# MIE1621 Computational Project

Comparison of multiple optimization algorithms

Vaughn Gambeta 1004612430

**(Part 1) Financial Optimization Problem Formulation**

The maximizing financial optimization model can be modeled as a minimization problem by negating the function. The minimization problem can be utilized in common optimization algorithms.

$$\text{minimize} \quad \frac{\delta}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_{ij}x_i x_j \; - \sum_{i=1}^{n}\mu_i x_i \tag{1}$$

$$\text{subject to} \qquad \sum_{i=1}^{n} x_i = 1$$

The single equality constraint allows the problem to be redefined as an unconstrained optimization problem through the construction of the Lagrangian,

$$L(x,\pi) = \frac{\delta}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_{ij}x_i x_j \; - \sum_{i=1}^{n}\mu_i x_i + \pi(\sum_{i=1}^{n} x_i - 1) \tag{2}$$

So the problem can be represented as an unconstrained minimization problem which will make determining the optimum solution easier.

$$\text{minimize} \quad \frac{\delta}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_{ij}x_i x_j \; - \sum_{i=1}^{n}\mu_i x_i + \pi(\sum_{i=1}^{n} x_i - 1)$$

$$\text{subject to} \qquad \mathrm{x} \in \mathbb{R}^n$$

This form can be represented in vector form to simplify the notation.

$$\mathrm{F(x)} = \quad \frac{\delta}{2}x^T Q x - C^T + \pi(Ax - b) \tag{3}$$

Where the variables are equal to the following financial information and constants,

$$x = \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} \quad Q = \sigma_{ij} = \begin{bmatrix} 0.02778 & 0.00387 & 0.00021 \\ 0.00387 & 0.01112 & -0.00020 \\ 0.00021 & -0.00020 & 0.00115 \end{bmatrix} \quad C = \mu_i = \begin{bmatrix} 0.1073 \\ 0.0737 \\ 0.0627 \end{bmatrix} \quad A = [1,1,1] \quad b = 1$$

The problem is a multivariate function with three financial products where the portfolio asset weight for each product will be determined to maximize risk-adjusted return.

**(Part 2) Apply Optimization Methods - Fixed Step Length**
The formulation (3) will be implemented in MATLAB with three different optimization methods and the resulting outcomes will be discussed. Using the provided financial covariance matrix and expected returns data the following three algorithms are implemented with a fixed step length of one.

Stopping Criteria: The stopping criteria for each method is the magnitude of the difference between newest iteration and the previous iteration. Each iteration the absolute value of the difference between iterations is calculated and compared to a tolerance level of 0.00000001. The iteration loop also includes a hard stop counter to stop iterating after a pre-set number of loops for any methods that do not converge.

Parameters: The following parameters are fixed for the problems and used across the three method. This includes the step length, risk aversion value and initial point, which represents weights of each asset. The Lagrange constant, $\pi$, is given an initial value of 0.8.

$$\delta = 3.5 \qquad \alpha = 1 \qquad x_0 = \begin{bmatrix}0.5, & 0.3, & 0.2\end{bmatrix} \quad \pi_0 = 0.8$$

**Multivariate Newton's Method**
The Multivariate Newton Method iterate (4) is implemented and the results tabulated below. The MATLAB code and output can be found in the Appendix.

$$x^{k+1} = x^k - \alpha^k \big[H(x^k)\big]^{-1} \cdot \nabla f(x^k) \tag{4}$$

Algorithm Design

1. Guess $x^0$, k = 0
2. while ||norm($x^{k+1}$) - norm($x^k$)|| > tolerance
3. $\qquad x^{k+1} = x^k - \alpha\big[H(x^k)\big]^{-1}\nabla f(x^k)$
4. $\qquad k = k + 1$
5. end while
6. return $x^{k+1}$

| Newton's Method - No Backtracking | | | | |
|---|---|---|---|---|
| Iterate | Lagrange | Asset 1 | Asset 2 | Asset 3 |
| 1 | 0.060988545617122 | 0.448803604749846 | 0.177138996295219 | 0.374057398954935 |
| 2 | 0.060988545617122 | 0.448803604749846 | 0.177138996295219 | 0.374057398954935 |

The newtons method converges in only one iteration in less then 0.05 seconds. This method runs very smoothly because the objective function is quadratic and when combined with newtons method, is the fastest possible solution. The iteration tolerance stopping condition is hit indicating a high degree of convergence. This outcome was expected with this type of objective function and optimization method.

**Steepest Descent Method**

The Steepest Descent Method iterate (5),

$$x^{k+1} = x^k - \alpha^k \cdot \nabla f(x^k) \tag{5}$$

<u>Algorithm Design</u>

1. Guess $x^0$, k = 0
2. while ||norm($x^{k+1}$) - norm($x^k$)|| > tolerance
3. $\quad x^{k+1} = x^k - \alpha \nabla f(x^k)$
4. $\quad k = k + 1$
5. end while
6. return $x^{k+1}$

| Steepest Descent Method - No Backtracking | | | | |
|---|---|---|---|---|
| Iterate | Lagrange | Asset 1 | Asset 2 | Asset 3 |
| 1 | 0.800000000000000 | -0.245525500000000 | -0.444608500000000 | -0.538262500000000 |
| 2 | 3.028396500000000 | -0.907935210565000 | -1.150655478032500 | -1.273526758145000 |
| 3 | 7.360513946742501 | -3.724231499424579 | -4.049161953131074 | -4.234235439398324 |
| 4 | 20.36814283869647 | -10.55738035577490 | -11.13090176580558 | -11.51510369171236 |
| 5 | 54.57152865198933 | -29.63249743684812 | -30.85719076344213 | -31.77423019472426 |
| 6 | 147.8354470470039 | -80.77424365496882 | -83.77492733427251 | -86.15498771809823 |
| 7 | 399.5396057543434 | -219.4506556846845 | -227.2423755705227 | -233.5802343195843 |
| . | - | - | - | - |
| 25 | | | | |

The steepest descent method did not converge and the maximum iteration limit was hit. The data reveals that the iterations continued to increase and is expected to goto infinity. The steepest descent method is known to be less efficient then newtons so a higher number of iterations would be expected, but convergence is only guaranteed if the step length is choosing carefully each iteration through a line search. In this model, the step length is fixed at 1 which is large compared to the size of the weights. This will lead to over stepping the solution each iteration resulting in the non-convergence.

