RAPPORT SUR L'IMPLEMENTATION EN PYTHON DE LA THEORIE DU PORTEFEUILLE DE HARRY MARKOWITZ

21 FEVRIER 2025

Nino Aubert – www.linkedin.com/in/ninoaubert
Etudiant en licence d'économie et gestion à SKEMA Business School

Table des Matières

Remerciements

Tout d'abord, je souhaite exprimer ma gratitude à Professeur Ram Janakiraman, PhD, dont le cours "Introduction to AI & Machine Learning for Business" et le projet de groupe "Machine Learning Applied to Finance" ont ravivé ma motivation pour le développement de projets en programmation appliquée à la finance. Son approche pédagogique et sa capacité à rendre l’intelligence artificielle accessible aux enjeux réels m’ont inspiré à approfondir mes compétences dans ce domaine.

J’adresse également mes sincères remerciements à Professeur Majed Al-Ghandour, PhD, PE, qui a pris le temps de relire mon rapport et de me fournir des retours précieux. Ses observations ont grandement contribué à l’amélioration de mon travail.

Enfin, je tiens à souligner l'importance de l’apprentissage continu et de la curiosité intellectuelle dans la réalisation de ce projet. Ce travail découle d'une véritable passion pour la programmation et l'analyse de données.

Nino Aubert,

Le 21/02/2025

À propos de moi et de mon projet

Je suis un étudiant français en deuxième année de Licence d'économie et de gestion à SKEMA Business School, actuellement en échange sur le campus de Raleigh, aux États-Unis. Mon parcours académique m'a permis d'explorer divers domaines de la gestion, de la finance et de l'économie, tout en développant un intérêt particulier pour la finance d'investissement et la gestion de portefeuille.

Au fil de mes études (lycée) et de mes projets personnels, j'ai acquis des compétences en programmation, notamment en Python. Au cours de ce projet, j'ai appris à manipuler des bibliothèques comme NumPy, Pandas, Matplotlib et SciPy, essentielles pour les débuts de l'analyse de données et la modélisation financière.

J'essaye maintenant de développer une capacité à structurer des algorithmes d'optimisation et à automatiser des tâches analytiques. Grâce à ces compétences, je veux appliquer des concepts financiers avancés à des ensembles de données réels, ce qui est un atout majeur pour l'analyse de portefeuille et la prise de décision en investissement.

Ce projet d'optimisation de portefeuille est né de ma volonté d'appliquer concrètement les théories financières apprises lors de mes recherche. J'ai voulu aller au-delà des concepts théoriques en mettant en œuvre la Théorie Moderne du Portefeuille de Harry Markowitz grâce à un outil de programmation puissant : Python.

En utilisant les bibliothèques numpy, pandas, et scipy.optimize, ce projet me permet d'explorer les notions suivantes :

- La construction d'un portefeuille d'actifs diversifiés
- Le calcul des rendements et des risques à l'aide de la matrice de covariance
- L'optimisation des poids des actifs pour maximiser le ratio de Sharpe
- La simulation de portefeuilles pour visualiser la frontière efficiente

Ce projet représente donc un premier pas vers l'application de modèles financiers avancés, et un moyen de renforcer mes compétences en programmation et en analyse de données. Il me permet d'associer mes connaissances en économie et gestion avec des outils quantitatifs, un atout indispensable pour mon futur académique et professionnel.

Qui est Harry Markowitz ?

Harry Markowitz et l'Optimisation de Portefeuille

Harry Markowitz est un économiste américain né en 1927, célèbre pour avoir développé la théorie moderne du portefeuille (Modern Portfolio Theory - MPT) dans les années 1950. Son travail révolutionnaire, qui lui a valu le prix Nobel d'économie en 1990, repose sur l'idée de diversification et d'optimisation du couple rendement-risque dans un portefeuille d'investissement. En Python, la théorie de portefeuille de Harry Markowitz, également connue sous le nom de Modern Portfolio Theory (MPT), est implémentée à l'aide de calculs mathématiques pour optimiser l'allocation des actifs dans un portefeuille en maximisant les rendements attendus tout en minimisant le risque. Cela se fait généralement en utilisant des bibliothèques comme NumPy et SciPy pour effectuer les calculs nécessaires à l'optimisation moyenne-variance, un concept clé de la théorie.

