Computational Mathematics for Learning and Data Analysis

Author: *Luca de Martino* and *Raffaele Villani*

Email: l.demartino1@studenti.unipi.it and r.villani4@studenti.unipi.it

Abstract

(P) is the problem of estimating the matrix norm $||A||_2$ for a (possibly rectangular) matrix $A \in \mathbb{R}^{m\times n}$, using its definition as an (unconstrained) maximum problem.
(A1) is a standard gradient descent (steepest descent) approach.
(A2) is an algorithm of the class of Conjugate Gradient methods.

1 Introduction

Given the definition of a 2-norm vector

$$||\boldsymbol{x}||_2 := \sqrt{\boldsymbol{x^T x}} \tag{1.1}$$

we can induce the definition of a 2-norm matrix

$$||\boldsymbol{A}||_2 := \sup_{\boldsymbol{x}\in\mathbb{R}^n_{\neq 0}} \frac{||\boldsymbol{Ax}||_2}{||\boldsymbol{x}||_2} \tag{1.2}$$

with $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{x} \in \mathbb{R}^n_{\neq 0}$. Combining the equation 1.1 and the equation 1.2 we can write:

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

Given that the square root is a monotone function we can arrange the problem as:

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

Since we want to estimate the norm as an unconstrained minimization problem we specify the following:

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

where our objective function $f(x)$ is defined as:

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

2 Function Study

2.1 Gradient

Since the derivative of fraction in a generic $h(x)$ function is given by:

$$h'(x) = \frac{f'(x) \cdot g(x) - f(x) \cdot g'(x)}{(g(x))^2} \tag{2.1}$$

We can compute the gradient of our objective function this way:

$$\begin{aligned}\nabla f(x) &= \frac{(\boldsymbol{x^T Q x})'(\boldsymbol{x^T x}) - (\boldsymbol{x^T Q x})(\boldsymbol{x^T x})'}{(\boldsymbol{x^T x})^2} = \\ &= \frac{2\boldsymbol{Qx}(\boldsymbol{x^T x}) - (\boldsymbol{x^T Q x})2\boldsymbol{x}}{(\boldsymbol{x^T x})^2} = \frac{2\boldsymbol{Qx}}{\boldsymbol{x^T x}} - \frac{(\boldsymbol{x^T Q x})2\boldsymbol{x}}{(\boldsymbol{x^T x})^2}\end{aligned} \tag{2.2}$$

where $\boldsymbol{Q} = \boldsymbol{A^T A}$ is a $\mathbb{R}^{nxn}$ matrix. Since the function is defined in $\mathbb{R}^n_{\neq 0}$ and the gradient exists it is differentiable therefore continuous in $\mathbb{R}^n_{\neq 0}$.

2.2 Computational complexity of the gradient

First of all, to calculate the computational complexity of our function we have to deal with the matrix $\boldsymbol{Q}$ which requires $O(mn^2)$ operations, it requires a very big amount of operations, but at least we have to compute $\boldsymbol{Q}$ one time only and we could compute only half of the entries because it is a symmetric matrix.
Compute $\boldsymbol{Qx}$ instead require $O(n^2)$ operations, the same is valid for $\boldsymbol{x^T(Qx)}$, instead compute $\boldsymbol{x^T x}$ require $O(n)$ operations and the division is a constant time operation so $O(1)$. Therefore the total complexity to calculate our function is:

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

The same amount of computation is valid for the computation of the gradient since it requires similar operations, however, we can decrease the computational cost by rewriting the gradient this way:

$$\nabla f(x) = \frac{2\boldsymbol{Qx}}{\boldsymbol{x^T x}} - \frac{(\boldsymbol{x^T Q x})2\boldsymbol{x}}{(\boldsymbol{x^T x})^2} = \frac{2\boldsymbol{Qx} - (f(\boldsymbol{x}))2\boldsymbol{x}}{\boldsymbol{x^T x}}$$

then if we use the value which we have already calculated for our function like $f(\boldsymbol{x})$, $\boldsymbol{x^T x}$ and $\boldsymbol{Qx}$, we just have to compute $\boldsymbol{x}f(\boldsymbol{x})$ therefore the complexity of the gradient becomes:

$$C(\nabla f(\boldsymbol{x})) = O(n) \tag{2.4}$$

2.3 Properties of the function

First of all, we can induce some important properties by studying the matrix $\boldsymbol{Q} = \boldsymbol{A^T A}$. Given the matrix $\boldsymbol{A} \in \mathbb{R}^{mxn}$ we have that both for $\boldsymbol{A^T A}$ and $\boldsymbol{A A^T}$:

- The matrix is square: $\boldsymbol{A^T A} \in \mathbb{R}^{nxn}$
- The matrix is symmetric: $\boldsymbol{(A^T A)^T} = \boldsymbol{A^T (A^T)^T} = \boldsymbol{A^T A}$

$\boldsymbol{Q}$ is symmetric and it is also positive semi-definite as stated below:

$$\boldsymbol{x^T Q x} = \boldsymbol{x^T A^T A x} = ||\boldsymbol{A x}||^2 \geq 0 \iff \boldsymbol{A^T A} \succeq 0 \tag{2.5}$$

additionally if $\boldsymbol{A}$ is full rank $\forall x \neq 0$

$$\boldsymbol{x^T Q x} = \boldsymbol{x^T A^T A x} = ||\boldsymbol{A x}||^2 > 0 \iff \boldsymbol{A^T A} \succ 0 \tag{2.6}$$

2.3.1 Boundness

Since the definition of bounded function is:

$$|f(x)| \leq M \quad : \quad \exists M > 0 \tag{2.7}$$

we can see that our function is bounded

$$||f(x)|| \leq M \implies \left\| \frac{\boldsymbol{x^T Q x}}{x^T x} \right\| \leq \frac{||\boldsymbol{x}|| \; ||\boldsymbol{Q}|| \; ||\boldsymbol{x}||}{||\boldsymbol{x}||^2} \leq ||\boldsymbol{Q}|| \leq M \tag{2.8}$$

Given the following:

Proposition 2.1 *if a function f has a bounded gradient then f is Lipschitz continuous*

we can prove that our function is Lipschitz continuous in every set of the following form $\{x : ||x|| \geq r > 0\}$ since if r is equal to zero the function is not defined.

$$||\nabla f(x)|| \leq M \implies \left\| \frac{-2\boldsymbol{x}(\boldsymbol{x^T Q x})}{(\boldsymbol{x^T x})^2} + \frac{2\boldsymbol{Qx}}{\boldsymbol{x^T x}} \right\| \leq \left\| \frac{-2\boldsymbol{x}(\boldsymbol{x^T Q x})}{(\boldsymbol{x^T x})^2} \right\| + \left\| \frac{2\boldsymbol{Qx}}{\boldsymbol{x^T x}} \right\| =$$

$$= |-2| \left\| \frac{\boldsymbol{x}(\boldsymbol{x^T Q x})}{(\boldsymbol{x^T x})^2} \right\| + |2| \left\| \frac{\boldsymbol{Qx}}{\boldsymbol{x^T x}} \right\| \leq 2\frac{||\boldsymbol{x}|| \; ||\boldsymbol{x}|| \; ||\boldsymbol{Q}|| \; ||\boldsymbol{x}||}{||\boldsymbol{x}||^2} + 2\frac{||\boldsymbol{Q}|| \; ||\boldsymbol{x}||}{||\boldsymbol{x}||^2} \leq$$

$$2||\boldsymbol{Q}|| \; ||\boldsymbol{x}|| + 2\frac{||\boldsymbol{Q}||}{||\boldsymbol{x}||} = 2\frac{||\boldsymbol{Q}|| \; ||\boldsymbol{x}||^2 + 2||\boldsymbol{Q}||}{||\boldsymbol{x}||} = 2\frac{||\boldsymbol{Q}||(||\boldsymbol{x}||^2 + 1)}{||\boldsymbol{x}||} \leq M \tag{2.9}$$

