Imperial College London

Risk Management and Financial Engineering

Computational Finance with C++

Markowitz Portfolio Optimizer

Talha Jamal - 01357497

May 31, 2024

Contents

1. Introduction

1.1 Portfolio Optimization

The aim of this project was to solve a Portfolio Optimization problem, backtest the results of that optimizer over time, and evaluate the performance of the Portfolio Optimization Method chosen.

An Optimization Problem itself is about finding the best solution to a given problem from all feasible solutions under certain constraints. Equation shows the simplest form of a continuous portfolio optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned} \tag{1.1}$$

where

- $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function to be minimized over the n-variable vector x,
- $g_i(x) \leq 0$ and $h_j(x) = 0$ are called inequality constraints,
- $m \geq 0$ and $p \geq 0$,

Within Quantitative Finance, optimization methods have been researched and used extensively to optimize portfolios under various constraints. Portfolio Optimization involves selecting assets with certain weights out of all feasible combinations of assets and weights that satisfy certain constraints on the portfolio. These constraints can be arbitrary and vary as per the investor's investment criteria. The objective function typically maximizes the Expected Return of the Portfolio or Minimizes the Risk of the Portfolio. One of the most famous and oldest Portfolio Optimization Method is the Markowitz Portfolio Optimization method.

1.2 Markowitz Portfolio Optimization

The Markowitz Model aims to capture the tradeoffs between the risks and rewards of investments. The Model attempts to maximize Expected Return for a given level of risk. This model assumes that investors are risk-averse and prefer portfolios with higher expected returns for a given level of risk, the returns of a portfolio are normally distributed, and investors only consider mean and variance of returns when making investment decisions.

The Markowitz Mean-Variance Optimization method involves the following key equations:

1. **Expected Return of the Portfolio:** The Expected Return of the Portfolio μ_p is the weighted average pf the Expected Return μ_i of each individual asset in the portfolio.

$$\mu_p = \sum_{i=1}^{n} w_i \mu_i \tag{1.2}$$

2. **Variance of the Portfolio:** The Model assumes all of the risk of an asset is explained by the Variance of it's Assets. The Variance of the Portfolio σ_p^2 is given by:

$$\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_{ij} \tag{1.3}$$

where σ_{ij} is the covariance between returns of asset i and j.

3. **Risk-Return Optimization:** The goal of Mean Variance Optimization here is to find a vector of weights that minimize the portfolio variance for a given level of returns subject to all the weights of individual assets in the portfolio summing up to 1 and an expected return constraint. Here $\boldsymbol{w}$ is the vector of portfolio weights, $\boldsymbol{\Sigma}$ is the covariance matrix of asset returns, and $\boldsymbol{\mu}$ is the vector of expected returns of the assets.

$$\min_{w} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_i \sigma_{ij} w_j = \frac{1}{2} \sigma_P^2 \tag{1.4}$$

subject to:

$$\sum_{i=1}^{n} w_i \bar{r}_i = \bar{r}_P \tag{1.5}$$

$$\sum_{i=1}^{n} w_i = 1 \tag{1.6}$$

4. **Lagrange Multipliers:** In Mathematical Optimization, Lagrange Multipliers is a technique to solve optimization problems involving finding a local minima or maxima subject to certain constraints. We use the Lagrangian Multiplier method as below to find the associate Lagrangian Function to minimize the Loss Function given the constraints (1.6) and (1.5):

$$L = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_i \sigma_{ij} w_j - \lambda \left(\sum_{i=1}^{n} w_i \bar{r}_i - \bar{r}_P \right) - \mu \left(\sum_{i=1}^{n} w_i - 1 \right) \tag{1.7}$$

where λ and μ are the Lagrangian Multipliers.

5. **Optimal Weights:** Taking the partial derivatives with respect to $\boldsymbol{w}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\mu}$ and equating them to 0 allows us to solve for the ideal portfolio weights. In matrix notation form this can be written as below:

$$\Sigma \mathbf{w} - \lambda \bar{r} - \mu \mathbf{e} = 0, \quad \bar{r}^T \mathbf{w} = \bar{r}_P \quad \text{and} \quad \mathbf{e}^T \mathbf{w} = 1 \tag{1.8}$$

$$\begin{pmatrix} \Sigma & -\bar{r} & -\mathbf{e} \\ -\bar{r}^T & 0 & 0 \\ -\mathbf{e}^T & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} 0 \\ -\bar{r}_P \\ -1 \end{pmatrix}. \tag{1.9}$$

This is solvable if Σ has full rank and $\bar{r}$ is not a multiple of $\mathbf{e}$.

$$\Rightarrow \begin{pmatrix} \mathbf{w} \\ \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} \Sigma & -\bar{r} & -\mathbf{e} \\ -\bar{r}^T & 0 & 0 \\ -\mathbf{e}^T & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ -\bar{r}_P \\ -1 \end{pmatrix}. \tag{1.10}$$

Solving this system yields the optimal portfolio weights $\mathbf{w}$.

