Finance II Project 1

Isaac Clark

March 7, 2025

Section I

Problem 1a

Using Python, I simulated the Ito's integral approximated by

$$\int_0^t f(s, W(s))dW(s) \approx \sum_{j=1}^{M} f(t_{j-1}, W(t_{j-1}))(W(t_j) - W(t_{j-1}))$$

for the functions $f(t, x) = x$ and $f(t, x) = x^2$. Using the variables $M = 1000$ and $K = 10000$, I estimated the mean and variance as functions of t in the simulation and the results were plotted in Figure 1.

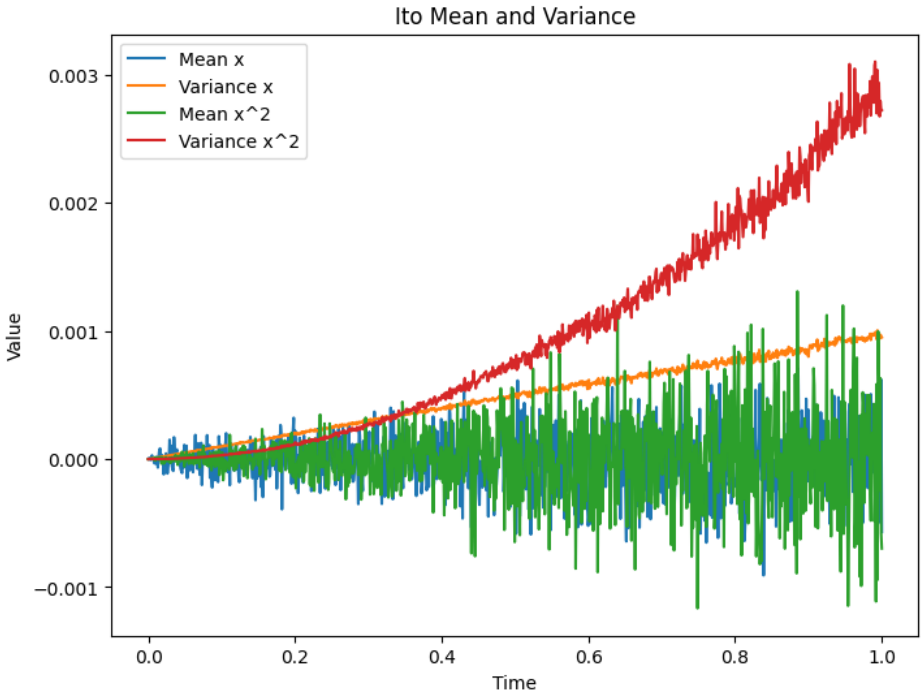

Figure 1: Ito Mean and Variance

Problem 1b

Then tasked with repeating the experiment using the Stratonovich integral approximated by

$$\int_0^t f(s, W(s))dW(s) \approx \sum_{j=1}^{M} \frac{f(t_{j-1}, W(t_{j-1})) + f(t_j, W(t_j))}{2}(W(t_j) - W(t_{j-1}))$$

I plotted the results of the repeated experiment using the Stratonovich integral in Figure 2.

