REPORT ON THE PYTHON IMPLEMENTATION OF HARRY MARKOWITZ'S PORTFOLIO THEORY

FEBUARY 21,2025

Nino Aubert – www.linkedin.com/in/ninoaubert
Bachelor's student in Economics and Management at SKEMA Business School

Table of Contents

Acknowledgments

First of all, I would like to express my gratitude to Professor Ram Janakiraman, PhD, whose course "Introduction to AI & Machine Learning for Business" and group project "Machine Learning Applied to Finance" reignited my motivation for developing programming projects in finance. His pedagogical approach and ability to make artificial intelligence accessible to real-world challenges inspired me to further develop my skills in this field.

I also extend my sincere appreciation to Professor Majed Al-Ghandour, PhD, PE, for taking the time to review my report and provide valuable feedback. His insights have greatly contributed to refining my work.

Additionally, I am grateful to Madam Nathalie Aubert for her meticulous language corrections, ensuring clarity and precision in my writing.

Finally, I want to emphasize the importance of continuous learning and intellectual curiosity in shaping this project. This work stems from a deep passion for programming and data analysis.

Nino Aubert,

February 21, 2025

About Me and My Project

I am a French student in my second year of a bachelor's degree in economics and management at SKEMA Business School, currently on exchange at the Raleigh campus in the United States. My academic journey has allowed me to explore various fields of management, finance, and economics while developing a strong interest in investment finance and portfolio management.

Throughout my studies (high school) and personal projects, I have acquired programming skills, particularly in Python. During this project, I learned to use libraries such as NumPy, Pandas, Matplotlib, and SciPy, which are essential for data analysis and financial modeling.

I am now striving to develop the ability to structure optimization algorithms and automate analytical tasks. With these skills, I aim to apply advanced financial concepts to real datasets, which is a major asset for portfolio analysis and investment decision-making.

This portfolio optimization project was born from my desire to practically apply the financial theories I acquired during my research. I wanted to go beyond theoretical concepts by implementing Harry Markowitz's Modern Portfolio Theory using a powerful programming tool: Python.

By leveraging the NumPy, Pandas, and scipy.optimize libraries, this project allows me to explore the following concepts:

- Constructing a diversified asset portfolio
- Calculating returns and risks using the covariance matrix
- Optimizing asset weights to maximize the Sharpe ratio
- Simulating portfolios to visualize the efficient frontier

This project represents a first step towards applying advanced financial models and a way to strengthen my programming and data analysis skills. It enables me to combine my knowledge in economics and management with quantitative tools, an essential asset for my academic and professional future.

Who is Harry Markowitz?

Harry Markowitz and Portfolio Optimization

Harry Markowitz is an American economist born in 1927, famous for developing Modern Portfolio Theory (MPT) in the 1950s. His groundbreaking work, which earned him the Nobel Prize in Economics in 1990, is based on the idea of diversification and optimizing the risk-return trade-off in an investment portfolio. In Python, Harry Markowitz's portfolio theory, also known as Modern Portfolio Theory (MPT), is implemented through mathematical calculations to optimize asset allocation within a portfolio by maximizing expected returns while minimizing risk, typically using libraries like NumPy and SciPy to perform the necessary calculations for mean-variance optimization, a core concept of the theory.

His Contribution to Finance

Markowitz introduced a mathematical framework to construct optimal portfolios based on expected returns and risk measured by variance (or standard deviation) of returns. His approach is based on several key concepts:

- Diversification: Investing in multiple assets helps reduce overall portfolio risk without necessarily decreasing expected return.
- Risk and return: He established a relationship between a portfolio's expected return and its risk, measured by volatility.
- Efficient frontier: He demonstrated that for a given level of risk, there is an optimal asset combination offering the highest possible return. This curve is known as the "efficient frontier."
- Sharpe Ratio: Although Markowitz did not invent this ratio, his work contributed to the idea that excess return (compared to the risk-free rate) should be evaluated relative to the risk taken.

Link to My Project

My code implements several principles of Markowitz's theory:

- Calculating expected returns and the covariance matrix: These elements help assess portfolio return and risk.
- Optimizing asset weights: The scipy.optimize minimize function is used to find the optimal combination of assets that minimizes risk while maximizing risk-adjusted return (Sharpe ratio).
- Simulating random portfolios: By generating thousands of portfolios, you can visualize the efficient frontier and compare the optimal portfolio to other possible combinations.

