MASTER Imagerie Robotique Ingénierie pour le Vivant

An introduction to optimization

Practical work

Yassir IDSOUGOU / Abdelhak KADDOUR

TPS - PROMO 2019 -
14/02/2018

Supervisor: M.OMRAN Hassan

ECOLE D'INGENIEURS TELECOM PHYSIQUE STRASBOURG

Part 1.

1.1 First problem

The first problem that we're studying in this practical work is:

$$\min f(x) = 3x_1^2 + 0.5x_2^2 + 2x_2 + 2$$

$$\text{s.t.} \quad x_1, x_2 \geq 0$$

We verify whether the FONC are satisfied at the following points $[1\ 2]^T$,$[0\ 3]^T$, $[1\ 0]^T$ and $[0\ 0]^T$.

First of all, taking into account the constraint of the problem, we can easily guess that $[0\ 0]^T$ is the global minimum of f, so the FONC is satisfied at this point.

For the 3 other points, we have to calculate the gradient. We name it G(x).

We have: $G(x) = \begin{pmatrix} 6x_1 \\ x_2 + 2 \end{pmatrix}$

For $[1\ 2]^T$, we have here an interior point and we easily see that the value of G($[1\ 2]^T$) is not 0. The FONC is not satisfied at this point.

For x=$[0\ 3]^T$,$G(x) = \begin{pmatrix} 0 \\ 5 \end{pmatrix}$. We can take $d = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ which is a feasible direction. However $d^T * G(x) < 0$. The FONC is not satisfied at the point $[0\ 3]^T$.

By the same way, using $d = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, we show that the FONC is not satisfied at the point $[1\ 0]^T$.

1.2 Local minimizers/local maximizers

The second work in this part consists in finding the saddle points, local minimizers and local maximizers (if they exist) of the following functions:

$$f_1(x) = 4.5 - x_1 + 2x_2 - 0.5x_1^2 - 2x_2^2$$

$$f_2(x) = 2x_1^3 + x_1x_2^2 + 5x_1^2 + x_2^2$$

Calculate the Gradients and the Hessian matrix, We name them G_1^1, G_1^2 for f1 and G_2^1, G_2^2 for f2.

We have:

- $G_1^1(x) = \begin{pmatrix} -1 - x_1 \\ 2 - 4x_2 \end{pmatrix}$ and $G_1^2(x) = \begin{pmatrix} -1 & 0 \\ 0 & 4 \end{pmatrix}$
- $G_2^1(x) = \begin{pmatrix} 6{x_1}^2 + {x_2}^2 + 10x_1 \\ 2x_2 + 2x_1x_2 \end{pmatrix}$ and $G_2^2(x) = \begin{pmatrix} 12x_1 + 10 & 2x_2 \\ 2x_2 & 2x_1 + x_2 \end{pmatrix}$

For f_1 , $G_1^1(x) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ for $x_0 = [-1\ 0.5]^T$ but $\det(G_1^2(x)) < 0$. There are not local minimizers or maximizers. x_0 is a saddle point.

For f_2 , $G_2^1(x) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ if $6{x_1}^2 + {x_2}^2 + 10x_1 = 0$ and $2x_2 + 2x_1x_2 = 0$

Then $x_2(1 + x_1) = 0 \rightarrow x_2 = 0$ or $x_1 = -1$.

First case : $x_2 = 0$. Then $x_1 = 0$ or $x_1 = \frac{-5}{3}$.

Second case $x_1 = -1$. Then $x_2 = -2$ or $x_2 = 2$.

We have four points to study : $x_a = [-1\ -2]^T$, $x_b = [-1\ 2]^T$, $x_c = [0\ 0]^T$, $x_d = [\frac{-5}{3}\ 0]^T$.

By calculating the Hessian Matrix, we see that x_d is a local maximizer. Determining the

1.3 Descent/Increase direction

Now we investigate, using two different methods, whether $d = [2\ -1]^T$ is a descent or an increase direction at $x_0 = [1\ 1]^T$ with respect to the function:

$$f(x) = x_1^2 + x_1 x_2 - 4x_2^2 + 5$$

We use two different methods. The first use the gradient, we name it G.

$$G(x) = \begin{pmatrix} 2x_1 + x2 \\ x_1 - 8x_2 \end{pmatrix}$$

Then $G(x_0) = \begin{pmatrix} 3 \\ -7 \end{pmatrix}$ and $G(x_0)^T * d = 13 > 0$. We can conclude d is an increase direction at x_0.

