Sharp(e) Selection Process

Mean Variance Sharpe Ratio Analysis

Ariel Sosnovsky
Victoria Tran
Kosal Chhin
Yosef Bisk

December 6, 2015

Contents

1 Summary

1.1 What Does Our Program Do?

Our program allows the user to find the most optimal portfolio for a given time period using the mean-variance portfolio (MVP) optimization method. This program selects a portfolio of stocks from the global market that has minimum variance, as well as a maximized Sharpe ratio.

1.2 How Does The Program Do It?

The program begins by partitioning the periods of a given time into smaller intervals. It then uses Yahoo Finance to download daily stock prices of over $30,000$ stocks traded on markets around the world. The stock data is then partitioned by industry and year with respect to indicated period, after which stocks that are missing more than 30% of the data are removed from a given partition. Next the program enters the filtering stage, where the existing set of stocks is divided into pairs of stocks that have the most negative correlation with each other, and then selects the top 25% of pairs. Within that subset, the program selects the individual stocks with the top 25% lowest variance. The portfolio now enters the selection stage.

The selection stage is divided into two parts, which repeat as needed. The first part, addition part, we provide the program with a set of indexes of the stocks within the market. If that set is empty then the stock with the lowest variance is added initially to the portfolio. Then the selection algorithm begins in which stocks are added to the portfolio only if they increase the Sharpe Ratio of the portfolio more than any other stock, when the weights are optimized using the Mean variance optimization method. Once a particular limit of stocks is reached, the algorithm ends.

In the second part of the selection stage, the reduction part, the program will be provided with a set of indexes of stocks. The program then will create every possible combination of portfolios with one less asset in it, than originally provided. Then the portfolio with the highest Sharpe ratio is kept.

This stage will alternate between addition and reduction, until a convergence will occur. The resulting portfolio should be the minimum variance portfolio that offers the highest Sharpe ratio.

2 Introduction

2.1 Objectives and Motivation

The goal of this paper is to test out how the classical mean variance portfolio (MVP) model with respect to the Sharpe ratio performs, when applied to a large dataset. We start by developing a method of minimizing the variance of a portfolio with respect to the mean while searching for the maximum Sharpe ratio of a portfolio. Then, we test the method to a large data set and investigate the weaknesses of the model. The motivation for this is to investigate the model and its application and show how the results of the model stack up against criticism. We develop a selection purpose as a way to simulate real world application of the model. This sort of selection will theorectically reduce the transaction costs that one may incur while following a classical MVP approach.

2.2 Background

2.2.1 Basics of Mean-Variance Portfolio Theory

Mean variance optimization is a theory developed by Henry Markowitz in 1952, for portfolio optimization that has been standard for creating efficient asset allocation strategies for more than half a century. The theory aims at finding the optimal set of weights that achieves a desired expected return, μ_p from the portfolio with minimal risk σ_p^2. It does so, by defining a portfolio as a weighted average of market assets, $P = \sum_{j=1}^{n} w_j X_j$, and then uses methods of calculus to approximate the values of w_j such that the variance function, $\sigma_p^2 = \sum_{j=1}^{n} \sum_{i=1}^{n} w_j w_i Cov[X_i, X_j]$ is minimized with respect to a pre-set value μ_p, where $\mu_p = \sum_{i=1}^{n} E[X_i]$. It has also been widely criticized by many statisticians and economics for some of the assumptions that it makes on the mean and volatility of assets. As will be shown in this paper, the underlying issues that the assumptions of the model bring, become visible when the model is applied to a complete market. However, these problems become less severe as we reduce the intervals of time and increase the number of optimization instances.

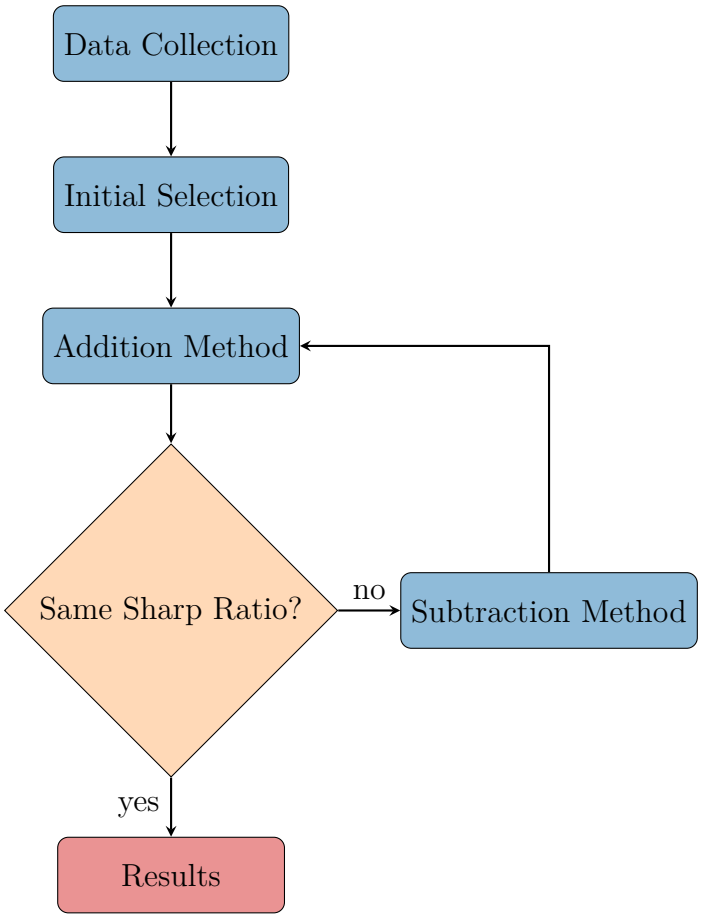
2.2.2 The Sharpe Ratio

The Sharpe Ratio, developed in 1966 by William F. Sharpe, is a method of relating the portfolio return relatively to its risk. It is defined as: $Sharpe = \frac{\mu_p - r_f}{\sqrt{\sigma_p^2}}$, where r_f is the risk free interest rate given by the market or a government. This is a measure for calculating risk-adjusted return, that is, the expected return earned in excess of the risk free rate per unit of standard deviation. In other words, for the amount of risk we are taking, how much expected return are we getting out of it. But one of the problems with the Sharpe Ratio is that it treats all volatility the same, and this can result in penalizing upside volatility (potential of very high returns).

3 Methodology

3.1 Overview

The way we constructed our algorithm can be summarized in the following flow chart:



The following chapters will elaborate on each step of this chart.

3.2 Data

For this project, we decided to use real life financial market data aggregated from the Yahoo Finance website. It provides a more interesting result and tests the accuracy of the algorithm. We begin by describing the conditions imposed on the data before collection, our data-mining algorithm, and lastly what methods were used to deal with missing data.

3.2.1 Collection

Below is a list of criteria and their justification chosen to fulfill the project requirements.

1. Years 2008-2012
    - the stock data has daily open and closing prices from Jan 2008 to Dec 2012
    - the time period was chosen to encapsulate any large financial movements in the market (ex. the 2008 financial crisis and the 2007-past bubble before)
2. Stock by Sector

- stocks were organized by the following sectors: Basic Materials, Conglomerates, Consumer Goods, Financial, Health-care, Industrial Goods, Services, Technology, and Utilities
- cut down computational cost of algorithm

3.2.2 Mining Algorithm

All the data was collected via a web-scraping algorithm programmed with R. The central idea of the program is to determine the URLs associated with each Yahoo Finance sector page with those the program can identify the stock symbol by inspecting the HTML content. See Appendix 6.1.1.