Code

main.py

```python
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
from scipy.optimize import minimize
import time

# 1. Data Retrieval
assets = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA']  # List of stocks
start_date = '2020-01-01'
end_date = '2024-01-01'

# Attempt to download data up to 3 times in case of failure
for i in range(3):
    try:
        data = yf.download(assets, start=start_date, end=end_date)['Adj Close']
        break
    except Exception as e:
        print(f"Attempt {i+1} failed: {e}")
        time.sleep(5)  # Pause before retrying

# Calculate daily returns and drop missing values
returns = data.pct_change(fill_method=None).dropna()

# 2. Financial Metrics Calculation
mean_returns = returns.mean() * 252  # Annualized return
cov_matrix = returns.cov() * 252  # Annualized covariance matrix

# 3. Minimization Function (Negative Sharpe Ratio)
def portfolio_performance(weights, mean_returns, cov_matrix, risk_free_rate=0.02):
    """
    Calculates the portfolio performance in terms of return, volatility, and Sharpe ratio.
    
    :param weights: Portfolio allocation weights
    :param mean_returns: Expected annual returns of assets
    :param cov_matrix: Annualized covariance matrix of assets
    :param risk_free_rate: Risk-free rate (default 2%)
    :return: Negative Sharpe ratio (for minimization)
    """
    returns = np.dot(weights, mean_returns)  # Expected return
    volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))  # Portfolio volatility
    sharpe_ratio = (returns - risk_free_rate) / volatility  # Sharpe ratio
    return -sharpe_ratio  # Negative for minimization

# Constraints and Bounds
num_assets = len(assets)
constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})  # Sum of weights must be 1
bounds = tuple((0, 1) for _ in range(num_assets))  # Each weight between 0 and 1
initial_guess = num_assets * [1. / num_assets]  # Equal allocation initial guess

# Optimization Process
optimal = minimize(portfolio_performance, initial_guess, args=(mean_returns, cov_matrix), method='SLSQP', bounds=bounds, constraints=constraints)
optimal_weights = optimal.x  # Extract optimal weights

# 4. Display Results
plt.figure(figsize=(10, 6))
num_portfolios = 10000  # Number of simulated portfolios
results = np.zeros((3, num_portfolios))  # Store return, volatility, and Sharpe ratio

# Monte Carlo Simulation to generate random portfolios
for i in range(num_portfolios):
    weights = np.random.dirichlet(np.ones(num_assets))  # Generate random weights
    portfolio_return = np.dot(weights, mean_returns)  # Compute return
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))  # Compute volatility
    results[0, i] = portfolio_return
    results[1, i] = portfolio_volatility
    results[2, i] = (portfolio_return - 0.02) / portfolio_volatility  # Compute Sharpe ratio

# Plot the Efficient Frontier
plt.gcf().canvas.manager.set_window_title("Efficient Frontier - Portfolio Optimization")
plt.scatter(results[1], results[0], c=results[2], cmap='viridis', marker='o', alpha=0.3)
plt.colorbar(label='Sharpe Ratio')  # Color bar for Sharpe Ratio
plt.scatter(np.sqrt(np.dot(optimal_weights.T, np.dot(cov_matrix, optimal_weights))), np.dot(optimal_weights, mean_returns), c='red', marker='*', s=200, label='Optimal Portfolio')  # Mark the optimal portfolio
plt.xlabel('Volatility')
plt.ylabel('Expected Return')
plt.title('Efficient Frontier of Portfolios')
plt.legend()
plt.show()

# Display optimal weights as a DataFrame
df_weights = pd.DataFrame({'Stocks': assets, 'Optimal Weights': optimal_weights})
print("\nOptimal Portfolio Weights:")
print(df_weights)
```

Download the code:

Or:

https://github.com/NinoAubert/report-on-the-python-implementation-of-harry-markowitz-s-portfolio-theory.git

Prerequisites

Python:

- Install Python: https://learn.microsoft.com/en-us/windows/python/beginners

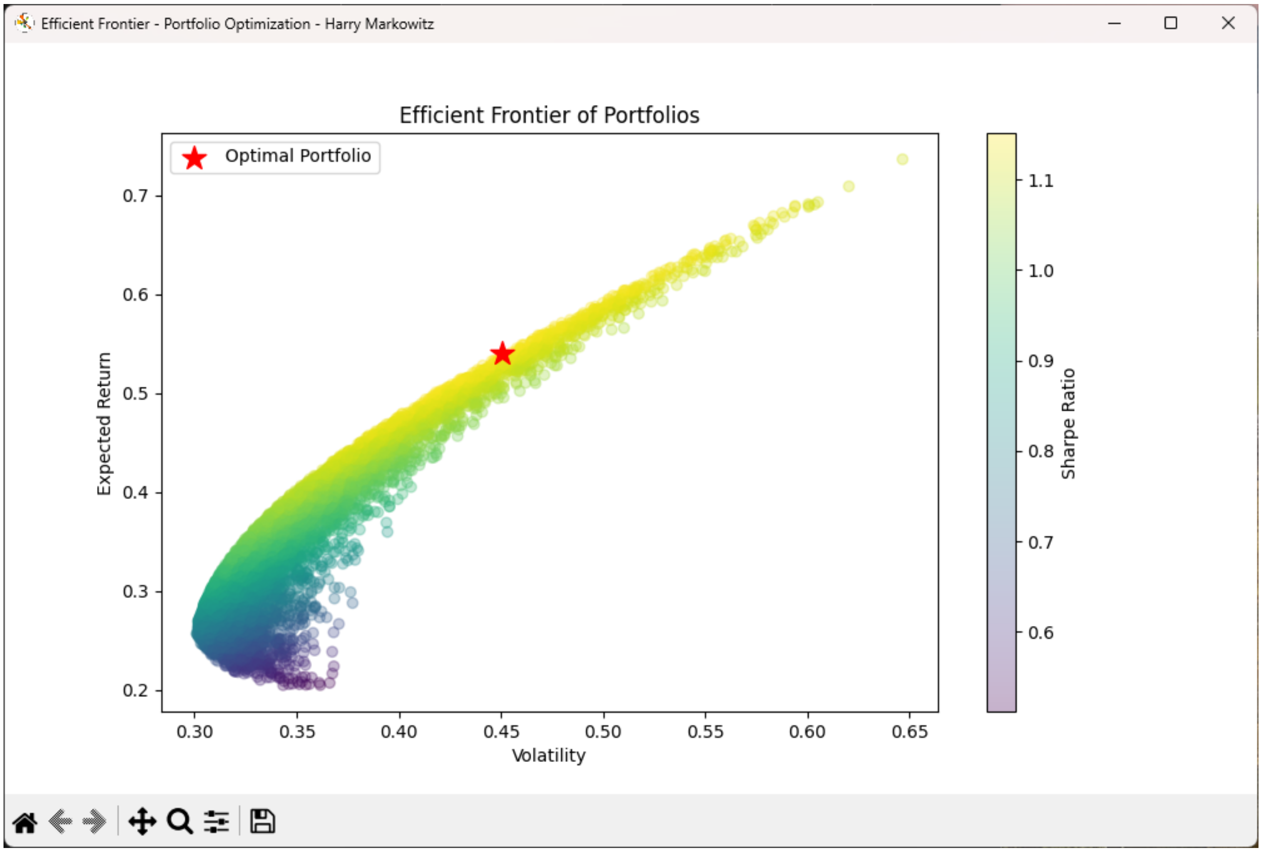
Python Libraries:

- Install NumPy: https://numpy.org/install/
- Install Pandas: https://pandas.pydata.org/
- Install yfinance: https://pypi.org/project/yfinance/
- Install Matplotlib: https://matplotlib.org/stable/install/index.html
- Install Spicy: https://docs.zeek.org/projects/spicy/en/latest/installation.html

Steps:

- Data Collection: Gather historical price data for the assets in the portfolio.
- Returns & Covariance: Compute expected returns and the covariance matrix to assess asset movement.
- Optimization: Use a solver (e.g., scipy.optimize.minimize) to maximize returns while constraining risk.
- Efficient Frontier: Plot the efficient frontier to visualize optimal portfolios balancing risk and return.

Final Result




```
PS C:\Users[redacted]> & C:/Users/[redacted]/Python/Python310/python.exe c:/Users[redacted]/main.py
[*********************100%***********************]  5 of 5 completed

Optimal Portfolio Weights:
  Stocks  Optimal Weights
0   AAPL     2.695688e-01
1   MSFT     2.114194e-17
2  GOOGL     0.000000e+00
3   AMZN     2.116647e-01
4   TSLA     5.187666e-01
```

Mathematical Reasoning and Application to Python

Data Retrieval and Return Calculation

```python
# 1. Data Retrieval
assets = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA']  # List of stocks
start_date = '2020-01-01'
end_date = '2024-01-01'

# Attempt to download data up to 3 times in case of failure
for i in range(3):
    try:
        data = yf.download(assets, start=start_date, end=end_date)['Adj Close']
        break
    except Exception as e:
        print(f"Attempt {i+1} failed: {e}")
        time.sleep(5)  # Pause before retrying

# Calculate daily returns and drop missing values
returns = data.pct_change(fill_method=None).dropna()
```

We retrieve the adjusted closing prices (Adj Close) of the stocks and calculate the daily returns.