Furthermore if we are in the unit sphere $\{x : ||x|| = 1\}$ we can estimate M as:

$$||\nabla f(x)|| \leq 4||\boldsymbol{Q}|| \tag{2.10}$$

2.3.2 Stationary points

Let suppose to have a stationary point $\overline{\boldsymbol{x}} \neq \mathbf{0}$ we want to find the place where $\nabla f(\overline{\boldsymbol{x}}) \neq \mathbf{0}$:

$$\nabla f(\overline{x}) = \frac{2\boldsymbol{Q}\overline{\boldsymbol{x}}}{\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}}} - \frac{(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{Q}\overline{\boldsymbol{x}})2\overline{\boldsymbol{x}}}{(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}})^2} = \frac{2\boldsymbol{Q}\overline{\boldsymbol{x}}(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}}) - (\overline{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{Q}\overline{\boldsymbol{x}})2\overline{\boldsymbol{x}}}{(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}})^2} = \mathbf{0}$$

$$2\boldsymbol{Q}\overline{\boldsymbol{x}}(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}}) - (\overline{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{Q}\overline{\boldsymbol{x}})2\overline{\boldsymbol{x}} = \mathbf{0}$$

$$2\boldsymbol{Q}\overline{\boldsymbol{x}}(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}}) = (\overline{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{Q}\overline{\boldsymbol{x}})2\overline{\boldsymbol{x}}$$

$$\boldsymbol{Q}\overline{\boldsymbol{x}}(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}}) = (\overline{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{Q}\overline{\boldsymbol{x}})\overline{\boldsymbol{x}}$$

$$\boldsymbol{Q}\overline{\boldsymbol{x}} = \frac{(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\boldsymbol{Q}\overline{\boldsymbol{x}})\overline{\boldsymbol{x}}}{(\overline{\boldsymbol{x}}^{\boldsymbol{T}}\overline{\boldsymbol{x}})}$$

$$\boldsymbol{Q}\overline{\boldsymbol{x}} = f(\overline{\boldsymbol{x}})\overline{\boldsymbol{x}} \tag{2.11}$$

then we can claim $\overline{\boldsymbol{x}}$ is a stationary point for our function if and only if $\overline{\boldsymbol{x}}$ is an eigenvector and $f(\overline{\boldsymbol{x}})$ is an eigenvalue for $\boldsymbol{Q}$.

2.3.3 Convexity

Let's assume that our function $f(x)$ is convex then the following inequality holds:

$$f(y) \geq f(x) + < \nabla f(x), y - x > \quad \forall x, y \in \mathbb{R}^n \tag{2.12}$$

but if exists two eigenvalues λ_i, λ_j and $\lambda_i > \lambda_j : i \neq j$ and the corresponding eigenvectors $\boldsymbol{x}_{\boldsymbol{\lambda}_i}, \boldsymbol{x}_{\boldsymbol{\lambda}_j}$ we can say that $f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) = \lambda_i$ so $f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) > f(\boldsymbol{x}_{\boldsymbol{\lambda}_j})$. But as stated above in 2.3.2 all the eigenvectors are stationary points so from the convexity inequality we would get $f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) \leq f(\boldsymbol{x}_{\boldsymbol{\lambda}_j})$ which contradicts the previous hypothesis.

3 Gradient Descent

In this section, we are going to show how to use the Steepest Descent algorithm to solve our minimization problem. The algorithm makes steps in the opposite direction of the gradient of the function, which is $\nabla f(\boldsymbol{x})$, this is why this algorithm is also called "Gradient Descent". The pseudo-code of the algorithm is shown below:

Algorithm 1 Gradient Descent

```
1: procedure GD($x_0$, $\epsilon$)
2:     while $||\nabla f(x_k)|| > \epsilon$ do
3:         $d_k = -\nabla f(x_k)$
4:         $\phi(\alpha) = f(x_k + \alpha d_k)$
5:         $\alpha_k = \min_\alpha \phi(\alpha)$
6:         $x_{k+1} = x_k + \alpha_k d_k$
7:     return $x_{k+1}$
```

3.1 Step-size optimization

As we can notice in line 5 of the algorithm, the formula to compute the step-size α is:

$$\phi(\alpha) = \min f(x + \alpha d) \tag{3.1}$$

then in order to minimize the function 3.1 we have to solve this equation $\phi'(\alpha) = 0$. Let's start managing $f(x + \alpha d)$ to let the derivation easier.

$$\begin{aligned} f(\boldsymbol{x} + \alpha \boldsymbol{d}) &= \frac{(\boldsymbol{x} + \alpha \boldsymbol{d})^T \boldsymbol{Q} (\boldsymbol{x} + \alpha \boldsymbol{d})}{(\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d})} = \\ &= \frac{\boldsymbol{x^T Q x} + 2\alpha(\boldsymbol{d^T Q x}) + \alpha^2(\boldsymbol{d^T Q d})}{(\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d})} \end{aligned} \tag{3.2}$$

Now to derive the function 3.2 we can claim the following property:
Given $\boldsymbol{Q} \in \mathbb{R}^{nxn}$ symmetric:

$$(\boldsymbol{x^T Q y}) = (\boldsymbol{x^T Q y})^T = \boldsymbol{y^T Q^T x} = \boldsymbol{y^T Q x}$$

Then:

$$\phi'(\alpha) = [\frac{(2\boldsymbol{d^T Q x} + 2\alpha \boldsymbol{d^T Q d})(\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}) -}{((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2}$$

$$\frac{((\boldsymbol{x^T Q x} + 2\alpha \boldsymbol{d^T Q x} + \alpha^2 \boldsymbol{d^T Q d})(2(\boldsymbol{x} + \alpha \boldsymbol{d})^T \boldsymbol{d}))}{((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2}] =$$

$$= \frac{[2\alpha^3((\boldsymbol{d^T Q d})\boldsymbol{d^T d}) + 2\alpha^2((\boldsymbol{d^T Q x})\boldsymbol{d^T d} + 2(\boldsymbol{d^T Q d})\boldsymbol{d^T x}) +}{((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2}$$

$$\frac{2\alpha((\boldsymbol{d^T Q d})\boldsymbol{x^T x}) + 2((\boldsymbol{d^T Q x})\boldsymbol{d^T x}) + 2((\boldsymbol{d^T Q x})\boldsymbol{x^T x})] -}{((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2}$$

$$\frac{[2\alpha^3((\boldsymbol{d^T Q d})\boldsymbol{d^T d}) + 2\alpha^2((\boldsymbol{d^T Q x})\boldsymbol{d^T d} + 2((\boldsymbol{d^T Q x})\boldsymbol{d^T d}) +}{((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2}$$

$$\frac{2\alpha(2(\boldsymbol{d^T Q x})(\boldsymbol{x^T d})) + (\boldsymbol{x^T Q x})\boldsymbol{d^T d}) + 2((\boldsymbol{x^T Q x})\boldsymbol{x^T d}]}{((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2}$$