After a list of stock symbols were created, the program then uses the function *getYahooData* from TTR: (Technical Trading Rules) package in R to collect the historical prices. This process yields in total approximately 25,000 stocks.

3.2.3 Missing Data

Unsurprisingly, there was missing data in some of the stock historical prices. This could be a due to a variety of factors:

- Stock was split during some time in the year
- Stock was newly created during some time in the year
- Different exchanges operate in different counties, thus some stocks that only operate on specific exchanges might have a lag of information due to holidays

To resolve the problem, only stocks that had a minimum of 70% of data were put in the selection pool. Here 70% of the data is defined for the stock to have 180 entries since there are typically 256 working days in a given year.

For those stocks with empty entries, the missing data was replaced with either the past day closing price or the next day opening price, depending on the information available. This method is commonly used when repairing financial stock data.

3.3 Mean-Variance Portfolio Optimization

3.3.1 Formulation of MVP

To formulate the mean-variance problem, let us consider a simple bi-variate case for simplification, then we can generalize it to n assets easily. To start, let the actual rate of return of asset S_1 and S_2 be X_1 and X_2 respectively. Then, the actual rate of return of our portfolio P is: $P = X_1 w_1 + X_2 w_2$.

These are all random variables, so we can get the expected rate of returns and variances from historical data. And so we can get the expected rate of return of P:

$$E[P] = E[X_1 w_1 + X_2 w_2]$$

By the linearity property of expectation, we get:

$$E[P] = w_1 E[X_1] + w_2 E[X_2]$$

Let $E[P] = \mu_p$, $E[X_1] = \mu_1$, and $E[X_2] = \mu_2$
Hence the expected rate of return of our portfolio is:

$$\mu_p = w_1 \mu_1 + w_2 \mu_2$$

Which in matrix form is: $\mu_p = \mathbf{w}^T\boldsymbol{\mu}$, where, $\mathbf{w}^T = [w_1\ , w_2]$ and $\boldsymbol{\mu}^T = [\mu_1\ , \mu_2]$.

Next, the expected risk (variance) is:

$$\text{Var}(P) = \text{Var}(X_1 w_1 + X_2 w_2)$$

With sum of variance property:

$$\text{Var}(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n}\sum_{j=1}^{n} Cov(X_i, X_j)$$

we get:

$$\text{Var}(P) = \text{Var}(w_1X_1) + \text{Var}(w_2X_2) + Cov(w_1X_1, w_2X_2) + Cov(w_2X_2, w_1X_1)$$

we also know that: $Cov(X_1, X_2) = Cov(X_2, X_1)$, $\text{Var}(w_iX_i) = w_i^2\text{Var}(X_i)$, $\text{Var}(X_1) = Cov(X_1, X_1)$
Therefore:

$$\text{Var}(P) = \text{Var}(w_1X_1) + \text{Var}(w_2X_2) + 2Cov(w_1X_1, w_2X_2) = \sum_{j=1}^{2}\sum_{i=1}^{2} w_j w_i Cov(X_j, X_i) \tag{1}$$

Let $\text{Var}(P) = \sigma_p^2$, $\text{Var}(X_i) = \sigma_{ii}^2$ and $Cov(X_i, X_j) = \sigma_{ij}^2$

In matrix form, (1) becomes: $\sigma_p^2 = \mathbf{w^T\Sigma w}$,where $\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$, and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11}^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 \end{pmatrix}$

We are now ready to formulate the problem. We want the minimal risk with a certain expected rate of return μ_p, while preserving our budget, therefore; we want to:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \sigma_p^2 = \mathbf{w^T\Sigma w} \\ \text{subject to} \quad & \mu_p = \mathbf{w}^T\boldsymbol{\mu} \\ & \mathbf{w}^T\mathbf{1} = 1 \end{aligned} \tag{2}$$

Like mentioned above that this can be easily generalize to n assets, doing so we get:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & \sigma_p^2 = \mathbf{w^T\Sigma w} \\ \text{subject to} \quad & \mathbf{w^T}\boldsymbol{\mu} = \mu_p \\ & \mathbf{w^T1} = 1 \end{aligned} \tag{3}$$

where $\mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$, $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_n \end{pmatrix}$, $\mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$, and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11}^2 & \dots & \sigma_{1n}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{n1}^2 & \dots & \sigma_{nn}^2 \end{pmatrix}$ is the covariance matrix.

It turns out that (3) is a quadratic problem which can be solved using one of the methods from constraint optimization problem. Some of those methods are: Frank Wolfe's method, active set method, interior point convex method, Lagrange method etc. Since (3) is a minimization problem with only equality constraints, we will use Lagrange method to solve for a solution $\mathbf{w}$ which is the weight of our portfolio.

3.3.2 Lagrange Method to Solve MVP Problem

To solve the constrained minimization problem (3), we first create the Lagrangian function

$$L(\mathbf{w}, \lambda_1, \lambda_2) = \mathbf{w^T\Sigma w} + \lambda_1(\mathbf{w^T}\boldsymbol{\mu} - \mu_p) + \lambda_2(\mathbf{w^T1} - 1)$$

There are only two Lagrange multipliers λ_1 and λ_2 because (3) has only two constraints. Lagrange Method requires taking derivative of $L(\mathbf{w}, \lambda_1, \lambda_2)$ with respect to every single variables and set them equal to 0, doing so we have:

$$\begin{aligned}
\frac{\partial L(\mathbf{w}, \lambda_1, \lambda_2)}{\partial \mathbf{w}} &= 2\boldsymbol{\Sigma}\mathbf{w} + \lambda_1\boldsymbol{\mu} + \lambda_2\mathbf{1} = 0 \\
\frac{\partial L(\mathbf{w}, \lambda_1, \lambda_2)}{\partial \lambda_1} &= \mathbf{w^T}\boldsymbol{\mu} - \mu_p = 0 \\
\frac{\partial L(\mathbf{w}, \lambda_1, \lambda_2)}{\partial \lambda_2} &= \mathbf{w^T}\mathbf{1} - 1 = 0
\end{aligned} \tag{4}$$

The number of variables is always equal to the number of asset $n + 2$ (2 for the additional Lagrange multipliers we introduced). Therefore, (4) consists of $n+2$ linear equations in $n+2$ unknowns $(\mathbf{w}, \lambda_1, \lambda_2)$. We can represent the system of linear equations using matrix algebra as:

$$\begin{pmatrix} 2\boldsymbol{\Sigma} & \boldsymbol{\mu} & \mathbf{1} \\ \mu^{\mathbf{T}} & 0 & 0 \\ 1^T & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mu_p \\ 1 \end{pmatrix}$$

or

$$\mathbf{M}\mathbf{y_w} = \mathbf{b_p}$$

where

$$\mathbf{M} = \begin{pmatrix} 2\boldsymbol{\Sigma} & \boldsymbol{\mu} & \mathbf{1} \\ \mu^{\mathbf{T}} & 0 & 0 \\ 1^T & 0 & 0 \end{pmatrix}, \mathbf{y_w} = \begin{pmatrix} \mathbf{w} \\ \lambda_1 \\ \lambda_2 \end{pmatrix}, \text{and } \mathbf{b_p} = \begin{pmatrix} \mathbf{0} \\ \mu_p \\ 1 \end{pmatrix}.$$

The solution for $\mathbf{y_w}$ is:

$$\mathbf{y_w} = \mathbf{M^{-1}}\mathbf{b_p} \tag{5}$$

In $\mathbf{y_w}$, the first n elements are the portfolio weight $\mathbf{w}$ for minimum variance.