Daily Return:

The return of a financial asset between two days is given by the formula:

$$r_{i,t} = \frac{P_{i,t}}{P_{i,t-1}} - 1 = \frac{P_{i,t} - P_{i,t-1}}{P_{i,t-1}}$$

where:

- $r_{i,t}$ is the return of asset i on day t,
- $P_{i,t}$ is the adjusted closing price on day t,
- $P_{i,t-1}$ is the adjusted closing price on day $t-1$.

In the code, we use:

```python
# Calculate daily returns and drop missing values
returns = data.pct_change(fill_method=None).dropna()
```

to calculate this automatically.

- data.pct_change() automatically calculates $R_{i,t}$ for each day and each asset.
- .dropna() removes missing values that appear on the first day (where no return can be calculated).

Application:

Date \ Stock	AAPL	AMZN	GOOGL	MSFT	TSLA
2020-01-02 00:00:00+00:00	72.716080	94.900497	68.186821	153.630707	28.684000
2020-01-03 00:00:00+00:00	72.009125	93.748497	67.830101	151.717712	29.534000
2020-01-06 00:00:00+00:00	72.582893	95.143997	69.638054	152.109848	30.102667

Date \ Action	AAPL	AMZN	GOOGL	MSFT	TSLA
2020-01-02 00:00:00+00:00	N/A	N/A	N/A	N/A	N/A
2020-01-03 00:00:00+00:00	-0.009722	-0.012139	-0.005232	-0.012452	0.029633
2020-01-06 00:00:00+00:00	0.007968	0.014886	0.026654	0.002585	0.019255
2020-01-07 00:00:00+00:00	-0.004703	0.002092	-0.001932	-0.009118	0.038801

We can observe that at the close of 2020-01-**01** (the close is the first second of the following day), the price of AAPL was 72.716080. The closing price of AAPL on 2020-01-**02** was 72.009125. Therefore:

$$r_{i,t} = \frac{P_{i,t}}{P_{i,t-1}} \approx \frac{72.009125}{72.716080} \approx -0.009722$$

We obtain the return of the financial asset AAPL from 2020-01-01 to 2020-01-02, which is approximately -0.9722%.

Note: Since the return of the financial asset cannot be calculated on the first day (N/A), we use .dropna() to remove these values.

Calculation of Financial Metrics

The goal is to obtain an estimate of the average annual return for each asset.

```
# 2. Financial Metrics Calculation
mean_returns = returns.mean() * 252  # Annualized return
cov_matrix = returns.cov() * 252  # Annualized covariance matrix
```

Annualized Average Return:

We calculate the average of the daily returns:

$$E(R_i) = \frac{1}{T}\sum_{t=1}^{T} r_{i,t}$$

where:

- $E(R_i)$ is the expected return of asset i. It represents the average of the returns over the observed periods,
- T is the total number of days (the period),
- $r_{i,t}$ is the daily return of financial asset i on day t.

Then, we annualize by multiplying by 252 (the average number of trading days in a year):

$$E(R_i^{annuel}) = E(R_i) \times 252$$

This corresponds to `mean_returns` in the code:

```
mean_returns = returns.mean() * 252  # Annualized return
```

- `returns.mean()` gives the average of the daily returns.
- We multiply it by 252 to obtain an estimate of the annual return.

Application:

Step / Stock	$\sum_{t=1}^{T} r_{i,t}$	$E(R_i) = \frac{1}{T}\sum_{t=1}^{T} r_{i,t}$	$E(R_i) \times 252$
AAPL	1.192500	0.001187	0.299015
AMZN	0.753563	0.000750	0.188953
GOOGL	0.938193	0.000934	0.235248
MSFT	1.100100	0.001095	0.275846
TSLA	3.085092	0.003070	0.773575

Here, it is important to note that in this application, we observe these assets over a period of 4 years, so the number of days T corresponds to 4 times the number of trading days.

Annualized Covariance Matrix:

Covariance is a statistical measure that indicates the extent to which two random variables (here, the returns of assets R_i and R_j) vary together. If the covariance is positive, it means that when the return of one asset increases, the return of the other tends to increase as well. If it is negative, when the return of one asset increases, the other tends to decrease. The covariance between two assets i and j is given by:

$$Cov(R_i, R_j) = \frac{1}{T-1}\sum_{t=1}^{T}(r_{i,t} - E(R_i))(r_{j,t} - E(R_j))$$

where:

- $Cov(R_i, R_j)$ is the covariance between the returns of assets i and j,
- $E(R_i)$ et $E(R_j)$ are the expected returns of assets i and j. hey represent the average of the returns of each asset over the observed periods,
- T is the total number of days,
- $r_{i,t}$ and $r_{i,t}$ are the daily returns of assets i and j.

