Numerical Methods for Solving Nonlinear Systems in Portfolio Optimization

Yijun Fang

March 2023

1 Abstract

In this project, we are interested in solving a nonlinear system of equations in portfolio optimization that helps investors to make decisions about how to allocate their assets while managing risk. To solve the problem, we will use numerical methods such as fixed-point iteration, Newton's method, Quasi-Newton's method and steepest descent method for nonlinear equations. We will apply these methods to a real-world scenario, with information on two stocks, A and B, to provide practical insights into optimal asset allocation. Our study compares the performance of these methods, and also demonstrates the usefulness of numerical methods in the application of non-linear systems in finance.

2 Introduction

The optimization of assets in the portfolio has important applications in finance and investment management. It is a crucial task for investors and fund managers. The optimal portfolio allocation can help investors achieve their financial goals while minimizing the risk of losses.

In this project, we will study a non-linear system of equations that arises in portfolio optimization. The system is formulated in terms of n variables representing the weights allocated to n different assets in the portfolio. The problem is to find the optimal weights that reaches the portfolio's expected return while managing the risk to a given level. To apply the numerical methods we studied, such as fixed point iteration and Newton's methods to the problem, we limit the number of variable n to n = 3, as there's three constraints that could be written as equations in this system.

Let w_1, w_2, w_3 denotes the corresponding weight to the three different assets. Then, the expected return of the portfolio will be calculated as

$$R = w_1 R1 + w_2 R2 + w_3 R3$$

where $R1, R2, R3$ denotes the expected return of the three assets respectively. The portfolio risk will be calculated as

$$\sigma^2 = w_1^2 SD(R1) + w_2^2 SD(R2) + w_3^2 SD(R3)$$
$$+2w_1 w_2 * Cov(R_1, R_2) + 2w_1 w_3 * Cov(R_1, R_3) + 2w_2 w_3 * Cov(R_2, R_3)$$

Standard deviation (SD) of return measures the volatility of the asset's returns and is expressed in the same units as the asset's returns. Covariance Cov(A,B) is given by Correlation Coefficient x Standard Deviation of Asset A x Standard Deviation of Asset B.

The weights $w1, w2, w3$ will sum to 1, so

$$w1 + w2 + w3 = 1$$

We will write the above three equations in the form

$$\mathbf{F}(\mathbf{w}) = \mathbf{0}$$

where

$$f1(\mathbf{w}) = w_1R1 + w_2R2 + w_3R3 - R = 0$$

$$f2(\mathbf{w}) = w_1^2SD(R_1) + w_2^2SD(R_2) + w_3^2SD(R_3)$$

$$+2w_1w_2 * Cov(R_1, R_2) + 2w_1w_3 * Cov(R_1, R_3) + 2w_2w_3 * Cov(R_2, R_3) - \sigma^2 = 0$$

$$f3(\mathbf{w}) = w_1 + w_2 + w_3 - 1 = 0$$

and solve the non-linear equation.

There are various methods available in the literature to solve this problem, including analytical methods, numerical methods, and heuristic algorithms. Analytical methods can only be apply to the problem when it's not too numerically complex, and its solution will be exact. Heuristic algorithms, on the other hand, is very experience-based strategy and require a lot of trials, though it may be effective sometimes to get a non-exact solution. Numerical method is the technique we use to solve problems that cannot be approached analytically and to approximate the solution in a simpler way. This optimization of assets problem appears to be a effective application of numerical methods.

In this project, we focus on four numerical methods for solving the non-linear system of equations: Fixed Point Iteration, Newton's Method, Quasi-Newton's Method, and Steepest Descent. We will study the efficiency and robustness of these four methods in solving our nonlinear system. By comparing the performance of these methods on a set of test problems, we aim to provide insights into the relative strengths and weaknesses of different numerical methods for solving non-linear systems of equations in portfolio optimization.

3 Numerical Methods

3.1 Fix Point Method

Suppose F is a continuous function from $R^n \to R^n$ on D. The fixed-point method is an iterative numerical method used to find the root of a non-linear equation system $F(x) = \mathbf{0}$. To use the fixed-point method, we first transform the original equation system into an equivalent form $x = G(x)$, where G is also a continuous function from $R^n \to R^n$ on D. Then, the fixed-point method iteratively computes new approximations x_k using the formula:

$$x_{k+1} = G(x_k)$$

starting from the initial guess x_0 of our choice. If this sequence of approximations converges to a fixed point x^* of the function G, then x^* is a root of the original equation $F(x) = \mathbf{0}$.

To ensure that there exists a fixed point within the target domain $D \subseteq R^n$, we need to check whether $G(x) \in D$ for all $x \in D$. Additionally, if all $g_i(x)$ have continuous and partial derivatives and there exists a constant $\alpha < 1$ such that $\left|\frac{\partial g_i(x)}{\partial x_j}\right| < \frac{\alpha}{n}$, we can ensure that the iteration will converge to a unique fixed point $p \in D$.

