Modern Portfolio Theory, Capital Asset Pricing, and Application in Python

Farouk Bouarara, B3 Economics Student

August 2025

1 Introduction

Objective: To find the minimum variance portfolio using the Capital Asset Pricing Model (CAPM) and Modern Portfolio Theory (MPT) in Python.

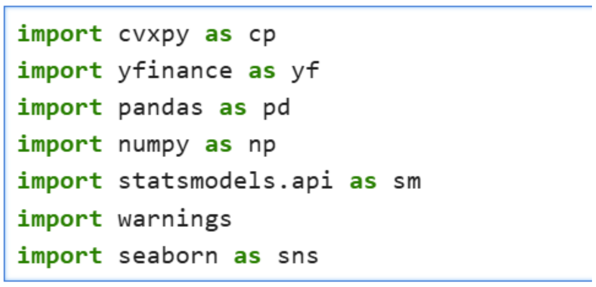
Plan:

- 1. Introduction
- 2. Returns, variance, volatility and covariance
- 3. Capital Asset Pricing Model (CAPM)
- 4. Markowitz's Mean-Variance Optimization (MVO)
- 5. Principle of Backtesting.

This project has been made possible thanks to the book *Optimization Methods in Finance* by Gerard Cornuéjols and Reha Tütüncü, Carnegie Mellon University, Pittsburgh, PA 15213 USA, January 2006 – Chapter 8: QP Models – Portfolio Optimization (p.139).

Notions used: Statistics, Econometrics, Optimization, Portfolio concepts.

The different methods will be explained, but the underlying financial theory will not be covered. All calculations will be carried out in Python. Below, the library used in this PDF.



```python
import cvxpy as cp
import yfinance as yf
import pandas as pd
import numpy as np
import statsmodels.api as sm
import warnings
import seaborn as sns
```

This PDF supports my project on building MVO and CAPM in Python. The code is available on my GitHub:

Disclaimer: This project was completed by a third-year Economics student and not by a professional. It may contain errors.

2 Returns, variance, volatility and covariance

2.1 Returns

The traditional method is the arithmetic returns:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

We can also use the log-return method for continuous-time modeling:

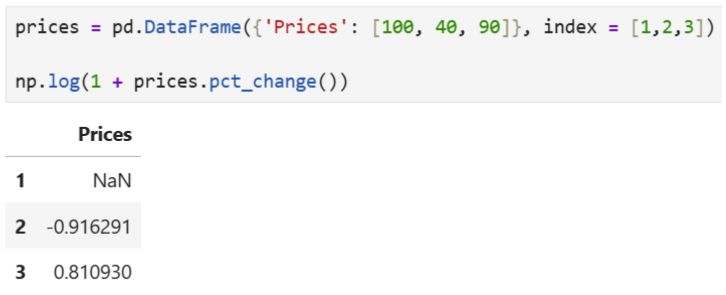
$$R_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

Period	Price	Arithmetic Return	Log Return
1	100	/	/
2	40	-0.60	-0.916
3	90	1.25	0.811
Arithmetic Mean		✗ 0.325	✗ -0.053
Geometric Mean		✓ -0.051	✓ $\exp(-0.053) - 1 \approx -0.051$
Summing Returns		✗ 0.65	✓ $\exp(\sum(R_i) = -0.105) - 1 = -0.1$
Multiplying Returns		✓ -0.1	-0.848 ✗

Table 1: Example of arithmetic vs. log returns and mean rates, with good (green) and bad (red) practices marked

To get the **average return**, we should use the **geometric mean**, as it accounts for the compounding effect. To get the **total return** over multiple periods, the method depends on the type of returns used: multiply $(1 + r)$ for simple returns, or sum them for log returns (then exponentiate if needed).

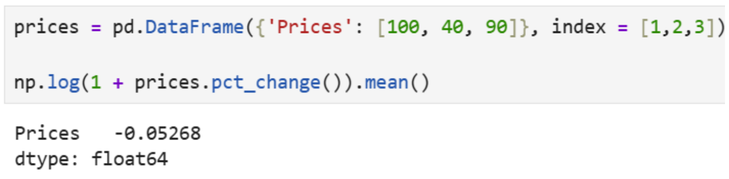
Screenshot in Python:



```
prices = pd.DataFrame({'Prices': [100, 40, 90]}, index = [1,2,3])

np.log(1 + prices.pct_change())
```

```
      Prices
1        NaN
2  -0.916291
3   0.810930
```

Figure 1: Log Returns



```
prices = pd.DataFrame({'Prices': [100, 40, 90]}, index = [1,2,3])

np.log(1 + prices.pct_change()).mean()
```

```
Prices   -0.05268
dtype: float64
```

Figure 2: Mean Returns

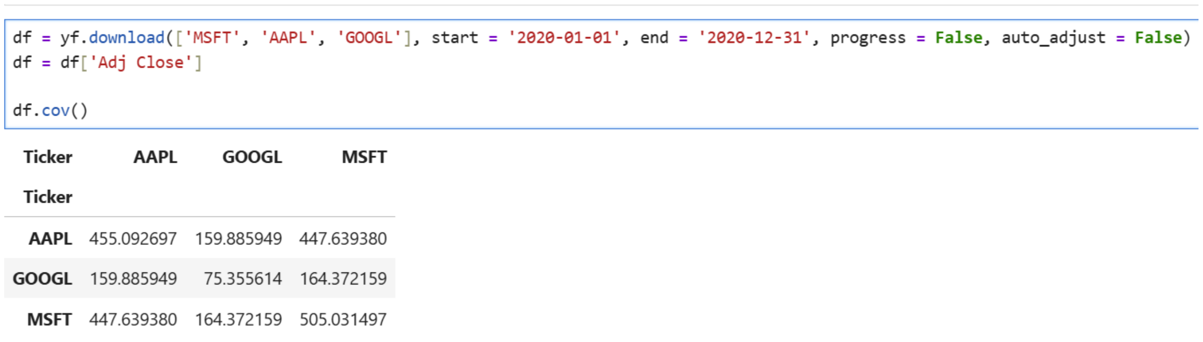
2.2 Variance, volatility, covariance and correlation