Variance between two assets i and j is given by:

$$Var(R_i) = \sigma^2 = \frac{1}{T-1}\sum_{t=1}^{T}(r_{i,t} - E(R_i))^2$$

Example: Calculating Covariance and Variance

Imagine we observe the returns of two assets (Asset A and Asset B) over a period of 5 days. The daily returns are given below:

Day	Return of Asset A ($r_{A,t}$)	Return of Asset B ($r_{B,t}$)
1	0.05	0.02
2	0.03	0.01
3	0.04	0.03
4	0.02	0.04
5	0.01	0.02

Step 1: Calculate the average (expected) returns:

We first calculate the average return of each asset:

- For Asset A: $E(R_A) = \frac{0.05+0.03+0.04+0.02+0.01}{5} = \frac{0.15}{5} = 0.03$
- For Asset B: $E(R_B) = \frac{0.02+0.01+0.03+0.04+0.02}{5} = \frac{0.12}{5} = 0.024$

Step 2: Calculate the deviations from the average returns:

Next, we calculate the deviations between the daily returns and the average returns for each day, for both assets.

Day	$r_{A,t} - E(R_A)$	$r_{B,t} - E(R_B)$
1	0.05−0.03=0.02	0.02−0.024=−0.004
2	0.03−0.03=0	0.01−0.024=−0.014
3	0.04−0.03=0.01	0.03−0.024=0.006
4	0.02−0.03=−0.01	0.04−0.024=0.016
5	0.01−0.03=−0.02	0.02−0.024=−0.004

Step 3: Calculate the products of the deviations:

For each day, we multiply the deviations calculated for Asset A and Asset B.

Day	$\left(r_{A,t} - E(R_A)\right) * (r_{A,t} - E(R_A))$
1	0.02×−0.004=−0.00008
2	0×−0.014=0
3	0.01×0.006=0.00006
4	−0.01×0.016=−0.00016
5	−0.02×−0.004=0.00008

Step 4: Calculate the sum of the products of the deviations:

We now add up the calculated products:

$$\sum_{t=1}^{5}(\left(r_{A,t} - E(R_A)\right) * (r_{A,t} - E(R_A))) = -0.00008 + 0 + 0.00006 - 0.00016 + 0.00008$$
$$= -0.0001$$

Step 5: Calculate the covariance:

Finally, we use the covariance formula:

$$Cov(R_A, R_B) = \frac{1}{T-1}\sum_{t=1}^{T}(r_{A,t} - E(R_A))(r_{B,t} - E(R_B))$$

Here, T = 5 days, so $T - 1 = 4$.

$$Cov(R_A, R_B) = \frac{-0.0001}{4} = -0.000025$$

Conclusion:

The covariance between the returns of Asset A and Asset B is negative (−0.000025). This means that, overall, the returns of the two assets tend to vary in opposite directions: when Asset A increases, Asset B tends to decrease, and vice versa.

This example illustrates how covariance is calculated and how it helps to understand the relationship between the variations in the returns of two assets in a portfolio.

Note: We won't repeat it here, but if we want to calculate the variance of A, for example, we use this formula by focusing solely on financial asset A:

$$Var(R_A) = \sigma^2 = \frac{1}{T-1}\sum_{t=1}^{T}(r_{A,t} - E(R_A))^2$$

A covariance matrix is a square table that presents the variances of the assets along the main diagonal and the covariances between each pair of assets in the other cells. If we have n assets in a portfolio, the covariance matrix Σ will have the form $n * n$, with each element representing either a variance (for an individual asset) or a covariance (between two assets).

Covariance matrix for n assets:

Let n assets with $R_{i,} R_{j,} \dots, R_{n,}$ representing the returns of each asset. The covariance matrix Σ has the following structure:

$$\Sigma = \begin{pmatrix} Var(R_i) & Cov(R_i, R_j) & \cdots & Cov(R_i, R_n) \\ Cov(R_j, R_i) & Var(R_j) & \cdots & Cov(R_j, R_n) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(R_n, R_i) & Cov(R_n, R_j) & \cdots & Var(R_n) \end{pmatrix}$$

We annualize the covariance matrix by multiplying by 252 (the average number of trading days in a year):

$$\Sigma annuel = \Sigma quotidien \times 252$$

In the code, the covariance matrix is calculated as follows:

```
cov_matrix = returns.cov() * 252  # Annualized covariance matrix
```

- returns.cov() calculates the covariance matrix of the daily returns.
- We multiply by 252 to annualize the covariance.