Fixed point method:
Input: x_0, $G(\mathbf{x})$, maximum number of iterations, tolerance.
Algorithm: In each iteration, calculate vector $G(x_k)$ with x_k from last iteration (if first iteration, use x_0), and set new x_{n+1} equals to $G(x_k)$. Continue the iteration until the infinity norm error between new x_{k+1} and x_k is smaller than the tolerance. Stop the iterations when the number of iteration is larger than the maximum number of iteration. The resulting x will be the root of the original nonlinear equation system $F(x) = 0$.

3.2 Newton's Method

Newton 's method is a numerical method that can be used on both root finding and optimization. We will use it to find the root of a non-linear equation system of the form $F(x) = 0$. Newton's method is derived

from Taylor's expansion, $F(p) = F(x_k) + J_F(x_k)(p - x_k)$, where $J_F(x_k)$ is the Jacobian matrix of F at x_k. From this fact, we approximate p by using formula $x_{k+1} = x_k - [J_F(x_k)]^{-1}F(x_k)$, starting with an initial approximation x_0. Here, $[J_F(x_k)]^{-1}$ is the inverse of the Jacobian matrix. Newton's method computes the next approximation point of p by finding the intersection point of the tangent line to the F with the x-axis and take this intersection point as the next approximation x_{k+1}.

If the initial approximation x_0 is sufficiently close to the root of the equation system, and the Jacobian matrix is non-singular satisfied, then the Newton's method will converge to a root of the equation system quickly, usually quadratically. However, if the Jacobian matrix is singular in some iteration or the initial approximation is too off from the root, then the iteration may fail to converge or may converge slowly.

Newton's Method:
Input: x_0, maximum number of iterations, tolerance
Algorithm: First compute the Jacobian matrix J_F. In each iteration, calculate $J_F(x_k)$ with x_k from last iteration, (if first iteration, use x_0) and its inverse, then update x by $x_k + 1 = x_k - J_F(x_k)^-1 * F(x_k)$. Continue the iteration until the infinity norm error between new x_{k+1} and x_k is smaller than the tolerance. Stop the iterations when the number of iteration is larger than the maximum number of iteration. The resulting x will be the root of the original nonlinear equation system $F(x) = 0$.

3.3 Quasi-Newton's method

Quasi-Newton's method is a variant of Newton's method. It can also be used for finding the root of a non-linear equation system. Like Newton's method, it computes a sequence of approximations that converge to a root of the equation system. Compared to Newton's method, Quasi-Newton's method does not require the computation of the Jacobian matrix of F and its inverse at each iteration. Instead, it uses an approximation A_k to the Jacobian matrix. A_k satisfies $F(x_k) - F(x_{k-1}) = A_k(x_k - x_k - 1)$. The A_k^{-1} can be updated at each iteration based on A_{k-1}^{-1} without calculating inverse of a new matrix. The goal is to find x_k such that $F(x_{k-1}) + A_{k-1}(x_k - x_{k-1}) = 0$. Let $s_k = x_{k+1} - x_k$, $y_k = F(x_{k+1}) - F(x_k)$, Then, the update formula for A_k^{-1} is given by:

$$A_k^{-1} = A_{k-1}^{-1} + \frac{(s_k - A_{k-1}^{-1}y_k)s_k^T A_{k-1}^{-1}}{s_k^T A_{k-1}^{-1} y_k}$$

Use the formula of A_k^-1 to update x in the Quasi-Newton's method iteration:

$$x_{k+1} = x_k - A_k^{-1}F(x_k)$$

where $F(x_k)$ is the function value at x_k.

Quasi-Newton's method avoids the expensive computation of the Jacobian matrix, while still provides a fast convergence to a root of the equation system. It has similar order of convergence as the newton's method.

Quasi-Newton's method:
Input: x_0, $A_0^{-1} = J(x_0)^{-1}$, maximum number of iterations, tolerance; Algorithm: In each iteration, compute $x_k = x_{k-1} - A_{k-1}^{-1}F(x_{k-1})$ with A_{k-1}^{-1} from last iteration(if first iteration use the known A_0^{-1}). Then update $s_k = x_{k+1} - x_k$, $y_k = F(x_{k+1}) - F(x_k)$, and $A_k^{-1} = A_{k-1}^{-1} + \frac{(s_k - A_{k-1}^{-1}y_k)s_k^T A_{k-1}^{-1}}{s_k^T A_{k-1}^{-1} y_k}$. Continue the iteration until the infinity norm error between new x_{k+1} and x_k is smaller than the tolerance. Stop the iterations when the number of iteration is larger than the maximum number of iteration. The resulting x will be the root of the original nonlinear equation system $F(x) = 0$.

3.4 Steepest Descent

Suppose $g(x)$ is a continuous function that map from $R^n \to R$ and $g(x) = \sum(f_i^2(x))$, where f_i are the member functions of $F(x)$. If $x^(*)$ is the root of $F(x)$, then $x^(*)$ will be the minimizer of $g(x)$. We transform the root finding of non-linear system to minimization problem of $g(x)$. The steepest descent method can be

used to find the minimizer of $g(x)$. It works by iteratively updating the current approximation x_k using the gradient of the function g(x) evaluated at x_k. Specifically, the update formula is given by:

$$x_{k+1} = x_k + \alpha_k * (-\nabla g(x_k))$$

where α_k is the step size at iteration n, and $\nabla g(x_k)$ is the gradient of g evaluated at x_k. The step size α_k is chosen to minimize the function $g(x_{k+1})$ along the direction of the gradient. This is done by exact line search or inexact line search. The steepest descent method is guaranteed to converge to a local minimum of the function g(x) and thus find a root of F(x).