1.3 Data

The dataset for this project involved 83 Assets from the FTSE 100 listed on the London Stock Exchange. The dataset included daily returns for each asset for 700 days. Dataset was already preprocessed hence the project did not involve any data wrangling.

2. Software Structure

2.1 Backtesting Procedure

This project's main task was using a Mean Variance Optimization Framework to create Portfolios and backtest these results over time. With 700 days of return data available, a sliding window backtesting procedure was chosen. For a certain target return, 100 Days of *In Sample* data were used to solve the optimization problem to produce weights for our portfolio, then a portfolio with aforementioned weights was held for 12 days *Out Of Sample* without rebalancing the portfolio. Then, for the same target return, the *In Sample Period* and *Out Of Sample Period* was shifted 12 days forward. This is illustrated in the Figure 2.1 below.

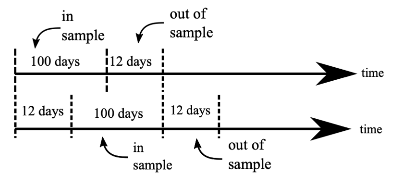

Figure 2.1: Sliding Window Technique

2.2 Object Oriented Design

The project was designed with an object oriented approach with multiple classes and functions handling separate tasks. The multiple classes used can be understood best in the Figure

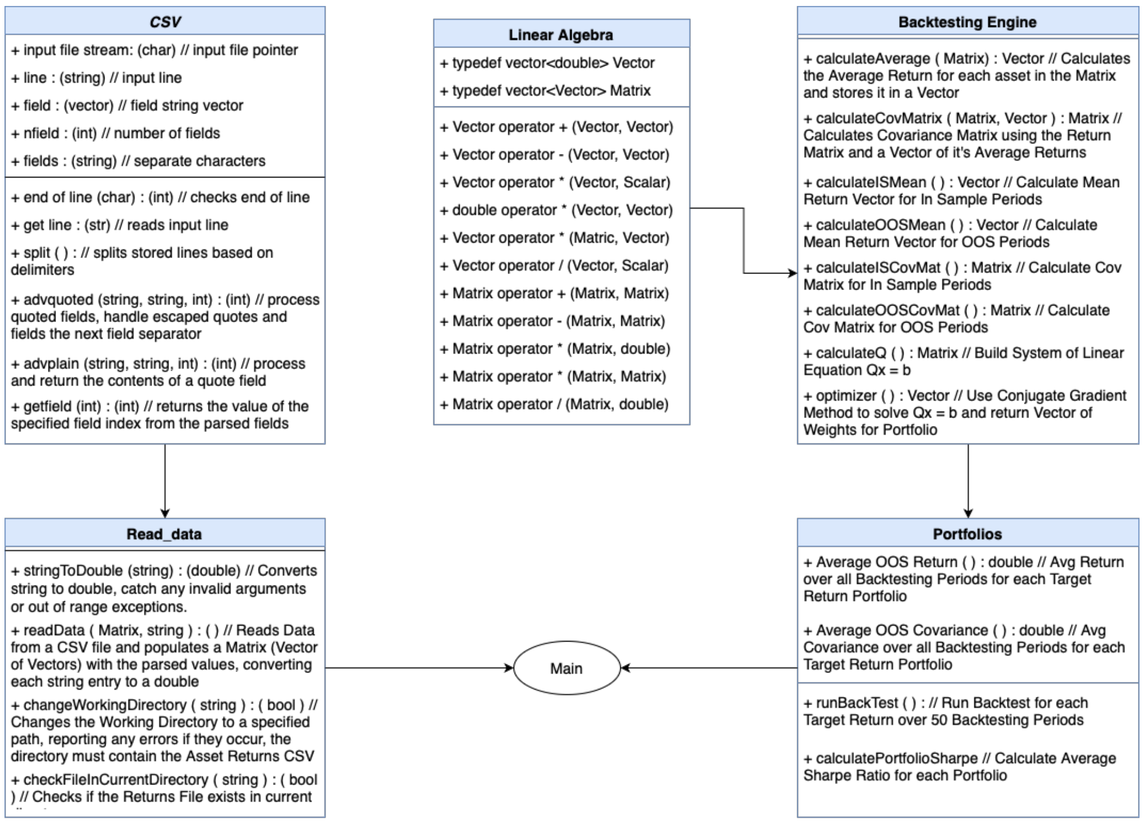
CSV
+ input file stream: (char) // input file pointer
+ line : (string) // input line
+ field : (vector) // field string vector
+ nfield : (int) // number of fields
+ fields : (string) // separate characters
+ end of line (char) : (int) // checks end of line
+ get line : (str) // reads input line
+ split ( ) : // splits stored lines based on delimiters
+ advquoted (string, string, int) : (int) // process quoted fields, handle escaped quotes and fields the next field separator
+ advplain (string, string, int) : (int) // process and return the contents of a quote field
+ getfield (int) : (int) // returns the value of the specified field index from the parsed fields
Linear Algebra
+ typedef vector<double> Vector
+ typedef vector<Vector> Matrix
+ Vector operator + (Vector, Vector)
+ Vector operator - (Vector, Vector)
+ Vector operator * (Vector, Scalar)
+ double operator * (Vector, Vector)
+ Vector operator * (Matric, Vector)
+ Vector operator / (Vector, Scalar)
+ Matrix operator + (Matrix, Matrix)
+ Matrix operator - (Matrix, Matrix)
+ Matrix operator * (Matrix, double)
+ Matrix operator * (Matrix, Matrix)
+ Matrix operator / (Matrix, double)
Backtesting Engine
+ calculateAverage (Matrix) : Vector // Calculates the Average Return for each asset in the Matrix and stores it in a Vector
+ calculateCovMatrix (Matrix, Vector) : Matrix // Calculates Covariance Matrix using the Return Matrix and a Vector of it's Average Returns
+ calculateISMean ( ) : Vector // Calculate Mean Return Vector for In Sample Periods
+ calculateOOSMean ( ) : Vector // Calculate Mean Return Vector for OOS Periods
+ calculateISCovMat ( ) : Matrix // Calculate Cov Matrix for In Sample Periods
+ calculateOOSCovMat ( ) : Matrix // Calculate Cov Matrix for OOS Periods
+ calculateQ ( ) : Matrix // Build System of Linear Equation Qx = b
+ optimizer ( ) : Vector // Use Conjugate Gradient Method to solve Qx = b and return Vector of Weights for Portfolio
Read_data
+ stringToDouble (string) : (double) // Converts string to double, catch any invalid arguments or out of range exceptions.
+ readData (Matrix, string) : ( ) // Reads Data from a CSV file and populates a Matrix (Vector of Vectors) with the parsed values, converting each string entry to a double
+ changeWorkingDirectory (string) : (bool) // Changes the Working Directory to a specified path, reporting any errors if they occur, the directory must contain the Asset Returns CSV
+ checkFileInCurrentDirectory (string) : (bool) // Checks if the Returns File exists in current
Main
Portfolios
+ Average OOS Return ( ) : double // Avg Return over all Backtesting Periods for each Target Return Portfolio
+ Average OOS Covariance ( ) : double // Avg Covariance over all Backtesting Periods for each Target Return Portfolio
+ runBackTest ( ) : // Run Backtest for each Target Return over 50 Backtesting Periods
+ calculatePortfolioSharpe // Calculate Average Sharpe Ratio for each Portfolio

