Optimization

Benoit KRIKKE
Victor PACOTTE
Riad SANCHEZ

University of NICE SOPHIA ANTIPOLIS
Departement of Applied Mathematics and Modeling

Implementation of optimization algorithms

Supervisor

P.DREYFUSS

June 15, 2012

Contents

Introduction

Optimization refers to a broad set of methods whose aim is to find the best solution to a problem called optimal. In this case, we are interested in describing the algorithms of numerical methods for solving optimization of real, continuous, differentiable and nonlinear functions.

Various approaches are possible and we'll distinguished the methods leading to a local optimum local and global methods to identify the global optimum. In this report, we look for minima, maximization problems can always be reduced equivalently to an minimization problems.

The algorithms presented are locals tools research. First we study the unconstrained optimization algorithms like the golden section, parabolic interpolation and gradient methods. Then we discuss algorithms for constrained optimization. It will be mainly the Uzawa method.

1 Unconstrained optimization

1.1 One dimension

In this part we are presenting unconstrained optimization methods for a one dimension problem. The aim will be finding the maximum of the function :

$$M(\lambda) = \frac{2\pi.h.C_0^2}{n^2\lambda^5}.\frac{1}{exp\{\frac{hC_0}{nkT\lambda}\}-1}$$

which represents the black body radiation.

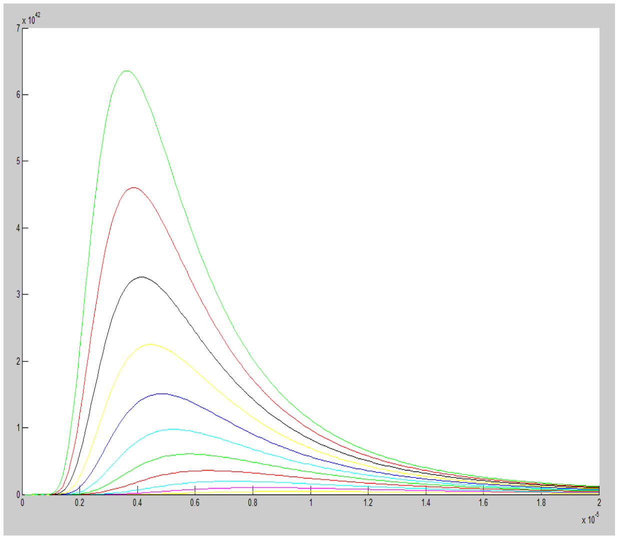


Figure 1: *M(λ) for different values of T.*

1.1.1 Golden section search

The golden section search is a technics for finding the extremum (minimum or maximum) of a unimodal function by successively narrowing the range of values inside which the extremum is known to exist. In the beginning we have an interval $[a, b]$. The aim of this algorithm is to built a decreasing sequence of intervals $[a_i, b_i]$.

Algorithm 1 Golden section search

```
Require: $n_{max} \in \mathbb{N}$, [a,b], $\epsilon > 0$
1: set $\tau = \frac{1+\sqrt{5}}{2}$ the golden ratio
2: n=0
3: while $n < n_{max}$ and $b_i - a_i > \epsilon$ do
4:     a'=$a_i + \frac{1}{\tau^2}(b_i - a_i)$
5:     b'=$b_i + \frac{1}{\tau}(b_i - a_i)$
6:     if M(a') < M(b') then
7:         $a_{i+1} = a_i$
8:         $b_{i+1} = b'$
9:     else if M(a') > M(b') then
10:        $a_{i+1} = a'$
11:        $b_{i+1} = b_i$
12:    else if M(a') = M(b') then
13:        $a_{i+1} = a'$
14:        $b_{i+1} = b'$
15:    end if
16:    n=n+1
17: end while
18: Extremum=$\frac{b_i+a_i}{2}$
19: return Extremum
```

We initialize the method with $a_0 = a$ and $b_0 = b$.
With each iteration, the interval $[a_i, b_i]$ become smaller until $n = n_{max}$ or the diference between b_i and a_i is lower than an $\epsilon > 0$ the tolerance of the method. Then the extremum is between the last b_i and a_i. We choose Extremum=$\frac{b_i+a_i}{2}$.

This method gives us the maximum of M(λ) is for $\lambda = 3.6222.10^{-6}$ in 72 iterations and the elapsed time is 0.004572 seconds.

1.1.2 Parabolic interpolation

The principal idea of the parabolic interpolation is to replace the function we want to minimize with its 2 degree interpolation polynomial in three points x_i, y_i and z_i in the interval [a,b]. The algorithm is

Algorithm 2 Parabolic interpolation

```
Require: $n_{max} \in \mathbb{N}$, [a,b], $\epsilon > 0$
1: choose $x_0$, $y_0$ and $z_0$ such as $M(x_0) \geq M(y_0)$ and $M(z_0) \geq M(y_0)$ and $x_0 < y_0 < z_0$
2: for $n = 1, ..., n_{max}$ do
3:     $M[x_i,y_i]=\frac{M(y_i)-M(x_i)}{y_i-x_i}$
4:     $M[x_i,y_i,z_i]=\frac{M[z_i,y_i]-M[x_i,y_i]}{z_i-x_i}$
5:     $y_{i+1}=\frac{x_i+y_i}{2}$ - $\frac{M[x_i,y_i]}{2M[x_i,y_i,z_i]}$
6:     if $y_{i+1} \in [x_i, y_i]$ then
7:         $x_{i+1} = x_i$
8:         $z_{i+1} = y_i$
9:     else if $y_{i+1} \in [y_i, z_i]$ then
10:        $x_{i+1} = y_i$
11:        $z_{i+1} = z_i$
12:    end if
13: end for
```

This method gives us the maximum of M(λ) is for $\lambda = 3.6222.10^{-6}$ in 200 iterations and the elapsed time is 0.083763 seconds.

1.2 Gradient method

In this part, we are going to use four different gradient methods to find the minimum of J_n.