This minimization problem is the basis of the Markowitz asset allocation strategies. However, we would also like to look into the Sharpe ratio for this problem.

3.4 Maximization of Sharpe Ratio

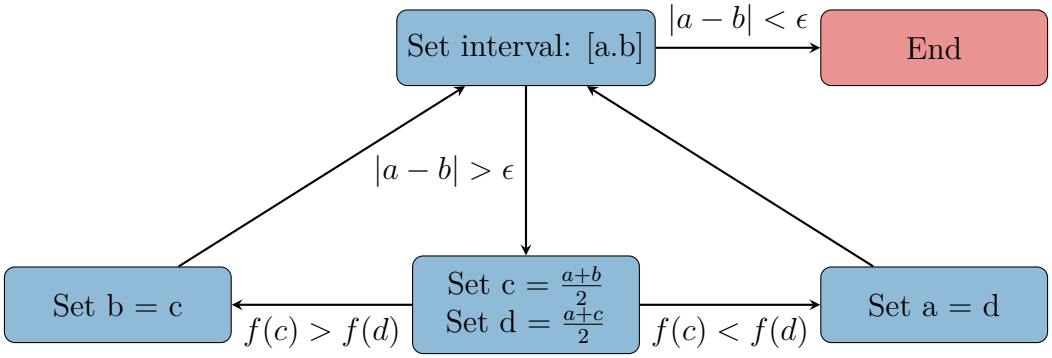
Recall the definition of the Sharpe ratio: $Sharpe = \frac{(\mu_p - X_f)}{\sqrt{\sigma_p^2}}$. Therefore, we can also look at the portfolio asset allocation problem as maximization of the Sharpe ratio:

$$\begin{aligned}
\underset{w}{\text{maximize}} \quad & Sharpe = \frac{(\mu_p - X_f)}{\sqrt{\mathbf{w^T}\boldsymbol{\Sigma}\mathbf{w}}} \\
\text{subject to} \quad & \mathbf{w^T}\boldsymbol{\mu} = \mu_p \\
& \mathbf{w^T}\mathbf{1} = 1
\end{aligned} \tag{6}$$

To solve the maximization problem (6), we are using a **quasi-bisection method** in search for an optimal weight. See appendix for the breakdown of the bisection method.

3.4.1 Quasi-Bisection Method

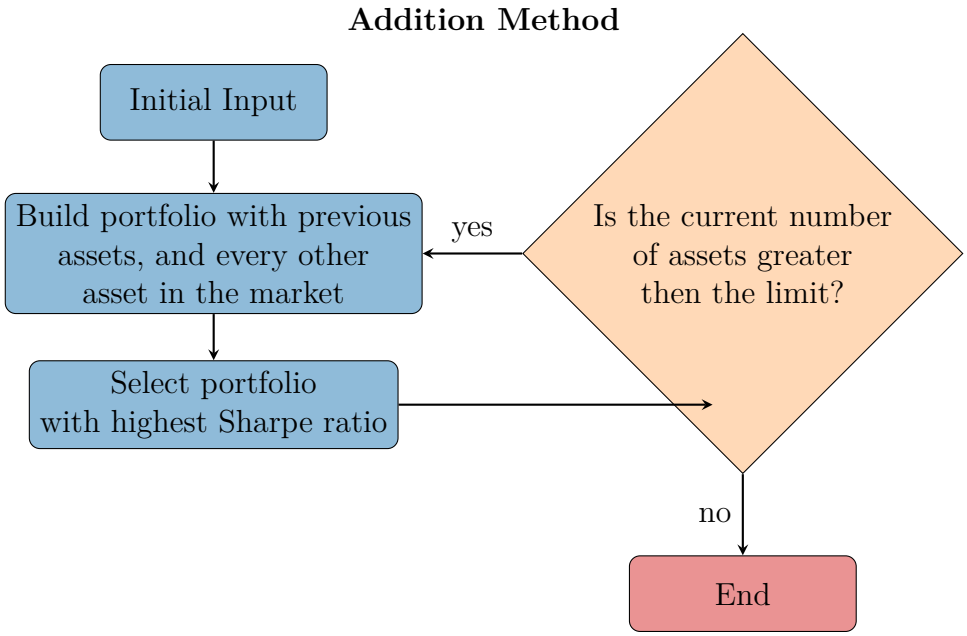
Since the Sharpe ratio is not a simple quadratic function, we could not apply the Lagrangian method without incurring higher level of complexity. Hence we chose to use the quasi-bisection method. Which is a modification of the original bisection method as originally suggest by Burden & Faires 1985. The method itself is described in the following chart. Where f is some concave function over the interval $[a, b]$. However, the first order derivative of this function is **not** defined or is complex. In order to find its maxima we do the following:



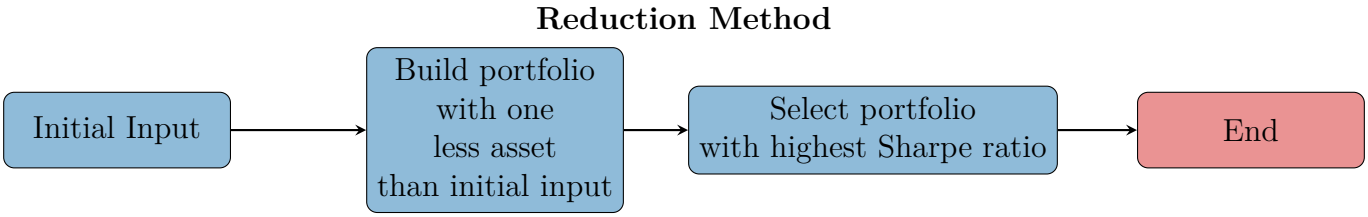
As you may notice, the values of a and b get closer and closer every iteration. Eventually, they will converge to the same point. At this point the maxima is assumed to have been achieved.

3.5 The Selection Process

The selection process, as discussed before, is divided into two parts, an addition part and a reduction part. This is done once the data is filtered and once the method for maximizing Sharpe has been completed. This algorithm uses the previous methods as a way to find the most optimal portfolio given some list of stocks. It will start by first building a portfolio with one asset, and then progressively add another asset by comparing every possible portfolio that can be made from the filtered list of stocks. It will do so, until a predetermined limit of stocks is reached (for our examples, we use `Portfolio_Limit`= 10). This sort of process is what we refer to as the **Addition Method**, and can be described in the following chart:



Once this stage is done, we begin to deduct assets from our list of stocks. We do this, since now that we added a number of assets we wish to check if there is a better alternative. This step is very simple, and simply requires producing a portfolio with one less asset, given every asset in the current list. This is known as the **Reduction Method**.



We then reiterate the two methods (addition and reduction), until a convergence in the portfolio indexes and weights is observed through the Sharpe ratio (in practice it seems to take up to 3 iterations before a convergence is observed).