The following formulas give the variance and covariance of returns for assets i and j:

$$\text{Var}(R_i) = \frac{\sum(R_i - \bar{R}_i)^2}{n-1}$$

$$\text{Cov}(R_i, R_j) = \frac{\sum(R_i - \bar{R}_i)(R_j - \bar{R}_j)}{n-1}$$

In Python, we can compute these using the covariance matrix:



```
df = yf.download(['MSFT', 'AAPL', 'GOOGL'], start = '2020-01-01', end = '2020-12-31', progress = False, auto_adjust = False)
df = df['Adj Close']

df.cov()
```

Ticker	AAPL	GOOGL	MSFT
Ticker			
AAPL	455.092697	159.885949	447.639380
GOOGL	159.885949	75.355614	164.372159
MSFT	447.639380	164.372159	505.031497

Figure 3: Computing the covariance matrix in Python

For the volatility of an asset i, we have:

$$\sigma_i = \sqrt{\text{Var}(R_i)}.$$

Finally, we can obtain the correlation between two assets using the formula:

$$\rho_{i,j} = \frac{\text{Cov}(R_i, R_j)}{\sigma_i \, \sigma_j}.$$

- Covariance $\rightarrow$ Tells us if two variables move in the same direction $(+)$ or opposite $(-)$.
- Correlation $\rightarrow$ Tells us both the direction (negative or positive) and the strength (close to 1) of the relationship.

The more assets are correlated with each other, the riskier it is to hold them both in a portfolio. The mean–variance optimization (MVO) consists of minimizing the portfolio variance by choosing asset weights such that the assets are as little correlated as possible.

Visualisation in Python:

```
df = yf.download(['MSFT', 'AAPL', 'GOOGL'], start = '2020-01-01', end = '2020-12-31', progress = False, auto_adjust = False)
df = df['Adj Close']

sns.heatmap(df.corr(), annot = True, cmap = 'Reds' )
```

```
<Axes: xlabel='Ticker', ylabel='Ticker'>
```

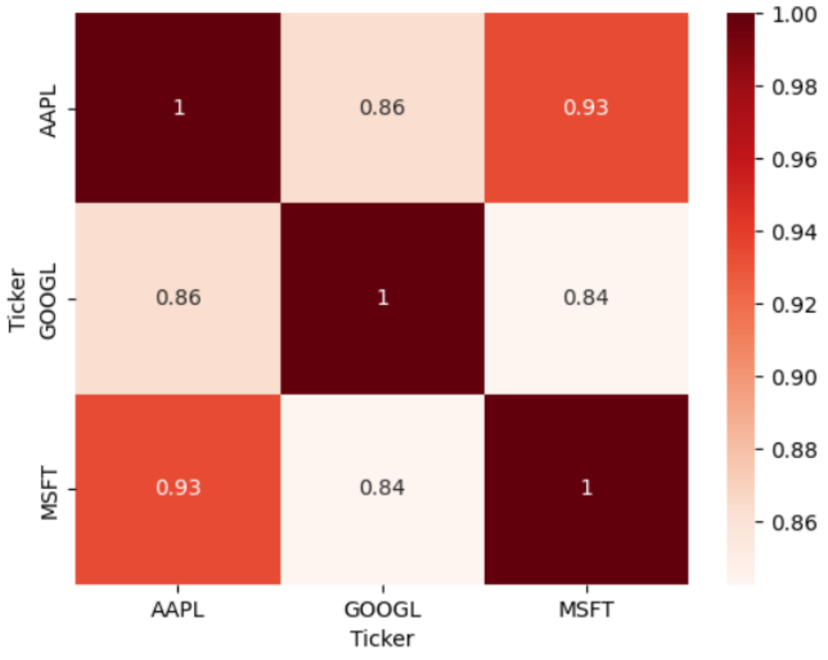


Figure 4: Heatmap for correlations in Python

3 Capital Asset Pricing Model (CAPM)

We perform the following regression (OLS) to calculate the β of an asset i:

$$R_{i,t} = \alpha + \beta i R_{m,t} + \epsilon$$

Alternatively, we can use directly the OLS formula for simple linear regression to determine β:

$$\beta_i = \frac{\text{Cov}(R_i, R_m)}{\text{Var}(R_m)}$$

Then, we compute the expected return of an asset i using the CAPM formula:

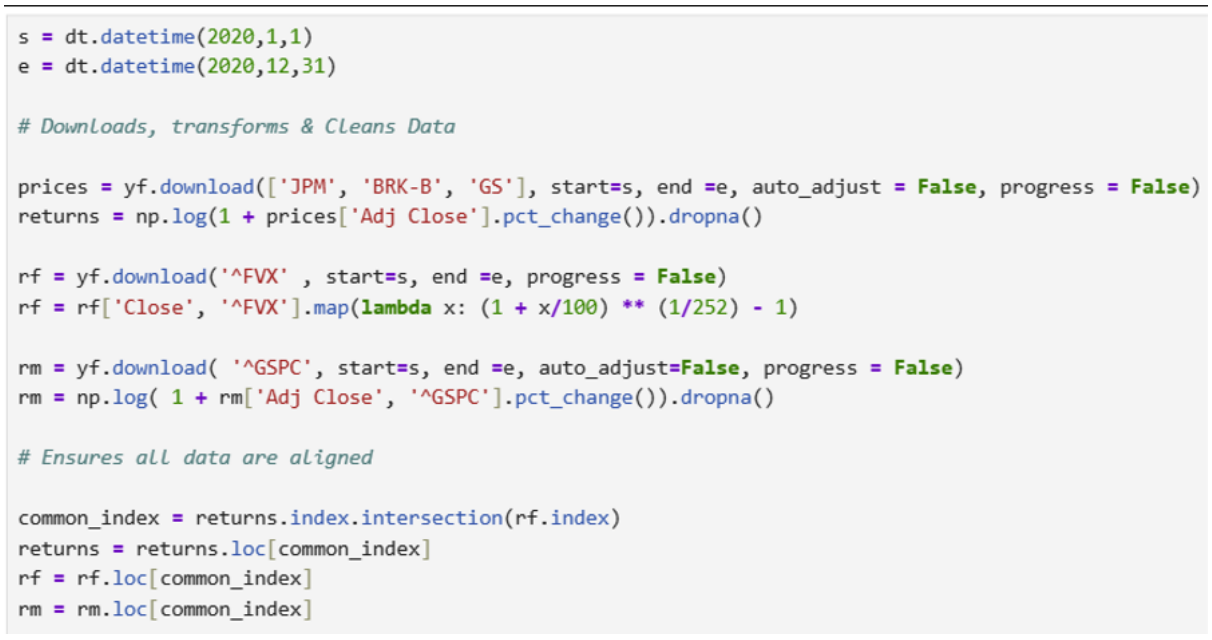
$$E(R_{i,t+1}) = E(R_f) + \beta_i E(R_m - R_f)$$