Steepest Descent:
Input: x_0, $g(x) = \sum f_i^2(x)$, s, t
Algorithm: In each iteration, compute $d_{k-1} = -\nabla g(x_{k-1})$ and $\beta =$ the square of two norm of $\nabla g(x_{k-1})$. Set $\alpha = 1$, update α with $s * \alpha$ until $g(x_{k-1}) - \alpha t\beta \geq g(x_{k-1} + \alpha d_{k-1})$. Set α_{k-1} to the resulting α. Update $x_k = x_{k-1} + \alpha_{k-1} d_{k-1}$. Continue the iteration until the infinity norm error between new x_{k+1} and x_k is smaller than the tolerance. Stop the iterations when the number of iteration is larger than the maximum number of iteration. The resulting x will be the root of the original nonlinear equation system $F(x) = 0$.

4 Numerical Tests

4.1 Non-Linear System A

Suppose we have three assets A, B, and C, with expected returns of 0.09, 0.06, and 0.08, respectively. The volatility or standard deviation of their returns are 0.04, 0.09, and 0.16, and the covariance between the three assets are as follows: Cov(A,B) = 0.02, Cov(B,C) = 0.05, Cov(A,C) = -0.03.

We want the expected return of this portfolio to be 0.07, while keeping the target risk of the portfolio to be 0.08.

As discussed earlier, we can construct a non-linear equation system as follows:

$$w_1^2 * 0.04 + w_2^2 * 0.09 + w_3^2 * 0.16 + 2 * w_1 * w_2 * 0.02 + 2 * w_1 * w_3 * (-0.03) + 2 * w_2 * w_3 * 0.05 - 0.08 = 0$$

$$0.09 * w_1 + 0.06 * w_2 + 0.08 * w_3 - 0.07 = 0$$

$$w_1 + w_2 + w_3 - 1 = 0$$

In the following sections, we will explore how fixed point method, Newton's method, Quasi-Newton's method and Steepest Descent method perform on this problem, and evaluate their advantage and disadvantage based on the result we got. Matlab will be used as the coding language for implementing the rooting finding methods.

4.1.1 Fixed Point Method for system A

As stated earlier, if we want to use fixed point iteration on the system, we need to make sure the fixed point function G(w) has fixed point in its domain. Let $w = [w1, w2, w3]^T$. After transforming F(w) = 0, we get

$$w = G(w) = \begin{bmatrix} 1 - w_2 - w_3 \\ \frac{0.07 - 0.08w_3 - 0.09w_1}{0.06} \\ \frac{0.08 - 0.04w_1^2 - 0.09w_2^2 - 0.04w_1w_2}{0.16w_3 - 0.06w_1 + 0.1w_2} \end{bmatrix}$$

where G(w) is a continuous function from $R^3 \to R^3$ on domain $D = \{0 \leq w1, w2, w3 \leq 1\}$. We failed to prove that $G(w) \in D$ for all $w \in D$, so fixed point iteration method may not apply to this problem, as its primary condition is not met. The Matlab program implementing fixed point iteration on this problem also shows that the method failed to converge on D, with an initial guess of $w_0 = [0.33, 0.33, 0.33]^T$, as shown in the graph. The inability to achieve convergence with the fixed point iteration method highlights

its limitations. For this method to be applicable in solving a nonlinear equation system F(x), the fixed point function G(w) must satisfy certain requirements.

```
x0 = [0.33,0.33,0.33];
max_iter = 10000;
iter = 0;
x = x0;
tol = 1e-6;
error = 10;

while error > tol && iter < max_iter

    x(1) = 1 - x0(2) - x0(3);
    x(2) = (0.07 - 0.08*x0(3) - 0.09*x0(1))/0.06;
    x(3) = (0.08-0.04*x0(1)^2-0.09*x0(2)^2-0.04*x0(1)*x0(2))/(0.16*x0(3)-0.06*x0(1)+0.1*x0(2));
    error = max([abs(x(1)-x0(1)),abs(x(2)-x0(2)),abs(x(3)-x0(3))]);
    iter = iter + 1;

    if error < tol
        fprintf('Solution found after %d iterations:\n', k);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x(1), x(2), x(3));
        break;
    end

    x0 = x;
end
fprintf("Failed to converge\n")
fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x(1), x(2), x(3));
```

```
Failed to converge
w1 = NaN
w2 = NaN
w3 = NaN
```

Figure 1: Fixed Point Method in Matlab

4.1.2 Newton's method

Jacobian matrix of this system $F(w)$ is

$$J_F = \begin{bmatrix} 0.08w_1 + 0.04w_2 - 0.06w_3 & 0.04w_1 + 0.18w_2 + 0.1w_3 & -0.06w_1 + 0.1w_2 + 0.32w_3 \\ 0.09 & 0.06 & 0.08 \\ 1 & 1 & 1 \end{bmatrix}$$

with initial guess $w_0 = [0.33, 0.33, 0.33]^T$, Newton's Method converge to the root of $F(w)$ in 5 iterations, as shown in Figure2 on next page.