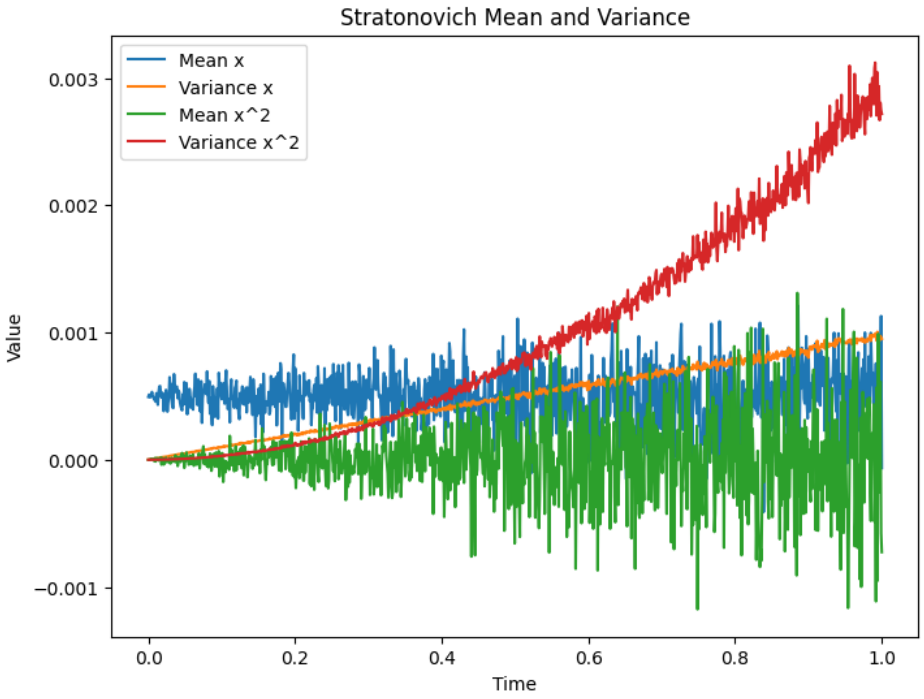

Figure 2: Stratonovich Mean and Variance

Problem 1c

Once again using Python, I simulated the stock prices in a Black-Scholes model for two hypothetical stocks, A and B, for $T = 1$. Using the variables $r = 0.03$, $\sigma_1 = 0.2$, $\sigma_2 = 0.3$, $\rho = 0.5$, and $S_0 = \$50$, I was able to simulate the stock prices and estimate the value of an exchange option to $V_T = 5.184$.

Problem 2a

With the model for stock price changes

$$\frac{dS(t)}{S(t)} = \mu dt + \sigma dW(t)$$

and discreteized as

$$\frac{S_i - S_{i-1}}{S_{i-1}} = \mu \Delta t + \sigma \sqrt{\Delta t} \epsilon_i$$

Using the variables $\mu = 0.05$, $\sigma = 0.25$, $\Delta t = 1/252$, and $S(0) = 50$, I simulated 1000 price paths daily returns for the hypothetical stock which I graphed in Figure 3

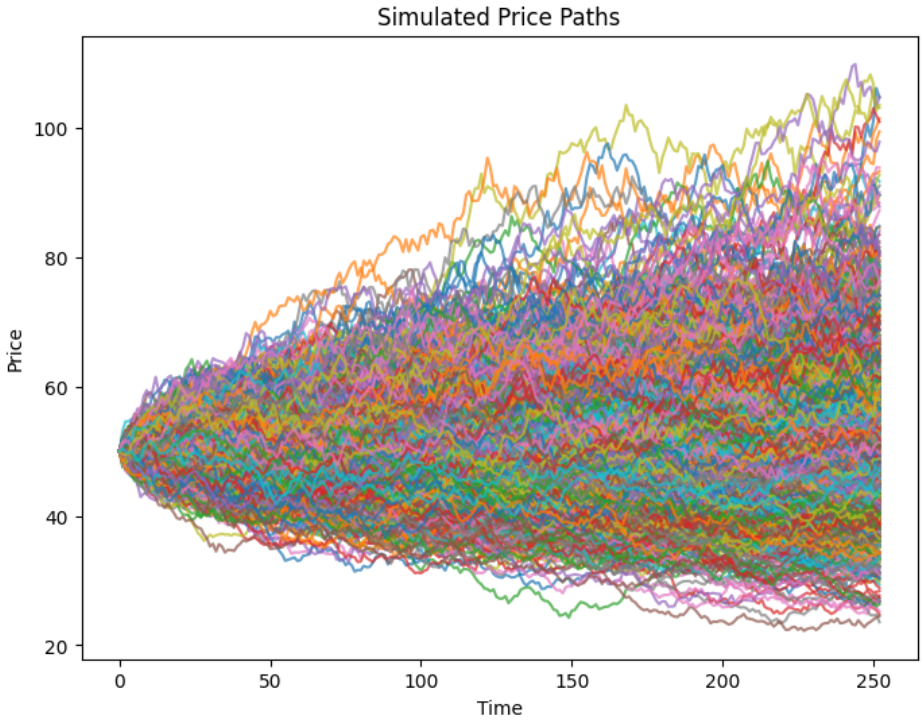

Figure 3: Simulated Price Paths

Problem 2b

To analyze the realized variance of the simulated stock and verify that there isn't much randomness in the simulated values, I calculated the realized variance using

