Computational mathematics for data analysis and learning

Gabriele Barreca, Mario Bonsembiante

2018/2019

Abstract

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

1 Norm as minimization problem

Given the norm definition induced by 2-norm vector definition

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

where the 2-norm is defined as

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

and $A \in \mathbb{R}^{m \times n}$. Now we can manage it using the 2-norm definition

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

but, as the square root is a monotonous function and if h is a monotonous function we can write $\sup_x h(g(x)) = h(\sup_x g(x))$ and the problem now can be written as

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

So now we can define our objective function as

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

so the norm can be written as an unconstrained minimization problem

$$||\boldsymbol{A}||_2 = \inf_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x})$$

1.1 Problem defined as constrained optimization problem

It's also interesting notice that the problem of matrix norm can be written, for all induced norm, also as

$$||\boldsymbol{A}|| = \max_{||\boldsymbol{x}||=1} ||\boldsymbol{A}\boldsymbol{x}||$$

because we have that

$$\sup_{x\in\mathbb{R}^n_{\neq 0}} \frac{||\boldsymbol{A}\boldsymbol{x}||}{||\boldsymbol{x}||} \geq \max_{||x||=1} ||\boldsymbol{A}\boldsymbol{x}||$$

but also, given $\boldsymbol{z} = \frac{\boldsymbol{x}}{||\boldsymbol{x}||}$

$$\frac{||\boldsymbol{A}\boldsymbol{x}||}{||\boldsymbol{x}||} = \left\|\boldsymbol{A}\frac{\boldsymbol{x}}{||\boldsymbol{x}||}\right\| = \|\boldsymbol{A}\boldsymbol{z}\| \leq \max_{||\boldsymbol{x}||=1} ||\boldsymbol{A}\boldsymbol{x}||$$

but this is true for each $\boldsymbol{x} \in \mathbb{R}^n_{\neq 0}$ so also for the sup, then we have

$$\sup_{\boldsymbol{x}\in\mathbb{R}^n_{\neq 0}} \frac{||\boldsymbol{A}\boldsymbol{x}||}{||\boldsymbol{x}||} \leq \max_{||\boldsymbol{x}||=1} ||\boldsymbol{A}\boldsymbol{x}||$$

the two inequalities gives the equality.
Now we can express the problem in a simpler way with constrain.

$$\min_{x}\{-\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{x} : \boldsymbol{x} \in \mathbb{R}^n \wedge \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{x} = 1\}$$

2 Function properties

First of all, in order to analyze the properties of this function we will introduce how to compute the gradient calculus.

2.1 Closed form for the gradient

In order to find the gradient of the objective function at each iteration we have written it in a closed form. The objective function can be written as

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

where a_{ij} are the elements of $\boldsymbol{Q} = \boldsymbol{A^T A}$. The complexity for this operation is $O(mn^2)$. Luckly we have to compute $\boldsymbol{Q}$ only one time which is a very good for the computational cost if we have a matrix very asymmetric with $m >> n$.
The complexity of $f(\boldsymbol{x})$ is $O(n^2)$ because $\boldsymbol{x^T Q}$ is $O(n^2)$ (of course we can compute first $\boldsymbol{Qx}$ and the result is the same) and $\boldsymbol{(x^T Q) x}$ is also $O(n^2)$ and $\boldsymbol{x^T x}$ is $O(n)$ and the division is $O(1)$.

$$C(f(\boldsymbol{x})) = 2O(n^2) + O(n) + 0(1) = O(n^2) \tag{1}$$

We can obtain the partial derivatives in all the point except 0

$$\frac{\partial f}{\partial x_k} = \frac{2x_k(\sum_{i=1}^n \sum_{j=0}^n x_i x_j a_{ij})}{(\sum_{i=1}^n x_i^2)^2} - \frac{\sum_{j=1}^n 2x_j a_{kj}}{\sum_{i=1}^n x_i^2}$$

which are continuous and derivable in all the points except 0. The function is additionally differentiable in $\mathbb{R}^n \setminus \{0\}$.
We can write in a more compact way

$$\frac{\partial f}{\partial x_k} = \frac{2x_k(\boldsymbol{x^T Q x})}{(\boldsymbol{x^T x})^2} - \frac{\sum_{j=1}^n 2x_j a_{kj}}{\boldsymbol{x^T x}}$$

so we can write the gradient as

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

The complexity for obtaining the gradient fro scratch is $O(n^2)$ as in the 1 because we have very similar operations. But if we write the gradient in a smarter way, like

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

preserve the partial results we have had for $f(\boldsymbol{x})$, like $\boldsymbol{x^T x}$ and $\boldsymbol{Qx}$, we can save many operations and we can obtain the gradient with just a $O(n)$ computation needed for $\boldsymbol{x}f(\boldsymbol{x})$.
So the complexity of the gradient after we have obtained the value of the function is:

$$C(\nabla f(\boldsymbol{x})) = O(n) \tag{3}$$

2.2 Important properties

First of all is interesting to notice that $\boldsymbol{A^T A}$ is symmetric and the matrix $\boldsymbol{Q} = \boldsymbol{A^T A}$ is positive semi-definite, because

$$\boldsymbol{x^T Q x} = \boldsymbol{x^T A^T A x} = ||\boldsymbol{Ax}||_2^2 \geq 0 \iff \boldsymbol{A^T A} \succeq 0$$

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

$$\boldsymbol{x^T Q x} = \boldsymbol{x^T A^T A x} = ||\boldsymbol{Ax}||_2^2 > 0 \iff \boldsymbol{A^T A} \succ 0$$

The function is limited below in the domain $\mathbb{R}^n_{\neq 0}$.

$$\exists M \in \mathbb{R}^+ : -f(\boldsymbol{x}) \leq M \quad \forall x \in \mathbb{R}^n_{\neq 0}$$

$$\iff ||-f(\boldsymbol{x})|| = \left\| \frac{\boldsymbol{x^T Q x}}{\boldsymbol{x^T x}} \right\| \leq \frac{||\boldsymbol{x}| \cdot ||\boldsymbol{Q}|| \cdot ||\boldsymbol{x}||}{||\boldsymbol{x}||^2} \leq ||\boldsymbol{Q}|| \leq M \tag{4}$$

The function is homogeneous of degree zero: $f(t\boldsymbol{x}) = f(\boldsymbol{x}) \quad \forall t \in \mathbb{R}^+$ so the gradient is homogeneous with degree -1: $\nabla f(t\boldsymbol{x}) = t^{-1}\nabla f(\boldsymbol{x})$
The gradient is bounded on every set of the form $\{\boldsymbol{x} : ||\boldsymbol{x}|| \geq r > 0\}$. Indeed, it is bounded on the unit sphere by some constant M (by continuity and compactness), hence bounded by M/r on the aforementioned set.
We can estimate M as

$$||\nabla f(\boldsymbol{x})|| \leq 4||\boldsymbol{Q}|| \quad ||\boldsymbol{x}|| = 1 \tag{5}$$

Then the function is lipschitz continuous for $\boldsymbol{x} : ||\boldsymbol{x}|| \geq \epsilon > 0$ since the norm of the gradient is bounded by M/ϵ out of this ball.

