Optimization for Data Science

EXAMINATION PROJECT REPORT

Estimating the Matrix Norm $\|A\|_2$ using Gradient Descent and Quasi-Newton Method

Hafiz Muhammad Umer (667081)
Nimra Nawaz (667573)

ACADEMIC YEAR 2023-2024

Contents

1 Introduction

(P) is the problem of estimating the matrix norm $\|A\|_2$ for a (possibly rectangular) matrix $\boldsymbol{A} \in \mathbb{R}^{mxn}$, using its definition as (unconstrained) maximum problem.
(A1) is a standard gradient descent (steepest descent) approach.
(A2) is a quasi-Newton method such as BFGS or L-BFGS.

We introduce the definition of the 2-norm induced by the matrix:

$$\|\boldsymbol{A}\|_2 := \sup_{\boldsymbol{x} \in \mathbb{R}^n_{\neq 0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \tag{1.1}$$

where the 2-norm is defined as:

$$\|\boldsymbol{x}\|_2 := \sqrt{\boldsymbol{x}^T\boldsymbol{x}} \tag{1.2}$$

and $\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{x} \in \mathbb{R}^n_{\neq 0}$.

By merging together the previous definitions, we can further explain the equation for $\|\boldsymbol{A}\|_2$ with:

$$\|\boldsymbol{A}\|_2 = \sup_{\boldsymbol{x} \in \mathbb{R}^n_{\neq 0}} \sqrt{\frac{(\boldsymbol{A}\boldsymbol{x})^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}} = \sup_{\boldsymbol{x} \in \mathbb{R}^n_{\neq 0}} \sqrt{\frac{\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}} \tag{1.3}$$

However, since the square root always increases, due to its monotonous nature, and if h also always increases, we can express the $\sup_x h(g(x))$ as $h(\sup_x g(x))$, and now the problem can be expressed as:

$$\|\boldsymbol{A}\|_2 = \sup_{\boldsymbol{x} \in \mathbb{R}^n_{\neq 0}} \frac{\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} \tag{1.4}$$

to express the estimation of the norm as the following unconstrained minimization problem:

$$\|\boldsymbol{A}\|_2 = \inf_{\boldsymbol{x} \in \mathbb{R}^n_{\neq 0}} -f(\boldsymbol{x}) \tag{1.5}$$

Ultimately, we can define our objective function as:

$$f(\boldsymbol{x}) = -\frac{\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} \tag{1.6}$$

2 Properties of the Function

2.1 Properties of the Matrix

Given a matrix $\boldsymbol{A}$ of size $\boldsymbol{m}$ x $\boldsymbol{n}$, the following properties hold true for the matrices $\boldsymbol{A^TA}$ and $\boldsymbol{AA^T}$:

- The matrix is a square matrix: $\boldsymbol{A}^T\boldsymbol{A} \in \mathbb{R}^{n\times n}$
- The matrix is symmetric: $(\boldsymbol{A}^T\boldsymbol{A})^T = \boldsymbol{A}^T(\boldsymbol{A}^T)^T = \boldsymbol{A}^T\boldsymbol{A}$
- The matrix is positive semi-definite for any $\boldsymbol{x} \in \mathbb{R}^n$: $\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}\boldsymbol{x})^T\boldsymbol{A}\boldsymbol{x} = \|\boldsymbol{A}\boldsymbol{x}\|_2 \geq 0$

We know that this matrix has only real, non-negative eigenvalues since $\boldsymbol{A}^T\boldsymbol{A}$ is positive semi-definite and the Spectral Theorem describes how to decompose the eigenvalues of symmetric matrices.
In the end, the following describes the relationship between the 2-norm of matrix $\boldsymbol{A}$ and the spectral radius of $\boldsymbol{A}^T\boldsymbol{A}$, which is the biggest eigenvalue of that matrix:

$$\|\boldsymbol{A}\|_2 = \sqrt{\rho(\boldsymbol{A}^T\boldsymbol{A})} \tag{2.1}$$

2.2 Gradient and Complexity of the Objective Function

To compute the partial derivatives of the function at each step $\boldsymbol{i}$ and to find the gradient of the objective function at each iteration, we have expressed it in a closed form, which can be written as:

$$f(\boldsymbol{x}) = -\frac{\boldsymbol{x^T M x}}{\boldsymbol{x^T x}} = -\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} x_i a_{ij} x_j}{\sum_{i=1}^{n} x_i^2} \tag{2.2}$$

where a_{ij} are the elements of $\boldsymbol{M = A^T A}$. The complexity of the above operation is $\boldsymbol{O(mn^2)}$. Fortunately, we only need to compute M only once, which significantly reduces the computational cost, especially when dealing with a highly asymmetric matrix where $\boldsymbol{m}$ is much larger than $\boldsymbol{n}$.

The complexity of $\boldsymbol{f(x)}$ is $\boldsymbol{O(n^2)}$ because $\boldsymbol{x^T Q}$ is $\boldsymbol{O(n^2)}$ (we can compute $\boldsymbol{Qx}$ first with the same result), $\boldsymbol{(x^T Q)}$ is also $\boldsymbol{O(n^2)}$, $\boldsymbol{x^T x}$ is $\boldsymbol{O(n^2)}$, and the division is $\boldsymbol{O(1)}$

$$\boldsymbol{C(f(x)) = 2O(n^2) + O(n) + O(1) = O(n^2)} \tag{2.3}$$

We can obtain the partial derivatives at all points except at 0.

$$\begin{aligned}\frac{\partial f}{\partial x_k} &= -\frac{(\boldsymbol{x^T M x})'\boldsymbol{(x^T x)} - (\boldsymbol{x^T M x})\boldsymbol{(x^T x)'}}{(\boldsymbol{x^T x})^2} \\ &= -\frac{\boldsymbol{(2Mx)(x^T x)} - (\boldsymbol{x^T M x})2x}{(\boldsymbol{x^T x})^2} \\ &= \frac{2x(\boldsymbol{x^T M x})}{(\boldsymbol{x^T x})^2} - \frac{\boldsymbol{2Mx}}{\boldsymbol{x^T x}}\end{aligned}$$

We can express our partial derivative result as:

$$\frac{\partial f}{\partial x_k} = \frac{2x_k(\sum_{i=1}^{n}\sum_{j=1}^{n} x_i m_{ij} x_j)}{(\sum_{i=1}^{n} x_i^2)^2} - \frac{\sum_{j=1}^{n} m_{kj} x_j}{\sum_{i=1}^{n} x_i^2} = \frac{2x_k(\boldsymbol{x^T M x})}{(\boldsymbol{x^T x})^2} - \frac{\boldsymbol{2Mx}}{\boldsymbol{x^T x}} \tag{2.4}$$

The derivative of our objective function exists, is continuous, and differentiable at all points except at 0. Therefore, the objective function is differentiable in $\mathbb{R}^n \setminus \{0\}$.