where $R_{i,t+1}$ is the expected return of the asset i in the next period, R_f is the risk-free rate, R_m is the market return, and β measures the asset's sensitivity to the market.

In the dataset (from Yahoo Finance), the returns of the risk-free asset R_f (e.g., US 5-year treasury yield index) are annual, unlike the daily computed returns of assets R_i and the market R_m (e.g. S&P 500). Considering 252 trading days in a year, we use the geometric mean formula:

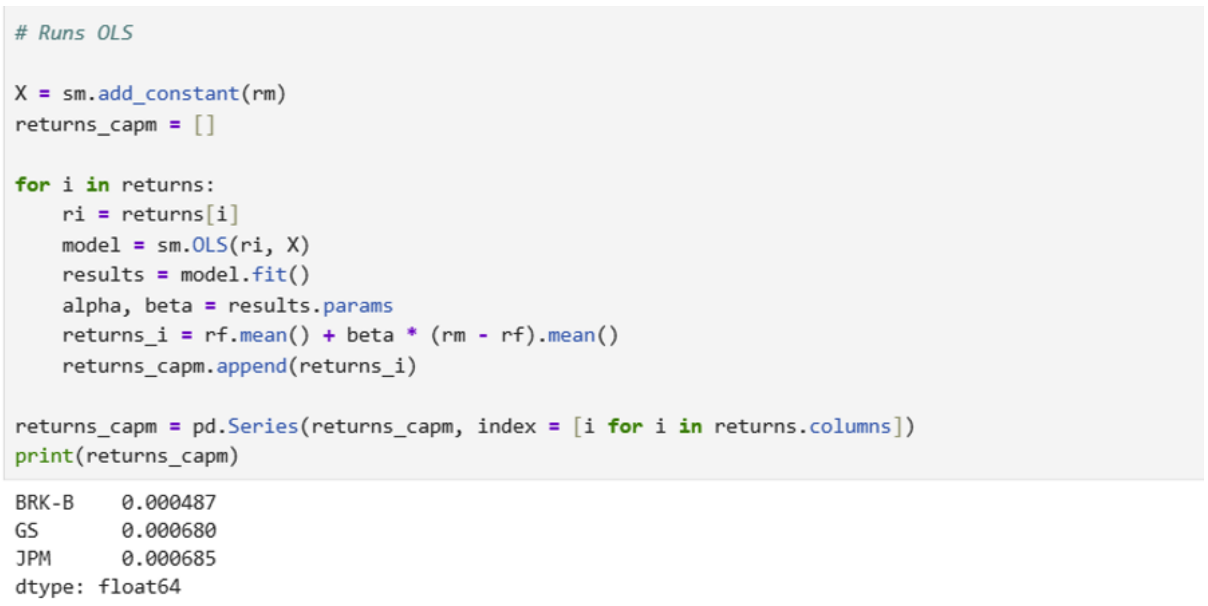
$$(1 + R_{\text{annual}}) = (1 + \bar{R}_{\text{daily}})^{252}$$

Screenshot from Python:



```python
s = dt.datetime(2020,1,1)
e = dt.datetime(2020,12,31)

# Downloads, transforms & Cleans Data

prices = yf.download(['JPM', 'BRK-B', 'GS'], start=s, end =e, auto_adjust = False, progress = False)
returns = np.log(1 + prices['Adj Close'].pct_change()).dropna()

rf = yf.download('^FVX' , start=s, end =e, progress = False)
rf = rf['Close', '^FVX'].map(lambda x: (1 + x/100) ** (1/252) - 1)

rm = yf.download( '^GSPC', start=s, end =e, auto_adjust=False, progress = False)
rm = np.log( 1 + rm['Adj Close', '^GSPC'].pct_change()).dropna()

# Ensures all data are aligned

common_index = returns.index.intersection(rf.index)
returns = returns.loc[common_index]
rf = rf.loc[common_index]
rm = rm.loc[common_index]
```

Figure 5: Performing CAPM for Daily Expected Returns (1/2)



```python
# Runs OLS

X = sm.add_constant(rm)
returns_capm = []

for i in returns:
    ri = returns[i]
    model = sm.OLS(ri, X)
    results = model.fit()
    alpha, beta = results.params
    returns_i = rf.mean() + beta * (rm - rf).mean()
    returns_capm.append(returns_i)

returns_capm = pd.Series(returns_capm, index = [i for i in returns.columns])
print(returns_capm)
```

```
BRK-B    0.000487
GS       0.000680
JPM      0.000685
dtype: float64
```

Figure 6: Performing CAPM for Daily Expected Returns (2/2)

4 Markowitz's Mean-Variance Optimization

4.1 Quadratic Optimization

4.1.1 Matrix representation of a quadratic form

We can rewrite any quadratic problem in a matrix form.

$$Q(x, y) = ax^2 + bxy + cy^2$$

This can be written as

$$Q(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & \frac{b}{2} \\ \frac{b}{2} & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The matrix is necessarily symmetric.

4.1.2 Solving quadratic optimization problem with KKT system

$$\min_{w \in \mathbb{R}^n} \ \tfrac{1}{2} w^T Q w$$

$$Aw = b, \quad Gw \leq h, \quad \mathbf{w} = \begin{bmatrix} x \\ y \end{bmatrix}$$

The $\frac{1}{2}$ is here just to simplify the calculus when we take derivatives (the factor 2 is cancelled, leaving only the matrix times the variable vector w).

