

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Polytechnique de Tunisie



Rapport du projet d'Optimisation

26/05/2018

Réalisé par :

ELHAMDI Kaouther

HOUSSEINI Anis

BEN RHOUMA Oussema

A.U : 2017/2018

L'Objectif du Projet :

L'objectif de ce projet est d'appliquer les différentes méthodes d'optimisation pour résoudre des problèmes de régression et de voir quelle méthode est plus adaptée pour résoudre un problème donné.

Langage de codage utilisé : MATLAB

Sommaire

- 1. Optimisation sans contrainte : La régression linéaire simple**
 - 1.1 Étude du problème
 - 1.2 Algorithmes de résolution
- 2. Optimisation avec contrainte : la régression linéaire multiple**
 - 2.1 Étude du problème
 - 2.2 Méthode de pénalisation
- 3. Méthode de gradient accélères**
 - 3.1 Système du premier ordre

1. Optimisation sans contrainte : La régression linéaire simple

1.1 Étude du problème :

- 1) La fonction d'erreur dépend des variables a_0 et a_1
- 2) L'Erreur peut s'écrire sous la forme

$$E(a) = \|Ma - m\|_2^2$$

$$\text{Avec } M = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} ; a = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} ; m = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

3)

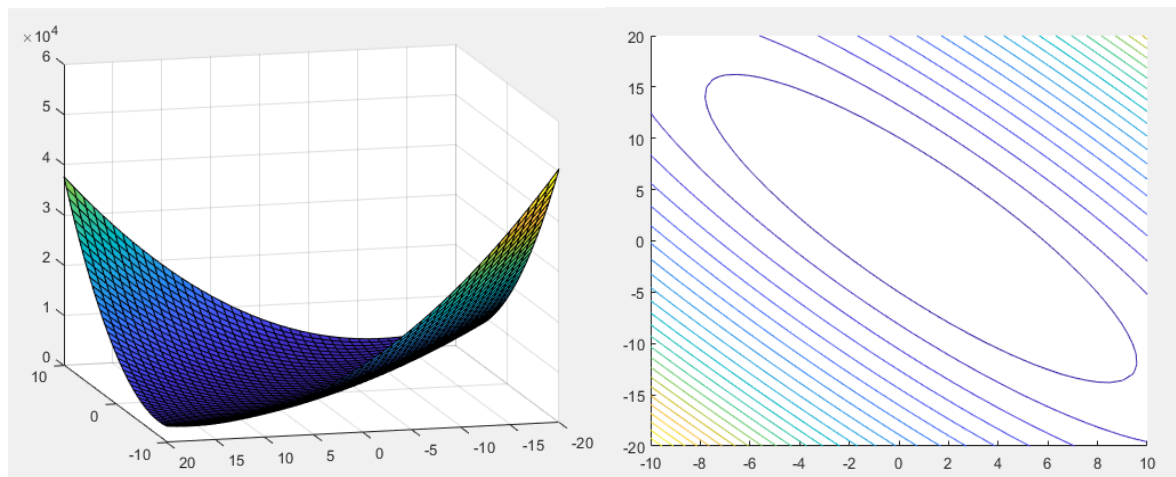
a)

```
function nrm=Erreur(u,v)
load("partiel");
M = [ones(size(x)) x];
m=y;
a=[u;v];
nrm=norm(M*a-m,2)^2;
end
```

b)

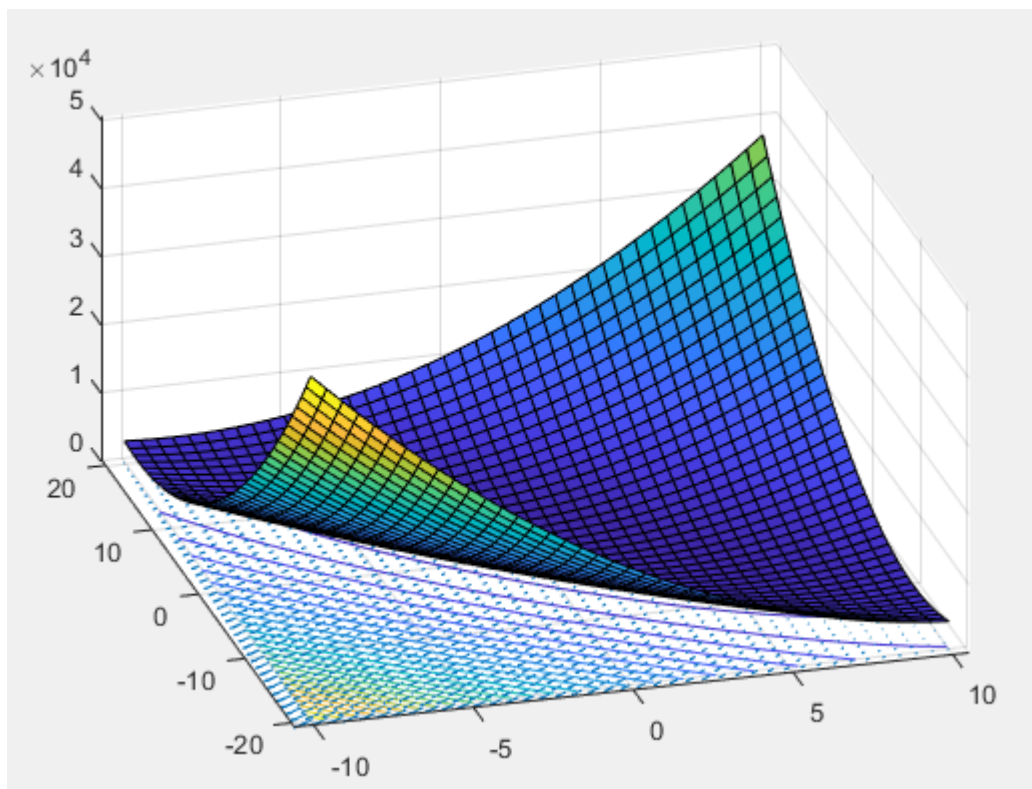
```
load("partiel");
M = [ones(size(x)) x];
m=y;
a0 = -10:0.5:10; %entre -10 et 10 de pas 0.5
a1 = -20:1:20; %entre -20 et 20 de pas 1
[X,Y] = meshgrid(a0,a1);
e = arrayfun(@Erreur, X, Y);
surf(X,Y,e)
hold on;
contour(X,Y,e,25);
hold on;
[gx,gy]=gradient(e);
quiver(X,Y,gx,gy);
```

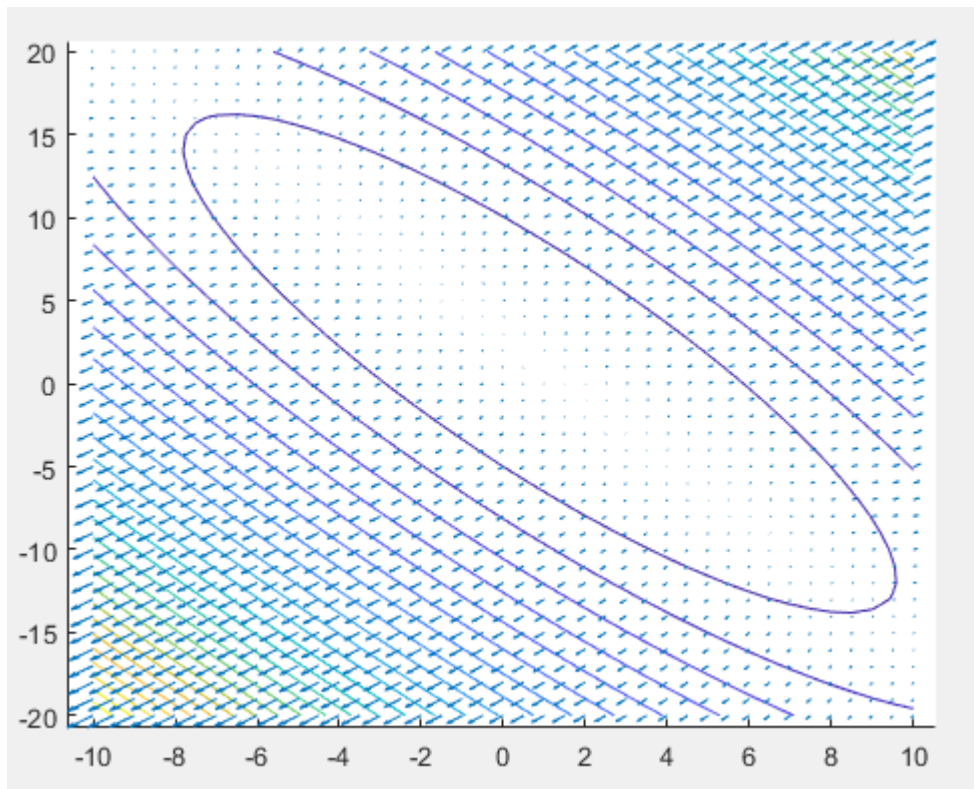
L'affichage des courbes :



L'Erreur en 3D Contour de l'Erreur

c)





**Champ du gradient de la fonction d'erreur superposé
avec le contour**

On voit clairement d'après le graphe que le gradient est nul au voisinage du centre de graphe de contours ce qui est justifié car le centre du graphe est le minimum de la fonction d'Erreur ou le gradient s'annule dans ce point.