As can be seen, iteration 2 jumps the solution. The orthogonal steps of this gradient method continue to di-verge from the solution.

**BFGS Quasi-Newton Method**

The BFGS Quasi-Newton Method iterate (9) requires the determination of the approximate Hessian (8) for each iteration. The method uses the following equations,

$$S = x^{k+1} - x^k \tag{6}$$

$$Y = \nabla f(x^{k+1}) - \nabla f(x^k) \tag{7}$$

$$H_{k+1} = H_k + \frac{YY^T}{Y^T S} - \frac{H_k S S^T H_k}{S^T H_k S} \tag{8}$$

$$x^{k+1} = x^k - \alpha^k H_{k+1} \cdot \nabla f(x^k) \tag{9}$$

The first iteration, $x^{k+1}$, is determined using the steepest descent direction with $x_0$, a step length of 1 and an easily invertible identity matrix as the initial $H_0$,

$$x^{k+1} = x^k + \sigma d_k \qquad \text{where} \qquad d_k = -\big[H_0\big]^{-1} \cdot \nabla f(x^k)$$

After the initial point is found equations (6) and (7) can be determined and used to find the approximate hessian (8). The next iteration can be calculated with equation (9).

The BFGS Quasi-Newton Method will now iterate using the new $x^{k+1}$ and $x^k$ iterates until it reaches a stopping condition. The results are tabulated below.

<u>Algorithm Design</u>

1. Guess $x^0$, k = 0
2. while ||norm($x^{k+1}$) - norm($x^k$)|| > tolerance
3. $\quad x^{k+1} = x^k - \alpha\big[H_{k+1}\big]^{-1}\nabla f(x^k)$
4. $\quad S = x^{k+1} - x^k$
5. $\quad Y = \nabla f(x^{k+1}) - \nabla f(x^k)$
6. $\quad H_{k+1} = H_k + \frac{YY^T}{Y^T S} - \frac{H_k S S^T H_k}{S^T H_k S}$
7. end while
8. return $x^{k+1}$

| BFGS Quasi-Newton Method - No Backtracking | | | | |
|---|---|---|---|---|
| Iterate | Lagrange | Asset 1 | Asset 2 | Asset 3 |
| 1 | 0.800000000000000 | -0.24552550000000 | -0.44460850000000 | -0.53826250000000 |
| 2 | 40.50887659030511 | -722.681436249582 | -722.784031893212 | -717.349339353524 |
| 3 | 0.057985697794962 | 0.263574216053485 | 0.078415072854796 | -0.00172807512272 |
| 4 | 0.057869532462336 | 0.486164846121205 | 0.300387470791205 | 0.224354806776043 |
| 5 | 0.057961638368289 | 0.481426054257449 | 0.295222015516385 | 0.225043168865037 |
| 6 | 0.059547127994193 | 0.432278600147684 | 0.237694003409938 | 0.268687116735391 |
| 7 | 0.060667063171462 | 0.416286921424328 | 0.213755034432992 | 0.317996221007498 |
| 8 | 0.061289963321549 | 0.419191306003884 | 0.208792491050244 | 0.357249151880279 |
| 9 | 0.061311511497575 | 0.425760254978729 | 0.210778841649375 | 0.365294795978402 |
| 10 | 0.061249110866476 | 0.434066908691092 | 0.210557547577018 | 0.370553163707902 |
| 11 | 0.061123628911109 | 0.445102758016332 | 0.203941783972458 | 0.375626877648248 |
| 12 | 0.060999586593628 | 0.452645595058327 | 0.190136929930173 | 0.377678259140225 |
| 13 | 0.060964901567318 | 0.452071717251528 | 0.179443186266813 | 0.375977187674454 |
| 14 | 0.060980260960727 | 0.449561218452555 | 0.176956104914042 | 0.374401825522858 |
| 15 | 0.060987725270366 | 0.448855974287342 | 0.177066569668208 | 0.374072018406017 |
| 16 | 0.060988524570238 | 0.448803678392896 | 0.177134547184867 | 0.374056643244764 |
| 17 | 0.060988545666316 | 0.448803541668104 | 0.177138932721694 | 0.374057340133925 |
| 18 | 0.060988545593663 | 0.448803603941808 | 0.177138997448344 | 0.374057396653461 |
| 19 | 0.060988545612953 | 0.448803604804194 | 0.177138996424428 | 0.374057398762496 |

The BFGS method determines the minima in a similar manner as the newton's method and was expected to show converge.

The BFGS method hits the stopping criteria at iterate 19, but can be said to really converge at iteration 15. This convergence time was approximately 0.4 seconds, making it around 8 times slower then the Newtons method to perform the same task. In this example, the hessian is not difficult to compute for the newtons method, so Newtons Method would be preferable to the BFGS. In more complex problems, the BFGS will be more viable with the approximate hessian being a lower computational requirement and may show higher reliability in convergence.

### (Part 3) Apply Optimization Methods - Backtracking Step Adjustment

Backtracking line search is implemented to determine an optimal step length each iteration. Backtracking line search is used to find a step length that meets the following Wolfe Conditions,

$$d^k \cdot \nabla f(x^{k+1}) > \beta d^k \cdot \nabla f(x^k) \tag{10}$$

$$f(x^{k+1}) \leq f(x^k) + \gamma\alpha^k d^k \cdot \nabla f(x^k) \tag{11}$$

The Wolfe Conditions (10) and (11) are implemented by choosing an adjustment coefficient, $\gamma \in [0, 1/2]$ and a $\beta \in [1/2, 1]$. For this case of backtracking implementation, the backtracking adjustment coefficient will be selected as 0.5 and a beta coefficient as 0.75.