Lagrangian :

$$L(w, \lambda_1, \lambda_2) = \tfrac{1}{2} w^T Q w - \lambda_1 (Aw - b) - \lambda_2 (Gw - h)$$

First-order condition:

$$\nabla L = 0$$

Thanks to the FOC we can fill the KKT system and solve the problem:

$$\begin{bmatrix} Q & A^T & G^T \\ A & 0 & 0 \\ G & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ b \\ h \end{bmatrix}$$

Second-order condition:

$$\nabla^2_w L = Q.$$

So we have to analyze the eigenvalues of the matrix Q to know wheter the solution found is a minimum or maximum.

$$\det(Q - \lambda I) = 0, \quad \lambda \in \mathrm{sp}(Q)$$

If all eigenvalues are positive (resp. non-negative), then the matrix is positive definite (resp. positive semidefinite). In that case, the objective function is strictly convex (resp. convex), and therefore there exists a global minimum.

If all eigenvalues are negative (resp. non-positive), then the matrix is negative definite (resp. negative semidefinite). In that case, the objective function is strictly concave (resp. concave), and therefore there exists a global maximum.

If the matrix has both positive and negative eigenvalues, then the matrix is indefinite, and the objective function is neither convex nor concave. In this case, there may exist saddle points instead of a global extremum.

Finally, once a solution is found, we need to check our KKT conditions:

- **Stationnarity:** This is verified by construction.

$$\nabla L = 0$$

- **Primal feasibility:** All constraints must be satisfied.
- **Dual feasibility:** All the Lagrange multipliers associated with inequality constraints must be non-negative.
- **Complementary slackness:** For inequality constraints, the product of each constraint function with its corresponding multiplier must be zero.

4.1.3 Quadratic optimization - Example

$$\begin{aligned} \text{min} \quad & Q = x^2 + 2y^2 + 3z^2 \\ \text{s.t} \quad & 5x - y - 3z \geq 3, \\ & 2x + y + 2z + y \geq 6, \\ & x, y, z \geq 0 \end{aligned}$$

To solve this by hand, we turn the inequality constraints into equalities. Then, we apply the first-order condition to verify stationarity. If the solutions found satisfy the 3 other points of the KKT conditions, we have reached the optimum. If not, we need to try a different set of bounds in the equality-turned constraints that still respect our initial inequality constraints. In practice, this method is iterative, and we generally rely on computer solvers to find the solution efficiently.

In our example, we have :

$$Q(x, y, z) = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

The optimization problem becomes :

$$
\begin{aligned}
\min \quad & \tfrac{1}{2}\, w^T Q w \\
\text{s.t.} \quad & \begin{bmatrix} 5 \\ -1 \\ -3 \end{bmatrix}^T w \geq 3, \\
& \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}^T w \geq 6, \\
& x, y, z \geq 0
\end{aligned}
$$

We bind our optimization problem:

$$
\begin{aligned}
\min \quad & \tfrac{1}{2}\, w^T Q w \\
\text{s.t.} \quad & \begin{bmatrix} 5 \\ -1 \\ -3 \end{bmatrix}^T w = 3, \\
& \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}^T w = 6,
\end{aligned}
$$

We write the Lagrangian :

$$
L(w, \lambda_1, \lambda_2) = \tfrac{1}{2} w^T Q w - \lambda_1 (\begin{bmatrix} 5 \\ -1 \\ -3 \end{bmatrix}^T w - 3) - \lambda_2 (\begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}^T w - 6,)
$$

First order condition:

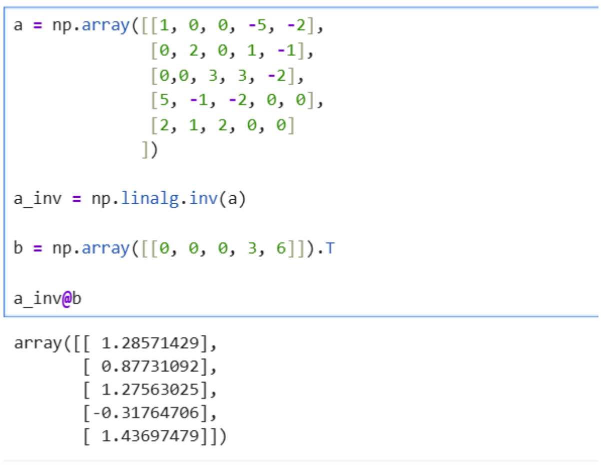
$$
\nabla L = \begin{cases} x - 5\lambda_1 - 2\lambda_2 = 0 \\ 2y + \lambda_1 - \lambda_2 = 0 \\ 3z + 3\lambda_1 - 2\lambda_2 = 0 \end{cases}
$$

Now we can fill our KKT system thanks to the FOC:

$$
Ax = b
$$

$$
\begin{bmatrix} 1 & 0 & 0 & -5 & -2 \\ 0 & 2 & 0 & 1 & -1 \\ 0 & 0 & 3 & 3 & -2 \\ 5 & -1 & -3 & 0 & 0 \\ 2 & 1 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 3 \\ 6 \end{bmatrix}
$$

To solve this, we can either use Gauss Pivot or invert the matrix A. We do the second option in Python :