1.2.1 Definition of the functional J_n

We define the functional J_n by :

$$J_n(\mathbf{x}) = \frac{1}{2} < A_n\mathbf{x}, \mathbf{x} > - < b_n, \mathbf{x} >.$$

With :

$$A_n = \begin{pmatrix} 4 & -2 & 0 & \dots & \dots & \dots & \dots & 0 \\ -2 & 4 & -2 & \ddots & & & & \vdots \\ 0 & -2 & 4 & -2 & \ddots & & & \vdots \\ \vdots & \ddots & -2 & 4 & -2 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & -2 & 4 & -2 & 0 \\ \vdots & & & & \ddots & -2 & 4 & -2 \\ 0 & \dots & \dots & \dots & \dots & 0 & -2 & 4 \end{pmatrix} \text{ and } b_n = \begin{pmatrix} 1 \dots 1 \end{pmatrix}$$

We take place in the case where $n = 2$, so we have :

$$J_2(x) = \frac{1}{2} < A_2x, x > + < b_2, x >$$

with :

$$A_2 = \begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix} \text{ et } b_2 = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

With Matlab, we observe that the eigenvalues of A_n are always positive, so the Hessian matrix of J_n is a positive-definite matrix, so the function J_n is convex.

According to the theorem of the existence of a minimum ($\mathbb{R}^n$ is a not empty closed set, J_n is continuous and $\lim\limits_{||x|| \to +\infty} J_n(x) = +\infty$), we can conclude that J_n owns a unique minimum.

We can draw the graphic representation of J_2:

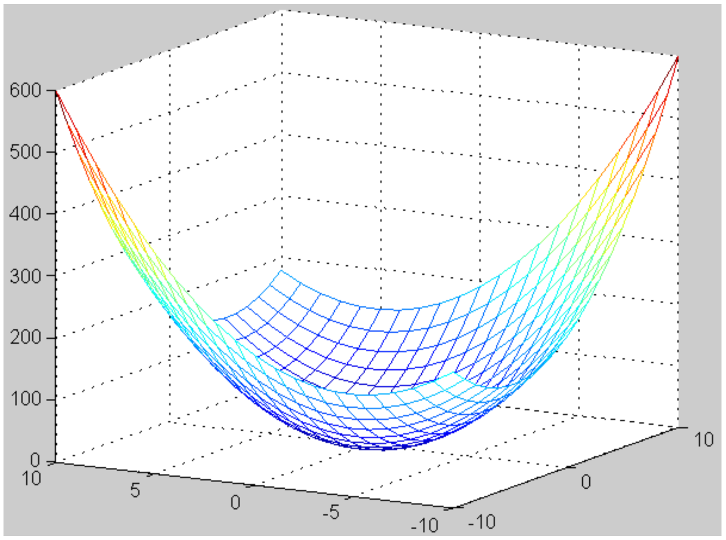


Figure 2: *The graphic representation of J_2 on [-10,10]X[-10,10]*

In addition, we draw the level set and the gradient of J_n:

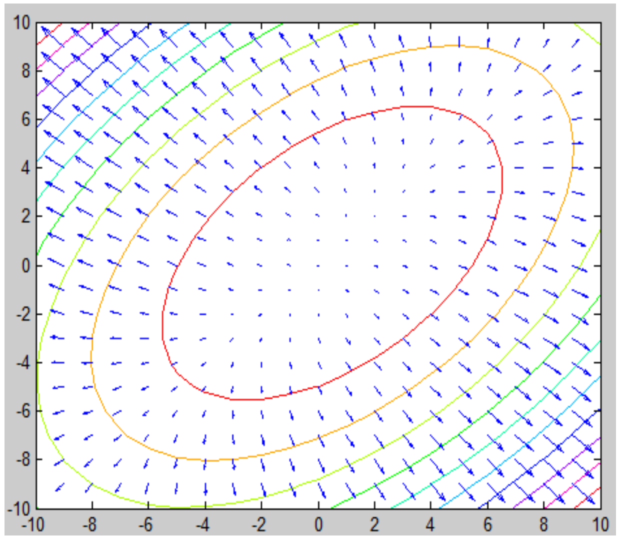


Figure 3: *Level set and the gradient*

1.2.2 Gradient method with fixed step size

This method allows to find a sequence $\mathbf{x}$ which converge to the minimum of J_n. This sequence is defined by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \rho\nabla.J_n(\mathbf{x}_k).$$

Where ρ is the fixed step size.

Thereafter, the algorithm which allows to calculate the gradient method with fixed step size. We take a tolerance of the method that we name K_{max} here.

Algorithm 3 Gradient method with fixed step size

```
Require: $\rho > 0$, $K_{max} \in \mathbb{N}$ $\mathbf{x}_0 \in \mathbb{R}^n$
1: k=0;
2: while $|\mathbf{x}_{k+1} - \mathbf{x}_k| > 10^{-4}$ and $k < K_{max}$ do
3:     $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \rho\nabla J_n(\mathbf{x}_k)$;
4:     $k+1 \leftarrow k$;
5: end while
6: return $\mathbf{x}_k$
```

1.2.3 Gradient method with optimal step length

The gradient method with optimal step length looks like the previous method. The difference is the step between two iterations is set with

$$\rho_n = \min_{\alpha\in\mathbb{R}} \; (x_n + \alpha\nabla J_n(\mathbf{x}_k))$$

Algorithm 4 Gradient method with optimimal step size

```
Require: $K_{max} \in \mathbb{N}$ $\mathbf{x}_0 \in \mathbb{R}^n$, [a,b]
1: i=0;
2: $\rho_0 = golden\_section(a, b)$
3: while $|\mathbf{x}_{k+1} - \mathbf{x}_k| > 10^{-4}$ and $k < K_{max}$ do
4:     $\rho_k = golden\_section(a, b)$
5:     $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \rho_k\nabla J_n(\mathbf{x}_k)$;
6:     $k+1 \leftarrow k$;
7: end while
8: return $\mathbf{x}_k$
```

The function $golden\_section(a, b)$ use the golden section search algorithms explained in the part 1.1.1 in order to search the value α which, in this part, minimize $J_n(X_k + \alpha\nabla J_n(\mathbf{x}_k))$.

1.2.4 Conjugate gradient method

Let us consider A, a positive definite matrix, and let J be a quadratic functional definite by

$$J : \mathbb{R}^n \to \mathbb{R}$$
$$x \to J(x) = \frac{1}{2} < Ax.x > - < b.x >$$

The function J is strictly convex function, twice continuously differentiable. The calculation of the gradient gives us : $\nabla J(x) = Ax - b$. So, the minimum of J is carried out in x^* such that : $Ax^* = b$.

Definition Two vectors (or directions) $\mathbf{d}_1$ and $\mathbf{d}_2$ are conjugates for the matrix A if $A\mathbf{d}_2.\mathbf{d}_1 = 0$.