$$RV = \sum_{i=1}^{M} (R_i - \bar{R})^2$$

and studied its descriptive statistics shown in Table 1.

Mean	0.062259
Standard Deviation	0.005527
Minimum	0.047032
25th Percentile	0.058600
50th Percentile	0.062127
75th Percentile	0.066019
Maximum	0.080058

Table 1: Realized Variance Descriptive Statistics

With a low standard deviation and very little spread between the percentile values, we can conclude that there is not much randomness in the simulated values since the realized variance of all 1000 simulations doesn't stray far from the average. Compared to the theoretical variance of 0.625, the mean realized variance is extremely close to the theoretical variance and without much deviation in realized variance from the theoretical variance across all 1000 simulations.

Problem 2c

Using the Heston model for stochastic volatility,

$$\frac{dS(t)}{S(t)} = \mu dt + \sqrt{V(t)} dW^{(1)}(t)$$

$$dV(t) = \kappa(\theta - V)dt + \sigma_V \sqrt{V(t)} dW^{(2)}(t)$$

and parameters $\kappa = 1$, $\theta = 0.02$, $\sigma_V = 0.5$, $\rho = -0.5$, and $V(0) = 0.0625$, I simulated 1000 price paths and calculated the quadratic variation, and estimated the variance of the quadratic variation to be approximately 0.00231.

When applying different values of σ_V to the simulation, it was observed that there was a slight change in the variance of the quadratic variation as different values of σ_V were used. As σ_V increased, so did the variance of the quadratic variation, and the same was true for the opposite, implying that a more volatile σ_V produced move volatility in the quadratic variation of the 1000 simulated price paths.

Section II

Problem 1

After acquiring the relevant S&P 500 options data from Yahoo Finance and selecting 10 call and put options, with various strike prices, expiration dates, and valuations, I calculated the implied volatility for each and plotted call the put options in Tables 4 and 5 respectively.