Each iteration of the optimization will check the condition (11). If the condition is not met, a loop will decrease the step length, $\alpha$, by the adjustment coefficient, $\gamma = 0.5$.

The algorithm for backtracking is as follows,

```
%Coefficients
damp = 0.0005;
adj = 0.5;
beta = 0.75;

%Function Evaluation
F1 = eval(subs(F,{P,X1,X2,X3},{X0(1,1),X0(2,1),X0(3,1),X0(4,1)}));
F2 = eval(subs(F,{P,X1,X2,X3},{XI(1,1),XI(2,1),XI(3,1),XI(4,1)}));

%Backtracking
while (F2 > F1 + damp*step*dot(d,G1))
   if(dot(d,G2) < beta*dot(d,G1))
      step = step * adj;
      XB = X0 + step*d;
      F2 = eval(subs(F, {P, X1,  X2, X3}, {XB(1,1), XB(2,1), XB(3,1), XB
         (4,1)}));
   else
        break
   end
end
```

The input into the algorithm will be the function and the gradient evaluated at $x_k$ along with the specific direction vector $d^k$ associated with the specific optimization method.

**Multivariate Newton's Method - with Backtracking**

The above backtracking method was introduced into the Newtons Method algorithm with the direction vector (12), the MATLAB code can be found in the appendix. The results of the iterations are tabulated.

$$d^k = -\left[H(x^k)\right]^{-1} \cdot \nabla f(x^k) \tag{12}$$

Algorithm Design

1. Guess $x^0$, k = 0, $\alpha = 1$
2. while ||norm($x^{k+1}$) - norm($x^k$)|| > tolerance
3. $\quad x^{k+1} = x^k - \alpha\big[H(x^k)\big]^{-1}\nabla f(x^k)$
4. $\quad$ while $f(x^{k+1}) > f(x^k) + \gamma\alpha \cdot d \cdot \nabla f(x^k)$ - (Wolfe Condition 4)
5. $\quad\quad$ if $d^k \cdot \nabla f(x^{k+1}) > \beta d^k \cdot \nabla f(x^k)$ - (Wolfe Condition 3)
6. $\quad\quad\quad \alpha = \alpha \cdot ADJ$
7. $\quad\quad\quad x^{k+1} = x_0 + \alpha \cdot d$
8. $\quad\quad\quad f(x^{k+1})$
9. $\quad\quad$ end if
10. $\quad$ end while
11. $\quad k = k + 1$
12. end while
13. return $x^{k+1}$

| Newton's Method - with Backtracking | | | | | |
|---|---|---|---|---|---|
| Iterate | Lagrange | Asset 1 | Asset 2 | Asset 3 | $\alpha$ |
| 1 | 0.060988545617122 | 0.448803604749846 | 0.177138996295219 | 0.374057398954935 | 1.0 |
| 2 | 0.060988545617122 | 0.448803604749845 | 0.177138996295218 | 0.374057398954937 | 1.0 |

Including the backtracking to find an optimal step length had no effect on the outcome of the Newtons Method optimization results. Other then increasing the computational time an insignificance amount, the algorithm never enters the backtracking calculation because the criteria is never met. The optimal solution speed for the given objective function is already met with a step length of 1 and no further optimization of the step length is useful.

**Steepest Descent Method - with Backtracking**

The above backtracking method was introduced into the Steepest Descent algorithm with the direction vector (13), the MATLAB code can be found in the appendix. The results of the iterations are tabulated.

$$d^k = -\nabla f(x^k) \tag{13}$$

Algorithm Design

1. Guess $x^0$, k = 0, $\alpha = 1$
2. while ||norm($x^{k+1}$) - norm($x^k$)|| > tolerance
3. $\quad x^{k+1} = x^k - \alpha \nabla f(x^k)$
4. $\quad$ while $f(x^{k+1}) > f(x^k) + \gamma\alpha \cdot d \cdot \nabla f(x^k)$ - (Wolfe Condition 4)
5. $\quad\quad$ if $d^k \cdot \nabla f(x^{k+1}) > \beta d^k \cdot \nabla f(x^k)$ - (Wolfe Condition 3)
6. $\quad\quad\quad \alpha = \alpha \cdot ADJ$
7. $\quad\quad\quad x^{k+1} = x_0 + \alpha \cdot d$
8. $\quad\quad\quad f(x^{k+1})$
9. $\quad\quad$ end if
10. $\quad$ end while
11. $\quad k = k + 1$
12. end while
13. return $x^{k+1}$

| Steepest Descent Method - with Backtracking | | | | | |
|---|---|---|---|---|---|
| Iterate | Lagrange | Asset 1 | Asset 2 | Asset 3 | $\alpha$ |
| 1 | 0.8000000000000 | -0.245525500000 | -0.444608500000 | -0.538262500000 | 0.500 |
| 2 | 1.91419825000 | -0.5767303552825 | -0.7976319890162 | -0.9058946290725 | 0.250 |
| 3 | 2.7342624933428 | -1.0115686052105 | -1.2482011706576 | -1.3678912464972 | 0.125 |
| 4 | 3.3127201211385 | -1.3254055293275 | -1.5731059620196 | -1.7011646176185 | 0.0625 |
| 5 | 3.6626998779488 | -1.5162800834070 | -1.7706705266717 | -1.9038708635349 | 0.03125 |
| 6 | 3.8561630489992 | -1.6219859861592 | -1.8800730290756 | -2.0161352948234 | 0.015625 |
| .. | .. | .. | .. | .. | .. |
| 25 | Stop | | | | |

It was expected that this gradient method with selection of an appropriate step length each iteration would converge. However, this was not the case. Although the step length was adjusted each iteration, the method di-verged. The steepest descent method is meant to find local minima, and being a multi-modal problem may have other minima. Using a global optimizer method such as Newtons or BFGS is a better option.