Portfolio Optimization (Maximization of the Sharpe Ratio)

Once we have the average returns and the covariance matrix, we can evaluate the performance of a given portfolio based on the weights of the assets.

Expected Portfolio Return:

The return of a portfolio $E(R_p)$ is the sum of the returns of the assets $E(R_i)$ weighted by their weights w_i:

$$E(R_p) = \sum_{i=1}^{n} w_i E(R_i)$$

In the code, the expected portfolio return is calculated as:

```
    returns = np.dot(weights, mean_returns)  # Expected return
```

where:

- weights is a vector containing the allocations of each asset.

Portfolio Volatility (Total Risk):

The total risk of a portfolio is given by the variance of the portfolio, which is calculated as follows:

$$\sigma_p{}^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j Cov(R_i, R_j)$$

où :

- σ_p^2 is the variance of the portfolio.

We then take the square root to obtain the portfolio volatility:

$$\sigma_p = \sqrt{\sigma_p{}^2} = \sqrt{w^T \Sigma w}$$

where:

- Σ is the covariance matrix.

In the code, the portfolio volatility is calculated as:

```
    volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))  # Portfolio volatility
```

where:

- np.dot(weights.T, np.dot(cov_matrix, weights)) applies the formula $w^T \Sigma w$.
- np.sqrt() takes the square root to obtain σ_p.

Ratio de Sharpe :

Le Ratio de Sharpe mesure la rentabilité ajustée au risque:

$$S = \frac{E(R_p) - R_f}{\sigma_p}$$

where:

- S is the Sharpe ratio,
- $E(R_p)$ is the expected return of the portfolio,
- R_f is the risk-free rate (set at 2% in the code),
  - The risk-free rate is a hypothetical rate with no risk, whose return is certain over a certain period, such as the yield on a one-year or two-year Treasury.
- σ_p est la volatilité du portefeuille.

In the code, the Sharpe ratio is calculated as:

```
    sharpe_ratio = (returns - risk_free_rate) / volatility  # Sharpe ratio
```

We minimize its opposite to maximize S:

```
    return -sharpe_ratio  # Negative for minimization
```

Optimization Constraints

We impose the following constraints:

a) Sum of weights = 1 (the portfolio is fully invested):

$$\sum_{i=1}^{n} w_i = 1$$

In the code, this constraint is implemented as:

```
constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})  # Sum of weights must be 1
```

- {'type': 'eq'}: This means that this constraint is an equality constraint. In optimization, constraints can be of type 'eq' (equality) or 'ineq' (inequality). Here, the type 'eq' imposes that the given function must be equal to zero.
- 'fun': lambda weights: np.sum(weights) – 1 :
    - lambda weights: is a lambda function that takes weights (the portfolio weights or decision variables) as an argument.
    - np.sum(weights) calculates the sum of the weights.
    - np.sum(weights) - 1 indicates that the sum of the weights must be equal to 1. This is a common constraint in portfolio optimization problems to ensure that the proportions of assets in a portfolio total 100%.

b) The weights are between 0 and 1 (no short selling):

```
bounds = tuple((0, 1) for _ in range(num_assets))  # Each weight between 0 and 1
```

- tuple((0, 1) for _ in range(num_assets)) :
    - (0, 1) : This means that each weight must be between 0 and 1. A weight of 0 indicates that the asset is not included in the portfolio, while a weight of 1 means that the entire investment is in that asset.
    - for _ in range(num_assets) : This is a loop generating a pair of bounds (0, 1) for each asset in the portfolio. The number of assets is given by the variable num_assets.

Harry Markowitz's Optimization Problem

Formulation of the Optimization Problem:

The goal of Markowitz's optimization is to find the optimal combination of assets that maximizes the Sharpe ratio:

$$S = \frac{E(R_p) - R_f}{\sigma_p}$$

where:

- $E(R_p) = \sum_{i=1}^{n} w_i E(R_i)$ is the expected return of the portfolio,
- $\sigma_p = \sqrt{w^T \Sigma w}$ is the portfolio volatility,
- R_f is the risk-free rate.