2.3 Stationary point

It's important to know, before we can continue that this function has many stationary points (it's possible because it's not convex). So later, when we will discuss if the algorithms will converge to a stationary point we have to know that this condition will not guarantee that this points are the global minimum. Indeed the function is not convex and the stationary point could not be global optimum.
Given a stationary point $\bar{\boldsymbol{x}} \neq 0$, such that

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

$$\iff (\bar{\boldsymbol{x}}^T \boldsymbol{Q} \bar{\boldsymbol{x}})\bar{\boldsymbol{x}} - (\bar{\boldsymbol{x}}^T \bar{\boldsymbol{x}})\boldsymbol{Q}\bar{\boldsymbol{x}} = 0$$
$$\iff (\bar{\boldsymbol{x}}^T \boldsymbol{Q} \bar{\boldsymbol{x}})\bar{\boldsymbol{x}} = (\bar{\boldsymbol{x}}^T \bar{\boldsymbol{x}})\boldsymbol{Q}\bar{\boldsymbol{x}}$$
$$f(\bar{\boldsymbol{x}})\bar{\boldsymbol{x}} = \boldsymbol{Q}\bar{\boldsymbol{x}}$$

which means that $\bar{\boldsymbol{x}}$ is a stationary point if and only if it's an eigenvector for $\boldsymbol{Q}$ and his eigenvalue is $f(\bar{\boldsymbol{x}})$.
Addiotanlly, given $\gamma_1, \ldots, \gamma_n$ $\boldsymbol{Q}$ eigenvalue (which exists and are real for the real spectral theorem) we have that, given an one of them λ_i whose eigenvector is $\boldsymbol{x}_{\boldsymbol{\lambda}_i} \neq 0$ then $f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) = \lambda_i$

$$f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) = \frac{\boldsymbol{x}_{\boldsymbol{\lambda}_i}^{\boldsymbol{T}} \boldsymbol{Q} \boldsymbol{x}_{\boldsymbol{\lambda}_i}}{\boldsymbol{x}_{\boldsymbol{\lambda}_i}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{\lambda}_i}} = \frac{\boldsymbol{x}_{\boldsymbol{\lambda}_i}^{\boldsymbol{T}} \lambda_i \boldsymbol{x}_{\boldsymbol{\lambda}_i}}{\boldsymbol{x}_{\boldsymbol{\lambda}_i}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{\lambda}_i}} = \lambda_i$$

2.4 Non convexity

The last important propriety is that the function is not convex, indeed the absolute value of the function, as we have proved before, is bounded and the only function bounded and convex is the constant function and our function is not constant if $\boldsymbol{Q}$ has at least two different eigenvalues. If exists $\lambda_i > \lambda_j : \quad i \neq j$ whose eigenvector are $\boldsymbol{x}_{\boldsymbol{\lambda}_i}, \boldsymbol{x}_{\boldsymbol{\lambda}_j}$ we have that $f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) = \lambda_i$ so $f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) > f(\boldsymbol{x}_{\boldsymbol{\lambda}_j})$. But as we have seen in 2.3 the gradient in both points is zero because they are eigenvector. But if the function was convex and differentiable then we have that

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \quad f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{y})^T (\boldsymbol{y} - \boldsymbol{x}).$$

which means that

$$f(\boldsymbol{x}_{\boldsymbol{\lambda}_i}) \leq f(\boldsymbol{x}_{\boldsymbol{\lambda}_j})$$

which contradicts the previous hypothesis.

2.5 Research on the descent direction

Before we start with the algorithms we will introduce the closed formula for the parameterization along a direction $\boldsymbol{d}$.
So, given a point $\boldsymbol{x}$ in the domain we want the closed formula for the derivative along the direction $\boldsymbol{d}$.

$$\phi(\alpha) = f(\boldsymbol{x} + \alpha\boldsymbol{d}) \quad \alpha > 0$$

Using the chain rule we can easily find the derivative of $\phi(\alpha)$ as

$$\phi'(\alpha) = \langle \nabla f(\boldsymbol{x} + \alpha\boldsymbol{d}), \nabla f(\boldsymbol{x}) \rangle$$

Whic can be written with the help of the closed formula of the gradient 2

$$\phi'(\alpha) = \left[\frac{2(\boldsymbol{x} + \alpha\boldsymbol{d})((\boldsymbol{x} + \alpha\boldsymbol{d})^T \boldsymbol{Q}(\boldsymbol{x} + \alpha\boldsymbol{d}))}{((\boldsymbol{x} + \alpha\boldsymbol{d})^T(\boldsymbol{x} + \alpha\boldsymbol{d}))^2} - \frac{2\boldsymbol{Q}(\boldsymbol{x} + \alpha\boldsymbol{d})}{(\boldsymbol{x} + \alpha\boldsymbol{d})^T(\boldsymbol{x} + \alpha\boldsymbol{d})^2} \right]^T \cdot \nabla f(\boldsymbol{x})$$

Which is, given $\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{Q}$, a polynomial of α with degree of 2.
We used that, given $Q \in \mathbb{R}^{n \times n} : Q^T = Q \ x, y \in \mathbb{R}^n \implies (x^TQy) = (x^TQy)^T = y^TQ^Tx = y^TQx$ because it's a scalar with linearity to simplify, and we got

$$2\frac{\alpha^2((\boldsymbol{x_0^T Q d})(\boldsymbol{d^T d}) - (\boldsymbol{d^T Q d})(\boldsymbol{x_0^T d})) + \alpha((\boldsymbol{d^T d})(\boldsymbol{x_0^T Q x_0}) - (\boldsymbol{x_0^T x_0})(\boldsymbol{d^T Q d}))+}{((\boldsymbol{x_0} + \boldsymbol{\alpha d})^{\boldsymbol{T}}(\boldsymbol{x_0} + \boldsymbol{\alpha d}))^2} \\ \frac{+((\boldsymbol{x_0^T d})(\boldsymbol{x_0^T Q x_0}) - (\boldsymbol{x_0^T x_0})(\boldsymbol{x_0^T Q d}))}{((\boldsymbol{x_0} + \boldsymbol{\alpha d})^{\boldsymbol{T}}(\boldsymbol{x_0} + \boldsymbol{\alpha d}))^2} \tag{6}$$