**BFGS Quasi-Newton Method - with Backtracking**
Again, backtracking was introduced into the BFGS Quasi-Newton method with the direction vector (14),

$$d^k = -\big[H_k\big]^{-1} \cdot \nabla f(x^k) \tag{14}$$

<u>Algorithm Design</u>

1. Guess $x^0$, k = 0, $H_0 = I(4)$
2. while ||norm($x^{k+1}$) - norm($x^k$)|| > tolerance
3. $\quad x^{k+1} = x^k - \alpha\big[H_{k+1}\big]^{-1}\nabla f(x^k)$
4. $\quad$ while $f(x^{k+1}) > f(x^k) + \gamma\alpha \cdot d \cdot \nabla f(x^k)$ - (Wolfe Condition 4)
5. $\qquad$ if $d^k \cdot \nabla f(x^{k+1}) > \beta d^k \cdot \nabla f(x^k)$ - (Wolfe Condition 3)
6. $\qquad\quad \alpha = \alpha \cdot ADJ$
7. $\qquad\quad x^{k+1} = x_0 + \alpha \cdot d$
8. $\qquad\quad f(x^{k+1})$
9. $\qquad$ end if
10. $\quad$ end while
11. $\quad S = x^{k+1} - x^k$
12. $\quad Y = \nabla f(x^{k+1}) - \nabla f(x^k)$
13. $\quad H_{k+1} = H_k + \frac{YY^T}{Y^TS} - \frac{H_kSS^TH_k}{S^TH_kS}$
14. $\quad k = k + 1$
15. end while
16. return $x^{k+1}$

| BFGS Quasi-Newton Method - with Backtracking | | | | | |
|---|---|---|---|---|---|
| Iterate | Lagrange | Asset 1 | Asset 2 | Asset 3 | $\alpha$ |
| 1 | 0.800000000000000 | 0.127237250000000 | -0.0723042500000 | -0.1691312500000 | 0.500 |
| 2 | 20.6544382951540 | -361.27709949981 | -361.42816807163 | -358.75923530179 | 0.250 |
| 3 | 10.3562253440309 | -180.51696614164 | -180.68506784178 | -179.39058324448 | 0.25 |
| 4 | 5.207049627673614 | -90.0151457801158 | -90.1920858504570 | -89.5828590675883 | 0.25 |
| 5 | 3.6626998779488 | -1.51628008340 | -1.77067052667 | -1.90387086353 | 0.25 |
| 6 | 1.345280056671624 | -22.1439876630759 | -22.3281377161176 | -22.2255219112861 | 0.25 |
| .. | .. | .. | .. | .. | .. |
| 49 | 0.060986345306614 | 0.448936099900986 | 0.176870636059063 | 0.375224940328196 | 0.125 |
| 50 | 0.060987182001102 | 0.448884229608179 | 0.176987388297122 | 0.374659028676965 | 0.125 |
| 51 | 0.060987767205310 | 0.448849132197818 | 0.177057362865749 | 0.374360111084511 | 0.125 |

The BFGS Quasi-Newton method appears to converge at iteration 51 in a time around 1.6 seconds. This method converges, unlike the steepest descent method, but it is slower then the original newtons method with quadratic function. Although it took a longer time to converge, as mentioned previously, the method does not require the calculation of complex matrix inversions. The addition of the backtracking line search did not improve the convergence efficiency over using s step length of 1. The decreasing step lengths slowed the convergence rate as is apparent in the increased number of iterations.

**(Part 4) Apply Optimization Methods - Scale Up**

Data was collected for a set of financial assets (stocks) utilizing python and Yahoo Finance. The following six assets were used, BTE, NVDA, TMO, LLY, JNJ and GOLD and the covariance matrix and expected returns are calculated and are as follows using the trailing 22 months of daily stock data.

The parameters, expected returns and covariances were determined as follows,

Daily Returns = $\frac{\text{Adjusted Close - Previous Adjusted Close}}{\text{Previous Adjusted Close}}$

Asset Expected Return = Average return of the daily returns for each asset over the 22 months.

Covariances - Covariance function used in Python with a data frame containing the 6 assets as columns along with the daily return for each day in the 22 month period.

| Covariance Matrix | | | | | | |
|---|---|---|---|---|---|---|
| $\sigma_{ij}$ | BTE | NVDA | TMO | LLY | JNJ | GOLD |
| BTE | 0.001284629 | -0.000048597 | -0.000026818 | -0.00003746 | -0.00002051 | 0.00004429 |
| NVDA | -0.000048597 | 0.000661936 | 0.000011103 | -0.000015712 | 0.00000499 | -0.00001052 |
| TMO | -0.000026818 | 0.000011103 | 0.000103117 | 0.000029431 | 0.00001932 | 0.00001798 |
| LLY | -0.000037468 | -0.000015712 | 0.000029431 | 0.000091503 | 0.00002321 | 0.00001160 |
| JNJ | -0.000020515 | 0.000004998 | 0.000019324 | 0.000023215 | 0.00005369 | 0.00000982 |
| GOLD | 0.000044292 | -0.000010528 | 0.000017985 | 0.000011602 | 0.00000982 | 0.00026926 |

| Expected Returns | | | | | |
|---|---|---|---|---|---|
| BTE | NVDA | TMO | LLY | JNJ | GOLD |
| -0.000873 | 0.003246 | 0.001282 | 0.000687 | 0.000927 | 0.000636 |

This data is used in the following variables and the objective function (3) is optimized using the three previous methods with backtracking. All MATLAB code can be found in the Appendix.