4 Method Testing

For testing of our code we looked at how well our functions performed relative to the native matlab functions. For the selection algorithm we only used matlab as a benchmark, as there is no existing function that parallels. Please note that all the tests found here can be found in the `implementation/compare_to_matlab.m` file.

4.1 Data

We start by taking a random portion of data within our dataset, as follows:

```
[Ret,CoRisk,stockNames,selData,data]=data_selector(folders,dates(1),sectors(6));
```

Note that `sectors(6)` is arbitrary sector name chosen from our sectors list, and `dates(1)`, is simply the data from `2008-2009`. This function will take the data from the folder, filter it, and then store it in the `selData` variable. Then it will compute the returns, and covariance and store them in the `Ret` and `CoRisk` variables respectively. The variable `data` is simply the unfiltered dataset.

4.2 Lagrange, Variance minimization method

Overview:

Building on the mathematical theory we take the covariance matrix and the returns matrix and construct the following matrices: Note that to ensure that the program does not run for too long, we control the number of assets we wish to test. If we wanted to test **all** of the assets we would set `n = length(Ret)`.

```
M   = Ret(1:n);
S   = 2.*CoRisk(1:n,1:n);
mp  = 0.05;

A = [ S M' ones(n,1); M 0 0 ; ones(1,n) 0 0 ];
x = [ zeros(n,1); mp; 1 ];
```

Hence, we can obtain the solution for our weights as follows: `Weights = A\x;`

Testing:

Next we will see how our function compares to the matlab base function `quadprog`. This will serve us as a benchmark for our function. The following code will compare the matlab function to ours.

```
mp  = 0.05;
n   = length(Ret);
S   = CoRisk(1:n,1:n);
M   = Ret(1:n);

% Matlab
tic
quadprog(2.*S,[],[],[],[ M ; ones(1,n)],[mp;1],...
],[],...
moptions('quadprog','Algorithm',...
erior-point-convex','Display','off'));
ntf('Matlab Time: ');
toc

% Us
tic
    WW = [ 2*S M' ones(n,1); M 0 0 ; ones(1,n) 0 0 ]\[ zeros(n,1); mp; 1 ];
    fprintf('\nUs Time: ');
toc

% Comparison
square_root_sum_of_error_squared = sqrt(sum((WW(1:end-2)-w).^2)./n)
```

```
Matlab Time: Elapsed time is 0.009409 seconds.

Us Time: Elapsed time is 0.000961 seconds.

square_root_sum_of_error_squared =

   1.2973e-13
```

As you can see, our function outperformed the matlab function in terms of time. This is due to the fact that the matlab function runs several tests on the matrices before actually computing the weights.

4.3 Test Sharpe Optimization

The Sharpe optimization procedure we wrote, makes use of the bisection method for finding the Sharpe ratio. It is stored in the `optimizeSupreme` function. For more detail, please refer to the math-section of this paper. The matlab function we chose as the benchmark is the `estimateMaxSharpeRatio` which is a part of the `Portfolio Optimization and Asset Allocation` package in matab.

We run our comparison code as follows:

```
clear n M S rfr WMp mLims
clc
```

```
n      = 10;
tP     = 1:n;
M      = Ret(tP);
S      = CoRisk(tP,tP);
rfr    = RFR(1);%list of risk-free rates (RFR(1) = 3.7%)
mLims  = 1E10;

% Matlab
tic
    p =  Portfolio('AssetMean',M,'AssetCovar',S,'RiskFreeRate',...
    rfr,'Budget',1,'LowerBound',-mLims,'UpperBound',mLims);
    WMp = estimateMaxSharpeRatio(p);
    Matlab_Sharpe = (M*WMp-rfr)/sqrt(WMp'*S*WMp)
    fprintf('Matlab Time: ');
toc
% Us
tic
    [ Sharpe, Wp, ~, ~ ] = optimizeSupreme( M, S, rfr );
    Our_Sharpe = (M*Wp-rfr)/sqrt(Wp'*S*Wp)
    fprintf('\nUs Time: ');
toc
```

```
Matlab_Sharpe =

    0.0963

Matlab Time: Elapsed time is 0.808331 seconds.

Our_Sharpe =

    0.0490

Us Time: Elapsed time is 0.010628 seconds.
```

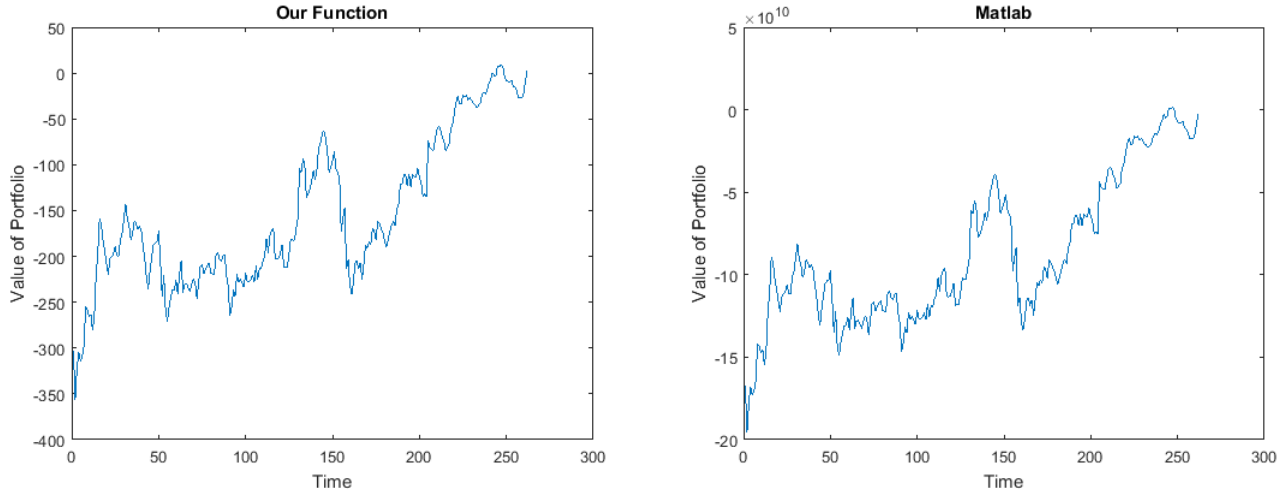
As you can see, our function outperformed the matlab function by a great deal. As a matter of fact, it performs significantly better with higher values of `n`. However, our Sharpe value seems to be at first glance much lower than matlab. This is because we found a *parallel* portfolio to the one of matlab. To show what we mean by that, one may simply observe the ratio between the matlab weights and our weights:

```
disp(WMp./Wp);
```

```
5.3632
5.1647
5.0474
5.1848
5.8016
5.2656
4.9734
5.1739
5.3251
4.9616
```

Or may look at the plot of the optimization period:

```
figure('Name','Our Optimization');
plot(Wp'*selData(:,1:n)')
title('Our Function');
xlabel('Time');
ylabel('Value of Portfolio');
figure('Name','Matlab Optimization');
plot(WMp'*selData(:,1:n)')
title('Matlab');
xlabel('Time');
ylabel('Value of Portfolio');
```



As you can see the only difference between our method and the matlab method is a constant multiplier. In this case the multiplier can be computed as approximately:

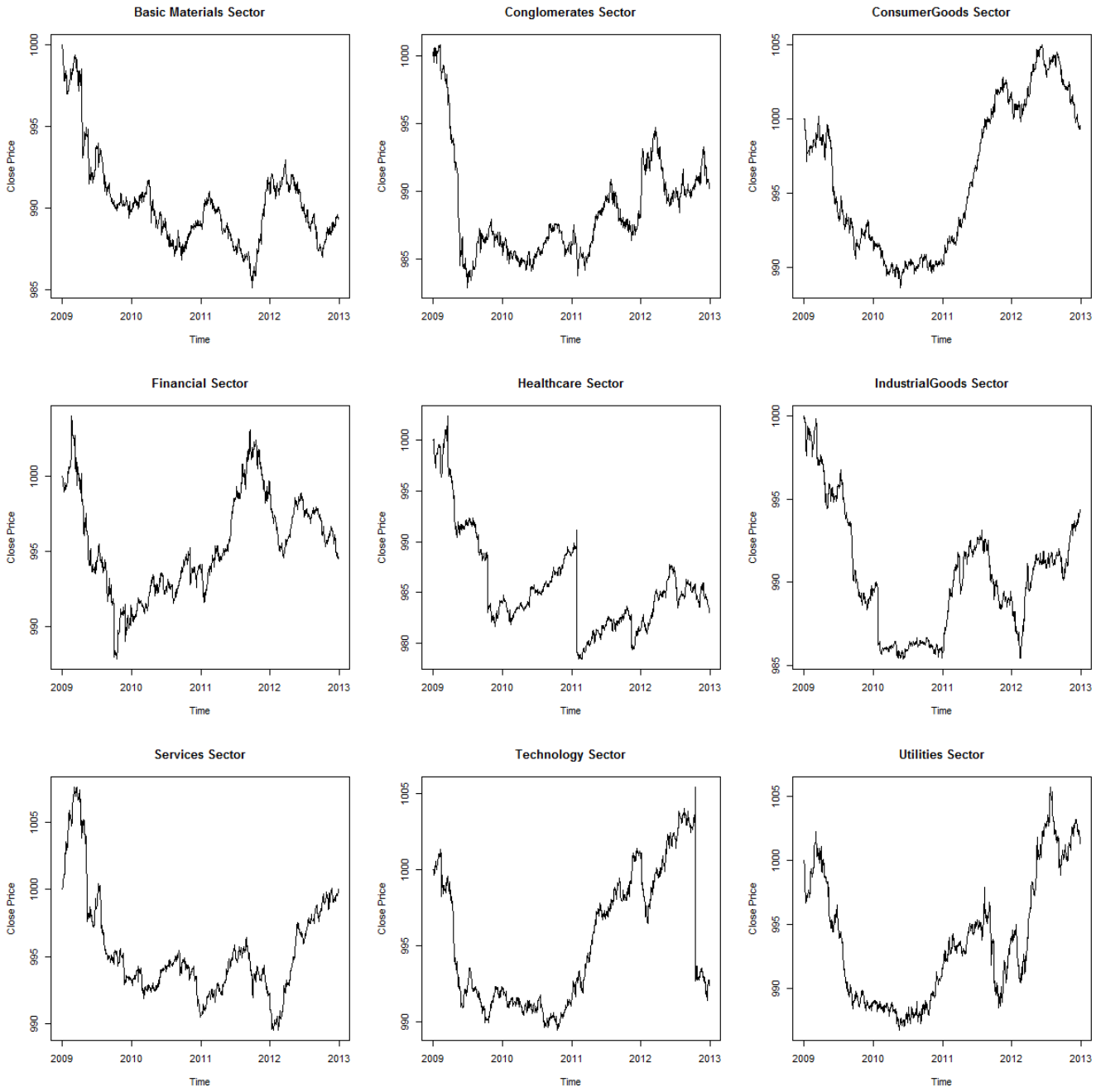
```
approximate_multiplier = mean(WMp./Wp)
WWW = approximate_multiplier.*Wp;
Our_Adjusted_Sharpe = (M*WWW-rfr)/sqrt(WWW'*S*WWW)

approximate_multiplier =
   5.2261e+08
Our_Adjusted_Sharpe =
    0.0929
```

Hence, our method compares well with matlab. As it provides us with a good-enough estimate for the Sharpe of the portfolio.

5 Analysis

5.1 Application

After the filtering process we run the model on the 2008-2012 period. The method involved partitioning the data into years, and then estimating the optimal weights of each year. We then took the weights and applied them to future prices.
That is, we estimated the weights $W_j^T = [w_1, ..., w_{10}]$, with their respected indexes $P_j = [...]$ for year j. Then applied them to the future market prices, where if M_j is a matrix of market prices of all stock prices at year j, then our future portfolios performance in 2009 is computed as follows $w_{2008}^T M_{2009}(:, P_{2008})$.
The results are as follows:



As you can see, each implementation follows a similar trend. We see a clear bad optimization period for the 2009-2010 years. This is due to the fact that the weights in this time are optimized assuming the trend of the 2008 year will continue to the next year. However, 2008 had a particularly bad period for the economy, which did not last after the year ended. Hence, the period itself provided the wrong idea as for what will be seen in the future.
The same can be said about the optimization period of 2012-2013, this period was optimized using prices from 2011-2012, which was the time when Greece defaulted, which sent a major shock to the economy. These sort of examples expose the inefficiency of MVP when it is applied to large datasets.

5.2 Discussion

5.2.1 Limitation of Sharpe and MVP

- One major limitation of the Mean Variance optimization technique is that it does not account for the constant changing of the parameters. This model treats the covariances of stocks as constants

when in essence they are variables. Likewise, the mean may also be shifting each period.

- Another issue with our method is that it does not account for transaction costs. Therefore, it is possible that the portfolio indicated as optimal based on our program. Is not really the optimal one, once transaction costs are factored into the equation.

- Another related problem with MVP is that the solution can be unstable, in other words, small changes in inputs can result in huge changes in the portfolio weight. Because of this instability, a small change in the expected return can lead to entirely different portfolio weights.

- The method seems to especially perform poorly when major market shocks are involved. As can be seen in the analysis stage, when the 2008 crash happened, the model indicated that we should take part in more shorting. However, the year after the crash the markets stabilized and large scale shorting became a poor strategy.

- The Sharpe ratio, as theorized did encourage more stagnant assets. This is because it penalized both negative and positive volatility. While, it can prove to create a more conservative fund, it also interferes with any sort of growth.

5.2.2 Future Study

- In the future, there are a couple of areas we would like to study, that we can potentially use to revise and improve our project.One such area would be a Stop Loss method. In this method, our portfolio will be automatically re-optimized with the available data every time that the portfolio exceeds a specified level of loss. The reasoning behind this decision, is the relatively worsening performance of the portfolio as the duration since the last optimization increases. we theorize that this kind of more frequent optimization would make the portfolio more profitable.

- We would also like to research the Black - Litterman model. The Black-Litterman model is a model used to estimate inputs for portfolio optimization. It mixes different types of estimates, some based on historical data, others based on equilibrium conditions to arrive at updated estimates. The Mixed Estimation Model was developed by Henri Theil in the early 1960's, but was applied to financial data by Fischer Black and Robert Litterman in the early 1990's. The beauty of this model is that one can blend a variety of views specified in different ways, absolute or relative, with a given prior estimate to generate a new and updated posterior estimate which includes all the views. We feel that study this model would be a good next step at improving our project.