4)

On a $E(a) = \langle Ma - m, Ma - m \rangle = \langle Ma, Ma \rangle - 2 \langle Ma, m \rangle + \langle m, m \rangle$

Donc minimiser $E(a)$ revient à minimiser

$$F(a) = \frac{1}{2} \langle Ma, Ma \rangle - \langle Ma, m \rangle$$

Car $\langle m, m \rangle$ est une constante

Or $F(a) = \frac{1}{2} \langle M^T M a, a \rangle - \langle M^T m, a \rangle$ D'où $A = M^T M$ et $b = M^T m$

5)

$$\nabla F(a) = Aa - b$$

$$\nabla^2 F(a) = A$$

6)

Le minimum de la fonction F vérifie $\nabla F(a) = 0$

Càd $Aa - b = 0$

Si A est inversible alors F est minimale en $a = A^{-1}b$

1.2 Algorithmes de résolution :

1)

```
%L'implémentation de la fonction F      %L'implémentation de la fonction gradient F
function t=F(M,m,x,y)                  function t=gradF(M,m,x,y)
a=[x;y];
A=transpose(M)*M;
b=transpose(M)*m;
t=0.5*(sum((A*a).*a)-sum((b.*a)));
end

%L'implémentation de la fonction gradient Carre de F
function t=gradCarreF(M)
t=transpose(M)*M;
end
```

a) La méthode de Newton :

```
%La méthode de résolution de Newton
function [u,iter]=newton(M,m,x0,tol)
    grad=1;
    u=x0;
    iter=1;
    tableX=[];
    tableY=[];
    tableX(iter)=u(1);
    tableY(iter)=u(2);
    while (grad>tol && iter<100)
        step=gradCarreF(M)\gradF(M,m,u(1),u(2));
        grad=norm(step);
        u=u-step;
        iter=iter+1;
        tableX(iter)=u(1); tableY(iter)=u(2);
    end
    a0 = -1:0.1:3;
    a1 = -1:0.1:3;
    [X,Y] = meshgrid(a0,a1);
    e = arrayfun(@Erreur, X, Y);
    contour(X,Y,e,25);
    hold on;
    plot(tableX,tableY,'--*'); %affichage de la trajectoire ( convergence vers la solution )
end
```

Test de méthode de Newton avec 0.01 de tol et [0 ; 1] point initial

```
>> [u,iter]=newton(M,y,[0;1],0.01)
```

```
u =
```

```
    0.8844
    1.2083
```

```
iter =
```

```
    3
```

b) La méthode du gradient à pas constant :

```
%La méthode de résolution du gradient à pas constant
function [A,iter]=gradientpasfixe(M,m,x,y,ro,epsilon)
A=[x;y];
iter=1;
tableX=[];
tableY=[];
tableX(iter)=A(1);
tableY(iter)=A(2);
while (true)
    x0=A(1);
    y0=A(2);
    A=[x0;y0]-ro*gradF(M,m,x0,y0);
    Z=[A(1)-x0;A(2)-y0];
    iter=iter+1;
    tableX(iter)=A(1); tableY(iter)=A(2);
    if norm(Z,1)<epsilon
        break
    end
end
a0 = -1:0.1:3;
a1 = -1:0.1:3;
[X,Y] = meshgrid(a0,a1);
e = arrayfun(@Erreur, X, Y);
contour(X,Y,e,25);
hold on;
plot(tableX,tableY,'--*'); %affichage de la trajectoire ( convergence vers la solution )
end
```

Test de méthode de Gradient à pas constant avec 0.00001 de tolet[0 ; 1] point initial

```
>> [A,iter]=gradientpasfixe(M,m,0,1,0.01,0.00001)
```

```
A =
```

```
    0.8843
    1.2084
```

```
iter =
```

```
    112
```


c) La méthode du gradient à pas optimal :

```
%La méthode de résolution du gradient à pas optimal
function [a1,nb]= gradientapasoptimal(epsilon,x,y)
H=[0 0;0 0];
xi=0;
yi=0;
xyi=0;
xi2=0;
i=1;
while (i<102)
    xi=xi+x(i);
    yi=yi+y(i);
    xyi=xyi+x(i)*y(i);
    xi2=xi2+x(i)^2;
    i=i+1;
end
H(1,1)=101*2;
H(1,2)=xi*2;
H(2,1)=xi*2;
H(2,2)=xi2*2;
b=zeros(2,1);
b(1)=2*yi;
b(2)=2*xyi;
a=[1;1];
r=H*a-b;
l=H*r;
p=norm(r,2)/(l(1)*r(1)+l(2)*r(2));
a1=a-p*r;
nb=1;
X=[a,a1];
tableX=[];
tableY=[];
tableX(nb)=a(1);
tableY(nb)=a(2);
while norm(a-a1,2)>epsilon
    a=a1;
    r=H*a-b;
    l=H*r;
    p=(norm(r,2)^2)/(l(1)*r(1)+l(2)*r(2));
    a1=a-p*r;
    nb=nb+1;
    X=[X,a1];
    tableX(nb)=a(1); tableY(nb)=a(2);
end
plot(tableX,tableY,'--*'); %affichage de la trajectoire ( convergence vers la solution )
end
```

Test de méthode de Gradient à pas optimal avec 0.000001 de tolér [0 ; 0] point initial

```
>> [a1,nb]= gradientapasoptimal(0.000001,x,y)
```

```
a1 =
```

```
0.8844  
1.2083
```

```
nb =
```

```
65
```

d) La méthode du gradient conjugué :

```
%La méthode de résolution du gradient conjugué  
function [a,nb] = gradientconjugué(epsilon ,x,y)  
H=[0 0;0 0];  
xi=0;  
yi=0;  
xyi=0;  
xi2=0;  
i=1;  
while(i<102)  
    xi=xi+x(i);  
    yi=yi+y(i);  
    xyi=xyi+x(i)*y(i);  
    xi2=xi2+x(i)^2;  
    i=i+1;  
end  
H(1,1)=101*2;  
H(1,2)=xi*2;  
H(2,1)=xi*2;  
H(2,2)=xi2*2;  
b=zeros(2,1);  
b(1)=2*yi;  
b(2)=2*xyi;  
a=[70;9];  
r=H*a-b;  
d=r;  
l=H*r;  
p=(norm(r,2)^2)/(l(1)*d(1)+l(2)*d(2));
```

```

al=a-p*d;
nb=1;
X=[a,al];
tableX=[];
tableY=[];
tableX(nb)=a(1);
tableY(nb)=a(2);
while norm(a-al)>epsilon
    a=al;
    r1=r;
    r=H*a-b;
    d=r + norm(r)^2/norm(r1)^2 * d;
    l=H*r;
    p=(norm(r,2)^2)/(l(1)*d(1)+l(2)*d(2));
    al=a-p*d;
    nb=nb+1;
    X=[X,al];
    tableX(nb)=a(1); tableY(nb)=a(2);
end
plot(tableX,tableY,'--*'); %affichage de la trajectoire ( convergence vers la solution )
end

```

Test de méthode de Gradient à pas optimal avec 0.000001 de tolet [0 ; 0] point initial

```
>> [al,nb]= gradientconjugue(0.000001,x,y)
```

```
al =
```

```

    0.8844
    1.2083