Figure 2.2: Multi Class Software Structure

2.3 Reading Data

The Dataset for FTSE Returns is provided in a csv file. As there is not package to directly open csv files in C++, a CSV class and helper functions in a *read_ data.cpp* were needed. The *CSV* class reads and parses the input file line by line and stores the data into a Matrix. The matrix here takes the form of a 2D Vector using a standard C++ library. The *CSV* class provides methods to read the file line by line, check for end of line characters, and split the line into fields while handling quoted and unquoted data. The *read_ data* helper functions open the file and utilize the aforementioned methods of the *CSV* class to read each line and convert each field from a string to a double using the *stringToDouble* function. Safety methods *changeWorkingDirectory* and *checkFileInCurrentDirectory* are included to verify whether the Asset Returns File is present in the current working directory or not, and change the directory to the correct one if needed.

2.4 Linear Algebra

As mentioned before in section 1.2 equation 1.10, the portfolio weights are obtained by solving a linear systems of equation. For this, we need to overload existing C++ operators with Matrix and Vector Linear Algebra functions that will allow us to solve the linear system of equations shown previously. The Matrix and Vector operators showing in Figure 2.2 inside the Linear Algebra Class display the various linear algebra methods created. Overloading the standard addition, subtraction, division and multiplication operators within C++ allows ease of use of these linear algebra methods when writing C++ code to solve mathematical equations. C++ is able to recognise which operator method to apply on the variables based on their type. The use of these methods is expanded further on in section 2.5 and 2.6.

2.5 Backtesting Engine

As shown in Figure 2.1, the backtesting procedure involves creating 50 portfolios per target return by solving equation 1.10. One method to solve the system of linear equation is called a *Conjugate Gradient Method*. This is described further in section 2.6. We generate 20 target returns from 0.5% to 10%, run 50 portfolios for each target return - one portfolio for each out of sample period - and use all 50 to calculate an average expected return $\bar{\mathbf{r}}\mathbf{w}$ of the Portfolio and $\mathbf{w}^T\Sigma\mathbf{w}^T$ its covariance matrix.

