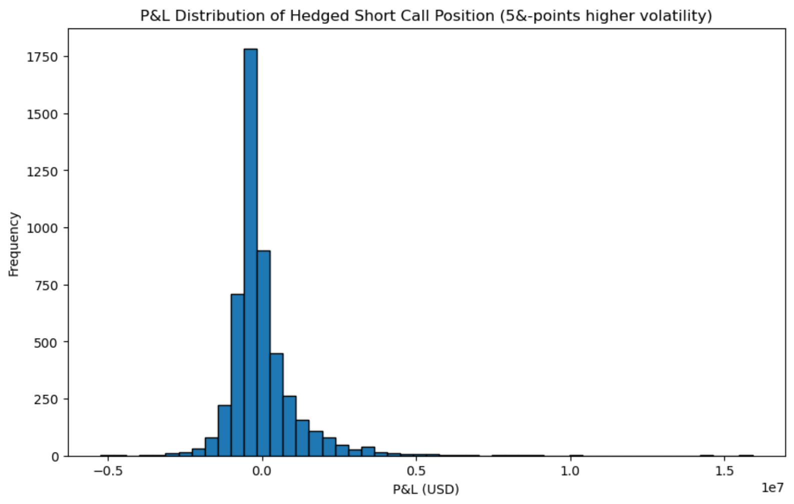

Figure 5: P&L Distribution (5%-points Higher Volatility)

(f)

In figure 5,while the P&L remains concentrated near zero for the majority of cases, with 5%-points higher volatility leads to large outliers, especially on the profit side. The range of outcomes extends from -0.5 million to +15 million USD. This differs from the results in part 2.e, where increasing the drift μ —a deterministic factor—did not introduce such extreme outcomes. The increased volatility adds more randomness and uncertainty, making it harder to hedge, which creates a wider range of potential outcomes. Consequently, the higher volatility increases the variation, resulting in large outliers, even though most of the P&L remains concentrated around zero.

(g)

To answer this question, we calculated the amount received from selling the call option for question (d) and (f). In question (e), the amount received from selling the call option was USD 210,136.31 when the implied volatility was lower. In question (f), with a higher implied volatility (by adding 0.05 to the original volatility), the amount received increased to USD 265,328.59. This shows that as the perceived future risk or uncertainty (higher volatility) increases, the option prices go up.
The reason for this is that when markets expect larger price movements in the underlying asset (S&P 500 index in your case), option sellers demand a higher price to compensate for the additional risk. Even though the historical volatility may not show such large movements, the market prices in a premium to cover potential unexpected changes in the asset's price.
Therefore, the options become more expensive because they account for not just the past behavior of the index but also the possibility of sudden, large changes in the future. This explains why, despite the actual volatility being lower, the market consistently prices options higher than expected

Appendix - Code

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
import statsmodels.api as sm
from scipy.stats import norm
```

Listing 1: Imports

Part 1

(a)

```python
So = 5648
sigma_monthly = 0.047
```

Listing 2: S&P-500 value and Monthly Variance of S&P-500

```python
def calculate_u_d(steps):
    r_u, r_d = sp.symbols('r_u r_d')

    monthly_return = 0.01
    monthly_variance = 0.047**2
    adjusted_return = monthly_return * (3 / steps)
    adjusted_variance = monthly_variance * (3 / steps)

    expected_return_eq = 0.56 * r_u + 0.44 * r_d - adjusted_return  # Expected return equation
    variance_eq = 0.56 * (r_u - adjusted_return)**2 + 0.44 * (r_d - adjusted_return)**2 - adjusted_variance  # Variance equation

    solution_discrete = sp.solve([expected_return_eq, variance_eq], (r_u, r_d))

    # Calculate u and d from the returns
    u_1 = np.round(float(1 + solution_discrete[0][0]), 5)
    d_1 = np.round(float(1 + solution_discrete[0][1]), 5)

    return u_1, d_1

u_1, d_1 = calculate_u_d(3)
print("u:", u_1)
print("d:", d_1)
```

Listing 3: Calculating u and d

```python
def price_european_call_binomial(S0, K, r, u, d, N, q_u=None, q_d=None):
    R = (r*0.25)/N
    if q_u is None:
        q_u = (1 + R - d) / (u - d)
    if q_d is None:
        q_d = 1 - q_u

