December 31, 2023

MSc Program – Academic Year 2023-2024

(Δ3) Optimization
Project 1 Report

Nefeli Eleftheria Sextou
Student ID: 503
E-Mail: pcs00503@uoi.gr

Dept. Of Computer Science and Engineering

University Of Ioannina

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Problem Description

1.1 Introduction

The goal of the present project is to assist an investor in deciding the percentage of capital that must be invested in each of four distinct commercial sectors in the USA: Computer Electronics and Goods Sales (1) , Defense Capital Goods Sales (2), Motor Vehicle Parts Sales (3) and Primary Metals Sales (4). This can be achieved by maximizing the expected relative performance of the investment while simultaneously minimizing the risk, which is expressed as the correlation of the performance values of the four sectors. In order to solve the portfolio optimization problem, three state-of-the-art gradient based optimization approaches are employed,examined and compared : BFGS with Wolfe Conditions Line Search, Dogleg BFGS and the Polak-Ribiere Conjugate Gradient Method.

1.2 Data Preprocessing

Four .csv files have been provided, each corresponding to a different commercial sector. They contain the monthly values (in $ million) of the serviced orders in the USA for the period between February 1992 and September 2023.

Considering February 1992 as the initial moment (the base month) of the temporal horizon, all given values can be converted to their respective relative performance value with regard to the base month as follows:

$$R_j^{(i)} = \frac{price\,in\,month\,j\,for\,sector\,K_i}{price\,in\,month\,1\,for\,sector\,K_i} \tag{1.1}$$

where $j = 1, 2, \ldots, 380$ and $i = 1, 2, 3, 4$

The vector of relative performance for a commercial sector is defined as :

$$R^{(i)} = [R_1^{(i)}, R_2^{(i)}, \ldots, R_{380}^{(i)}] \tag{1.2}$$

where $i = 1, 2, 3, 4$. It is utilized to extract the mean value of relative performance, which is described by the formula below:

$$\bar{R}_j^{(i)} = mean(R^{(i)}) = \frac{1}{380}\sum_{j=1}^{380} R_j^{(i)} \tag{1.3}$$

The means of relative performance of each distinct sector are then used to create the vector of mean relative performance:

$$\bar{R} = [\bar{R}^{(1)}, \bar{R}^{(2)}, \bar{R}^{(3)}, \bar{R}^{(4)}] \tag{1.4}$$

The relative performance vectors of each sector, are also necessary to define the covariance matrix of relative performance values:

$$M = \begin{bmatrix} cov(R^{(1)}, R^{(1)}) & cov(R^{(1)}, R^{(2)}) & cov(R^{(1)}, R^{(3)}) & cov(R^{(1)}, R^{(4)}) \\ cov(R^{(2)}, R^{(1)}) & cov(R^{(2)}, R^{(2)}) & cov(R^{(2)}, R^{(3)}) & cov(R^{(2)}, R^{(4)}) \\ cov(R^{(3)}, R^{(1)}) & cov(R^{(3)}, R^{(2)}) & cov(R^{(3)}, R^{(3)}) & cov(R^{(3)}, R^{(4)}) \\ cov(R^{(4)}, R^{(1)}) & cov(R^{(4)}, R^{(2)}) & cov(R^{(4)}, R^{(3)}) & cov(R^{(4)}, R^{(4)}) \end{bmatrix} \tag{1.5}$$

All of the above were implemented using Python's Pandas and Numpy libraries on a Jupyter Notebook which is provided along with the report and the rest of the implementation files (dataPreprocessing.ipynb).

1.3 The Objective Function

The desired outcome is the maximization of expected performance and the simultaneous minimization of the risk. Hence, the optimization problem that must be solved is the following:

$$\max_{w} f(w) = w^T \bar{R} - \lambda w^T M w \tag{1.6}$$

The term $w^T \bar{R}$ represents the performance while the term $w^T M w$ represents the risk. $w = (w_1, w_2, w_3, w_4)$ are the percentages of participation per commercial sector and $\lambda > 0$ is a parameter that models the importance of the risk. $\bar{R}$ and M are described by Eq.(1.4) and Eq.(1.5) respectively and $w_i \in [0.0, 1.0]$ with $i = 1, 2, 3, 4$.

The problem can be described as a minimization problem as well as provided in a more analytical form:

$$\min_{w} F(w) = -(\sum_{i=1}^{4} w_i \bar{R}^{(i)} - \lambda \sum_{j=1}^{4} \sum_{k=1}^{4} w_j w_k cov(R^{(j)}, R^{(k)})) \tag{1.7}$$

The participation percentages are bound by the constraint:

$$\sum_{i=1}^{4} w_i = 1 \tag{1.8}$$

Given that the integration of such a constraint is not presently feasible, each w_i may be expressed as:

$$w_i = \frac{x_i}{\sum_{j=1}^{4} x_j} \tag{1.9}$$

where $x_i \in [0.0, 1.0]$ and $i = 1, 2, 3, 4$.

Thus, the objective function $F(w)$ is transformed into a function of x_i $(i = 1, 2, 3, 4)$ that must be optimized within the search space $X \triangleq [0.0, 1.0]^4$. It is given that $\lambda = 1.5$.

As noted in the introduction, the methods that are employed are gradient based. Therefore, it is important to include information about any necessary partial derivations.