Second method : $lim_{\alpha \to 0} \frac{f(x+\alpha d) - f(x)}{\alpha}$

$f(x_0) = 3$ and $f(x_0 + \alpha d) = 3 + 13\alpha - 2\alpha^2$. Then $lim_{\alpha \to 0} \frac{f(x_0+\alpha d) - f(x_0)}{\alpha} = 13 > 0$.

We found the same result for the two methods.

1.4 Quadratic programming problem

The last problem that we're studying in this first part of our work consists in solving the following unconstrained quadratic programming problem:

$$\min f(x) = (x_2 + x_1 - 3)^2 + 2(x_2 - x_1 + 1)^2$$

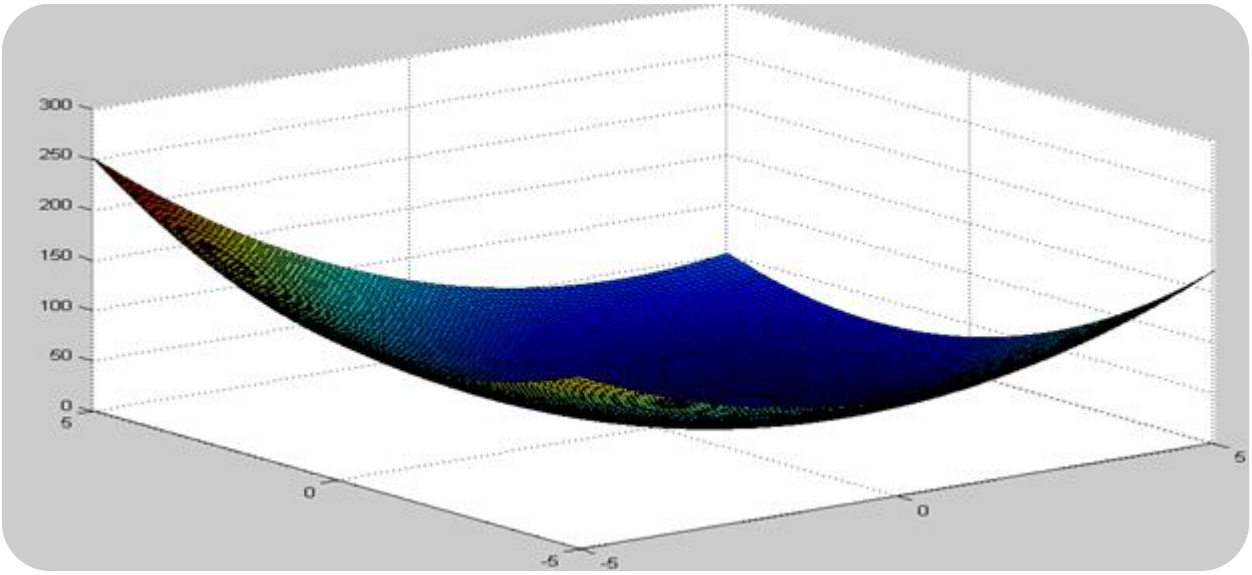
f is a sum of two positive functions, then the minimum is reached when these two functions reach at the same time the 0 value. Then we must have :

$$x_2 + x_1 = 3 \; ; x_2 - x_1 = -1$$

Then we can conclude that $x = [2\ 1]^T$ is the global minimum of this quadratic function.

Here is the Matlab script to see the function:

```
clear all; close all; clc;

a = [-5:0.1:5];
b = a;
for (i = 1:length(a))
    for (j = 1:length(b))
        y(i,j) = (a(i)+b(j)-3)^2+2*(a(i)-b(j)+1)^2;
    end
end
surf(a,b,y);
```



Part 2.

2.1 Golden section

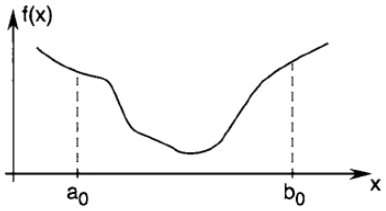
In this part, we study the problem

$$\min f(x) = x^4 + 4x^3 + 9x^2 + 6x + 6$$

$$\text{s.t.} \quad x \in [-2,2]$$

using the golden section with a tolerance $\varepsilon = 0.01$.