Mathematical formulation of the problem:

$$\max_{w} S = \frac{\sum_{i=1}^{n} w_i E(R_i) - R_f}{\sqrt{w^T \Sigma w}}$$

subject to the constraint:

$$\sum_{i=1}^{n} w_i = 1$$

and the bounds:

$$0 \leq w_i \leq 1, \qquad \forall i$$

Solution with the Lagrange Multipliers Method:

To solve this problem, we use the Lagrange multipliers method.

We define the Lagrange function:

$$\mathcal{L}(w,\lambda) = \frac{\sum_{i=1}^{n} w_i E(R_i) - R_f}{\sqrt{w^T \Sigma w}} - \lambda\left(\sum_{i=1}^{n} w_i - 1\right)$$

Where λ is the Lagrange multiplier.

We derive the Lagrange function with respect to w and λ :

- First optimality condition: Derivative with respect to w :

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{E(R_i) - R_f}{\sigma_p} - \lambda \Leftrightarrow \frac{E(R_i) - R_f}{\lambda \sigma_p}$$

- Second condition: Allocation constraint:

$$\sum_{i=1}^{n} w_i = 1$$

By replacing w_i :

$$\sum_{i=1}^{n} \frac{E(R_i) - R_f}{\lambda \sigma_p} = 1$$

This allows us to find λ, and then calculate the w_i.

In the code, we use the SLSQP (Sequential Least Squares Programming) to solve this problem:

```
# Optimization Process
optimal = minimize(portfolio_performance, initial_guess, args=(mean_returns, cov_matrix), method='SLSQP', bounds=bounds, constraints=constraints)
```

Tangential Portfolio and Sharpe Ratio:

The tangential portfolio is the optimal portfolio where an investor can include a risk-free asset. This portfolio is obtained when the Sharpe ratio is maximized.

The tangential portfolio is the one located on the efficient frontier and tangent to the Capital Market Line (CML). It maximizes:

$$S = \frac{E(R_p) - R_f}{\sigma_p}$$

Calculation of the Optimal Weights of the Tangential Portfolio:

The optimal weights are given by:

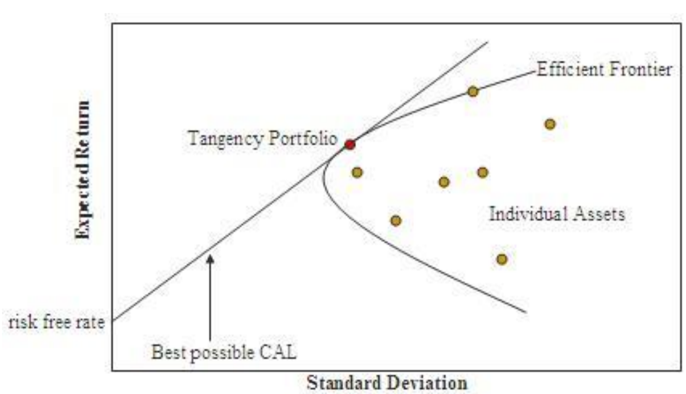
$$w^* = \frac{\Sigma^{-1}(E(R) - R_f 1)}{1^T \Sigma^{-1}(E(R) - R_f 1)}$$

where:

- Σ^{-1} is the inverse of the covariance matrix,
- $E(R)$ is the vector of expected returns,
- 1 is a vector of 1s of size n.

Graphical Interpretation:

If we represent all possible portfolios (scatter plot), the efficient frontier is the curve of the best possible portfolios. The tangential portfolio is the one that touches this frontier and has the best Sharpe ratio.



In the code, the vector of optimal weights w^* is obtained by:

```
optimal_weights = optimal.x  # Extract optimal weights
```

Monte Carlo Simulation (Efficient Frontier)

We generate 10,000 random portfolios using weights w_i drawn from the Dirichlet distribution:

Monte Carlo simulation is a technique that allows us to explore a large number of random portfolios to visualize the efficient frontier and compare these portfolios to the optimal portfolio determined by Markowitz's optimization.

We generate the random weights w_i for each portfolio using the Dirichlet distribution. This distribution ensures that the weights:

- Are positive ($w_i \geq 0$),
- Sum to 1 $\sum_{i=1}^{n} w_i = 1,$ thus respecting the total allocation constraint.

