**Financial Portfolio Optimization using Monte Carlo Simulation**

Veda Sahaja Bandi

University of Massachusetts Dartmouth

North Dartmouth, MA 02747

## **Abstract**

In volatile financial markets, efficient portfolio optimization is crucial for achieving the desired risk-return objectives. This study proposes a novel approach that combines parallel Monte Carlo simulations and Markowitz optimization to address the challenges of large portfolios and complex market dynamics. By leveraging the power of parallel computing, this approach significantly accelerates Monte Carlo simulations, enabling efficient exploration of the efficient frontier [5]. This leads to several advantages, including scalability for large portfolios, robustness in accounting for market uncertainties, employing historical data for accurate risk and return models, and parallelization for faster and more informed decisions [4].

**Keywords:** Portfolio optimization, parallel computing, Monte Carlo simulation, Markowitz optimization, risk-return analysis, investment decision-making

# **Introduction**

Achieving the desired risk-return objectives in portfolio optimization is a pressing challenge in today's volatile markets and intricate asset classes. This study delves into an innovative strategy that merges parallel Monte Carlo simulations and Markowitz optimization. This novel approach is effective in crafting resilient and efficient portfolio allocation strategies.

Markowitz's efficient frontier, a cornerstone concept introduced in 1952, embodies the ideal balance between expected return and risk within a set of assets. However, conventional optimization struggles with larger portfolios and intricate market dynamics. Monte Carlo simulation, a robust tool for simulating asset returns and assessing portfolio performance, faces computational constraints with expansive datasets [2]. This study introduces a pioneering method that merges parallel Monte Carlo simulations and Markowitz optimization to overcome these hurdles. Leveraging parallel computing, it accelerates Monte Carlo simulations, enabling efficient exploration of the frontier for sizable portfolios [5]. Integrated with Markowitz optimization, this approach precisely identifies optimal asset allocations and maximizes expected returns while adhering to specific risk tolerance levels.

# **Literature Review**

The concept of portfolio optimization has been extensively researched in academic and financial literature. Markowitz (1952) pioneered the field with his seminal work on modern portfolio theory, introducing the mean-variance optimization framework and the concept of the efficient frontier [6]. Numerous subsequent studies have expanded on this work by exploring various optimization techniques and risk-return models. Monte Carlo simulation has emerged as a powerful tool for simulating asset returns and assessing portfolio performance [3]. Virines and Trumper (2021) and PM World Journal (2021) highlight the effectiveness of parallel Monte Carlo simulations in addressing the computational challenges of large portfolios [5]. Additionally, research by Nomad (2023) and Kommadi (2023) emphasizes the importance of Monte Carlo simulations in generating robust portfolio allocations that account for market uncertainties [3,4].

This study builds upon the existing body of knowledge by combining parallel Monte Carlo simulations with Markowitz optimization for portfolio optimization. This approach leverages the advantages of both techniques: the computational efficiency of parallel computing and the theoretical foundation of Markowitz optimization. By integrating these methodologies, this study presents a novel framework for achieving efficient and robust portfolio allocation strategies.

# **Methodology**

This study investigates the application of parallel Monte Carlo simulations and Markowitz optimization to portfolio optimization. The methodology comprises the following steps.

## **3.1 Data Sources:**

Historical price data for a selected set of assets were obtained using the yfinance library in Python. The specific tickers analyzed were AAPL, GOOGL, MSFT, AMZN, TSLA, and META. The data span from January 1, 2002, to December 31, 2022, providing a comprehensive timeframe for capturing market trends and dynamics [1]. We performed 30,000 simulations on the selected data.

## **3.2 Monte Carlo Simulation**

Monte Carlo simulation is a statistical method used to generate random samples from a probability distribution. In the context of portfolio optimization, it is used to simulate the potential returns and risks of different portfolio allocations [4]. The basic steps are as follows:

* **Define the parameters:** These include the expected returns and variances of the individual assets, and the desired number of simulations.
* **Generate random portfolios:** For each simulation, we generate a random set of weights for the assets in the portfolio.
* **Calculate portfolio returns and risks:** Using random weights and historical data, calculate the expected return and risk of the portfolio.
* **Repeat:** Repeat steps 2 and 3 for the desired number of simulations.

