

Financial Modelling: How to invest in a Portfolio?

umaygok, David Ou, Martyna Wojciechowska

*

1 Introduction

Have you ever thought about investing in stocks or about how to predict the stock price to maximize your profit return on stock investment? In this paper, we discuss mathematical tools that may help you make more informed decisions regarding your investments. Alternatively, we propose Geometric Brownian motion (GBM) which is a mathematical model that simulates continuous-time stochastic processes. GBM is widely used in finance to describe random fluctuations in a stock price over time. Further, we present a stochastic model that simulates multiple four-correlated stock prices under GBM, incorporating both systemic and idiosyncratic shocks. The four stocks that we will be investigating are Microsoft, IBM, Starbucks, and Apple.

This paper is structured as followed: Section 2 includes a simple model used as an introduction to financial analysis with the use of statistical tools like GBM. To create more realistic scenarios, we propose a more complex model in Section 3. To bring in more practical example, we then introduce a dataset with real world stock data. We select a time period up until 2014, and run a number of Monte Carlos analysis to infer about the future stock's behaviour. We also make analysis useful for portfolio planning. The results of that analysis are in Section 4. Section 5 proposes further analysis and final results. The code used to create all the visualizations and analysis is accessible in the Appendix 6.

2 Simple Model

2.1 Geometric Brownian Motion (GBM)

Geometric Brownian Motion (GBM) is a stochastic process, often used in the financial industry to model the stocks' behaviour. It assumes that stock prices follow a combination of deterministic growth and random fluctuations.(Ross, 2014)

The stochastic differential equation (SDE) defining GBM is:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \quad (1)$$

where:

S_t is the stock price at time t ,

μ is the drift rate (expected return of the stock),

σ is the volatility (degree of randomness in the price movements),

dW_t represents a standard Brownian motion (Wiener process).

Using Itô's Lemma, we derive the explicit solution:

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t} \quad (2)$$

where:

- S_0 is the initial stock price,
- $e^{(\mu - \frac{1}{2}\sigma^2)t}$ represents deterministic growth,
- σW_t introduces randomness in the stock price.

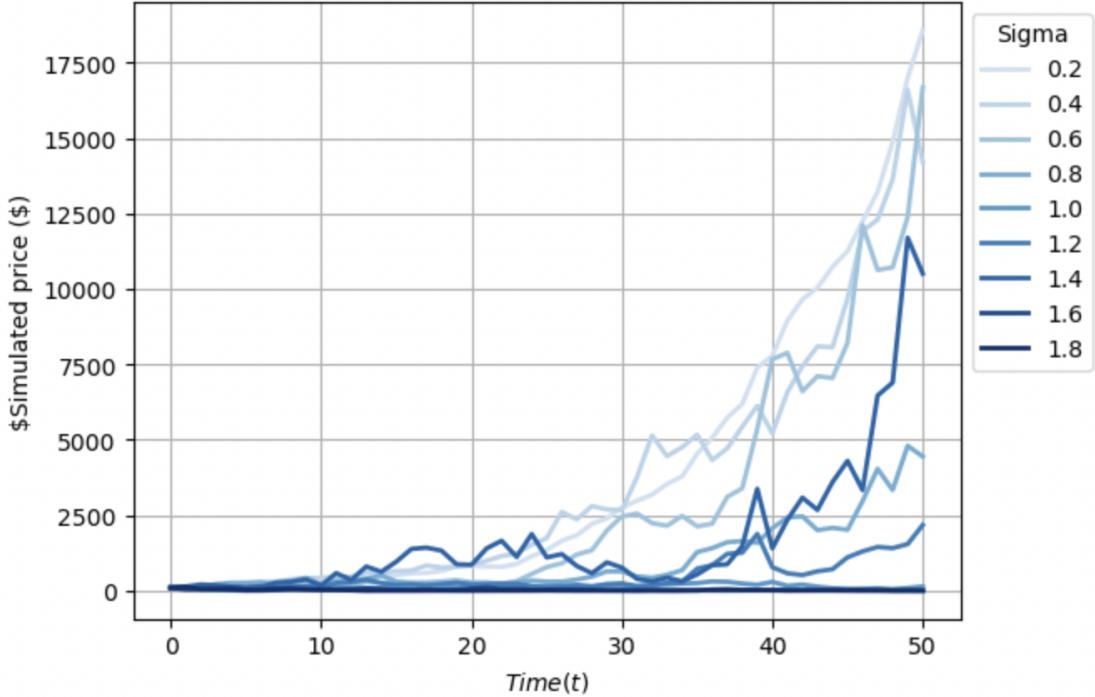


Figure 1: Realizations of Geometric Brownian Motion with different variances $\mu = 1$. For the code, see Appendix I

Figure 1 displays simulated data for an arbitrary stock. The drift parameter μ is fixed at 1, while the volatility, represented by the standard deviation σ , vary between 0.2 and 1.8. Depending on the volatility, the graphs differ significantly. The randomness is introduced by dW_t , which follows a normal distribution with mean 0 and variance dt .

2.2 Portfolio Value Computation

The total portfolio value at time t is defined as the equally weighted sum of all stock prices:

$$P_t = \frac{1}{n} \sum_{i=1}^n S_{i,t}. \quad (3)$$

This allows us to track the overall gain and loss of an investor who has a diversified portfolio.

2.3 Geometric Brownian Motion (GBM) for Multiple Stocks

The price dynamics of the stock S_i in GBM is given by (Musiela and Rutkowski, 2004)

$$dS_{i,t} = \mu_i S_{i,t} dt + \sigma_i S_{i,t} dW_{i,t}, \quad (4)$$

where:

- $S_{i,t}$ is the price of stock i at time t ,
- μ_i is the drift rate (expected return) of stock i ,
- σ_i is the volatility of stock i ,
- $dW_{i,t}$ is a standard Brownian motion term.

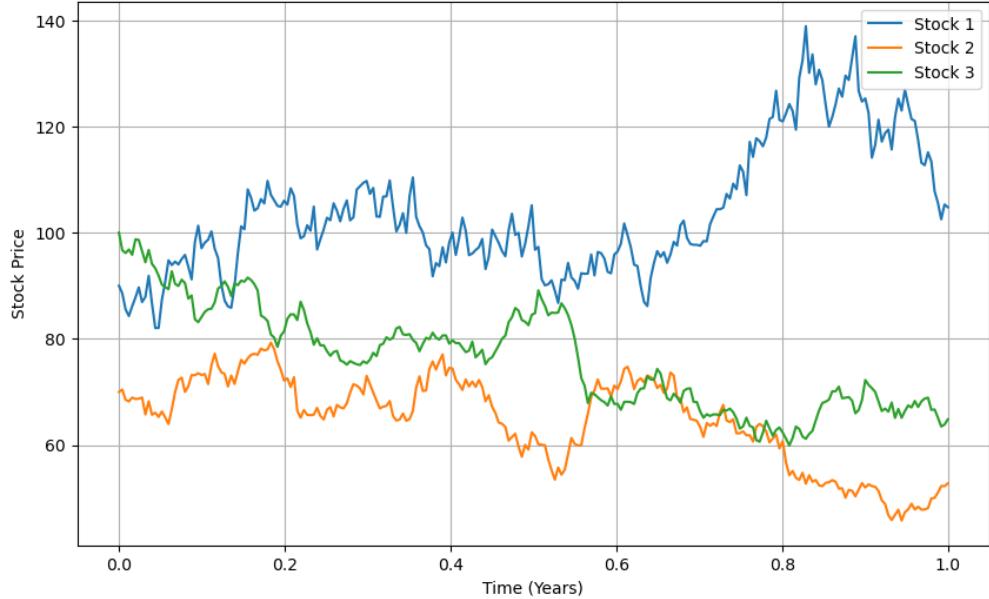


Figure 2: Multiple Stock GBM Simulation. For code, see Appendix II.