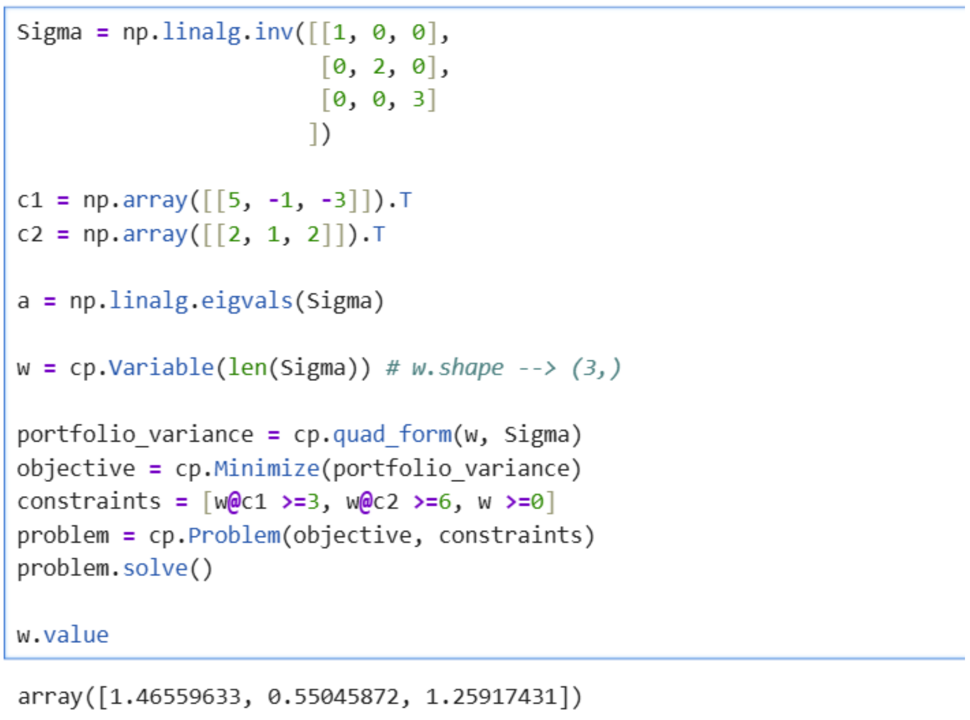


```python
a = np.array([[1, 0, 0, -5, -2],
              [0, 2, 0, 1, -1],
              [0,0, 3, 3, -2],
              [5, -1, -2, 0, 0],
              [2, 1, 2, 0, 0]
             ])

a_inv = np.linalg.inv(a)

b = np.array([[0, 0, 0, 3, 6]]).T

a_inv@b
```

```
array([[ 1.28571429],
       [ 0.87731092],
       [ 1.27563025],
       [-0.31764706],
       [ 1.43697479]])
```

Figure 7: Manual Optimization in Python

We can see that λ_1 is negative while it is associated with an inequality constraint. Therefore, one of our KKT conditions is not verified. We can build some intuition graphically and try another set, or we can use a different method in Python that directly takes into account our KKT conditions. We solve the problem with CVXPY in Python. CVXPY specializes in convex optimization problems (i.e., linear and quadratic programs), and returns solutions that satisfy the KKT conditions for optimality.



```python
Sigma = np.linalg.inv([[1, 0, 0],
                       [0, 2, 0],
                       [0, 0, 3]
                      ])

c1 = np.array([[5, -1, -3]]).T
c2 = np.array([[2, 1, 2]]).T

a = np.linalg.eigvals(Sigma)

w = cp.Variable(len(Sigma)) # w.shape --> (3,)

portfolio_variance = cp.quad_form(w, Sigma)
objective = cp.Minimize(portfolio_variance)
constraints = [w@c1 >=3, w@c2 >=6, w >=0]
problem = cp.Problem(objective, constraints)
problem.solve()

w.value
```

```
array([1.46559633, 0.55045872, 1.25917431])
```

Figure 8: Solving quadratic problems with CVXPY in Python

This is our optimum.

4.2 Portfolio optimization Definition

The following formulas give the portfolio's expected return and variance:

$$R_p = \sum_i w_i R_i$$

$$\sigma_p^2 = \text{Var}\left(\sum_i w_i R_i\right)$$

where w_i is the weight of asset i in the portfolio, and R_i is its expected return (e.g., from CAPM or another model such as Fama–French or Black–Litterman).

Let's assume a portfolio composed of three assets i, j, and k, we have:

$$R_p = w_i R_i + w_j R_j + w_k R_k$$

$$\sigma_p^2 = \text{Var}(w_i.R_i + w_j.R_j + w_k.R_k)$$

$$\sigma_p^2 = w_i^2.\text{Var}(R_i)+w_j^2.\text{Var}(R_j)+w_k^2.\text{Var}(R_k)+2.w_i.w_j.\text{Cov}(R_i,R_j)+2.w_i.w_k.\text{Cov}(R_i,R_k)+2.w_k.w_j.\text{Cov}(R_k,R_j)$$

We can rewrite this in a symmetric matrix form:

$$\sigma_p^2 = \begin{bmatrix} w_i\text{Var}(R_i) + w_j\text{Cov}(R_i,R_j) + w_k\text{Cov}(R_i,R_k) \\ w_i\text{Cov}(R_j,R_i) + w_j\text{Var}(R_j) + w_k\text{Cov}(R_j,R_k) \\ w_i\text{Cov}(R_k,R_i) + w_j\text{Cov}(R_k,R_j) + w_k\text{Var}(R_k) \end{bmatrix}^T \begin{bmatrix} w_i \\ w_j \\ w_k \end{bmatrix}$$

$$\sigma_p^2 = \begin{bmatrix} w_i & w_j & w_k \end{bmatrix} \begin{bmatrix} \text{Var}(R_i) & \text{Cov}(R_i,R_j) & \text{Cov}(R_i,R_k) \\ \text{Cov}(R_i,R_j) & \text{Var}(R_j) & \text{Cov}(R_j,R_k) \\ \text{Cov}(R_i,R_k) & \text{Cov}(R_j,R_k) & \text{Var}(R_k) \end{bmatrix} \begin{bmatrix} w_i \\ w_j \\ w_k \end{bmatrix}$$

$$\sigma_p^2 = w^t \Sigma w$$

This is the covariance matrix. We can demonstrate that such matrices have two properties: their eigenvalues are real (=symmetric) and non-negative (=covariance). Therefore, they are either positive definite (strictly convex with a global minimum) or positive semi-definite (convex with a global minimum). Consequently, we can define the minimization problem as follows:

$$\begin{aligned} \min_{w} \quad & \sigma_p^2 = \frac{1}{2} w^T \Sigma w \\ \text{s.t.} \quad & \mathbf{1}^T w = 1, \\ & Cw \geq d, \\ & Cw \leq e \\ & w^T \mu \geq R \end{aligned}$$

- The objective function is the portfolio variance.
- The first constraint states that the sum of all weights w_i is equal to 1 (we want to invest all our money).
- The second constraint enforces lower bounds on the portfolio weights, expressed as $Cw \geq d$. If only individual asset bounds are considered (e.g., no short selling with $d = \mathbf{0}$), then C is simply the identity matrix. For more complex constraints such as sector weight limits or other group exposure limits, C contains the appropriate coefficients for each group in its rows.