```

```
nb =
```

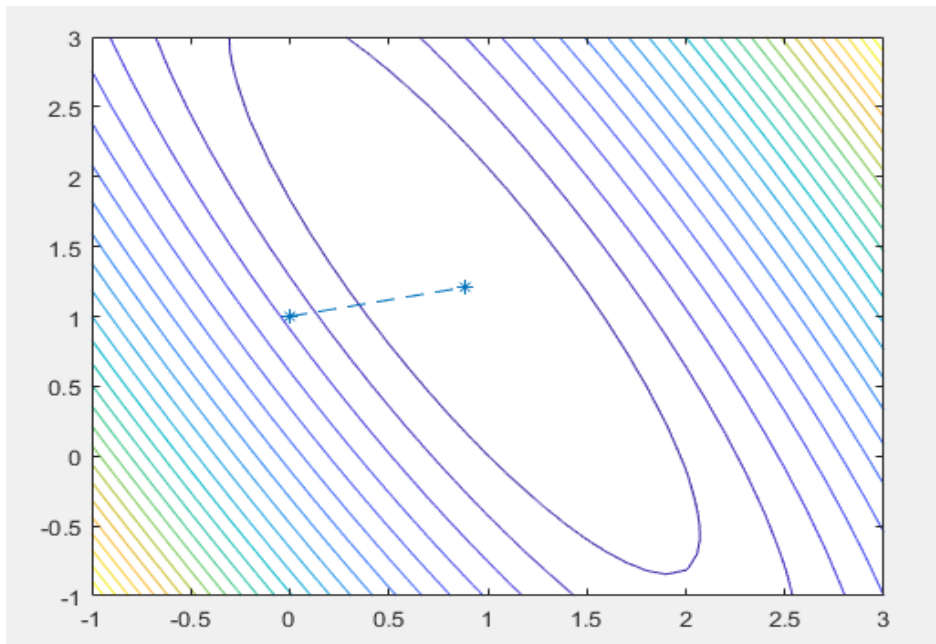
```

    3

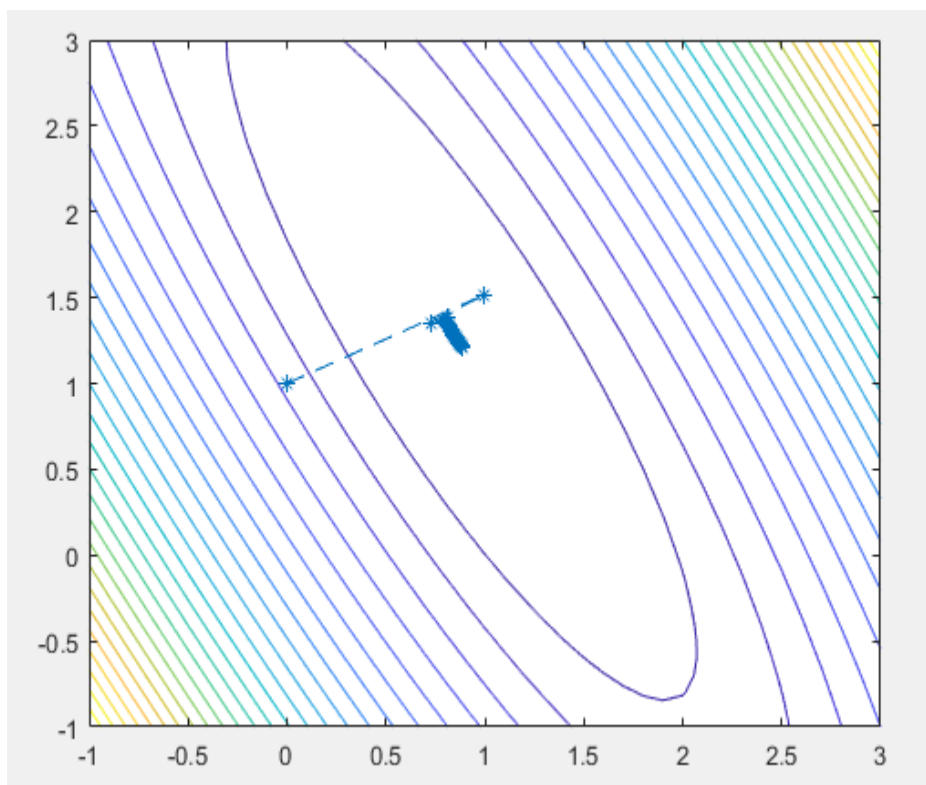
```

2)

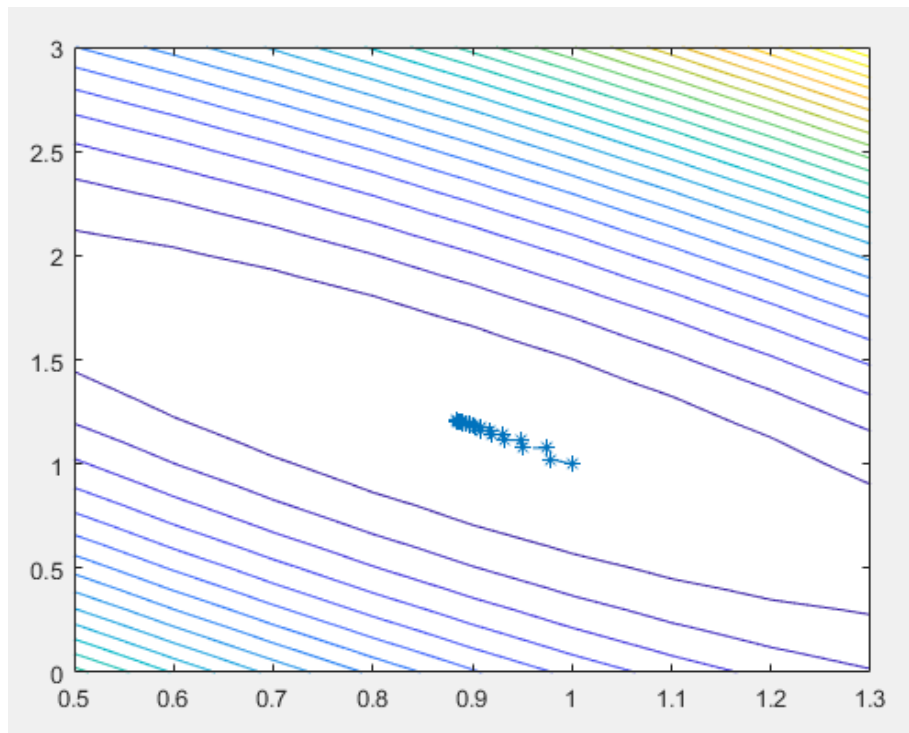
a) Comparaison des trajectoires des itérations de ces méthodes de résolution



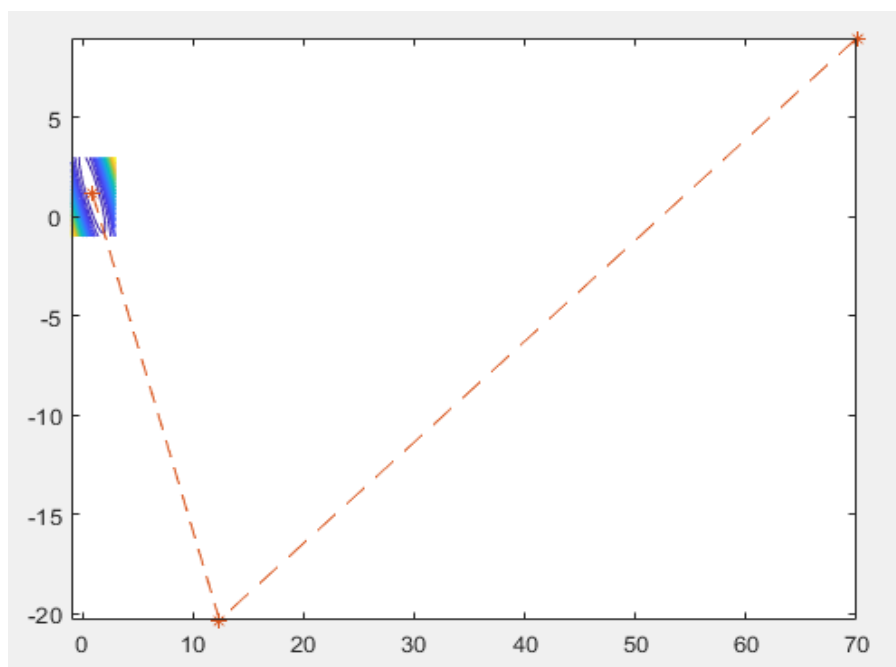
Trajectoire des itérations de méthode de
Newton pour $\text{tol}=0.01$ et $[0 ; 1]$ point de départ



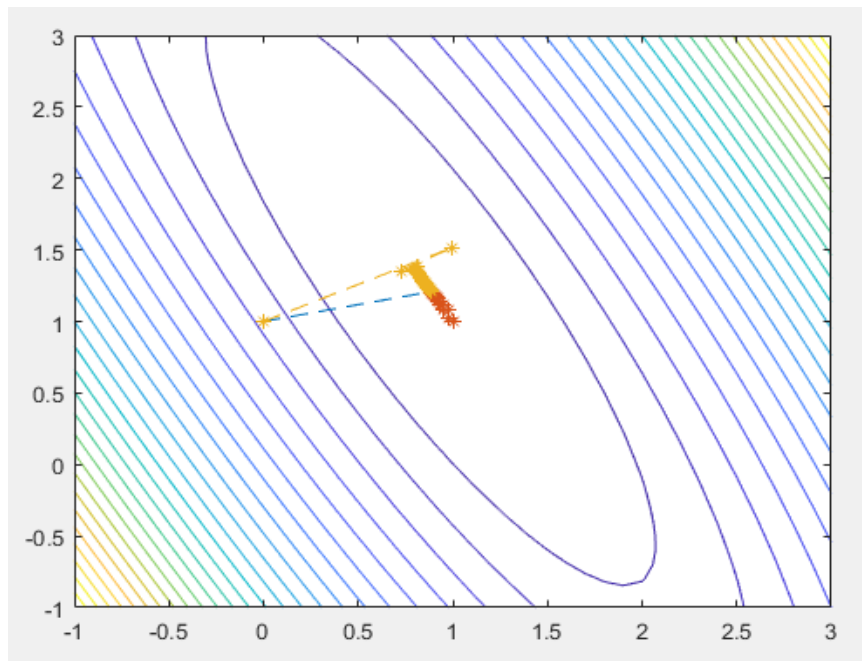
Trajectoire des itérations de méthode de
Gradient à pas constant pour $\text{tol}=0.01$ et
 $[0; 1]$ point de départ et $\text{ro}=0.01$