$F(w)$ expressed after the replacement of w_i in terms of x_i may be described as having two parts A and B, as described below:

$$\min_{w} F = -(A - \lambda B) \tag{1.10}$$

$$A = \frac{x_1 \bar{R}^{(1)} + x_2 \bar{R}^{(2)} + x_3 \bar{R}^{(3)} + x_4 \bar{R}^{(4)}}{\sum_{j=1}^{4} x_j} \tag{1.11}$$

B =

$$\frac{x_1^2}{(\sum_{j=1}^4 x_j)^2}cov(R^{(1)},R^{(1)})+\frac{x_1x_2}{(\sum_{j=1}^4 x_j)^2}cov(R^{(1)},R^{(2)})+\frac{x_1x_3}{(\sum_{j=1}^4 x_j)^2}cov(R^{(1)},R^{(3)})+\frac{x_1x_4}{(\sum_{j=1}^4 x_j)^2}cov(R^{(1)},R^{(4)})$$
$$+\frac{x_2x_1}{(\sum_{j=1}^4 x_j)^2}cov(R^{(2)},R^{(1)})+\frac{x_2^2}{(\sum_{j=1}^4 x_j)^2}cov(R^{(2)},R^{(2)})+\frac{x_2x_3}{(\sum_{j=1}^4 x_j)^2}cov(R^{(2)},R^{(3)})+\frac{x_2x_4}{(\sum_{j=1}^4 x_j)^2}cov(R^{(2)},R^{(4)})$$
$$+\frac{x_3x_1}{(\sum_{j=1}^4 x_j)^2}cov(R^{(3)},R^{(1)})+\frac{x_3x_2}{(\sum_{j=1}^4 x_j)^2}cov(R^{(3)},R^{(2)})+\frac{x_3^2}{(\sum_{j=1}^4 x_j)^2}cov(R^{(3)},R^{(3)})+\frac{x_3x_4}{(\sum_{j=1}^4 x_j)^2}cov(R^{(3)},R^{(4)})$$
$$+\frac{x_4x_1}{(\sum_{j=1}^4 x_j)^2}cov(R^{(4)},R^{(1)})+\frac{x_4x_2}{(\sum_{j=1}^4 x_j)^2}cov(R^{(4)},R^{(2)})+\frac{x_4x_3}{(\sum_{j=1}^4 x_j)^2}cov(R^{(4)},R^{(3)})+\frac{x_4^2}{(\sum_{j=1}^4 x_j)^2}cov(R^{(4)},R^{(4)})$$

(1.12)

Looking at Eq.(1.12), four distinct cases for partial derivation are observed within the sum. For the purposes of demonstration, only examples of the partial derivatives with regard to x_1 will be provided for each case. All similar terms are derived in the same manner.

The first case is the partial derivative of part A :

$$\frac{\partial A}{\partial x_1} = \frac{\bar{R}^{(1)}\sum_{j=1}^4 x_j - (x_1\bar{R}^{(1)}+x_2\bar{R}^{(2)}+x_3\bar{R}^{(3)}+x_4\bar{R}^{(4)})}{(\sum_{j=1}^4 x_j)^2} \tag{1.13}$$

The second case is found in B and corresponds to x_1^2 being in the numerator:

$$\frac{\partial x_1^2}{\partial x_1} = \frac{2*x_1(x_2+x_3+x_4)}{(\sum_{j=1}^4 x_j)^3} \tag{1.14}$$

The third case, found in B, corresponds to x_1 existing in the numerator, like in the chosen example:

$$\frac{\partial x_1x_2}{\partial x_1} = \frac{x_2(-x_1+x_2+x_3+x_4)}{(\sum_{j=1}^4 x_j)^3} \tag{1.15}$$

Finally, the fourth case is also found in B and corresponds to the scenario where x_1 does not exist in the numerator at all, just like in the following:

$$\frac{\partial x_2x_3}{\partial x_1} = \frac{-2x_2x_3}{(\sum_{j=1}^4 x_j)^3} \tag{1.16}$$

The full partial derivation with regard to x_1 is provided below. For legibility purposes it is noted in two parts, following Eq. (1.10) below. All partial derivations for the rest of the x_i are derived following the same scheme.

$$\frac{\partial F}{\partial x_1} = -(\frac{\partial A}{\partial x_1} - \lambda \frac{\partial B}{\partial x_1}) \tag{1.17}$$

$$\frac{\partial A}{\partial x_1} = \frac{\bar{R}^{(1)} \sum_{j=1}^{4} x_j - (x_1 \bar{R}^{(1)} + x_2 \bar{R}^{(2)} + x_3 \bar{R}^{(3)} + x_4 \bar{R}^{(4)})}{(\sum_{j=1}^{4} x_j)^2} \tag{1.18}$$

$$\begin{aligned}
\frac{\partial B}{\partial x_1} = \\
&\frac{2x_1(x_2+x_3+x_4)}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(1)}, R^{(1)}) + \\
&\frac{x_2(-x_1+x_2+x_3+x_4)}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(1)}, R^{(2)}) + \\
&\frac{x_3(-x_1+x_2+x_3+x_4)}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(1)}, R^{(3)}) + \\
&\frac{x_4(-x_1+x_2+x_3+x_4)}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(1)}, R^{(4)}) + \\
&\frac{x_2(-x_1+x_2+x_3+x_4)}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(2)}, R^{(1)}) + \\
&\frac{-2x_2x_2}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(2)}, R^{(2)}) + \\
&\frac{-2x_2x_3}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(2)}, R^{(3)}) + \\
&\frac{-2x_2x_4}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(2)}, R^{(4)}) + \\
&\frac{x_3(-x_1+x_2+x_3+x_4)}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(3)}, R^{(1)}) + \\
&\frac{-2x_3x_2}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(3)}, R^{(2)}) + \\
&\frac{-2x_3x_3}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(3)}, R^{(3)}) + \\
&\frac{-2x_3x_4}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(3)}, R^{(4)}) + \\
&\frac{x_4(-x_1+x_2+x_3+x_4)}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(4)}, R^{(1)}) + \\
&\frac{-2x_4x_2}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(4)}, R^{(2)}) + \\
&\frac{-2x_4x_3}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(4)}, R^{(3)}) + \\
&\frac{-2x_4x_4}{(\sum_{j=1}^{4} x_j)^3} cov(R^{(4)}, R^{(4)})
\end{aligned} \tag{1.19}$$

No second order derivatives are necessary since the methods that will be used approximate the Hessian matrix wherever it is required.