This process allows us to generate many potential portfolio outcomes and visualize the relationship between risk and return on the efficient frontier. For each portfolio in the simulation:

## **3.3 Markowitz Optimization**

Markowitz optimization is a method for constructing optimal portfolios based on the expected returns and variances of individual assets, and the investor's risk tolerance [7]. The goal is to find a portfolio that maximizes the expected return for a given level of risk or minimizes the risk of a given level of the expected return.

* **Objective Function:**
* **Constraints:** Portfolio weights sum to 1, and individual asset weights range between 0 and 1.
* **Optimization:** This method employs the Sequential Least Squares Programming (SLSQP) to determine the optimal asset weights.

This model can be solved using various optimization techniques, such as quadratic programming.

## **3.4 Monte Carlo Simulation with Parallel Processing**

Traditional Monte Carlo simulations are computationally expensive, especially for large portfolios with many assets. To address this challenge, Parallel processing can be leveraged using the Joblib library from Python, significantly reducing the computational cost. This involves dividing the simulations into smaller tasks and running them simultaneously on multiple processors, enabling efficient exploration of the efficient frontier for large portfolios.

## **3.5 Speedup and Efficiency**

The speedup of parallelization was calculated as the ratio of the execution time of the sequential algorithm to that of the parallel algorithm.

The efficiency of parallelization is the speedup divided by the number of processors used.

This study applies these methodologies to construct diversified portfolios, emphasizing long-term success and risk-aware investment strategies. The code implementation in Python leverages libraries such as NumPy, SciPy, Matplotlib, and Joblib to execute simulations and analyses. This comprehensive approach combines the principles derived from various sources to provide an extensive framework for portfolio optimization research.

# **Results**

The study used financial data retrieved from Yahoo Finance for six prominent stocks - AAPL, GOOGL, MSFT, AMZN, TSLA, and META, spanning from January 1, 2002, to December 31, 2022 [1]. The results are as follows.

## **4.1 Monte Carlo Simulation Results**

The Monte Carlo simulation generated 30,000 random portfolio weights, which were then used to calculate the expected return and risk (standard deviation) of each portfolio. The results are visualized in a scatter plot, with volatility plotted on the x-axis, return on the y-axis, and the Sharpe Ratio as the color gradient in Figure 1.