The complexity of obtaining the gradient from scratch is $O(n^2)$, similar to the (2.3), due to the similar operations involved. However, if we write the gradient in a more efficient manner, like:

$$\nabla f(\boldsymbol{x}) = \frac{2\boldsymbol{x}(\boldsymbol{f(x)})}{(\boldsymbol{x^T x})} - \frac{\boldsymbol{2Mx}}{\boldsymbol{x^T x}} \tag{2.5}$$

preserve the partial results we have for $\boldsymbol{f(x)}$, such as $\boldsymbol{x^T x}$ and $\boldsymbol{Qx}$, we can save many operations and obtain the gradient with just an $O(n)$ computation needed for $\boldsymbol{x f(x)}$. After determining the value of the function, the final computational cost of the gradient is as follows:

$$C(\nabla f(\boldsymbol{x})) = O(n). \tag{2.6}$$

2.3 Stationary Point

It's important to determine if our function has stationary points, as our algorithm must converge to these points. Given a stationary point $\boldsymbol{\hat{x} \neq 0}$ we need to determine where $\nabla f(\hat{x})$ is equal to 0. However, since the function is not convex, these stationary points might not be global minima.

$$\nabla f(\hat{\boldsymbol{x}}) = \frac{\boldsymbol{2M\hat{x}}}{\boldsymbol{\hat{x}^T\hat{x}}} - \frac{2\boldsymbol{\hat{x}}(\boldsymbol{\hat{x}^T M\hat{x}})}{(\boldsymbol{\hat{x}^T\hat{x}})^2} = \frac{\boldsymbol{2M\hat{x}}(\boldsymbol{\hat{x}^T\hat{x}}) - 2\boldsymbol{\hat{x}}(\boldsymbol{\hat{x}^T M\hat{x}})}{(\boldsymbol{\hat{x}^T\hat{x}})^2} = 0 \tag{2.7}$$

$$\boldsymbol{2M\hat{x}}(\boldsymbol{\hat{x}^T\hat{x}}) - 2\boldsymbol{\hat{x}}(\boldsymbol{\hat{x}^T M\hat{x}}) = 0$$

$$\boldsymbol{M\hat{x}}(\boldsymbol{\hat{x}^T\hat{x}}) = \boldsymbol{\hat{x}}(\boldsymbol{\hat{x}^T M\hat{x}})$$

$$\frac{\boldsymbol{\hat{x}}(\boldsymbol{\hat{x}^T M\hat{x}})}{(\boldsymbol{\hat{x}^T\hat{x}})} = \boldsymbol{M\hat{x}}$$

$$f(\boldsymbol{\hat{x}})\boldsymbol{\hat{x}} = \boldsymbol{M\hat{x}} \tag{2.8}$$

If $\hat{x}$ is an eigenvector of $\boldsymbol{M}$ and its eigenvalue is $f(\boldsymbol{\hat{x}})$, then $\hat{x}$ is a stationary point for our function.

2.4 Non Convexity of the Function

Since our function is bounded but not constant it is not convex, a constant function is the only function that satisfies both requirements. Assuming that $f(\boldsymbol{x}))$ is convex, the following inequality must hold since $\nabla f(\boldsymbol{x})$ exists everywhere.

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle \quad \forall x, y \in R^n$$

If we take two different eigenvalues, $\tilde{x}_i$ and $\tilde{x}_j$, with corresponding eigenvectors $f(\tilde{x}_i)$ and $f(\tilde{x}_j)$, assuming $\tilde{x}_i > \tilde{x}_j$, then $f(\tilde{x}_i) > f(\tilde{x}_j)$. However, according to the convexity assumption, we should have $f(\tilde{x}_i) \leq f(\tilde{x}_j)$, which contradicts the idea that the function is convex.

2.5 Descent Direction

Next, we will discuss the formula to calculate the step-size α^i at a given point x^i during the i^{th} iteration in the direction d^i.

$$\varphi(\alpha^i) = f(x + \alpha^i d^i) \quad \alpha > 0$$

We are trying to find the minimum of the function $\varphi(\alpha^i)$. To do this, we solve the derivative of the equation $\varphi'(\alpha^i) = 0$ which can be found easily using the chain rule.

$$\begin{aligned}
&\varphi(\alpha) = f(x(\alpha)), \ \textit{where}\ x(\alpha) = x + \alpha d \\
&f(x(\alpha)) = \frac{-(x + \alpha d)^T M (x + \alpha d)}{(x + \alpha d)^T (x + \alpha d)} \\
&\varphi(\alpha) = -\left(\frac{x^T M x + \alpha(d^T M x + x^T M d) + \alpha^2 (d^T M d)}{x(\alpha)^T x(\alpha)}\right)
\end{aligned}$$

Since $x^T M d \in \mathbb{R}$ is a real number and and M is symmetric, we have $x^T M d = (x^T M d)^T = d^T M^T x = d^T M x$. Therefore, the equation can be rewritten as follows.

$$\varphi(\alpha) = -\left(\frac{x^T M x + 2\alpha d^T M x + \alpha^2 d^T M d}{x(\alpha)^T x(\alpha)}\right)$$

$$\varphi'(\alpha) = -\left[\frac{(2d^T M x + 2\alpha d^T M d)(x(\alpha)^T x(\alpha)) - ((x^T M x + 2\alpha d^T M x + \alpha^2 d^T M d)(2(x + \alpha d)^T d))}{(x(\alpha)^T x(\alpha))^2}\right]$$

$$= -\big[2\alpha^3((d^TMd)d^Td) + 2\alpha^2((d^TMx)d^Td + 2(d^TMd)d^Tx) +$$

$$2\alpha((d^TMd)x^Tx) + 2((d^TMx)d^Tx) + 2((d^TMx)x^Tx)\big] +$$

$$\frac{\big[2\alpha^3((d^TMd)d^Td) + 2\alpha^2((d^TMx)d^Td + 2(d^TMx)d^Td) + 2\alpha(2(d^TMx)(x^Td)) + (x^TMx)d^Td) + 2((x^TMx)x^Td)\big]}{(x(\alpha)^Tx(\alpha))^2}$$

By simplifying the above equation, we get:

$$\varphi'(\alpha) = \frac{2\alpha^2((d^TMx)d^Td - (d^TMd)d^Tx) + 2\alpha((x^TMx)d^Td) - (d^TMx)x^Tx) + 2(x^TMx)x^Td - (d^TMx)x^Tx}{((x+\alpha d)^T(x+\alpha d))^2}$$

We can simplify our equation to a shorter form like this:

$$\varphi'(\alpha) = 2.\frac{\alpha^2a + \alpha b + c}{R(\alpha)}$$

Where,

$$a = (d^TMx)d^Td \tag{2.9}$$

$$b = ((x^TMx)d^Td - (d^TMd)(x^Tx))$$

$$c = (x^TMx)x^Td - (d^TMx)x^Tx$$

$$R(\alpha) = (x(\alpha)^Tx(\alpha))^2$$

To talk about the complexity of computing this, in general, it takes $\mathcal{O}(n^2)$ time to calculate the parameters a, b, and c because $\mathcal{O}(n^2)$ complexity is needed to find the values for x^TMx and d^TMd.

$$C(\varphi'(\alpha)) = \mathcal{O}(n^2) \tag{2.10}$$

3 Steepest Gradient Descent

In this section, we introduce the first algorithm we'll use to solve the minimization problem from Chapter 1, called the Steepest Gradient Descent. This is a type of descent algorithm. It works by moving in the steepest direction, which, when using the $\boldsymbol{l_2}$ norm, is -$\nabla f(\boldsymbol{x})$. Therefore, this algorithm is also known as "Gradient Descent."

Algorithm 1 Algorithm for Steepest Gradient Method

```
1: procedure SGD($\boldsymbol{x_0} \in \mathbb{R}^N_{\neq 0}$)
2:     while $\|\nabla f(x_k)\| > \epsilon$ do
3:         $\boldsymbol{d_k} = -\nabla f(x_k)$
4:         $\phi(\alpha) = f(x_k + \alpha d_k)$
5:         $\bar{\alpha} = \arg\min_\alpha \phi(\alpha) : \alpha > 0$
6:         $\boldsymbol{x_{k+1} = x_k + d_k \bar{\alpha}}$
7:     return $\boldsymbol{x_{k+1}}$
```

3.1 Exact Line Search Towards Descent Direction

d_k represents the direction of the descent and the exact search along this direction is given as $\phi^{'}(\alpha) = 0$. We must ensure that the $\phi^{'}(\alpha)$ denominator is not zero to find a stationary point in this direction.

$$\begin{aligned} P(\alpha) &= (\boldsymbol{x} - \alpha \nabla f(x)^T)(\boldsymbol{x} - \alpha \nabla f(x)) \\ &= \|(\boldsymbol{x} - \alpha \nabla f(\boldsymbol{x}))\|_2^2 = 0 \\ &\iff \boldsymbol{x} = \alpha \nabla f(\boldsymbol{x}) \end{aligned}$$

But there is a solution for α iff $\boldsymbol{x}$ and $\nabla f(x)$ are linearly dependent. However, it is clear that $\boldsymbol{x}$ and $\nabla f(x)$ are orthogonal.

$$\begin{aligned} < \boldsymbol{x}, \nabla f(x) > = \boldsymbol{x^T}.\nabla f(x) &= \frac{x^T\left((x^TQx)x - (x^Tx)Qx\right)}{(x^Tx)^2} \\ &= \frac{((x^TQx)(x^Tx) - (x^Tx)(x^TQx))}{(x^Tx)^2} = 0 \quad \forall x \neq 0 \end{aligned}$$

$\phi^{'}(\alpha)$ can be written as a quadratic polynomial:

$$a\alpha^2 + b\alpha + c = 0$$

The smallest positive root of this polynomial gives the stationary point $\bar{\alpha}$ such that $\phi^{'}(\bar{\alpha}) = 0$ which satisfies the Wolfe condition. This ensures a sufficient decrease in the algorithm's convergence.