    # Initialize stock price tree
    stock_tree = np.zeros((N + 1, N + 1))

    # Fill the stock price tree
    for i in range(N + 1):
        for j in range(i + 1):
            stock_tree[j, i] = S0 * (u ** (i - j)) * (d ** j)

    # Initialize option price tree
    option_tree = np.zeros((N + 1, N + 1))

    # Calculate option payoff at maturity (t = N)
    for j in range(N + 1):
        option_tree[j, N] = max(stock_tree[j, N] - K, 0)  # Payoff at
   maturity

    # Backward induction to calculate option price at earlier nodes
    for i in range(N - 1, -1, -1):
        for j in range(i + 1):
            option_tree[j, i] = (q_u * option_tree[j, i + 1] + q_d *
   option_tree[j + 1, i + 1]) / (1 + R)

    # Return the option price at the root of the tree (at t = 0), along
   with stock_tree and option_tree
    option_price_at_t0 = option_tree[0, 0]
    return stock_tree, option_tree, option_price_at_t0
```

Listing 4: Pricing European Call Option using Binomial Model

```
def black_scholes(S0, K, r, T, sigma, option_type):
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == "Call":
        option_price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2
   )
    elif option_type == "Put":
        option_price = K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-
   d1)
    else:
        raise ValueError("Invalid option type. Please input 'Call' or 'Put
   '.")

    return option_price
```

Listing 5: Black-Scholes Model for Option Pricing

```
# Parameters
S0 = 5648
K = 5600
T = 3 / 12
r = 0.04
sigma_monthly = 0.047
sigma_annual = sigma_monthly * np.sqrt(12)
N = 1000
bs_call_price = black_scholes(S0, K, r, T, sigma_annual,'Call')
u, d = calculate_u_d(N)
stock_tree, option_tree, price = price_european_call_binomial(S0, K, r, u,
    d, N)
```

```
print("Price of CE using Binomial Model:", np.round(price,3))
print("Price of CE using BSM Model:", np.round(bs_call_price,3))
```

(b)

```
def euler_call_option_price(S0, K, r_annual, T, sigma_annual, M, N):
    dt = T / N
    r = r_annual
    sigma = sigma_annual

    S = np.full(M, S0, dtype=float)

    # Simulating Stock prices as per Euler Method
    for n in range(N):
        phi = np.random.normal(0, 1, M)
        S += S * (r * dt + sigma * np.sqrt(dt) * phi)


    payoffs = np.maximum(S - K, 0)
    option_price = np.exp(-r * T) * np.mean(payoffs)

    return option_price, S

# Parameters
S0 = 5648
K = 5600
r_annual = 0.04
T = 3/12       # dt = 1 month
sigma_annual = 0.047*np.sqrt(12)
N = 3

for i in [100, 500, 1000, 5000, 10000]:
    print(f"Price for M = {i} is:", np.round(euler_call_option_price(S0, K,
     r_annual, T, sigma_annual, M=i, N=N),2))
```

Listing 6: Euler Method for Stock simulation and Option pricing

(c)

```

# Parameters
S0 = 5648
K = 5600
r_annual = 0.04
T = 63/252
sigma_annual = 0.047*np.sqrt(12)
N = 63

for i in [100, 500, 1000, 5000, 10000]:
    print(f"Price for M = {i} is:", np.round(euler_call_option_price(S0, K,
     r_annual, T, sigma_annual, M=i, N=N),2))
```

Listing 7: Euler Method for Stock simulation and Option pricing

(d)

```python

# Parameters
S0 = 5648
K = 5600
r_annual = 0.04
T = 3/12
sigma = 0.047*np.sqrt(12)

for M in [100, 500, 1000, 5000, 10000]:
    x = np.random.normal(0, 1, M)
    # Simulating Stock prices using GBM method
    S = S0 * np.exp(((r - 0.5*(sigma**2))*T)+(sigma*np.sqrt(T)*x))
    payoffs = np.maximum(S - K, 0)
    option_price = np.exp(-r * T) * np.mean(payoffs)
    print(f"Option price for M = {M} is: {option_price:.2f}")
```

Listing 8: GBM Method for Stock simulation

(f)