Assume that we know k conjugate directions $\mathbf{d}^{(0)}$,..., $\mathbf{d}^{(k-1)}$. The descent method is, starting from $\mathbf{x}^{(0)} \in \mathbb{R}^n$, to compute $\mathbf{x}^{(k+1)}$ such that

$$J(\mathbf{x}^{(k+1)}) = J(\mathbf{x}^{(k)} + \rho^{(k)}\mathbf{d}^{(k)}) = \min_{\rho\in\mathbb{R}} J(\mathbf{x}^{(k)} + \rho\mathbf{d}^{(k)})$$

Using the first-order condition for the minimum, we have :

$$\nabla J(\mathbf{x}^{(k)} + \rho^{(k)}\mathbf{d}^{(k)}).\mathbf{d}^{(k)} = 0$$

In our case we obtain

$$\rho^{(k)} = -\frac{(r^{(0)}, d^{(k)})_A}{||d^{(k)}||_A^2}$$

where $(.)_A$ is the dot product associated to the matrix A and the residual vector $\mathbf{r}^{(0)}$ is initialized with : $\mathbf{r}^{(0)} = A\mathbf{r}^{(0)}$ - $\mathbf{b}$.

Algorithm 4 implements the conjugate gradient method in the case of quadratic functionals. The usefulness of this algorithm lies in the next corollaries.

Corollary *The conjugate gradient algorithm converges in at most n iterations.*

We compute the direction $\mathbf{d}^{(k)}$ using Gram-Schmidt. So we have the following proposition

Proposition The descent directions $\mathbf{d}^{(k)}$ are mutually conjugate.

Algorithm 5 Conjugate gradient algorithm for a quadratic functional

```
1: k = 0
2: choose $\mathbf{x}^{(0)} \in \mathbb{R}^n$
3: choose $\epsilon > 0$
4: choose $\epsilon_1 > 0$
5: set $\mathbf{r}^{(0)} = A\mathbf{r}^{(0)}$ - $\mathbf{b}$
6: while ($||\mathbf{x}^{(k+1)}$ - $\mathbf{x}^{(k)}|| \geq \epsilon$) and ($k \leq k^{max}$) do
7:    if $||\mathbf{r}^{(k)}|| < \epsilon_1$ then
8:       stop
9:    else
10:      if ($k = 0$) then
11:         $\alpha^{(k)} = -\frac{(r^{(k)}, d^{(k-1)})_A}{||d^{(k-1)}||_A^2}$
12:         set $\mathbf{d}^{(k)} = \mathbf{r}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k-1}$
13:      end if
14:      $\rho^{(k)} = -\frac{(r^{(k)}, d^{(k)})_A}{||d^{(k)}||_A^2}$
15:      set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \rho\mathbf{d}^{(k)}$
16:      $\mathbf{r}^{(k+1)} = A\mathbf{x}^{(k+1)} - \mathbf{b}$
17:      $k = k + 1$
18:   end if
19: end while
```

1.2.5 Preconditioned conjugate gradient method

We begin this section by reminding what is the condition number of a matrix. We aim to solve the linear system $A\mathbf{X} = b$, where for example

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$$

If $\mathbf{b} = \begin{pmatrix} 32 \\ 33 \\ 33 \\ 31 \end{pmatrix}$, the solution of the system is $\mathbf{X} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

Now if b is modified slightly in the following manner : $\mathbf{b} = \begin{pmatrix} 32.1 \\ 32.9 \\ 33.1 \\ 30.9 \end{pmatrix}$.

In this case, the solution is $\mathbf{X} = \begin{pmatrix} 92 \\ -12.6 \\ 4.5 \\ -11 \end{pmatrix}$. We note that the solution is greatly modified.

Thus a small change in $\mathbf{b}$ produces a large change in $\mathbf{X}$. The matrix A is ill-conditioned. let's remind the following definition

Definition Let $A \in \mathscr{M}_{n,n}(\mathbb{R})$. We define $\kappa(A)$, the condition number of a matrix by $\kappa(A) = ||A||.||A^{-1}||$

The condition number of a matrix can be approximated by $\kappa(A) = \left|\frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}\right|$, where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are maximal and minimal eigenvalues of A respectively.

We want to compare two numerical method of resolution of linear system in the case of sparse matrix : the conjugate gradient and the preconditioned conjugate gradient method.

Let us consider the system $A_n\mathbf{X} = \mathbf{b}$ where $\mathbf{X} \in \mathscr{M}_{n,1}(\mathbb{R})$ and

$$A_n = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix} \text{ and } b_n = \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{pmatrix}$$

First we implement the conjugate gradient method. Let remind the algorithm

$$\begin{cases} \mathbf{x}^0 \text{ is given, } \mathbf{r}^0 = A\mathbf{x}^0 - \mathbf{b} \text{ and } \mathbf{d}^0 = -\mathbf{r}^0; \\ \\ \mathbf{x}^{n+1} = \mathbf{x}^n + \rho_n \mathbf{d}^n,\ \rho_n = -\frac{(r^n, d^n)}{(Ad^n, d^n)} \\ \\ \mathbf{r}^{n+1} = A\mathbf{x}^{n+1} - \mathbf{b}; \\ \\ \mathbf{d}^{n+1} = -\ \mathbf{r}^{n+1} + \beta_n \mathbf{d}^n,\ \beta_n = \frac{||r^{n+1}||^2}{||r^n||^2} \end{cases}$$

Now we implement the preconditioned conjugate gradient method. Instead of solve the system $A_n\mathbf{X} = \mathbf{b}$, we first preconditioned the matrix A_n with the incomplete Cholesky factorization. The Cholesky factorization of a positive definite matrix A is $A = LL^t$ where L is a lower triangular matrix.

We want to find a matrix M such that the condition number of $M^{-1}A_n$ is better than the condition number of A_n. Let's denote by RI the incomplete Cholecky factorization, we set $M = RI^t.RI$.

The system becomes : $M^{-1}A_n\mathbf{X} = M^{-1}\mathbf{b}$. The algorithm changes a bit (see below).

$$\left\{\begin{array}{c} \mathbf{x}^0 \text{ is given, } \mathbf{r}^0 = M^{-1}A\mathbf{x}^0 - M^{-1}\ \mathbf{b} \text{ and } \mathbf{d}^0 = -\mathbf{r}^0; \\ \mathbf{x}^{n+1} = \mathbf{x}^n + \rho_n\mathbf{d}^n,\ \rho_n = -\frac{(r^n, d^n)}{(Ad^n, d^n)} \\ \mathbf{r}^{n+1} = M^{-1}A\mathbf{x}^{n+1} - M^{-1}\ \mathbf{b}; \\ \mathbf{d}^{n+1} = -\ \mathbf{r}^{n+1} + \beta_n\mathbf{d}^n,\ \beta_n = \frac{||r^{n+1}||^2}{||r^n||^2} \end{array}\right.$$

In the next section, we compare the two methods (number of iterations, computing time).