Figure 2 shows the simulated price movements of three randomly selected stocks following a Geometric Brownian Motion (GBM) model. Each stock has different levels of volatility and drift selected by us. Each one of the stocks follows a different trajectory because of these different parameters. From Figure 2, we can see Stock 1 (blue) shows an overall increasing trend, which indicates a higher drift rate. Stock 2 (orange) declines over time, and this indicates a negative or lower drift. Stock 3 (green) fluctuates but remains relatively stable. Thus, we can see how different μ and σ choices affect stock price behavior in stochastic modeling.

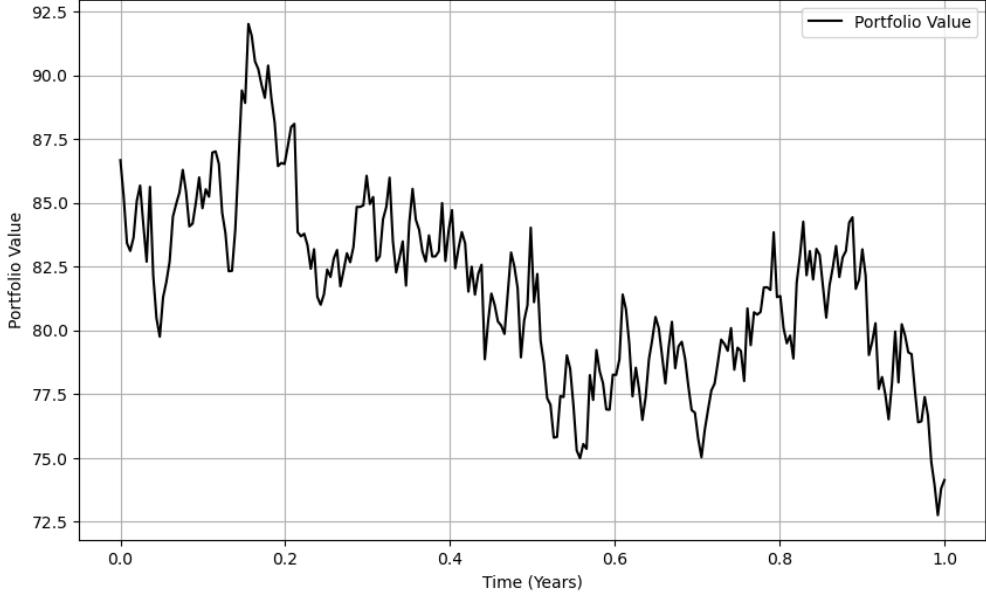


Figure 3: Multiple Stock Portfolio Simulation. For code, see Appendix II.

Figure 3 represents the portfolio value of 3 randomly selected stocks over time which is calculated by giving the same weight to all of the stocks. We calculate this portfolio values from equation (3). The portfolio experiences fluctuations and an overall downward trend which indicates that the combined effect of the stocks did not yield positive returns. From the declining value, we can see that stocks with lower or negative drift rates had a significant influence. Also, we can say that the volatility led to losses over time. If we have given different weights to the stock with more positive drift rate, our overall portfolio value could give us a better performance and outcome.

2.4 Correlation of Multiple Stocks with GBM

The Brownian motions $W_{i,t}$ are correlated according to a correlation matrix Σ , such that:

$$dW_t = LZ_t\sqrt{dt}, \quad (5)$$

where L is obtained by Cholesky decomposition of Σ and Z_t is a vector of independent standard normal variables.(QuantEssence, n.d.)

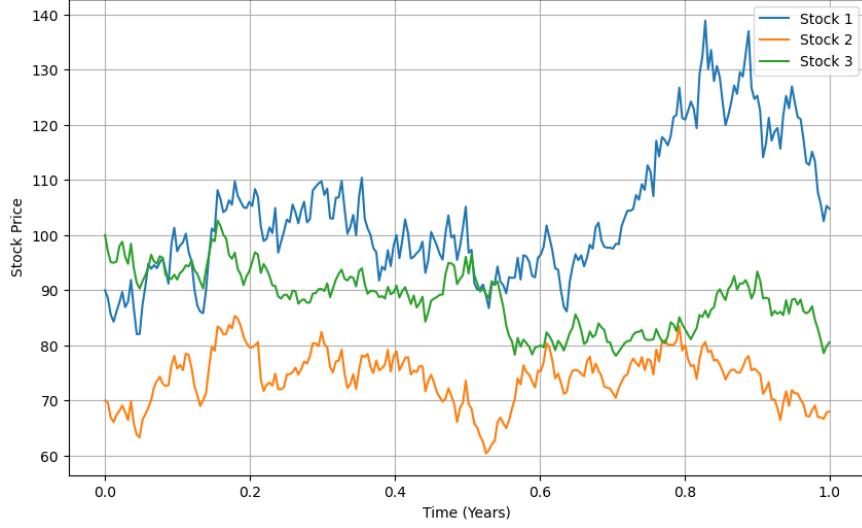


Figure 4: Multiple Stock with Correlation GBM Simulation. For code, see Appendix III.

With the added correlation between the stocks Figure 4, we can say that the stocks are not following independent trajectories. They are influenced by shared external factors such as economic or political conditions. Now, the stocks follow more synchronized movements, where trends and fluctuations shows some similarity. We can see that if two stocks have a strong positive correlation, their prices may increase or decrease together, while negatively correlated stocks may move in opposite directions. Correlation gives us a more realistic market behavior, since stocks are affected by broader systemic risks rather than purely independent random fluctuations.