We can now reduce the above equation by obtaining:

$$\phi'(\alpha) = \frac{2\alpha^2((\boldsymbol{d^T Q x})\boldsymbol{d^T d} - (\boldsymbol{d^T Q d})\boldsymbol{d^T x}) +}{(\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d})^2}$$

$$\frac{2\alpha((\boldsymbol{x^T Q x})\boldsymbol{d^T d} - (\boldsymbol{d^t Q x})\boldsymbol{x^T x}) + 2(\boldsymbol{x^T Q x})\boldsymbol{x^T d} - (\boldsymbol{d^T Q x})\boldsymbol{x^T x}}{((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2} \tag{3.3}$$

Furthermore, we can rewrite the equation 3.3 as a second-degree polynomial in α:

$$\phi'(\alpha) = \frac{a\alpha^2 + b\alpha + c}{P(\alpha)} \tag{3.4}$$

$$a = \boldsymbol{(d^T Q d) d^T x - (d^T Q x) d^T d}$$

$$b = \boldsymbol{(d^T Q d)(x^T x) - (x^T Q x) d^T d}$$

$$c = \boldsymbol{(d^T Q x) x^T x - (x^T Q x) x^T d}$$

$$P(\alpha) = ((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2$$

The closed form is indicated in the above equation 3.4 and to reach a stationary point along this direction, we need the denominator not equal to 0:

$$P(\alpha) = ((\boldsymbol{x} + \alpha \boldsymbol{d})^T (\boldsymbol{x} + \alpha \boldsymbol{d}))^2 = ((\boldsymbol{x} - \alpha \nabla f(\boldsymbol{x}))^T (\boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})))^2 =$$

$$= ||(\boldsymbol{x} - \alpha \nabla f(\boldsymbol{x}))||_2^2 = 0 \iff \boldsymbol{x} = \alpha \nabla f(\boldsymbol{x})$$

Consequently, there is a solution for α only if $\boldsymbol{x}$ and $\nabla f(\boldsymbol{x})$ are linearly dependent but we can demonstrate these vectors are orthogonal:

$$< \boldsymbol{x}, \nabla f(\boldsymbol{x}) >= \boldsymbol{x}^T \cdot \nabla f(\boldsymbol{x}) = \boldsymbol{x}^T \cdot \frac{2\boldsymbol{Q x}(\boldsymbol{x^T x}) - (\boldsymbol{x^T Q x})2\boldsymbol{x}}{(\boldsymbol{x^T x})^2} =$$

$$= \frac{2(\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x})(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x}) - 2(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{x})(\boldsymbol{x}^T\boldsymbol{x})}{(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x})^2} = 0 \quad \forall \boldsymbol{x} \neq 0 \tag{3.5}$$

Consequently the equation 3.4 is well defined $\forall\alpha > 0$ and $\phi'(0) < 0$ since we are using $-\nabla f(\boldsymbol{x})$ as descent direction. We can also state that $\phi'(\alpha)$ is continuous and the roots of $a\alpha^2 + b\alpha + c$ are stationary points for $\phi(\alpha)$ and $\phi(\overline{\alpha}) < \phi(0)$ where $\overline{\alpha}$ is a root of the polynomial.
Finally, the time complexity of computing the step size is $O(n^2)$ because calculating a, b, and c involves the same vector and matrix operations used in the gradient then:

$$C(\phi'(\alpha)) = O(n^2) \tag{3.6}$$

Using the $\overline{\alpha}$ such that $\phi'(\overline{\alpha}) = 0$ it satisfies the Wolfe condition:

$$\overline{\alpha}: \quad \phi'(\overline{\alpha}) = 0: \quad \phi'(\overline{\alpha}) \geq m_3\phi'(0) \tag{3.7}$$

where m_3 is positive since $m_1 < m_3 < 1$ given $0 < m_1 < 1$ and as we said before $\phi'(0) < 0$.

3.2 Convergence Analysis

In this subsection, for dealing with the convergence analysis, we can consider the fact the function is Lipschitz continuous out of a ball $B(0,\epsilon), \forall\epsilon > 0$ then we can search the solution in the domain ($D = \mathbb{R}^n \backslash B(0,\epsilon)$). Indeed, we can assume $||\boldsymbol{x_0}||_2 > \epsilon$ then applying in $\mathbb{R}^n$ the Generalized Pitagora theorem:

$$||\boldsymbol{x}+\boldsymbol{y}||_2^2 = (\boldsymbol{x}+\boldsymbol{y})^{\boldsymbol{T}}(\boldsymbol{x}+\boldsymbol{y}) = ||\boldsymbol{x}||_2^2 + ||\boldsymbol{y}||_2^2 + 2\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{y} \tag{3.8}$$

we can demonstrate that the succession of point starting from $||\boldsymbol{x_0}||$ belongs to D which is $||\boldsymbol{x_{k+1}}||_{\boldsymbol{2}} > ||\boldsymbol{x_k}||_{\boldsymbol{2}}$.

$$||\boldsymbol{x_{k+1}}||_2^2 = ||\boldsymbol{x_k} - \overline{\alpha}\nabla f(\boldsymbol{x_k})||_2^2 = ||\boldsymbol{x_k}||_2^2 + ||\overline{\alpha}\nabla f(\boldsymbol{x_k})||_2^2 + 2\boldsymbol{x_k^T}\overline{\alpha}\nabla f(\boldsymbol{x_k})$$

We know the last term of the equation above is 0 because of the orthogonality of the point and the gradient as demonstrated in the equation 3.5 then we can rearrange this way:

$$||\boldsymbol{x_{k+1}}||_2^2 = ||\boldsymbol{x_k}||_2^2 + ||\overline{\alpha}\nabla f(\boldsymbol{x_k})||_2^2 > ||\boldsymbol{x_k}||_2^2$$

To prove the convergence we would like to take advantage of Zoutendijk's theorem [6, p. 38]. To apply it we should use the hypothesis just matched above, which proves that the succession of points belongs to D where the function is Lipschitz continuous. We have also demonstrated that our function is bounded below from the equation 2.8 and the Wolfe condition is satisfied as in the equation 3.7. The theorem claims the following:

Theorem 3.1 *Consider any iteration of the form $x_{k+1} = x_k + \alpha_k d_k$ where d_k is a descent direction and α_k satisfies the Wolfe conditions. Suppose that f is bounded below in $\mathbb{R}^n$ and f is continuously 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 of the iteration. Assume also that the gradient ∇f is Lipschitz continuous on $\mathcal{N}$, that is, there exists a constant $L > 0$ such that*

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

Then

$$\sum_{K \geq 0} cos^2\theta_k ||\nabla f_k||^2 < \infty \quad cos\theta_k := \frac{d_k^T \nabla f(x_k)}{||d_k^T||_2 ||\nabla f(x_k)||_2}$$