3.2 Convergence of the Algorithm

As we know the function is Lipschitz continuous for $x : \|x\| \geq \epsilon > 0$ since the norm of the gradient is bounded by M/ϵ out of a ball $B(0, \epsilon) \quad \forall_\epsilon > 0$. Fortunately, we can look for our solution in a domain where the function is Lipschitz continuous $\mathcal{D} = \mathbb{R}^n \setminus \mathcal{B}(0, \epsilon) \quad \forall \epsilon > 0$. In fact, we can choose $\epsilon < \|x_0\|_2$ and $\|x_k\|_2 \in \mathcal{D} \quad \forall k \geq 0$ because $\|x_k\|_2 > \|x_{k-1}\|_2$. This results from the fact that the gradient and the point are orthogonality, as previously proved. To show the orthogonality, we can use the generalized Pythagorean theorem in $\mathbb{R}^n$[1].

$$\|\mathbf{x}_{k+1}\|_2^2 = \|\mathbf{x}_k - \bar{\alpha}\nabla f(\mathbf{x}_k)\|_2^2 = \|\mathbf{x}_k\|_2^2 + \bar{\alpha}\,\|\nabla f(\mathbf{x}_k)\|_2^2 > \|\mathbf{x}_k\|_2^2$$

We have shown that if $||x_0||_2 \geq \epsilon > 0$, the sequence of points lies in $D = \mathbb{R}^n \setminus B(0, \epsilon)$, where the function is Lipschitz continuous and differentiable. Indeed, with these hypotheses, along with the Wolfe condition demonstrated in 4.1, we can apply Zoutendijk's theorem:

Theorem 1. *Consider an algorithm which searches along a descending direction $\mathbf{d}_k$ a point which satisfies the Wolfe condition $\varphi'(\alpha_k) \leq m_2\varphi'(\alpha_k)$: $m_2 < 1$. f is bounded below in $\mathbb{R}^n$ and it's differentiable in an open set $\mathcal{N}$ containing the level set $\mathcal{L} = \{x : f(x) \leq f(x_0)\}$, where x_0 is the starting point. Assume also that f is Lipschitz continuous on $\mathcal{N}$, that is, exists an L:*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \quad \forall x, y \in \mathcal{N}$$

[1] $x^T y = 0 \implies \|x + y\|_2^2 = (x + y)^T(x + y) = \|x\|_2^2 + \|y\|_2^2 + \underbrace{2x^T y}_{=0}$

Then

$$\sum_{k=0}^{+\infty} \cos^2\theta_k \|\nabla f(x_k)\|_2^2 < \infty \quad \cos\theta_k := \frac{\mathbf{d}_k^T \nabla f(x_k)}{\|\mathbf{d}_k\|_2 \|\nabla f(x_k)\|_2}$$

The complete proof is provided at [1, pg. 38-40]

In this scenario, the $\cos\theta_k$ is consistently -1 because $\mathbf{d}_k^T = -\nabla f(\mathbf{x}_k)$. Consequently, we have $\sum_{k=0}^{+\infty} \|\nabla f(\mathbf{x}_k)\|_2^2 = k < \infty$, which implies that $\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\|_2^2 = 0$, and thus the algorithm will converge to a stationary point.

3.3 Convergence Rate of the Algorithm

Let f be a quadratic function and the exact line search is used. Such as

$$f(x) = \frac{1}{2}x^T Q x - b^T x \tag{3.1}$$

where Q is symmetric and positive definite, the gradient of the function is $\nabla f(x) = Qx - b$ and x^* the minimizer is the unique solution of the linear system $Qx = b$.

The step length α_k that minimizes $f(x_k - \alpha\nabla f_k)$ can be calculated by differentiating the function

$$f(x_k - \alpha\nabla f_k) = \frac{1}{2}(x_k - \alpha\nabla f_k)^T Q (x_k - \alpha\nabla f_k) - b^T(x_k - \alpha\nabla f_k) \tag{3.2}$$

with respect to α, and setting the derivative to zero, we get

$$\alpha_k = \frac{\nabla f_k^T \nabla f_k}{\nabla f_k^T Q \nabla f_k} \tag{3.3}$$

Using this particular minimizer α_k, the steepest descent iteration for 3.1, is provided by

$$x_{k+1} = x_k - \left(\frac{\nabla f_k^T \nabla f_k}{\nabla f_k^T Q \nabla f_k}\right) \nabla f_k \tag{3.4}$$

This equation gives a closed-form expression for x_{k+1} in terms of x_k since $\nabla f(x) = Qx - b$.

To measure the rate of convergence, we introduce the weighted norm $||x||_Q^2 = x^T Q x$. Using the relation $Qx^* = b$, we can demonstrate that

$$\frac{1}{2}\|x - x^*\|_Q^2 = f(x) - f(x^*) \tag{3.5}$$

so this norm measures the difference between the current objective value and the optimal value. By using the equality 3.4 and noting that $\nabla f(x) = Qx - b$, we can derive the equality

$$\|x_{k+1} - x^*\|_Q^2 = \left\{1 - \frac{(\nabla f_k^T \nabla f_k)^2}{(\nabla f_k^T Q \nabla f_k)(\nabla f_k^T Q^{-1} \nabla f_k)}\right\} \|x_k - x^*\|_Q^2 \tag{3.6}$$

This expression describes the exact decrease in f at each iteration, but since the term inside the brackets is difficult to interpret, it is more useful to bound it in terms of the condition number of the problem.

Theorem 2. *When the steepest descent method with exact line searches 3.4 is applied to the strongly convex quadratic function 3.1, the error norm 3.5 satisfies*

$$\|x_{k+1} - x^*\|_Q^2 \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 \|x_k - x^*\|_Q^2, \tag{3.7}$$

where $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of Q.

Luenberger [2] provides the proof for this result. The inequalities 3.5 and 3.7 demonstrate that the function values f_k converge to the minimum f^* at a linear rate. In a particular case of this finding the convergence can be reached in a single iteration if all of the eigenvalues are equal. Since Q is a multiple of the identity matrix in this case, the solution is always pointed in the direction of the steepest descent. In general, as the condition number $k(Q) = \frac{\lambda_n}{\lambda_1}$ increases, the contours of the quadratic get longer, and 3.7 implies that the convergence rate decreased. Even though 3.7 is a worst-case bound, it gives an accurate indication of the behaviour of the algorithm when $n > 2$.

The convergence rate for the steepest descent method is similar for general nonlinear objective functions. In the following result, we assume that the step length is the global minimum along the search direction.

Theorem 3. *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, and the iterates generated by the steepest descent method with exact line searches converge to a point x^* where the Hessian matrix $\nabla^2 f(x^*)$ is positive definite. Let r be any scalar satisfying*

$$r \in \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}, 1 \right),$$

where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of $\nabla^2 f(x^*)$. Then for all k sufficiently large, we have

$$f(x_{k+1}) - f(x^*) \leq r^2[f(x_k) - f(x^*)].$$

In general, using an inexact line search does not improve the rate of convergence. Thus, Theorem 3 suggests that the steepest descent method may have an unacceptably slow convergence rate, even when the Hessian is relatively well-conditioned. If $k(Q) = 800$, $f(x_1) = 1$, and $f(x^*) = 0$, for instance, According to Theorem 3, even after a thousand iterations of the steepest descent method with exact line search, the function value will remain approximately 0.08.