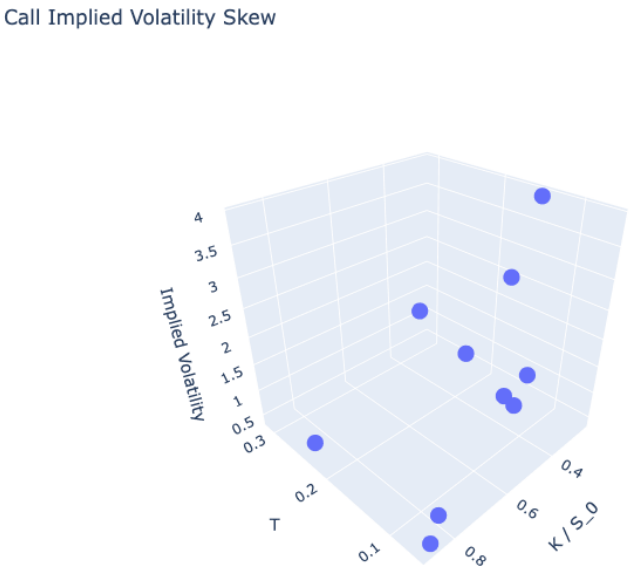

Figure 4: Call Options $((K/S_0, T, \sigma))$

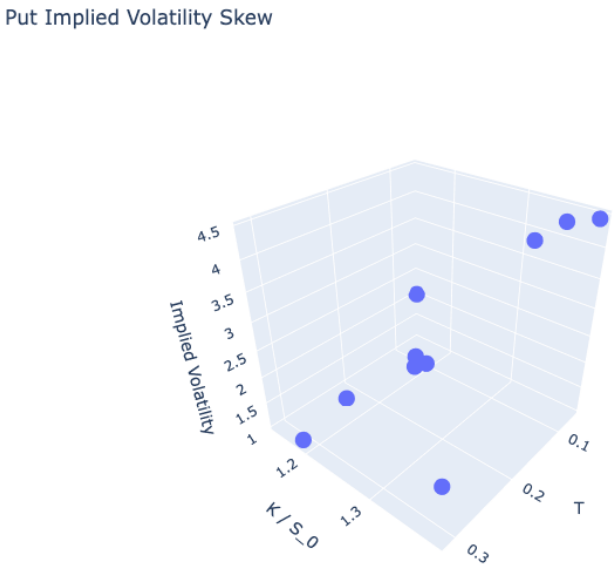

Figure 5: Put Options ($(K/S_0, T, \sigma)$)

While observing the plots, it became obvious that longer expiration times T produced lower implied volatility overall and would lower still as the strike price to stock price ratio K/S_0 grew closer to 1. However, the closer K/S_0 is to 1, the smaller T can be without increasing the implied volatility. While the option types being opposites of one another, and therefore having flipped ratios (Put options desiring a value greater than 1 and Call options a value less than 1) they follow a similar trend of safer options being ones with longer expiration times and strike prices that are close to the current stock price.

Comparing the implied volatility to the historic volatility of the stock from the last 30 days, we observe that the implied volatility is much higher than the calculated historic volatility of 14.21%, and implies that there is a lot of price fluctuation expected in the future by comparison.

Problem 2

Using hedging when we have an option with α shares, we can reduce the Vega exposure by adding a second option with β shares. If the second option has a high strike price K that is far from the stock value S_0, then we need to choose a higher β and a lower β if the opposite is true and K is close to S_0. If the time to maturity T is short, we need to choose a higher β and if there is a long time to maturity, we need to choose a lower β.

If we change β, then we must also change α in order to balance the portfolio and neutralize the delta risk of each option. Given the information we know about implied volatility we can make more accurate hedges by purchasing options where the strike price K is close to S_0 and we choose a maturity time T such that we minimize the implied volatility. Given these choices, we can calculate α and β to create a more accurate and effective portfolio.

Attempting this strategy in a hypothetical portfolio simulation of a week of trading, I chose hypothetical values of $S_0 = 5000$, $K_1 = 4900$, $K_2 = 5100$, $T = 0.25$, $r = 0.02$, $\sigma_1 = 0.2$, and $\sigma_2 = 0.25$. Simulated over the 5 days that would be in a trading week, I graphed the portfolio performance in Figure 6.

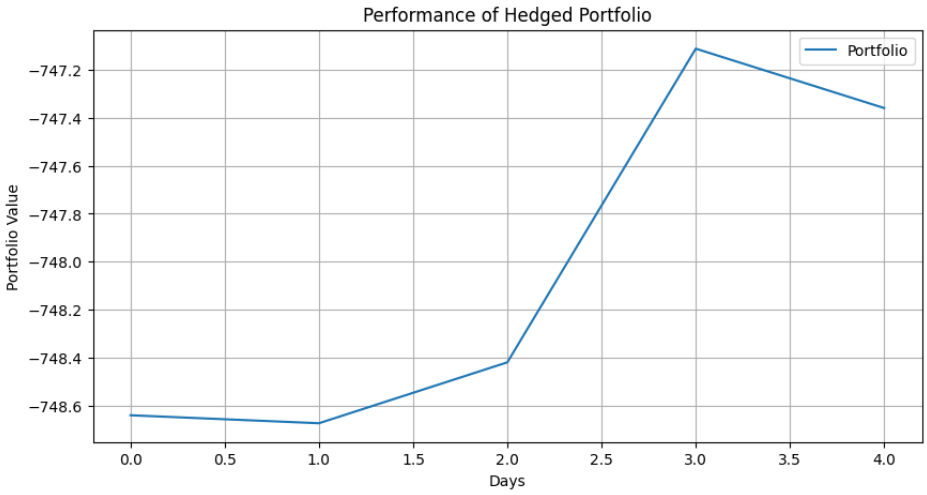

Figure 6: Hedged Portfolio

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as si
import yfinance as yf
import plotly.express as px
from scipy.stats import norm
from scipy.optimize import root_scalar