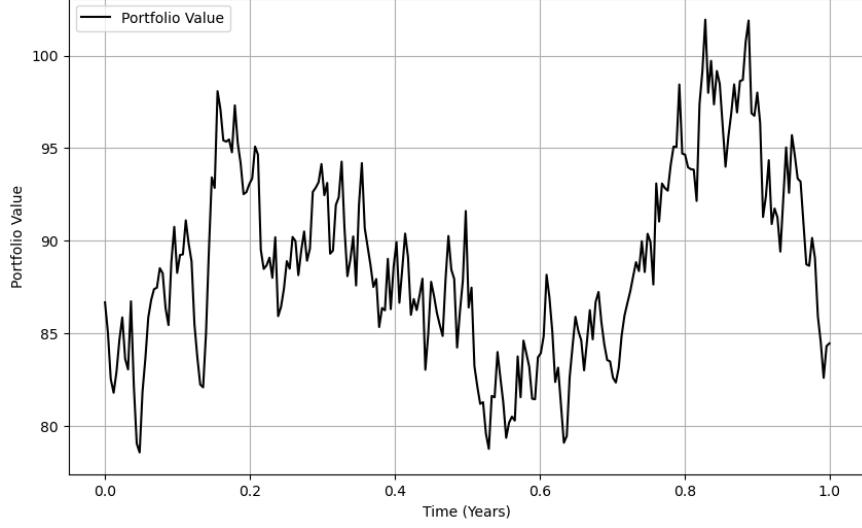


Figure 5: Multiple Stock Portfolio with Correlation Simulation. For code, see Appendix III.

After we add the correlation factor to the stocks, the portfolio value now reflects the impact of stock correlations, meaning the overall performance is influenced not just by individual stock behaviors but also by how they interact with each other. If the stocks are positively correlated, large swings in one stock may be mirrored by others, and this will increase the volatility in the portfolio. Otherwise,

if there are negatively correlated stocks, some losses may be offset by gains in other stocks, and this will lead to a more stable portfolio. From Figure 5, the overall trend of the portfolio value suggests whether the combined correlation structure contributes to diversification benefits at some point in time where without correlation the overall portfolio gains from Figure 3 was leading the portfolio to losses, now we can have more gains but having dependencies among the stocks still amplifies risk, since there are more fluctuations.

3 Added Complexity to the Model

3.1 Shock Modelling

Our simple model introduced in Section 2 successfully showcases simplified stock behaviour. To simulate more realistic scenarios, we need to add more advanced statistical tools, as GBM is not enough to capture the 'unpredictable' stocks' fluctuations (PyQuant, 2024). To add the complexity, we introduce Poisson Jumps, and we specifically follow the approach of (Quant Next, 2024). We use the Merton Jump-Diffusion (MJD) model to introduce the complexity. Unlike the continuous-time method, we propose the use of discrete time steps where jumps occur according to Poisson Distribution.

$$dS_t = \mu S_t dt + \sigma S_t dW_t + \sigma S_t dJ_t \quad (6)$$

and

$$J_t = \sum_{j=1}^{N_t} Y_j \quad (7)$$

The final solution derived is:

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t + J_t} \quad (8)$$

where

Y_i : jump sizes following a Normal distribution: $Y_i \sim \mathcal{N}(m, \delta^2)$, where m represents a mean of the jump size, and δ^2 represents the standard deviation on the jump size,

N_t : follows Poisson Distribution and determines the time of the jump. If $N_t > 0$ a jump occurs, otherwise there is no jump event.

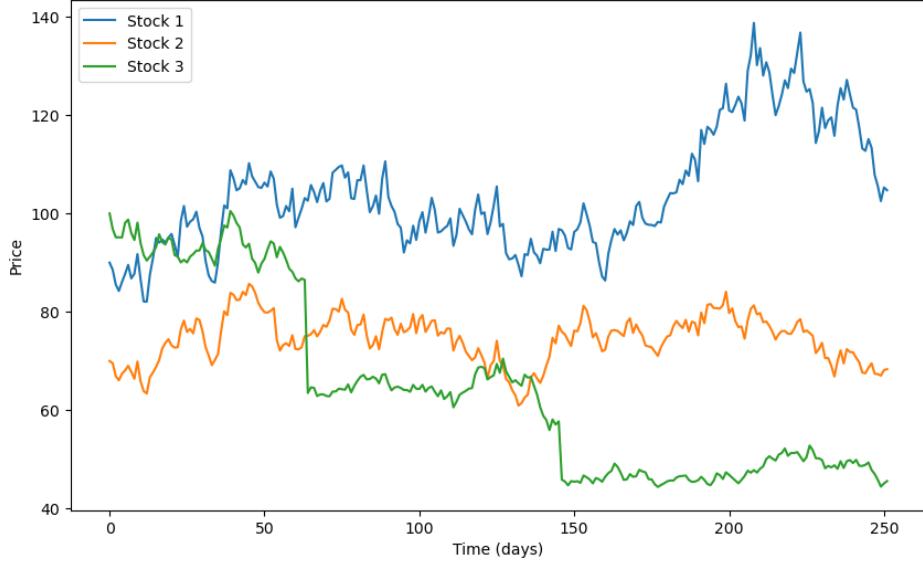


Figure 6: Multiple Stock with Shocks and Correlation GBM Simulation. For code, see Appendix IV.

To add more complexity and make our model more similar with the real-world examples, we have added shocks to our model using Poisson Distribution. After we add the shocks, we can see that there are some sudden jumps(decreases) in Figure 6. These shocks' time and magnitude can be determined with the past data we currently have but these cannot give a good representation of what can happen in the real-world. Since stocks can be affected by so many different factors such as political or economical issues, there is no way to simulate it with 100% accuracy. To simulate the shocks we have used a parameter for number of shocks should appear each year, their magnitudes μ of the jump and σ of the jump. And then to make it more accurate we have used them randomly.

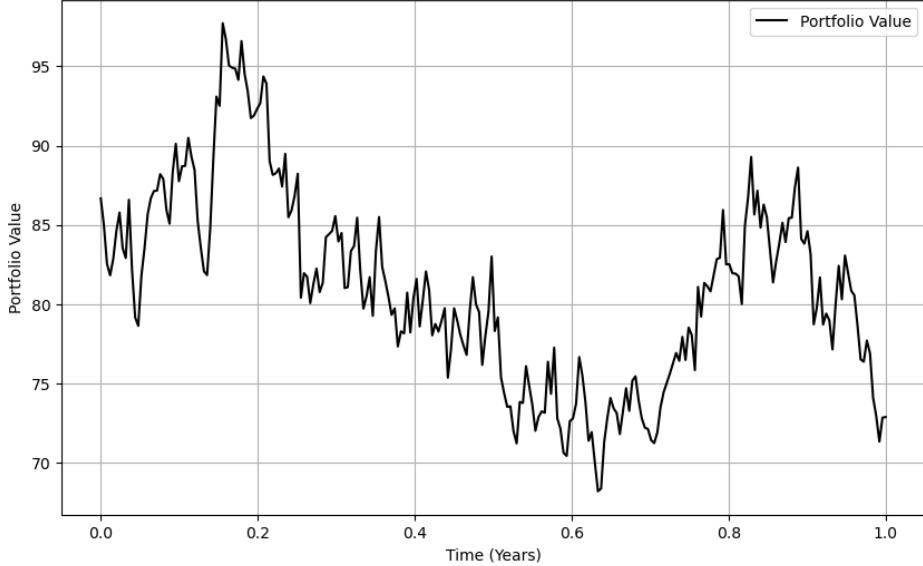


Figure 7: Multiple Stock Portfolio with Shocks and Correlation Simulation. For code, see Appendix IV.