  This constraint can also be used to promote portfolio diversification. In practice, mean–variance optimization often over-allocates to a small subset of assets (typically those with the lowest pairwise correlations) at the expense of the rest. It is common to obtain a portfolio that selects only a few dozen assets out of several hundred, sometimes with a single asset dominating (e.g., a 20% weight). Lower-bound constraints help prevent this by ensuring a minimum allocation to selected assets and by limiting the degree of concentration in the portfolio.
- The third constraint works the same way as the second one, but it imposes upper bounds (maximum allocations).
- The last constraint sets a minimum return to reach. With μ the expected returns for each asset (estimated thanks to CAPM or other models).

4.3 Solving Portfolio optimization

We can write our Lagrangian as:

$$L(w, \lambda) = \frac{1}{2} w^T \Sigma w - \lambda_1 \left(\mathbf{1}^T w - 1 \right) - \lambda_2^T \left(Cw - d \right) - \lambda_3 \left(w^T \mu - R \right)$$

To find an acceptable solution, we can define the Karush–Kuhn–Tucker (KKT) conditions:

- **Stationarity:** There is no better feasible solution that reduces our variance:

$$\nabla L = 0$$

  To solve this, we define the KKT system:

$$\begin{bmatrix} \Sigma & -\mathbf{1} & -C_J^\top & -\mu \\ \mathbf{1}^\top & 0 & 0 & 0 \\ C_J & 0 & 0 & 0 \\ \mu^\top & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda_1 \\ \lambda_{2,J} \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ d_J \\ R \end{bmatrix}$$

- **Primal feasibility:** All constraints must be satisfied.

$$\begin{aligned} &\mathbf{1}^T w^* = 1, \\ &Cw^* \geq d, \\ &w^{*T} \mu \geq R \end{aligned}$$

- **Dual feasibility:** All the Lagrange multipliers associated with inequality constraints must be non-negative.

- **Complementary slackness:** For inequality constraints, the product of each constraint function with its corresponding multiplier must be zero.

$$\lambda_2(Cw^* - d) = 0,$$
$$\lambda_4(w^{*T}\mu - R) = 0$$

4.4 Example

Let's take the following matrix :

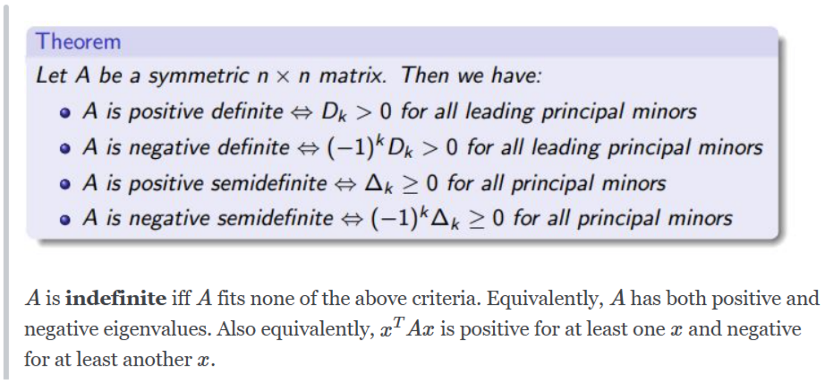
$$\Sigma = \begin{bmatrix} 7 & 3 & 4 \\ 3 & 6 & 5 \\ 4 & 5 & 10 \end{bmatrix}$$

- It is a symmetric matrix.
- All its diagonal values are positive.
- It is a positive definite (PD) or semi-definite (PSD) matrix. This can be shown in two ways:
  1. By determining its leading principal minors and showing that they are all non-negative.
  2. By determining all its eigenvalues and showing that they are non-negative.

For the first method:

- $\det(|7|) = 7 > 0$
- $\det\left(\begin{vmatrix} 7 & 3 \\ 3 & 6 \end{vmatrix}\right) = 33 > 0$
- $\det(\Sigma) = 179 > 0$

More generally (we can also extend this theorem using eigenvalues):



Theorem

Let A be a symmetric $n \times n$ matrix. Then we have:

- *A is positive definite $\Leftrightarrow D_k > 0$ for all leading principal minors*
- *A is negative definite $\Leftrightarrow (-1)^k D_k > 0$ for all leading principal minors*
- *A is positive semidefinite $\Leftrightarrow \Delta_k \geq 0$ for all principal minors*
- *A is negative semidefinite $\Leftrightarrow (-1)^k \Delta_k \geq 0$ for all principal minors*

A is **indefinite** iff A fits none of the above criteria. Equivalently, A has both positive and negative eigenvalues. Also equivalently, $x^T Ax$ is positive for at least one x and negative for at least another x.

Figure 9: Leading principal minors theorem for symmetric matrices

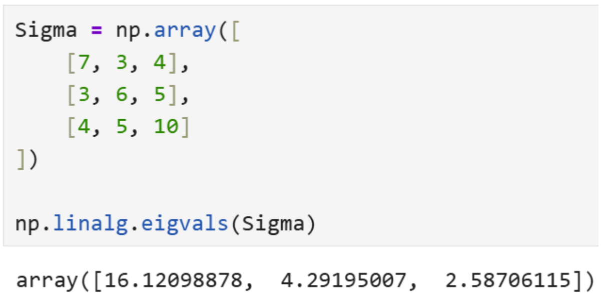
For the second method, we find the eigenvalues λ by solving:

$$\Lambda(\lambda) = \det(\Sigma - \lambda I) = 0$$

For our example, we obtain the following expanded formula:

$$\Lambda(\lambda) = (7-\lambda)\big((6-\lambda)(10-\lambda)-25\big) - 3\big(3(10-\lambda)-20\big) + 4\big(15-4(6-\lambda)\big) = 0$$