1.3 Comparison

In this part, we wanted to compare the gradient methods definied in the previous part. First, we are going to define the *Rosenbrock banana* function.

1.3.1 Comparison between the Gradient methods

1.3.1.a Gradient method with fixed step size

We used this algorithm of the gradient method with fixed step to compute the minimum of J_n for different values of n. We have taken $\rho = 0.1$ and $\mathbf{x}^0 = \mathbf{1}$. We wanted a tolerance of the method of 10^{-4}. We put the number of iteration and the time of execution in a table:

n	Number if iterations	Time of computation (second)
2	43	0.0054
3	67	0.01
5	197	0.03
10	722	0.079
20	2 950	0.21
30	6 820	0.55
50	19 789	2.44
100	84 705	25.05

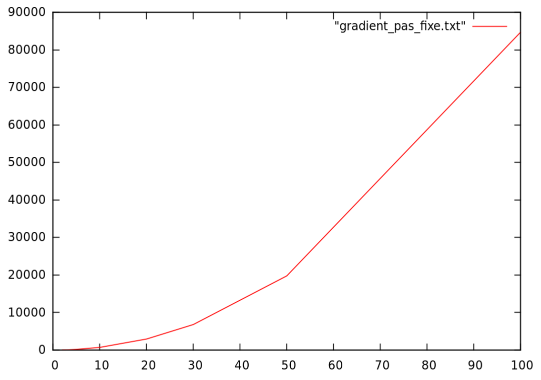


Figure 4: *Iterations as a function of dimension*

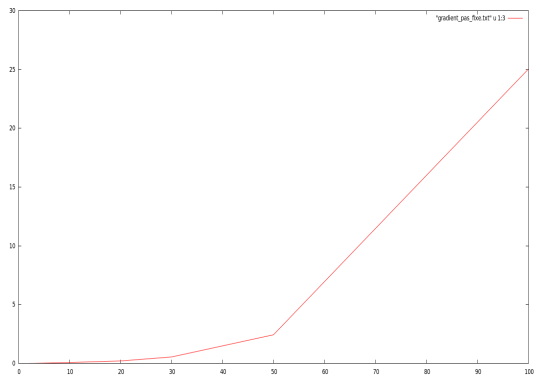


Figure 5: *Time of computation as a function of dimension*

1.3.1.b Gradient method with optimal step length

We have taken $\mathbf{x}^0 = \mathbf{1}$. The stopping criteria was $||x_n - x_{n-1}|| < 10^{-4}$.

n	Number if iterations	Time of computation (second)
2	2	0.00154
3	11	0.0415
5	74	0.3257
10	248	1.372
20	926	8.744
30	2 102	26.949
50	5 962	124.602
100	22 338	902.007

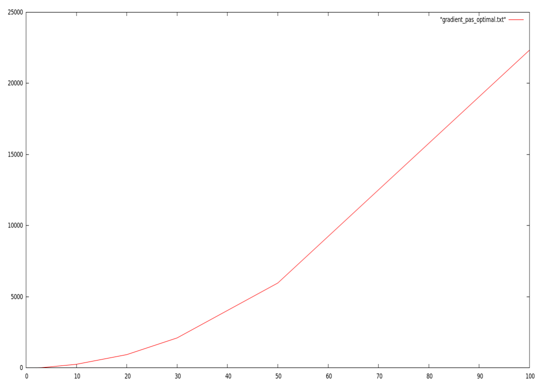


Figure 6: *Iterations as a function of dimension*

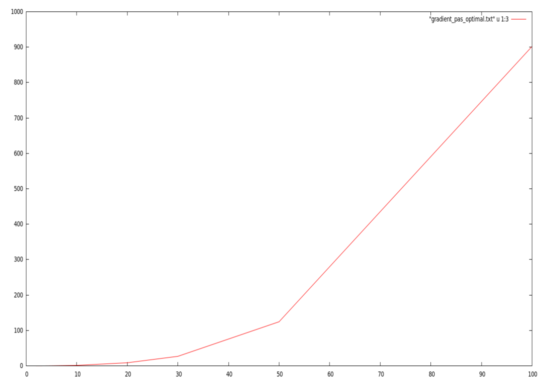


Figure 7: *Time of computation as a function of dimension*

We notice than this method takes more time than the Gradient method with fixed step size. This difference can be explained by the use of the golden section search algorithm. Each iteration takes 0.004572 seconds and we use it a lot of time. We can also notice than this method takes less iteration than the first algorithm.

1.3.1.c Conjugate gradient method

We have taken $\mathbf{x}^0 = \mathbf{1}$. The stopping criteria were $||r|| < 10^{-20}$ and $||x^n - x^{n-1}|| < 10^{-30}$

n	Numbers of iterations	Computing time (second)
2	1	0.0027
3	3	0.0036
5	4	0.0040
10	7	0.0078
20	14	0.0078
30	19	0.0024
50	30	0.0061
100	54	0.0072

This method is faster in terms of numbers of iterations and computing time than the two previous methods (fixed step size and optimal step length). This method is efficient in the case where the matrix A is symmetric, positive definite, which is the case here.

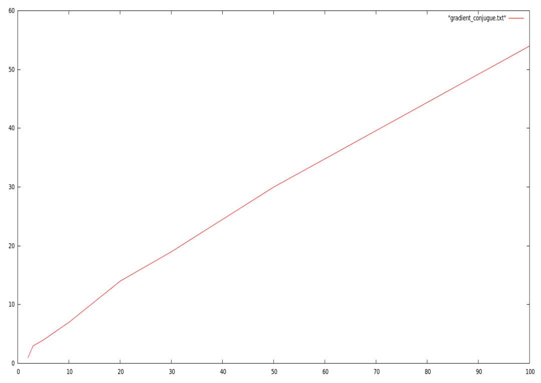


Figure 8: *Iterations as a function of dimension*

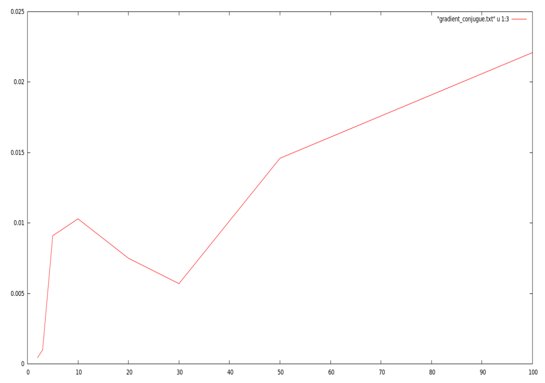


Figure 9: *Time of computation as a function of dimension*

1.3.1.d Preconditioned conjugate gradient method

Here we compare the conjugate gradient and the preconditioned gradient method with the linear system defined in section 1.3.4.