The problem consists on through a closed interval $[a_0, b_0]$, a **unimodal** function, and only objective value information, we find a point that is no more than ε away from the local minimize.

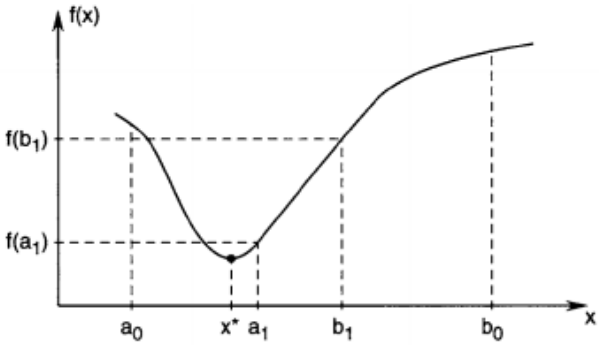


First thing to do is to input our function f:

```
function [fx] = fct21(x)
fx=x^4+4*x^3+9*x^2+6*x+6;

end
```

The idea is to narrow the search range and evaluating the function at **two points** to reduce the searching range. In the first iteration, we evaluate at $a_1, b_1 \in [a_0, b_0]$, where $a_1 < b_1$.

- ✓ if $f(a_1) < f(b_1)$ then $x^* \in [a_0, b_1]$
- ✓ if $f(a_1) > f(b_1)$ then $x^* \in [a_1, b_0]$



At iteration k, we consider the interval $[a_0^k, b_0^k]$. We choose the points in a way to have symmetry:

$$a_1^k - a_0^k = b_0^k - b_1^k = \rho(b_0^k - a_0^k) \quad \text{With} \quad \rho = \frac{a_1 - a_0}{b_0 - a_0} < 1/2$$

Using this equation, we will find a two-degree question with ρ. After the resolution of the equation, we find that:

$$\rho = \frac{3 \pm \sqrt{5}}{2}$$

Note that after N iteration the search range is reduced by $(1 - \rho)^N$.

So here is the Matlab script to solve our problem:

```
clearall; close all; clc;

a0k=-2;
b0k=2;
eps=0.01;
rho=(3-5^(1/2))/2;
m=0;                                %number of iterations

while abs(a0k-b0k)>eps
    m=m+1;
    a1k = a0k+rho*(b0k-a0k);
    b1k = b0k-rho*(b0k-a0k);
if fct21(b1k)>fct21(a1k)
        b0k=b1k;
else a0k=a1k;
end

end

val=(a0k+b0k)/2;
```

After compiling the program, we found as a result:

```
>>>m = 13

>>val =  -0.4482

>> a0k = -0.4520

>> b0k = -0.4444
```

2.2 Newton and Secant method

The second problem that we solve in this part is:

$$\min f(x) = 2x^4 - 5x^3 + 100x^2 + 30x - 75$$

$$\text{s.t.} \quad x \in \mathbb{R}$$

with a stopping criterion $\left|\frac{df}{dx}(x_k)\right| \leq 0.001$ using two different methods:

2.2.1 Newton's method, with an initial condition $x_0 = 2$

2.2.2 Secant's method, with initial conditions $x_0 = 2.1$ and $x_1 = 2$

First thing to do is to input our function f and its derivative that will be necessary for both methods:

```
function [fx2] = fct22(x)
    fx2=2*x^4-5*x^3+100*x^2+30*x-75;

end
```

```
function [dfx2] = dfct22(x)
    dfx2=8*x^3-15*x^2+200*x+30;

end
```

2.2.1 Newton's method

The problem consists in through a twice continuously differentiable function and objective, derivative and 2nd derivative function information, find an approximate minimizer.

Newton's method doesn't need intervals but must start with an initial condition x_0.

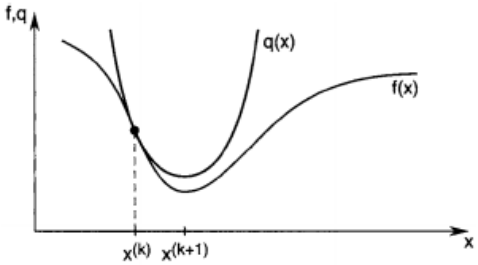
First thing to do is to input our 2nd derivative of our function f:

```
function [d2fx2] = d2fct2(x)
d2fx2=24*x^2-30*x+200;

end
```

The idea is to minimize a second-order approximation of f at each iteration. We consider the case where $f''(x) > 0$ and at each iteration minimize second-order approximation of $f$at x_k:

$$q_k(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2$$

In this case the minimum of q_k is obtained by finding the point where $q'_k(x) = 0$, which yields $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$.



