Testing and Code

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 exisiting function that parrellels. Please note that all the tests found here can be found in the `Mean-Value-Opt/implementation/data_matlabazer.m` file.

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.

Lagrange, Variance minimization method

Overview:

Building on the mathematical theory we take the covariance matrix and the returns matrix and constructs 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)`.

```
n   = 100;
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 wieghts as follows: `Weights = A\x;`

Testing:

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

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

% Matlab
tic
    w = quadprog(2.*S,[],[],[],[ M ; ones(1,n)],[mp;1],...
            [],[],[],...
            optimoptions('quadprog','Algorithm','interior-point-convex','Display','off'));
    fprintf('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.

Test Sharpe Optimization

The sharpe optimization procudure 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);
mLims   = 1E10;

% Matlab
tic
    p =  Portfolio('AssetMean',M,'AssetCovar',S,'RiskFreeRate',rfr,'Budget',1,'LowerBound',-
    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 significally better with higher values of `n`. However, our sharpe value seems to be at first glance much lower than matlab. This is beacuse we found a *parrellel* 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:

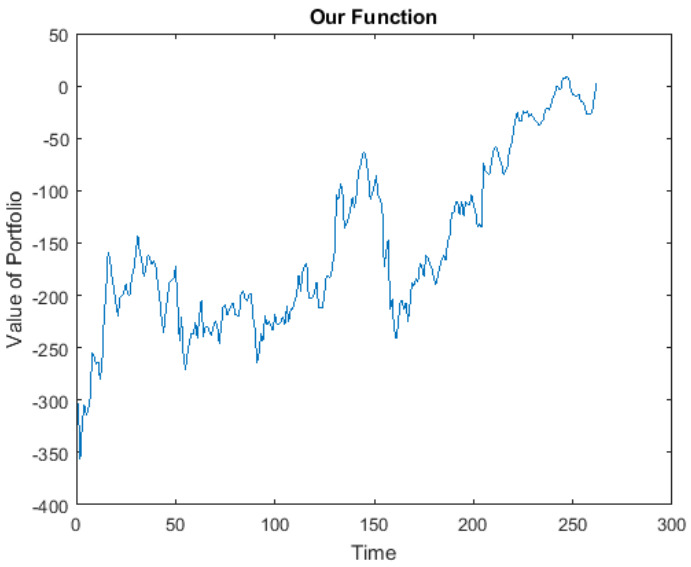
```
disp(WMp./Wp);
```

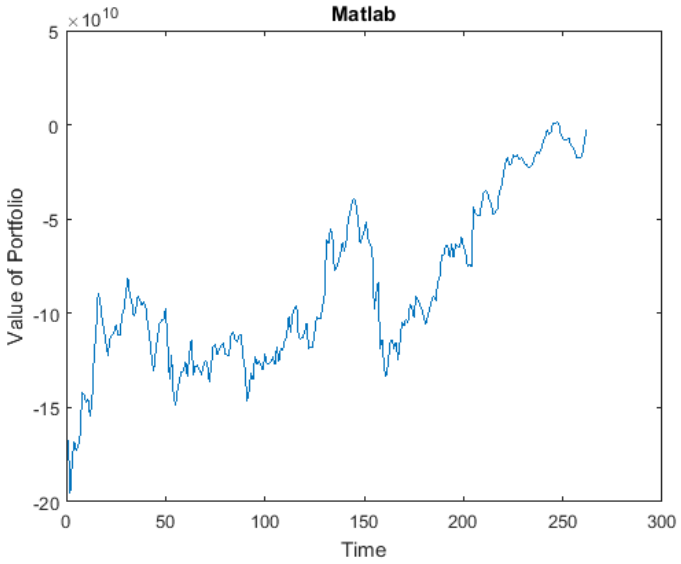
```
1.0e+08 *

 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 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 a good-enough estimate for the sharpe of the portfolio.