Seminar - Markowitz Portfolio Optimization

PUJOL Martin
RAMPONT Martin
THOMASSIN Pablo
STRIEBIG Maximilien

December 17, 2023

Theoretical Part

Problem Description

In the realm of financial portfolio management, the Markowitz portfolio optimization problem is a classical and essential topic. The primary objective is to allocate weights to different assets in a portfolio to maximize the expected return while minimizing the overall portfolio risk. Let's consider a portfolio with n assets. The goal is to find the optimal set of weights for these assets.

Formalization

An other formulation of the problem is to minimize the portfolio risk σ_p while achieving a target expected return μ:

The objective is to find the vector of weights $\mathbf{w} = [w_1, w_2, \ldots, w_n]$ that minimizes the portfolio risk σ_p while achieving a given expected portfolio return μ:

$$
\begin{aligned}
\text{Minimize} \quad & \sigma_p = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j \sigma_i \sigma_j \rho_{ij}} \quad \text{(Portfolio risk)} \\
\text{Subject to} \quad & \mu = \sum_{i=1}^{n} r_i w_i \quad \text{(Expected portfolio return)} \\
& \sum_{i=1}^{n} w_i = 1 \quad \text{(Sum of weights equals 1)} \\
& w_i \geq 0 \quad \text{(Non-negativity constraint)}
\end{aligned} \tag{1}
$$

In our problem we only want to minimize the portfolio risk σ_p.

After modifying the objective function to be unconstrained, we obtain the following problem formulation:

$$
\text{Minimize} \quad \sigma_p^2 = \frac{1}{(\sum_{k=1}^{n} e^{x_k})^2} \sum_{i=1}^{n}\sum_{j=1}^{n} e^{x_i} e^{x_j} \sigma_i \sigma_j \rho_{ij} \quad \text{(Portfolio risk)} \tag{2}
$$

With the variable change:

$$
w_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3}
$$

We also calculate the derivate of the objective function:

$$
\frac{\partial}{\partial x_n}\sigma_p^2 = \frac{-2e^{x_n}}{(\sum_{k=1}^{N} e^{x_k})^3} \sum_{i=1}^{N}\sum_{j=1}^{N} e^{x_i} e^{x_j} \sigma_i \sigma_j \rho_{ij} + 2\sigma_n e^{x_n} \sum_{j=1}^{N} e^{x_j} \sigma_j \rho_{nj} \tag{4}
$$

Numerical Part

Selected Optimization Methods

We have chosen to implement two different numerical optimization methods to solve the Markowitz portfolio optimization problem:

1. Method 1: [Fixed step : Gradient descent]
2. Method 2: [Variable step : Golden section search]

Algorithm Implementation

Below are the basic functions describing the two chosen algorithms:

Method 1: [Fixed step : Gradient descent]

The gradient step method can be formulated as follows:

$$x_{k+1} = x_k - \alpha \nabla f(x_k) \tag{5}$$

Where α_k is the step size and $\nabla f(x_k)$ is the gradient of the objective function at x_k.

We implemented in the following python code :

```
def descente_markowitz(x0,covariance_matrix,e,p, max_iter=10000):
    """
    Gradient descent algorithm to minimize the portfolio risk.

    Parameters:
        x0 (numpy array): Initial portfolio weights.
        covariance_matrix (numpy array): Covariance matrix of asset
        e (float): Tolerance for the stopping criterion.
        p (float): Step size.
        max_iter (int): Maximum number of iterations.

    Returns:
        numpy array: Optimal portfolio weights.
        int: Number of iterations.
        float: Portfolio risk.
    """
    k = 0
    ek = 2*e
    while ek>=e and k<max_iter:
        wk = -1*grad_f(x0,covariance_matrix)
        x0 = x0+p*wk
        ek = np.linalg.norm(p*wk)
        k+=1
    return np.exp(x0)/np.sum(np.exp(x0)),k,f(x0,covariance_matrix)
```

Method 2: [Variable step : Golden section search]