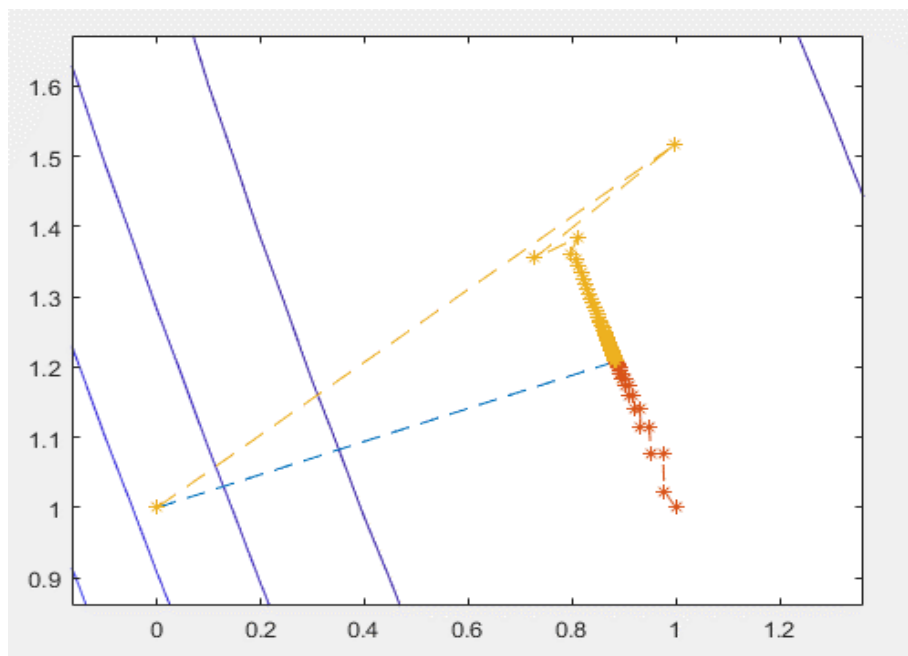
Trajectoire des itérations de méthode de
Gradient à pas optimal pour une $\text{tol}=0.000001$
et $[1 ; 1]$ point de départ



Trajectoire des itérations de méthode de
Gradient conjugué pour $\text{tol}=0.000001$
et $[1 ; 1]$ point de départ



Superposition des trajectoires



Après un Zoom

• Newton

• Gradient à pas optimal

• Gradient à pas fixe

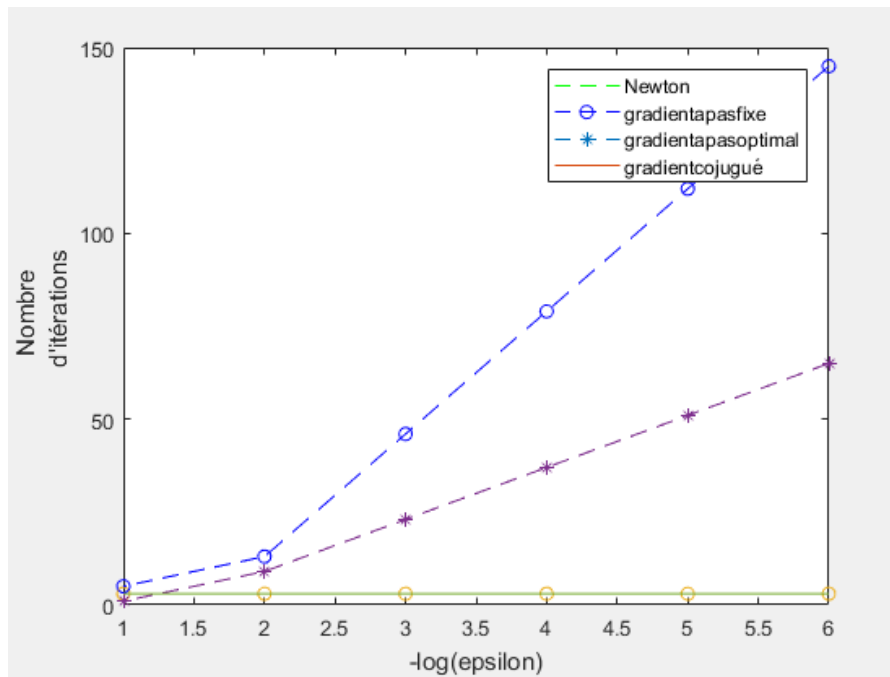
Donc après avoir implémenté ces algorithmes et les tester on remarque que celle de Newton converge plus rapidement que les autres.

b)

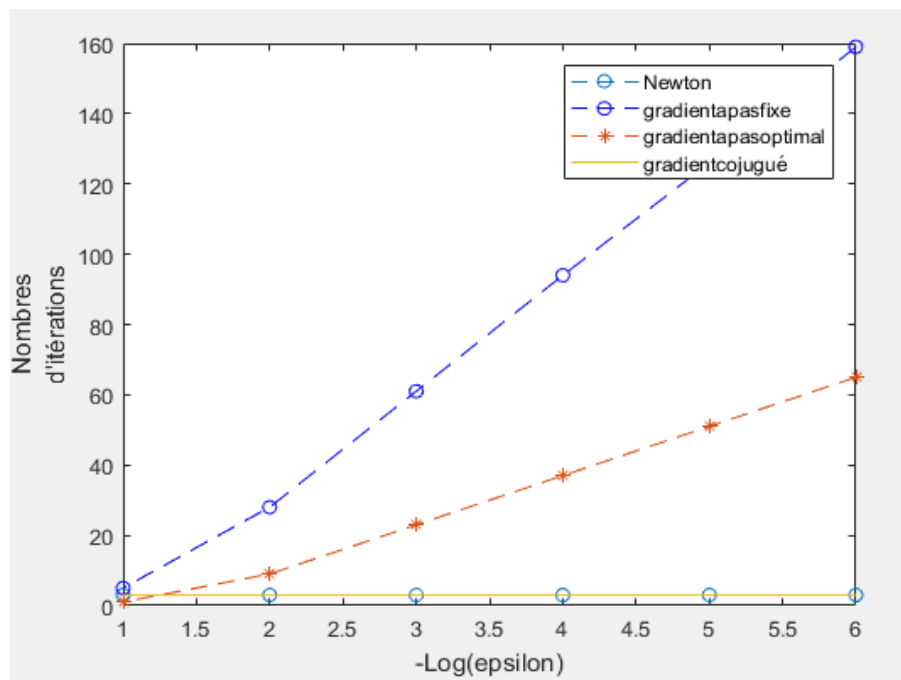
```
load("partiel");
M = [ones(size(x)) x];
epsilon=[0.000001 0.00001 0.0001 0.001 0.01 0.1];
table1=[];
table2=[];
table3=[];
table4=[];
for i=1:6
    [a,iter1]=newton(M,y,[0;1],epsilon(i));
    table1(i)=iter1;
    [b,iter2]=gradientpasfixe(M,m,0,1,0.01,epsilon(i));
    table2(i)=iter2;
    [c,iter3]=gradientapasoptimal(epsilon(i),x,y);
    table3(i)=iter3;
    [d,iter4]=gradientconjugue(epsilon(i),x,y);
    table4(i)=iter4;
end
plot(-log(epsilon)/log(10),table1,'--o');
hold on;
plot(-log(epsilon)/log(10),table2,'b--o')
hold on;
plot(-log(epsilon)/log(10),table3,'--*')
hold on ;
plot(-log(epsilon)/log(10),table4)
legend('Newton','gradientapasfixe','gradientapasoptimal','gradientcojugué')
```

Pour visualiser les courbes d'itérations il faut mettre en commentaires les commandes liées à plot dans les fonctions Newton, gradient à pas fixe, gradient à pas optimal et gradient conjugué dans le programme afin d'éviter les duplications des courbes d'itérations avec celle des trajectoires.