The methods used within the Backtesting Engine are described below:

1. **calculateAverage:** This function takes in a matrix and returns the average across the columns. For our data, the return matrix is a 700 x 83 matrix with 700 days of return data and 83 assets. Over here this function then returns the average return for each asset across 700 days if the entire matrix is given to it as an input.

$$\bar{r}_i = \frac{1}{n}\sum_{k=1}^{n} r_{i,k} \tag{2.1}$$

2. **calculateCovMatrix:** This function calculates a covariance matrix for the return matrix provided. The function takes the return matrix as an input and a vector of the mean returns of each asset in the number of periods available within the return matrix. The formula for this Covariance Matrix is shown below:

$$\Sigma_{ij} = \frac{1}{n-1}\sum_{k=1}^{n}(r_{i,k} - \bar{r}_i)(r_{j,k} - \bar{r}_j). \tag{2.2}$$

3. **calculateISMean:** For each in sample period, a matrix of returns for that specific period is created. For example, for the first in sample period, a matrix of returns fro day 1 to day

100 is created. For the second in sample period, a matrix holding returns from day 12 to day 112 is created, and so on. The calculateAverage function is called to calculate a vector of mean returns for the in sample matrix of returns.

4. **calculateISCovMat:** Similarly, for each in sample period, an 83 x 83 covariance matrix is created using the calculateCovMatrix and vector of mean returns calculated previously.

5. **calculateOOSMean:** For each out of sample period, a matrix of returns for that specific period is created. For example, the first out of sample period is from day 100 to day 112 (12 day window), a matrix holding returns for those days is created. For the second period, a matrix holding returns for day 112 - 124 is created. For each of those matrices, the calculateAverage function is called to create a vector of mean returns in the out of sample period. The weights of the portfolio remain constant throughout the out of sample period.

6. **calculateOOSCovMat:** Similarly, the calculateCovMatrix is called upon for the return matrix in each out of sample period to create an 83 x 83 covariance matrix.

7. **calculateQ:** The Linear System of Equations is solved via a method called Conjugate Gradient Method. This method is used to solve equations taking the following form:

   $$\mathbf{Q} * \mathbf{x} = \mathbf{b} \tag{2.3}$$

   Equation 1.9 in Section 1.2 represents the above equation 2.3 in Matrix Form as follows:

   (a) Q is a (n + 2) x (n + 2) matrix with the top left values being the covariance matrix.

   (b) n represents the number of Assets - 83 in this scenario.

   (c) $\mathbf{\bar{r}_i}$ represents a vector of mean returns.

   (d) $-\mathbf{e}$ represents a vector of 1s.

   (e) The bottom right 4 values of the Matrix Q are 0s.

   (f) The Vector x is a vector with (n + 2) rows, with the first n values being the weights of each asset in the portfolio, and the final two values being the Lagrange multipliers.

   (g) The Vector b is a vector with (n + 2) rows, the first n values being 0s, the final two values being the negative of the target portfolio return and -1, respectively.

   This function essentially creates the whole linear systems of equation by filling in the values of Q, x, and b with their relevant sizes for each in sample backtesting period.

8. **optimizer:**

   As mentioned earlier, the portfolio optimizer utilizes the Conjugate Gradient Method to solve the system of linear equation. The function *optimizer* implements this method. It utilizes the linear algebra functions created in section 2.4

The pseudo-code for the entire program is written below:

Algorithm 1 Mean Variance Portfolio Optimization

```
Require: Number of Assets $n = 83$, Number of Returns $m = 700$
 1: Initialize a Vector of Target Portfolio Returns ranging from 0.005 to 0.1
 2: Set backtesting parameters:
      • $\epsilon = 10^{-7}$
      • in-sample window $w = 100$
      • out-of-sample window $o = 12$
      • sliding window $s = 12$
 3: Calculate number of sliding windows: $n\_sliding = 1 + \frac{m-w-o}{s} = 50$
 4: Store Asset Returns Data from CSV File into a Matrix
 5: for each Target Return do
 6:     Initialize a Portfolio Object with parameters $w$, $o$, $s$, $returnMatrix$, $TargetReturns[i]$
 7:     for all In Sample Periods do
          • Calculate in-sample mean using portfolio.calculateIsMean()
          • Calculate in-sample covariance matrix using portfolio.calculateIsCovMat()
 8:     end for
 9:     for all Out of Sample Periods do
          • Calculate out-of-sample mean using portfolio.calculateOOSMean()
          • Calculate out-of-sample covariance matrix using portfolio.calculateOOSCovMatrix()
10:     end for
11:     for all In Sample Periods do
          • Calculate matrix $Q$ using portfolio.calculateQ()
          • Optimize portfolio weights using portfolio.optimizer($\epsilon$)
12:     end for
13:     Run backtest using portfolio.runBacktest() and calculate portfolio measures such as:
          • Average Return for all backtesting periods per Target Return
          • Average Covariance for all backtesting periods per Target Return
14: end for
15: A Python Script is then used to calculate the Average Return, Average Covariance, and Average Sharpe Ratio for each Target Return Portfolio. The Python Script also creates some visualization images to display the portfolio's performance.
```