We can write the problem in a more compact notation, give a general direction $\boldsymbol{d}$ we have

$$\begin{aligned}
&\phi'(\alpha) = \frac{a\alpha^2 + b\alpha + c}{P(\alpha)} \\
&a = \boldsymbol{(x_k^T Q d)(d^T d) - (d^T Q d)(x_k^T d)} \\
&b = \boldsymbol{(d^T d)(x_k^T Q x_k) - (x_k^T x_k)(d^T Q d)} \\
&c = \boldsymbol{(x_k^T d)(x_k^T Q x_k) - (x_k^T x_k)(x_k^T Q d)} \\
&P(\alpha) = \boldsymbol{((x_k + \alpha d)^T (x_k + \alpha d))}^2
\end{aligned} \tag{7}$$

As we have done in for the gradient and the function we have to estimate the complexity for obtaining this parametrization. In general we have that the complexity for having each terms a,b,c is $O(n^2)$ because we have some products for $\boldsymbol{Q}$ which is $O(n^2)$. We can save some operations also here for example for $\boldsymbol{x^T Q x}$ but we can't do better than $O(n^2)$ because some operations like $\boldsymbol{d^T Q d}$ were never been made and they take $O(n^2)$.

$$C(\phi'(\alpha)) = O(n^2) \tag{8}$$

3 Steepest Descent Direction

In this section we present the first algorithm that we will use for solving the minimization problem we have presented in the 1 chapter, the Steepest Descent Direction. This algorithm is one of the descent algorithm. The algorithm search along the steepest direction, which is, if we use l_2 norm, $-\nabla f(\boldsymbol{x})$. For this reason the algorithm could be called also "Gradient Descent".

Algorithm 1 Steepest Descent Direction

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

3.1 Exact search along descent direction

Now we will face the problem of search along the direction $\boldsymbol{d_k} = -\nabla f(\boldsymbol{x_k})$ for the point which minimize the function, which is the line 5 of the algorithm 1.
Using the result in 7 we have the closed form along the direction. But, if we want to find a stationary point along this direction we have to be sure that the denominator of $\phi'(\alpha)$ is not equal to zero.

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

$$\iff \boldsymbol{x} = \alpha \nabla f(\boldsymbol{x})$$

But there is a solution in α if and only if $\boldsymbol{x}$ and $\nabla f(\boldsymbol{x})$ are linearly dependent. But we can easily notice that these two vectors are orthogonal, indeed

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

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

hence $\phi'(\alpha)$ is well defined for all the value of alpha greater than zero. Then, since $\phi'(0) < 0$ (because $-\nabla f(\boldsymbol{x})$ is a descend direction) and $\phi'(\alpha)$ is continuous, the smallest root of $a\alpha^2 + b\alpha + c$ greater than zero is a stationary point for $\phi(\alpha)$ and $\phi(\bar{\alpha}) < \phi(0)$.
We can say now that the $\bar{\alpha}$ such that $\phi'(\bar{\alpha}) = 0$ will satisfy the Wolf condition, which is important for the convergence of the algorithm

$$\bar{\alpha} : \quad \phi'(\alpha) = 0 : \quad (W) \tag{9}$$

3.2 Convergence of the algorithm

As we have explained in the 2.2 the function is lipschitz continuous out of a ball $B(0,\epsilon) \quad \forall \epsilon > 0$. Fortunately we can search our solution on a domain $\mathcal{D} = \mathbb{R}^n \backslash \mathcal{B}(0,\epsilon) \quad \forall \epsilon > 0$ where the function is lipschitz continuous. Indeed we can chose $\epsilon < ||\boldsymbol{x_0}||_2$ and $||\boldsymbol{x_k}||_2 \in \mathcal{D} \quad \forall k \geq 0$ because $||\boldsymbol{x_k}||_2 > ||\boldsymbol{x_{k-1}}||_2$. This is a consequence of the orthogonality of the gradient and the point, proved before. For the orthogonality we can apply the generalized Pitagora theorem in $\mathbb{R}^n$ [1]

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

We have proved that, given $||x_0||_2 \geq \epsilon > 0$ the succession of point belongs to the $\mathcal{D} = \mathbb{R}^n \backslash \mathcal{B}(0,\epsilon)$, where the function is lipsichz continuous and differntiable2. Indeed with these hypothesis combined with the Wolf condition showed in 9 and the bounded condition 4 we can apply the Zoutendijk's therorem:

Theorem 1 *Consider an algorithm which search along a descending direction* $\boldsymbol{d_k}$ *a point which satisfy the Wolf condition* ($\phi'(a_k) \leq m_2\phi'(a_k) : \quad m_2 < 1$)*.* 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 lipsichz continuous on* $\mathcal{N}$*, that is, exists an* L :

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

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

Then

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

The full proof can be found on [1, p. 38-40].

In this case the $\cos\theta_k$ are always equal to -1 because $\boldsymbol{d_k^T} = -\nabla f(\boldsymbol{x_k})$ so we have that $\sum_{k=0}^{+\infty} ||\nabla f(\boldsymbol{x_k})||_2^2 = k < \infty$ which imply that $\lim_{k\to\infty} ||\nabla f(\boldsymbol{x_k})||_2^2 = 0$ and the algorithm will converge to a stationary point.

3.3 Algorithm complexity

In this paragraph we will discuss the complexity of each iteration of the algorithm. We have proved in 1, 3, 8 the complexity for having $f(\boldsymbol{x})$, $\nabla f(\boldsymbol{x})$ and $\phi'(\alpha)$. The last passage that is not just a sum or allocating is the minimization along $\phi(\alpha)$. But as we have seen in 3.1 this can be obtained with a simple resolution of a two degree polynomial zeros which is obtained by a closed formula in $\mathbb{R}$ so it's $O(1)$. So we can say that the complexity of one iteration of the Steepest Descent Direction algorithm is $O(n^2)$

$$C(SDD) = O(n^2) \tag{10}$$

4 Conjugate Gradient

This optimization algorithm search the optimum along the conjugate gradient direction. In particular we will use and discuss the Fletcher–Reeves method. First of all we introduce the algorithm with his proprieties.