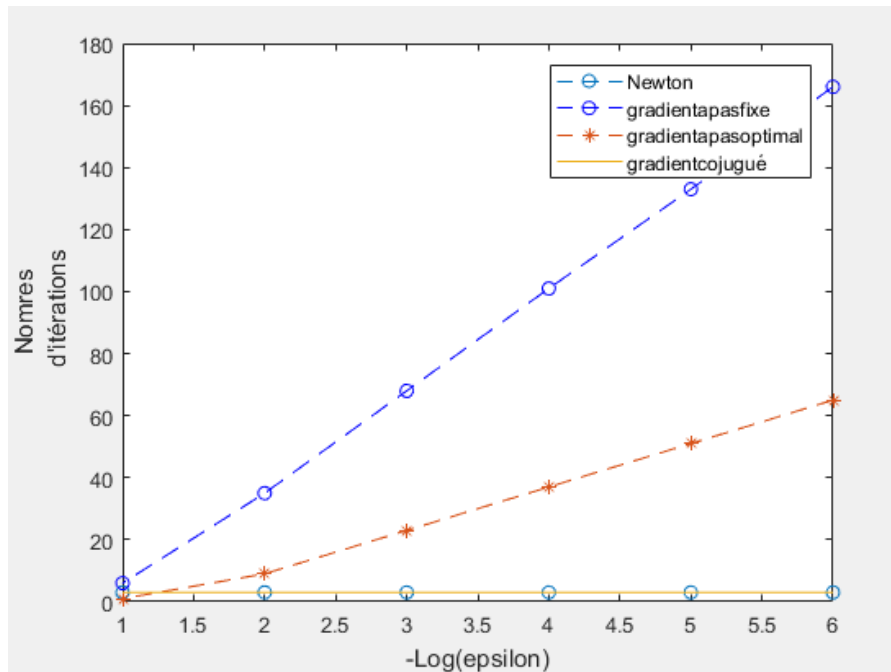
1. Pour $[0 ; 1]$ point de départ



2. Pour $[0 ; 0]$ point de départ



3. Pour $[-1 ; -1]$ point de départ



c)

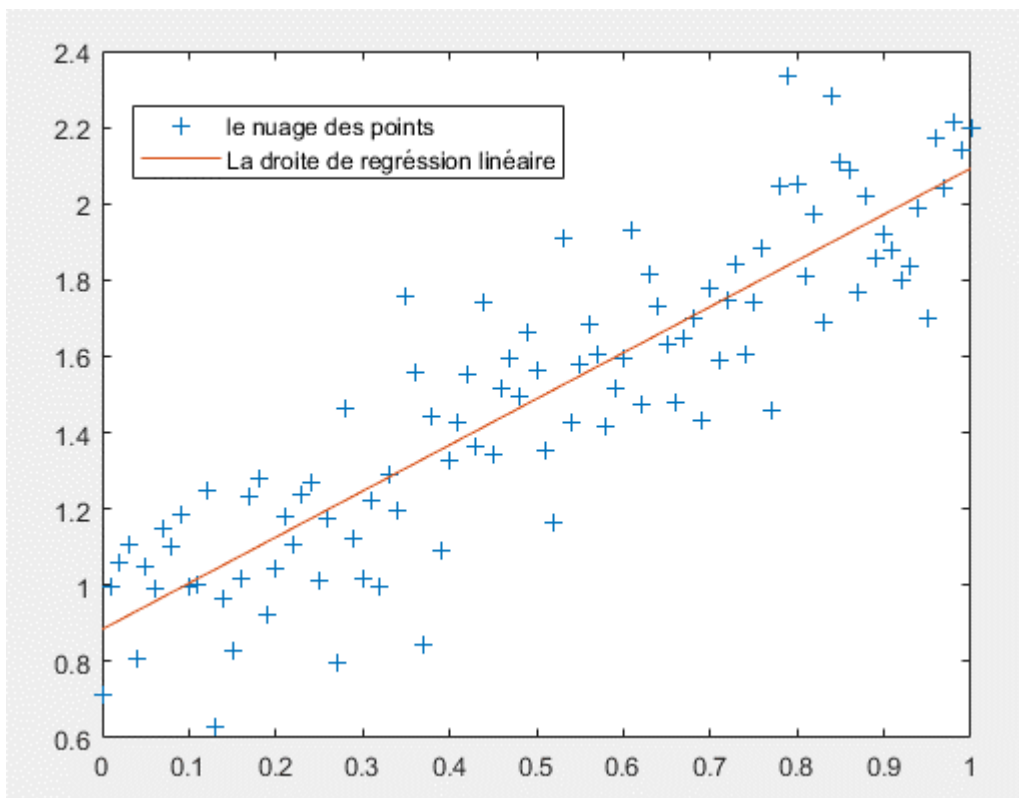
D'après les courbes d'itérations en fonction de la précision

On remarque que le nombre d'itérations de la méthode de Newton et de la méthode du gradient conjugué ne dépendent pas de ϵ et ne dépendent pas aussi du point de départ, il est toujours constant et égale à 3 par contre les nombres d'itérations des autres méthodes de résolution (gradient à pas constant, gradient à pas optimal) dépendent de ϵ et du point de départ

Donc pour notre problème on va utiliser la méthode de résolution de Newton parce qu'elle est la plus rapide en convergence et elle ne dépend pas de ϵ et du point de départ.

3)

```
M = [ones(size(x)) x];  
theta = newton(M,m,[0;1],0.001);  
yfit = M * theta;  
plot(x, y, '+'); hold on;  
plot(x, yfit, '-');  
legend('le nuage des points','La droite de régression linéaire')
```



Graphe du nuage des points
et la droite de régression linéaire

2. Optimisation avec contrainte : la régression linéaire multiple

2.1 Étude du problème

1) On a $\hat{y}[i] = \sum_1^n a[j] * X[i, j]$

Donc : $\hat{y} = Xa$

2) $E(a) = \sum_1^n (y_i - \hat{y}_i)^2 = \|y - \hat{y}\|^2 = \|y - Xa\|^2$

On a : $E(a) = \|y - Xa\|^2 = f_2 \circ f_1(a)$

Avec : $f_1(a) = (Xa - y, Xa - y) = (Xa, Xa) - (y, y)$

$f_2(x, y) = \langle x, y \rangle$

- Mq la fonction $E : a \rightarrow E(a)$ est différentiable :

On a f_1 est différentiable (comme étant le couple de 2 fonctions différentiables) et on a : $Df_1(a)(h) = (Xh, Xh)$

f_2 bilinéaire et continue (Cauchy Schwarz) donc différentiable et $Df_2(x, y)(h, k) = \langle x, k \rangle + \langle h, y \rangle$

D'où E est différentiable

- $\text{grad}E(a) :$

On a :

$$DE(a)(h) = Df_2(f_1(a)) \circ Df_1(a)(h) = Df_2((Xa - y, Xa - y))(Xh, Xh) =$$

$$2\langle Xa - y, h \rangle = 2\langle X^+(Xa - y), h \rangle = \langle \text{grad}E(a), h \rangle$$

D'où $\text{grad}E(a) = -2X^+y + 2X^+Xa$

- $\text{hess}E(a) :$

$$\text{hessE}(a) = \text{grad}^2 E(a) = 2X^+X$$

- Mq E est convexe :

Pour tout a , $\text{hessE}(a) = 2X^+X$ est symétrique .

$\text{Det}(\text{hessE}(a)) \geq 0$ et $\text{hessE}(a)[0,0] \geq 0$ donc $\text{hessE}(a)$ est semi-définie positive . D'où E est convexe .

$$3) \text{gradE}(a) = 0 \text{ éq à } -2X^+y + 2X^+Xa = 0$$

Si X^+X est inversible la solution du problème de minimisation de E est :

$$a = H \text{ où } H = (\text{inv}(X^+X)) * X^+$$

4)

```
>> load partie2
>> who

Your variables are:

X  a  y

>> a = inv(transpose(X)*X)*(transpose(X))*y
Warning: Matrix is singular to working precision.

a =

    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
    NaN
```

On remarque que le système diverge car la matrice X^+X n'est pas inversible .

2.2 Méthode de pénalisation :

1) Comme on a déjà vu à la partie précédente $X^T X$ n'était pas inversible ce qui cause la divergence du solution. D'où la nécessité d'une nouvelle méthode.

En fait , la méthode de pénalisation cherche à trouver des solution toute en assurant la non divergence .

● On a : $\text{grad}E(a) = -2X^T y + 2X^T X a + 2\lambda a$

Et $\text{hess}E(a) = 2X^T X + 2\lambda I_n$ (avec I_n est la matrice d'identité)

On a : $X^T X + \lambda I_n$ est toujours inversible d'où la solution

$a = (\text{Inv}(X^T X + \lambda I_n))^T (X^T)^T y$ existe .