The problem with using this theorem is that we cannot use the set $\mathcal{N}$ as defined in the assumptions of the theorem because the function is not differentiable in 0. At this point, we can change the set $\mathcal{N} = D$, where $D = \{\mathbb{R}^n \backslash B(0, \epsilon))\}$ as defined before, this way we know our function will be continuous and differentiable in this set. To check the goodness of the results we just re-do the steps of the proof using the new set to arrive at the same conclusions.
Given $x_{k+1} = x_k + \alpha_k d_k$, where $x_{k+1}, x_k \in \mathcal{N}$, we re-define the Armijo and the Wolfe condition we need for the proof:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T d_k$$

$$\nabla f(x_k + \alpha_k d_k)^T d_k \geq c_2 \nabla f_k^T d_k$$

we can subtract $\nabla f_k d_k$ from the Wolfe condition to obtain:

$$(\nabla f_{k+1} - \nabla f_k)^T d_k \geq (c_2 - 1) \nabla f_k d_k$$

Then given the Lipschitz condition:

$$||\nabla f_{k+1} - \nabla f_k|| \leq L||x_{k+1} - x_k||$$

we can substitute $x_{k+1} = x_k + \alpha_k d_k$ to obtain:

$$||\nabla f_{k+1} - \nabla f_k|| \leq L\alpha_k ||d_k||$$

consequently, we can write:

$$(\nabla f_{k+1} - \nabla f_k)^T d_k \leq L \alpha_k d_k^T d_k$$

by combining the relation of the Wolfe condition and the Lipschitz continuity we obtain:

$$L \alpha_k d_k^T d_k \geq (c_2 - 1) \nabla f_k d_k$$

therefore

$$\alpha_k \geq \frac{c_2 - 1}{L} \frac{\nabla f_k d_k}{||d_k||^2}$$

Now substituting α_k with the above inequality into the Armijo condition we can get:

$$f_{k+1} \leq f_k - c_1 \frac{1 - c_2}{L} \frac{(\nabla f_k d_k)^2}{||d_k||^2}$$

Given θ_k as the angle between ∇f_k and d_k we can rewrite the above equation:

$$f_{k+1} \leq f_k - c * \frac{||\nabla f_k||^2 ||d_k||^2 cos^2\theta_k}{||d_k||^2} = f_k - c * ||\nabla f_k||^2 cos^2\theta_k$$

where $c = \frac{1-c_2}{L}$. Now summing the above expression's overall indices less than or equal to k, we obtain

$$f_{k+1} \leq f_0 - c \sum_{k=0}^{\infty} cos^2\theta_k ||\nabla f_k||^2$$

Since f is bounded below, we have that $f_0 - f_{k+1}$ is less than some positive constant, for all k. Hence:

$$\sum_{k=0}^{\infty} cos^2\theta_k ||\nabla f_k||^2 < \infty$$

We know the rate of convergence is linear on strongly convex quadratic function because of the theorem 3.2 but as explained in the book *the rate-of-convergence behavior of the steepest descent method is essentially the same on general nonlinear objective functions* [6, p. 43]. To use the next theorem we need to define the norm on Q which is given by:

$$||x||_Q^2 = x^T Q x \tag{3.9}$$

Theorem 3.2 *When the steepest descent method with exact line searches is applied to the strongly convex quadratic function the error norm satisfies:*

$$||\boldsymbol{x_{k+1}} - \boldsymbol{x^*}||_Q^2 \leq \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)^2 ||\boldsymbol{x_k} - \boldsymbol{x^*}||_Q^2 \tag{3.10}$$

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

Unfortunately, the theorem does not hold in every situation because we cannot guarantee $\lambda_1 > 0$ (therefore that Q is positive definite) but can claim that $\lambda_1 \geq 0$ because for sure we know Q is positive semi-definite. Nevertheless, as we will see in the section 5.1, in practice we will generally achieve a linear convergence.

3.3 Algorithm Complexity

We have just proved the complexity for f(x) and $\nabla f(x)$ in the subsection 2.2 and for $\phi'(\alpha)$ in the equation 3.6. Then we can assert that the complexity for each iteration of the algorithm is:

$$C(GD) = O(n^2) \tag{3.11}$$

4 Conjugate Gradient

In this section, we approach the Conjugate Gradient method and use it to optimize our function. The pseudo-code of the algorithm, which uses the Fletcher-Reeves method to calculate the value of β_k, is the following:

Algorithm 2 Conjugate Gradient

```
1: procedure CG($x_0$, $\epsilon$)
2:     while $||\nabla f(x_k)|| > \epsilon$ do
3:         $\beta_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})}$
4:         $d_k = -\nabla f(x_k) + \beta_k d_{k-1}$
5:         $\phi(\alpha) = f(x_k + \alpha d_k)$
6:         $\alpha_k = \min_\alpha \phi(\alpha)$
7:         $x_{k+1} = x_k + \alpha_k d_k$
8:     return $x_k$
```

4.1 Step-size optimization

As we have already done for the Gradient Descent algorithm, we want to perform an exact search along the direction d_k to find the optimal value of α, this is given by $\phi'(\alpha) = 0$:

$$\phi'(\alpha) = \nabla f(\boldsymbol{x_{k+1}})^T \boldsymbol{d_k} = \nabla f(\boldsymbol{x_k} + \boldsymbol{\alpha_k d_k})^T \boldsymbol{d_k} \tag{4.1}$$

Before proceeding with the analysis we can also prove the gradient and the descent direction have opposite directions, then $\nabla f(\boldsymbol{x_k})^T \boldsymbol{d_k} \leq 0$:

$$\nabla f(\boldsymbol{x_k})^T \boldsymbol{d_k} = \nabla f(\boldsymbol{x_k}^T)(-\nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{d_{k-1}}) =$$

$$-||\nabla f(\boldsymbol{x_k})||_2^2 + \beta_k \nabla f(\boldsymbol{x_k})^T \boldsymbol{d_{k-1}} \leq 0 \tag{4.2}$$

As we know the $\nabla f(\boldsymbol{x_k})$ and $\boldsymbol{d_{k-1}}$ are orthogonal consequently the dot product is equal to 0, and as a consequence, the value of 4.2 is always less than 0. We can use the same reasoning used in the section 3.1 to obtain the value of α along the direction $\boldsymbol{d}$ that minimizes the function $\phi(\alpha) = f(\boldsymbol{x} + \alpha\boldsymbol{d})$. So we have to solve the equation

$$\phi'(\alpha) = \frac{a\alpha^2 + b\alpha + c}{P(\alpha)} = 0 \tag{4.3}$$

Where:

$$a = \boldsymbol{(d^T Q d) d^T x - (d^T Q x) d^T d}$$

$$b = \boldsymbol{(d^T Q d)(x^T x) - (x^T Q x) d^T d}$$

$$c = \boldsymbol{(d^T Q x) x^T x - (x^T Q x) x^T d}$$

$$P(\alpha) = ((\boldsymbol{x} + \alpha\boldsymbol{d})^T(\boldsymbol{x} + \alpha\boldsymbol{d}))^2 \tag{4.4}$$

Now we have to prove we can always have the denominator different from zero ($P(\alpha) \neq 0$). If we succeed to prove it we can claim the roots of the equations are stationary points along the direction $\boldsymbol{d}$.

$$\begin{gathered} P(\alpha) \neq 0 \\ ((\boldsymbol{x} + \alpha\boldsymbol{d})^T(\boldsymbol{x} + \alpha\boldsymbol{d})) \neq 0 \\ ||\boldsymbol{x} + \alpha\boldsymbol{d}||_2^2 \neq 0 \end{gathered} \tag{4.5}$$