2.6 Conjugate Gradient Method

The Conjugate Gradient Method is used to solve linear equations taking the form $\mathbf{Q} * \mathbf{x} = \mathbf{b}$ if Q is a symmetric positive definite matrix. It is useful for large-scale problems like portfolio optimization because it converges to the solution faster than traditional methods like Gaussian Elimination or LU decomposition. Here are it's advantages:

1. **Efficient:** Converges in at most n iterations for n dimensional problems.
2. **Memory Usage:** Only requires storing a few vectors or matrices, making it suitable for large scale problems.

3. **No need for Matrix Inversion:** the algorithm directly works with Matrix-Vector products and avoids computationally expensive matrix inversion.

The algorithm is summarized below:

Algorithm 2 Conjugate Gradient Method – Quadratic Programming

```
Require: Initial point $x_0$, matrix $Q$, right-hand side vector $b$, and solution tolerance $\epsilon$.
1: Initialize:
     • Compute the initial residual: $s_0 = b - Qx_0$
     • Set the Initial Direction: $p_0 = s_0$
2: for $k = 0, 1, \ldots,$ until $s_k^T s_k \leq \epsilon$ do
     • Compute the step size: $\alpha_k = \frac{s_k^T s_k}{p_k^T Q p_k}$
     • Update the solution: $x_{k+1} = x_k + \alpha_k p_k$
     • Update the residual: $s_{k+1} = s_k - \alpha_k Q p_k$
     • Compute the conjugate direction coefficient: $\beta_k = \frac{s_{k+1}^T s_{k+1}}{s_k^T s_k}$
     • Update the search direction: $p_{k+1} = s_{k+1} + \beta_k p_k$
3: end for
4: The Algorithm iterates until the residual is sufficiently small (smaller than the predefined tolerance i.e. epsilon).
```

We validate the result from the Conjugate Gradient Method by using its solution vector x to calculate the residual vector $Qx - b$ and its norm, dividing it by the norm of b, and then comparing the relative residual to a specified tolerance. If the relative residual is within the tolerance, the solution is considered valid. This *ValidateConjugateGradientMethod* function is better explained in the algorithm below:

Algorithm 3 Validate Results from Conjugate Gradient Method

```
Require: Matrix $Q$, Vector $x$, Vector $b$, Tolerance $\epsilon = 10^{-3}$
Ensure: Boolean indicating if the solution is valid
1: Compute: $Qx \leftarrow Q \times x$
2: Compute residual vector: $r \leftarrow Qx - b$
3: Compute residual norm: residualNorm $\leftarrow \|r\|$
4: Compute norm of $b$: bNorm $\leftarrow \|b\|$
5: Compute relative residual relativeResidual $\leftarrow \frac{\text{residualNorm}}{\text{bNorm}+10^{-10}}$
6: If relativeResidual $\leq \epsilon$ Then Return True Else Return False
```

The Relative Residual provides a measure of accuracy, stability of our solution, and a convergence criterion for our solution compared to the true solution of the system of linear equation. A smaller residual fit norm indicates a closer fit between Qx and b. Taking the relative residual gives us a scale independent measure of the residual and is easier to interpret across problems with varying magnitudes of b.

2.7 Portfolios

This class inherits all the data items and attributes of the Backtesting Engine class. It is used to calculate portfolio metrics for each target return.

1. Once the weights are calculated for an In Sample Period, the Portfolio for the next out of sample period has those weights fixed for the assets. Therefore, the return of a Portfolio in the out of sample period can be calculated as:

$$\bar{r}^T w \tag{2.4}$$

   As there are 50 out of sample backtesting periods, the actual average return for a target return is the average of the portfolio returns amongst all k periods:

$$\bar{r}_{portfolio} = \frac{1}{50} \sum_{k=1}^{50} \bar{r}_k^T w_k \tag{2.5}$$

2. Similarly, the actual average covariance of the portfolio is also calculated as the average covariance across all k out of sample periods:

$$w^T \Sigma w = \frac{1}{50} \sum_{k=1}^{50} w_k^T \Sigma w \tag{2.6}$$

3. Finally, it calculates the average Sharpe Ratio of the portfolios for each target return i over all k out of sample periods:

$$\bar{SR}_i = \frac{\bar{r}_{portfolio,k} - r_f}{\sigma_{portfolioreturns,k}} \tag{2.7}$$

   Where r_f is the risk free rate and $\sigma_{portfolioreturns,k}$ is the standard deviation of portfolio returns in out of sample period k. The risk free rate is assumed to be 0 here.