We chose the quadratic functional $f(x) = \frac{1}{2} < Ax, x > - < b, x >$. We have taken $\mathbf{x^0} = \mathbf{1.5}$.

n	$Cond(A_n)$	$Cond(M^{-1}A_n)$
10	60	1
50	$1.3\ 10^3$	1
100	$5.1\ 10^3$	1
500	$1.255\ 10^5$	1
1000	$5\ 10^5$	1

This is an approximation calculated thanks to Matlab. Anyway the condition number of $M^{-1}A_n$ is better than the condition number of A_n.

The preconditioned conjugate gradient method is faster than the gradient conjugate method (3 iterations against 502 for the case $n = 1000$). Figure 11 shows the computing time for the two methods as a function of dimension.

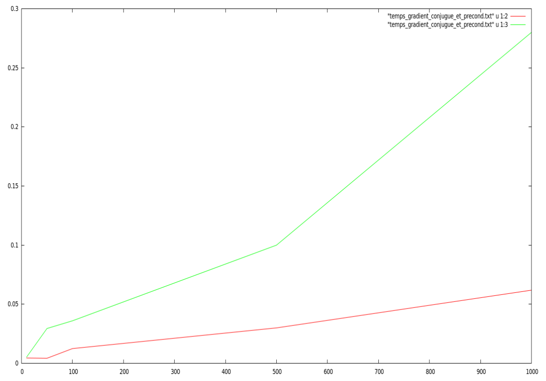


Figure 10: *Computing time for the gradient conjugate (in red) and the preconditioned conjugate method (in green)*

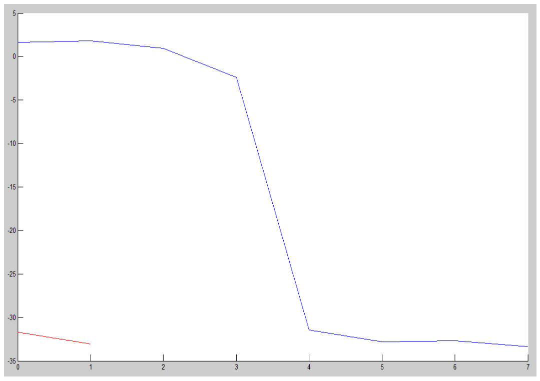


Figure 11: *ln(*$||X_k - X^*||$*) with* $n = 10$

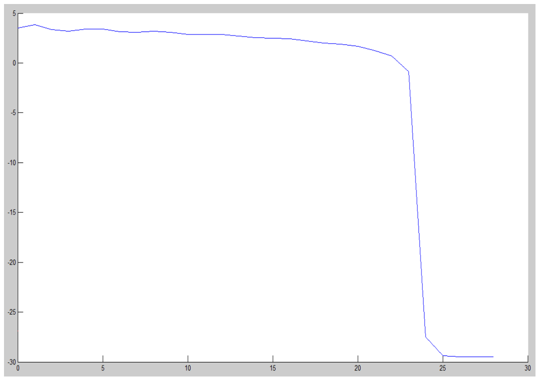


Figure 12: *ln(*$||X_k - X^*||$*) with* $n = 50$

The previous figures show the logarithmic residue as function of number of iterations (in blue the conjugate gradient method and the preconditioned gradient method in red). So we can see the convergence of the two methods.

1.3.2 The *Rosenbrock banana* function

1.3.2.a Definition

We define $f \in \mathbb{R}^n$, the famous *Rosenbrock banana* function:

$$f(x, y) = (x - 1)^2 + 10(x^2 - y)^2.$$

1.3.2.b The critical points of f

We are going to calculate the critical point(s) of f. In first, we have the gradient:

$$\nabla f(x, y) = \begin{pmatrix} 2(x-1) + 20x(x^2 - y) \\ 20(y - x^2) \end{pmatrix}.$$

We determined where $\nabla f(x, y) = 0$) and we got only one critical point: (1,1).
According to the theorem of the existence of a minimum and the Euler equality (on the open set $\mathbb{R}^2$), we can conclude that (1,1) is the unique point which minimizes f.

1.3.2.c The Hessian matrix of f

We computed the gradient of the gradient and we obtained the Hessian matrix of f:

$$\nabla^2 f(x, y) = \begin{pmatrix} 2(60x^2 - 20y + 1) & -40x \\ -40x & 20 \end{pmatrix}.$$

Then, in (1,1), the Hessian matrix of f is:

$$H = \begin{pmatrix} 82 & -40 \\ -40 & 20 \end{pmatrix}.$$

The eigenvalues of H are 101,6 and 0,39, so H is a positive-definite matrix. According to the function *cond* in Matlab, the condition number of H is **258**. We remark we find the famous approximation of the condition number : $\frac{\lambda_{max}}{\lambda_{min}}$.

This number is high, it seems that the minimum of f takes place in a kind of "valley", it discomforts the convergence of gradient methods.

1.3.2.d The Taylor expansion

The Taylor expansion of f is :

$$f(x,y) = f(1,1) + \frac{1}{2}\frac{\partial^2 f(x,y)}{\partial x^2}(x-1)^2 + \frac{1}{2}\frac{\partial f(x,y)}{\partial x \partial y}(x-1)(y-1) + \frac{1}{2}\frac{\partial^2 f(x,y)}{\partial y^2}(y-1)^2 + o(x^2,y^2).$$

$$f(x,y) = f(1,1) + 41(x-1)^2 - 40(x-1)(y-1) + 10(y-1)^2 + o(x^2,y^2).$$

To finish:

$$f(x,y) = 41(x-1)^2 - 40(x-1)(y-1) + 10(y-1)^2 + o(x^2,y^2).$$

1.3.2.e Graphic representation of f

Figure 4 shows the graphic representation of f.