Algorithm 2 Conjugate Gradient

```
1: procedure CG($\boldsymbol{x_0} \in \mathbb{R}^N_{\neq 0}$)
2:     while $||\nabla f(\boldsymbol{x_k})|| > \epsilon$ do
3:         $\beta_k = \frac{\nabla f(\boldsymbol{x_k})^T \nabla f(\boldsymbol{x_k})}{\nabla f(\boldsymbol{x_{k-1}})^T \nabla f(\boldsymbol{x_{k-1}})}, \quad k \geq 1, \quad \beta_0 = 0$
4:         $\boldsymbol{p_k} = -\nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{p_{k-1}}, \quad \boldsymbol{p_0} = -\nabla f(\boldsymbol{x_0})$
5:         $\phi(\alpha) = f(\boldsymbol{x_k} + \alpha \boldsymbol{p_k})$
6:         $\alpha_k = \arg\min_\alpha \phi(\alpha) : \alpha > 0$
7:         $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \alpha_k \boldsymbol{p_k}$
8:     return $\boldsymbol{x_k}$
```

4.1 Exact search along the direction

First of all is important to notice that the direction $\boldsymbol{p_k}$ is a descent direction if the search along the direction is exact. The exact search means that

$$0 = \phi'(\alpha_k) = \nabla f(\underbrace{\boldsymbol{x_k} + \alpha_k \boldsymbol{p_k}}_{\boldsymbol{x_{k+1}}})^T \boldsymbol{p_k} = \nabla f(\boldsymbol{x_{k+1}})^T \boldsymbol{p_k}$$

then we can easily prove that $\nabla f(\boldsymbol{x_k})^T \boldsymbol{p_k} \leq 0$, indeed

$$\nabla f(\boldsymbol{x_k})^T \boldsymbol{p_k} = -||\nabla f(\boldsymbol{x_k})||_2^2 + \underbrace{\beta_k \nabla f(\boldsymbol{x_k})^T \boldsymbol{p_{k-1}}}_{\text{0 with exact search}} \leq 0 \tag{11}$$

We have found in the section 2.5 that we can found the value of alpha along a descent direction $\boldsymbol{d}$ which minimize the function $\phi(\alpha) = f(\boldsymbol{x} + \alpha \boldsymbol{d})$, and the closed formula for α is given by the smallest positive root of the equations 6 and 7

$$\begin{aligned}
&\phi'(\alpha) = \frac{a\alpha^2 + b\alpha + c}{P(\alpha)}\\
&a = \boldsymbol{(x_k^T Q d)(d^T d) - (d^T Q d)(x_k^T d)}\\
&b = \boldsymbol{(d^T d)(x_k^T Q x_k) - (x_k^T x_k)(d^T Q d)}\\
&c = \boldsymbol{(x_k^T d)(x_k^T Q x_k) - (x_k^T x_k)(x_k^T Q d)}\\
&P(\alpha) = \boldsymbol{((x_k + \alpha d)^T (x_k + \alpha d))}^2\\
&\boldsymbol{d} = \boldsymbol{p_k} = -\nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{p_{k-1}}
\end{aligned}$$

If we prove that also in this case doesn't exist a value of α such that the denominator is equal to zero we can use the same proof we did before, and we can say that the smallest positive root is a stationary point along this direction.
As we did before $P(\alpha)$ can be equal to zero if and only if $\boldsymbol{x_k}$ and $\boldsymbol{p_k}$ are linearly dependent. If they are linearly dependent then $| < \boldsymbol{x_k}, \boldsymbol{p_k} > | = ||\boldsymbol{x_k}||_2 \cdot ||\boldsymbol{p_k}||_2$ otherwise $| < \boldsymbol{x_k}, \boldsymbol{p_k} > | < ||\boldsymbol{x_k}||_2 \cdot ||\boldsymbol{p_k}||_2$. We can obtain the second from

$$||\boldsymbol{x_k^T p_k}||_2 = ||\boldsymbol{x_k}^T \cdot (-\nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{p_{k-1}})||_2 = ||0 + \beta_k \boldsymbol{x_k}^T \boldsymbol{p_{k-1}}||_2 = ||\beta_k \boldsymbol{x_k}^T \boldsymbol{p_{k-1}}||_2 =$$

$$= ||\beta_k \boldsymbol{x_k}^T \boldsymbol{p_{k-1}}||_2 \leq ||\boldsymbol{x_k}||_2 \cdot ||\beta_k \boldsymbol{p_{k-1}}||_2 < ||\boldsymbol{x_k}||_2 \cdot ||\boldsymbol{p_k}||_2$$
$$\iff ||\beta_k \boldsymbol{p_{k-1}}||_2 < ||\boldsymbol{p_k}||_2 = || - \nabla f(\boldsymbol{x_k}) + \beta_k \boldsymbol{p_{k-1}}||_2$$

but, as we have seen before, $\nabla f(\boldsymbol{x_k})$ and $\boldsymbol{p_{k-1}}$ are orthogonal, so, for the generalized Pitagora theorem, we have that

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

since $||\nabla f(\boldsymbol{x_k})||_2 > 0$. Is also interesting that this result doesn't depend on the choice of β_k so we can choose also other β_k as the Polak–Ribière, defined as

$$\beta_k = \frac{\nabla f(\boldsymbol{x_k})^T (\nabla f(\boldsymbol{x_k}) - \nabla f(\boldsymbol{x_{k-1}}))}{\nabla f(\boldsymbol{x_{k-1}})^T \nabla f(\boldsymbol{x_{k-1}})}$$

We have demonstrate that doesn't exists value of α that zero the denominator, so we can find the root of the function $\phi'(\alpha)$ without constrain in α and so we can find the point $\bar{\alpha} : \quad \phi(\bar{\alpha}) = 0$ which of course satisfy the wolf condition as in the steepest descent direction.