# Section 1 Problem 1a

T = 1
M = 1000
K = 10000
dt = T / M

dW = np.random.normal(0, np.sqrt(dt), (K, M))
W = np.hstack((np.zeros((K, 1)), np.cumsum(dW, axis=1)))

Ito_integral_x = np.sum(W[:, :-1] * dW, axis=1)

Ito_integral_x2 = np.sum((W[:, :-1]**2) * dW, axis=1)

time = np.linspace(0, T, M)

mean_I_x = np.mean(W[:, :-1] * dW, axis=0)
var_I_x = np.var(W[:, :-1] * dW, axis=0)

mean_I_x2 = np.mean((W[:, :-1]**2) * dW, axis=0)
var_I_x2 = np.var((W[:, :-1]**2) * dW, axis=0)

# Prepare data for visualization
```

```
df_results = pd.DataFrame({
    'Time': time,
    'Mean_I_x': mean_I_x,
    'Var_I_x': var_I_x,
    'Mean_I_x2': mean_I_x2,
    'Var_I_x2': var_I_x2
})

plt.figure(figsize=(8, 6))
plt.plot(time, mean_I_x, label='Mean x')
plt.plot(time, var_I_x, label='Variance x')
plt.plot(time, mean_I_x2, label='Mean x^2')
plt.plot(time, var_I_x2, label='Variance x^2')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Ito Mean and Variance')
plt.legend()
plt.show()

# Section 1 Problem 1b

strat_x = np.sum(0.5 * (W[:, :-1] + W[:, 1:]) * dW, axis=1)

strat_x2 = np.sum(0.5 * ((W[:, :-1]**2) + (W[:, 1:]**2)) * dW, axis=1)

mean_strat_x = np.mean(0.5 * (W[:, :-1] + W[:, 1:]) * dW, axis=0)
var_strat_x = np.var(0.5 * (W[:, :-1] + W[:, 1:]) * dW, axis=0)

mean_strat_x2 = np.mean(0.5 * ((W[:, :-1]**2) + (W[:, 1:]**2)) * dW,
   axis=0)
var_strat_x2 = np.var(0.5 * ((W[:, :-1]**2) + (W[:, 1:]**2)) * dW,
   axis=0)

df_strat_results = pd.DataFrame({
    'Time': time,
    'Mean_strat_x': mean_strat_x,
    'Var_strat_x': var_strat_x,
    'Mean_strat_x2': mean_strat_x2,
    'Var_strat_x2': var_strat_x2
})

plt.figure(figsize=(8, 6))
plt.plot(time, mean_strat_x, label='Mean x')
plt.plot(time, var_strat_x, label='Variance x')
plt.plot(time, mean_strat_x2, label='Mean x^2')
plt.plot(time, var_strat_x2, label='Variance x^2')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Stratonovich Mean and Variance')
plt.legend()
plt.show()

# Section 1 Problem 1c
```

```
r = 0.03
sigma_A = 0.2
sigma_B = 0.3
rho = 0.5
S_0_A = 50
S_0_B = 50

W_1 = np.random.randn(K, M)
W_2 = rho * W_1 + np.sqrt(1 - rho**2) * np.random.randn(K, M)

dt_sqrt = np.sqrt(dt)