3.4 Complexity of the Algorithm

In this section, we will discuss the complexity of each iteration of the algorithm. We have shown in equations 2.3, 2.6, and 2.10 the complexity of computing $f(x)$, $\nabla f(x)$, and $\varphi'(\alpha)$. The final step, which is not merely addition or allocation, involves minimizing $\varphi(\alpha)$. However, as discussed in section 4.1, this can be achieved by solving a quadratic polynomial equation using a closed-form solution in $\mathbb{R}$, which is $O(1)$. Therefore, we can conclude that the complexity of one iteration of the Steepest Descent Direction algorithm is $O(n^2)$.

$$C(SGD) = O(n^2) \tag{3.8}$$

4 Quasi-Newton method: BFGS

Similar to steepest descent, quasi-Newton methods only require the gradient of the objective function to be provided at each iteration. They create a model of the objective function that is sufficient to produce superlinear convergence by measuring the changes in gradients. The improvement over the steepest descent is significant, particularly for challenging problems.

One of the well-known quasi-Newton algorithms is the BFGS method, named after its founders Broyden, Fletcher, Goldfarb, and Shanno. BFGS repeatedly updates an estimate of the Hessian matrix to find the optimal solution.

The H_k matrix is the inverse of the B_k matrix, $H_k = B_k^{-1}$. It is an $n \times n$ symmetric positive definite matrix that serves as an approximate Hessian and is updated at each iteration.

Algorithm 2 BFGS

```
procedure BFGS($\boldsymbol{H}_0 \in \mathbb{R}^{n\times n}, \boldsymbol{x}_0 \in \mathbb{R}^n, \epsilon \in \mathbb{R}$)
    while $\|\nabla f_k\| > \epsilon$ do
        $p_k = -H_k \nabla f_k$                    ▷ Computes the search direction
        $x_{k+1} = x_k + \alpha_k p_k$    ▷ Where $\alpha_k$ satisfies either the Wolfe or the Armijo conditions
        $s_k = x_{k+1} - x_k, \quad y_k = \nabla f_{k+1} - \nabla f_k$
        $\rho_k = \frac{1}{y_k^T s_k}$
        $H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$
        $k = k + 1$
```

Two of the most popular formulas for updating the Hessian approximation B_k are the symmetric-rank-one (SR1) formula, defined by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}, \tag{4.1}$$

and the BFGS formula defined by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \tag{4.2}$$

Notice that the difference between matrices B_k and B_{k+1} is a rank-one matrix in the case of

4.1 and a rank-two matrix in the case of 4.2. Both updates satisfy the secant equation and maintain symmetry. The secant equation is given by

$$H_{k+1} y_k = s_k \tag{4.3}$$

The secant equation requires the matrix H_{k+1} to map the displacement $s_k = x_{k+1} - x_k = \alpha_k p_k$ into the difference between the gradients $y_k = \nabla f_{k+1} - \nabla f_k$. This is only possible if s_k and y_k satisfy the *curvature condition.*

$$s_k^T y_k > 0 \tag{4.4}$$

However, this condition will not always apply to nonconvex functions, and in this case, we must expressly enforce 4.4 by placing limitations on the line search process that determines the step length α. Indeed, if we apply the Wolfe conditions to the line search, we can be sure that the condition 4.4 will hold.

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \le f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f_k^T \boldsymbol{p}_k,$$

$$\nabla f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)^T \boldsymbol{p}_k \ge c_2 \nabla f_k^T \boldsymbol{p}_k,$$

where $0 < c_1 < c_2 < 1$.
When the curvature condition is satisfied, the secant equation 4.3 always has a solution H_{k+1}.

Instead of recalculating the matrix from scratch at every step, we can approximate the Hessian at each iteration, following the quasi-Newton condition. In one dimension, this means approximating the second derivative using a finite difference method, known as the secant method. In multiple dimensions, the quasi-Newton condition does not uniquely determine the Hessian estimate B. Therefore, additional constraints are needed, such as satisfying the secant equation and ensuring the new approximation H_k is close to the previous iteration.

We look for the matrix that is closest to the present H_k among all symmetric matrices that satisfy the secant equation in order to uniquely define H_{k+1}. The following defines the clossenes condition:

$$\min_{\boldsymbol{H}_{k+1}} \|\boldsymbol{H}_{k+1} - \boldsymbol{H}_k\|$$
$$\text{subject to } \boldsymbol{H} = \boldsymbol{H}^T \text{ and } \boldsymbol{H}\boldsymbol{y}_k = \boldsymbol{s}_k.$$

It is possible to utilize various matrix norms, each of which results in a unique quasi-Newton method. The weighted Frobenius norm is applied in the BFGS method.

$$\|\boldsymbol{A}\|_W = \|\boldsymbol{W}^{1/2}\boldsymbol{A}\boldsymbol{W}^{1/2}\|_F,$$

where the weight matrix $\boldsymbol{W}$ is any matrix satisfying $\boldsymbol{W}\boldsymbol{s}_k = \boldsymbol{y}_k$.
Given the previous optimization problem, the unique solution H_{k+1} is provided by

$$\boldsymbol{H}_{k+1} = (\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T)\boldsymbol{H}_k(\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T, \tag{4.5}$$

with

$$\rho_k = \frac{1}{\boldsymbol{y}_k^T \boldsymbol{s}_k}.$$

The initialization of the H_0 matrix is a problem for which there is no set approach. It can be expressed as the identity matrix I, the inverse approximate Hessian computed by finite differences at x_0, or even as a multiple of the identity matrix βI.

4.1 BFGS for Non-Convex Function

The BFGS method is a well-known quasi-Newton method for solving unconstrained optimization problems. Significant progress has been made in studying its global convergence, and its local convergence theory is well established [3, 4]. When applying the exact line search or a particular inexact line search to convex minimization problems, it has been demonstrated that the iterates produced by BFGS are globally convergent. However, for nonconvex minimization problems, little is known about the global convergence of the BFGS method.

Li and Fukushima recently proposed a modified BFGS method [5], which also demonstrated the global convergence of BFGS for nonconvex unconstrained optimization problems by introducing a cautious approach update the $\boldsymbol{H}_k$ matrix. The cautious update rule proposed is as follows:

$$\boldsymbol{B}_{k+1} = \begin{cases} \boldsymbol{B}_k - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k} + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}, & \text{if } \frac{\boldsymbol{y}_k^T \boldsymbol{s}_k}{\|\boldsymbol{s}_k\|^2} > \epsilon \|\nabla f_k\|^\alpha, \\ \boldsymbol{B}_k, & \text{otherwise,} \end{cases} \tag{4.6}$$

where ϵ and α are positive constants. As anticipated, we will adjust this rule to allow for the substitution of the H_k matrix for B_k. The following update formula is the outcome of this modification:

$$\boldsymbol{H}_{k+1} = \begin{cases} (\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T) \boldsymbol{H}_k (\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T, & \text{if } \frac{\boldsymbol{y}_k^T \boldsymbol{s}_k}{\|\boldsymbol{s}_k\|^2} > \epsilon \|\nabla f_k\|^\alpha, \\ \boldsymbol{H}_k, & \text{otherwise,} \end{cases} \tag{4.7}$$

Now, we modify the BFGS algorithm with the cautious update:

Algorithm 3 CBFGS

```
procedure CBFGS($\boldsymbol{H}_0 \in \mathbb{R}^{n \times n}, \boldsymbol{x}_0 \in \mathbb{R}^n, \epsilon \in \mathbb{R}$)
    while $\|\nabla f_k\| > \epsilon$ do
        $p_k = -H_k \nabla f_k$
        $x_{k+1} = x_k + \alpha_k p_k$
        $s_k = x_{k+1} - x_k, \quad y_k = \nabla f_{k+1} - \nabla f_k$
        Computes $H_{k+1}$ according to 4.7
        $k = k + 1$
```

4.2 Convergence of the Algorithm

As described in the previous subsection, the local convergence theory of the BFGS method is well-established and significant progress has also been made in its global convergence for convex optimization, which is not the case in nonconvex optimization. In fact, no one has demonstrated the global convergence of the BFGS method for nonconvex minimization problems or provided a counterexample demonstrating its non-convergence.