4.2 Convergence of the algorithm

As for the steepest gradient descent algorithm we want the lipsichz continuity so we can apply the Zoutendijk's therorem. So, given $||\boldsymbol{x_0}||_2$ we want to prove that, given $\mathcal{D} = \mathbb{R}^n \backslash \mathcal{B}(0, \epsilon)$

$$\exists \epsilon > 0 : \quad \boldsymbol{x_k} \in \mathcal{D} = \mathbb{R}^n \backslash \mathcal{B}(0, \epsilon) \quad \forall k \geq 0 \iff$$

$$\exists \epsilon > 0 : \quad ||\boldsymbol{x_k}||_2 \geq \epsilon \quad \forall k \geq 0$$

but $||\boldsymbol{x_{k+1}}||_2^2$ can be written as

$$||\boldsymbol{x_{k+1}}||_2^2 = ||\boldsymbol{x_k} + \bar{\alpha}\boldsymbol{p_k}||_2^2 \geq ||\boldsymbol{x_k}||_2^2 \quad \forall k \geq 0$$

$$(\boldsymbol{x_k} + \bar{\alpha}\boldsymbol{p_k})^T(\boldsymbol{x_k} + \bar{\alpha}\boldsymbol{p_k}) = ||\boldsymbol{x_k}||_2^2 + \bar{\alpha}^2||\boldsymbol{p_k}||_2^2 + 2\bar{\alpha}\boldsymbol{x_k^T p_k} > ||\boldsymbol{x_k}||_2^2$$

Which is true if $\boldsymbol{x_k^T p_k} > 0$ because $\bar{\alpha} > 0$. But, as we have demonstrate before, and using that $\beta_k > 0$

$$\boldsymbol{x_k^T p_k} = \underbrace{-\boldsymbol{x_k}\nabla f(\boldsymbol{x_k})}_{=0} + \beta_k \boldsymbol{x_k^T p_{k-1}} = \beta_k(\underbrace{\boldsymbol{x_{k-1}} + \bar{\alpha}_{k-1}\boldsymbol{p_{k-1}}}_{=\boldsymbol{x_k}})^T p_{k-1} =$$

$$= \beta_k \boldsymbol{x_{k-1}^T p_{k-1}} + \underbrace{\beta_k \bar{\alpha}_{k-1}||\boldsymbol{p_{k-1}}||_2^2}_{\gamma_{k-1}>0}$$

We can use the same passages for each $k > 0$ and find all the γ_i as

$$\gamma_i = \left(\prod_{j=k+1-i}^{k+1} \beta_j\right)\bar{\alpha}_i||\boldsymbol{p_i}||_{\boldsymbol{2}}^{\boldsymbol{2}} > 0 \quad \forall i$$

For $k = 0$ instead we use that

$$\boldsymbol{x_0^T p_0} = \gamma_0 = -\boldsymbol{x_0^T}\nabla f(\boldsymbol{x_0}) = 0$$

$$\boldsymbol{x_k^T p_k} = \sum_{i=0}^{k-1} \gamma_i > 0$$

We have proved that, given $||x_0||_2 \geq \epsilon > 0$ the succession of point belongs to the $\mathcal{D} = \mathbb{R}^n \backslash \mathcal{B}(0, \epsilon)$, where the function is lipsichz continuous.
The lipsichz continuity combined with the wolf condition, the limitation below and the differentiability guaranteed the convergence of the algorithm. As we have seen for the gradient method the proof involves the Zoutendijk's theorem but we don't show the full proof here. It is shown in [1, p. 125-130].

Theorem 2 *Suppose that in the domain $\mathcal{D}$ the function is bounded, differentiable and lipsichz continuous. The line search that satisfies the Wolf condition, with $0 < c_1 < c_2 < \frac{1}{2}$, then*

$$\liminf_{k\to\infty} ||\nabla f(\boldsymbol{x}|| = 0$$

4.3 Algorithm complexity

The complexity of each iteration of this algorithm is almost the same as the SDD.
We have proved in 1, 3, 8 the complexity for having $f(\boldsymbol{x})$, $\nabla f(\boldsymbol{x})$ and $\phi'(\alpha)$. The minimization along $\phi(\alpha)$ is as for the SDD $O(1)$. For the conjugate gradient we have also the computation of β_k and $\boldsymbol{p_k}$: the first can be obtained in $O(n)$ but if we save the norm of the gradient which we will use for the stop criteria we will have that this can be done just with a division.
For the $\boldsymbol{p_k}$ we have to do a sum and a multiplication of $\mathbb{R}^n$ vectors which is $O(n)$ and we can't save operations with previus result.
So we can say now that the complexity of one iteration of the Conjugate Gradient algorithm is $O(n^2)$

$$C(CG) = O(n^2) \tag{12}$$

5 Performance evaluation

Now we are going to evaluate the performance of this method for norm calculus. We are going to evaluate the performance of the algorithm compared to the most famous norm calculator by two important parameters: time and accuracy.
In order to perform an extensive analysis we created many matrix belonging to many different categories. Considering that, with x we mean a generic element of the matrix, with d the density of it and with $range$ a random number bigger than 1, we have the following kind of matrices:

- Type $A^{1000\times 50}$ with $x \in [-range, range]$, d = 1
- Type $B^{1000\times 50}$ with $x \in [-range, range]$, d = 0.5
- Type $C^{1000\times 5}$ with $x \in [-range, range]$, d = 0.25
- Type $D^{1000\times 5}$ with $x \in [-range, range]$, d = 0.01
- Type $E^{1000\times 1000}$ with $0 < x < 1$, d = 1, Ill conditioned with a condition number between 1×10^{-18} and 1×10^{-19}

The evaluation, also, depends on some algorithm parameters that are:

- **MaxFeval** = the max number of the iteration that the algorithm will do before stopping itself.
- **eps** = the accuracy in the stopping criterion: the algorithm is stopped when the norm of the gradient is less than or equal to eps.
- **mina** = the minimum step size.
- **X** = the starting vector.

All experiments have been made with: MaxFevall = 500, mina = 1×10^{-16}, eps = 1×10^{-6} and the vector X, chosen randomly, that is the same for each matrix in both algorithms. It means that there are, in our case, ten matrices per type and for each matrix the two algorithms compute the problem with the same initial vector X. The figures below, represent the trend of the Relative Error and Norms of Gradient, in natural logarithmic scale, for the Steepest Descent Gradient in blue and for the Conjugate Gradient in orange.

The lines stand for the geometric mean of the trend of a type of matrices while ,the clearest part symbolizes the interval of confidence. At least, before proceeding with plotting, the final values have been modified to improve their representation:

- values like minus infinite are approximated with 1×10^{-20} to avoid fluctuations around a value;
- the vectors to represent are all made of the same size, adding their optimum value if necessary, to avoid that the value of the average increases.