After we add the shock factor to the stocks, the portfolio value now reflects the impact of sudden changes to the individual stocks. Since we still have correlation in our model, each stock is correlated while some of them experiencing some sudden drops or jumps, the others are going to be affected by these drops and jumps. This gives us more fluctuating portfolio value model, which can also be seen in Figure 7. Since there will be abrupt and big drops and jumps in the stock models, these drops will also be seen in the portfolio value. If these drops appear at the same time and towards the same direction (can happen because of some outside factors such as if the whole market crashes), this drop of the stocks will be seen as a bigger drop in the overall portfolio value. With the added shocks we are able to create a more realistic model that can also regard for changes which can happen in real world, rather giving us a fully imaginary stock market.

4 Monte-Carlos Simulations of the Stocks

In this section, we created Monte Carlos simulation of the Microsoft, IBM, Starbucks, and Apple stock using parameters aquired from real life data and computed the average of all the simulations for each of the stock. Below is the real-world data collected about the stocks of these companies from 2007 and 2016. We have taken the data from 2007 up until 2014.

The [Stock Dataset](#) used in the model can be found here.

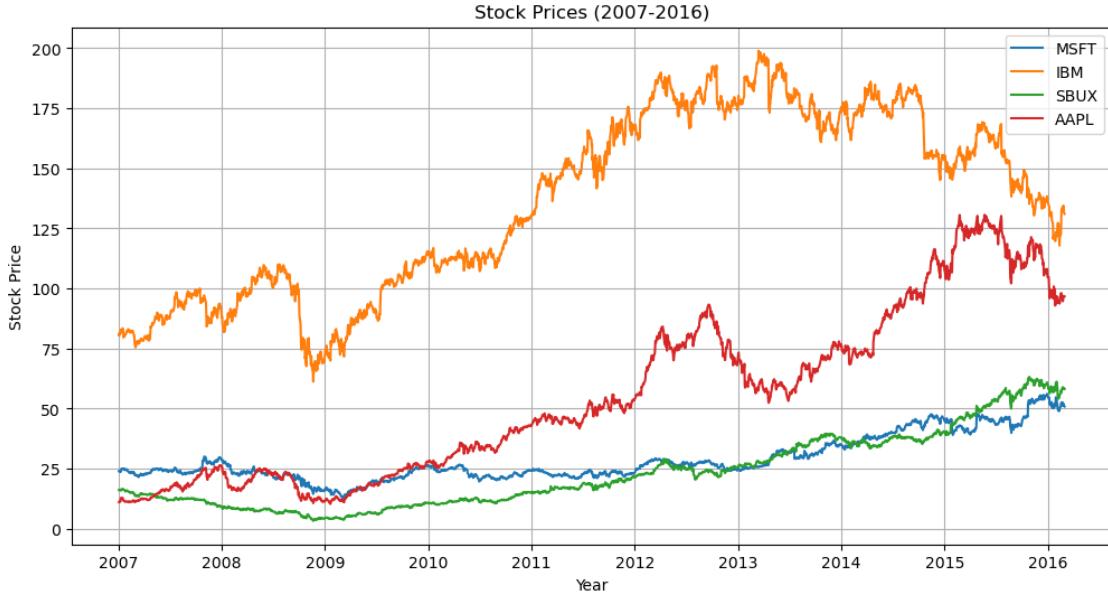


Figure 8: Real-World Stock Data of Microsoft, IBM, Starbucks, and Apple. For code, see Appendix V.

For each of the stocks above, we use a negative log likelihood function to determine the optimal parameters μ and σ . The negative log-likelihood (NLL) function is a loss function that determines how a model does not fit the real data. Hence, we wrote a code that minimizes the results of the loss function to determine the most optimal parameters.

When we acquired the most optimal parameters for each of the four stocks, we created a Monte-Carlo simulation to see how each stock can behave. For each one of the stocks we have a Monte-Carlo Simulation and a Histogram at the end of the simulation to be able to determine the characteristics of the possible stock behavior.

To make the results of Monte Carlo simulations more visualizable and understandable, we have calculated the estimated values of μ and σ for all four stocks. For the code of the estimations and Monte-Carlo Simulations, see Appendix VI.

The estimated values are:

- **Microsoft (MSFT):** Estimated $\mu : 0.0003$, Estimated $\sigma : 0.0182$
- **IBM:** Estimated $\mu : 0.0003$, Estimated $\sigma : 0.0145$
- **Starbucks (SBUX):** Estimated $\mu : 0.0005$, Estimated $\sigma : 0.0219$
- **Apple (AAPL):** Estimated $\mu : 0.0011$, Estimated $\sigma : 0.0218$

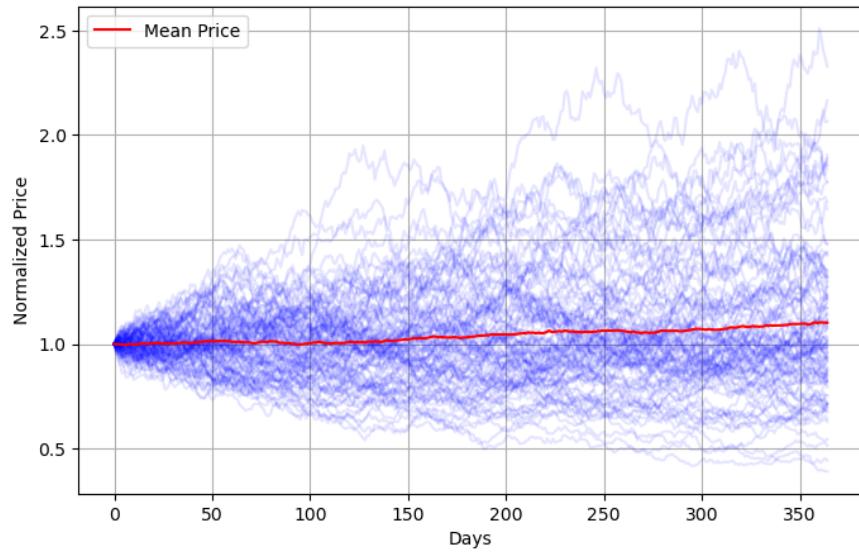


Figure 9: Monte Carlo Simulation of Microsoft (MSFT) $\mu = 0.0003, \sigma = 0.0182$

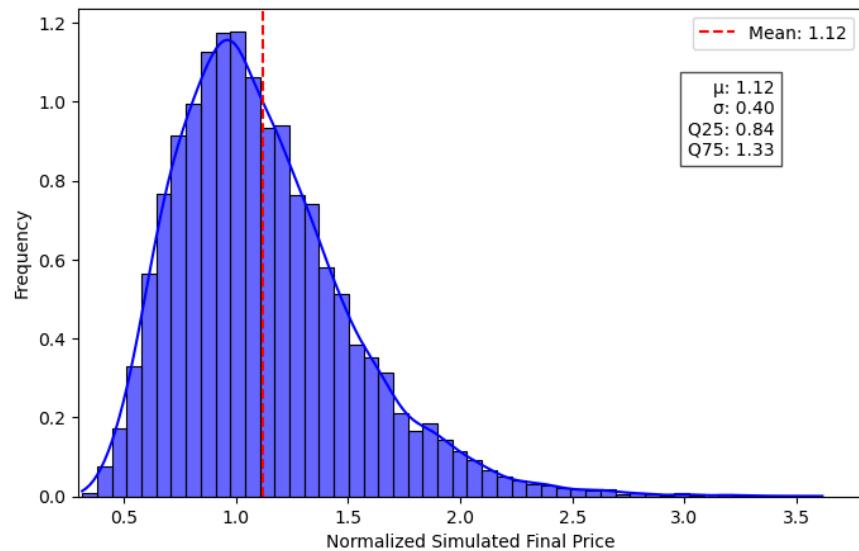


Figure 10: Monte Carlo Simulation Histogram of Microsoft (MSFT)

The Figure 10, shows that 'a year from now' the Microsoft stock will likely increase relative to now, as the mean is 1.12 (normalized price). It's only 25% chance that the stock will end up higher than 1.33 of the current price. It seems to be quite stable, and we don't note trends suggesting rapid spikes in price.