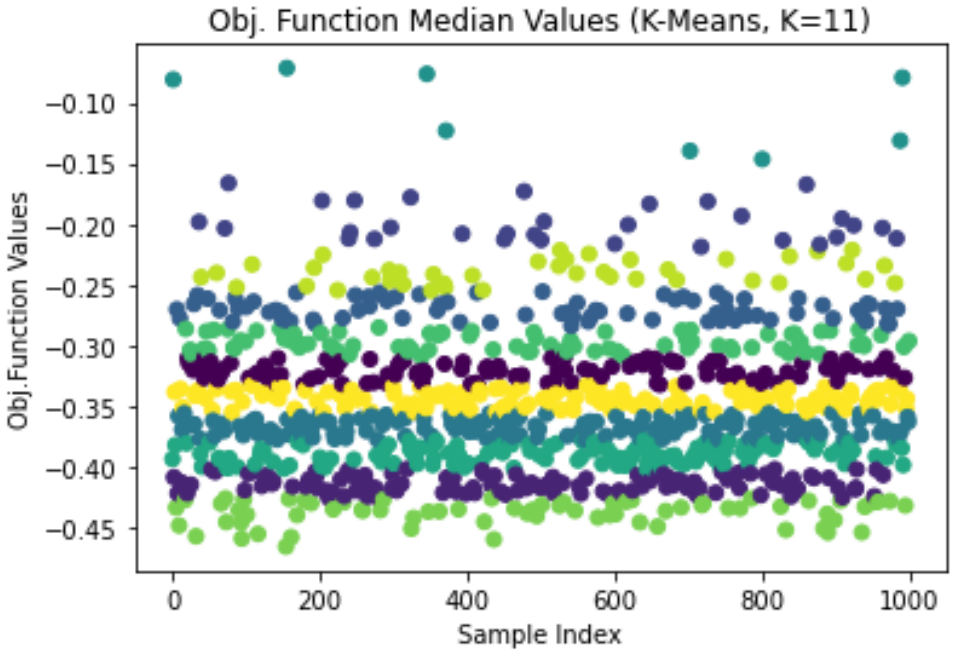

Figure 1.1: Obj. Function Medians - K-Means (k=11)

1.4 Objective Function Exploration

The search space is constrained in [0.0,1.0], so it may be useful to run the function multiple times to get an idea of what kind of values the objective function produces with random input. If the search space was wider, this may not provide any useful information. Thus, it must be highlighted that there is no guarantee that a value outside what is evident from this procedure does not exist. The result of the median values of 10 runs where 10000 samples are produced is provided in Fig1.1. The results were clustered in 11 groups using K-Means in order to attempt identifying possible areas that hold local minima. It is easily observed that the smallest values produced are in and around the area -0.4 to -0.45. However, a more detailed solution will be approximated with the methods presented in the chapter that follows to examine if a specific value stands out or even exists beyond the produced points.

Chapter 2

Optimization Algorithms

2.1 A Brief Introduction

This section is dedicated to the description of all algorithms and important subroutines implemented. The purpose of this chapter is to make clear all algorithmic details, parameterizations and any alterations or additions proposed to their main structure as it is described in well known bibliography.

Two variations of BFGS, a well established Quasi-Newton gradient-based optimization method are applied. This class of algorithms is notable for its exclusive dependence on gradient information, employing approximations of the inverse Hessian matrix to efficiently incorporate essential curvature information. The first variant of BFGS leverages a Line Search approach. This involves iteratively adjusting the step

size along the gradient direction, ensuring efficient convergence.The Wolfe Conditions Line Search BFGS algorithm dynamically adapts its step size, effectively balancing exploration and exploitation. The second variant utilizes a Trust Region method. Dogleg BFGS constrains the optimization process within a region where a quadratic model provides a reliable approximation. By carefully managing step sizes within this trust region, the algorithm achieves robust convergence and stability. The third algorithm applied is the Polak-Ribiere Conjugate Gradient method. This approach leverages gradient information combined with conjugate directions in each iteration, ensuring that the search directions are not only gradient-based but also orthogonal to the previous ones.

Algorithm 2.1 Zoom

Require: x_k ,p_k , $\phi(0)$, $\phi'(0)$,a_{low} ,a_{high}

while $True$ **do**
  $a_j = bisectionInterpolation(a_{low}, a_{high})$
  $Evaluate\, \phi(a_j) = f(x_k + a_j * p_k)$
  $Evaluate\, \phi(a_{low}) = f(x_k + a_{low} * p_k)$
  if $(\phi(a_j) \leq \phi(0) + c_1 a_j \phi'(0))$ **then**
    if $(|a_{low} - a_j| \leq 10^{-4})\, AND\, (|a_{high} - a_j| \leq 10^{-4})$ **then**
      $return\, a_j$
    end if
  end if
  if $(\phi(a_j) > \phi(0) + c_1 a_j \phi'(0))\, OR\, (\phi(a_j) > \phi(a_{low}))$ **then**
    $a_{high} = a_j$
  else
    $Evaluate\, \phi'(a_j) = f'(x_k + a_j * p_k)$
    if $|\phi'(a_j)| \leq -c_2 \phi'(0)$ **then**
      $return\, a_j$
    end if
    if $\phi'(a_j)(a_{high} - a_{low}) \geq 0$ **then**
      $a_{high} = a_{low}$
    end if
    $a_{low} = a_j$

2.2 Line Search with Wolfe Conditions

Before getting into the details of each of the three algorithms applied, it is important to briefly present crucial algorithms that are used in more than one of those methods.