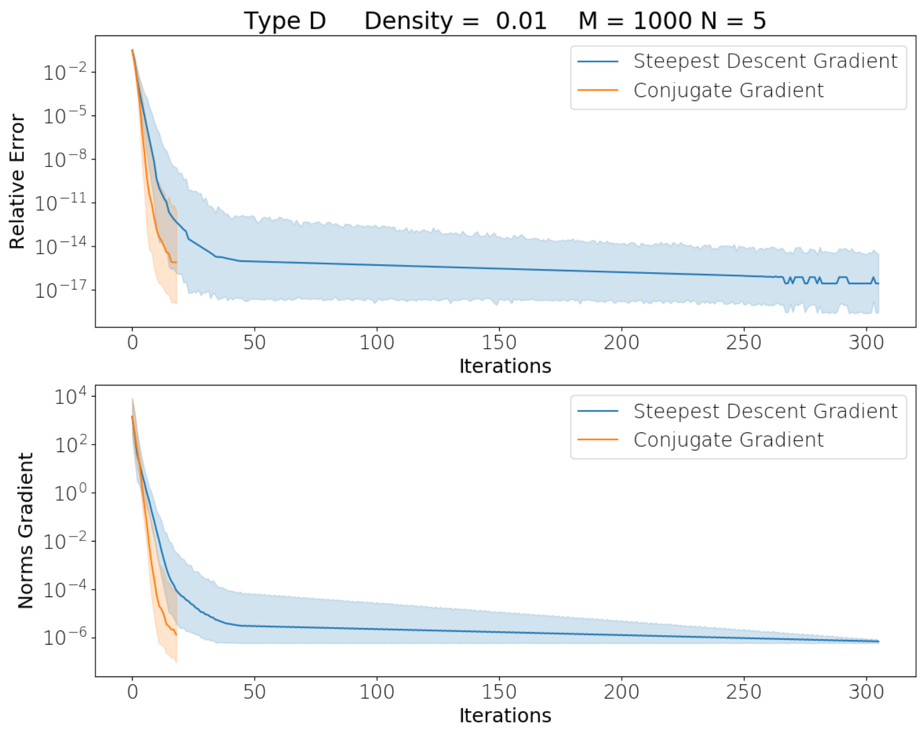


Figure 1: Plot of Relative Error and Norms of Gradient of matrix of Type D

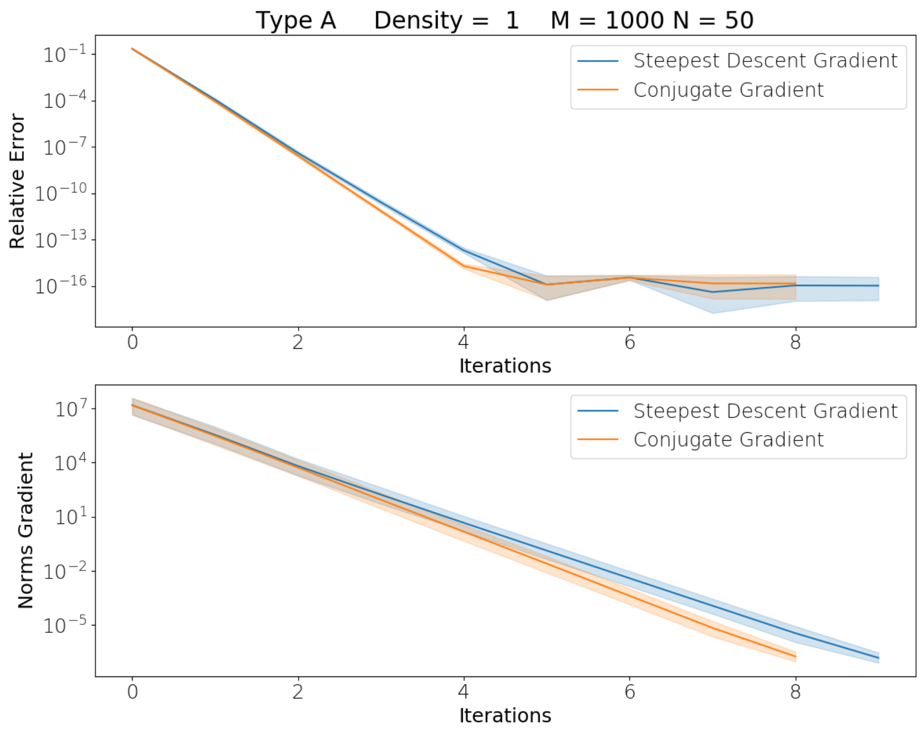


Figure 2: Plot of Relative Error and Norms of Gradient of matrix of Type C

Looking at the figures 1 and 2, after the trivial observation that Conjugate Method converges with less iterations, we can deduce something interesting like that the Norms of Gradient could be stopped first with a norm of 1×10^{-4} because the function keep going down but it does not improve.

5.1 Time performance

We analyzed the average time performance for each type of matrix of the two algorithms compared to the numpy norm calculator. In order to avoid the variance induced by the starting point we performed many attempts for each matrix (100) and we will report the average value and the deviation. We did the same also with the library calculators because we don't know their implementation and they could be afflicted by variance too.

The table 2 provides the comparison of the gradient method and the linalog function provided by numpy. The table 3 shows the results of the conjugate gradient.
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

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

The last thing to discuss before showing this results is the stopping criteria. We have chosen the gradient norm as stopping criteria and we have to chose the bound. With the results showed in section 5 we can see that if the norm of the gradient is lower than 10^{-4} we have very good results with an average relative error of 10^{-15} and also lower form many matrixes. Additionally we have chosen to use the absolute norm of the gradient and not the relative one because we have seen that with the random starting point $\boldsymbol{x_0}$ we have very different starting norms and so the algorithm is less stable and sometimes is very accurate but many times is not. Of course greater is the norm before the algorithm has the solution and the time is lower so we have to do a trade-off between time and accuracy. We have also seen that with the norm bounded by 10^{-3} we have still very good results in error but we prefer the 10^{-4} for stability reason. We will report on the appendix also the tables with the stopping criteria $||\nabla f(\boldsymbol{x_k})|| \leq 10^{-3}$.
We can see from the table 1 that for all the matrices of type A we have good accuracy and time performance for both algorithms. With matrices of type B, C, D (which are less dense) we have worst time performance compared to the numpy tool but good accuracy. With the type E matrices, the ill conditioned, we have very good time performance and also good accuracy. In particular the Conjugate Gradient over-performs the others, as we can see also with image 3.