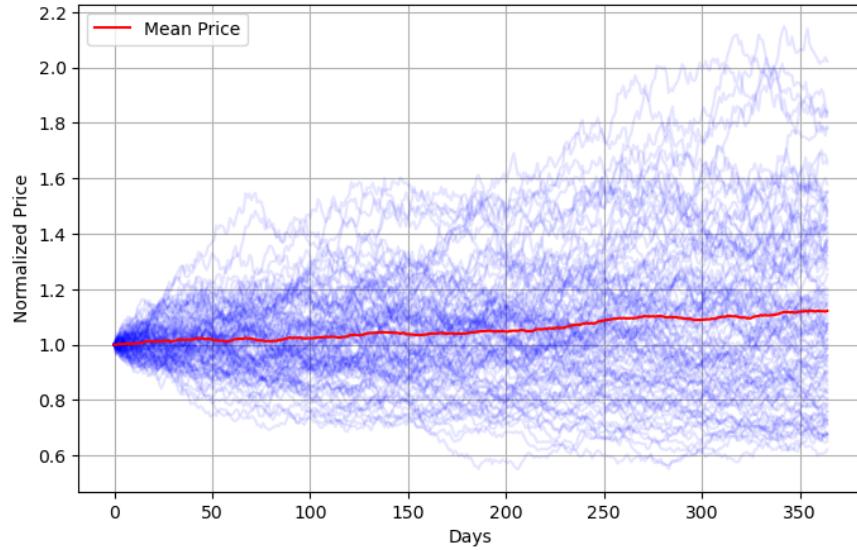


Figure 11: Monte Carlo Simulation of IBM $\mu = 0.0003, \sigma = 0.0145$

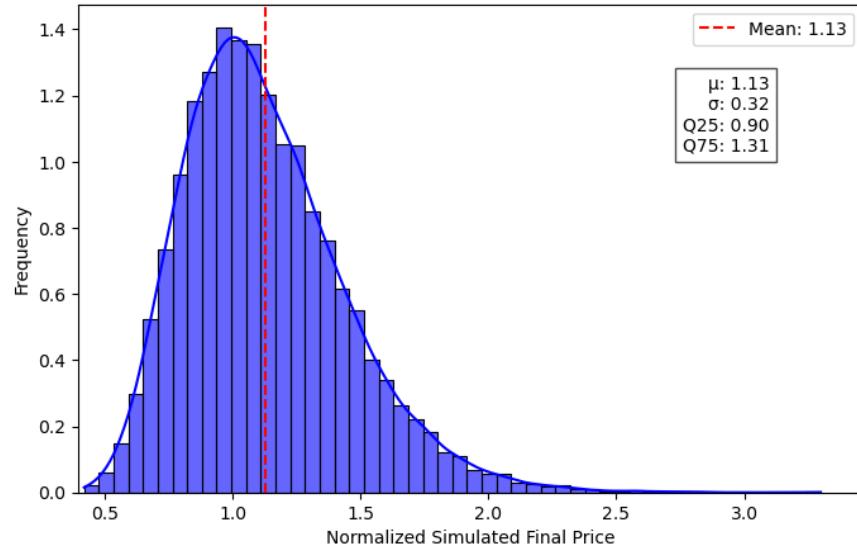


Figure 12: Monte Carlo Simulation Histogram of IBM

Our analysis (Figure 11) similarly suggest that IBM's stock values are likely to increase as the mean is 1.13. Similarly to Microsoft stock, it is very stable and unlikely to either fall or increase rapidly.

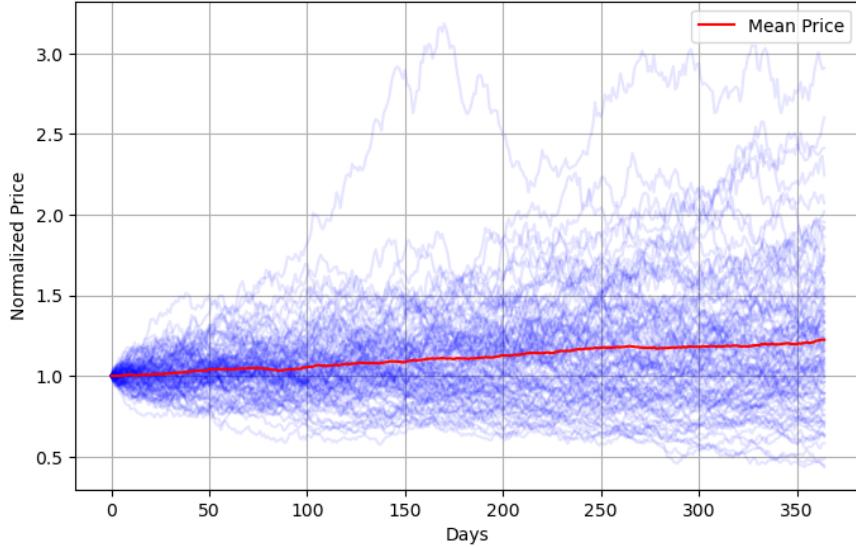


Figure 13: Monte Carlo Simulation of Starbucks (SBUX) $\mu = 0.0005, \sigma = 0.0219$

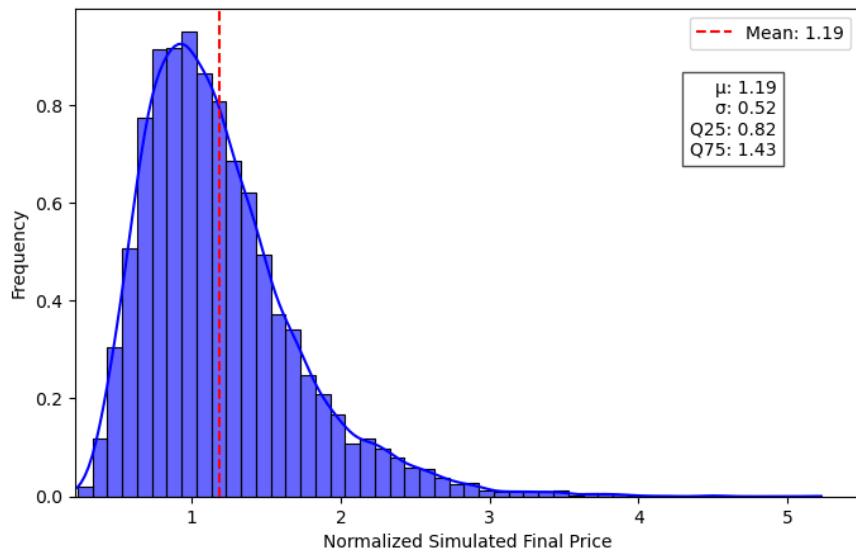


Figure 14: Monte Carlo Simulation Histogram of Starbucks (SBUX)

Our analysis visualized on Figure 12, shows that Starbucks has the likelihood of the highest returns as the mean is at 1.19. It could be a good investment for those who are interested in greater risk and greater returns, as there is 25% chance that the stock will end up at 1.49 and the same chance that it falls under 0.9 of the current price.