The resulting root $w^{(*)} = [0.031152, 0.515576, 0.453271]^T$, and the error after 5 iteration is 1.171249e-07. The result indicates that the weights of assets A, B and C should be $w_1 = 0.031152, w_2 = 0.515576, w_3 = 0.453271$. Newton's method converged quickly and efficiently in this problem.

4.1.3 Quasi-Newton's method

In Quasi-Newton's Method, $A0^{-1}$ is needed as the input. We calculate it with

$$A_0^{-1} = J_F(w_0)^{-1} = \begin{bmatrix} -9.4697 & 6.2500 & 0.6250 \\ -4.7348 & -46.8750 & 4.3125 \\ 14.2045 & 40.6250 & -3.9375 \end{bmatrix}$$

where initial guess $w_0 = [0.33, 0.33, 0.33]^T$. Quasi-Newton's method converge to the root of F(w) in 7 iterations, as shown in Figure 3 on next page.

The resulting root $w^{(*)} = [0.031152, 0.515576, 0.453271]^T$. The error after 7 iterations is 5.348046e-08. It gets the same result as the Newton's method, yet it takes 2 more iterations to converge. This may be the trade-off for computational inexpensiveness of Quasi-Newton's method.

```
w0 = [0.33; 0.33; 0.33];


F = @(w) [w(1)^2*0.04 + w(2)^2*0.09 + w(3)^2*0.16 + 2*w(1)*w(2)*0.02 + 2*w(1)*w(3)*(-0.03) + 2*w(2)*w(3)*0.05 - 0.08;
          0.09*w(1) + 0.06*w(2) + 0.08*w(3) - 0.07;
          w(1) + w(2) + w(3) - 1];


J = @(x) [2*x(1)*0.04 + 2*x(2)*0.02 - 0.06*x(3), 2*x(1)*0.02 + 2*x(2)*0.09 + 2*x(3)*0.05, 2*x(1)*(-0.03) + 2*x(2)*0.05 + 2*x(3)*0.16;
          0.09, 0.06, 0.08;
          1, 1, 1];


tol = 1e-6;
max_iter = 100;


w = w0;
for i = 1:max_iter
    f = F(w);
    error = norm(f);
    if error < tol
        fprintf('Solution found after %d iterations:\n', i);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', w(1), w(2), w(3));
        fprintf('Error = %.6e\n', error);
        break;
    end
    J_inv = inv(J(w));
    w = w - J_inv*f;
end
```

```
Solution found after 5 iterations:
w1 = 0.031152
w2 = 0.515576
w3 = 0.453271
Error = 1.171249e-07
```

Figure 2: Newton's Method in Matlab

```
x0 = [0.33,0.33,0.33]';
F = @(x) [x(1)^2*0.04 + x(2)^2*0.09 + x(3)^2*0.16 + 2*x(1)*x(2)*0.02 + 2*x(1)*x(3)*(-0.03) + 2*x(2)*x(3)*0.05 - 0.08;
          0.09*x(1) + 0.06*x(2) + 0.08*x(3) - 0.07;
          x(1) + x(2) + x(3) - 1];
J = @(x) [2*x(1)*0.04 + 2*x(2)*0.02 - 0.06*x(3), 2*x(1)*0.02 + 2*x(2)*0.09 + 2*x(3)*0.05, 2*x(1)*(-0.03) + 2*x(2)*0.05 + 2*x(3)*0.16;
          0.09, 0.06, 0.08;
          1, 1, 1];
A0_inv = inv(J(x0));
disp(A0_inv)
k = 0;
error = 100;

x1 = x0;
N = 1000;
tol = 10^(-6);
while error >= tol && k <= N
    x0 = x1;
    F1 = F(x0);
    x1 = x0 - A0_inv*F1;
    s = x1-x0;
    Fx = F(x1);
    error = max([abs(x1(1)-x0(1)),abs(x1(2)-x0(2)),abs(x1(3)-x0(3))]);
    y = Fx-F1;
    A0_inv = A0_inv + ((s-A0_inv*y)*s'*A0_inv)/(s'*A0_inv*y);
    k = k+1;

    if error < tol
        fprintf('Solution found after %d iterations:\n', k);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x1(1), x1(2), x1(3));
        fprintf('Error = %.6e\n', error);
        break;
    end

end
```

```
   -9.4697    6.2500    0.6250
   -4.7348  -46.8750    4.3125
   14.2045   40.6250   -3.9375

Solution found after 7 iterations:
w1 = 0.031153
w2 = 0.515576
w3 = 0.453271
Error = 5.348046e-08
```