Here how it's work in the code:

```
    weights = np.random.dirichlet(np.ones(num_assets))  # Generate random weights
```

For each portfolio:

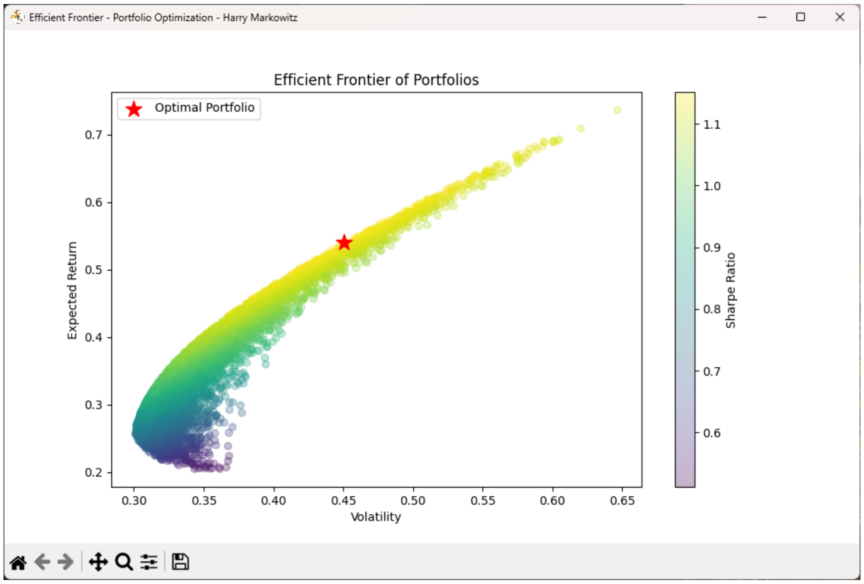
- We calculate the expected return R_p,
- We calculate the volatility σ_p,
- We deduce the Sharpe ratio.

The results are stored and displayed on an Efficient Frontier graph.

Display of Results

a) Efficient Frontier Graph:

- The yellow/green/… points represent the random portfolios.
- The red star represents the optimal portfolio.

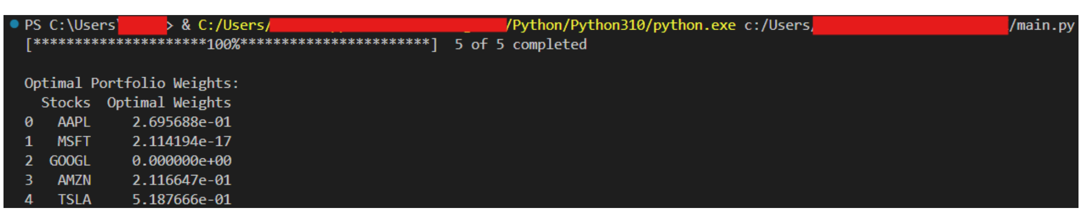


b) Display of the Optimal Weights in a DataFrame:

```
# Display optimal weights as a DataFrame
df_weights = pd.DataFrame({'Stocks': assets, 'Optimal Weights': optimal_weights})
```

Which displays something close to this:

	Optimal Portfolio Weights	
	Stocks	Optimal Weights
0	AAPL	2.695681e-01
1	MSFT	6.049848e-17
2	GOOGL	0.000000e+00
3	AMZN	2.116649e-01
4	TSLA	5.187670e-01



```
PS C:\Users[redacted]> & C:/Users/[redacted]/Python/Python310/python.exe c:/Users[redacted]/main.py
[*********************100%***********************]  5 of 5 completed

Optimal Portfolio Weights:
  Stocks  Optimal Weights
0   AAPL     2.695688e-01
1   MSFT     2.114194e-17
2  GOOGL     0.000000e+00
3   AMZN     2.116647e-01
4   TSLA     5.187666e-01
```

Sources

PDF: *Markowitz Mean-Variance Portfolio Theory*:

- https://sites.math.washington.edu/~burke/crs/408/fin-proj/mark1.pdf

Image: *Representation of the Efficient Frontier* – Wikipédia :

- https://en.wikipedia.org/wiki/Efficient_frontier

YouTube Video: *Finance quantitative/ Markowitz(1952), théorie portefeuille* - Olivier M :

- https://youtu.be/pw5dRqxLZds?si=GQMd44LzQvl3GCER

YouTube Video: *Markowitz Model and Modern Portfolio Theory – Explained* - Finance Explained:

- https://youtu.be/VsMpw-qnPZY?si=8L7n35layAbtg6BL

YouTube Playlist: *Markowitz Portfolio Theory – cfa-course.com* - cfa-course.com

- https://www.youtube.com/watch?v=VI64oviqs8&list=PL6iML4gJVrYfRTRD_EEw34n2oObUfQ81p

YouTube Video: *Le ratio de Sharpe* - Marc's Workshop :

- https://youtu.be/HSD7TbuztCo?si=J8SQiFBL_4EBntxj