- The MVP is known to be overly sensitive to estimation error in risk-return estimates and have poor out-of-sample performance characteristics. The Resampled Efficiency (RE) techniques presented in Michaud (1998) introduce Monte Carlo methods to properly represent investment information uncertainty in computing MVP portfolio optimality and in defining trading and monitoring rules. we would like to study this technique in the future and perhaps integrate it into our program.

5.3 Concluding Remarks

Generally, the returns of the created portfolios were slowly increasing but still proved to be a worse investment then a "safe" portfolio based on the risk free rate. One of the main reasons for poor performance lies in the inherent strategy of mean variance portfolio theory. The theorem requires the assumption of constant mean, variance and assumes that the past will predict the future perfectly. While,theoretically constant more frequent optimization (i.e. partition the data into months or weeks, rather than years) should lead to better performance, as it will force the algorithm to change the mean and variance. This sort of solution will only solve part of the problem inherit to MVP. Ideally, the program would require

second-by-second optimization to ensure best performance. In real world applications, transaction costs, supply/demand and capital accumulation severely decrease any benefit from implementing the strategy. In conclusion, despite modern portfolio theory's reputation as an important advance in mathematical finance, our project has proven that is a poor model for investment strategies, if used without any other supplementary method.

6 Appendices

6.1 Data Retrieval

—R Code—, for the purpose of this section, we decided to use `R` in order to pull data from `yahoo-finance`, as it was a more convenient tool for data cleaning and web-scrapping.

6.1.1 Web scrapping

First we had to scrap yahoo for its links and sector-titles.

```
# Load packages and clean variables
rm(list = ls())
library(httr)
library(rvest)
library(magrittr)

# Start an html session
   yahoo <- html_session("http://biz.yahoo.com/p/")
# Analyze html table
   sector_html <- yahoo %>%
         html_nodes(xpath = "//td//td//a/@href") %>%
         html_text()
         sector_html_titles <- yahoo %>%
         html_nodes("td td a font") %>%
           html_text()
# Save all of the html links for later use
   write.csv(gsub("[\r\n]", "",sector_html_titles),file = "sector_html_titles.csv",row.names =
       FALSE)
   base <- "http://biz.yahoo.com/p/"
   html_pages <- paste(base,sector_html,sep = "")
   write.csv(html_pages,file = "html_links/sector_html.csv",row.names = FALSE)

# Get the sector list and store it into a csv file
  yahoo_industry <- html_session(html_pages[1])
  industry_html <- yahoo_industry %>%
    html_nodes(xpath = "//td//td//a[contains(@href,'conameu')]/@href") %>%
    html_text()
  titles_text <- paste("html_links/",gsub("[\r\n]", "",sector_html_titles),".csv",sep = "")
  for (i in 1:length(html_pages))
  {
    yahoo_industry <- html_session(html_pages[i])
    industry_html <- yahoo_industry %>%
      html_nodes(xpath = "//td//td//a[contains(@href,'conameu')]/@href") %>%
      html_text()
    for (j in 1:length(industry_html))
    {
      sector_html_url <- paste(base,industry_html,sep = "")
      yahoo_industry_stock <- html_session(sector_html_url[j])
      industry_stock_html <- yahoo_industry_stock %>%
       html_nodes("td td a+ a") %>%
       html_text()
```

```
    write.table(industry_stock_html, file = titles_text[i],append = "TRUE",sep = ",",row.
        names = FALSE)

  }
}
```

6.1.2 Filter

Between this step and the last step, we downloaded all the data using a simple loop, and store it in a folder hierarchy, where they were divided by sector. We omitted that process as it was not relevant to the theory.

In this section, we take the downloaded data, partition it, and then filter it, to ensure complete case per partition. Please note that this process took around 4hrs.

```
rm(list=ls())
require(chron)
require(dplyr)
require(timeSeries)
require(zoo)

# Helper function that will generate only working days
getDateVector <- function(d1,d2,format="%Y%m%d") {
  dates <-seq.Date(to = as.Date(d2,format=format),
                   from = as.Date(d1,format=format),
                   by = 1)
  return(as.Date(dates[!is.weekend(dates)]))
}

# get sector titles
sector_html_titles <- gsub("[\r\n]", "", read.csv(file = "sector_html_titles.csv")$x);

partitions_year = rbind(c("20080101","20081231"),
                        c("20090101","20091231"),
                        c("20100101","20101231"),
                        c("20110101","20111231"),
                        c("20120101","20121231"))
# loop through every sector
for (sect in 6:length(sector_html_titles))
{
  sector = sector_html_titles[sect];
  stock_dir <- paste0("/run/media/ari/data-600-ntfs/data/data/time_series/2008-2012/",sector)
  stock_list <- gsub(".csv","",dir(stock_dir))
  # Create a directory
  dir.create("filtered_stocks/",showWarnings = FALSE)
    # loop through every partition
    for(part in 1:length(partitions_year[,1]))
    {
      # create general date vector
      merged_data <- data.frame(Index=getDateVector(partitions_year[part,1],partitions_year[
          part,2]))
      year <- substr(partitions_year[part,1],1,4)
```

```
    #create directory name data/filtered_stocks/YEAR
    merged_data_directory <- paste0("filtered_stocks/",year)
    dir.create(merged_data_directory,showWarnings = FALSE)

    #create file name data/filtered_stocks/YEAR/SECTOR.csv
    merged_data_file <- paste0(merged_data_directory,"/",sector,".csv")

    # loop through each stock
    for (stock in stock_list)
    {
      # Load stock data
      data <- read.csv(paste0(stock_dir,"/",stock,".csv"))
      data[,1] <- as.Date(data[,1]);
      # filter data by year
      data_year <- dplyr::filter(data,grepl(year,Index))

      # coerce index to date and close prices to numeric
      data_year$Index <- as.Date(data_year$Index)
      data_year$Close <- as.numeric(data_year$Close)

      # rename col.names to specific stock
      data_year_short <- data_year[c("Index","Close")]
      colnames(data_year_short) <- c("Index",paste0("Close.",stock))

      #if the stock has more than 85% of time data we keep it
      if (nrow(data_year) > nrow(merged_data)*.85)
      {
        merged_data <- merge(merged_data,data_year_short,by.x = "Index",by.y = "Index",all.x
            = TRUE)
        # determine if last column of merged_data had NA values if so replace with previous
            open price.

        last.na = 0;
        # If there is still holes in that data complete it with other open/close price
        for (naInd in which(is.na(merged_data[,ncol(merged_data)])) ) {
          possibs = which(data[,1] < merged_data[naInd,1]);
          if( length(possibs) > 0 ) {
            merged_data[naInd,ncol(merged_data)] = data[max(possibs),4];
          } else {
            merged_data[naInd,ncol(merged_data)] = data[min(which(data[,1] >= merged_data[
                naInd,1])),2];
          }
        }

      }
      # Print completion
      print(paste0(round(which(stock_list == stock)/length(stock_list)*100,2),'% Done, [',
          ncol(merged_data),'/',length(stock_list),'] year: ', year, ' sector: ', sector ));
    }
    # save
    write.csv(merged_data,file = merged_data_file,row.names = FALSE);
  }
}
```