Figure 3: Quasi-Newton's Method in Matlab

4.1.4 Steepest Descent Method

Suppose $g(w)$ is a continuous function from $R^3 \to R$, and $g(w) = \sum f_i^2(w)$, where f_i are member functions of $F(w)$. We get

$$g(x) = (0.04w_1^2 + 0.09w_2^2 + 0.16w_3^2 + 0.04w_1w_2 - 0.06w_1w_3 + 0.05w_2w_3 - 0.08)^2 +$$
$$(0.09w_1 + 0.06w_2 + 0.08w_3 - 0.07)^2 + (w_1 + w_2 + w_3 - 1)^2$$

Root finding problem of F(w) can be transform to the minimization problem of g(w), with initial guess $w_0 = [0.33, 0.33, 0.33]^T$.We choose s = 0.2, t = 0.1 in the inexact line search for α_k. Steepest Descent Method converge to the minimizer of g(w), which is also the root of F(w) in 8430 iterations, as shown in Figure 4.

The resulting root $w^{(*)} = [0.032873, 0.509201, 0.457913]^T$, with error of 9.979576e-07. The result from steepest descent is slightly different with that from Newton's method and Quasi-Newton's method. It also takes significantly longer time and more iterations to converge. Steepest Descent method is not as efficient in this problem.

```
x0 = [0.3;0.3;0.4];
tolerance = 10^(-6);
g = @(w) (w(1)^2*0.04 + w(2)^2*0.09 + w(3)^2*0.16 + 2*w(1)*w(2)*0.02 + 2*w(1)*w(3)*(-0.03) + 2*w(2)*w(3)*0.05 ...
          - 0.08)^2 + ...
         (0.09*w(1) + 0.06*w(2) + 0.08*w(3) - 0.07)^2 + ...
         (w(1) + w(2) + w(3) - 1)^2;
grad = @(w) [2*(w(1)^2*0.04 + w(2)^2*0.09 + w(3)^2*0.16 + 2*w(1)*w(2)*0.02 + 2*w(1)*w(3)*(-0.03) + 2*w(2)*w(3)*0.05 ...
           - 0.08)*(0.08*w(1)+0.04*w(2)-0.06*w(3))+2*(0.09*w(1) + 0.06*w(2) + 0.08*w(3) - 0.07)*(0.09)+...
           2*(w(1) + w(2) + w(3) - 1); ...
             2*(w(1)^2*0.04 + w(2)^2*0.09 + w(3)^2*0.16 + 2*w(1)*w(2)*0.02 + 2*w(1)*w(3)*(-0.03) + 2*w(2)*w(3)*0.05 ...
           - 0.08)*(0.18*w(2)+0.04*w(1)+0.1*w(3))+2*(0.09*w(1) + 0.06*w(2) + 0.08*w(3) - 0.07)*(0.06)+...
           2*(w(1) + w(2) + w(3) - 1); ...
             2*(w(1)^2*0.04 + w(2)^2*0.09 + w(3)^2*0.16 + 2*w(1)*w(2)*0.02 + 2*w(1)*w(3)*(-0.03) + 2*w(2)*w(3)*0.05 ...
           - 0.08)*(0.32*w(3)-0.06*w(1)+0.1*w(2))+2*(0.09*w(1) + 0.06*w(2) + 0.08*w(3) - 0.07)*(0.08)+...
           2*(w(1) + w(2) + w(3) - 1)];

alpha_k = 1;
s = 0.2;
t = 0.1;

N = 10000;
k = 0;
x = x0;
error = 100;

while error >= tolerance && k <= N

    dg = -grad(x0);

    alpha = 1;
    xnew = x0 + alpha*dg;

    while g(xnew) > g(x0)- alpha*t*norm(-dg,2)^2
        alpha = alpha * s;
        xnew = x0 + alpha*dg;

    end

    alpha_k = alpha;
    x = xnew;


    error = max([abs(x(1)-x0(1)),abs(x(2)-x0(2)),abs(x(3)-x0(3))]);
    k = k+1;
    x0 = x;

    if error < tol
        fprintf('Solution found after %d iterations:\n', k);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x(1), x(2), x(3));
        fprintf('Error = %.6e\n', error);
        break;
    end
end
```

```
Solution found after 8430 iterations:
w1 = 0.032873
w2 = 0.509201
w3 = 0.457913
Error = 9.979576e-07
```

Figure 4: Steepest Descent Method in Matlab

4.2 Non-Linear System B

Suppose we want to invest in three real-world stocks Apple, P&G, and Amazon. They have expected returns of 16.14%, 7.67% and 15.4%. We use implied volatility to approximate the volatility or standard deviation of the stock's return. Apple has volatility of 25.77%, P&G has 19.33% and Amazon has 38.45%.