Sa Contribution à la Finance

Markowitz a introduit un cadre mathématique permettant de construire des portefeuilles optimaux en fonction du rendement espéré et du risque mesuré par la variance (ou l'écart-type) des rendements. Son approche repose sur plusieurs concepts clés :

- Diversification : Investir dans plusieurs actifs permet de réduire le risque total du portefeuille sans nécessairement diminuer le rendement attendu.
- Risque et rendement : Il établit une relation entre le rendement espéré d'un portefeuille et son risque, mesuré par la volatilité.
- Frontière efficiente : Il démontre que, pour un niveau de risque donné, il existe une combinaison optimale d'actifs offrant le rendement maximal possible. Cette courbe est connue sous le nom de « frontière efficiente ».
- Ratio de Sharpe : Bien que Markowitz n'ait pas inventé ce ratio, son travail a contribué à l'idée que le rendement excédentaire (par rapport au taux sans risque) devait être rapporté au risque pris.

Lien avec mon Projet

Mon code met en œuvre plusieurs principes de la théorie de Markowitz :

- Calcul des rendements espérés et de la matrice de covariance : Ces éléments permettent d'évaluer le rendement et le risque du portefeuille.
- Optimisation des poids des actifs : L'utilisation de la fonction minimize de scipy.optimize vise à trouver la combinaison optimale d'actifs minimisant le risque tout en maximisant le rendement ajusté au risque (Sharpe ratio).
- Simulation de portefeuilles aléatoires : En générant des milliers de portefeuilles, tu peux visualiser la frontière efficiente et comparer le portefeuille optimal aux autres combinaisons possibles.

Code

main.py

```python
import numpy as np
import pandas as yf
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)
```

Télécharger le code :

Où :

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

Prérequis

Python :

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

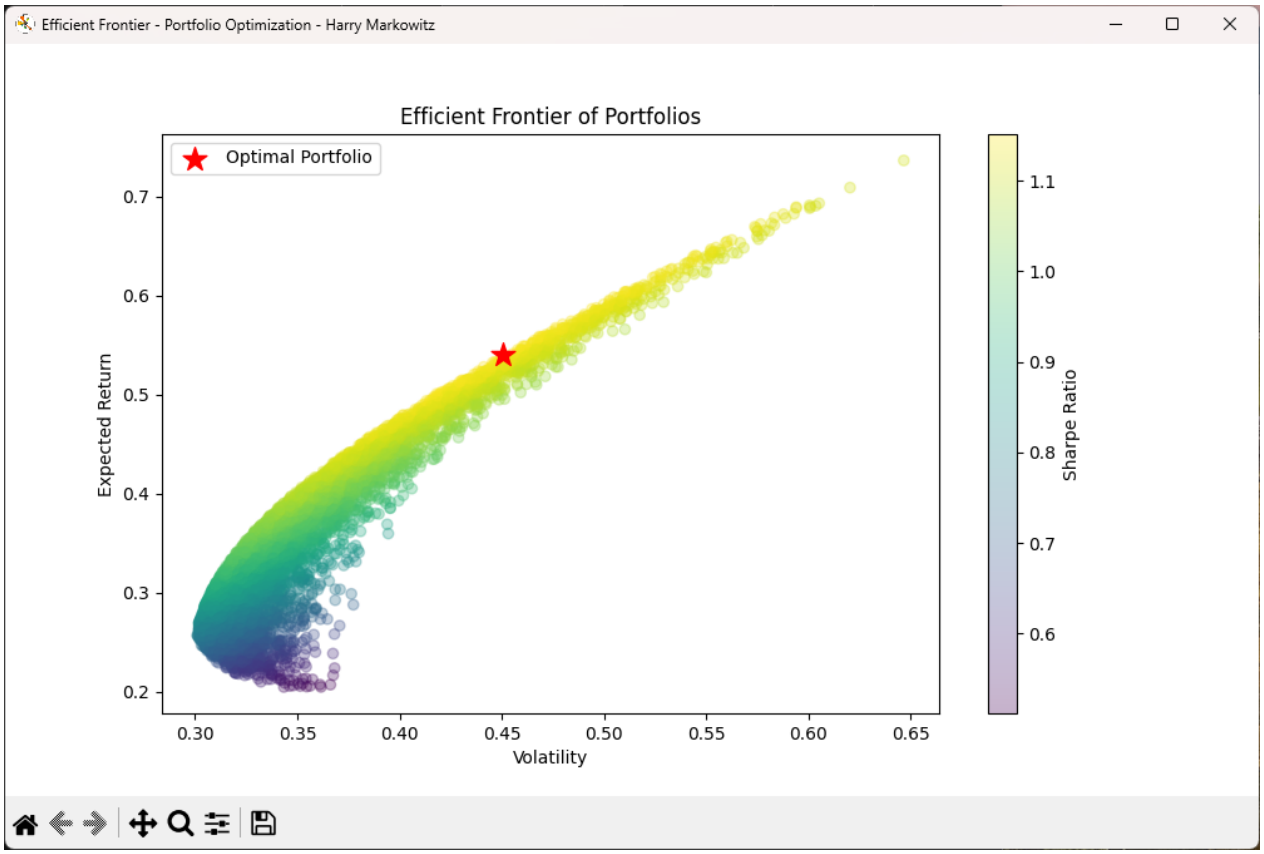
Librairies Python :