3. Evaluation

3.1 Results

Three portfolio metrics were calculated for each target return portfolio: average return (2.5), average covariance (2.6), and average Sharpe Ratio (2.7). These are shown in the table 3.1 below:

Target	Average Return	Percentage Error	Average Covariance	Average Sharpe Ratio
0.005	0.002403	0.476613	0.000211	0.454729
0.010	0.003339	0.652182	0.000371	0.453499
0.015	0.004012	0.697819	0.000562	0.403816
0.020	0.004424	0.715774	0.000779	0.350200
0.025	0.005071	0.760646	0.001003	0.312114
0.030	0.005756	0.777110	0.001238	0.288082
0.035	0.006705	0.775011	0.001498	0.277079
0.040	0.007414	0.767489	0.001774	0.248068
0.045	0.008712	0.788664	0.002078	0.252751
0.050	0.010353	0.789730	0.002385	0.265882
0.055	0.011340	0.793255	0.002736	0.256791
0.060	0.011697	0.801057	0.003082	0.239769
0.065	0.011662	0.800731	0.003454	0.213200
0.070	0.013581	0.814491	0.003833	0.225845
0.075	0.013499	0.815848	0.004200	0.209402
0.080	0.014249	0.828315	0.004641	0.205913
0.085	0.015469	0.822883	0.005087	0.206878
0.090	0.016294	0.819021	0.005549	0.204630
0.095	0.016117	0.830432	0.006038	0.190250
0.100	0.017043	0.822452	0.006558	0.190816