To show this equation is not equal to zero we have to prove $\boldsymbol{x_k}$ and d_k are linearly independent then:

$$| < \boldsymbol{x_k}, \boldsymbol{d_k} > | < ||\boldsymbol{x_k}||_2 \cdot ||\boldsymbol{d_k}||_2 \tag{4.6}$$

we can rewrite this way:

$$||\boldsymbol{x_k^T d_k}||_2 = \boldsymbol{x_k^T}(-\nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{d_{k-1}})||_2 = ||-\boldsymbol{x_k^T}\nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{x_k^T d_{k-1}}||_2 = ||\beta_k \boldsymbol{x_k^T d_{k-1}}||_2 \leq$$

$$\leq ||\boldsymbol{x_k}||_2 ||\beta_k \boldsymbol{d_{k-1}}||_2 < ||\boldsymbol{x_k}||_2 ||\boldsymbol{d_k}||_2 \iff ||\beta_k \boldsymbol{d_{k-1}}||_2 < ||\boldsymbol{d_k}||_2$$

but we know that:

$$||\boldsymbol{d_k}||_2 = || - \nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{d_{k-1}}||_2^2$$

and we also know $\nabla f(\boldsymbol{x_k})$ and $\boldsymbol{d_{k-1}}$ are orthogonal, then applying the generalized Pitagora theorem (equation 3.8):

$$||\boldsymbol{d_k}||_2 = || - \nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{d_{k-1}}||_2^2 = || - \nabla f(\boldsymbol{x_k})||_2^2 + ||\beta_k \boldsymbol{d_{k-1}}||_2^2 > ||\beta_k \boldsymbol{d_{k-1}}||_2^2$$

which is true because surely $||\nabla f(\boldsymbol{x_k})||_2^2 > 0$ therefore we have proved $P(\alpha) \neq 0$ then we can always find a solution for $\phi'(\alpha) = 0$.

4.2 Convergence Analysis

Starting with the analysis of the convergence we can take care of the fact that the function is Lipschitz continuous out of a ball $B(0, \epsilon), \forall \epsilon > 0$ then we can search the solution in this domain ($D = \mathbb{R}^n \backslash B(0, \epsilon)$). We can start with the assumption that $||\boldsymbol{x_0}||_2^2 \geq \epsilon > 0$, this way we can take an $||\boldsymbol{x_k}||_2 \in D, \forall k \geq 0$ where $||\boldsymbol{x_k}||_2 > ||\boldsymbol{x_{k-1}}||_2$ because if we compute $||\boldsymbol{x_1}||_2^2$:

$$||\boldsymbol{x_1}||_2^2 = ||\boldsymbol{x_0}||_2^2 + 2\alpha_0 < \boldsymbol{x_0}, \boldsymbol{d_0} > + \alpha_0^2 ||d_0||_2^2 > 0$$

We can claim $2\alpha_0 < \boldsymbol{x_0}, \boldsymbol{d_0} >$ is 0 since $\boldsymbol{d_0} = -\nabla f(\boldsymbol{x_0})$ and $(\alpha_0^2 ||d_0||_2^2) > 0$ consequently $||\boldsymbol{x_1}||_2 > ||\boldsymbol{x_0}||_2 \geq \epsilon > 0$ then we can generalize for $||\boldsymbol{x_{k+1}}||_2 \geq \epsilon > 0, \forall k > 0$ obtaining:

$$||\boldsymbol{x_{k+1}}||_2^2 = ||\boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}||_2^2 \geq ||\boldsymbol{x_k}||_2^2 \quad \forall k > 0$$

$$(\boldsymbol{x_k} + \alpha_k \boldsymbol{d_k})^T (\boldsymbol{x_k} + \alpha_k \boldsymbol{d_k}) = ||\boldsymbol{x_k}||_2^2 + \alpha_k^2 ||d_k||_2^2 + 2\alpha_k x_k^T d_k > ||x_k||_2^2 \tag{4.7}$$

In the equation 4.7 we have that $\alpha_k > 0$ and we can demonstrate that $\boldsymbol{x_k^T d_k} > 0$ then we can explicit this term this way:

$$\boldsymbol{x_k^T d_k} = -\boldsymbol{x_k} \nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{x_k^T d_{k-1}}$$

But as we have proved in the equation 3.5 the first term is 0 and considering $\beta_k > 0$ we can write:

$$\boldsymbol{x_k^T d_k} = \beta_k (\boldsymbol{x_{k-1}} + \alpha_{k-1} \boldsymbol{d_{k-1}})^T \boldsymbol{d_{k-1}} =$$

$$= \beta_k \boldsymbol{x_{k-1}^T d_{k-1}} + \beta_k \alpha_{k-1} ||d_{k-1}||_2^2 \tag{4.8}$$

We can call the second member of the equation above γ_{k-1}, which is greater than zero, and then we can use the same passages $\forall k > 0$ and find all the γ_i as:

$$\gamma_i = (\prod_{j=k+1-i}^{k+1} \beta_j) a_i ||\boldsymbol{d_i}||_2^2 > 0 \quad \forall i \geq 0$$

$$\boldsymbol{x_k^T d_k} = \beta_k \boldsymbol{x}_{k-1}^T \boldsymbol{d_{k-1}} + \sum_{i=0}^{k-1} \gamma_i > 0 \tag{4.9}$$

Instead for k = 0:

$$\boldsymbol{x_0^T d_0} = \gamma_0 = -\boldsymbol{x_0^T} \nabla f(\boldsymbol{x_0}) = 0 \tag{4.10}$$

This way, finally, we have proved that given $||\boldsymbol{x_0}||_2 \geq \epsilon > 0$ the succession of point belongs to ($D = \mathbb{R}^n \backslash B(0, \epsilon)$) in which the function is lipschitz continuous. Finally giving the Lipschitz continuity, the limitation below (as shown in the 2.9 equation), and the differentiable (as shown in the 2.1 subsection) using the Zoutendijk's theorem we can claim the convergence of the algorithm, but as explained in the section 3.2 we can't use the set $\mathcal{N}$ as defined in the theorem, but we are going to use $\mathcal{N} = D$ (the full proof is shown [6, p. 127-131]).

The way of calculating beta brings not a few differences if we are dealing with a nonquadratic and nonconvex function, e.g. Polak-Ribière equals Fletcher-Reeves only if the function is a strongly convex quadratic function, furthermore, we know that *when applied to general nonlinear functions with inexact line searches, however, the behavior of the two algorithms differs markedly. Numerical experience indicates that Algorithm PR tends to be the more robust and efficient of the two. A surprising fact about Algorithm PR is that the strong Wolfe conditions do not guarantee that p_k is always a descent direction* [6, p.122].

For these reasons, several theorems demonstrate the rate of convergence of the algorithm taking on the beta used. *In general Crowder and Wolfe [4] shows that the rate of convergence is linear, and shows by constructing an example that Q-superlinear convergence is not achievable* [6, p.132].