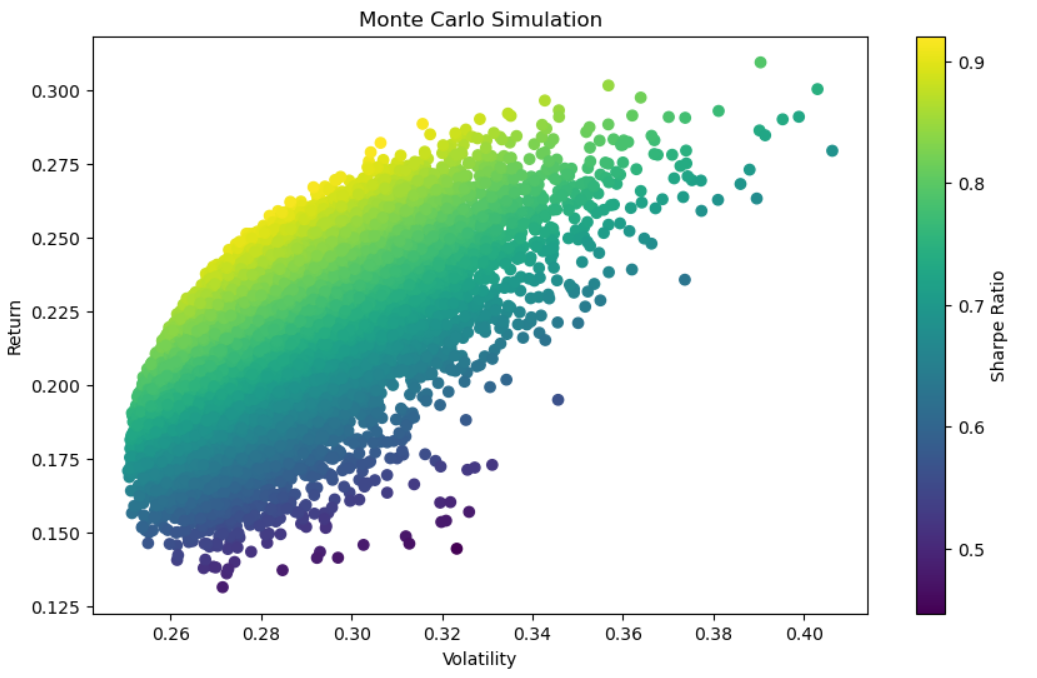


Figure 1 – Monte Carlo Simulation

The scatter plot reveals a clear trade-off between risk and return. Portfolios with higher expected returns tend to have higher standard deviations, indicating greater risk. Conversely, portfolios with lower expected returns tend to have lower standard deviations, indicating lower risk.

## **4.2 Individual Stock Performance**

A multi-line graph (Figure 2) tracks individual stock prices over time, revealing their historical trends and potential for diversification within a portfolio. This visual analysis helps assess individual volatilities and identify diversification benefits.

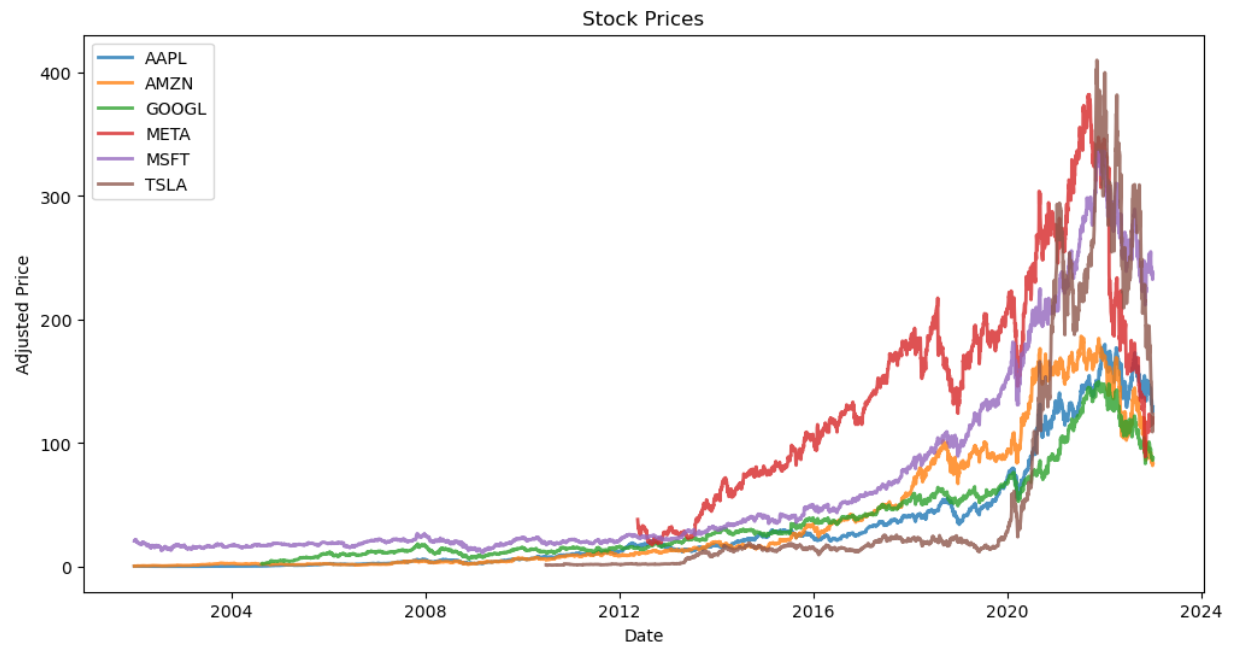


Figure 2 – Individual Stock Performance

## **4.3 Efficient Frontier**

The efficient frontier (Figure 3) is visualized using a scatter plot with three distinct elements:

* **Scatter points:** These represent simulated portfolios, colored by their Sharpe Ratio.
* **Individual Stock markers:** These represent the individual stocks, plotted with star symbols, and labeled with their respective tickers.
* **Markowitz Portfolio marker:** This represents the optimal portfolio calculated using Markowitz optimization, highlighted with a large red star.