2) A ce niveau, on va choisir la méthode de newton qui est la mieux adaptée pour résoudre le problème.

```
gradE.m  x  hessE.m  x  newton2.m  x  va
1  function z=gradE(X,y,a,lambda)
2  -   n=size(X'*X);
3  -   z=-2*X'*y+2*X'*X*a+2*lambda*a;
4  -   end
```

Fonction qui calcul le gradient de E en a.

La méthode `gradE` calcul le gradient du fonction E où ce gradient égal à :
 $\text{grad}E(a) = -2X^T y + 2X^T X a + 2\lambda a$

```
+4  grad2E.m  x  hessE.m  x  newton2.m  x  gradp
1  function z=hessE(X,lambda)
2  -   n=size(X'*X);
3  -   z=2*transpose(X)*X+2*lambda*eye(n);
4  -   end
```

Fonction qui calcul la Hessienne de E

La méthode `hessE` calcul la hessienne du fonction E où ce gradient égal à :
 $\text{hess}E(a) = 2X^T X + 2\lambda I_n$

```

1 function [z,i]=newton2(X,y,a0,lambda,eps)
2     H=hessE(X,lambda);
3     a=a0-(inv(H))*gradE(X,y,a0,lambda);
4     i=0;
5     disp(000000);
6     while (norm((a-a0),2)>eps)
7         a0=a;
8         a=a0-(inv(H))*gradE(X,y,a0,lambda);
9         i=i+1;
10        disp(55555);
11    end
12    z=a;
13 end

```

Ici z va contenir le vecteur a (contenant les paramètres)
Et i représente le nombre d'itérations

Fonction de newton pour la fonction E

Ici c'est l'exécution

```

Command Window
New to MATLAB? See resources for Getting Started.

>> load('partie2.mat')
>> [a,n]=newton2(X,y,[1;2;3;6;7;2;8],20,10^(-15))
0

55555

55555

a =

1.3163
1.5327
1.5327
0.4736
0.1685
0.6102
0.6102

n =

2

```

Exécution

On remarque que le programme ne converge plus, donnant le vecteur **a** après 2 itérations.

3)

```

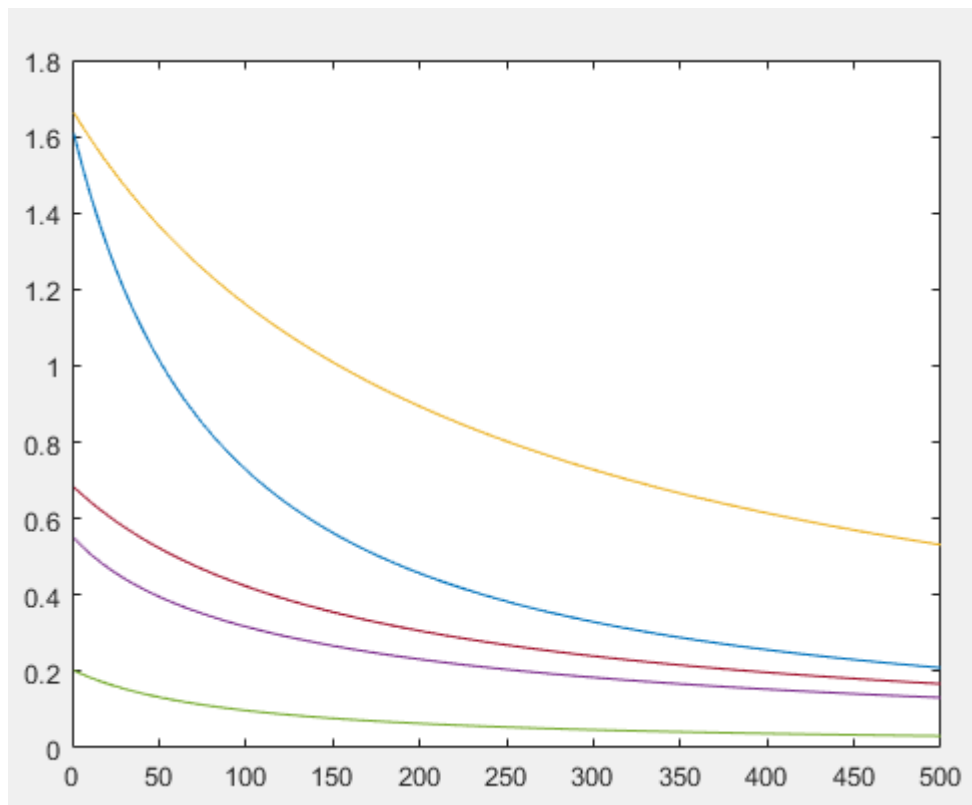
gradE.m x hessE.m x newton2.m x var_coeff.m x estim.m x
function var_coeff(X,y)
- a0=[1;2;3;4;5;6;7];
- i=1;
- while(i<= (size(X,2)) )
-     v=[];
-     lambdas=linspace(1,500);
-     for j = 1:length(lambdas)
-         [a,n]=newton2(X,y,a0,lambdas(j),10^(-15));
-         v=[v,a(i)];
-     end
-     plot(lambdas,v);
-     hold on ;
-     i=i+1;
- end
end

```

Appel au fonction
var coeff

`fx >> var_coeff(X,y)`

Figure obtenue



Traçage de la variation des valeurs des coefficients en fonction de λ .

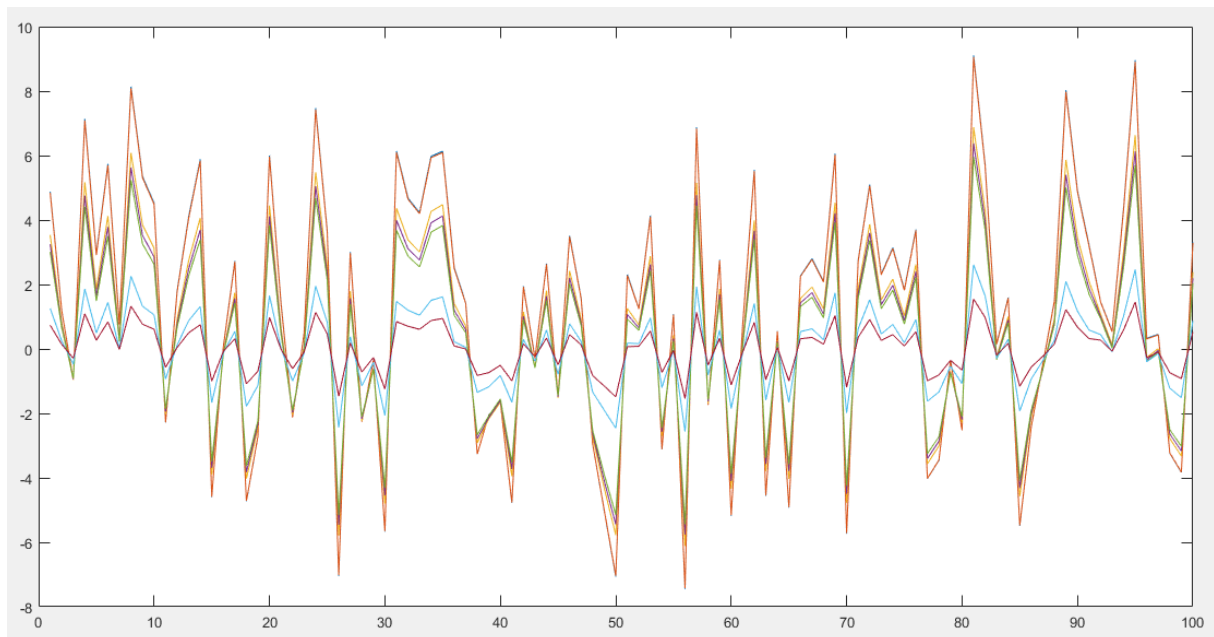
On remarque que les valeurs des coefficient ($a[i]$) diminuent lorsque λ (lambda) augmente tout en tendant vers des valeurs limites bien définies.

4)

Dans la partie commande :

Après exécution

```
gradE.m x hessE.m x newton2.m x var_coeff.m x estim.m x +
1 - load('partie2.mat')
2 - lambda=[1,60,80,100,500,1000];
3 - k=linspace(1,100);
4 - plot(k,y);
5 - hold on;
6 - for i = 1:6
7 -     [a,n]=newton2(X,y,[1;2;3;4;5;6;7],lambda(i),10^(-15));
8 -     yopt=X*a;
9 -     plot(k,yopt);
10 -    hold on ;
11 - end
```



3. Méthode de gradient accélérés

1)

$$E = \log\left(\sum_{i=1}^n (y_i - \hat{y}_i)^2\right)$$

$$\hat{y}_i = a(1 - \exp(bt_i))$$

$$S = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

D'où

$$\nabla E(a, b) = \frac{1}{S} \begin{pmatrix} \sum_{i=1}^n -2(1 - \exp(bt_i))(y_i - \hat{y}_i) \\ \sum_{i=0}^n 2at_i \exp(bt_i)(y_i - \hat{y}_i) \end{pmatrix}$$

2) Avec t et y deux listes données, et a et b deux points quelconques.