Table 3.1: Performance Metrics for Different Target Returns

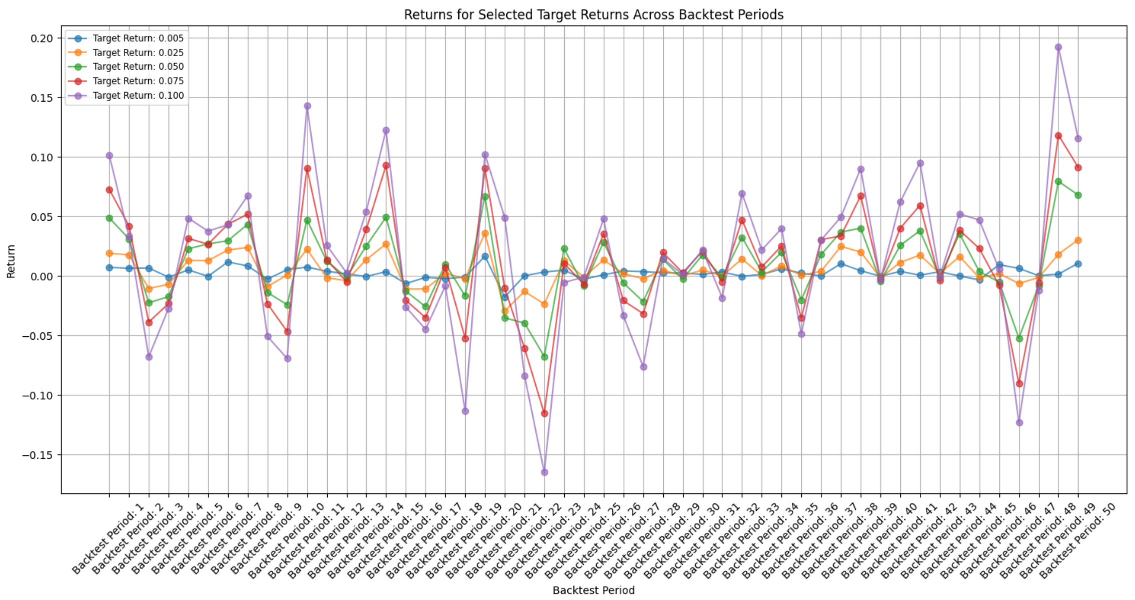

Figure 3.1: Returns for each Backtesting Period

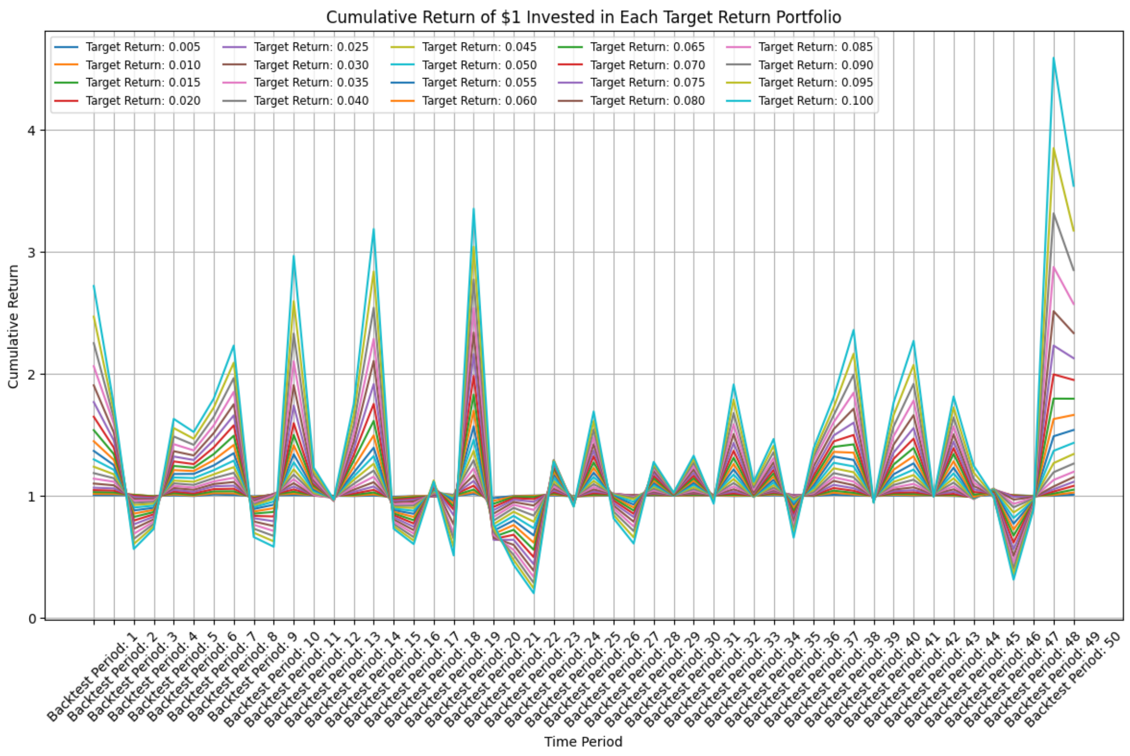

Figure 3.2: Cumulative Return for Target Portfolios

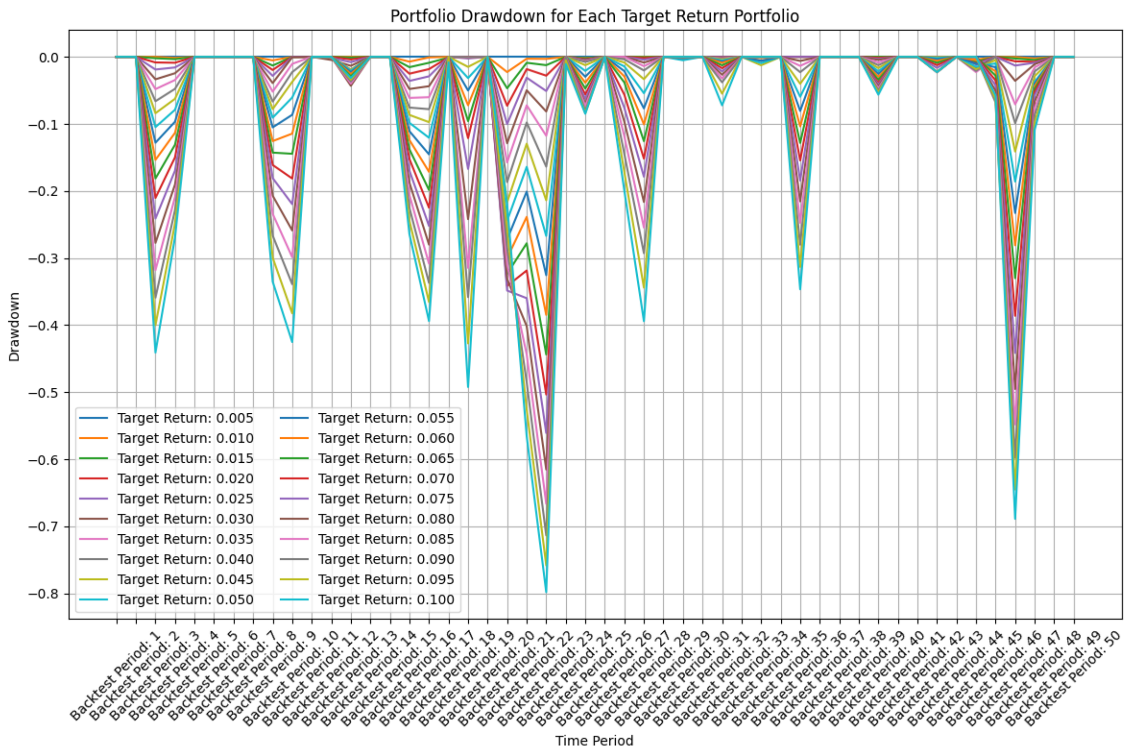

Figure 3.3: Target Portfolio Drawdowns

Figure 3.1 shows the fluctuating out of sample returns for 5 Target Returns. The portfolios targeting a higher return tend to have higher volatility. Figure 3.2 and 3.3 show the cumulative returns and portfolio drawdown for all target returns, respectively.