where:

* *w* is the vector of portfolio weights
* is the covariance matrix of asset returns
* is the vector of expected asset returns
* is the minimum desired expected return

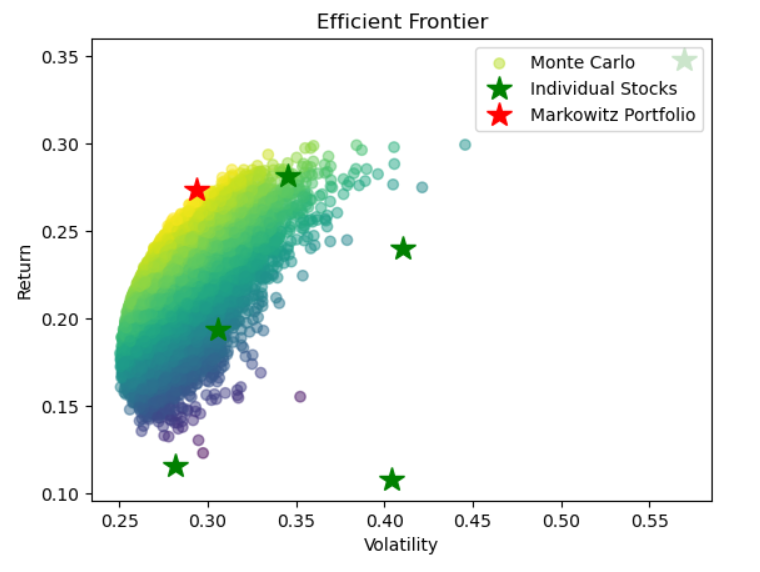


Figure 3 – Efficient Frontier

The efficient frontier illustrates that the optimal trade-off between risk and return is achievable through portfolio diversification. The Markowitz portfolio lies on the frontier, representing the portfolio with the highest Sharpe Ratio for a given level of risk.

## **4.4 Performance Comparison**

The table below presents the performance comparison between the sequential and parallel Monte Carlo simulations:

|  |  |
| --- | --- |
| **Method** | **Execution time (in seconds)** |
| Sequential | 6.45 |
| Parallelization | 3.24 |

The parallel implementation achieved a significant **speedup** of **1.99** times compared with the sequential approach, demonstrating the effectiveness of parallel processing in reducing the computational cost of Monte Carlo simulations. The **efficiency** of **0.25** indicates that the available processors were used effectively.

The results demonstrate the effectiveness of combining parallel Monte Carlo simulation and Markowitz optimization for efficient portfolio optimization. This approach enables the exploration of many potential portfolios and facilitates the identification of optimal allocations that balance risk and return based on individual investor preferences [4]. The use of parallel processing significantly reduces the computational burden, making this approach practical for large and complex portfolios.

# **Conclusion**

This study tackles the challenge of efficient asset allocation in volatile markets by marrying parallel Monte Carlo simulations with Markowitz optimization. The parallelization boost lets us dive deep into numerous portfolio scenarios, simultaneously assessing risk and return across various asset combinations [5]. This significantly speeds up analysis, especially for large portfolios or extensive historical data. Markowitz optimization, paired with this parallel power, equips investors with a robust method for building diversified portfolios that maximize returns within their risk tolerance, or vice versa. However, we acknowledge the inherent assumptions and limitations, like sensitivity to input parameters and the reliance on historical data [2].

Despite these limitations, the synergy between parallel computing and portfolio optimization opens up exciting avenues for more sophisticated analysis. This empowers investors to make informed decisions while considering various risk-return trade-offs [3]. Future research could focus on refining simulations, exploring alternative optimization models, or integrating machine learning to enhance portfolio construction and risk management in ever-evolving markets. The future of asset allocation looks bright, fueled by the power of parallel processing and data-driven insights.