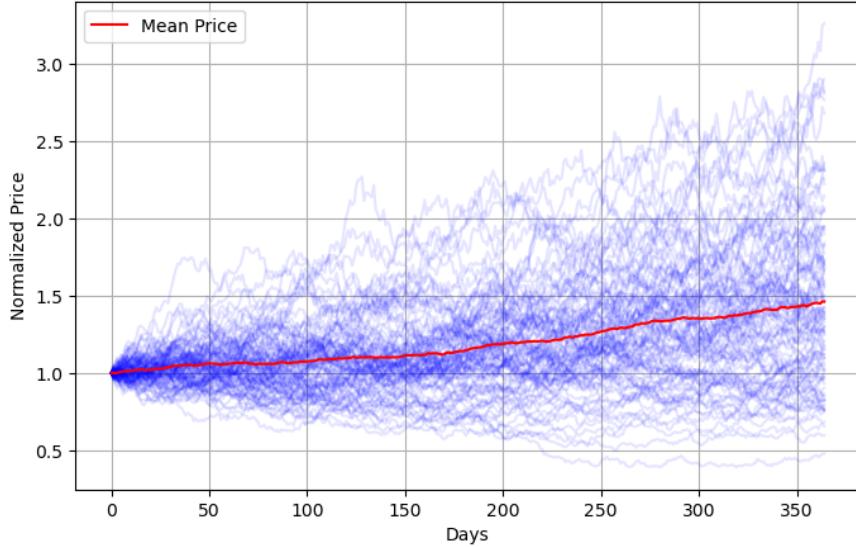


Figure 15: Monte Carlo Simulation of Apple (AAPL) $\mu = 0.0011, \sigma = 0.0218$

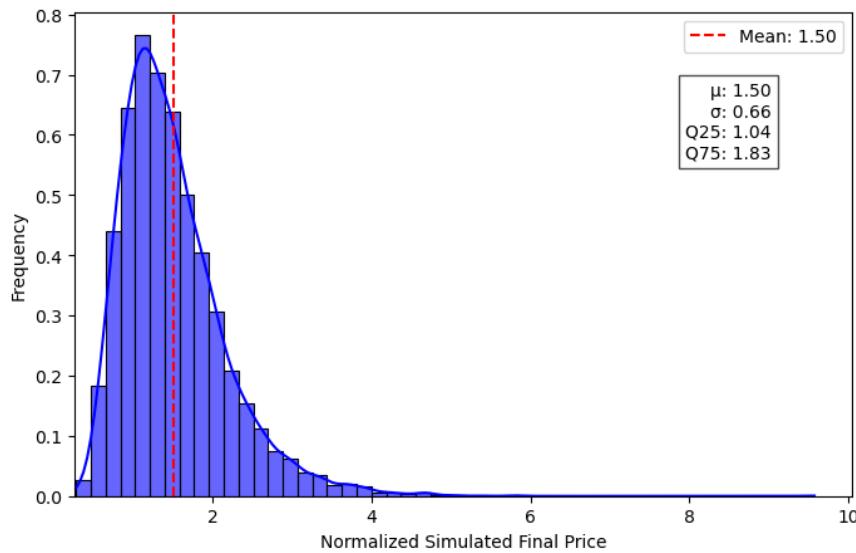


Figure 16: Monte Carlo Simulation Histogram of Apple (AAPL)

The analysis of the Apple stock presents the greatest returns without much-added risk. The mean is said to be at 1.5 of the current price which is a considerable growth. Moreover, it is only 25% likely that the stock will fall under 1.04 of the current price which is the best result out of all considered stocks. There is precisely the same chance, that the value will increase surpass 1.83 of the current price.

5 Results and Analysis

Our Monte Carlo Simulation allow us to infer some information about the volatility of the four stocks. The Apple stock is the most volatile followed by Starbuck, Microsoft, and IBM. The Apple stock has the highest risk and the most reward. On the other hand, the IBM Stock is the most stable since it has the smallest standard deviation and is the least likely to increase nor decrease in value.

A likely explanation of those volatilities, here namely of Apple and Starbuck, may be due to their strong competitions. For Apple, it could be: Samsung, Huawei, Xiaomi, Oppo, Vivo, and LG. Similarly, Tim Hortons, Dukin', McCafe, and Luckin Coffee are also competitors to Starbuck. Moreover, the bar to start a coffee chain isn't high. Companies like Microsoft may be less impacted by the competition factor, as their operation systems are established and fierce on the market.

Monte Carlo Analysis also allowed us to create a method to successfully create a Portfolio. We suggested, that in 2014, investing in Apple stocks could be the most lucrative. Figure 8, shows success in those statements. At the end of 2015, similarly to our predictions, the Apple stock grew rapidly. Microsoft and IBM had steady, perhaps somehow unimpressive growth, while IBM's stocks fell considerably which was missed by our model.

The main result is that the stock market behaviour could be more often than not unpredictable. However, statistical tools like MJD, GBM, Monte Carlo and Software Engineering can help minimize the risks related to trading, and perhaps make better, more informed financial decisions.

6 Conclusion

This model effectively captures the dynamics of stock prices under random fluctuations and sudden market shocks. Future extensions could include variable shock distributions and mean-reverting dynamics post-shock.