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

Étapes :

- Collecte de données : Récupérer les données historiques des prix des actifs du portefeuille.
- Rendements & Covariance : Calculer les rendements attendus et la matrice de covariance pour évaluer les mouvements des actifs.
- Optimisation : Utiliser un solveur (par exemple, scipy.optimize.minimize) pour maximiser les rendements tout en limitant le risque.
- Frontière efficiente : Tracer la frontière efficiente pour visualiser les portefeuilles optimaux équilibrant risque et rendement.

Résultat Final




```
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
```

Raisonnement mathématique et application à Python

Récupération des données et calcul des rendements

```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()
```

Nous récupérons les prix ajustés (Adj Close) des actions et calculons les rendements journaliers.

Rendement journalier :

Le rendement d'un actif financier entre deux jours est donné par la formule :

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

où :

- $r_{i,t}$ est le rendement de l'actif i au jour t,
- $P_{i,t}$ est le prix de clôture ajusté au jour t,
- $P_{i,t-1}$ est le prix de clôture ajusté au jour $t-1$.

Dans le code, on utilise

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

pour calculer cela automatiquement.

- data.pct_change() calcule automatiquement $R_{i,t}$ pour chaque jour et chaque actif.
- .dropna() supprime les valeurs manquantes qui apparaissent au premier jour (où aucun rendement ne peut être calculé).

Application :

Action / Date	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

Action / Date	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

On peut observer que à la clôture du 2020-01-**01** (la clôture est la première seconde du jour suivant), le prix de l'action AAPL avait un prix de 72.716080. Le prix à la clôture de l'action AAPL le 2020-01-**02** avait un prix de 72.009125.
On a donc :

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

Nous obtenons bien rendement de l'actif financier AAPL du 2020-01-01 au 2020-01-02 soit environ -0.9722%.

A noter : Puisque le premier jour le rendement de l'actif financier ne peux pas être calculé (N/A), on utilise donc .dropna() qui supprime ces valeurs.

Calcul des métriques financières

L'objectif est d'obtenir une estimation du rendement annuel moyen pour chaque actif.

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

Rendement moyen annualisé :

On calcule la moyenne des rendements journaliers :

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

où :

- $E(R_i)$ est le rendement espéré de l'actifs i. Il représente la moyenne des rendements sur les périodes observées,
- T est le nombre total de jours (la période),
- $r_{i,t}$ est le rendement journalier de l'actif financier i au jour t.

Puis on annualise en multipliant par 252 (nombre moyen de jours de bourse dans une année):

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

Cela correspond à `mean_returns` dans le code :

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

- `returns.mean()` donne la moyenne des rendements journaliers.
- On le multiplie par 252 pour obtenir une estimation du rendement annuel.

Application :

Étape / Action	$\sum_{t=1}^{T} r_{i,t}$	$E(R_i) = \frac{1}{T}\sum_{t=1}^{T} r_{i,t}$	$E(R_i) \times 252$
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

Ici, il faut bien retenir que dans cette application, nous observons ces actifs sur une période de 4 ans, donc le nombre de jours T correspond à 4 fois le nombre de jours boursier.