# **Appendix**

import yfinance as yf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import minimize

import joblib

from joblib import Parallel, delayed

import time

*# Function to fetch stock data from Yahoo Finance*

def get\_stock\_data(tickers, start\_date, end\_date):

data = yf.download(tickers, start=start\_date, end=end\_date)['Adj Close']

return data

*# Function to calculate returns and volatility*

def calculate\_returns\_volatility(stocks):

returns = np.log(stocks / stocks.shift(1))

mean\_daily\_returns = returns.mean()

cov\_matrix = returns.cov()

return returns, mean\_daily\_returns, cov\_matrix

tickers = ['AAPL', 'GOOGL', 'MSFT', 'AMZN', 'TSLA', 'META']

start\_date = '2002-01-01'

end\_date = '2022-12-31'

num\_portfolios = 30000

*# Fetch stock data*

stock\_data = get\_stock\_data(tickers, start\_date, end\_date)

returns, mean\_daily\_returns, cov\_matrix = calculate\_returns\_volatility(stock\_data)

*# Function to perform Monte Carlo simulation*

def monte\_carlo\_simulation(returns, mean\_daily\_returns, cov\_matrix, num\_portfolios):

results = np.zeros((3, num\_portfolios))

weights\_record = np.zeros((len(returns.columns), num\_portfolios))

for i in range(num\_portfolios):

weights = np.random.random(len(returns.columns))

weights /= np.sum(weights)

portfolio\_return = np.sum(mean\_daily\_returns \* weights) \* 252

portfolio\_std\_dev = np.sqrt(np.dot(weights.T, np.dot(cov\_matrix, weights))) \* np.sqrt(252)

results[0, i] = portfolio\_return

results[1, i] = portfolio\_std\_dev

results[2, i] = results[0, i] / results[1, i]

weights\_record[:, i] = weights

return results, weights\_record

*# Perform Monte Carlo simulation - Sequential*

start\_time = time.time()

mc\_results, mc\_weights = monte\_carlo\_simulation(returns, mean\_daily\_returns, cov\_matrix, num\_portfolios)

end\_time = time.time()

serial\_time = end\_time - start\_time

*# Plotting Monte Carlo simulation results*

plt.figure(figsize=(10, 6))

plt.scatter(mc\_results[1, :], mc\_results[0, :], c=mc\_results[2, :], marker='o', cmap='viridis')

plt.colorbar(label='Sharpe Ratio')

plt.title('Monte Carlo Simulation')

plt.xlabel('Volatility')

plt.ylabel('Return')

plt.show()

def simulate(i, returns, mean\_daily\_returns, cov\_matrix):

weights = np.random.random(len(returns.columns))

weights /= np.sum(weights)

portfolio\_return = np.sum(mean\_daily\_returns \* weights) \* 252

portfolio\_std\_dev = np.sqrt(np.dot(weights.T, np.dot(cov\_matrix, weights))) \* np.sqrt(252)

return portfolio\_return, portfolio\_std\_dev, i

*# Perform Monte Carlo Simulation - Parallelization*

n\_jobs = joblib.cpu\_count()

start\_time = time.time()

results\_parallel = Parallel(n\_jobs=n\_jobs)(delayed(simulate)(i, returns, mean\_daily\_returns, cov\_matrix) for i in range(num\_portfolios))

results\_parallel = np.array(results\_parallel).T

mc\_results\_joblib = results\_parallel[:2] # Extract return and volatility

mc\_weights\_joblib = np.zeros((results\_parallel.shape[1], num\_portfolios))

end\_time = time.time()

parallel\_time = end\_time - start\_time

*# Function to calculate optimal portfolio using Markowitz method*

def calculate\_markowitz\_portfolio(returns):

mean\_daily\_returns = returns.mean()

cov\_matrix = returns.cov()

def objective(weights):

weights = np.array(weights)

portfolio\_return = np.sum(mean\_daily\_returns \* weights) \* 252

portfolio\_std\_dev = np.sqrt(np.dot(weights.T, np.dot(cov\_matrix, weights))) \* np.sqrt(252)

return -portfolio\_return / portfolio\_std\_dev

constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

boundaries = tuple((0, 1) for \_ in range(len(returns.columns)))

init\_guess = np.array(len(returns.columns) \* [1. / len(returns.columns)])

optimal\_weights = minimize(objective, init\_guess, method='SLSQP', bounds=boundaries, constraints=constraints)

return optimal\_weights

*# Calculate Markowitz portfolio*

optimal\_weights = calculate\_markowitz\_portfolio(returns)

print("Optimal Allocation (Markowitz):")

print(optimal\_weights.x)