$$Q = \sigma_{ij} = \begin{bmatrix} 0.001284629 & -0.000048597 & -0.000026818 & -0.00003746 & -0.00002051 & 0.00004429 \\ -0.000048597 & 0.000661936 & 0.000011103 & -0.000015712 & 0.00000499 & -0.00001052 \\ -0.000026818 & 0.000011103 & 0.000103117 & 0.000029431 & 0.00001932 & 0.00001798 \\ -0.000037468 & -0.000015712 & 0.000029431 & 0.000091503 & 0.00002321 & 0.00001160 \\ -0.000020515 & 0.000004998 & 0.000019324 & 0.000023215 & 0.00005369 & 0.00000982 \\ 0.000044292 & -0.000010528 & 0.000017985 & 0.000011602 & 0.00000982 & 0.00026926 \end{bmatrix}$$

$$C = \mu_i = \begin{bmatrix} -0.000873 & 0.003246 & 0.001282 & 0.000687 & 0.000927 & 0.000636 \end{bmatrix}$$

**Newton Method Results**

The newtons method converged quickly in this model as expected. The weighting of the assets include both going long and short on the different assets. The step length was never adjusted in the backtracking due to the speed of convergence of the method already.

| Newton Method - with Backtracking | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Itr | Lagrange | BTE | NVDA | TMO | LLY | JNJ | GOLD | $\alpha$ |
| 1 | 0.00084567 | -0.33617 | 0.96433 | 1.2724 | -0.89852 | 0.18068 | -0.18271 | 1.0 |
| 2 | 0.00084567 | -0.33617 | 0.96433 | 1.2724 | -0.89852 | 0.18068 | -0.18271 | 1.0 |

**Steepest Descent Method Results**

As in the previous model, the SDM method with backtracking did not converge with this larger asset set. The determination of the dampening coefficient and beta would need to be further explored to determine if convergence could be met with better selections of input parameters.

**BFGS Quasi-Newton Results**

Without backtracking, the BFGS Quasi-Newton method converges with this asset model in 60 iterations at a time of 2.5 seconds. The results match the same convergence point as in the Newton's method as expected, but at a slower rate. The addition of the backtracking line search did not improve the efficiency of the convergence. After checking for each Wolfe condition each iteration, convergence was not seen until iteration 112 at 4.8 seconds. This indicates that a step length of 1 is still the better step length for this optimization method.

| BFGS Quasi-Newton Method - with Backtracking | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Itr | Lagrange | BTE | NVDA | TMO | LLY | JNJ | GOLD | $\alpha$ |
| 1 | -0.79 | 0.53648 | 0.04317 | -0.03877 | 0.64053 | -0.04911 | 0.36012 | 1.0 |
| 2 | -8.179 | 0.99192 | 0.56006 | 0.46014 | 1.133 | 0.44672 | 0.84888 | 1.0 |
| 3 | -0.00068 | 0.45465 | -0.03888 | -0.12105 | 0.55822 | -0.13141 | 0.27793 | 0.5 |
| .. | .. | .. | .. | .. | .. | .. | .. | 0.5 |
| 111 | 0.00085 | -0.33617 | 0.96433 | 1.2724 | -0.89852 | 0.18068 | -0.18271 | 0.5 |
| 112 | 0.00085 | -0.33617 | 0.96433 | 1.2724 | -0.89852 | 0.18068 | -0.18271 | 0.5 |

**Conclusions**

This computational project set out to compare three methods of optimization. It was determined that the newtons method, superior due to the quadratic objective function, functioned well with and without the additional backtracking line search for determining step length. The steepest descent method, although reliable under certain circumstances does not converge with given objective function even with the backtracking to determine the optimal step length. It is witnessed that the step length goes to zero with this method which results in a non-convergence. Lastly, the BFGS Quasi-Newton method converged with similar results to Newton's Method but was more inefficient with speed. With larger models, this is a more viable option due to not needing to calculate inverse hessian's.

The application of the optimization methods to the financial asset parameters and the objective function produced a set of weights for allocating resources to a specific asset to maximize risk-adjusted expected returns. The optimal quantity weight values was confirmed by two of the optimization techniques which focus on finding global minima of an objective function rather then local minima.

## APPENDIX - MATLAB CODE OUTPUT

### Multivariate Newton's Method - No Backtracking

```
%Initlialize Variables
format long
syms('X1','X2','X3','P');

%Formulated Function
Q = [0.02778,0.00387,0.00021; 0.00387,0.01112,-0.00020;
    0.00021,-0.00020,0.00115];
C = [0.1073; 0.0737; 0.0627];
X = [X1; X2; X3];
A = [1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);
H = hessian(F);

%Multi-Variate Newtons Method
step = 1;
X0 = [0.8; 0.5; 0.3; 0.2];

fprintf("NEWTONS METHOD \n")
fprintf("                P                    X1                    X2
                X3\n")
while (diff >= 0.00000001)

    %Multi-Variate Newton Iterate
    G1 = eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));
    H1 = eval(subs(H, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));
    XI = X0 - step * H1\G1;

    %Display Iterate
    disp(double(transpose(XI)))

    %Stopping Criteria
    diff = abs(norm(XI)-norm(X0));

    %Prepare Iterate
```

```
    X0 = XI;
    count = count + 1;

    if count > 100
        break
    end
end
```

**Steepest Descent Method - No Backtracking**

```
%Initlialize Variables
format long
syms('X1','X2','X3','P');

%Formulated Function
Q = [0.02778,0.00387,0.00021; 0.00387,0.01112,-0.00020;
    0.00021,-0.00020,0.00115];
C = [0.1073; 0.0737; 0.0627];
X = [X1; X2; X3];
A = [1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);
H = hessian(F);

% Steepest Descent Method Parameters
step = 1;
X0 = [0.8; 0.5; 0.3; 0.2];

fprintf("\n\nSTEEPEST DESCENT METHOD \n")
fprintf("                P                    X1                    X2
                X3\n")
while (diff >= 0.00000001)

    %Steepest Descent Iterate
    G1 = eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));
    XI = X0 - step * G1;

    %Display Iterate
    disp(double(transpose(XI)))

    %Stopping Criteria
```

```
    diff = abs(norm(XI)-norm(X0));

   %Prepare Iterate
   X0 = XI;
   count = count + 1;

   if count > 25
       break
   end
end
```