Algorithm 2.2 Line Search

Require: x_k ,p_k , a_{max}

1: $a_0 = 0$
2: $consecutive = 0$
3: $Evaluate\, \phi(0i) = f(x_k + 0 * p_k)\ \phi'(0) = f'(x_k + 0 * p_k)$
4: **while** $True$ **do**
5: $\quad a_i = bisectionInterpolation(a_0, a_{max})$
6: $\quad Evaluate\, \phi(a_i) = f(x_k + a_i * p_k)\ \phi(a_{i-1}) = f(x_k + a_{i-1} * p_k)$
7: $\quad$ **if** $|\phi(a_i) - \phi(a_{i-1})|$ **then**
8: $\quad\quad consecutive = consecutive + 1$
9: $\quad$ **else**
10: $\quad\quad consecutive = 0$
11: $\quad$ **end if**
12: $\quad$ **if** $consecutive \geq 10$ **then**
13: $\quad\quad$ **if** $(\phi(a_i) \leq \phi(0) + c_1 a_i \phi'(0))$ **then**
14: $\quad\quad\quad return\, a_i$
15: $\quad\quad$ **end if**
16: $\quad$ **end if**
17: $\quad$ **if** $(\phi(a_i) > \phi(0) + c_1 a_i \phi'(0))\, OR\, (\phi(a_i) > \phi(a_{i-1}))$ **then**
18: $\quad\quad return\, Zoom(x_k, p_k, \phi(0), \phi'(0), a_{i-1}, a_i)$
19: $\quad$ **else**
20: $\quad\quad Evaluate\, \phi'(a_j) = f'(x_k + a_j * p_k)$
21: $\quad\quad$ **if** $|\phi'(a_i)| \leq -c_2 \phi'(0)$ **then**
22: $\quad\quad\quad return\, a_i$
23: $\quad\quad$ **end if**
24: $\quad\quad$ **if** $\phi'(a_i) \geq 0$ **then**
25: $\quad\quad\quad return\, Zoom(x_k, p_k, \phi(0), \phi'(0), a_i, a_{i-1})$
26: $\quad\quad$ **end if**

The first of these is the Line Search procedure that is employed in both the Line Search BFGS method and the Polak-Ribiere Conjugate Gradient method.

The line search algorithm implemented follows the state-of-the-art approach, as it is presented in Nocedal and Wright's Numerical Optimization [1]. It utilizes the Strong Wolfe Conditions. Two safeguard measures were added to ensure the routine returns an appropriate step size.

The first is within the "Zoom" subroutine, as presented in Alg.2.1. Due to numerical instability, it is possible that the curvature condition in line 14 does not hold and the routine enters an infinite loop where the upper and lower bound are extremely close to each other as well as the computed step size. To bypass this, a check for the distance of the bounds in comparison to the computed step is added in line 6. It

is also necessary for the Armijo condition to hold in order to ensure an acceptable step size that achieves sufficient decrease in the objective function evaluation. So, if the Armijo condition holds and the computed step size is too close to the upper and lower bounds, that step is returned.

The second safeguard is within the "Line Search" routine as described in Alg.2.2 (lines 7-16). It ensures that the procedures returns a value if the same value has been approximated over 10 consecutive times. For this value to be returned, again the Armijo condition must hold to ensure that sufficient decrease occurs.

It must also be noted that all interpolations are implemented using the Bisection Method.

Algorithm 2.3 Clamp X

Require: x_k ,x_{k+1} , x_{low} , x_{high}

1: $e = 10^{-10}$
2: **if** *any* x_i *in* $x \in [-e, e]$ *OR any* x_i *in* $x \in [1-e, 1+e]$ **then**
3: $return\ onBoundsClamp(x_{k+1}, x_{low}, x_{high})$
4: **else if** *any* x_i *in* $x \leq x_{low}$ *OR any* x_i *in* $x \geq x_{high}$ **then**
5: $z = bisectionLineClamp(x_k, x_{k+1}, x_{low}, x_{high})$
6: **for** z_i *in* z **do**
7: **if** $z_i \leq x_{low}$ **then**
8: $z_i = x_{low}$
9: **end if**
10: **if** $z_i \geq x_{high}$ **then**
11: $z_i = x_{high}$
12: **end if**
13: **end for**
14: $return\ z$
15: **else**
16: $return\ x_{k+1}$
17: **end if=0**

Algorithm 2.4 On Bounds Clamp

Require: x_{k+1} , x_{low} , x_{high}

1: $clampedVec = empty$
2: **for** $x_i in x_{k+1}$ **do**
3: **if** $(x_i \leq (x_{low} + e)\ AND\ x_i \geq (x_{low} - e))\ OR\ (x_i < x_{low})$ **then**
4: $clampedVec_i = x_{low}$
5: **else if** $(x_i \leq (x_{high} + e)\ AND\ x_i \geq (x_{high} - e))\ OR\ (x_i > x_{high})$ **then**
6: $clampedVec_i = x_{high}$
7: **else**
8: $clampedVec_i = x_i$
9: **end if**
10: **end for**
11: $return\ clampedVec$

Algorithm 2.5 Bisection Clamp

Require: $x_k,\ x_{k+1}, x_{low}, x_{high}, epsilon$

1: $a_k = x_{low}$
2: $b_k = x_{high}$
3: **while** $True$ **do**
4: $\quad t = 0.5 * (a_k + b_k)$
5: $\quad z = (1 - t)x_k + tx_{k+1}$
6: $\quad$ **if** $any\ z_i\ inz \leq x_{low}\ OR\ any\ z_i\ in\ z \geq x_{high}$ **then**
7: $\quad\quad b_k = t$
8: $\quad$ **else**
9: $\quad\quad a_k = t$
10: $\quad$ **end if**
11: $\quad$ **if** $(bk - ak) < epsilon$ **then**
12: $\quad\quad break$
13: $\quad$ **end if**
14: **end while**
15: $t = 0.5 * (a_k + b_k)$
16: $return\ (1 - t)x_k + tx_{k+1}$

2.3 Handling Values Outside The Search Space

The search space for the problem is strictly defined to be $[0.0, 1.0]^4$, but the optimization algorithms applied may produce points outside the search space. For this reason, it is necessary to implement a clamping procedure that ensures that the values are within bounds while loosing as little information as possible. This is all ensured by Alg.2.3. There exist three cases. The first case is encountered when the point is essentially on the bounds (give or take a small threshold). The second clamping scenario occurs when the point is out of bounds on at least one dimension. Finally the third scenario occurs when the point does not require clamping because it already is fully within the search space.

The first case (lines 2-3) is handled by the subroutine described in Alg.2.4 which clamps the out of bounds values within the vector to their nearest bound and leaves any within bounds components unchanged.