In addition, we can theoretically define the restart, which serves to periodically refresh the algorithm, erasing old information that may not be beneficial, which is performed whenever two consecutive gradients are far from orthogonal, as measured by the test:

$$\frac{|\nabla f_k^T \nabla f_{k-1}|}{||\nabla f_k||^2} \geq v \tag{4.11}$$

where a typical value for the parameter is 0.1. This way, using the restart, we can apply to both Fletcher-Reeves and Polak-Ribière methods the results presented by: Cohen [3] and Burmeister [2] prove n-step quadratic convergence (5.51) for general objective functions and Ritter [1] shows that in fact, the rate is super quadratic:

$$||x_{k+n} - x*|| = o(||x_k - x*||^2)$$

Anyway *because nonlinear conjugate gradient methods can be recommended only for solving problems with large n. Restarts may never occur in such problems because an approximate solution may be located in fewer than n steps. Hence, nonlinear CG methods are sometimes implemented without restarts, or else they include strategies for restarting that are based on considerations other than iteration counts* [6, p.124].

4.3 Algorithm Complexity

As we also said in the 3.3 subsection we have just proved the complexity for $f(x)$ and $\nabla f(x)$ in the subsection 2.2 and for $\phi'(\alpha)$ in the equation 3.6. Concerning the Gradient Descent, in this case, we also have to take care of β_k and d_k. β_k can be obtained in $O(n)$ but saving the norm of the gradient used for the stopping condition, we can obtain it just with a division. d_k is obtained as sum and a multiplication of $\mathbb{R}^n$ vectors in $O(n)$. Then we can assert the complexity for each iteration of the algorithm is:

$$C(CG) = O(n^2) \tag{4.12}$$

5 Results and Comparisons

All the experiments are run on a Lenovo Yoga S740 with an Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 16GB RAM. We randomly generate $\boldsymbol{x}$ and different kinds of matrices with different shapes and densities, as indicated in the following list:

- Type $M_1^{500x100}$, d = 1
- Type $M_2^{2000x50}$, d = 1
- Type $M_3^{2000x50}$, d = 0.25
- Type $M_4^{50x2000}$, d = 1
- Type $M_5^{1000x1000}$ ill-conditioned, d = 1 and condition number = 10^5

Furthermore, the evaluation is dependent on the following parameters:

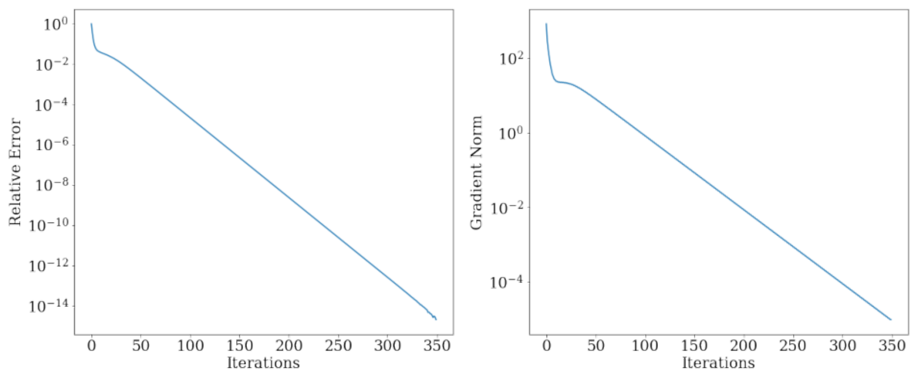
- **x** which is the starting vector
- **Maximum Number Iterations** which is the maximum number of iterations the algorithm is allowed to perform
- $\boldsymbol{\epsilon}$ which is the accuracy in the stopping condition
- **min** $\boldsymbol{\alpha}$ which is the minimum step-size

For all the experiments we have used a *Maximum Number Iterations*: 500, a *min* $\boldsymbol{\epsilon}$: 10^{-5} and *min* α : 10^{-16}.

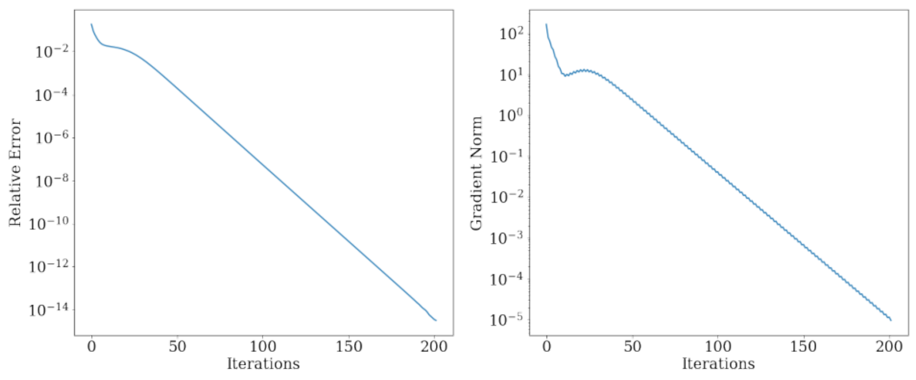
5.1 Gradient Descent Performance Evaluation

This subsection evaluates the performance analysis of our algorithm and compares the results with the numpy library [5], using the time and the accuracy as metrics. In the following plots, in natural logarithmic scale, we represent the trend of the Norms of the Gradient and of the Relative Error which is equal to $|f(x_k) - f_*|/|f_*|$, where f_* is the result obtained from the numpy algorithm.

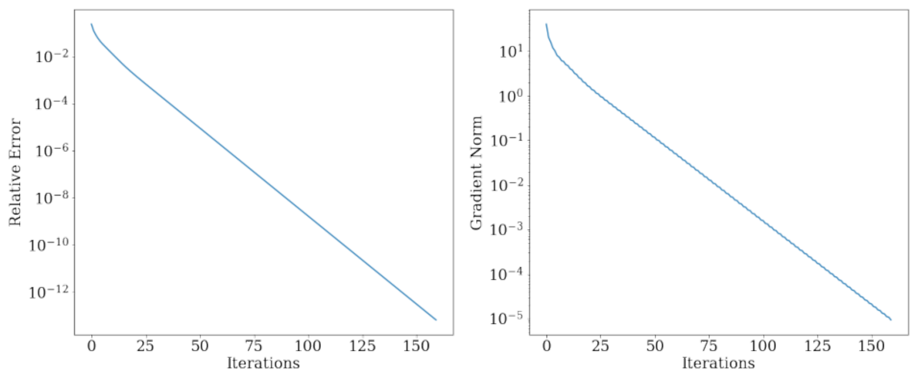
Matrix Type M1 Relative error and Gradient norm