The correlation coefficient between Apple and P&G is -0.77, that between Apple and Amazon is 0.72, that between Amazon and P&G is -0.6, so we get Cov(Apple, P&G) = -0.0384, Cov(Apple, Amazon) = 0.0713, Cov(Amazon, P&G) = -0.0446.

we want the expected return of this portfolio to be 13%, while keeping the target risk of this portfolio to be 15%. Similar to the process in System A, we have F(w) = 0 as follows:

$$w_1^2*0.2577+w_2^2*0.1933+w_3^2*0.3845+2*w_1*w_2*(-0.0384)+2*w_1*w_3*(0.0713)+2*w_2*w_3*(-0.0446)-0.15=0$$

$$0.1614*w_1+0.0767*w_2+0.154*w_3-0.13=0$$

$$w_1+w_2+w_3-1=0$$

4.2.1 Fixed Point Method

Let $w=[w1,w2,w3]^T$. After transforming F(w) = 0, we get

$$w=G(w)=\begin{bmatrix} 1-w_2-w_3 \\ \frac{0.13-0.154w_3-0.1614w_1}{0.0767} \\ \frac{0.15-0.2577w_1^2-0.1933w_2^2-2*(-0.0384)w_1w_2}{0.3845w_3-2*(0.0713)w_1-2*(-0.0446)w_2} \end{bmatrix}$$

We still failed to prove that $G(w) \in D$. Matlab program cannot converge on D with initial guess of $w_0=[0.33,0.33,0.33]^T$.

```
x0 = [0.33,0.33,0.33];
max_iter = 10000;
iter = 0;
x = x0;
tol = 1e-6;
error = 10;

while error > tol && iter < max_iter

    x(1) = 1 - x0(2) - x0(3);
    x(2) = (0.13 - 0.154*x0(3) - 0.1614*x0(1))/0.0767;
    x(3) = (0.25-0.2577*x0(1)^2-0.1933*x0(2)^2-2*(-0.0384)*x0(1)*x0(2))/(0.3845*x0(3)-2*(0.0713)*x0(1)-2*(-0.0446)*x0(2));
    error = max([abs(x(1)-x0(1)),abs(x(2)-x0(2)),abs(x(3)-x0(3))]);
    iter = iter + 1;

    if error < tol
        fprintf('Solution found after %d iterations:\n', k);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x(1), x(2), x(3));
        break;
    end

    x0 = x;
end
fprintf("Failed to converge\n")
fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x(1), x(2), x(3));
```

```
Failed to converge
w1 = NaN
w2 = NaN
w3 = NaN
```

Figure 5: Fixed Point Method in Matlab

4.2.2 Newton's Method

Jacobian matrix of this system $F(w)$ is $J_F =$

$$\begin{bmatrix} 2(0.2677)w_1+2(-0.0284)w_2-2(0.0713)w_3 & 2(0.1933)w_2+2(-0.0384)w_1+2(-0.0446)w_3 & 2(0.3845)w_3+2(0.0713)w_1+2(-0.0446)w_2 \\ 0.1614 & 0.0767 & 0.154 \\ 1 & 1 & 1 \end{bmatrix}$$

with initial guess $w_0=[0.33,0.33,0.33]^T$, Newton's Method converge to the root of $F(w)$ in 11 iterations, as shown in Figure 6 on next page.

The resulting root $w^{(*)}=[0.075620,0.317718,0.606662]^T$, and the error after 11 iteration is 7.749665e-07. The result indicates that the weights of assets A, B and C should be $w_1=0.075620, w_2=0.317718, w_3=0.606662$.

```
w0 = [0.33; 0.33; 0.33];


F = @(w) [w(1)^2*0.2577 + w(2)^2*0.1933 + w(3)^2*0.3845 + 2*w(1)*w(2)*(-0.0384) + 2*w(1)*w(3)*(0.0713) + 2*w(2)*w(3)*(-0.0446) - 0.15;
          0.1614*w(1) + 0.0767*w(2) + 0.154*w(3) - 0.13;
          w(1) + w(2) + w(3) - 1];


J = @(x) [2*0.2577*x(1)+2*(-0.0284)*x(2)-2*(0.0713)*x(3), 2*(0.1933)*x(2)+2*(-0.0384)*x(1)+2*(-0.0446)*x(3),2*(0.3845)*x(3)+2*(0.0713)*x(1)+2*(-0.0446)*x(2);
          0.1614,0.0767,0.154;
          1, 1, 1];


tol = 1e-6;
max_iter = 100;


w = w0;
for i = 1:max_iter
    f = F(w);
    error = norm(f);
    if error < tol
        fprintf('Solution found after %d iterations:\n', i);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', w(1), w(2), w(3));
        fprintf('Error = %.6e\n', error);
        break;
    end
    J_inv = inv(J(w));
    w = w - J_inv*f;
end
```

```
Solution found after 11 iterations:
w1 = 0.075620
w2 = 0.317718
w3 = 0.606662
Error = 7.749665e-07
```

Figure 6: Newton's Method in Matlab

4.2.3 Quasi-Newton's Method

$$A_0^{-1} = J_F(w_0)^{-1} = \begin{bmatrix} -5.3728 & 13.8034 & -0.6676 \\ -0.5143 & -11.6152 & 1.9283 \\ 5.8871 & -2.1882 & -0.2607 \end{bmatrix}$$

where initial guess $w_0 = [0.33, 0.33, 0.33]^T$. Quasi-Newton's method converge to the root of F(w) in 7 iterations, as shown in Figure 7 on next page.