*# Function to plot individual stocks*

def plot\_individual\_stocks(returns):

plt.figure(figsize=(12, 6))

for c in returns.columns.values:

plt.plot(returns.index, returns[c], lw=2, alpha=0.8, label=c)

plt.legend(loc='upper left', fontsize=10)

plt.title('Stock Prices')

plt.xlabel('Date')

plt.ylabel('Adjusted Price')

plt.show()

plot\_individual\_stocks(stock\_data)

*# Function to plot efficient frontier with Markowitz portfolio*

def plot\_efficient\_frontier(returns, mean\_daily\_returns, cov\_matrix, num\_portfolios, optimal\_weights):

results, \_ = monte\_carlo\_simulation(returns, mean\_daily\_returns, cov\_matrix, num\_portfolios)

plt.scatter(results[1, :], results[0, :], c=results[2, :], marker='o', alpha=0.5, cmap='viridis', label='Monte Carlo')

plt.plot(np.sqrt(np.diagonal(cov\_matrix)) \* np.sqrt(252), mean\_daily\_returns \* 252, 'g\*', markersize=15, label='Individual Stocks')

plt.plot(np.sqrt(np.dot(optimal\_weights.x.T, np.dot(cov\_matrix, optimal\_weights.x))) \* np.sqrt(252), np.dot(optimal\_weights.x.T, mean\_daily\_returns) \* 252, 'r\*', markersize=15, label='Markowitz Portfolio')

plt.legend(loc='upper right')

plt.title('Efficient Frontier')

plt.xlabel('Volatility')

plt.ylabel('Return')

plt.show()

plot\_efficient\_frontier(returns, mean\_daily\_returns, cov\_matrix, num\_portfolios, optimal\_weights)

*# Time taken - Sequential*

print(f"Time taken for Monte Carlo Simulation - Sequential: {serial\_time:.2f}s")

*# Time taken - parallelization*

print(f"Time taken for Monte Carlo Simulation - Parallelization: {parallel\_time:.2f}s")

*# Calculate Speedup*

speedup = serial\_time / parallel\_time

*# Calculate Efficiency*

efficiency = speedup / n\_jobs

print(f"Speedup: {speedup:.2f}")

print(f"Efficiency: {efficiency:.2f}")

# **References**

[1] Yahoo! Finance. <https://finance.yahoo.com/>

[2] Shaw, W (2010). Monte Carlo Portfolio Optimization for General Investor Risk-Return Objectives and Arbitrary Return Distributions: A Solution for Long-Only Portfolios

[3] Nomad. (2023). Portfolio Optimization with Monte Carlo Simulation: Maximizing Returns and Minimizing Risk. https://medium.com/coinmonks/portfolio-optimization-with-monte-carlo-simulation-maximizing-returns-and-minimizing-risk-2f106feaa44d

[4] Kommadi, B. (2019). Massively Parallel Monte Carlo Simulation. https://medium.com/@thisismetis/what-is-a-monte-carlo-simulation-part-1-32c01b48cc49

[5] Virines, F., & Trumper, M. (2021). Improving Monte Carlo Simulations for Portfolio Risk Analysis.

[6] Dospatliev, L., & Ivanova, I. (2018). APPLICATION OF MARKOWITZ PORTFOLIO OPTIMIZATION ON THE BULGARIAN STOCK EXCHANGE. International Research Journal of Finance and Economics, 32(1), 131-137.

[7] Torris, L. (2023). Portfolio Optimization: The Markowitz Mean-Variance Model. https://medium.com/latinxinai/portfolio-optimization-the-markowitz-mean-variance-model-c07a80056b8a

[8] Kulali, I. (2016). Portfolio Optimization Analysis with Markowitz Quadratic Mean-Variance Model

[9] Quantpedia. (n.d.). Case Study - Markowitz Portfolio Optimization. https://quantpedia.com/case-study-markowitz-portfolio-optimization/

[10] Zhu, L., & Li, J. (2018). Portfolio Optimization with Transaction Costs and Estimation Risk. Journal of Banking & Finance, 94, 148-166.