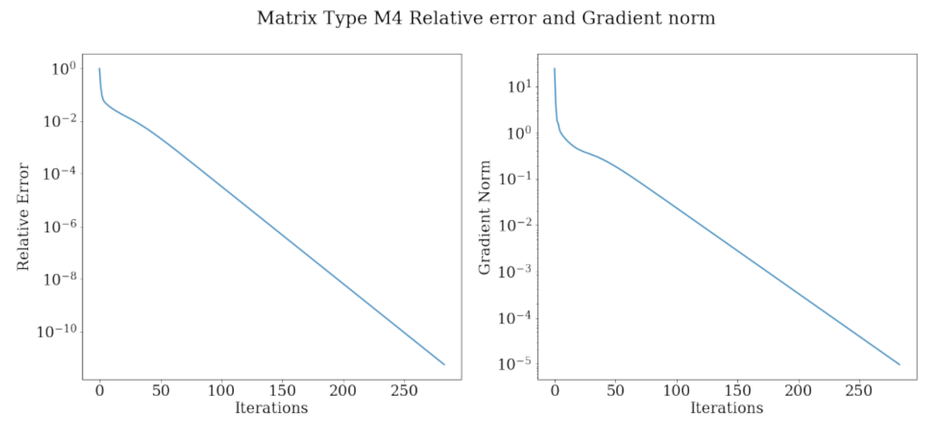


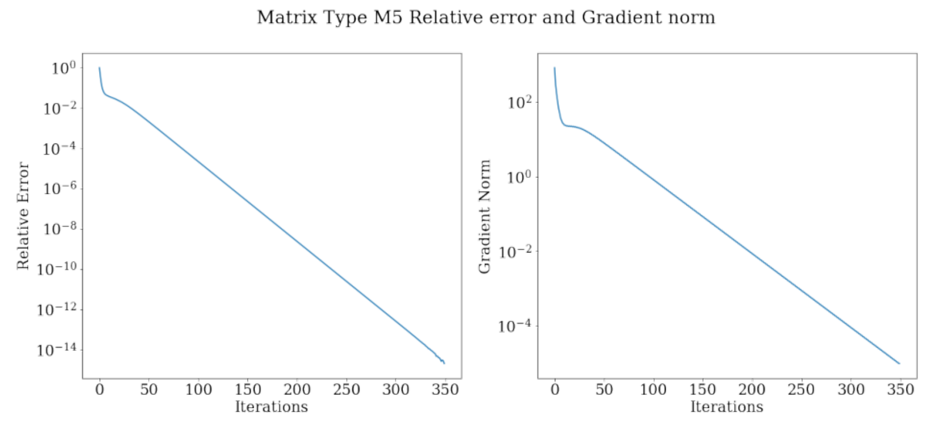
Matrix Type M2 Relative error and Gradient norm



Matrix Type M3 Relative error and Gradient norm







In all the experiments represented in the plots, we can see how the rate of convergence is linear, this fulfills our expectation given by the theory as demonstrated in 3.2 subsection. In addition, the algorithm is pretty stable since it can achieve a good approximation of the result obtained with the off-shell algorithm.

5.2 Gradient Descent Time Evaluation

To analyze the time performance of our algorithm implementation, we ran our algorithm and the numpy algorithm 1000 times since the performance of the algorithm can change based on the initial starting point $\boldsymbol{x_0}$, in the following table we will report the meantime and the standard deviation for both the algorithm and finally the error column which represents the relative error between our algorithm and the numpy one.

As we can see in the table the numpy algorithm performs much better than our algorithm on M_1, M_2 and M_3 matrices but it is worse on the M_4 and M_5 matrices, probably this is because the numpy one does not rely on an iterative approach but on a

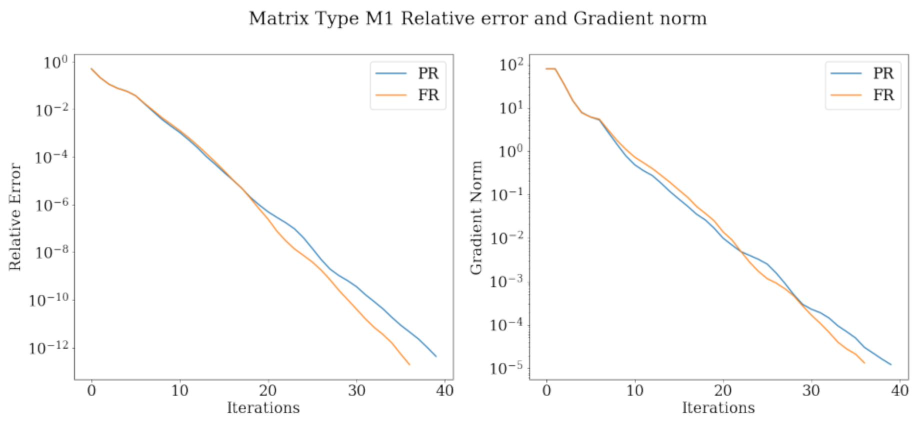
Matrix	GD(Time)	np.norm(Time)	Error
M_1	57.3 ms ± 9.16 ms	9.98 ms ± 1.17 ms	$5.61e^{-16}$
M_2	35.7 ms ± 9.73 ms	5.41 ms ± 761 µs	$6.88e^{-16}$
M_3	22.7 ms ± 2.32 ms	4.42 ms ± 260 µs	$1.17e^{-15}$
M_4	663 ms ± 55 ms	7.4 ms ± 556 µs	$5.18e^{-12}$
M_5	324 ms ± 37.5 ms	504 ms ± 44.1 ms	$2.47e^{-15}$

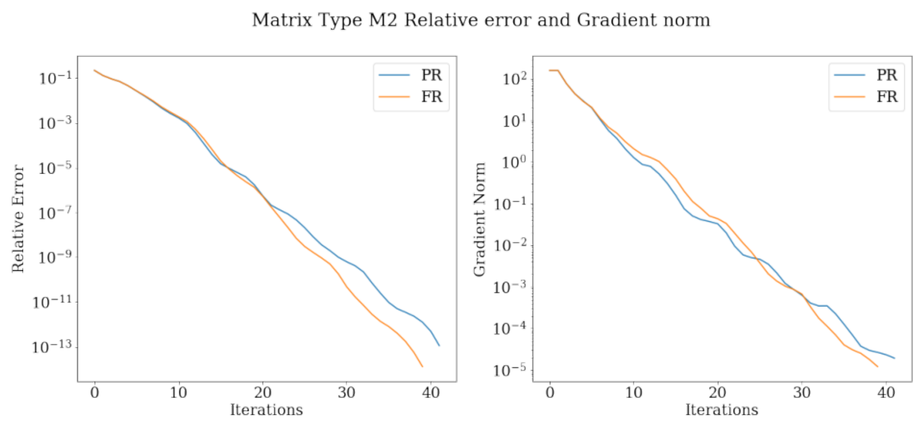
Table 1: Time performance and error difference between algorithm and numpy tool.

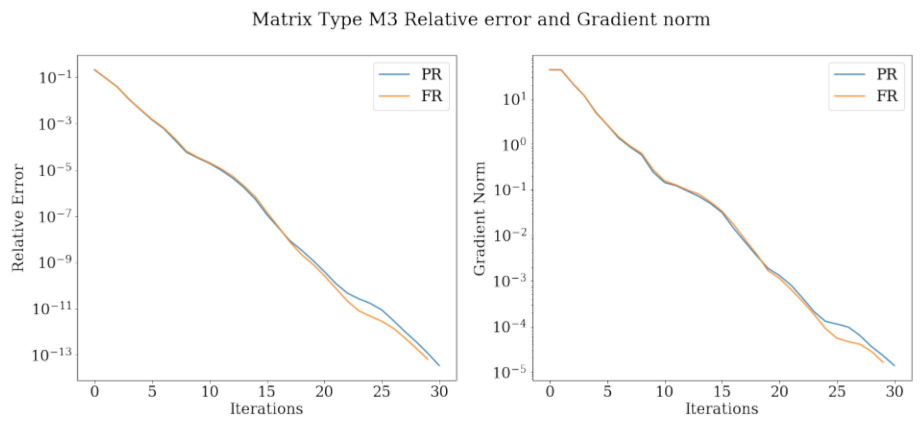
direct one. Consequently, the numpy approach generates more accurate results, but it is getting slower and slower as soon the dimension of the matrix increase.