Convergence of the BFGS method for nonconvex minimization is considered as one of the most fundamental open problems in the theory of quasi-Newton methods, this problem has been discussed numerous times [6, 7]. There are several different methods propose small changes to the original BFGS method for nonconvex minimization which mostly centre around the modifications to the B_k matrix to maintain the positive definiteness of the Hessian approximation or to employ carefully designed line search techniques [5, 8, 9].

To elaborate more on the assumptions above, we will state:

Lemma 5.1 (Lemma 2.1 in [5]): If BFGS method with Wolfe-type line search is applied to a continuously differentiable function f that is bounded below, and if there exists a constant $\boldsymbol{M} > 0$ such that the inequality

$$\frac{\|\boldsymbol{y}_k\|^2}{\boldsymbol{y}_k^T \boldsymbol{s}_k} \leq \boldsymbol{M} \tag{4.8}$$

holds for all k, then

$$\liminf_{k\to\infty} \|g(\boldsymbol{x}_k)\| = 0. \tag{4.9}$$

Observe that 4.8 always holds if f is both uniformly convex and twice continuously differentiable. Therefore, Lemma 5.1 immediately implies global convergence of the BFGS method. On the other hand, it appears challenging to ensure 4.8 when f is nonconvex.

But, a cautious update in BGFS for matrix B_k, or in our case, H_k, ensures for all values of k an approximate, symmetric, and positive definite Hessian. We now demonstrate the global convergence of Algorithm 3 under the assumptions:

Assumption 4.1 (Assumption A in [5]): The level set

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n | f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\}$$

is contained in a bounded convex set D. The function f is continuously differentiable on D

and there exists a constant $L > 0$ such that

$$\|g(\boldsymbol{x}) - g(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|, \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in D.$$

Given that $f(x_k)$ is a decreasing sequence, it follows that Ω contains the sequence $\{f_x\}$ produced by Algorithm 3. The complete proof is provided at [5, pg. 5]. Nonetheless, we present the two primary theorems from the source.

Theorem 4. ***(Theorem 3.2 in [5]):*** *Let Assumption 4.1 holds and $\boldsymbol{x}_k$ be generated by Algorithm 3 with α_k being generated by Armijo-type line search. Then*

$$\liminf_{k \to \infty} \|g(\boldsymbol{x}_k)\| = 0. \tag{4.10}$$

The complete proof is provided at [5, pg. 8-9].

Theorem 5. ***(Theorem 3.3 in [5]):*** *Let Assumption 4.1 holds and $\boldsymbol{x}_k$ be generated by Algorithm 3 with α_k being generated by Wolfe-type line search. Then (4.10) holds.*

The complete proof is provided at [5, pg. 10-11].

4.3 Convergence Rate of the Algorithm

The source provide two main theorems to proof the rate of convergence of BFGS with cautious update is superlinear. The first theorem states:

Theorem 6. *Let f be twice continuously differentiable. Suppose that $s_k \to 0$. If there exists an accumulation point x^* of $\{x_k\}$ at which $g(x^*) = 0$ and $G(x^*)$ is positive definite, then the whole sequence $\{x_k\}$ converges to x^*. If in addition, G is Hölder continuous and the parameters in the line searches satisfy $\sigma, \sigma_2 \in (0, 1/2)$, then the convergence rate is superlinear.*

The complete proof is provided at [5, pg. 11-12].

Let $\sigma_3 \in (0,1)$ and $\sigma_4 > 0$ be given constants. We determine a stepsize λ_k satisfying the inequality

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + c_1 \alpha_k \nabla f_k^T \boldsymbol{p}_k - \sigma_4 \|\lambda_k p_k\|^2 \tag{4.11}$$

Then

Theorem 7. *Let Assumption 4.1 hold and $\{x_k\}$ be generated by Algorithm 3 with λ_k satisfying 4.11. Then 4.9 holds. If we further suppose that f is twice continuously differentiable and there exists an accumulation point x^* of $\{x_k\}$ at which $g(x^*) = 0$ and $G(x^*)$ is positive definite, then the whole sequence $\{x_k\}$ converges to x^*. If in addition, G is Hölder continuous at x^* and $\sigma_3 \in (0, 1/2)$, then the convergence rate is superlinear.*

Proof excluded

4.4 Complexity of the Algorithm

In this section, we calculate the complexity of executing a BFGS algorithm iteration without taking into consideration the cost of determining the value and gradient of the function.

The search direction $\boldsymbol{p}_k = -\boldsymbol{H}_k \nabla f_k$ can be calculated with a simple matrix-vector multiplication by directly computing and updating the matrix $\boldsymbol{H}_k$ as the inverse of the approximated Hessian, rather than of computing the actual approximation $\boldsymbol{B}_k$ and then inverting it. We can avoid the extra computational load of $O(n^3)$ operations required for matrix inversion by using this method, which only needs $O(n^2)$ operations.

Because most internal operations of the algorithm are handled by matrix-vector, vector-vector, or scalar operations, it does not require costly matrix-matrix operations.

The overall computational complexity of running an iteration of the BFGS algorithm, just as for CBFGS, is:

$$C(\text{BFGS}) = C(\text{CBFGS}) = O(n^2)$$

5 Experiments and Results

In this section, we will run multiple experiments to analyze the performance, report results, and analyze the efficiency of the algorithms. For this purpose, we have generated the following dataset with different properties. We will use Apple MacBook Pro 2021 with Apple M1 chip and 8 GB Memory to run our experiments.

- **[A]** $A_{10000\times 1000}$, where a $\in$ [-50,50], $x_0 \in \mathbb{R}^{1000}$ vector.
- **[B]** $A_{1000\times 100}$, where a $\in$ [-50,50], $x_0 \in \mathbb{R}^{100}$ vector.
- **[C]** $A_{100\times 1000}$, where a $\in$ [-50,50], $x_0 \in \mathbb{R}^{1000}$ vector.
- **[D]** $A_{100\times 100}$, where a $\in$ [-50,50], $x_0 \in \mathbb{R}^{100}$ vector.
- **[E]** $A_{1000\times 100}$ sparse matrix with density $= 0.3$, $x_0 \in \mathbb{R}^{100}$.
- **[F]** is ill-conditioned matrix $A_{1000\times 1000}$, where a $\in$ [-2,3]and cond. no = 1e18, $x_0 \in \mathbb{R}^{1000}$.

Where x_0 is the initial guess for the specified matrix generated randomly and stored along with the matrices so both algorithms must work using the same initial vector. In our case, we generated 10 matrices for each type. The evaluation, also, depends on some algorithm parameters that are:

- **MaxFeval** = the max number of the iterations which is set to 1000
- **eps** = the accuracy in the stopping criterion: the algorithm is stopped when the norm of the gradient is less than or equal to eps which is set to 1×10^{-5}
- **mina** = the minimum step size which is set to 1×10^{-16}.