Matrice de covariance annualisée :

La covariance est une mesure statistique qui indique dans quelle mesure deux variables aléatoires (ici, les rendements des actifs R_i et R_j) varient ensemble. Si la covariance est positive, cela signifie que lorsque le rendement d'un actif augmente, le rendement de l'autre tend à augmenter aussi. Si elle est négative, lorsque le rendement d'un actif augmente, l'autre tend à diminuer. La covariance entre deux actifs i et j est donnée par :

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

où :

- $Cov(R_i, R_j)$ est la covariance entre les rendements des actifs i et j,
- $E(R_i)$ et $E(R_j)$ sont les rendements espérés des actifs i et j. Ils représentent la moyenne des rendements de chaque actif sur les périodes observées,
- T est le nombre total de jours,
- $r_{i,t}$ et $r_{i,t}$ sont les rendements journaliers des actifs i et j.

La variance entre deux actifs i et j est quant à elle donnée par :

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

Exemple : Calculer une covariance et une variance

Imaginons que nous observons les rendements de deux actifs (Actif A et Actif B) sur une période de 5 jours. Les rendements journaliers sont donnés ci-dessous :

Jour	Rendement de l'actif A ($r_{A,t}$)	Rendement de l'actif 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

Étape 1 : Calculer les rendements moyens (espérés) :

Nous devons d'abord calculer le rendement moyen de chaque actif :

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

Étape 2 : Calculer les écarts par rapport aux rendements moyens :

Ensuite, nous calculons les écarts entre les rendements journaliers et les rendements moyens pour chaque jour, pour les deux actifs.

Jour	$r_{A,t} - E(R_A)$	$r_{B,t} - E(R_B)$
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

Étape 3 : Calculer les produits des écarts :

Pour chaque jour, nous multiplions les écarts calculés pour l'actif A et l'actif B.

Jour	$\left(r_{A,t} - E(R_A)\right) * (r_{A,t} - E(R_A))$
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

Étape 4 : Calculer la somme des produits des écarts :

Nous additionnons maintenant les produits calculés :

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

Étape 5 : Calculer la covariance :

Enfin, nous utilisons la formule de la covariance :

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

Ici, T = 5 jours, donc $T - 1 = 4$.

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

Conclusion :

La covariance entre les rendements de l'actif A et de l'actif B est négative (−0.000025). Cela signifie que, globalement, les rendements des deux actifs ont tendance à varier de manière opposée : lorsque l'actif A augmente, l'actif B a tendance à diminuer, et vice versa. Cet exemple illustre comment la covariance est calculée et comment elle permet de comprendre la relation entre les variations des rendements de deux actifs dans un portefeuille.

A noter : On ne va pas le refaire ici mais si on veut calculer la variance de A par exemple, on utilise cette formule en se concentrant uniquement sur la l'actif financier A :

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

Une matrice de covariance est un tableau carré qui présente les variances des actifs le long de la diagonale principale et les covariances entre chaque paire d'actifs dans les autres cellules. Si nous avons n actifs dans un portefeuille, la matrice de covariance Σ aura la forme $n * n$, avec chaque élément représentant soit une variance (pour un actif individuel) soit une covariance (entre deux actifs).

Matrice de covariance pour n actifs :

Soit n actifs avec $R_{i,}R_{j,} \dots, R_{n,}$ représentant les rendements de chaque actif. La matrice de covariance Σ a la structure suivante :

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

On annualise la matrice de covariance en multipliant par 252 :

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

Dans le code,

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

- returns.cov() calcule la matrice de covariance des rendements quotidiens.
- On multiplie par 252 pour annualiser la covariance.

Optimisation du portefeuille (Maximisation du Ratio de Sharpe)

Une fois qu'on a les rendements moyens et la matrice de covariance, on peut évaluer la performance d'un portefeuille donné en fonction des poids des actifs.

Rendement espéré du portefeuille :

Le rendement d'un portefeuille $E(R_p)$ est la somme des rendements des actifs $E(R_i)$ pondérés par leurs poids w_i :

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

Dans le code,

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

où :

- weights est un vecteur contenant les allocations de chaque actif.

Volatilité du portefeuille (Risque total):

Le risque total d'un portefeuille est donné par la variance du portefeuille, qui est calculée comme suit :

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

où :

- σ_p^2 est la variance du portefeuille.

On prend ensuite la racine carrée pour obtenir la volatilité du portefeuille :

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

où :

- Σ est la matrice de covariance.