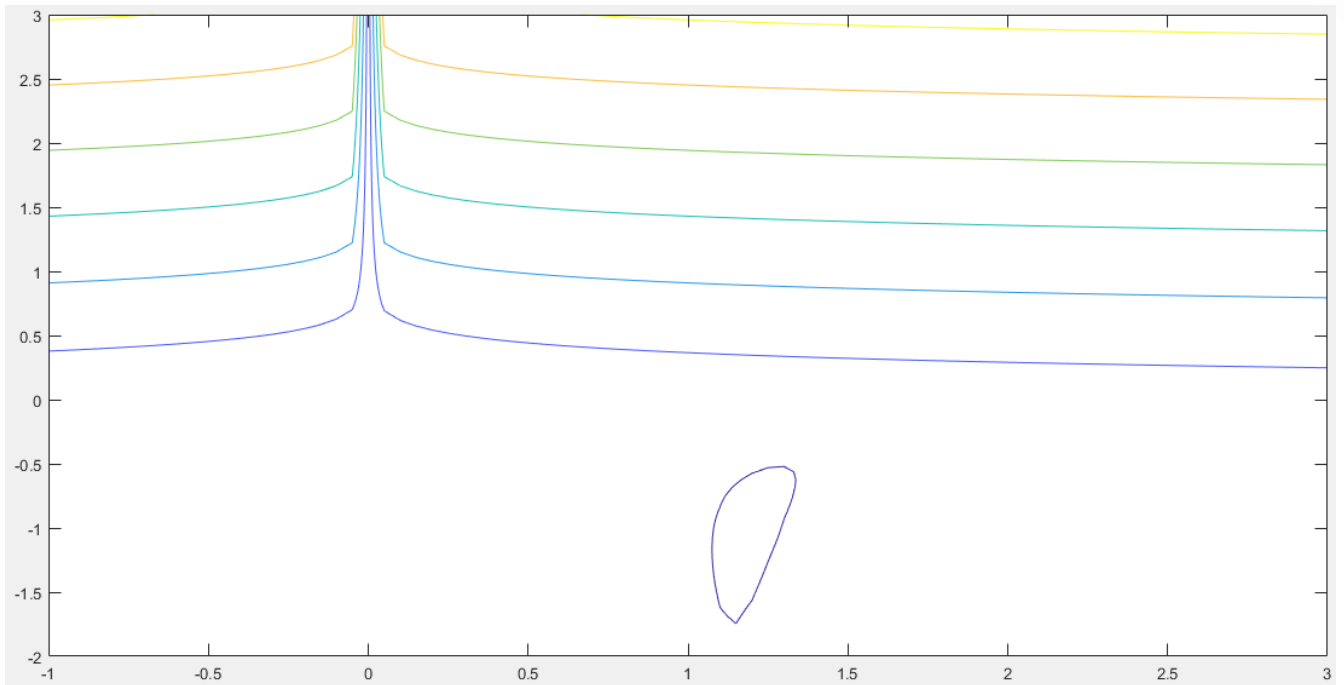
```
function E=erreur(t,y,a,b)
S=0;
for i=1:1:length(y)
    S=S+(y(i)-a*(1-
exp(b*t(i))))^2);
end
E=log(S)
```

```
function DE=grad_erreur(t,y,a,b)
S1=0;
S2=0;
S=0;
for i=1:1:length(y)
    S=S+(y(i)-a*(1-exp(b*t(i))))^2);
    S1=S1+(-2)*(1-exp(b*t(i)))*(y(i)-a*(1-exp(b*t(i))));
    S2=S2+2*a*t(i)*exp(b*t(i))*(y(i)-a*(1-exp(b*t(i))));
end
DE=[(S1/S);(S2/S)];
```

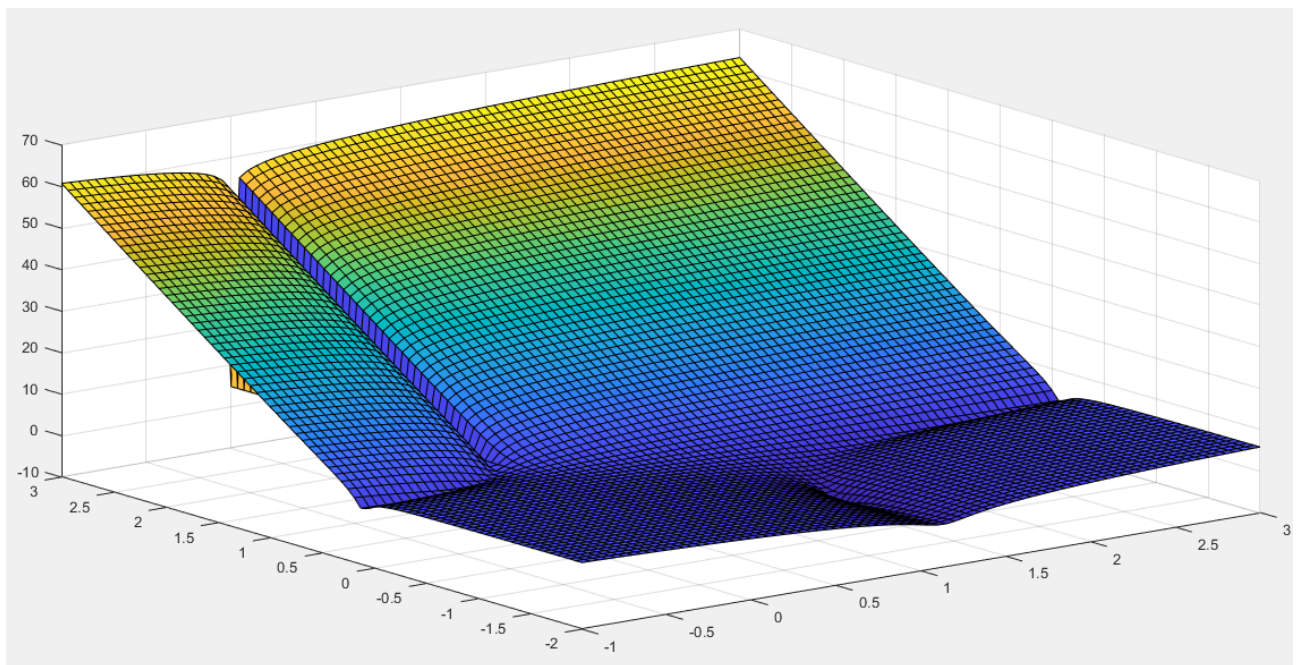
3)

```
function visualisation()  
load partie3;  
A=-1:0.05:3;  
B=-2:0.0625:3;  
[X,Y]=meshgrid(A,B);  
Z=zeros(length(A));  
for i=1:length(A)  
    for j=1:length(A)  
        Z(i,j)=erreur(t,y,X(i,j),Y(i,j));  
    end  
end  
surf(X,Y,Z) %%visualiser la courbe en 3D
```

```
function visualisation()  
load partie3;  
A=-1:0.05:3;  
B=-2:0.0625:3;  
[X,Y]=meshgrid(A,B);  
Z=zeros(length(A));  
for i=1:length(A)  
    for j=1:length(A)  
        Z(i,j)=erreur(t,y,X(i,j),Y(i,j));  
    end  
end  
contour(X,Y,Z) %%visualiser le contour
```



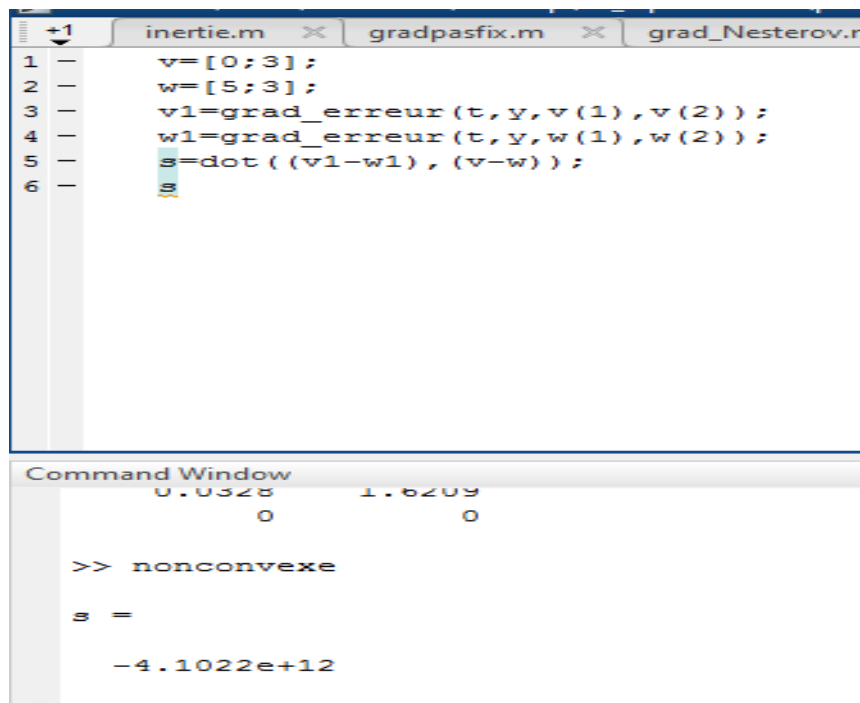
Visualisation du contour du gradient



Visualisation 3D du gradient

NB : Pour visualiser en 3D ou contour il suffit de remplacer `surf(X,Y,Z)` par `contour(X,Y,Z)` dans la fonction `visualisation.m` ou inversement.