So here is the Matlab script to solve our problem:

```
xk=2;
m=0;                          %number of iterations
while (abs(dfct22(xk))>0.0001 & m<500)
xk=xk-(dfct22(xk)/d2fct22(xk));
        m=m+1;
end

val=xk1;
```

After compiling the program, we found as a result:

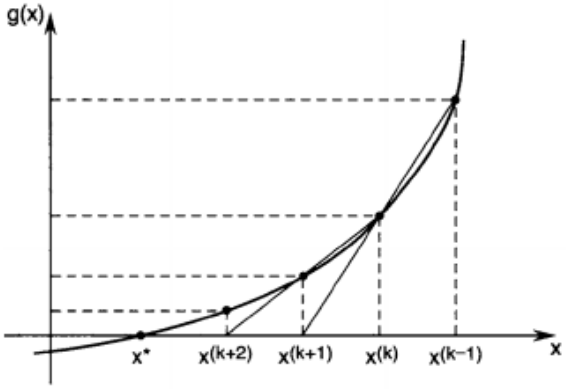
```
>>newton

>>val = -0.1482
```

2.2.2 Secant method

Recall Newton's method for minimizing f uses f''. If the second derivative is not available, then we approximate it: $f''(x_k) \approx \frac{f'(x_k)-f'(x_{k-1})}{x_k-x_{k-1}}$, than we have the following iterative method

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})} f'(x_k)$$

This method needs two initial points.



Note that the secant method is slightly slower than Newton's method but is cheaper (don't need '').

So here is the Matlab script to solve our problem:

```
x0=2.1;
x1=2;
a=0;

while abs(dfct22(x1))>0.0001
    a=x1;
x1=x1-(x1-x0)/(dfct22(x1)-dfct22(x0))*dfct22(x1);
x0=a;
end

val=x1;
```

After compiling the program, we found as a result:

```
>>secante

>> val= -0.1482
```

Part 3.

3.1 Steepest descent method

In this part, we study the problem

$$\min f(x) = 1 + 2x_1 e^{-x_1^2 - x_2^2}$$

$$\text{s.t.} \quad x \in \mathbb{R}^2$$

using the steepest descent method, starting from $x_0 = [-0.5\ 0.5]^T$. We consider the stopping criterion $\|\nabla f(x_k)\| < \varepsilon = 0.001$.

First thing to do is to input our function f and its derivative ∇ that we name as "nabla"

```
function [f] = f(x)
x1=x(1,1);
x2=x(1,2);
    f= 1+2*x1*exp(-x1^2-x2^2);

end
```

```
function [nabla] = nabla(x)
x1=x(1,1);
x2=x(1,2);
nabla = [2*exp(-x1^2-x2^2)*(1-2*x1^2) -4*x1*x2*exp(-x1^2-x2^2)];

end
```

Here is the Matlab script to solve our problem:

```
m=0;                              %number of iterations
eps=0.001;
x=[0.5 -0.5];
x1=x(1,1);
x2=x(1,2);
N=nabla(x);
nabla1=N(1,1);
nabla2=N(1,2);

a = [-2:0.1:2];
b = a;
for (i = 1:length(a))
for (j = 1:length(b))
y(i,j) = f([a(i),b(j)]);
end
end
surf(a,b,y);
hold on;
figure(1);
while (sqrt(nabla1^2+nabla2^2)>eps)
  P=[4*nabla1*(nabla1^2+nabla2^2) -
4*(x1*(nabla1^2+nabla2^2)+nabla1*(nabla1*x1+nabla2*x2)) -
2*nabla1+4*x1*(nabla1*x1+nabla2*x2)];
alpha=roots(P);
if f(x-alpha(1)*N)<f(x-alpha(2)*N)
      x=x-alpha(1)*N;
else
      x=x-alpha(2)*N;
end
```