Dans le code,

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

calcule cette volatilité,

où :

- np.dot(weights.T, np.dot(cov_matrix, weights)) applique la formule $w^T \Sigma w$.
- np.sqrt() prend la racine carrée pour obtenir σ_p.

Ratio de Sharpe :

Le Ratio de Sharpe mesure la rentabilité ajustée au risque:

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

où :

- S est le ratio de Sharpe,
- $E(R_p)$ est le rendement espéré du portefeuille,
- R_f est le taux sans risque (fixé 2% dans mon code),
    - Le taux sans risque est un taux hypothèquement sans risque, dont la rentabilité est certaine sur une certaine période, comme le rendement du Trésor à un an ou à deux ans.
- σ_p est la volatilité du portefeuille.

Dans le code,

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

On minimise son opposé pour maximiser S :

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

Contraintes de l'optimisation

On impose les contraintes suivantes :

a) Somme des poids = 1 (le portefeuille est entièrement investi) :

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

Dans le code :

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

- {'type': 'eq'} : Cela signifie que cette contrainte est une contrainte d'égalité. En optimisation, les contraintes peuvent être de type 'eq' (égalité) ou 'ineq' (inégalité). Ici, le type 'eq' impose que la fonction donnée doit être égale à zéro.
- 'fun': lambda weights: np.sum(weights) – 1 :
    - lambda weights: est une fonction lambda qui prend weights (les poids du portefeuille ou les variables de décision) comme argument.
    - np.sum(weights) calcule la somme des poids.
    - np.sum(weights) - 1 indique que la somme des poids doit être égale à 1. C'est une contrainte courante dans les problèmes d'optimisation de portefeuille pour s'assurer que les proportions des actifs dans un portefeuille totalisent 100 %.