The second case (lines 4-14) on the other hand is slightly more complex and is handled by the procedure presented in Alg.2.5. In this scenario, the vector has at least one component which is out of bounds. It is required to not only bring values

Algorithm 2.6 Hessian Approximation Repair

Require: $square\ symmetric\ A = (a_{ij})\ , beta > 0\ k_{max}$

if $min_i\, a_{ij} > 0$ **then**
    $\tau_0 = beta$
else
    $\tau_0 = -min(a_i j) + beta$
end if
for $k\ in\ range\ (0, k_{max})$ **do**
    $B = A + \tau_k I$
    if $eigenvalues(B) \leq 0$ **then**
        $\tau_{k+1} = max(2\tau_k, beta)$
    else
        $return\ B$
    end if
end for

within bounds but to also retain directional information. This is tackled by applying a bisection procedure on a line defined by the current point x_k and the newly produced point x_{k+1}:

$$line_z = (1 - t)x_k + x_{k+1} \tag{2.1}$$

After the bisection procedure, some component may still be out of bounds, in this case, it is simply clamped to the nearest bound value.

2.4 Hessian and Inverse Hessian Approximation Repair

The BFGS methods employ updates for the Hessian and the inverse Hessian. Due to numerical instability, it is possible for these matrices to loose the property of positive definiteness which is of paramount importance for the numerical stability and convergence of these methods. To get past this issue, if it ever arises, a repair procedure is required as described in Alg.2.6. The method adds multiples of the identity matrix until positive definiteness holds. Positive definiteness is evaluated using the Eigenvalue criterion: if the eigenvalues of the matrix are all positive then the square symmetric matrix is positive definite. This is a necessary condition for positive definiteness. For the purposes of the present implementation, the parameters are set to $beta = 1$ and $k_{max} = 20$.

Algorithm 2.7 BFGS with Wolfe Conditions Line Search

Require: initial point x_0 , $\epsilon > 0$, $c1$, $c2$, a_{max}

$H_0 = I$
$k = 0$
$global\ f_{calls} = 0\ global\ gradient_{calls} = 0$
while $(||\nabla f_k|| > \epsilon)\ OR\ f_{calls} \leq 2000$ **do**
    $p_k = -H_k \nabla f_k$
    $a_k = lineSearch(xk, pk, c2, c2, epsilon, a_{max})$
    $x_{k+1} = x_k + a_k p_k$
    $x_{k+1} = clampX(x_k, x_{k+1}, x_{lo}, x_{hi}, e)$
    $s_k = x_{k+1} - x_k$
    $y_k = \nabla f_{k+1} - \nabla f_k$
    $H_{k+1} = BFGS_{update} H(H_k, s_k, y_k)$
    if $H_{k+1}\ positive\ definite$ **then**
        $pass$
    else
        $H_{k+1} = hessianRepair(H_{k+1}, beta = 1, k_{max} = 20)$
    end if
    $k = k + 1$
end while

2.5 The BFGS Method with Wolfe Conditions Line Search

The algorithm follows the scheme of a typical Line Search BFGS as defined in [1]. The pseudocode is presented in Alg.2.7. The method requires an initial point as input as well as a threshold value $\epsilon > 0$ necessary for the convergence check. A common value for $\epsilon = 10^{-4}$ was chosen. For the Wolfe Conditions Line Search (Alg.2.2), c_1 and c_2 were chosen to have typical for Quasi-Newton methods values of 10^{-4} and 0.9 respectively. a_{max} was tried and set equal to 1, a value that usually leads to stable step choices and avoids overshooting the minimum. The initial Hessian approximation matrix was set to the identity matrix which ensures the first p_k will be a descending direction (Steepest Descent step). The clamping procedure described in Section 2.3 is applied before a new approximation for x_{k+1} is used, to guarantee all x values produced remain within the search space. The inverse Hessian approximation is updated in accordance to the formula below:

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_K (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

(2.2)

where:

$$s_k = x_{k+1} - x_k \qquad y_k = \nabla f_{k+1} - \nabla f_k \qquad \rho_k = \frac{1}{y_k^T s_k}$$

A positive definiteness check and a repair procedure has also been added in accordance to Section 2.4. There are two termination criteria, the first is the l_2 norm reaching the set threshold and the second is for function calls to reach a limit of 2000.

2.6 The Dogleg BFGS Method

The main framework of Dogleg BFGS presented in Alg.2.8 is similar to the Wolfe Conditions Line Search BFGS , only there is no need to calculate an appropriate a_k step given this is a Trust Region method. Instead of Line Search assisting the descent process, the new direction p_k is approximated using the Dogleg Method as it is defined in [1] and described by Alg.2.9. Again ϵ is chosen to be equal to a common value of 10^{-4}. The initial trust region radius is set to the bisection interpolation point between 0 and the maximum trust region radius $\hat{\Delta} = 1$. The maximum trust region radius being 1 is a typical choice that ensures stability and balances exploration and exploitation. The acceptable p_k threshold is set to $h = 0.2$, requiring that an acceptable new step offers at least 20% reduction in the objective function approximation. The clamping procedure for x_{k+1} is also applied as described in Section 2.3 whenever the approximated p_k is accepted. Both hessian and inverse hessian approximations are initialized to the identity matrix. The hessian approximation is updated according to the following formula:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{2.3}$$

with:

$$s_k = x_{k+1} - x_k \qquad y_k = \nabla f_{k+1} - \nabla f_k \qquad \rho_k = \frac{1}{y_k^T s_k}$$

while the inverse hessian approximation is updated using Eq.2.5. The repair procedure and positive definiteness check is also applied to both, as described in Section

2.6. The termination criteria are the same as in the Wolfe Conditions Line Search method (Section 2.7).

Algorithm 2.8 Dogleg BFGS

Require: initial point x_0 , $\epsilon > 0$, $\Delta_0 \in (0, \hat{\Delta})$, $h \in [0, 1/4)$