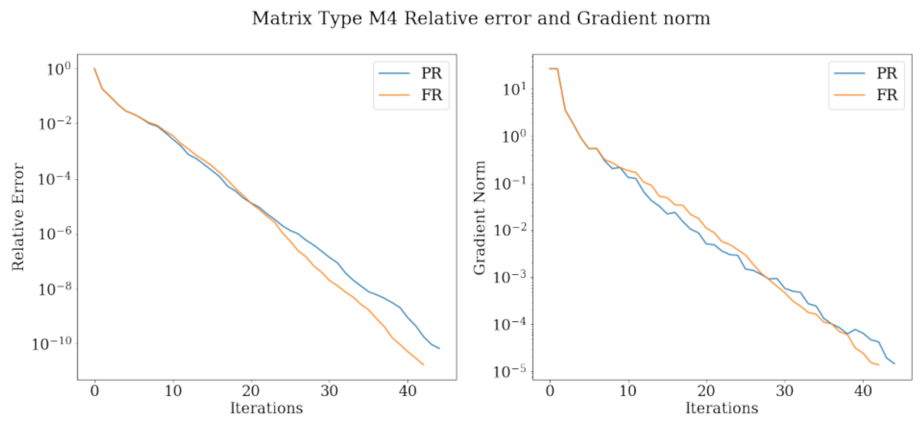
5.3 Conjugate Gradient Performance Evaluation

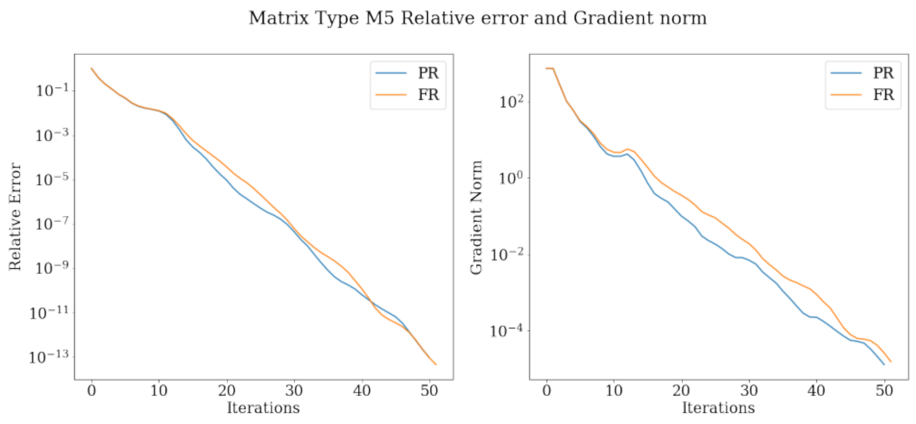
In this subsection to analyze the performance of our Conjugate Gradient descent, we compare our implementation with scipy one [7], using the time and the accuracy as metrics as in the gradient descent subsection 5.1. The plots are again represented in logarithmic scale and represent the trend of the Norms of the Gradient and the Relative Error.




Matrix Type M2 Relative error and Gradient norm
Relative Error
Gradient Norm
Iterations
PR
FR


Matrix Type M3 Relative error and Gradient norm
Relative Error
Gradient Norm
Iterations
PR
FR


Matrix Type M4 Relative error and Gradient norm
Relative Error
Gradient Norm
Iterations
PR
FR



The convergence rate of the Conjugate Gradient in our experiments is linear, which is what we expected from the theory in the subsection 4.2 since we have decided not to implement the restart. Anyway, we can claim that the conjugate convergence rate is faster than the gradient descent one since it always converges in a maximum of 50 iterations. It is easily explainable by looking at the following plot where we compare the convergence rate of the conjugate gradient with the gradient descent one in the matrix M_1.

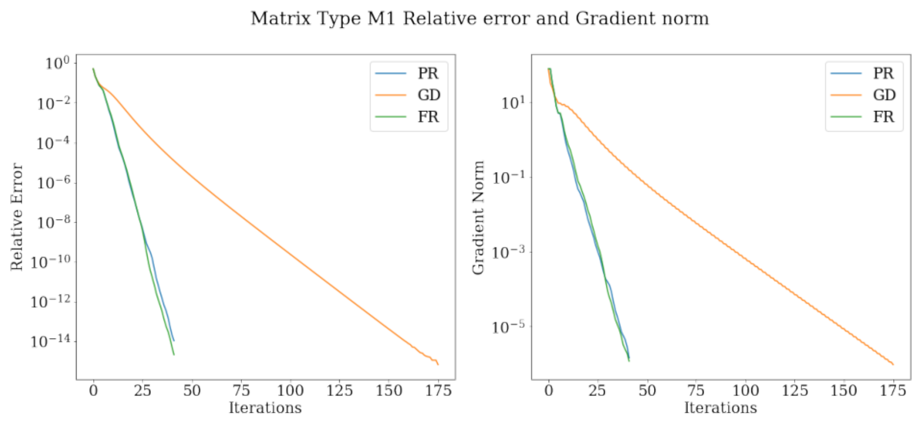


Figure 1: Comparison of the convergence rate between gradient descent and conjugate gradient

5.4 Conjugate Gradient Time Evaluation

As we have done for the gradient descent we run our algorithm and the scipy implementation 1000 times since the performance of the algorithms can change based on the initial point $\boldsymbol{x_0}$. The table below have the same structure as the one in the gradient descent evaluation subsection 5.2, then we have the meantime and the standard deviation for both the algorithm but since our conjugate implementation use two different way to compute β, using the Fletcher-Reeves method and the Polak-Ribière one, we have two error column.

Matrix	CG-FR(Time)	CG-PR(Time)	scipy.cg(Time)	FR-Error	PR-Error
M_1	13.2 ms ± 1.64 ms	13.2 ms ± 1.03 ms	17.9 ms ± 1.92 ms	$3.67e^{-13}$	8.36^{-13}
M_2	6.53 ms ± 1.11 ms	6.69 ms ± 1.59 ms	9.2 ms ± 1.2 ms	$1.37e^{-12}$	$8.17e^{-14}$
M_3	5.42 ms ± 891 µs	4.11 ms ± 333 µs	6.62 ms ± 1.22 ms	$3.32e^{-13}$	$5.10e^{-14}$
M_4	184 ms ± 15.8 ms	259 ms ± 63.5 ms	320 ms ± 31.7 ms	$5.93e^{-12}$	$6.35e^{-11}$
M_5	84 ms ± 12.3 ms	100 ms ± 8.61 ms	96.6 ms ± 5.24 ms	$3.08e^{-14}$	$1.31e^{-11}$

Table 2: Time performance and error difference between conjugate gradient algorithm with FR beta and scipy tool

As we can see from the table we can achieve similar results with the off-shell algorithm with both the β implementation and in most cases we perform even better.

6 Bibliography

[1] On the rate of superlinear convergence of a class of variable metric methods. *Numerische Mathematik*, 35:293–314, 1980.

[2] Walther Burmeister. Die konvergenzordnung des fletcher-powell-algorithmus. *Zammzeitschrift Fur Angewandte Mathematik Und Mechanik*, 53:693–699, 1973.

[3] Arthur I. Cohen. Rate of convergence of several conjugate gradient algorithms. *SIAM Journal on Numerical Analysis*, 9(2):248–259, 1972.

[4] Harlan Crowder and Philip Wolfe. Linear convergence of the conjugate gradient method. *IBM Journal of Research and Development*, 16(4):431–433, 1972.

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[6] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.