The resulting root $w^{(*)} = [0.075622, 0.317718, 0.606660]^T$. The error after 7 iterations is 5.201783e-10. In this problem, Quasi-Newton's method performs better than Newton's method. It reaches smaller error while taking less iterations.

4.2.4 Steepest Descent Method

$$g(x) = (w_1^2 * 0.2577 + w_2^2 * 0.1933 + w_3^2 * 0.3845 + 2 * w_1 * w_2 * (-0.0384) + 2 * w_1 * w_3 * (0.0713) + 2 * w_2 * w_3 * (-0.0446) - 0.15)^2 +$$
$$(0.1614w_1 + 0.0767w_2 + 0.154w_3 - 0.13)^2 + (w_1 + w_2 + w_3 - 1)^2$$

Root finding problem of F(w) can be transform to the minimization problem of g(w), with initial guess $w_0 = [0.33, 0.33, 0.33]^T$.We choose s = 0.2, t = 0.1 in the inexact line search for α_k.Steepest Descent Method converge to the minimizer of g(w), which is also the root of F(w) in 1451 iterations, as shown in Figure 8 on next page.

The resulting root $w^{(*)} = [0.076059, 0.317329, 0.606609]^T$, with error after 1451 iterations of 9.530946e-07. The result from steepest descent is slightly different with that from Newton's method and Quasi-Newton's method. It also takes significantly longer time and more iterations to converge. Steepest Descent method is not as efficient in this problem as well.

```
x0 = [0.33,0.33,0.33]';

F = @(w) [w(1)^2*0.2577 + w(2)^2*0.1933 + w(3)^2*0.3845 + 2*w(1)*w(2)*(-0.0384) + 2*w(1)*w(3)*(0.0713) + 2*w(2)*w(3)*(-0.0446) - 0.15;
          0.1614*w(1) + 0.0767*w(2) + 0.154*w(3) - 0.13;
          w(1) + w(2) + w(3) - 1];


J = @(x) [2*0.2577*x(1)+2*(-0.0284)*x(2)-2*(0.0713)*x(3), 2*(0.1933)*x(2)+2*(-0.0384)*x(1)+2*(-0.0446)*x(3),2*(0.3845)*x(3)+2*(0.0713)*x(1)+2*(-0.0446)*x(2);
          0.1614,0.0767,0.154;
          1, 1, 1];

A0_inv = inv(J(x0));
disp(A0_inv)
k = 0;
error = 100;

x1 = x0;
N = 1000;
tol = 10^(-6);
while error >= tol && k <= N
    x0 = x1;
    F1 = F(x0);
    x1 = x0 - A0_inv*F1;
    s = x1-x0;
    Fx = F(x1);
    error = max([abs(x1(1)-x0(1)),abs(x1(2)-x0(2)),abs(x1(3)-x0(3))]);
    y = Fx-F1;
    A0_inv = A0_inv + ((s-A0_inv*y)*s'*A0_inv)/(s'*A0_inv*y);
    k = k+1;

    if error < tol
        fprintf('Solution found after %d iterations:\n', k);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x1(1), x1(2), x1(3));
        fprintf('Error = %.6e\n', error);
        break;
    end

end
```

```
   -5.3728   13.8034   -0.6676
   -0.5143  -11.6152    1.9283
    5.8871   -2.1882   -0.2607

Solution found after 7 iterations:
w1 = 0.075622
w2 = 0.317718
w3 = 0.606660
Error = 5.201783e-10
```