S_A = np.zeros((K, M + 1))
S_B = np.zeros((K, M + 1))
S_A[:, 0] = S_0_A
S_B[:, 0] = S_0_B

for j in range(M):
    S_A[:, j+1] = S_A[:, j] * np.exp((r - 0.5 * sigma_A**2) * dt +
       sigma_A * dt_sqrt * W_1[:, j])
    S_B[:, j+1] = S_B[:, j] * np.exp((r - 0.5 * sigma_B**2) * dt +
       sigma_B * dt_sqrt * W_2[:, j])

V_T = np.maximum(S_B[:, -1] - S_A[:, -1], 0)

exchange_option_value = np.exp(-r * T) * np.mean(V_T)

print(f'Exchange Option Value: {exchange_option_value}')

# Section 1 Problem 2a

S_0 = 50
mu = 0.05
sigma = 0.25
T = 1
dt = 1/252
M = int(T/dt)
n = 1000

np.random.seed(42)
dW = np.random.normal(0, np.sqrt(dt), (M, n))
S = np.zeros((M+1, n))
S[0] = S_0

for t in range(1, M+1):
    S[t] = S[t-1] * np.exp((mu - 0.5 * sigma**2) * dt + sigma * dW[t
       -1])

plt.figure(figsize=(8, 6))
plt.plot(S[:, :1000], alpha=0.7)
plt.xlabel('Time')
plt.ylabel('Price')
plt.title('Simulated Price Paths')
plt.show()
```

```
# Section 1 Problem 2b

R_i = np.log(S[1:] / S[:-1])
R_hat = np.mean(R_i, axis=0)
RV = np.sum((R_i - R_hat)**2, axis=0)

variance = sigma**2 * T

df_variance = pd.DataFrame({
    'Realized Variance': RV,
    'Theoretical Variance': variance
})

print(df_variance['Realized Variance'].describe())
df_variance

# Section 1 Problem 2c

kappa = 1
theta = 0.02
sigma_V = 0.2
rho = -0.5
V_0 = 0.0625

np.random.seed(42)
dW_1 = np.random.normal(0, np.sqrt(dt), (M, n))
dW_2 = rho * dW_1 + np.sqrt(1 - rho**2) * np.random.normal(0, np.sqrt(
   dt), (M, n))

V = np.zeros((M+1, n))
S_Heston = np.zeros((M+1, n))

V[0, :] = V_0
S_Heston[0, :] = S_0

for t in range(1, M+1):
    V[t, :] = np.maximum(V[t-1, :] + kappa * (theta - V[t-1, :]) * dt
       + sigma_V * np.sqrt(V[t-1, :]) * dW_2[t-1, :], 0)
    S_Heston[t, :] = S_Heston[t-1, :] * np.exp((mu - 0.5 * V[t-1, :])
       * dt + np.sqrt(V[t-1, :]) * dW_1[t-1, :])

quadratic_variation = np.sum((np.sqrt(V[:-1, :]) * dW_1) ** 2, axis=0)
sigma_quadratic_variation = np.var(quadratic_variation)

print(f'Variance of Quadtatic Variation: {sigma_quadratic_variation}')

# Section 2 Problem 1

spx = yf.Ticker('^SPX')

expiration = ['2025-03-31', '2025-04-30', '2025-05-30', '2025-06-30']

call_options = pd.concat([spx.option_chain(exp).calls.assign(
```

```
    expiration=exp) for exp in expiration])
call_options = call_options[call_options['lastPrice'] > 800]
call_sample = call_options.sample(10)

put_options = pd.concat([spx.option_chain(exp).puts.assign(expiration=
    exp) for exp in expiration])
put_options = put_options[put_options['lastPrice'] > 800]
put_sample = put_options.sample(10)

def black_scholes_call(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.
        sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)

def implied_volatility(S, K, T, r, market_price):
    try:
        result = root_scalar(lambda sigma: black_scholes_call(S, K, T,
            r, sigma) - market_price, bracket=[0.001, 5], method='
            brentq')
        return result.root
    except ValueError:
        return np.nan

r = 0.03
S_0 = spx.history(period='1d')['Close'].iloc[-1]