$H_0 = I$
$B_0 = I$
$D_0 = 0.5(0 + \hat{\Delta})$
$k = 0$
$global\ f_{calls} = 0\ global\ gradient_{calls} = 0$
while $(||\nabla f_k|| > \epsilon)\ OR\ f_{calls} \leq 2000$ **do**
    $p_k = doglegMethod(H_k, B_k, \Delta_k, \nabla f_k)$
    $\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
    if $\rho_k < 1/4$ **then**
        $\Delta_{k+1} = (1/4)\Delta_k$
    else if $(\rho_k > 3/4)\ AND\ (||\rho_k|| = \Delta_k)$ **then**
        $\Delta_{k+1} = min(2\Delta_k, \hat{\Delta})$
    else
        $\Delta_{k+1} = \Delta_k$
    end if
    if $\rho_k > h$ **then**
        $x_{k+1} = x_k + a_k p_k$
        $x_{k+1} = clampX(x_k, x_{k+1}, x_l o, x_h i, e)$
    else
        $x_{k+1} = x_k$
    end if
    $s_k = x_{k+1} - x_k$
    $y_k = \nabla f_{k+1} - \nabla f_k$
    $H_{k+1} = BFGS_{update} H(H_k, s_k, y_k)$
    if $H_{k+1}\ positivedefinite$ **then**
        $pass$
    else
        $H_{k+1} = hessianRepair(H_{k+1}, beta = 1, k_{max} = 20)$
    end if
    $B_{k+1} = BFGS_{update} B(B_k, s_k, y_k)$
    if $B_{k+1}\ positive\ definite$ **then**
        $pass$
    else
        $B_{k+1} = hessianRepair(B_{k+1}, beta = 1, k_{max} = 20)$
    end if
    $k = k + 1$
    $k = k + 1$
end while

Algorithm 2.9 Dogleg Method

Require: $H_k, B_k, \Delta_k, \nabla f_k$

$p^B = -B^{-1}\nabla f_k$
if $||p^B|| \leq \Delta_k$ **then**
    $return\ p^B$
else
    $p^U = -\frac{\nabla^T f_k \nabla f_k}{\nabla^T f_k B_k \nabla f_k}\nabla f_k$
    if $||p^U|| \geq \Delta_k$ **then**
        $return\ -\frac{\Delta_k}{||\nabla f_k||}\nabla f_k$
    else
        $\tau^* = solve(||p^U + (\tau - 1)(p^B - p^U)||^2 = \Delta_k^2)$
        $return\ p^U + (\tau^* - 1)(p^B - p^U)$
    end if
end if

Algorithm 2.10 Polak-Ribiere Conjugate Gradient

Require: initial point x_0 , $\epsilon > 0$, $c1$, $c2$, a_{max}

$p_0 = \nabla f_k$
$k = 0$
$global\ f_{calls} = 0\, global\ gradient_{calls} = 0$
while $||\nabla f_k|| \neq 0$ **do**
    $a_k = lineSearch(xk, pk, c_1, c_2, epsilon, a_{max})$
    $x_{k+1} = x_k + a_k p_k$
    $\beta_{k+1}^{PR} = \frac{\nabla^T f_{k+1}(\nabla f_{k+1} - \nabla f_k)}{\nabla^T f_k \nabla f_k}$
    $\beta_{k+1} = max(\beta^P R_{k+1}, 0)$
    if $(k\, mod\, 50 = 0)\ AND\ (k > 1)$ **then**
        $p_{k+1} = -\nabla f_{k+1}$
    else
        $p_{k+1} = -\nabla f_{k+1} + p_k \beta_{k+1}$
    end if
end while

2.7 The Polak-Ribiere Conjugate Gradient Method

The Polak-Ribiere Conjugate Gradient Method follows the standard framework for non-linear conjugate gradient methods as defined in [1] and presented in Alg.2.10. It employs the Wolfe Conditions Line Search ,as presented in Section 2.2, with the following parameters, which are typical for use in conjunction with a conjugate gradient method:$c_1 = 10^{-4}$, $c_2 = 0.1$ and $a_{max} = 1$. Again, a_{max} is set to 1 given it is a safe option that leads to steps that won't overshoot the minimum. As input, the method requires an initial point and a threshold $\epsilon > 0$ to define an area that is around zero, because numerically it is unlikely the gradient's l_2 norm is exactly zero as is required by one of the termination criteria. ϵ is set equal to 10^{-6} because the convergence check is done with regard to equality to zero. It must be noted here that the second termination criterion is the same limitation of 2000 objective function evaluations imposed on the other two optimization algorithms. For each x_{k+1} update, it once again is necessary for the clamping procedure explained in Section 2.3 to be applied. Also, a restart for the descent direction p_k occurs every 50 iterations. The limit is set quite high for the algorithm to be be able to improve its search without being reset too often, given it the direction it moves remains descending. It must be highlighted that the restart's main purpose is to counteract accumulated approximation errors.

2.8 Implementation Details

All implementations have been coded in Python (Version 3.11.5) and executed on a laptop equipped with an Intel(R) Core(TM) i7-8550U CPU (@ 1.80GHz 1.99 GHz) and 8GB of RAM.
The corresponding .py files are:

1. Wolfe Conditions Line Search BFGS : ls_bfgs_portf_opti.py
2. Dogleg BFGS: dogleg_portf_opti.py
3. Polak-Ribiere Conjugate Gradient : polak_ribiere_portf_opti.py

Chapter 3

Experimental Results

3.1 Methodology

Starting Point Name	[x1,x2,x3,x4]
p1	[0.90362412, 0.96392195, 0.50211596, 0.69145483]
p2	[0.70834141, 0.05891796, 0.80561353, 0.64116189]
p3	[0.10062929, 0.56744996, 0.96195682, 0.03156802]
p4	[0.7172232, 0.10910059, 0.28863615, 0.96416363]
p5	[0.42583021, 0.95432923, 0.07941098, 0.28174186]
p6	[0.60384535, 0.11496978, 0.90015125, 0.91582732]
p7	[0.61422968, 0.16807904, 0.71007527, 0.66524447]
p8	[0.57583732, 0.14106864, 0.53758021, 0.53470457]
p9	[0.72509836, 0.19215341, 0.54516135, 0.02049204]
p10	[0.60436998, 0.66122049, 0.0072185 , 0.73981168]

Table 3.1: Starting Points Used for All Experiments

Each method was executed from each of the starting points presented in Table 3.1, using the same computational cost limit of 2000 function calls. It must be noted that the number of function evaluations is usually a few units higher because the implementations terminate when the corresponding counter surpasses the limit. The

results produced and examined include the x and w vectors, the corresponding objective function value in terms of x, the corresponding performance and risk values in terms of w as well as the total number of iterations, objective function calls and gradient calls.