```python

def asian_call_option_price(S0, K, r_annual, T, sigma_annual, M, N):
    dt = T / N
    r = r_annual
    sigma = sigma_annual

    S = np.full((N+1, M), S0, dtype=float)

    for n in range(1, N+1):
        phi = np.random.normal(0, 1, M)
        S[n, :] = S[n-1, :] * (1 + (r * dt + sigma * np.sqrt(dt) * phi))

    avg_price = np.mean(S[-21:, :], axis=0)
    payoffs = np.maximum(avg_price - K, 0)
    option_price = np.exp(-r * T) * np.mean(payoffs)

    return option_price, S

S0 = 5648
K = 5600
r_annual = 0.04
T = 63 / 252
sigma_annual = 0.047 * np.sqrt(12)
M = 10000
N = 63

asian_price, S = asian_call_option_price(S0, K, r_annual, T, sigma_annual,
    M, N)

print(f"Asian call option price: {asian_price:.2f}")
```

Listing 9: Asian Call Option Pricing

Part 2

(a)

```python
def black_scholes(S0, K, r, T, sigma, option_type): # function for BSM model
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == "Call":
        option_price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif option_type == "Put":
        option_price = K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)
    else:
        raise ValueError("Invalid option type. Please input 'Call' or 'Put'.")

    return option_price, d1

option_price, _ =  black_scholes(S0=5600, K=5600, r=0.04, T=3/12, sigma_annoual=0.047*np.sqrt(12), option_type= "Call")
contract_size = 100
num_contracts = 10
total_amount = option_price * contract_size * num_contracts
```

Listing 10: Amount of Money the Traders Receives

(b)

```python
# Parameters
S0 = 5600
K = 5600
T = 0.25
r = 0.04
sigma=0.047*np.sqrt(12)
mu = 0.06
num_contracts = 10
contract_size = 100
num_paths = 5000
num_steps = 13
dt = T / num_steps

def generate_paths(S0, mu, sigma, T, num_steps, num_paths):
    dt = T / num_steps
    paths = np.zeros((num_paths, num_steps + 1))
    paths[:, 0] = S0
    for i in range(1, num_steps + 1):
        dW = np.random.normal(0, np.sqrt(dt), num_paths)
        paths[:, i] = paths[:, i-1] * np.exp((mu - 0.5*sigma**2)*dt + sigma*dW)
    return paths

# Simulate hedging strategy
def simulate_hedging(paths, K, T, r, sigma, num_steps):
    dt = T / num_steps
    num_paths = paths.shape[0]
    option_values = np.zeros((num_paths, num_steps + 1))
    hedge = np.zeros((num_paths, num_steps + 1))
    deltas = np.zeros((num_paths, num_steps + 1))
    payoff = np.zeros((num_paths, num_steps + 1))

    for i in range(num_steps + 1):
        t = i * dt
        option_values[:, i], deltas[:, i] = black_scholes(paths[:, i], K, r
, T-t, sigma, "Call")

        if i == 0:
            hedge[:, 0] = option_values[:, 0]
        else:
            stock = deltas[:, i-1] * paths[:, i]
            payoff = np.maximum(paths[:, -1] - K, 0)
            cash = hedge[:,i-1] - deltas[:, i-1] * paths[:, i-1]
            hedge[:, i] = stock + cash * np.exp(r*dt)

    return option_values, hedge, deltas, payoff

# Run simulation
paths = generate_paths(S0, mu, sigma, T, num_steps, num_paths)
option_values, hedge, deltas, payoff = simulate_hedging(paths, K, T, r,
    sigma, num_steps)

# Calculate P&L
initial_option_value = option_values[0, 0]

final_hedge_value = hedge[:, -1]

pnl = ( - payoff + final_hedge_value) * num_contracts * contract_size

# Plot histogram of P&L
plt.figure(figsize=(10, 6))
plt.hist(pnl, bins=50, edgecolor='black')
plt.title('P&L Distribution of Hedged Short Call Position (Weekly Hedging
    Frequency)')
plt.xlabel('P&L (USD)')
plt.ylabel('Frequency')
plt.show()
```

Listing 11: Dynamic Hedging with Weekly Frequency

(c)