Figure 7: Quasi-Newton's Method in Matlab

```
x0 = [0.3;0.3;0.4];
tolerance = 10^(-6);
g = @(w) (w(1)^2*0.2577 + w(2)^2*0.1933 + w(3)^2*0.3845 + 2*w(1)*w(2)*(-0.0384) + 2*w(1)*w(3)*(0.0713) + 2*w(2)*w(3)*(-0.0446) - 0.15)^2 + ...
          (0.1614*w(1) + 0.0767*w(2) + 0.154*w(3) - 0.13)^2 + ...
          (w(1) + w(2) + w(3) - 1)^2;
grad = @(w) [2*(w(1)^2*0.2577 + w(2)^2*0.1933 + w(3)^2*0.3845 + 2*w(1)*w(2)*(-0.0384)+ 2*w(1)*w(3)*(0.0713) + 2*w(2)*w(3)*(-0.0446) ...
           - 0.15)*(2*0.2577*w(1)+2*(-0.0384)*w(2)+2*(-0.0446)*w(3))+2*(0.1614*w(1) + 0.0767*w(2) + 0.154*w(3) - 0.13)*(0.1614)+...
           2*(w(1) + w(2) + w(3) - 1);
             2*(w(1)^2*0.2577 + w(2)^2*0.1933 + w(3)^2*0.3845 + 2*w(1)*w(2)*(-0.0384)+ 2*w(1)*w(3)*(0.0713) + 2*w(2)*w(3)*(-0.0446) ...
           - 0.15)*(2*0.1933*w(2)+2*(-0.0384)*w(1)+2*(-0.0446)*w(3))+2*(0.1614*w(1) + 0.0767*w(2) + 0.154*w(3) - 0.13)*(0.0767)+...
           2*(w(1) + w(2) + w(3) - 1);
             2*(w(1)^2*0.2577 + w(2)^2*0.1933 + w(3)^2*0.3845 + 2*w(1)*w(2)*(-0.0384)+ 2*w(1)*w(3)*(0.0713) + 2*w(2)*w(3)*(-0.0446)...
           - 0.15)*(2*0.3845*w(3)+2*(0.0713)*w(1)+2*(-0.0446)*w(2))+2*(0.1614*w(1) + 0.0767*w(2) + 0.154*w(3) - 0.13)*(0.154)+...
           2*(w(1) + w(2) + w(3) - 1)];

alpha_k = 1;
s = 0.2;
t = 0.1;

N = 10000;
k = 0;
x = x0;
error = 100;

while error >= tolerance && k <= N

    dg = -grad(x0);

    alpha = 1;
    xnew = x0 + alpha*dg;

    while g(xnew) > g(x0)- alpha*t*norm(-dg,2)^2
        alpha = alpha * s;
        xnew = x0 + alpha*dg;

    end

    alpha_k = alpha;
    x = xnew;


    error = max([abs(x(1)-x0(1)),abs(x(2)-x0(2)),abs(x(3)-x0(3))]);
    k = k+1;
    x0 = x;

    if error < tol
        fprintf('Solution found after %d iterations:\n', k);
        fprintf('w1 = %.6f\nw2 = %.6f\nw3 = %.6f\n', x(1), x(2), x(3));
        fprintf('Error = %.6e\n', error);
        break;
    end
end
```

```
Solution found after 1451 iterations:
w1 = 0.076059
w2 = 0.317329
w3 = 0.606609
Error = 9.530946e-07
```

Figure 8: Steepest Descent Method in Matlab

4.3 Conclusion and Evaluation

Our study included two non-linear system of equations A and B. System A is of R^3 and has imagery data, while System B uses real-life data from three stocks Apple, PG and Amazon. We compared four different methods for finding the roots of a nonlinear system of equations, including the fixed point method, Newton's method, Quasi-Newton's method, and the steepest descent method.

Our results have shown that the fixed point method failed to converge to a root in both cases, indicating that the application of fixed point method is more limited than the other three with its primary conditions. Newton's method and Quasi-Newton's method were both successful and efficient in finding a root, and their results tend to be rather similar in both cases. In System A, Newton's method uses less iterations, while in System B, Quasi-Newton's method uses less iteration as well as reaches a smaller error. The steepest descent method was the least efficient, requiring significantly more iterations to converge compared to the other methods. It took 8430 iterations for system A and 1451 iterations for system B.

For System A, the root of F(w) is found to be approximately [0.031, 0.516, 0.453]. Under target return 7% and target risk of 8%, the weights of assets A, B and C are $w_1 = 0.031, w_2 = 0.516, w_3 = 0.453$.
For System B, the root of F(w) is found to be approximately [0.076, 0.317, 0.607]. Under target return 17% and target risk of 15%, the weights of Apple, P&G and Amazon stock are $w_1 = 0.076, w_2 = 0.317, w_3 = 0.607$.
In conclusion, this study gives insights on the effectiveness of numerical methods on solving non-linear system of equations and the application in portfolio optimization field.

Table 1: Root Finding Methods for Nonlinear System of Equations A

Method	Number of Iterations	Root	Error
Fixed Point Method	Failed	NA	NA
Newton's Method	5	[0.031152, 0.515576, 0.453271]	1.171249e-07
Quasi-Newton's Method	7	[0.031152, 0.515576, 0.453271]	5.348046e-08
Steepest Descent Method	8430	[0.032873, 0.509201, 0.457913]	9.979576e-07

Table 2: Root Finding Methods for Nonlinear System of Equations B

Method	Number of Iterations	Root	Error
Fixed Point Method	Failed	NA	NA
Newton's Method	11	[0.075620, 0.317718, 0.606662]	7.749665e- 07
Quasi-Newton's Method	7	[0.075622, 0.317718, 0.606660]	5.201783e-10
Steepest Descent Method	1451	[0.076059, 0.317329, 0.606609]	9.530946e-07

References

[1] Chen, J. (2023, February 10). Expected return: Formula, how it works, limitations, example. Investopedia. Retrieved March 26, 2023, from https://www.investopedia.com/terms/e/expectedreturn.asp

[2] Dybek, M. (2022, October 29). Apple Inc. (nasdaq:AAPL): CAPM. Stock Analysis on Net. Retrieved March 26, 2023, from https://www.stock-analysis-on.net/NASDAQ/Company/Apple-Inc/DCF/CAPMExpected-Rate-of-Return

[3] PG - Procter Gamble Company Stock Price. Barchart.com. (n.d.). Retrieved March 26, 2023, from https://www.barchart.com/stocks/quotes/PG

[4] Sinra. (2023, January 21). What is portfolio risk, and how is it calculated? CFAJournal. Retrieved March 26, 2023, from https://www.cfajournal.org/portfolio-risk/