3.2 Results

Starting Point	Obj.Func(x)	Performance(w)	Risk(w)
p1	-0.390887	0.528320	0.091623
p2	-0.448983	0.575054	0.084048
p3	-0.372072	0.611126	0.159370
p4	-0.420493	0.542257	0.081176
p5	-0.441074	0.595157	0.102722
p6	-0.448787	0.600478	0.101128
p7	-0.449004	0.587567	0.092375
p8	-0.445814	0.573797	0.085322
p9	-0.448112	0.559816	0.074469
p10	-0.403720	0.539115	0.090263
Median	**-0.443444**	**0.574426**	**0.090943**
St.Dev	**0.027102**	**0.0267397**	**0.022565**

Table 3.2: Line Search BFGS Results

Starting Point	x=[x1 x2 x3 x4]	w=[w1 w2 w3 w4]
p1	[0.94894294 0.63014737 0.90056231 1.0]	[0.27271198 0.18109491 0.25880811 0.28738501]
p2	[0.68778137 0.0 0.83334154 0.67527645]	[0.3131404 0.0 0.37941258 0.30744702]
p3	[0.12392547 0.52487964 1.0 0.09966587]	[0.07087648 0.30019351 0.57192828 0.05700173]
p4	[0.69535992 0.0 0.40531149 0.98393728]	[0.33356856 0.0 0.19443049 0.47200095]
p5	[0.58930683 0.09333869 0.97180559 1.0]	[0.22200704 0.03516309 0.36610416 0.37672572]
p6	[0.59353907 0.0 0.95005707 0.96627485]	[0.2364819 0.0 0.37852825 0.38498985]
p7	[0.58101303 0.0 0.79485417 0.75066191]	[0.27322129 0.0 0.37378005 0.35299865]
p8	[0.53567499 0.0 0.61623617 0.60574942]	[0.30476589 0.0 0.35060021 0.3446339]
p9	[0.6169619 0.0 0.70801602 0.27613888]	[0.38533222 0.0 0.44220136 0.17246642]
p10	0.67089938 0.26344951 0.51980445 1.0]	[0.27337305 0.10734843 0.21180602 0.4074725]

Table 3.3: Line Search BFGS Best x and w Values

The objective function results for the Wolfe Conditions Line Search BFGS implementation, and the corresponding x and w are presented in Table 3.2 and Table

3.3 respectively. The minimum value achieved is -0.449007 for starting point p7. This is equivalent to an investment with a performance percentage of 0.587567 and risk percentage equal to 0.092375. The matching vector w shows that the participation percentages are 27.322129% for sector 1 (Electronics), 0% for sector 2 (Defense), 37.378005% for sector 3 (Motor Vehicle Parts) and 35.299865% for sector 4 (Primary Metals).

Starting Point	Obj.Func(x)	Performance(w)	Risk(w)
p1	-0.443539	0.580465	0.091284
p2	-0.448983	0.575054	0.084048
p3	-0.372072	0.611126	0.159370
p4	-0.422314	0.544616	0.081534
p5	-0.447190	0.600655	0.102310
p6	-0.449069	0.601167	0.101399
p7	-0.449954	0.590061	0.093405
p8	-0.446004	0.574342	0.085559
p9	-0.453023	0.573935	0.080607
p10	-0.434364	0.572831	0.092311
Median	**-0.444772**	**0.5777597**	**0.091798**
St.Dev	**0.267309**	**0.018119**	**0.021935**

Table 3.4: Dogleg BFGS Results

Starting Point	x=[x1 x2 x3 x4]	w=[w1 w2 w3 w4]
p1	[0.55552752 0.0 0.6669045 0.78098617]	[0.27728984 0.0 0.33288332 0.38982684]
p2	[0.68778137 0.0 0.83334154 0.67527645]	[0.3131404 0.0 0.37941258 0.30744702]
p3	[0.12392547 0.52487965 1.0 0.09966587]	[0.07087648 0.30019351 0.57192828 0.05700173]
p4	[0.70369008 0.0 0.43417335 0.99138475]	[0.33048758 0.0 0.20390923 0.46560319]
p5	[0.46731002 0.0 0.74311421 0.80636904]	[0.23170943 0.0 0.36846325 0.39982732]
p6	[0.59288778 0.0 0.959329 0.96599723]	[0.23543979 0.0 0.3809561 0.38360411]
p7	[0.57974891 0.0 0.82232621 0.75281462]	[0.26903878 0.0 0.38160941 0.34935181]
p8	0.53818644 0.0 0.62366233 0.6103306]	[0.30368621 0.0 0.35191829 0.3443955]
p9	[0.62614232 0.0 0.81335612 0.36986844]	[0.34605603 0.0 0.44952526 0.20441871]
p10	[0.53509213 0.0 0.53346568 0.87183939]	[0.27576422 0.0 0.27492602 0.44930975]

Table 3.5: Dogleg BFGS Best x and w Values

The objective function results for the Dogleg BFGS implementation, and the corresponding x and w are presented in Table 3.4 and Table 3.5 respectively. The minimum value achieved is -0.453023 for starting point p9. This is equivalent to an investment

with a performance percentage of 0.573935 and risk percentage equal to 0.080607. The matching vector w shows that the participation percentages are 34.605603% for sector 1 (Electronics), 0% for sector 2 (Defense), 44.952526% for sector 3 (Motor Vehicle Parts) and $20.441871]\%$ for sector 4 (Primary Metals).