The golden section search method can be formulated as follows:

$$x_{k+1} = x_k + \alpha_k d_k \tag{6}$$

Where α_k is the step size and d_k is the search direction.
We implemented in the following python code :

```python
def descente2(x0,covariance_matrix,e,method='golden_section', max_i
    """
    Golden section algorithm to minimize the portfolio risk.

    Parameters:
        x0 (numpy array): Initial portfolio weights.
        covariance_matrix (numpy array): Covariance matrix of asset
        e (float): Tolerance for the stopping criterion.
        method (str): Line search method.
        max_iter (int): Maximum number of iterations.

    Returns:
        numpy array: Optimal portfolio weights.
        int: Number of iterations.
        float: Portfolio risk.
    """
    k = 0
    ek = 2*e
    while ek>=e and k<max_iter:
        wk = -1*grad_f(x0,covariance_matrix)
        p = pk(x0,covariance_matrix,wk,method)
        x0 = (x0+p*wk)
        ek = np.linalg.norm(p*wk)
        k+=1
    return np.exp(x0)/np.sum(np.exp(x0)),k,f(x0,covariance_matrix)

    def pk(x,covariance_matrix, wk, method='golden_section'):
    """
    Line search to find the optimal step size.
```

```
    Parameters:
        x (numpy array): Portfolio weights.
        covariance_matrix (numpy array): Covariance matrix of asset
        wk (numpy array): Negative gradient of the portfolio risk.
        method (str): Line search method.

    Returns:
        float: Optimal step size.
    """


    if method == 'golden_section':

        alpha = golden_section_line_search(x,covariance_matrix, wk)

    else:

        alpha = wolfe_conditions_line_search(x, wk)

    return alpha

def golden_section_line_search(x,covariance_matrix, wk, c1=1e-4, ma
    """
    Golden section line search to find the optimal step size.

    Parameters:
        x (numpy array): Portfolio weights.
        covariance_matrix (numpy array): Covariance matrix of asset
        wk (numpy array): Negative gradient of the portfolio risk.
        c1 (float): Parameter for the stopping criterion.
        max_iter (int): Maximum number of iterations.

        Returns:
        float: Optimal step size.
        """
    a = 0.0
    b = 1.0
    tau = 0.618
```

```python
    for _ in range(max_iter):
        alpha1 = a + (1 - tau) * (b - a)
        alpha2 = a + tau * (b - a)

        f1 = f(x+ alpha1* wk, covariance_matrix)
        f2 = f(x +alpha2* wk, covariance_matrix)

        if f2 > f1:
            b = alpha2
        else:
            a = alpha1

        if abs(alpha2 - alpha1) < c1:
            break

    return (alpha1 + alpha2) / 2.0
```

We have applied both methods to the Markowitz portfolio optimization problem and obtained the following results:

[Insert results, tables, or graphs]

Interpretation

[Provide interpretation of the results]

Comparison

To compare the two methods, we analyze factors such as computational time and the number of iterations:

[Insert comparison results]

Annexe : Objective function and constraints

Objective function

For solving the Markowitz portfolio optimization problem, we have chosen two numerical optimization methods:

I order to simplify the problem, we will first forget about the expected return constraint. We will only focus on minimizing the portfolio risk σ_p.

$$\begin{aligned}
\text{Minimize} \quad & \sigma_p = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j \sigma_i \sigma_j \rho_{ij}} \quad \text{(Portfolio risk)} \\
\text{Subject to} \quad & \sum_{i=1}^{n} w_i = 1 \quad \text{(Sum of weights equals 1)} \\
& w_i \geq 0 \quad \text{(Non-negativity constraint)}
\end{aligned} \tag{7}$$