**BFGS Quasi-Newton Method - No Backtracking**

```
%Initilize Variables
syms('X1','X2','X3','P');
Q = [0.02778,0.00387,0.00021; 0.00387,0.01112,-0.00020;
    0.00021,-0.00020,0.00115];
C = [0.1073; 0.0737; 0.0627];
X = [X1; X2; X3];
A = [1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);

%Initial Points
H0 = eye(4);
X0 = [0.8; 0.5; 0.3; 0.2];
alpha = 1;

fprintf("\n\nBFGS QUASI NEWTON METHOD \n")
fprintf("        P                    X1                    X2
    X3\n")
while (diff >= 0.00000001)

    d = - H0\eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1),
        X0(4,1)}));
    XI = X0 + alpha*d;

    %Gradient at X0
    G1 = eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));
    G2 = eval(subs(G, {P, X1, X2, X3}, {XI(1,1), XI(2,1), XI(3,1), XI
        (4,1)}));
```

```

    %BFGS Inputs
    S = XI - X0;
    Y = G2 - G1;
    H1 = H0 + ((Y*transpose(Y))/(transpose(Y)*S)) - (H0*S*transpose(S)*
        H0)/(transpose(S)*H0*S);

    %Stopping Criteria
    diff = abs(norm(XI)-norm(X0));

    %Re-Assign new Values for Next Iterations
    H0 = H1;
    X0 = XI;
    count = count + 1;

   %Display Iterate
   disp(count)
   disp(double(transpose(XI)))

   if count > 50
       break
   end
end
```

**Multivariate Newton's Method - With Backtracking**

```
%Initlialize Variables
format long
syms('X1','X2','X3','P');

%Formulated Function
Q = [0.02778,0.00387,0.00021; 0.00387,0.01112,-0.00020;
    0.00021,-0.00020,0.00115];
C = [0.1073; 0.0737; 0.0627];
X = [X1; X2; X3];
A = [1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);
H = hessian(F);

%Backtracking
damp = 0.0005;
```

```
adj = 0.5;
beta = 0.75;

%Multi-Variate Newtons Method
step = 1;
X0 = [0.8; 0.5; 0.3; 0.2];

fprintf("NEWTONS METHOD \n")
fprintf("           P                   X1                   X2                   X3\n")
tic;
while (diff >= 0.00000001)

   %Multi-Variate Newton Iterate
   G1 = eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0(4,1)}));
   H1 = eval(subs(H, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0(4,1)}));

   %Direction Vector
   d = -H1\G1;

   %Calc Iterate
   XI = X0 + step * d;

   %Function Evaluation
   F1 = eval(subs(F, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0(4,1)}));
   F2 = eval(subs(F, {P, X1, X2, X3}, {XI(1,1), XI(2,1), XI(3,1), XI(4,1)}));

   %Backtracking
    while (F2 > F1 + damp*step*dot(d,G1))
        if(dot(d,G2) < beta*dot(d,G1))
            step = step * adj;
            XB = X0 + step*d;
            F2 = eval(subs(F, {P, X1,  X2, X3}, {XB(1,1), XB(2,1), XB(3,1), XB(4,1)}));
        else
            break
        end
    end

   %Display Iterate
   disp(double(transpose(XI)))

   %Stopping Criteria
   diff = abs(norm(XI)-norm(X0));
```

```
    %Prepare Iterate
    X0 = XI;
    count = count + 1;

    if count > 100
        break
    end
end
```

**Steepest Descent Method - With Backtracking**

```
%Initlialize Variables
clear
clc
format long
syms('X1','X2','X3','P');

%Formulated Function
Q = [0.02778,0.00387,0.00021; 0.00387,0.01112,-0.00020;
    0.00021,-0.00020,0.00115];
C = [0.1073; 0.0737; 0.0627];
X = [X1; X2; X3];
A = [1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

%Function Defintion
F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);
H = hessian(F);

%Backtracking
damp = 0.0005;
adj = 0.5;
beta = 0.75;

% Steepest Descent Method
step = 1;
X0 = [0.8; 0.5; 0.3; 0.2];

fprintf("\n\nSTEEPEST DESCENT METHOD \n")
fprintf("                P                X1                X2
                X3\n")
while (diff >= 0.00000001)
```

```
    %Gradient at X0
    G1 = eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));
    F1 = eval(subs(F, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));

    %Direction Vector
    d = -G1;

    %Steepest Descent Iterate
    XI = X0 + step * d;

    %Fuction at next point
    G2 = eval(subs(G, {P, X1, X2, X3}, {XI(1,1), XI(2,1), XI(3,1), XI
        (4,1)}));
    F2 = eval(subs(F, {P, X1, X2, X3}, {XI(1,1), XI(2,1), XI(3,1), XI
        (4,1)}));

    %Backtracking - Meet Wolfe Conditions 3 + 4
    while (F2 > F1 + damp*step*dot(d,G1))
        if(dot(d,G2) < beta*dot(d,G1))
            step = step * adj;
            XB = X0 + step*d;
            F2 = eval(subs(F, {P, X1,  X2, X3}, {XB(1,1), XB(2,1), XB
                (3,1), XB(4,1)}));
        else
            break
        end
    end

   %Display Iterate
   disp(step)
   disp(double(transpose(XI)))

   %Stopping Criteria
   diff = abs(norm(XI)-norm(X0));

   %Prepare Iterate
   X0 = XI;
   count = count + 1;

   if count > 100
       break
   end
end
```