Starting Point	Obj.Func(x)	Performance(w)	Risk(w)
p1	-0.392705	0.529727	0.091347
p2	-0.448983	0.575054	0.084048
p3	-0.372072	0.611126	0.159370
p4	-0.420667	0.542496	0.081219
p5	-0.424368	0.574429	0.100041
p6	-0.448797	0.600506	0.101140
p7	-0.449046	0.587698	0.092434
p8	-0.446092	0.574578	0.085657
p9	-0.450195	0.565374	0.076786
p10	-0.399298	0.534819	0.090347
Median	**-0.4352299**	**0.574504**	**0.0908474**
St.Dev	**0.0270610**	**0.025763**	**0.022275**

Table 3.6: C.J Polak-Ribiere Results

Starting Point	x=[x1 x2 x3 x4]	w=[w1 w2 w3 w4]
p1	[0.93987165 0.60707979 0.89826982 1.0]	[0.27280444 0.17620923 0.26072921 0.29025712]
p2	[0.68778137 0.0 0.83334154 0.67527645]	[0.3131404 0.0 0.37941258 0.30744702]
p3	[0.12392547 0.52487965 1.0 0.09966586]	[0.07087648 0.30019351 0.57192828 0.05700173]
p4	[0.6965958 0.0 0.40832771 0.98550794]	[0.33323064 0.0 0.19533178 0.47143758]
p5	[0.65429715 0.28068203 0.96324778 1.0]	[0.22575773 0.09684612 0.33235761 0.34503854]
p6	[0.59370131 0.0 0.95072335 0.96662353]	[0.23643565 0.0 0.37861613 0.38494822]
p7	[0.58189582 0.0 0.7975144 0.75227533]	[0.27297451 0.0 0.37412385 0.35290164]
p8	[0.53915876 0.0 0.62681574 0.61200498]	[0.3032424 0.0 0.35254385 0.34421375]
p9	[0.619359 0.0 0.7438527 0.31468172]	[0.36912893 0.0 0.44332536 0.18754571]
p10	[0.68293517 0.30094772 0.51076813 1.0]	[0.2737598 0.1206372 0.20474532 0.40085767]

Table 3.7: C.J Polak-Ribiere Best x and w Values

Finally, the objective function results for the Polak-Ribiere Conjugate Gradient implementation, and the corresponding x and w are presented in Table 3.6 and Table 3.7 respectively. The minimum value achieved is -0.450195 for starting point p9. This is equivalent to an investment with a performance percentage of 0.565374 and risk percentage equal to 0.076786. The matching vector w shows that the participation

percentages are $.36.912893\%$ for sector 1 (Electronics), 0% for sector 2 (Defense), 44.332536% for sector 3 (Motor Vehicle Parts) and 18.754571% for sector 4 (Primary Metals).

Line Search BFGS	Dogleg BFGS	C.J Polak-Ribiere
666.0/2001.5/2002.5	500.0/2004.0/1002.0	111.5/2016.0/1906.0

Table 3.8: Median Iterations/Function Calls/Gradient Calls

Before comparing the implementations to each other, it is must be highlighted that all executions were terminated because they reached the limit of 2000 function calls. This wasn't affected by parameter changes or by an increase of the limit from 2000 up to 200000. Therefore, it is possible that it is a result of the initial points that were randomly chosen, the result of numerical instability, or it is related to characteristics of the function itself that may render these gradient based methods an unideal choice for the problem at hand. For all methods, the most major improvements were observed to occur in the first few iterations, with very small improvements following.

However, it must be noted that the values found in most cases aren't necessarily bad solutions. The characterizations of their quality depends on the level of accuracy the investor requires to make a decision. The best outcomes of all three methods indicate that a choice of investment within the ranges defined by the values of x of the best outcomes of each method, optimized using one of the methods, should deliver an investment with returns over or around 56% and risk less than or equal to about 10%. This was experimented with by running the methods for starting points with values for the components of x within the following bounds (uniformly chosen):

$$x_1 \in [0.58, 0.63]\ x_2 \in [0.0, 0.0]\ x_3 \in [0.74, 0.82]\ x_4 \in [0.31, 0.75] \tag{3.1}$$

Furthermore, the ranges that can be defined for w:

$$w_1 \in [0.27, 0.37]\ w_2 \in [0.0, 0.0]\ w_3 \in [0.37, 0.45]\ w_4 \in [0.19, 0.35] \tag{3.2}$$

provide direct performance values roughly around 60% to 70% but with an increase in risk around 5% to 10%. If those percentage ranges are something the investor is comfortable working with, then they can be utilized with no further optimization information being required besides the results provided in this report.

A final comparison note needs to be made in terms of the iterations, function calls and gradient calls results, as presented in Table 3.8, which contains the median

values of the executions of each method. The least expensive of the three methods is the Polak-Ribiere Conjugate Gradient method. Dogleg BFGS comes in second place and Line Search BFGS is the most expensive choice. Nevertheless, the result of the method is returned with no perceptible wait time, so either method may be chosen in terms of computational cost for this specific problem. Wait time generally becomes notable but still insignificant when setting the function call limit to values akin to $100k$.

Taking into account all of the above, depending on the needs of the investor, the methods applied and the results presented in this report may provide a good solution to the portfolio optimization problem and other crucial information for decision making. The quality of the solution can be further corroborated by looking at the overall values the objective function produces at random in Chapter1 on Fig.1.1. There, it is evident that the smallest function values tend to lie roughly around -0.4 to -0.45. Of course there is no guarantee that a smaller value does not lie outside that range but it is a helpful indication, given the search space is relatively small. Good solutions may overall be considered to be around the -0.44 to -0.45. All methods provide such solutions from multiple random starting points.

Bibliography

[1] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. [Online]. Available: https://books.google.gr/books?id=VbHYoSyelFcC