6.2 Data Selector

—Matlab Code—: This function is ran after the data has been scrapped from yahoo

```
function [ Ret, CoRisk, fStocknames, filData, data, stockNames, P, returns ] = ...
              data_selector( folders, date, sector )
%data_selector This function will select a given partition given a sector,
%date and folder object. Then will filter that data based on variance and
%correlation.
%
%   [Inputs/Outpus]
%   folders       a ‘structure‘ that contains the field ‘data‘ as
the real location of data
%   date          a ‘string‘-cell that contains the requested date,
%   sector        a ‘string‘-cell that contains the requested sector
%   Ret           a ‘vector‘ of the means of the selected data
%   CoRisk        a covariance ‘matrix‘ for the selected data
%   fStocknames   a ‘cell‘ with the names of all the fitered stocks
%   fillData      a ‘matrix‘ with all the filtered data
%   data          a ‘matrix‘ with the unfiltered data
%   stockNames    a ‘cell‘ with the names of the unfiltered stocks
%   P             a ‘vector‘ with the indexes of the filtered
stocks inside the unfilterd ‘data‘
%   returns       a ‘matrix‘ with the returns of the filtered data
%
%

    % (0) Folder Location
    folders.tmp = strcat(folders.data, date, '/',sector,'.csv');
    fulldataset = importdata(char(folders.tmp));
    stockNames = regexprep(strrep(fulldataset.textdata(1,:),'Close.',''),'["|.]','');
    stockNames = stockNames(2:end);
    data = fulldataset.data;

    % (1) Compute returns
    data(data == 0) = 1E-20;
    returns = (data(2:end,:)-data(1:end-1,:))./data(1:end-1,:);
    Ret = mean(returns);
    CoRisk = cov(returns);

    % (2) Data Reduction
    % (2.1) by correlation
    temp = (sort(diag(CoRisk)));
    temp = temp(1:floor(0.5*length(Ret)));
    P = [];
    for (pot=temp')
        [xx, ~] = find(diag(CoRisk) == pot);
        P = [P xx'];
    end
    P = unique(P);

    % (2.2) by variance
    temp = sort(reshape(CoRisk(P,P),[1,numel(P)^2]));
    temp = temp(1:floor(0.5*length(P)));
```

```
    P = [];
    for (pot=temp)
        [xx, ~] = find(CoRisk == pot);
        P = [P xx'];
    end
    P = unique(P);

    % Saving of fileterd data
    filData = data(:,P);
    returns = returns(:,P);
    Ret     = Ret(P);
    CoRisk = CoRisk(P,P);
    fStocknames = stockNames(P);
end
```

6.3 Bisection Method for Sharpe

—Matlab Code—: This function will run the bisection method, until a maximized Sharpe ratio will occur.

```
function [ sharpe, Wp, sharpes, Wps ] = optimizeSupreme( M, S, rfr )
%optimizeSupreme finds the most optimal sharpe value using the bisection
%method.
%   [Input/Output]
%   'M'         a 'vector' with the means of all the assets
%   'S'         the Covariance 'matrix' for all the assets
%   'rfr'       a 'double' with the risk-free-rate
%   'sharpe'    a 'double' containing the most optimal sharpe
%   'Wp'        a 'vector' with the most optimal weights
%   'sharpes'   a 'vector' with all attempted sharpes (can be
used to generate an efficiency frontier)
%   'Wps'       a 'vector' with all attempted weights
%
%

% Globals to this function
M = M -rfr;
% Note, E[P] - rfr = E[w_1*X_1+w_2*X_2] - rfr*(w_1+w_2) = E[w_1(X_1-rfr)+w_2(X_2-rfr)]
n = length(M);

% this function will compute the sharpe ratio for us
sharpe = @(w) (M*w)/sqrt(w'*S*w);

% optimo is the lagrangian method, will take a fixed return 'mp' and return
% optimal weights
function w = optimo(mp)
    % Optimize weights
     w = [ 2*S M' ones(n,1); M 0 0 ; ones(1,n) 0 0 ]\[ zeros(n,1); mp; 1 ];
     w = w(1:end-2);% this will get rid of the 'lambda' values
end

% Containers
Wps = zeros(1,n);
```

```
sharpes = [Inf 0];

% Variables
% initial limit, abs is taken in case of negative returns.
% This way shorting will not be penalized
limmp = [max([min(abs(M)) rfr]) max(abs(M))];
c = 2;% initial starting point
m0 = limmp(1);
mk = limmp(2);
while (abs(diff(limmp))> 1E-200 && c < 500)
    %disp(sharpes);
    ud = [ mk<m0 mk>m0 ];%if mk > m0 then limmp([0 1])= max else limmp([1 0])= min
    %^ this is done, in case negative values are enetered and the
    % max, min are not in any given order
    m0 = median(limmp);% compute first median [----*----]
    mk = median([m0 limmp(ud)]);% compute second median [--*--|----]

    Wp = optimo(m0);% find optimal weights
    sharpes(c) = sharpe(Wp);% store them

    Wp = optimo(mk);% compute the next optimal weights
    sharpes(c+1) = sharpe(Wp);% store them

    Wps(c,:) = Wp;% save weights

    % Test
    if(sharpes(c+1) < sharpes(c))
        limmp(ud) = mk;
    else
        limmp(~ud) = m0;
    end
    c = c+1;
end
    % store resultig sharpe
    sharpe = sharpes(end);
end
```

6.4 Selection Algorithm

—Matlab Code—: This function will run the addition and reduction methods on a sector, until an optimal portfolio gets chosen, with a maximized sharpe return.

```
function [ Wp, P, sharpe ] = optimizeSelect( Ret, CoRisk, rfr, portlim )
%optimizeSelect builds the most optimal portfolio, given a limit, a
%risk-free-rate, and the Covariance matrix and vector mean.
%   [Inputs/Outputs]
%   'Ret'      a 'vector' of the means of a given market
%   'CoRisk'   the Covariance 'matrix' of a given market
%   'rfr'      a 'double' containing the risk free rate
%   'portlim'  a 'double' containing the limit for this portfolio
%   'Wp'       a 'vector' contating the final selected weights
%   'P'        a 'vector' containing the final selected indexes of the given assests
%   'sharpe'  a 'double' containing the sharpe ratio
```

```
%

% Initialize algorothimn
[~,P]  = min(diag(CoRisk));
Wp     = [1];
sharpe = 0;

% Helpers

% function addPort, will add assets to the portfolio,
% untill a limit is reached
function addPort()
    %Begin by generating a loop equalling the total number of empty "spots"
    for repNum = 1:portlim-length(P)
        % Generated a list of Unselected assets
        Unsel = setdiff(1:length(Ret),P);

        % Set local variables
        fSha  = 0;
        fID   = [];
        fW    = Wp;

        % For every unselected asset
        c = 0;
        for assID = Unsel
            c = c+1;
            % Get the Expected Returns
            M = Ret([P assID]);
            % Get Covariance Matrix and count the current number of assets
            S = CoRisk([P assID],[P assID]);
            % Computing Sharpe
            [ s, w ] = optimizeSupreme( M, S, rfr );
            % Determine if Portfolio is optimal
            if (~isempty(w) && s(end) > fSha)
                fID    = assID;
                fSha   = s(end);
                fW     = w;
            end
        end
        % save results
        P      = [P fID];
        Wp     = fW;
        sharpe = fSha;
    end
end

% fucntion redPort will generate all potential portfolios containing one
% less asset, and then select the one with the highest sharpe.
function redPort()
    % Initialize local variables
    fSha   = 0;
    fIDs   = P;
    fW     = Wp;
```

```
    % run loop on every selected index
    for portID = P
        % Get a reduced list ( a list without the currently selected asset)
        redList = setdiff(P,portID);
        % check in case empty
        if(~isempty(redList))
            % Get variables
            M   = Ret(redList);
            S   = CoRisk(redList,redList);

            % Computing Sharpe
            [ s, w ] = optimizeSupreme( M, S, rfr );

            % Determine if Portfolio is optimal
            if (~isempty(w) && s(end) > fSha)
                fSha   = s(end);
                fIDs   = redList;
                fW     = w;
                sharpes = [s; fSha];
            end
        end
    end
    % Set the selected Portfolio as our current
    P      = fIDs;
    Wp     = fW;
    sharpe = fSha;
end

% Run initial addition
addPort();

% initialize watchers
lSharpe = 0;
% run loop until a divergence occurs
while abs(lSharpe-sharpe) > 1E-10
    lSharpe = sharpe;
    redPort();
    addPort();
end

end
```