To restruct the weight vector to be positive, we can use a variable change:

$$w_i = e^{x_i} \quad \text{with} \quad x_i \in \mathbb{R} \tag{8}$$

The problem becomes:

$$\begin{aligned}
\text{Minimize} \quad & \sigma_p = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} e^{x_i} e^{x_j} \sigma_i \sigma_j \rho_{ij}} \quad \text{(Portfolio risk)} \\
\text{Subject to} \quad & \sum_{i=1}^{n} e^{x_i} = 1 \quad \text{(Sum of weights equals 1)}
\end{aligned} \tag{9}$$

We can also forget about the square root in the objective function, as it does not change the optimal solution.

$$\begin{aligned}
\text{Minimize} \quad & \sigma_p^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} e^{x_i} e^{x_j} \sigma_i \sigma_j \rho_{ij} \quad \text{(Portfolio risk)} \\
\text{Subject to} \quad & \sum_{i=1}^{n} e^{x_i} = 1 \quad \text{(Sum of weights equals 1)}
\end{aligned} \tag{10}$$

Finally we can use the softmax function to ensure that the sum of the weights equals 1, such as:

$$w_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{11}$$

Leading to the following problem formulation (with softmax):

$$\text{Minimize} \quad \sigma_p^2 = \frac{1}{(\sum_{k=1}^{n} e^{x_k})^2} \sum_{i=1}^{n}\sum_{j=1}^{n} e^{x_i} e^{x_j} \sigma_i \sigma_j \rho_{ij} \quad \text{(Portfolio risk)} \tag{12}$$

Derivate of the objective function

Now, let's delve into the derivate of the objective function:

We are going to derive it term by term using the chain rule, we first derive the term outside the sum such as :

$$\frac{\partial}{\partial x_n}\frac{1}{(\sum_{k=1}^{N} e^{x_k})^2} = \frac{-2e^{x_n}}{(\sum_{k=1}^{N} e^{x_k})^3} \tag{13}$$

Where N is the number of assets in the portfolio and x_n is the variable we are deriving with respect to.

The second term is a bit more complicated, we will use the product rule:

$$\frac{\partial}{\partial x_n}\sum_{i=1}^{N}\sum_{j=1}^{N} e^{x_i}e^{x_j}\sigma_i\sigma_j\rho_{ij} = \sum_{i=1}^{N}\sum_{j=1}^{N}(\frac{\partial}{\partial x_n}e^{x_i})e^{x_j}\sigma_i\sigma_j\rho_{ij}+\sum_{i=1}^{N}\sum_{j=1}^{N} e^{x_i}(\frac{\partial}{\partial x_n}e^{x_j})\sigma_i\sigma_j\rho_{ij} \tag{14}$$

Because the two terms are similar, we will only focus on the first one: We can see that the derivate is not null if $i \neq n$:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}(\frac{\partial}{\partial x_n}e^{x_i})e^{x_j}\sigma_i\sigma_j\rho_{ij} = \sum_{j=1}^{N} e^{x_n}e^{x_j}\sigma_n\sigma_j\rho_{nj} = \sigma_n e^{x_n}\sum_{j=1}^{N} e^{x_j}\sigma_j\rho_{nj} \tag{15}$$

We can simplify the two sums by using the fact that $\rho_{ij} = \rho_{ji}$ and $\sigma_i\sigma_j = \sigma_j\sigma_i$:

$$\frac{\partial}{\partial x_n}\sum_{i=1}^{N}\sum_{j=1}^{N} e^{x_i}e^{x_j}\sigma_i\sigma_j\rho_{ij} = 2\sigma_n e^{x_n}\sum_{j=1}^{N} e^{x_j}\sigma_j\rho_{nj} \tag{16}$$

Finally, we can derive the whole objective function:

$$\frac{\partial}{\partial x_n}\sigma_p^2 = \frac{-2e^{x_n}}{(\sum_{k=1}^{N} e^{x_k})^3}\sum_{i=1}^{N}\sum_{j=1}^{N} e^{x_i}e^{x_j}\sigma_i\sigma_j\rho_{ij} + 2\sigma_n e^{x_n}\sum_{j=1}^{N} e^{x_j}\sigma_j\rho_{nj} \tag{17}$$