To solve this, we can directly use Python. As the screenshot below shows, all eigenvalues obtained are real and positive. Therefore, this matrix is indeed positive definite (PD), its optimum corresponds to a minimum, and consequently, together with the previous two points, it qualifies as a covariance matrix.



```
Sigma = np.array([
    [7, 3, 4],
    [3, 6, 5],
    [4, 5, 10]
])

np.linalg.eigvals(Sigma)
```

```
array([16.12098878,  4.29195007,  2.58706115])
```

Figure 10: Eigenvalues in Python

We can check the following equality to ensure that the eigenvalues obtained are consistent:

$$\mathrm{tr}(\Sigma) = \sum \lambda_i \quad \text{with } \lambda_i \in \mathrm{sp}(\Sigma)$$

Finally, we can conclude about the signature of the matrix:

$$\mathrm{sign}(\Sigma) = (p = 3,\, q = 0,\, r = 0)$$

We can now solve the optimization problem. Let's rewrite it such as :

$$\begin{aligned} \min_{w} \quad & \sigma_p^2 = \frac{1}{2} w^T \Sigma w \\ \text{s.t.} \quad & \mathbf{1}^T w = 1, \\ & w \geq 0, \\ & w^T \mu \geq 0.5 \end{aligned}$$

We assume that the expected returns are such as:

$$\mu = \begin{bmatrix} R_i = 0.7 \\ R_j = 0.2 \\ R_k = 0.4 \end{bmatrix}$$

This is a convex (PD) quadratic (degree 2) optimization problem.

All we can do to solve optimization problems with constraints that have inequalities by hand is bind them. The problem is, if we bind the second constraint in our optimization problem, the problem itself no longer exists. For instance, if we take the entire vector $w = 0$ in this case, it becomes useless to optimize weights since the problem is already solved. Moreover, it violates our first constraint. If we bind only w_i, that won't guarantee that one of the two other weights will not be negative either. Therefore, we solve the system without taking this constraint into account first. If one weight is negative, we will correct it *ex post* by binding it to 0, then optimize again.

We define the Lagrangian problem as:

$$L(w, \lambda) = \frac{1}{2} w^T \Sigma w - \lambda_1(\mathbf{1}^T w - 1) - \lambda_2(w^T \mu - 0.5)$$

First-order condition:

$$\frac{\partial L}{\partial w} = \Sigma w - \lambda_1 \mathbf{1^T} - \lambda_2 \mu = 0$$

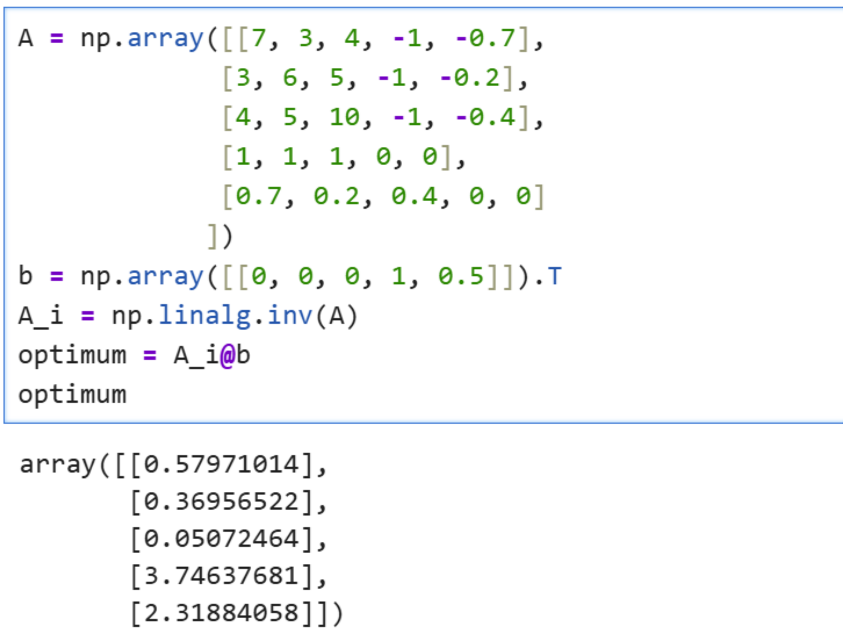
We write the KKT system as:

$$\begin{bmatrix} \Sigma & -\mathbf{1} & -\mu \\ \mathbf{1}^\top & 0 & 0 \\ \mu^\top & 0 & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ R \end{bmatrix}$$

$$Ax = b$$

$$\begin{bmatrix} 7 & 3 & 4 & -1 & -0.7 \\ 3 & 6 & 5 & -1 & -0.2 \\ 4 & 5 & 10 & -1 & -0.4 \\ 1 & 1 & 1 & 0 & 0 \\ 0.7 & 0.2 & 0.4 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_i \\ w_j \\ w_k \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0.5 \end{bmatrix}$$

There are two main methods for solving this: either we use Gaussian elimination or we invert the matrix A. We perform the second method in Python:

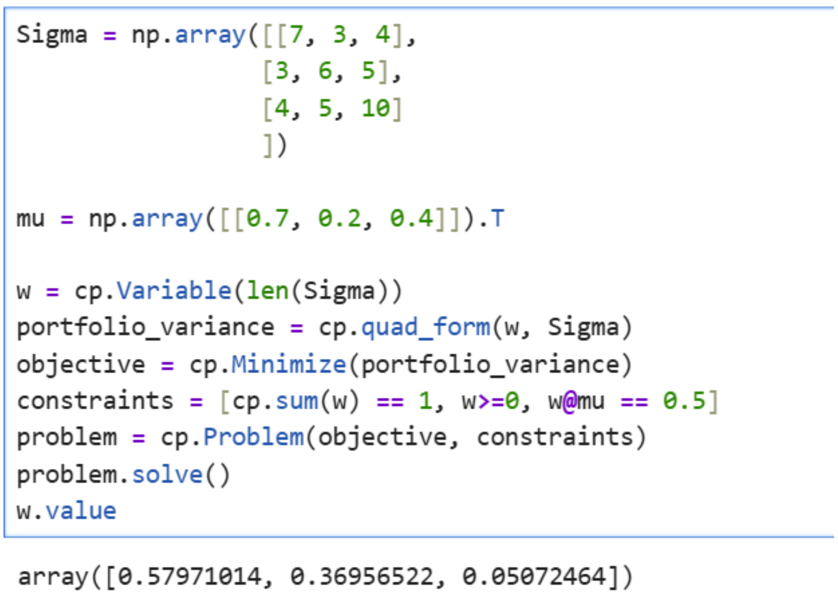


```
A = np.array([[7, 3, 4, -1, -0.7],
              [3, 6, 5, -1, -0.2],
              [4, 5, 10, -1, -0.4],
              [1, 1, 1, 0, 0],
              [0.7, 0.2, 0.4, 0, 0]
             ])
b = np.array([[0, 0, 0, 1, 0.5]]).T
A_i = np.linalg.inv(A)
optimum = A_i@b
optimum
```

```
array([[0.57971014],
       [0.36956522],
       [0.05072464],
       [3.74637681],
       [2.31884058]])
```