b) Les poids sont entre 0 et 1 (pas de vente à découvert) :

```
48    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) : Cela signifie que chaque poids doit être compris entre 0 et 1. Un poids de 0 indique que l'actif n'est pas inclus dans le portefeuille, tandis qu'un poids de 1 signifie que l'intégralité de l'investissement est dans cet actif.
    - for _ in range(num_assets) : Ceci est une boucle générant une paire de bornes (0, 1) pour chaque actif dans le portefeuille. Le nombre d'actifs est donné par la variable num_assets.

Problème d'optimisation d'Harry Markowitz

Formulation du problème d'optimisation :

L'objectif de l'optimisation de Markowitz est de trouver la combinaison optimale des actifs qui maximise le ratio de Sharpe :

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

où :

- $E(R_p) = \sum_{i=1}^{n} w_i E(R_i)$ est le rendement espéré du portefeuille,
- $\sigma_p = \sqrt{w^T \Sigma w}$ est la volatilité du portefeuille,
- R_f est le taux sans risque.

Formulation mathématique du problème :

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

sous la contrainte :

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

et les bornes :

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

Solution avec la méthode des multiplicateurs de Lagrange :

Pour résoudre ce problème, on utilise la méthode des multiplicateurs de Lagrange.

On définit la fonction de Lagrange :

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

où λ est le multiplicateur de Lagrange.

On dérive la fonction de Lagrange par rapport à w et λ :

- Première condition d'optimalité : Dérivée par rapport à w :

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

- Deuxième condition : contrainte d'allocation :

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

En remplaçant w_i :

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

Ce qui permet de trouver λ, puis de calculer les w_i.

Dans le code, on utilise la méthode SLSQP (Sequential Least Squares Programming) pour résoudre ce problème :

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

Portefeuille tangentiel et ratio de Sharpe :

Le portefeuille tangentiel est le portefeuille optimal lorsqu'un investisseur peut inclure un actif sans risque. Ce portefeuille est obtenu lorsque l'on maximise le ratio de Sharpe.

Le portefeuille tangentiel est celui qui est situé sur la frontière efficiente et tangent à la ligne de marché des capitaux (CML). Il maximise :

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

Calcul des poids optimaux du portefeuille tangentiel :

Les poids optimaux sont donnés par :

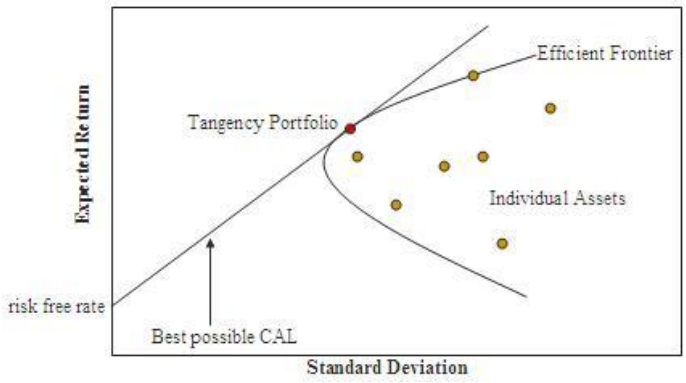
$$w^* = \frac{\Sigma^{-1}(E(R) - R_f 1)}{1^T \Sigma^{-1}(E(R) - R_f 1)}$$

où :

- Σ^{-1} est l'inverse de la matrice de covariance,
- $E(R)$ est le vecteur des rendements espérés,
- 1 est un vecteur de 1 de taille n.

Interprétation graphique :

Si l'on représente tous les portefeuilles possibles (nuage de points), la frontière efficiente est la courbe des meilleurs portefeuilles possibles. Le portefeuille tangentiel est celui qui touche cette frontière et a le meilleur ratio de Sharpe.



Dans le code, le vecteur de poids optimal w^* est obtenu par :

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

Simulation Monte Carlo (Frontière efficiente)

On génère 10 000 portefeuilles aléatoires en utilisant des poids w_i tirés de la distribution de Dirichlet :

La simulation de Monte Carlo est une technique qui permet d'explorer un grand nombre de portefeuilles aléatoires afin de visualiser la frontière efficiente et comparer ces portefeuilles au portefeuille optimal déterminé par l'optimisation de Markowitz.

Nous générons les poids aléatoires w_i pour chaque portefeuille en utilisant la distribution de Dirichlet. Cette distribution garantit que les poids :

- Sont positifs ($w_i \geq 0$),
- Somment à 1 $\sum_{i=1}^{n} w_i = 1,$ respectant ainsi la contrainte d'allocation totale

Voici la ligne correspondante dans le code :

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

Pour chaque portefeuille :

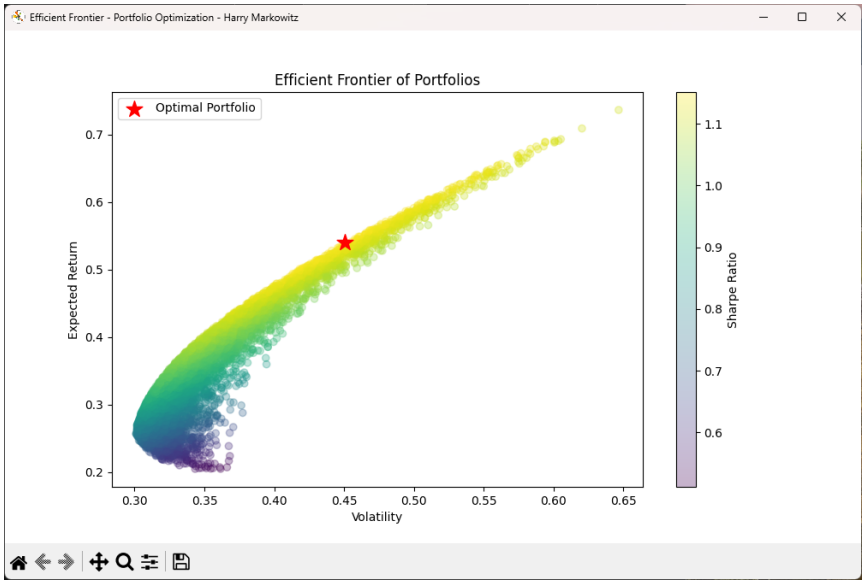
- On calcule le rendement attendu R_p,
- On calcule la volatilité σ_p,
- On déduit le Ratio de Sharpe.

Les résultats sont stockés et affichés sur un graphique Efficient Frontier (frontière efficiente).

Affichage des résultats

a) Graphique Efficient Frontier :

- Les points jaunes/verts/... représentent les portefeuilles aléatoires.
- L'étoile rouge représente le portefeuille optimal.



b) Affichage des poids optimaux dans un DataFrame :

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

Ce qui affiche quelque chose qui s'approche de cela :

	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 : *Représentation de la Frontière efficiente* – Wikipédia :

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

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

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

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

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

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

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

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

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