6.5 Final Analysis

—Matlab Code—: This is the code that we ran, when we generated the plots in section (5.1). This code uses the previous `matlab` functions to generate optimal weights, store the names of selected stocks and then generate the price of our portfolio in for every upcoming year.

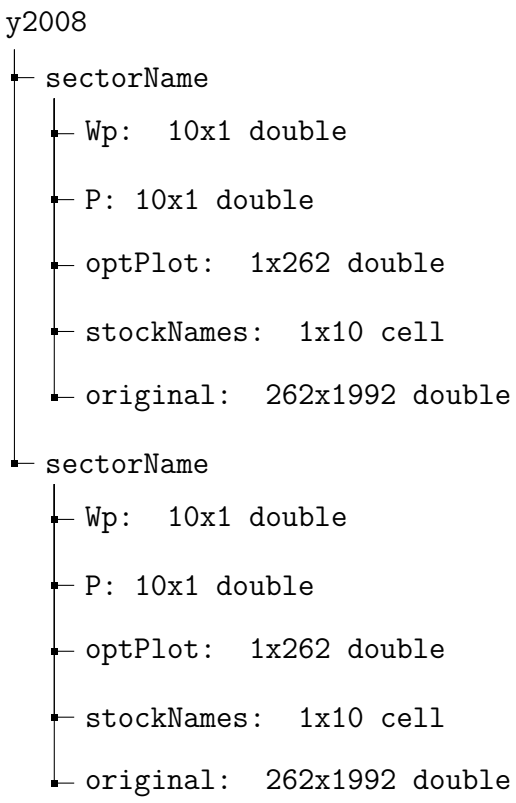
Please note that the variables `sectors`, `dates` and `folders` were generated by a function which was omitted from this report. We also omitted the explanation behind how —findlocSubstrings—, as it was not relevant.

1. `sectors`, is a cell containing strings, with the names of each sector.
2. `dates`, is a cell containing strings, with the dates of each partition
3. `folders`, is an object containing strings with all the needed folders
4. `findlocSubstrings`, is a function that will find all unique strings that are in one list, and will return their indexes.

Please note that this function took around 30min to run, and so we added a lot of —display— functions in different fields.

```
% Inputs
RFR = [0.0365   0.0117   0.0143   0.0169   0.0100 ];
PortfolioLimit = 10;

%% (2) Analysis
clc
% loop on every sector
for sector = sectors
    disp(sector);
    % Start analyzing the first year (2008)
           DATA = [1000];
           date = dates(1);%2008
    % Get Data
           [ Ret, CoRisk, fstockNames, Fdata, Odata ] = data_selector( folders, date, sector )
              ;
    % Optimize
           [ Wp, P ] = optimizeSelect( Ret, CoRisk, RFR(1), PortfolioLimit );
    % Store
           tmpData = struct('Wp',Wp,'P',P,'optPlot',Wp'*Fdata(:,P)');
           tmpData.stockNames = cellstr(fstockNames(P));
           tmpData.original = Odata;
           data.(char(strcat('y',date))).(char(strrep(sector,' ','_'))) = tmpData;
    % Loop on all next years
    for date=dates(2:end)
        % Get Data
           [ Ret, CoRisk, fstockNames, Fdata, ~, stockNames, ~,returns ] =...
                     data_selector( folders, date, sector );
        % Apply Past Prices
           [foundP, foundI] = findlocSubstrings(stockNames, tmpData.stockNames);
           DATA = [DATA (DATA(end)+cumsum(Wp(foundI)'*returns(:,foundP)'))];
        % Optimize
           [ Wp, P ] = optimizeSelect( Ret, CoRisk, RFR(1), PortfolioLimit );
        % Store
           tmpData = struct('Wp',Wp,'P',P,'optPlot',Wp'*Fdata(:,P)');
              tmpData.original = Odata;
              tmpData.stockNames = cellstr(fstockNames(P));
              data.(char(strcat('y',date))).(char(strrep(sector,' ','_'))) = tmpData;
    end
end
```

This ultimately creates a `structure` called `data`. This structure will look as follows:



The years are obviously going to be $2008 - 2012$, this is not an exhaustive list. Also, `sectorName` will be replaced with every sector name (i.e. Services, Technology, Basic Materials etc).

References

[1] Brandimarte, Paolo. "Stock Portfolio Optimization." *Numerical Methods in Finance and Economics: A MATLAB-based Introduction*. 2nd ed. Hoboken, N.J.: Wiley Interscience, 2006. 65 - 81, 571. Print.

[2] Corliss, George. "Which Root Does the Bisection Algorithm Find?" *SIAM Rev. SIAM Review* 19.2 (1977): 325-27. Print.

[3] Lummer, Scott. "Taming Your Optimizer: A Guide Through the Pitfalls of Mean-Variance Optimization." *Global Asset Allocation: Techniques for Optimizing Portfolio Management*. Ed. Jess Lederman. Illustrated ed. Vol. 29. New York: Wiley, 1994. Print.

[4] Michaud, Richard O., and Robert Michaud. "Estimation Error and Portfolio Optimization: A Resampling Solution." *Journal Of Investment Management* 6.1 (2008): 8-28. Print.

[5] Pat. "Alpha Alignment." *Portfolio Probe*. 9 July 2012. Web. 13 Oct. 2015.

[6] Pat. "The Top 7 Portfolio Optimization Problems." *Portfolio Probe*. 5 Jan. 2012. Web. 4 Dec. 2015.

[7] Walters, Jay. "What Is the Black-Litterman Model." *BlackLitterman.org*. 2013. Web. 20 Nov. 2015.

[8] Wayne, Thorp. "Mean Variance Optimization." *Computerized Investing*. Web. 10 Oct. 2015.

[9] Winston, Wayne L. "Nonlinear Programming." *Operations Research: Applications and Algorithms*. 4th ed. Belmont, Calif.: Duxbury, 2003. Print.