call_sample['T'] = (pd.to_datetime(call_sample['expiration']) - pd.
    Timestamp.today()).dt.days / 365
call_sample.loc[:, 'Implied Volatility'] = call_sample.apply(
    lambda row: implied_volatility(S_0, row['strike'], row['T'], r,
        row['lastPrice']), axis=1
)

put_sample['T'] = (pd.to_datetime(put_sample['expiration']) - pd.
    Timestamp.today()).dt.days / 365
put_sample.loc[:, 'Implied Volatility'] = put_sample.apply(
    lambda row: implied_volatility(S_0, row['strike'], row['T'], r,
        row['lastPrice']), axis=1
)

fig1 = px.scatter_3d(x=call_sample['T'], y=call_sample['strike'] / S_0
    , z=call_sample['Implied Volatility'],
                     labels={'x':'T', 'y':'K / S_0', 'z':'Implied
                         Volatility'}, title='Call Implied Volatility
                         Skew')
fig1.update_layout(width=800, height=600)
fig1.show()

fig2 = px.scatter_3d(x=put_sample['T'], y=put_sample['strike'] / S_0,
    z=put_sample['Implied Volatility'],
                     labels={'x':'T', 'y':'K / S_0', 'z':'Implied
                         Volatility'}, title='Put Implied Volatility
                         Skew')
```

```
fig2.update_layout(width=800, height=600)
fig2.show()

data = yf.download('^GSPC', start='2025-01-23', end='2025-03-06')

data['Log Returns'] = np.log(data['Close'] / data['Close'].shift(1))

std_dev = data['Log Returns'].std()

annualized_volatility = std_dev * np.sqrt(252)

print(f"Annualized Historical Volatility: {annualized_volatility:.2%}"
    )

# Section 2 Problem 2

def black_scholes(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.
        sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == 'call':
        price = S * si.norm.cdf(d1) - K * np.exp(-r * T) * si.norm.cdf
            (d2)
    elif option_type == 'put':
        price = K * np.exp(-r * T) * si.norm.cdf(-d2) - S * si.norm.
            cdf(-d1)
    else:
        raise ValueError('Invalid option type. Choose 'call' or 'put'.
            ')

    return price

def delta(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.
        sqrt(T))
    if option_type == 'call':
        return si.norm.cdf(d1)
    else:
        return si.norm.cdf(d1) - 1

def vega(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.
        sqrt(T))
    return S * np.sqrt(T) * si.norm.pdf(d1)

np.random.seed(42)
days = 5
S0 = 5000
K1, K2 = 4900, 5100
T = 0.25
r = 0.02
sigma1, sigma2 = 0.2, 0.25
```

```
returns = np.random.normal(r / 252, sigma1 / np.sqrt(252), days)
SPX_prices = S0 * np.cumprod(1 + returns)

option1_price = black_scholes(S0, K1, T, r, sigma1, 'call')
option2_price = black_scholes(S0, K2, T, r, sigma2, 'call')

delta1 = delta(S0, K1, T, r, sigma1, 'call')
delta2 = delta(S0, K2, T, r, sigma2, 'call')

vega1 = vega(S0, K1, T, r, sigma1)
vega2 = vega(S0, K2, T, r, sigma2)

beta = -vega1 / vega2

alpha = delta1 + beta * delta2

portfolio_values = []
for S in SPX_prices:
    V1 = black_scholes(S, K1, T, r, sigma1, 'call')
    V2 = black_scholes(S, K2, T, r, sigma2, 'call')
    portfolio_value = V1 + beta * V2 - alpha * S
    portfolio_values.append(portfolio_value)

plt.figure(figsize=(8, 6))
plt.plot(portfolio_values, label='Portfolio')
plt.xlabel('Days')
plt.ylabel('Portfolio Value')
plt.title('Performance of Hedged Portfolio')
plt.legend()
plt.grid(True)
plt.show()

df_summary = pd.DataFrame({
    'Day': np.arange(1, days + 1),
    'SPX Price': SPX_prices,
    'Portfolio Value': portfolio_values
})
```