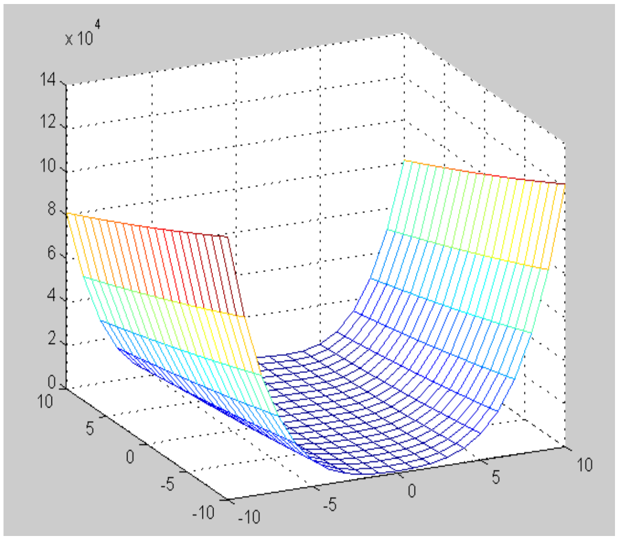


Figure 13: *Rosenbrock banana.*

1.3.2.f Numerical study

In this part, we are doing a comparison of two gradient methods, the gradient method with fixed step size and the Gradient method with optimal step size.

We choose a step size $\beta = 0.01$.

We plot on the same figure the points we obtained in each iteration and hundred level set of the Rosenbrock Banana with this two methods.

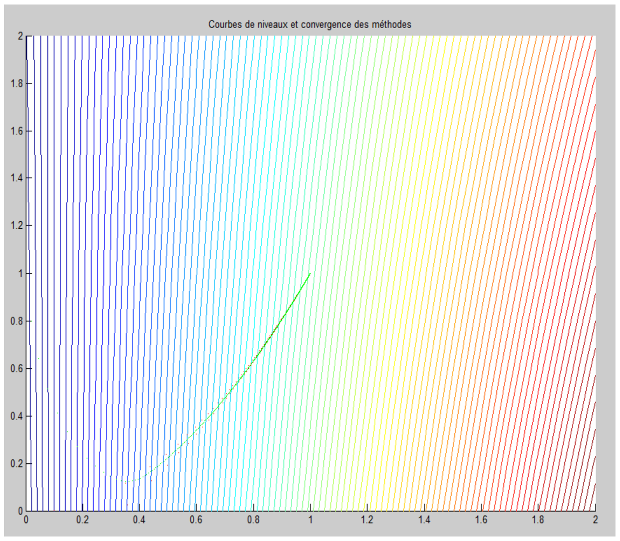


Figure 14: *level set on [-10,10].*

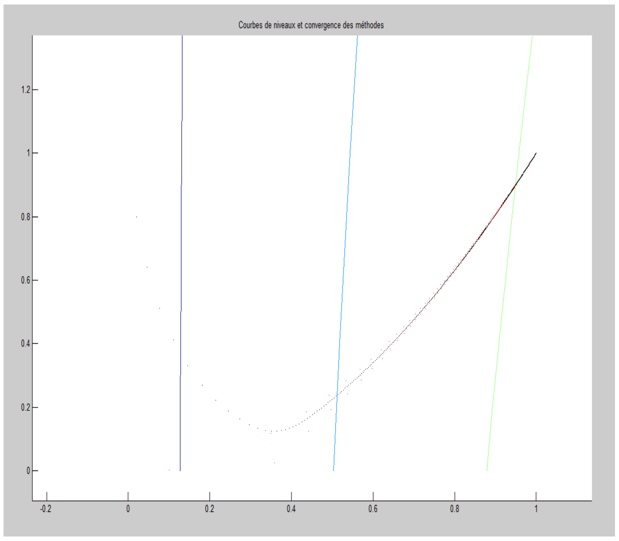


Figure 15: *level set on [0,1.2].*

Then we plot the curve which represent $ln(\|X_k - X\|)$.

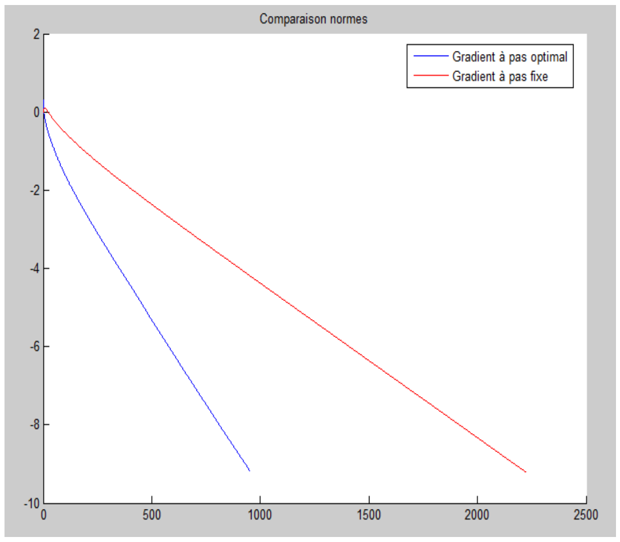


Figure 16: $ln(\|X_k - X\|)$.

We can see that the gradient method with optimal step size takes less iterations than the other one.

2 Constrained optimization

2.1 Obstacle problem

2.1.1 Problem

Let g be a continuous function in [0,1]. The obstacle problem will be:

$$\text{find u : [0,1]} \to \mathbb{R} \text{ such as } \begin{cases} -u''(x) \geq 1 & x \in [0,1] \\ u(x) \geq g(x) & x \in [0,1] \\ (-u''(x) - 1)(u(x) - g(x)) = 0 & x \in [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

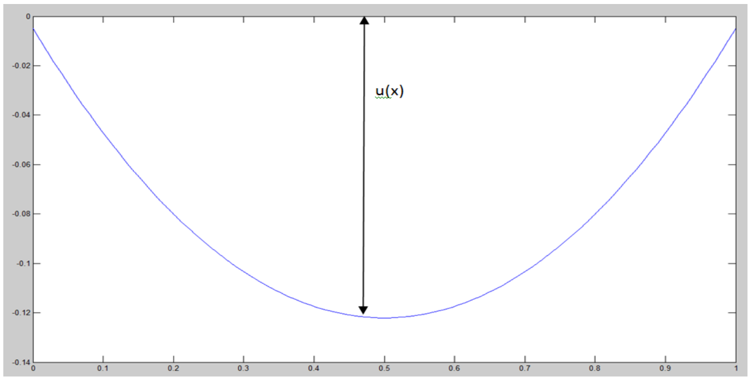


Figure 17: *Equation of a rope clamped at the extremities*

This figure represents a rope subject to its weight (equal to 1 here) and clamped at the extremities $x = 0$ and $x = 1$. We add an obstacle in the problem and the rope has to be above this.