```python
# Parameters
num_steps = 3  # Monthly adjustments for 3 months
dt = T / num_steps

# Run simulation
paths = generate_paths(S0, mu, sigma, T, num_steps, num_paths)
option_values, hedge, deltas, payoff = simulate_hedging(paths, K, T, r,
    sigma, num_steps)

```

```
# Calculate P&L
initial_option_value = option_values[0, 0]

final_hedge_value = hedge[:, -1]

pnl = ( - payoff + final_hedge_value) * num_contracts * contract_size

# Plot histogram of P&L
plt.figure(figsize=(10, 6))
plt.hist(pnl, bins=50, edgecolor='black')
plt.title('P&L Distribution of Hedged Short Call Position (Monthly Hedging
    Frequency)')
plt.xlabel('P&L (USD)')
plt.ylabel('Frequency')
plt.show()
print("Mean P&L:", np.mean(pnl))
print("Standard Deviation of P&L:", np.std(pnl))

call_price, _ = black_scholes(S0, K, r, T, sigma, option_type="Call")

# Total amount received from selling the call options
amount_received = call_price * num_contracts * contract_size
```

Listing 12: Dynamic Hedging with Monthly Frequency

(d)

```
# Parameters
num_steps = 63 # Daily adjustments for 3 months (21 trading days per month)
dt = T / num_steps

# Run simulation
paths = generate_paths(S0, mu, sigma, T, num_steps, num_paths)
option_values, hedge, deltas, payoff = simulate_hedging(paths, K, T, r,
    sigma, num_steps)

# Calculate P&L
initial_option_value = option_values[0, 0]

final_hedge_value = hedge[:, -1]

pnl = ( - payoff + final_hedge_value) * num_contracts * contract_size

# Plot histogram of P&L
plt.figure(figsize=(10, 6))
plt.hist(pnl, bins=50, edgecolor='black')
plt.title('P&L Distribution of Hedged Short Call Position (Daily Hedging
    Frequency)')
plt.xlabel('P&L (USD)')
plt.ylabel('Frequency')
plt.show()
print("Mean P&L:", np.mean(pnl))
print("Standard Deviation of P&L:", np.std(pnl))
```

Listing 13: Dynamic Hedging with Daily Frequency

(e)

```
paths = generate_paths(S0, 0.1, sigma, T, num_steps, num_paths) # now mu is
     0.1
option_values, hedge, deltas, payoff = simulate_hedging(paths, K, T, r,
    sigma, num_steps)

# Calculate P&L
initial_option_value = option_values[0, 0]

final_hedge_value = hedge[:, -1]

pnl = ( - payoff + final_hedge_value)* num_contracts * contract_size

# Plot histogram of P&L
plt.figure(figsize=(10, 6))
plt.hist(pnl, bins=50, edgecolor='black')
plt.title('P&L Distribution of Hedged Short Call Position (mu = 10%)')
plt.xlabel('P&L (USD)')
plt.ylabel('Frequency')
plt.show()
print("Mean P&L:", np.mean(pnl))
print("Standard Deviation of P&L:", np.std(pnl))
```

Listing 14: Dynamic Hedging with mu

(f)

```
paths = generate_paths(S0, 0.1, sigma + 0.05, T, num_steps, num_paths) #
    not the sigma is 5% higher
option_values, hedge, deltas, payoff = simulate_hedging(paths, K, T, r,
    sigma +0.05, num_steps)

# Calculate P&L
initial_option_value = option_values[0, 0]

final_hedge_value = hedge[:, -1]

pnl = ( - payoff + final_hedge_value) * num_contracts * contract_size

# Plot histogram of P&L
plt.figure(figsize=(10, 6))
plt.hist(pnl, bins=50, edgecolor='black')
plt.title('P&L Distribution of Hedged Short Call Position (5&-points higher
     volatility)')
plt.xlabel('P&L (USD)')
plt.ylabel('Frequency')
plt.show()
print("Mean P&L:", np.mean(pnl))
print("Standard Deviation of P&L:", np.std(pnl))

call_price, _ = black_scholes(S0, K, r, T, sigma+0.05, option_type="Call")

# Total amount received from selling the call options
amount_received = call_price * num_contracts * contract_size
```

Listing 15: Dynamic Hedging with Higher Volatility