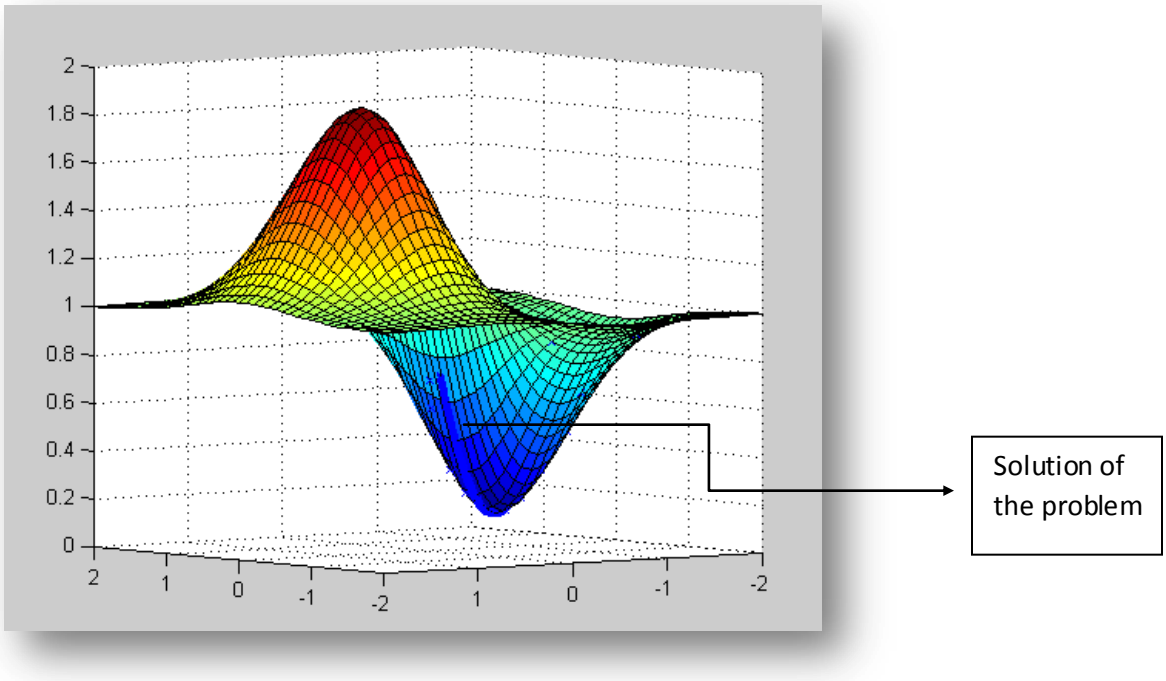
We solved the problem using three initial conditions : $[1\ 0]^T$, $[0.5 - 0.5]^T$ and $[-0.5\ 0.5]^T$.

For the first vector we found $[-0{,}7071\ 0]^T$

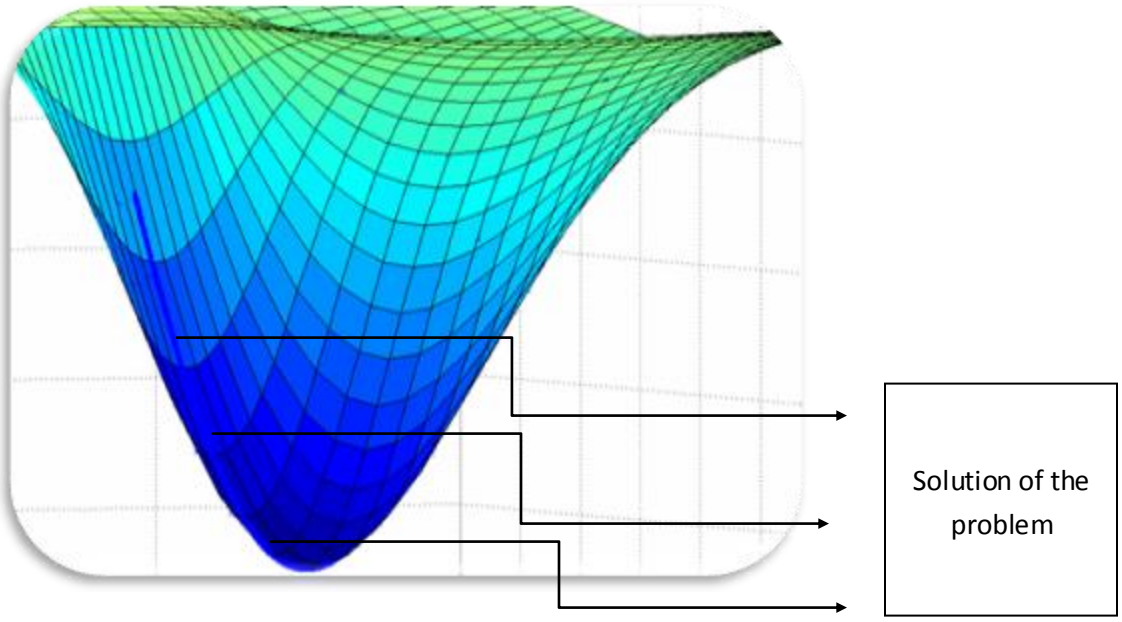
For the Second vector : $[-0.7074\ 0]^T$

For the third vector we were not able to find a result because of the speed of the program. We can understand how important is the initial condition .