The graphs below, represent the trend of the Relative Error and Norms of Gradient, in a natural logarithmic scale. Relative error is defined as the ratio between absolute error and accepted norm. They are defined as follow.

$$
\begin{aligned}
Residual &= \|\nabla f(x^i)\| \\
RelativeError &= (f(x^i) - f^*)/|f^*|
\end{aligned}
\tag{5.1}
$$

Where f^* is the actual matrix norm $\|A\|_2$ calculated using linalg module from numpy. Figures 1, 2, 3, and 4 represent plots for Relative Error and Norms Gradient by three algorithms SDG, BFGS, and CBGFS with different types of matrices

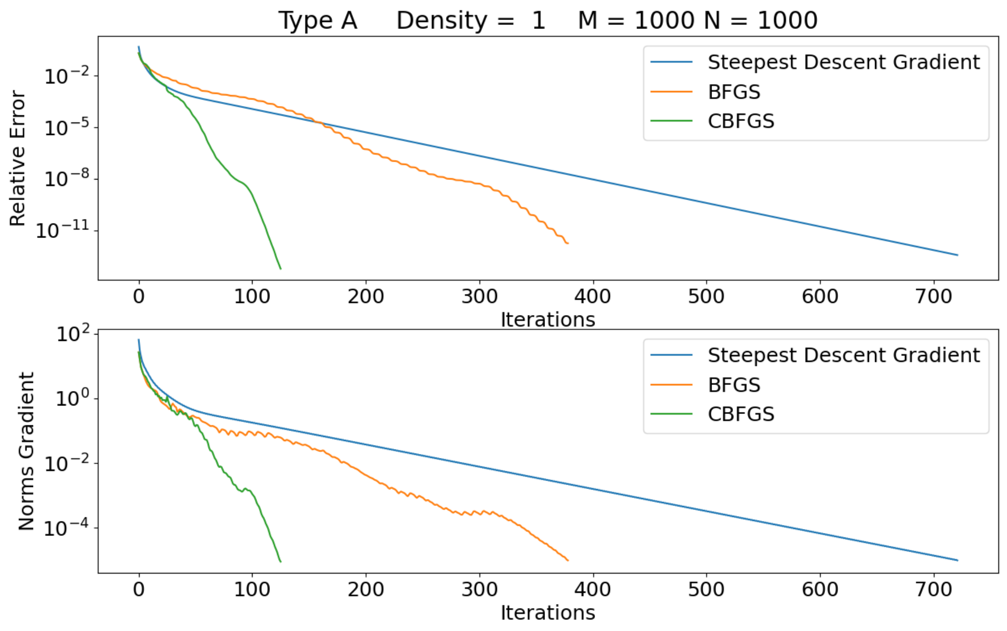

Figure 1: Plot of Relative Error and Norms of Gradient of matrix of Type A

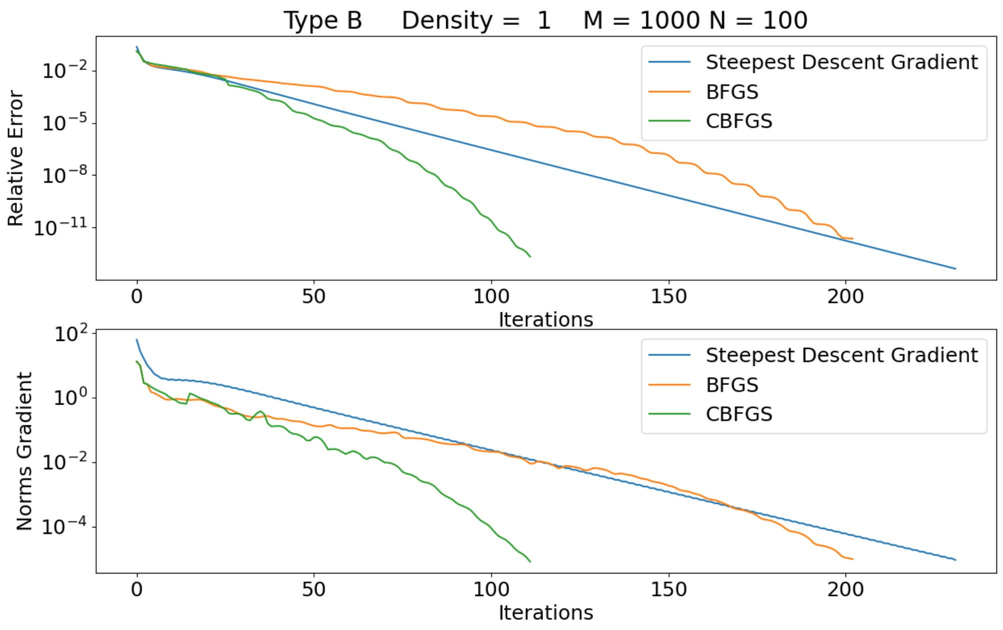

Figure 2: Plot of Relative Error and Norms of Gradient of matrix of Type B

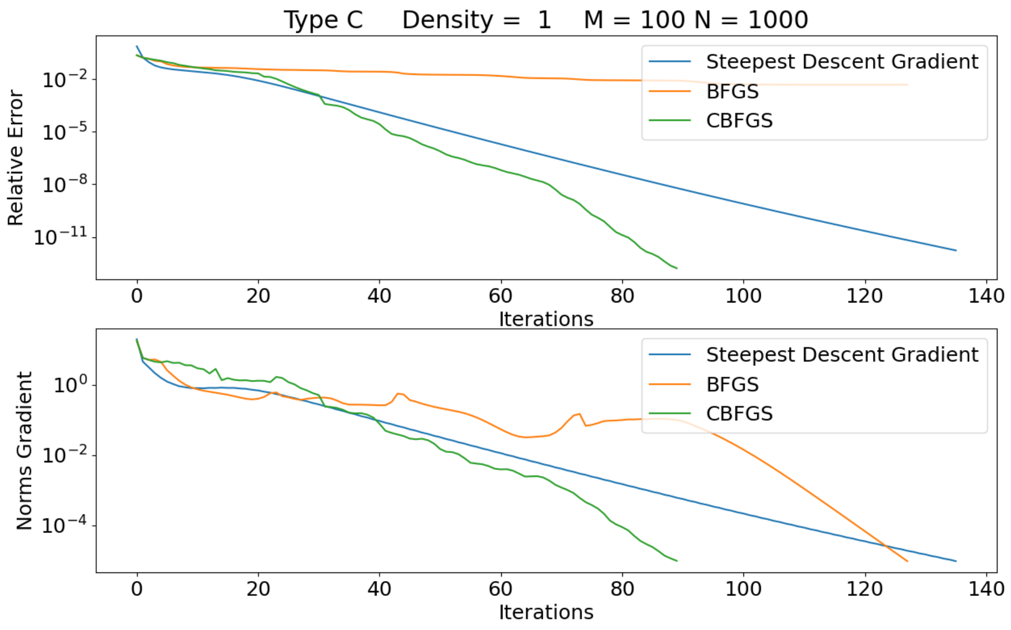

Figure 3: Plot of Relative Error and Norms of Gradient of matrix of Type C

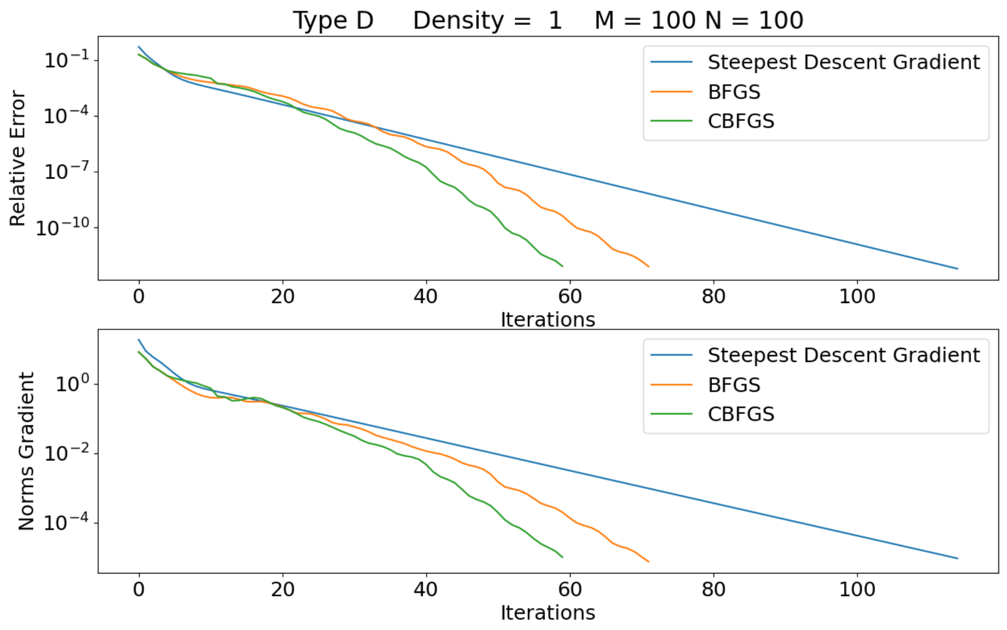

Figure 4: Plot of Relative Error and Norms of Gradient of matrix of Type D

Looking at the plots Figure 1, Figure 2, Figure 3 and Figure 4 we can observe that CBFGS outperformed the other two algorithms in almost all cases. It is interesting to see Figure 3 where the BFGS algorithm seems to be reducing the norm of the gradient effectively, but it does not translate to a decrease in relative error. This could be due to BFGS finding points where the gradient is small, but these points are not necessarily close to the optimal solution. We tried changing our x_0 which produces the results in Figure 5 which produces the expected behaviour but still BFGS converges slower than SDG and the possible reasons for this could be the Hessian approximation becomes ill-conditioned.