3.2 Analysis

Evidently, the Mean Variance Optimization method does not perform well out of sample when the portfolio is held constant for the 12 days of each backtesting period. The percentage error between the Target Return and the Actual Average Return tends to increase as the Target Return increases. This can occur due to the below reasons:

1. **Limitations of Historical Data:** The Mean Variance Optimization framework uses historical data for estimates of mean returns, variances, and co-variances. The distribution of returns and the level of volatility and correlation of assets can possible change in the out of sample period. This could mean that the sample estimates from the previous 100 days may not be an accurate reflection of the underlying distribution of the returns over the next 12 days.

2. **Input Window Sensitivity:** The sample estimates of mean, variance, and covariances can be extremely sensitive to the in sample window chosen and can often lead to overfitting.

3. **Model Assumption - Normally Distributed Returns:** One of the fundamental assumptions of the Mean Variance framework is that returns are normally distributed. However, financial returns tend to exhibit fat tails and skewness - significant deviations from a normal distribution as shown by the negatively skewed return and fat tailed distribution here in Figure 3.4 below.

4. **Model Assumption - Investors are Rational:** The framework assumes that investors stick to a risk-return optimization framework. However, in real life this may not be the case and the behavioural biases of investors may skew their risk tolerance and thus a mean variance framework would not be appropriate to build a portfolio.

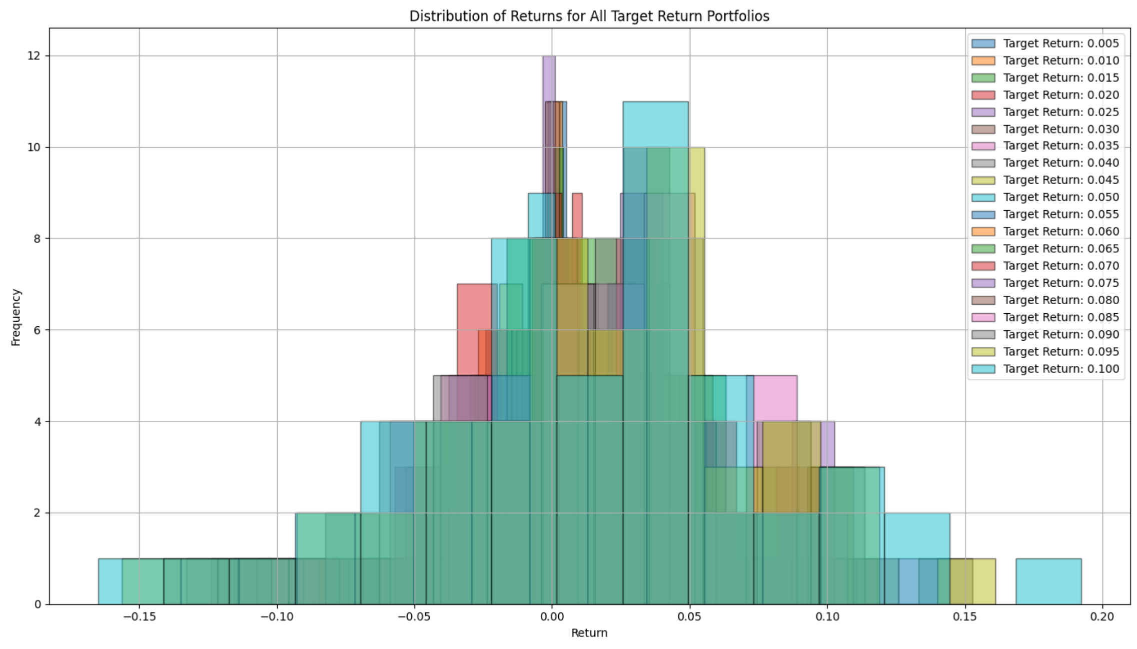

Figure 3.4: Distribution of Returns

3.3 Conclusion

To conclude, it can be shown that whilst the Mean Variance Optimization framework works great in a world where it's assumptions are true and the in sample distribution of returns reflects the out of sample distribution of returns. However, as the results in 3.1 show, the optimizer does not perform well out of sample, fails to achieve the target return, has significant drawdown periods and is not a high Sharpe Ratio strategy available for investors to trade - this is without even including an transaction costs in our model. The Mean Variance Optimization model does, however, provide a framework portfolio construction that can be built upon further. By taking a look at simple return metrics such as mean returns and their variance, it does show how an investor optimizing for their risk and return attributes may allocate their capital in a theoretical setting.