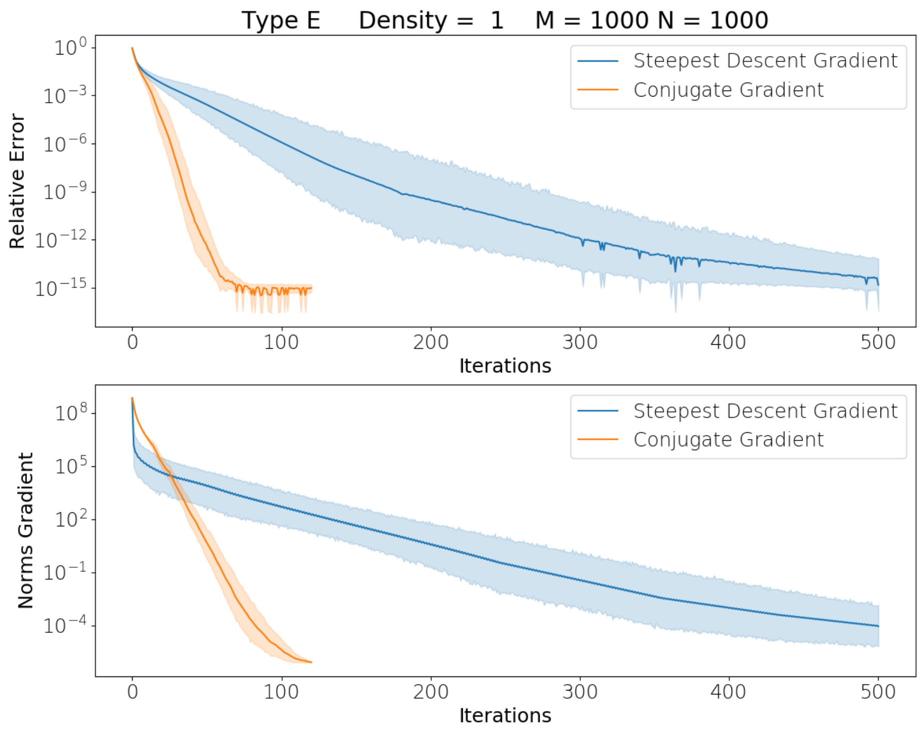

Type E Density = 1 M = 1000 N = 1000
Relative Error
10^{0}
10^{-3}
10^{-6}
10^{-9}
10^{-12}
10^{-15}
Steepest Descent Gradient
Conjugate Gradient
0
100
200
300
400
500
Iterations
Norms Gradient
10^{8}
10^{5}
10^{2}
10^{-1}
10^{-4}
Steepest Descent Gradient
Conjugate Gradient
0
100
200
300
400
500
Iterations

Figure 3

COMPARISON BETWEEN ALGORITHMS $\|\|\nabla f(\boldsymbol{x_k})\|\|_2 \leq 1 \times 10^{-4}$					
Type	np.norm	SDD		CG	
	mean time	*mean time and mean error*		*mean time and mean error*	
A	1.01×10^{-3}	1.26×10^{-3}	10×10^{-15}	1.17×10^{-3}	10×10^{-15}
B	1.72×10^{-3}	4.26×10^{-2}	10×10^{-15}	9.44×10^{-3}	10×10^{-15}
C	8.10×10^{-5}	3.97×10^{-3}	10×10^{-16}	1.19×10^{-3}	10×10^{-14}
D	7.76×10^{-5}	2.60×10^{-3}	10×10^{-13}	1.07×10^{-3}	10×10^{-13}
E	2.71×10^{-1}	2.40×10^{-1}	10×10^{-15}	7.31×10^{-2}	10×10^{-15}

Table 1: Time performance and accuracy for both algorithm compared with the numpy tool. Stopping criteria for both algorithms is $\|\|\nabla f(\boldsymbol{x_k})\|\|_2 \leq 1 \times 10^{-4}$.

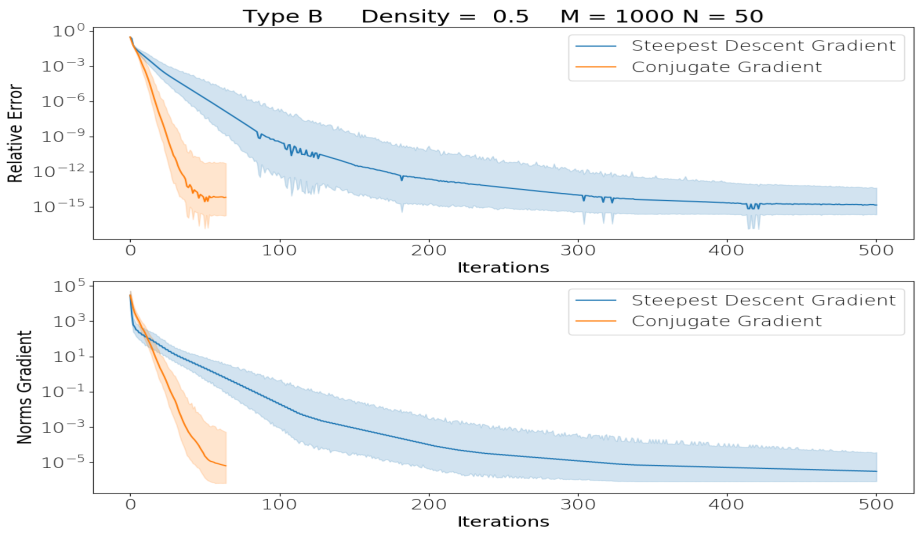
STEEPEST DESCENT DIRECTION $||\nabla f(\boldsymbol{x_k})||_2 \leq 1 \times 10^{-4}$					
Type	*np.norm, time mean and std*		*SDD, time mean and std*		*mean error*
A	1.01×10^{-3}	1.61×10^{-4}	1.26×10^{-3}	1.47×10^{-4}	10×10^{-15}
B	1.72×10^{-3}	4.66×10^{-4}	4.26×10^{-2}	2.33×10^{-3}	10×10^{-15}
C	8.10×10^{-5}	3.11×10^{-6}	3.97×10^{-3}	3.65×10^{-4}	10×10^{-16}
D	7.76×10^{-5}	1.98×10^{-6}	2.60×10^{-3}	2.73×10^{-4}	10×10^{-13}
E	2.71×10^{-1}	3.20×10^{-2}	2.40×10^{-1}	3.10×10^{-2}	10×10^{-15}