**BFGS Quasi-Newton Method - With Backtracking**

```
%Initilize Variables
clear; clc;
syms('X1','X2','X3','P');
Q = [0.02778,0.00387,0.00021; 0.00387,0.01112,-0.00020;
    0.00021,-0.00020,0.00115];
C = [0.1073; 0.0737; 0.0627];
X = [X1; X2; X3];
A = [1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);

%Backtracking
damp = 0.0001;
adj = 0.5;
beta = 0.8;

%Initial Points
H0 = eye(4);
X0 = [0.8; 0.5; 0.3; 0.2];
step = 1;

fprintf("\n\nBFGS QUASI NEWTON METHOD \n")
fprintf("                    P                    X1                    X2
                    X3\n")
while (diff >= 0.00000001)

    %Direction Vector
    d = -H0\eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1),
        X0(4,1)}));
    XI = X0 + step*d;

    %Gradient at X0
    G1 = eval(subs(G, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));
    G2 = eval(subs(G, {P, X1, X2, X3}, {XI(1,1), XI(2,1), XI(3,1), XI
        (4,1)}));

    %BFGS Inputs
    S = XI - X0;
    Y = G2 - G1;
```

```
    H1 = H0 + ((Y*transpose(Y))/(transpose(Y)*S)) - (H0*S*transpose(S)*
        H0)/(transpose(S)*H0*S);

    %Function at Each Point
    F1 = eval(subs(F, {P, X1, X2, X3}, {X0(1,1), X0(2,1), X0(3,1), X0
        (4,1)}));
    F2 = eval(subs(F, {P, X1, X2, X3}, {XI(1,1), XI(2,1), XI(3,1), XI
        (4,1)}));
    F2 = F1;

    %Backtracking - Meet Wolfe Conditions 3 + 4
    while (F2 > F1 + damp*step*dot(d,G1))
        if(dot(d,G2) < beta*dot(d,G1))
            step = step * adj;
            XB = X0 + step*d;
            F2 = eval(subs(F, {P, X1,  X2, X3}, {XB(1,1), XB(2,1), XB
                (3,1), XB(4,1)}));
        else
            break
        end
    end

    %Stopping Criteria
    diff = abs(norm(XI)-norm(X0));

    %Re-Assign new Values for Next Iterations
    H0 = H1;
    X0 = XI;
    count = count + 1;

   %Display Iterate
  disp(step)
  disp(count)
   disp(double(transpose(XI)))

   if count > 50
       break
   end
end
```

**Multivariate Newton's Method - Scale Up**

```
%Initlialize Variables
clear; clc;
syms('X1','X2','X3','X4','X5','X6','P','F');

Q =    [   1.28462907e-03,   -4.85979426e-05,   -2.68182984e-05,
```

```
    -3.74681934e-05, -2.05155057e-05, 4.42920732e-05;
        -4.85979426e-05, 6.61936638e-04, 1.11035038e-05,
            -1.57122599e-05, 4.99880926e-06, -1.05281564e-05;
        -2.68182984e-05, 1.11035038e-05, 1.03117358e-04,
            2.94318841e-05, 1.93242416e-05, 1.79852927e-05;
        -3.74681934e-05, -1.57122599e-05, 2.94318841e-05,
            9.15032741e-05, 2.32157567e-05, 1.16020161e-05;
        -2.05155057e-05, 4.99880926e-06, 1.93242416e-05,
            2.32157567e-05, 5.36922808e-05, 9.82900409e-06;
         4.42920732e-05, -1.05281564e-05, 1.79852927e-05,
            1.16020161e-05, 9.82900409e-06, 2.69267858e-04];
C = [-0.0008727288011368683; 0.0032457521609619735;
    0.0012820815841300232; 0.0006872967456270 91; 0.0009273986806007257;
    0.0006356549214420348];
X = [X1; X2; X3; X4; X5; X6];
A = [1,1,1,1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);
H = hessian(F);

%Backtracking
damp = 0.0005;
adj = 0.5;
beta = 0.75;

%Multi-Variate Newtons Method
step = 1;
X0 = [0.8; 0.2; 0.1; 0.2; 0.1; 0.2; 0.2];

fprintf("NEWTONS METHOD \n")
fprintf("          P                    BTE                  NVDA
           TMO                  LLY                  JNJ
          GOLD\n")
while (diff >= 0.00000001)

    %Multi-Variate Newton Iterate
    G1 = eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1), X0
        (3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    H1 = eval(subs(H, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1), X0
        (3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    XI = X0 - step * H1\G1;

    d = -H1\G1;
```

```
    F1 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1), X0(3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    F2 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {XI(1,1), XI(2,1), XI(3,1), XI(4,1), XI(5,1), XI(6,1), XI(7,1)}));
    G2 = eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {XI(1,1), XI(2,1), XI(3,1), XI(4,1), XI(5,1), XI(6,1), XI(7,1)}));


    %Backtracking
     while (F2 > F1 + damp*step*dot(d,G1))
         if(dot(d,G2) < beta*dot(d,G1))
             step = step * adj;
             XB = X0 + step*d;
             F2 = eval(subs(F, {P, X1,  X2, X3}, {XB(1,1), XB(2,1), XB(3,1), XB(4,1)}));
         else
             break
         end
     end

    %Display Iterate
    disp(double(transpose(XI)))

    %Stopping Criteria
    diff = abs(norm(XI)-norm(X0));

    %Prepare Iterate
    X0 = XI;
    count = count + 1;

    if count > 100
        break
    end
end
```