References

- Haydock, J. (2020). The impact of sectoral performance on the stock market: Does volatility equal explanatory power? *Bryant Economic Research Paper*. <https://digitalcommons.bryant.edu/cgi/viewcontent.cgi?article=1010&context=eeb>
- Lindskog, F., & McNeil, A. (2001). Common poisson shock models: Applications to insurance and credit risk modelling. *ASTIN Bulletin*, 33. <https://doi.org/10.2143/AST.33.2.503691>
- Lindskog, F., & McNeil, A. J. (2001). Common poisson shock models: Applications to insurance and credit risk. *Astin Bulletin*, 33(2). <https://doi.org/10.2143/AST.33.2.503691>
- Musiela, M., & Rutkowski, M. (2004). *Martingale methods in financial modelling* (2nd). Springer Verlag.
- Nikulin, M. S., Limnios, N., Balakrishnan, N., Kahle, W., & Huber-Carol, C. (2010). *Advances in degradation modeling: Applications to reliability, survival analysis, and finance*. Birkhäuser. <https://doi.org/10.1007/978-0-8176-4924-1>
- Programmer, P. (2023). Forecasting the stock market with python - geometric brownian motion. https://www.youtube.com/watch?v=0uCuvp_cZ9o
- PyQuant. (2024). Poisson jumps: Better pricing with a more realistic simulation. https://www.pyquantnews.com/the-pyquant-newsletter/poisson-jumps-better-pricing-more-realistic-simulation?utm_source=linkedin&utm_medium=post&utm_campaign=7.7.24.8.LI
- Quant Next. (2024). The merton jump diffusion model. <https://quant-next.com/the-merton-jump-diffusion-model/>
- QuantEssence. (n.d.). Generating correlated brownian motions. <https://quantessence.wordpress.com/wp-content/uploads/2012/01/multibrownianmotion.pdf>
- QuantPy. (2025). Geometric brownian motion simulation in python. https://www.youtube.com/watch?v=0uCuvp_cZ9o
- Ross, S. M. (2014). *Introduction to probability models* (11th). Elsevier.
- Wikipedia contributors. (2025). Geometric Brownian motion — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Geometric_Brownian_motion
- with Josh Starmer, S. (2025). Poisson process and the exponential distribution. <https://www.youtube.com/watch?v=jejjUPfaIXY>

Appendices

I Appendix

Python code for Figure 1. (QuantPy, 2025)

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

#PARAMETERS
mu = 1
n = 50
dt = 0.1
x0 = 100
np.random.seed(10)

sigma = np.arange(0.2, 2, 0.2)

#GBM
x = np.exp(
    (mu - sigma ** 2 / 2) * dt
    + sigma * np.random.normal(0, np.sqrt(dt), size=(len(sigma), n)).T
)
x = np.vstack([np.ones(len(sigma)), x])
x = x0 * x.cumprod(axis=0)

#PLOT
colors = cm.Blues(np.linspace(0.2, 1, len(sigma)))
for i in range(len(sigma)):
    plt.plot(x[:, i], color=colors[i], linewidth=2)

# Labels and legend
plt.legend(np.round(sigma, 2), title="Sigma", loc='upper left',
bbox_to_anchor=(1, 1))
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.grid()

plt.show()

#Realizations of Geometric Brownian Motion with different variances\n $\mu=1
```

II Appendix

Python code for Figure 2 and Figure 3.

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

#PARAMTERS
np.random.seed(15)
n_stocks = 3          # Number of stocks
T = 1.0               # Time in years
dt = 1/252             # Time step
N = int(T / dt)        # Number of time steps

# GBM PARAMETERS
mu = np.array([0.05, 0.08, 0.07])      # Expected return (drift)
sigma = np.array([0.5, 0.4, 0.3])        # Volatility
S0 = np.array([90, 70, 100])            # Initial stock prices

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

S = np.zeros((N, n_stocks))
S[0, :] = S0    # Initial Prices

#Independent Brownian Motion
dW = np.random.normal(0, np.sqrt(dt), size=(N, n_stocks))

#GBM
for t in range(1, N):
    dS = mu * S[t-1, :] * dt + sigma * S[t-1, :] * dW[t, :]
    S[t, :] = S[t-1, :] + dS

#Portfolio
portfolio_value = np.sum(S, axis=1) / n_stocks

#PLOT STOCKS
plt.figure(figsize=(10, 6))
for i in range(n_stocks):
    plt.plot(time, S[:, i], label=f'Stock {i+1}')
plt.xlabel('Time (Years)')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()

#PLOT PORTFOLIO
plt.figure(figsize=(10, 6))
```

```
plt.plot(time, portfolio_value, label='Portfolio Value', color='black')
plt.xlabel('Time (Years)')
plt.ylabel('Portfolio Value')
plt.legend()
plt.grid()
plt.show()

# plt.title('Multi-Stock Simulation Without Correlation & Without Shocks (Monte Carlo Formula)')
# plt.title('Overall Portfolio Gain/Loss Without Correlation & Without Shocks (Monte Carlo Formula)')
```

III Appendix

Python code for Figure 4 and Figure 5.(QuantEssence, n.d.)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import cholesky

#PARAMETERS
np.random.seed(15)
n_stocks = 3          # Number of stocks
T = 1.0               # Time in years
dt = 1/252             # Time step
N = int(T / dt)        # Number of time steps

# GBM PARAMETERS
mu = np.array([0.05, 0.08, 0.07])      # Expected return
sigma = np.array([0.5, 0.4, 0.3])        # Volatility
S0 = np.array([90, 70, 100])            # Initial stock prices

#Correlation Matrix
corr_matrix = np.array([
    [1.0, 0.8, 0.6],   #Correlation of [stock 1 - stock 1, stock 1 - stock 2, stock 1 - stock 3]
    [0.8, 1.0, 0.5],   #Stock 2 with others
    [0.6, 0.5, 1.0]    #Stock 3 with others
])

#Cholesky Decomposition
L = cholesky(corr_matrix, lower=True)

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

S = np.zeros((N, n_stocks))
S[0, :] = S0  # Initial Prices

Z = np.random.normal(size=(N, n_stocks))

#Correlated Brownian Motions
dW = np.dot(Z, L.T) * np.sqrt(dt)

#GBM
for t in range(1, N):
    dS = mu * S[t-1, :] * dt + sigma * S[t-1, :] * dW[t, :]
    S[t, :] = S[t-1, :] + dS

portfolio_value = np.sum(S, axis=1) / n_stocks
```

```

# PLOT STOCKS
plt.figure(figsize=(10, 6))
for i in range(n_stocks):
    plt.plot(time, S[:, i], label=f'Stock {i+1}')
plt.xlabel('Time (Years)')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()

# PLOT PORTFOLIO
plt.figure(figsize=(10, 6))
plt.plot(time, portfolio_value, label='Portfolio Value', color='black')
plt.xlabel('Time (Years)')
plt.ylabel('Portfolio Value')
plt.legend()
plt.grid()
plt.show()