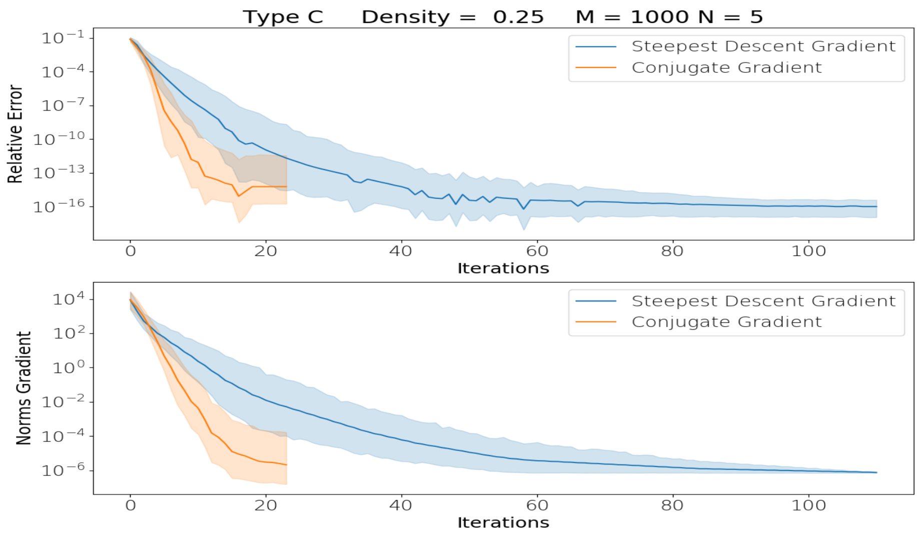
Table 2: Time and errors for the steepest descent direction(SDD) approach compared with the tool provided by numpy. Stopping criteria is $||\nabla f(\boldsymbol{x_k})||_2 \leq 1 \times 10^{-4}$.

CONJUGATE GRADIENT $||\nabla f(\boldsymbol{x_k})||_2 \leq 1 \times 10^{-4}$					
Type	*np.norm, time mean and std*		*CG, time mean and std*		*mean error*
A	1.01×10^{-3}	1.61×10^{-4}	1.17×10^{-3}	6.14×10^{-5}	10×10^{-15}
B	1.72×10^{-3}	4.66×10^{-4}	9.44×10^{-3}	8.03×10^{-4}	10×10^{-15}
C	8.10×10^{-5}	3.11×10^{-6}	1.19×10^{-3}	3.92×10^{-4}	10×10^{-14}
D	7.76×10^{-5}	1.98×10^{-6}	1.07×10^{-3}	1.80×10^{-4}	10×10^{-13}
E	2.71×10^{-1}	3.20×10^{-2}	7.31×10^{-2}	1.16×10^{-2}	10×10^{-15}

Table 3: Time and errors for the conjugate gradient(CG) approach compared with the tool provided by numpy. Stopping criteria is $||\nabla f(\boldsymbol{x_k})||_2 \leq 1 \times 10^{-4}$.

A More plots and tables





COMPARISON BETWEEN ALGORITHMS $\|\|\nabla f(\boldsymbol{x_k})\|\|_2 \leq 1 \times 10^{-3}$					
Type	np.norm	SDD		CG	
	mean time	mean time and mean error		mean time and mean error	
A	1.01×10^{-3}	1.10×10^{-3}	10×10^{-15}	1.13×10^{-3}	10×10^{-15}
B	2.59×10^{-3}	2.95×10^{-2}	10×10^{-14}	6.72×10^{-3}	10×10^{-14}
C	9.63×10^{-5}	3.74×10^{-3}	10×10^{-14}	1.17×10^{-3}	10×10^{-12}
D	7.62×10^{-5}	2.14×10^{-3}	10×10^{-11}	8.79×10^{-4}	10×10^{-11}
E	2.71×10^{-1}	2.84×10^{-1}	10×10^{-15}	8.33×10^{-2}	10×10^{-15}

Table 4: Time performance and accuracy for both algorithm compared with the numpy tool. Stopping criteria for both algorithms is $||\nabla f(\boldsymbol{x_k})||_2 \leq 1 \times 10^{-3}$.

STEEPEST DESCENT DIRECTION $\|\|\nabla f(\boldsymbol{x_k})\|\|_2 \leq 1 \times 10^{-3}$					
Type	*np.norm, time mean and std*		*SDD, time mean and std*		*mean error*
A	1.01×10^{-3}	1.90×10^{-4}	1.10×10^{-3}	2.47×10^{-5}	10×10^{-15}
B	2.59×10^{-3}	5.91×10^{-3}	2.95×10^{-2}	4.87×10^{-3}	10×10^{-14}
C	9.63×10^{-5}	1.91×10^{-5}	3.74×10^{-3}	6.29×10^{-4}	10×10^{-14}
D	7.62×10^{-5}	1.04×10^{-5}	2.14×10^{-3}	2.63×10^{-4}	10×10^{-11}
E	2.71×10^{-1}	3.20×10^{-2}	2.84×10^{-1}	7.95×10^{-2}	10×10^{-15}

Table 5: Time and errors for the steepest descent direction(SDD) approach compared with the tool provided by numpy. Stopping criteria is $||\nabla f(\boldsymbol{x_k})||_2 \leq 1 \times 10^{-3}$.

CONJUGATE GRADIENT $\|\|\nabla f(\boldsymbol{x_k})\|\|_2 \leq 1 \times 10^{-3}$					
Type	*np.norm, time mean and std*		*CG, time mean and std*		*mean error*
A	1.01×10^{-3}	1.90×10^{-4}	1.13×10^{-3}	7.09×10^{-5}	10×10^{-15}
B	2.59×10^{-3}	5.91×10^{-3}	6.72×10^{-3}	4.94×10^{-4}	10×10^{-14}
C	9.63×10^{-5}	1.91×10^{-5}	1.17×10^{-3}	4.09×10^{-4}	10×10^{-12}
D	7.62×10^{-5}	1.04×10^{-5}	8.79×10^{-4}	1.99×10^{-4}	10×10^{-11}
E	2.71×10^{-1}	3.20×10^{-2}	8.33×10^{-2}	2.22×10^{-2}	10×10^{-15}

Table 6: Time and errors for the conjugate gradient(CG) approach compared with the tool provided by numpy. Stopping criteria is $||\nabla f(\boldsymbol{x_k})||_2 \leq 1 \times 10^{-3}$.

References

[1] S. Wright J. Nocedal. *Numerical Optimization*. Spinger, 1999.