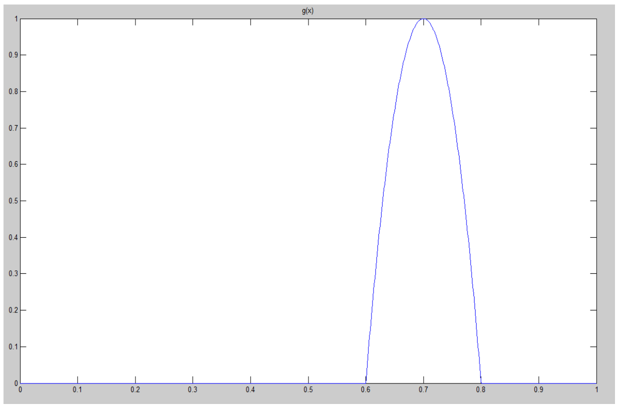


Figure 18: *Representation of the obstacle*

In this case $g(x) = max(0, 1 - 100(x - 0,7)^2)$.

2.1.2 Discretization

We discretize this problem by introducing an uniform mesh: $x_j = jh$ where h is the step of the mesh and $j \in \{0, ..., n+1\}$ where n is a integer. We set $h = \frac{1}{n+1}$ and g_j=g(x_j).

The problem becomes:

$$\text{find } u_j = u(x_j) \text{ such as } \begin{cases} -\frac{u_{j-1}-2u_j+u_{j+1}}{h^2} \geq 1 & j \in \{0,..,n+1\} \\ u(x) \geq g(x) & j \in \{0,..,n+1\} \\ (-\frac{u_{j-1}-2u_j+u_{j+1}}{h^2} - 1)(u(x) - g(x)) = 0 & j \in \{0,..,n+1\} \\ u_0 = u_{n+1} = 0 \end{cases}$$

We introduce the matrix A and the vectors b and g:

$$\text{A}=\frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & \dots & 0 \\ -1 & 2 & -1 & \ddots & & & \vdots \\ 0 & -1 & 2 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 2 & -1 & 0 \\ \vdots & & & \ddots & -1 & 2 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 2 \end{pmatrix}, \text{ b}=\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \text{ et g}=\begin{pmatrix} g_1 \\ \vdots \\ g_n \end{pmatrix}$$

with u=$\begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}$ we have

$$\text{u is a solution of the problem} \Leftrightarrow \begin{cases} \underset{v\in K}{min}\ \frac{1}{2} < Av, v > - < b, v > \\ K = \{v \in \mathbb{R}^n : v \geq g\} \end{cases}$$

Let J be the functional in $\mathbb{R}^n$ such as $J(v) = \frac{1}{2} < Av, v > - < b, v >$

2.1.3 Numerical resolution

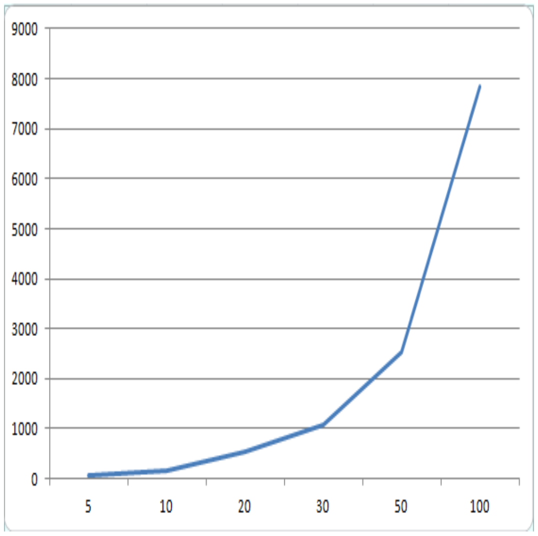
To solve this problem, we use the projected gradient method. This algorithm is very close to a gradient method with fixed step size. In each iteration the gradient method can gives us a X_k outside K. So we have to project the result with $\Pi_K(v)$. In this problem we used $\Pi_K(v) = max(v_i, g_i)$.

Algorithm 6 Projected gradient method

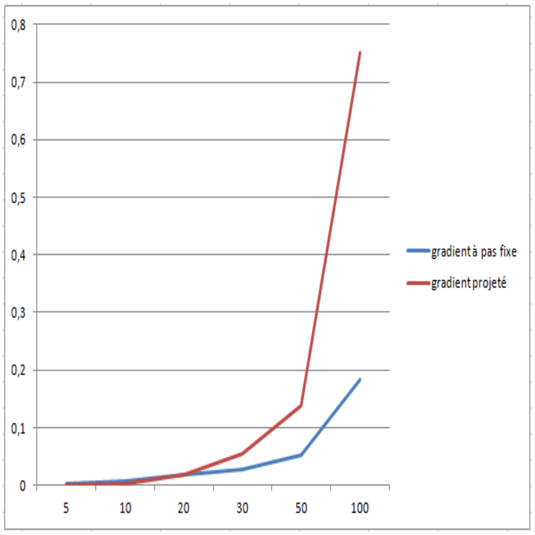
```
Require: $K_{max} \in \mathbb{N}$, $\mathbf{X}_0 \in \mathbb{R}^n$, [a,b], $\lambda_1$ and $\lambda_n$ the eigenvalues
1: n=size($X_0$)
2: i=0;
3: while $x_{k+1} - x_k > 10^{-4}$ and $k < K_{max}$ do
4:     for j=1:n do
5:         $X_j = max(X_j, g_j)$
6:     end for
7:     $\mathbf{x}_{k+1}$=$\mathbf{x}_k - \frac{2}{\lambda_1+\lambda_n}\nabla J_n(\mathbf{x}_k)$;
8: end while
9: return $\mathbf{x}_k$
```

λ_1 and λ_n are the first and last eigenvalues of the matrix A.

We compute this algorithm for $n = 5$ to $n = 100$ and we display the number of iteration and the computing time regarding to n.



a - Number of iteration



b - Computing time

Figure 19: Comparison between the gradient method with fixed step and the projected gradient method

The number of iteration is exactly the same between the gradient method with fixed step and the projected gradient method.

Here is representations of the result of the projected gradient method in our problem with $n = 20$ and $n = 200$.