#plt.title('Multi-Stock Simulation with Correlation (PDF Method)')
#plt.title('Overall Portfolio Gain/Loss with Correlation (PDF Method)')

```

IV Appendix

Python code for Figure 6 and Figure 7.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import cholesky

#PARAMETERS
np.random.seed(15)
n_stocks = 3
T = 1
dt = 1/252
N = int(T / dt)

#GBM PARAMETERS
mu = np.array([0.05, 0.08, 0.07]) # Drift
sigma = np.array([0.5, 0.4, 0.3]) # Volatility
S0 = np.array([90, 70, 100]) # Initial prices

# Correlation Matrix
corr_matrix = np.array([
    [1.0, 0.8, 0.6],
    [0.8, 1.0, 0.5],
    [0.6, 0.5, 1.0]
])

# Cholesky Decomposition
L = cholesky(corr_matrix, lower=True)

# Jump Parameters
lambda_i = 0.8 # Average number of jumps per year
mu_j = -0.2 # Mean of jump size
sigma_j = 0.2 # Standard deviation of jump size

prices = np.zeros((N, n_stocks))
prices[0, :] = S0

#GBM
Z = np.random.normal(0, 1, (N, n_stocks))
Z = np.dot(Z, L.T) # Apply correlation

for t in range(1, N):
    # Brownian Motion Component
    dW = Z[t, :] * np.sqrt(dt)

    # Jump Component
```

```

J = np.where(np.random.poisson(lambda_i * dt, n_stocks) > 0,
             np.random.normal(mu_j, sigma_j, n_stocks), 0)

# Update stock prices
prices[t, :] = prices[t-1, :] * np.exp((mu - 0.5 * sigma**2) * dt + sigma * ↵
                                         dW + J)

#Portfolio
portfolio_value = np.sum(prices, axis=1) / n_stocks

# PLOT STOCK
plt.figure(figsize=(10, 6))
for i in range(n_stocks):
    plt.plot(prices[:, i], label=f'Stock {i+1}')
plt.xlabel('Time (days)')
plt.ylabel('Price')
plt.legend()
plt.show()

# PLOT PORTFOLIO
plt.figure(figsize=(10, 6))
plt.plot(time, portfolio_value, label='Portfolio Value', color='black')
plt.xlabel('Time (Years)')
plt.ylabel('Portfolio Value')
plt.legend()
plt.grid()
plt.show()

# plt.title('Simulated Stock Prices with Correlation & Poisson Jumps (MJD Model)')
# plt.title('Simulated Portfolio Prices with Correlation & Poisson Jumps (MJD Model)')

```

V Appendix

Python code for Figure 8.

```
import pandas as pd
import matplotlib.pyplot as plt

#Load dataset
file_path = "stockdata2.csv"
df = pd.read_csv(file_path)
df['Date'] = pd.to_datetime(df['Date'])

#data between 2007 and 2016
df_filtered = df[(df['Date'] >= '2007-01-01') & (df['Date'] <= '2016-12-31')]

#Stock names
stocks_to_plot = ['MSFT', 'IBM', 'SBUX', 'AAPL']

#PLOT
plt.figure(figsize=(12, 6))
for stock in stocks_to_plot:
    stock_data = df_filtered[df_filtered['stock'] == stock]
    plt.plot(stock_data['Date'], stock_data['value'], label=stock)

plt.xlabel('Year')
plt.ylabel('Stock Price')
plt.title('Stock Prices (2007-2016)')
plt.legend()
plt.grid(True)
plt.show()
```

VI Appendix

Python code for Figure 9, Figure 10, Figure 11, Figure 12, Figure 13, Figure 14, Figure 15, and Figure 16. Programmer, 2023

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import minimize
import seaborn as sns

# Load dataset
file_path = "stockdata2.csv"
df = pd.read_csv(file_path)
df['Date'] = pd.to_datetime(df['Date'])

# Only take data up to 2014
df = df[df['Date'] <= '2014-12-31']

stocks = ['MSFT', 'IBM', 'SBUX', 'AAPL']
params = {}

# Estimate mu and sigma for each stock
for stock_name in stocks:
    stock_data = df[df['stock'] == stock_name].copy()
    stock_data.sort_values(by='Date', inplace=True)
    stock_data['log_return'] = np.log(stock_data['value'] / stock_data['value'].shift(1))
    stock_data.dropna(inplace=True)

    def negative_log_likelihood(params):
        mu, sigma = params
        likelihoods = -((stock_data['log_return'] - mu) ** 2) / (2 * sigma**2) - np.log(sigma)
        return -np.sum(likelihoods)

    initial_guess = [0.01, 0.02] # Initial values for μ and σ
    result = minimize(negative_log_likelihood, initial_guess, method='L-BFGS-B', bounds=[(-1, 1), (1e-6, 1)])
    mu_est, sigma_est = result.x
    params[stock_name] = (mu_est, sigma_est, stock_data['value'].iloc[-1])

    print(f"{stock_name} -> Estimated μ: {mu_est:.4f}, Estimated σ: {sigma_est:.4f}")

# MONTE CARLO FUNCTION
def monte_carlo_gbm(S0, mu, sigma, T=365, dt=1, sims=10000):
```

```

timesteps = int(T / dt)
prices = np.zeros((timesteps, sims))
prices[0, :] = S0

# Generate Brownian motion increments
dW = np.random.normal(0, np.sqrt(dt), (timesteps, sims))

# GBM Simulation
for t in range(1, timesteps):
    dW = np.random.normal(0, np.sqrt(dt), sims)
    dS = mu * prices[t-1] * dt + sigma * prices[t-1] * dW
    prices[t] = prices[t-1] + dS

return prices / S0

# PLOT
for stock_name in stocks:
    S0 = params[stock_name][2]
    mu_est, sigma_est = params[stock_name][:2]

    np.random.seed(None)
    simulated_prices = monte_carlo_gbm(S0, mu_est, sigma_est)

    plt.figure(figsize=(8, 5))
    plt.plot(simulated_prices[:, :100], alpha=0.1, color='blue')
    plt.plot(np.mean(simulated_prices[:, :100], axis=1), color='red', u
    ↪label="Mean Price")
    plt.xlabel("Days")
    plt.ylabel("Normalized Price")
    plt.legend()
    plt.grid()
    plt.show()

# plt.title(f"Monte Carlo Simulations: {stock_name}\nμ={mu_est:.4f}, σ={sigma_est:
    ↪.4f}")
stocks = ['MSFT', 'IBM', 'SBUX', 'AAPL']

#PLOT HISTOGRAMS
for stock_name in stocks:
    S0 = params[stock_name][2]
    mu_est, sigma_est = params[stock_name][:2]

    np.random.seed(None)
    simulated_prices = monte_carlo_gbm(S0, mu_est, sigma_est)

    final_prices = simulated_prices[-1]

```

```

mean_price = np.mean(final_prices)
median_price = np.median(final_prices)
std_dev = np.std(final_prices)
q25, q75 = np.percentile(final_prices, [25, 75])

plt.figure(figsize=(8, 5))

# Plot with KDE curve
sns.histplot(final_prices, bins=50, kde=True, color='blue',  

→edgecolor='black', alpha=0.6, stat="density")
    plt.axvline(mean_price, color='red', linestyle='--', label=f"Mean:  

→{mean_price:.2f}")

text_x = plt.xlim()[1] * 0.9
text_y = plt.ylim()[1] * 0.7
plt.text(text_x, text_y,  

    f" $\mu$ : {mean_price:.2f}\n $\sigma$ : {std_dev:.2f}\nQ25: {q25:.2f}\nQ75: {q75:.  

→2f}",
    ha='right', color='black', fontsize=10,  

→bbox=dict(facecolor='white', alpha=0.7))
    plt.xlabel("Normalized Simulated Final Price")
    plt.ylabel("Frequency")
    plt.legend()
    plt.xlim(min(final_prices) * 0.95, max(final_prices) * 1.05)
    plt.show()

#plt.title(f"Monte Carlo Simulation Histogram: {stock_name}"")

```