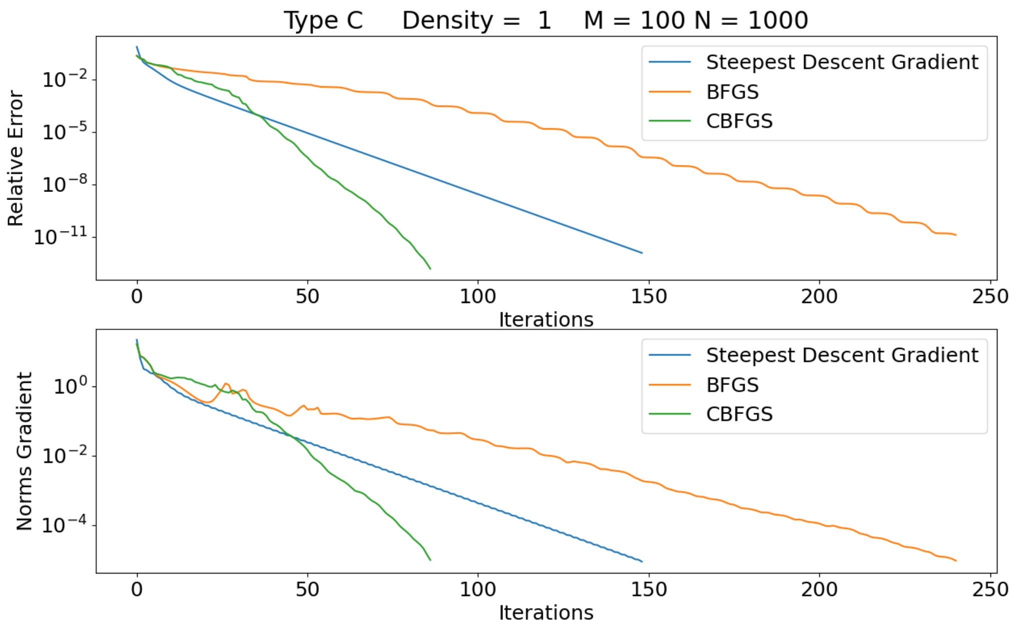

Figure 5: Plot of Relative Error and Norms of Gradient of matrix of Type C with Updated x_0

Overall, the algorithms converge faster so we can impose an early stopping even before $1e{-}5$ without affecting the final results significantly. Next, we are going to run our experiment on type E and F matrices.

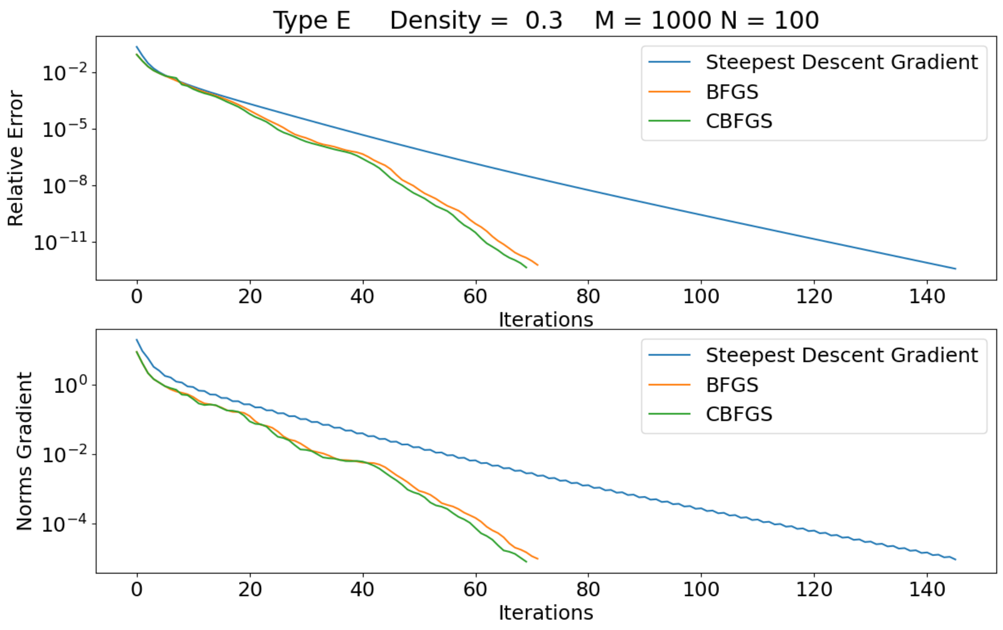

Figure 6: Plot of Relative Error and Norms of Gradient of matrix of Type E

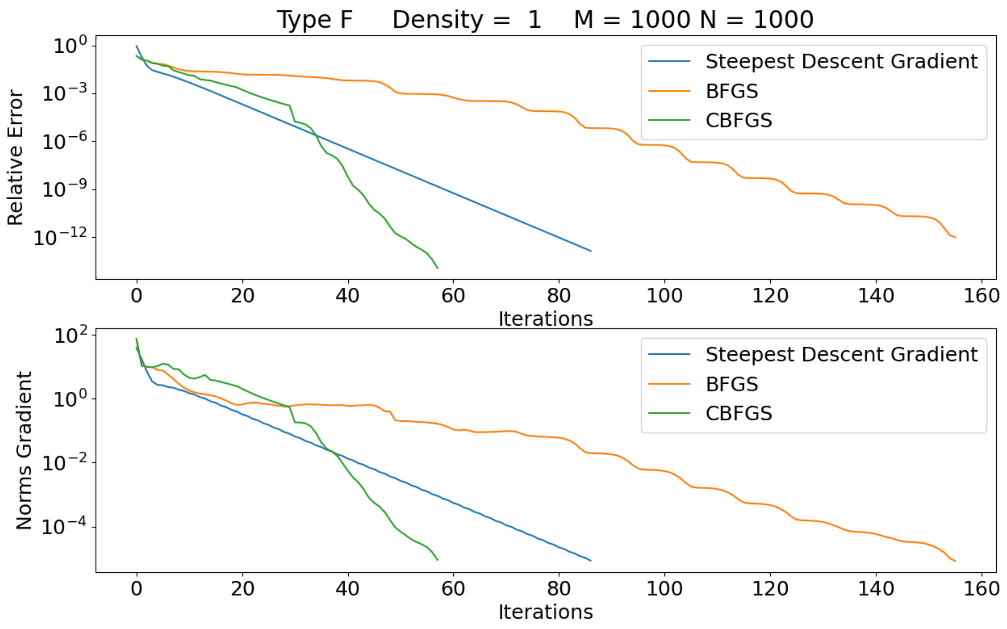

Figure 7: Plot of Relative Error and Norms of Gradient of matrix of Type F

Observing plots in Figure 6 and Figure 7 we can conclude that CBFGS again worked well with sparse and ill-condition matrices whereas in ill-condition matrix simple BFGS didn't work very well which proves the statement that BFGS method does not work very well for nonconvex optimization but a modified version of BFGS such as CBFGS works well for nonconvex optimization. Type F matrices are ill-conditioned matrices which means a small change in input could cause a significant change in the output. The BFGS algorithm relies on approximating the inverse of the Hessian matrix. If the Hessian is ill-conditioned, the approximation can be poor, leading to slow convergence of the algorithm as we can see the effect in Figure 7. Furthermore, the ill-conditioned Hessian can make the optimization process highly sensitive to the initial guess. Small changes in the initial guess can lead to significantly different optimization paths and results. In all experiments, we can also see that SDG is the smoothest algorithm with fewer zigzags.