After compiling the program, we found as a result:



If we zoom in on the lower part of the graph, we clearly see the solution of our problem



3.2 Rosenbrock's function

3.2.1 Analytical study

We can use the same method like 1.4. In fact , we have here the sum of two positive functions. Then we have :

$$x_2 - {x_1}^2 = 0 \ ; 1 - x_1 = 0$$

Then : $x_1 = 1$ and $x_2 = {x_1}^2$ and then $x_1 = x_2 = 1$.

There is a unique solution .

Thanks to that, we can affirm that $[1\ 1]^T$ is the unique global minimizer of Rosenbrock's function.

3.2.2 Implement of the Matlab code

First thing to do is to input our function f and its derivative

```
function [f32] = f32(x)
x1=x(1,1);
x2=x(1,2);
    f= 100*(x2-x1^2)^2 + (1-x1)^2;
end
```

```
function [nabla2] = nabla2(x)
x1=x(1,1);
x2=x(1,2);
   nabla2 = [-400*x1*(x2-x1^2)-2*(x1*(1-x1)) 200*(x2-x1^2)];
end
```

```
function [G] = Gp(alpha,Y,X)

    x1=X(1,1);
    x2=X(1,2);
    y1=Y(1,1);
    y2=Y(1,2);
    G= 200*((x2-alpha*y2)-(x1-alpha*y1)^2)*(-y2+2*y1*(x1-
alpha*y1))+2*y1*(1-x1+y1*alpha);

end


function [GG] = Gpp(alpha,Y,X)

    x1=X(1,1);
    x2=X(1,2);
    y1=Y(1,1);
    y2=Y(1,2);
    GG= 200*((-y2+2*y1*(x1-alpha*y1))*(-y2+2*y1*(x1-alpha*y1))+((x2-
alpha*y2)-(x1-alpha*y1)^2)*(-2*y1^2))+2*y1^2;

end

function [nabla3] = nabla3(x)

   x1=x(1,1);
   x2=x(1,2);
   nabla3= [(-400*x1*(x2-(x1)^2)+2*(x1-1)) 200*(x2-x1^2)];

end
```

Gp and Gpp are the functions used to find the best alpha value to use between each iteration of xk. Gp and Gpp use alpha, the gradient of Rosenbrock's function at the iteration xk and xk.

Here is the Matlab script to solve our problem:

The result found is not satisfying. In fact, we found

```
clear all;close all;clc;
x0=[-1 2];
eps1=0.001;
eps2=0.001;
alpha0=0.1;

alpha1=alpha0-Gp(alpha0,nabla3(x0),x0)/Gpp(alpha0,nabla3(x0),x0);

while (abs(alpha1-alpha0)/alpha0)>eps2
    alpha0=alpha1;
    alpha1=alpha1-Gp(alpha1,nabla3(x0),x0)/Gpp(alpha1,nabla3(x0),x0);

end

x1=x0-alpha1*nabla3(x0);

% calcul de x1 pour le début de l'algorithme final

m=0;
while (norm(x1-x0)/norm(x0))>eps1
    alpha0=0.1;
    alpha1=alpha0-Gp(alpha0,nabla3(x1),x1)/Gpp(alpha0,nabla3(x1),x1);


    while norm(alpha1-alpha0)/(alpha0)>eps2
        alpha0=alpha1;
        alpha1=alpha1-Gp(alpha1,nabla3(x1),x1)/Gpp(alpha1,nabla3(x1),x1);

    end

    x0=x1;
    x1=x1-alpha1*nabla3(x1);
```

We need to calculate x_1 if we want to use the stopping criterion.

To conclude, this practical was very interesting because it presented all the different parts of the lesson. Furthermore, thanks to that, we can take more hindsight on the domain. Most of the important functions were studied using matlab which helped to understand the importance of the parameters like initial conditions.