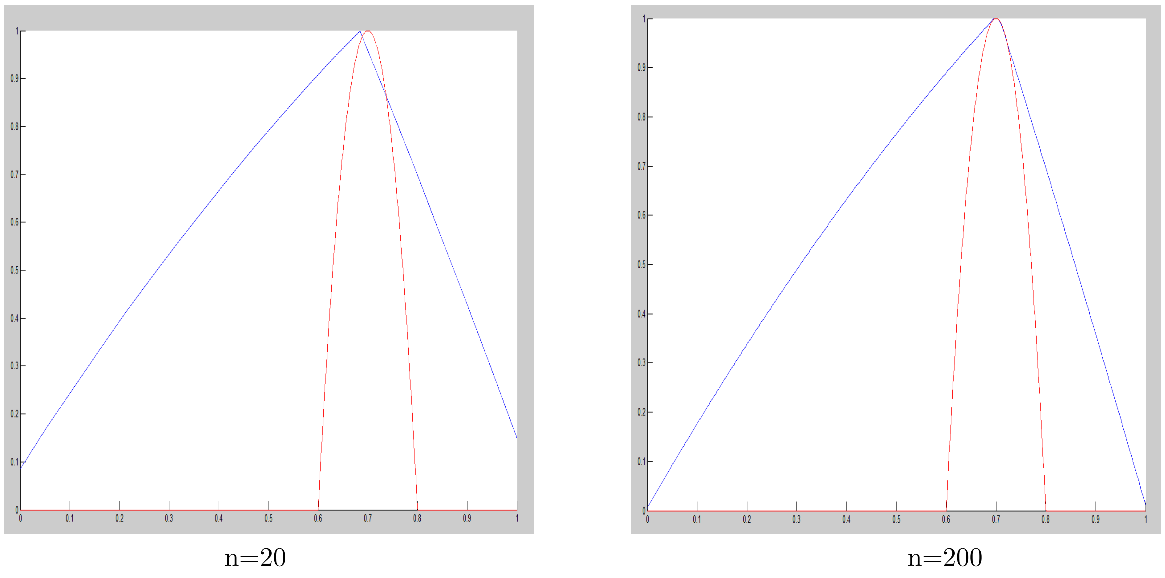


Figure 20: Solution to the obstacle problem

We can see that, with $n = 20$, the number of nodes in the mesh is not enought to have a great representation.

2.2 Distance from a point to a line

2.2.1 Theory

We want to numerically determine the lowest distance from a point, $\mathbf{x}_0$ in dimension n to a hyperplane, $(\mathscr{H})$ defined by the equation $A\mathbf{x} = b$, where $A \in \mathscr{M}_{m,n}(\mathbb{R})$ with $m < n$.

The constraint minimization is written :

$$\min_{F(x)=0} J(\mathbf{x}).$$

With: $F(\mathbf{x}) = A\mathbf{x} + b$ and $J(\mathbf{x}) = \frac{1}{2}.^{t}(\mathbf{x} - \mathbf{x}_0).(\mathbf{x} - \mathbf{x}_0)$

The Lagrangian of this problem is :

$$L(\mathbf{x}) = J(\mathbf{x}) + \lambda.F(\mathbf{x})$$

At the optimum, the derivative of L is zero, so :

$$\nabla J(\mathbf{x}) + \lambda^{*}\nabla F(\mathbf{x}) = 0$$

$$\mathbf{x} - \mathbf{x}_0 + \lambda^{*} A^{t} = 0$$

$$A\mathbf{x} - \mathrm{x}_0 + \lambda^{*} AA^{t} = 0$$

$$b - \mathbf{x}_0 + \lambda^{*} AA^{t} = 0$$

$$\lambda^{*} AA^{t} = b - \mathbf{x}_0$$

$$\lambda^{*} = (AA^{t})^{-1} b - \mathbf{x}_0$$

We replace λ^{*} by the expression obtained in the previous compute and we get:

$$\nabla J(\mathbf{x}^{*}) + \lambda^{*}\nabla F(\mathbf{x}) = 0$$

$$\mathbf{x}^{*} = \mathbf{x}_0 + A^{t}(AA^{t})^{-1}(b - \mathbf{x}_0).$$

To compute the distance from a point to a line, we calculate $\mathbf{x}^{*} - \mathrm{x}_0$ and we take A like a row vector.

Then $AA^{t} = ||A||_2^2$ and $(AA^{t})^{-1} = \frac{1}{||A||_2^2}$. By the same way, we have $A^{t} = ||A||_2$.

So $A^{t}(AA^{t})^{-1}$ corresponds to $\frac{1}{||A||_2}$.

Finally, we have:

$$d(\mathbf{x}_0, \mathscr{H}) = \frac{||b - A\mathbf{x}_0||}{||A||}$$

2.2.2 Uzawa Algorithm

2.2.2.a Algorithm

The Uzawa algorithm is a kind of gradient method applied to the Lagrangian. Thereafter, a description of the algorithm where f et g are the constraints.

Algorithm 7 Uzawa algorithm

```
Require: $\rho > 0$, $K_{max} \in \mathbb{N}$ $\mathbf{x}_0 \in \mathbb{R}^n$
1: k=0;
2: while $|\mathbf{x}_{k+1} - \mathbf{x}_k| > 10^{-4}$ and $k < K_{max}$ do
3:     Compute $\mathbf{x}_k$ solution of min $L(\mathbf{x}, \mu_k, \lambda_k)$
4:     Compute $\mu_{k+1}, \lambda_{k+1}$ with:
5:     $\mu_{i,k+1} \leftarrow \mu_{i,k} - \rho f_i(\mathbf{x}_k)$; for $i = 1..p$
6:     $\lambda_{i,k+1} = \max(0, \lambda_{i,k} + \rho g_i(\mathbf{x}_k)$; for $j = 1..m$
7:     $k + 1 \leftarrow k$;
8: end while
9: return $\mathbf{x}_k$
```

2.2.2.b Example

To test the Uzawa Algorithm we took $A = \begin{pmatrix} 1 & 1 \end{pmatrix}$, $b = 2$ et $\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.
In this case, the hyperplane $\mathscr{H}$ is a line defined by the equation $y = 2 - x$.
We obtain the point $\mathbf{x}_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, which is well the orthographic projection of $\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ of the line of equation $y = 2 - x$.

Conclusion

In definitive, this project allows us to see in practice any algorithms that we studied in theory. We can see some gradient method are useless in some cases, like the gradient method with optimal step size in first problem solved. Indeed the number of iterations is lower than the gradient method with fixed step size but the method takes more time to compute.