Figure 11: Solving analytically a system in Python (1/2)

We now see that all KKT conditions are satisfied. To check this, we apply the 2nd method, we should recover the same solution.



```
Sigma = np.array([[7, 3, 4],
                  [3, 6, 5],
                  [4, 5, 10]
                  ])

mu = np.array([[0.7, 0.2, 0.4]]).T

w = cp.Variable(len(Sigma))
portfolio_variance = cp.quad_form(w, Sigma)
objective = cp.Minimize(portfolio_variance)
constraints = [cp.sum(w) == 1, w>=0, w@mu == 0.5]
problem = cp.Problem(objective, constraints)
problem.solve()
w.value
```

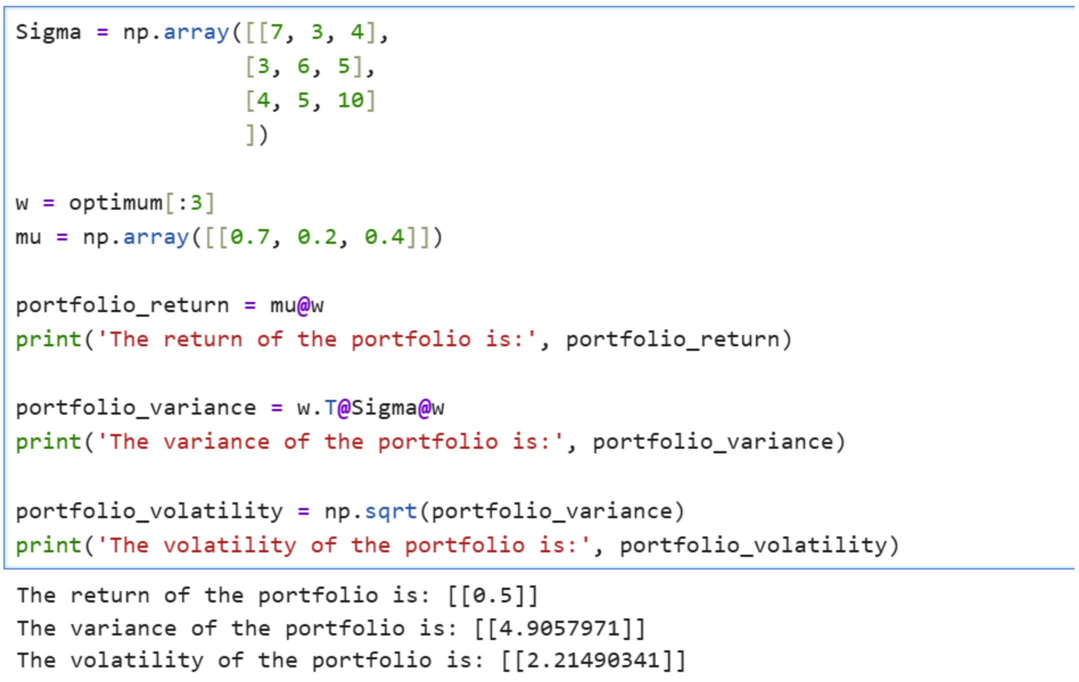
```
array([0.57971014, 0.36956522, 0.05072464])
```

Figure 12: Solving the problem with CVXPY

As shown, after including all constraints, we obtain the same solution as before.

Remark: The target portfolio return (here 0.5) must lie between the minimum and maximum expected returns of the individual assets in the portfolio. Otherwise, the optimization problem becomes infeasible and Python will return an error.

Finally, we conclude about the variance, returns and volatility of the portfolio:



```
Sigma = np.array([[7, 3, 4],
                  [3, 6, 5],
                  [4, 5, 10]
                  ])

w = optimum[:3]
mu = np.array([[0.7, 0.2, 0.4]])

portfolio_return = mu@w
print('The return of the portfolio is:', portfolio_return)

portfolio_variance = w.T@Sigma@w
print('The variance of the portfolio is:', portfolio_variance)

portfolio_volatility = np.sqrt(portfolio_variance)
print('The volatility of the portfolio is:', portfolio_volatility)
```

```
The return of the portfolio is: [[0.5]]
The variance of the portfolio is: [[4.9057971]]
The volatility of the portfolio is: [[2.21490341]]
```

Figure 13: Solving analytically a system in Python (2/2)

5 Principle of Backtesting

Backtesting is a method used to evaluate the performance and robustness of a trading strategy, portfolio optimization model, or risk-management approach by applying it to historical data.

In the portfolio-management context, the idea of backtesting is the following: we place ourselves at a point in the past and use only the information that would have been available at that time. For each period, we compute the ex-ante optimal portfolio using the Mean–Variance Optimization (MVO) based on past historical data and an asset pricing model (here CAPM), and then compare it with the ex-post realized performance using the actual returns observed in the following period. Repeating this process over several periods allows us to evaluate how well the MVO strategy performed.

To evaluate the quality of this performance, we may visualize the results graphically and/or rely on a quantitative metric such as the tracking error.

$$\text{TE} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}\left(r_{p,t} - r_{b,t}\right)^2}$$

The tracking error quantifies how closely the portfolio follows its benchmark. A low value (near 0) indicates that the portfolio closely replicates the benchmark, while a high value (greater than $|4|$) reflects significant deviations in performance.