**Steepest Descent Method - Scale Up**

```
%Initlialize Variables
clear
clc
format SHORT G
syms('X1','X2','X3','X4','X5','X6','P','F');

Q =    [   1.28462907e-03,   -4.85979426e-05,   -2.68182984e-05,
    -3.74681934e-05,   -2.05155057e-05,    4.42920732e-05;
           -4.85979426e-05,    6.61936638e-04,    1.11035038e-05,
               -1.57122599e-05,   4.99880926e-06,   -1.05281564e-05;
```

```
        -2.68182984e-05,   1.11035038e-05,   1.03117358e-04,   2.94318841e-05,   1.93242416e-05,   1.79852927e-05;
        -3.74681934e-05,  -1.57122599e-05,   2.94318841e-05,   9.15032741e-05,   2.32157567e-05,   1.16020161e-05;
        -2.05155057e-05,   4.99880926e-06,   1.93242416e-05,   2.32157567e-05,   5.36922808e-05,   9.82900409e-06;
         4.42920732e-05,  -1.05281564e-05,   1.79852927e-05,   1.16020161e-05,   9.82900409e-06,   2.69267858e-04];
C = [-0.000872728801136868; 0.00324575216096197; 0.00128208158413002; 0.000687296745627091; 0.000927398680600725; 0.000635654921442034];
X = [X1; X2; X3; X4; X5; X6];
A = [1,1,1,1,1,1];
b = 1;
DELTA = 4.0;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);

%Backtracking
damp = 0.005;
adj = 0.5;
beta = 0.75;

%Initial Points
H0 = eye(7);
X0 = [0.8; 0.2; 0.1; 0.2; 0.1; 0.2; 0.2];
step = 1;

fprintf("\n\nBFGS QUASI NEWTON METHOD \n")
fprintf("          P                    BTE                    NVDA                    TMO                    LLY                    JNJ                    GOLD\n")
while (diff >= 0.00000001)

    %Gradient at X0
    d = -eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1), X0(3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    XI = X0 + step*d;

    %Gradient at X0
    G1 = eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1), X0(3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    G2 = eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {XI(1,1), XI(2,1), XI(3,1), XI(4,1), XI(5,1), XI(6,1), XI(7,1)}));

```

```
    F1 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1),
        X0(3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    F2 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {XI(1,1), XI(2,1),
        XI(3,1), XI(4,1), XI(5,1), XI(6,1), XI(7,1)}));

   %Backtracking
    while (F2 > F1 + damp*step*dot(d,G1))
        if(dot(d,G2) < beta*dot(d,G1))
            step = step * adj;
            XB = X0 + step*d;
            F2 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {XI(1,1), XI
                (2,1), XI(3,1), XI(4,1), XI(5,1), XI(6,1), XI(7,1)}));
        else
            break
        end
    end

   disp(step)

   %Display Iterate
   disp(double(transpose(XI)))

   %Stopping Criteria
   diff = abs(norm(XI)-norm(X0));

   %Prepare Iterate
   X0 = XI;
   count = count + 1;


   if count > 50
       break
   end
end
```

**BFGS Quasi-Newton Method - Scale Up**

```
%Initlialize Variables
clear
clc
format SHORT G
syms('X1','X2','X3','X4','X5','X6','P','F','F1','F2');

Q =   [  1.28462907e-03,  -4.85979426e-05,  -2.68182984e-05,
    -3.74681934e-05,  -2.05155057e-05,   4.42920732e-05;
           -4.85979426e-05,   6.61936638e-04,  1.11035038e-05,
               -1.57122599e-05,  4.99880926e-06,  -1.05281564e-05;
```

```
            -2.68182984e-05,    1.11035038e-05,    1.03117358e-04,
                2.94318841e-05,    1.93242416e-05,    1.79852927e-05;
            -3.74681934e-05,   -1.57122599e-05,    2.94318841e-05,
                9.15032741e-05,    2.32157567e-05,    1.16020161e-05;
            -2.05155057e-05,    4.99880926e-06,    1.93242416e-05,
                2.32157567e-05,    5.36922808e-05,    9.82900409e-06;
             4.42920732e-05,   -1.05281564e-05,    1.79852927e-05,
                1.16020161e-05,    9.82900409e-06,    2.69267858e-04];
C = [-0.0008727288011368683; 0.003245752160961973 5;
    0.0012820815841300232; 0.0006872967456270 91; 0.0009273986806007257;
    0.0006356549214420348];
X = [X1; X2; X3; X4; X5; X6];
A = [1,1,1,1,1,1];
b = 1;
DELTA = 3.5;
diff = 1;
count = 0;

F = (DELTA/2)*transpose(X)*Q*X - transpose(C)*X + P*(A*X - b);
G = gradient(F);

%Backtracking
damp = 0.0005;
adj = 0.5;
beta = 0.75;

%Initial Points
H0 = eye(7);
X0 = [0.8; 0.2; 0.1; 0.2; 0.1; 0.2; 0.2];
step = 1;

tic;
fprintf("\n\nBFGS QUASI NEWTON METHOD \n")
fprintf("            P            BTE          NVDA            TMO            LLY
            JNJ            GOLD\n")
while (diff >= 0.00000001)

    d = -H0\eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1)
        , X0(3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    XI = X0 + step*d;

    %Gradient at X0
    G1 = eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1),
        X0(3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    G2 = eval(subs(G, {P, X1, X2, X3, X4, X5, X6}, {XI(1,1), XI(2,1),
        XI(3,1), XI(4,1), XI(5,1), XI(6,1), XI(7,1)}));

    %BFGS Inputs
```

```
    S = XI - X0;
    Y = G2 - G1;
    H1 = H0 + ((Y*transpose(Y))/(transpose(Y)*S)) - (H0*S*transpose(S)*
        H0)/(transpose(S)*H0*S);

    F1 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {X0(1,1), X0(2,1),
        X0(3,1), X0(4,1), X0(5,1), X0(6,1), X0(7,1)}));
    F2 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {XI(1,1), XI(2,1),
        XI(3,1), XI(4,1), XI(5,1), XI(6,1), XI(7,1)}));

    %Backtracking
    if (F2 > F1 + damp*step*dot(d,G1) )
        if(dot(d,G2) < beta*dot(d,G1))
            step = step * adj;
            XB = X0 + step*d;
            F2 = eval(subs(F, {P, X1, X2, X3, X4, X5, X6}, {XB(1,1), XB
                (2,1), XB(3,1), XB(4,1), XB(5,1), XB(6,1), XB(7,1)}));
        else
            break
        end
    end

    %Stopping Criteria
    diff = abs(norm(XI)-norm(X0));

    %Re-Assign new Values for Next Iterations
    H0 = H1;
    X0 = XI;
    count = count + 1;

   %Display Iterate
   disp(count)
   disp(step)
   disp( round(double(transpose(XI)),5) )


   if count > 150
       break
   end
end
toc;
```