5.1 Time and Accuracy Analysis

In this section, we are going to analyze the time and accuracy performance of the algorithms compared to Scipy's implementation of the BFGS function. The error provided is the average of the log_{10} of the relative error. Called $\bar{x}$ the norm calculated with numpy and x our norm we have

$$\epsilon_r(A) := \frac{1}{N} \sum_{i=1}^{N} \log_{10} \left(\frac{|\bar{x} - x|}{|\bar{x}|} \right)$$

Table 1 represents a comparison of the Steepest Descent Gradient and Scipy Implementation of the BFGS algorithm where we can see that BFGS performed really slow as compared to the SDG because BFGS is sensitive to non-complex problems but interestingly converges with fewer iterations. In terms of error, SDG performed very well and reached a point close to the solution. Whereas, Scipy's BFGS also performed well with a relatively minimal error at the end.

STEEPEST DESCENT GRADIENT $\|\nabla f(\mathbf{x}_k)\|_2 \leq 1 \times 10^{-4}$						
Type	scipy, mean iterations and time		mean error	SDG, mean iterations and time		mean error
A	170	2.51×10^{1}	1.81×10^{-3}	464	6.76×10^{-1}	3.43×10^{-11}
B	96	4.18×10^{-2}	1.02×10^{-3}	186	1.40×10^{-2}	5.66×10^{-12}
C	243	1.58×10^{1}	4.31×10^{-4}	219	2.54×10^{-1}	2.69×10^{-7}
D	82	4.34×10^{-2}	7.02×10^{-3}	87	7.68×10^{-3}	6.97×10^{-11}
E	148	4.80×10^{-2}	1.25×10^{-3}	324	1.04×10^{-2}	4.93×10^{-11}
F	144	6.10×10^{0}	2.49×10^{-3}	67	6.41×10^{-2}	1.35×10^{-11}

Table 1: Iterations and time for the steepest descent gradient(SDG) algorithm compared with the Scipy Implementation of BFGS. Stopping criteria is $\|\nabla f(\mathbf{x}_k)\|_2 \leq 1 \times 10^{-4}$.

Table 2 represents the comparison of our implementation of BFGS with Scipy's implementation of BFGS. Both are performed equally but in terms of time, our implementation of BFGS performed well comparatively. In terms of Error, both algorithms converge almost to the same point but in some cases such as matrix types A, C, D, and F our implementation converges to a better point. Whereas, the table 3 gives a representation of the comparison of our CBFGS implementation with Scipy's implementation of BFGS where we can clearly see our implementation of CBFGS outperformed the former implementation in terms of time and accuracy. Although CBFGS took more iterations in matrices type A and B, it would be considered a better algorithm in terms of time efficiency as it reaches to a better solution.

6 Conclusion

In conclusion, this report compared the performance of Steepest Descent Gradient (SDG), BFGS, and Cautious BFGS (CBFGS) algorithms in estimating matrix norms across various matrix types. The results consistently demonstrated the superiority of CBFGS, which out-

BFGS $\|\nabla f(\mathbf{x}_k)\|_2 \leq 1 \times 10^{-4}$						
Type	scipy, mean iterations and time		mean error	BFGS, mean iterations and time		mean error
A	170	2.51×10^{1}	1.81×10^{-3}	184	1.79×10^{1}	1.33×10^{-3}
B	96	4.18×10^{-2}	1.02×10^{-3}	184	6.48×10^{-2}	1.16×10^{-9}
C	243	1.58×10^{1}	4.31×10^{-4}	211	1.06×10^{1}	1.45×10^{-2}
D	82	4.34×10^{-2}	7.02×10^{-3}	61	2.91×10^{-2}	2.08×10^{-3}
E	148	4.80×10^{-2}	1.25×10^{-3}	130	3.88×10^{-2}	1.26×10^{-3}
F	144	6.10×10^{0}	2.49×10^{-3}	155	4.99×10^{0}	1.74×10^{-2}

Table 2: Iterations and time for the BFGS algorithm compared with the Scipy Implementation of BFGS. Stopping criteria is $\|\nabla f(\mathbf{x}_k)\|_2 \leq 1 \times 10^{-4}$.

CBFGS $\|\nabla f(\mathbf{x}_k)\|_2 \leq 1 \times 10^{-4}$						
Type	scipy, mean iterations and time		mean error	CBFGS, mean iterations and time		mean error
A	170	2.51×10^{1}	1.81×10^{-3}	238	1.26×10^{0}	9.66×10^{-11}
B	96	4.18×10^{-2}	1.02×10^{-3}	149	1.68×10^{-2}	1.30×10^{-10}
C	243	1.58×10^{1}	4.31×10^{-4}	203	1.07×10^{0}	1.89×10^{-7}
D	82	4.34×10^{-2}	7.02×10^{-3}	78	1.61×10^{-2}	2.01×10^{-10}
E	148	4.80×10^{-2}	1.25×10^{-3}	92	1.80×10^{-2}	1.79×10^{-10}
F	144	6.10×10^{0}	2.49×10^{-3}	105	1.05×10^{0}	7.41×10^{-6}

Table 3: Iterations and time for the CBFGS algorithm compared with the Scipy Implementation of BFGS. Stopping criteria is $\|\nabla f(\mathbf{x}_k)\|_2 \leq 1 \times 10^{-4}$.

performed both SDG and standard BFGS in terms of efficiency and robustness, particularly when dealing with sparse and ill-conditioned matrices. Notably, our CBFGS implementation showed significant improvements in computational time compared to Scipy's BFGS, while maintaining comparable accuracy. All methods achieved similar levels of accuracy in terms of mean log10 error, indicating reliable convergence to solutions close to the true matrix norm. Overall, CBFGS emerges as the most effective method, offering a balanced compromise between convergence speed, accuracy, and robustness, making it a valuable tool for practitioners dealing with diverse and challenging matrix norm estimation tasks.

References

[1] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[2] D. G. Luenberger, “Introduction to linear and nonlinear programming”, 1973.

[3] J. E. Dennis and J. J. Moré, “A characterization of superlinear convergence and its application to quasi-newton methods”, *Mathematics of computation*, vol. 28, no. 126, pp. 549–560, 1974.

[4] J. E. Dennis Jr and J. J. Moré, “Quasi-newton methods, motivation and theory”, *SIAM review*, vol. 19, no. 1, pp. 46–89, 1977.

[5] D.-H. Li and M. Fukushima, “A modified bfgs method and its global convergence in nonconvex minimization”, *Journal of Computational and Applied Mathematics*, vol. 129, no. 1-2, pp. 15–35, 2001.

[6] R. Fletcher, *An overview of unconstrained optimization, algorithms for continuous optimization: The state of art, e. spedicato, ed*, 1994.

[7] J. Nocedal, “Theory of algorithms for unconstrained optimization”, *Acta numerica*, vol. 1, pp. 199–242, 1992.

[8] G. Yuan, Z. Sheng, B. Wang, W. Hu, and C. Li, “The global convergence of a modified bfgs method for nonconvex functions”, *Journal of Computational and Applied Mathematics*, vol. 327, pp. 274–294, 2018.

[9] P. Li, J. Lu, and H. Feng, “The global convergence of a modified bfgs method under inexact line search for nonconvex functions”, *Mathematical Problems in Engineering*, vol. 2021, no. 1, p. 8 342 536, 2021.