4) Cette fonction n'est pas convexe il suffit de choisir deux vecteurs x et y tel que $\langle \nabla x - \nabla y, x - y \rangle \leq 0$



The image shows a MATLAB script editor with three tabs: 'inertie.m', 'gradpasfix.m', and 'grad_Nesterov.r'. The script in 'inertie.m' contains the following code:

```
1 - v=[0;3];
2 - w=[5;3];
3 - v1=grad_erreur(t,y,v(1),v(2));
4 - w1=grad_erreur(t,y,w(1),w(2));
5 - s=dot((v1-w1),(v-w));
6 - s
```

Below the script editor is the Command Window, which displays the following output:

```
0.0328    1.6209
      0         0

>> nonconvexe

s =

-4.1022e+12
```

D'où cette fonction n'est pas convexe même si elle le paraît sur le graphe.

5) On implémente les méthodes et on trace les trajectoires des itérations sur les courbes des niveaux

Le « tic » et « toc » sont écrit afin d'afficher le temps d'exécution.

Méthode du gradient à pas fixe

```
function [iter]=gradpasfix(t,y,x0,r,eps)
tic ;
x=x0-r*grad_erreur(t,y,x0(1),x0(2));
l=[x0,x];
k=1;
tableX=[];           %tableau contenant les valeurs de a à
chaque itération
tableY=[];           %tableau contenant les valeurs de b à
chaque itération
tableX(1)=x(1);
tableY(2)=x(2);
while (norm(x-x0)>eps)
x0=x;
x=x0-r*grad_erreur(t,y,x0(1),x0(2));
l=[l,x];
    k=k+1;
tableX(k)=x(1);
tableY(k)=x(2);
end
iter=k;
x
visualisation();holdon; % on visualise le contour
plot(tableX,tableY,'--o')% visualisation de la trajectoire des
itérations
toc ;
```

Méthode d'inertie

```
function [iter]=inertie(t,y,x0,v0,r,n,eps)
tic ;
v=n*v0+r*grad_erreur(t,y,x0(1),x0(2));
x=x0-v;
l=[x0,x];
k=1;
tableX=[];%tableau contenant les valeurs de a à chaque itération
tableY=[];%tableau contenant les valeurs de b à chaque itération
tableX(1)=x(1);
tableY(2)=x(2);
while (norm(x-x0)>eps)
x0=x;
v=n*v+r*grad_erreur(t,y,x(1),x(2));
x=x-v;
    l=[l,x];
    k=k+1;
tableX(k)=x(1);
tableY(k)=x(2);
end
iter=k;
x
visualisation();holdon;% on visualise le contour
plot(tableX,tableY,'--*')% visualisation de la trajectoire des itérations
toc ;
end
```

Méthode du gradient accéléré de Nesterov

```
function [iter]=grad_Nesterov(t,y,x0,v0,r,n,eps)
tic ;
v=n*v0+r*grad_erreur(t,y,x0(1),x0(2));
x=x0-v;
l=[x0,x];
k=1;
tableX=[];%tableau contenant les valeurs de a à chaque itération
tableY=[];%tableau contenant les valeurs de b à chaque itération
tableX(1)=x(1);
tableY(2)=x(2);
while (norm(x-x0)>eps)
x0=x;
v=n*v+r*grad_erreur(t,y,x(1)-r*v(1),x(2)-r*v(2));
x=x-v;
    l=[l,x];
    k=k+1;
tableX(k)=x(1);
tableY(k)=x(2);
end
iter=k;
x
visualisation();holdon;% on visualise le contour
plot(tableX,tableY,'--.')% visualisation de la trajectoire des itérations
toc ;
end
```

6) a)

Méthode du gradient à pas fixe :

Test pour un pas 0.0004 , $x_0=[1 ;0]$ et une précision de 0.000001.

```
Command Window

>> gradpasfix(t,y,[1;0],0.0004,0.000001)

x =

    1.1991
   -0.8919

Elapsed time is 0.012223 seconds.

ans =

    385
```

L'algorithme converge vers $x=[1.1991 ; -0.8919]$ après 385 itérations et 0.01223 seconds.

Méthode d'inertie :

Test pour $\eta=0.1, p=0.0004, x_0=[1 ;0], v_0=\text{grad}(x_0)$ et une précision de 0.000001.

```
Command Window

>> inertie(t,y,[1;0],grad_erreur(t,y,1,0),0.0004,0.1,0.000001);

iter =

    165

x =

    1.1991
   -0.8919

Elapsed time is 0.007044 seconds.

fx >>
```

L'algorithme converge vers $x=[1.1991 ; -0.8919]$ après 165 itérations et 0.007044 seconds

Méthode accélérée de Nesterov :

Test pour $\eta=0.1, p=0.0004, x_0=[1 ;0], v_0=\text{grad}(x_0)$ et une précision de 0.000001.

```
Command Window

>> grad_Nesterov(t,y,[1;0],grad_erreur(t,y,1,0),0.0004,0.1,0.000001)

iter =

    165

x =

    1.1991
   -0.8919

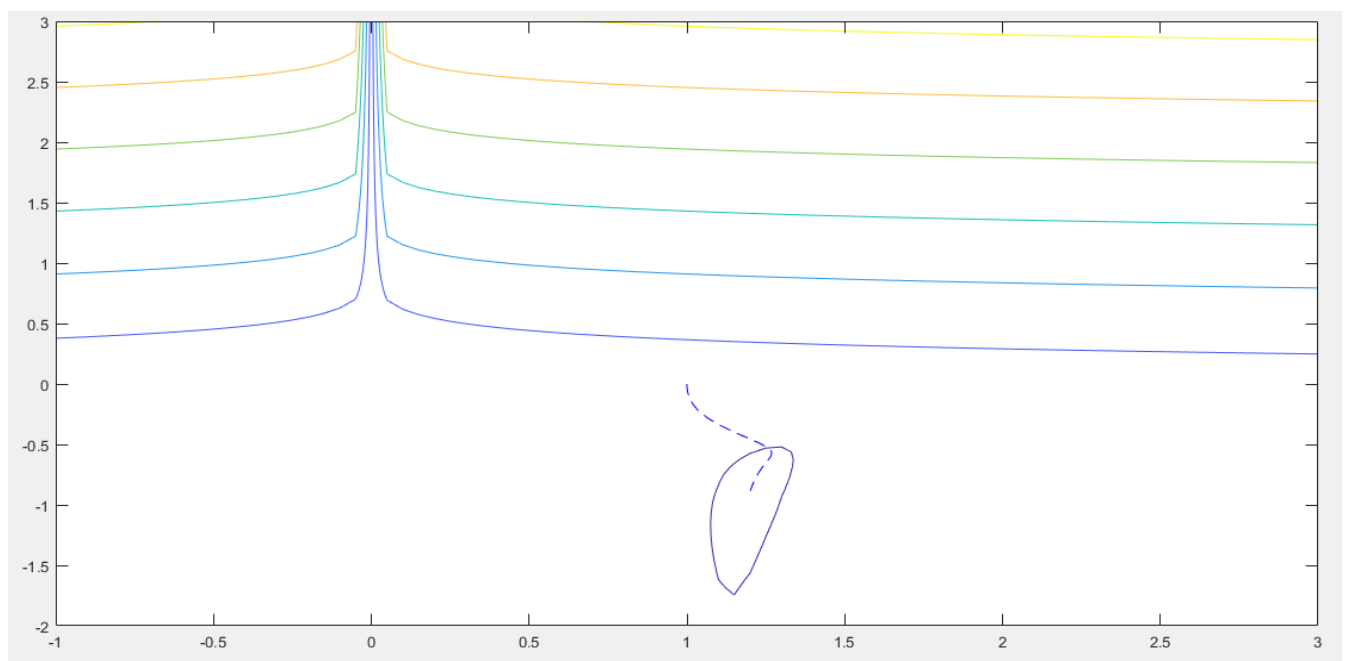
Elapsed time is 0.010050 seconds.
```

L'algorithme converge vers $x=[1.1991 ; -0.8919]$ après 165 itérations.

6.b.i

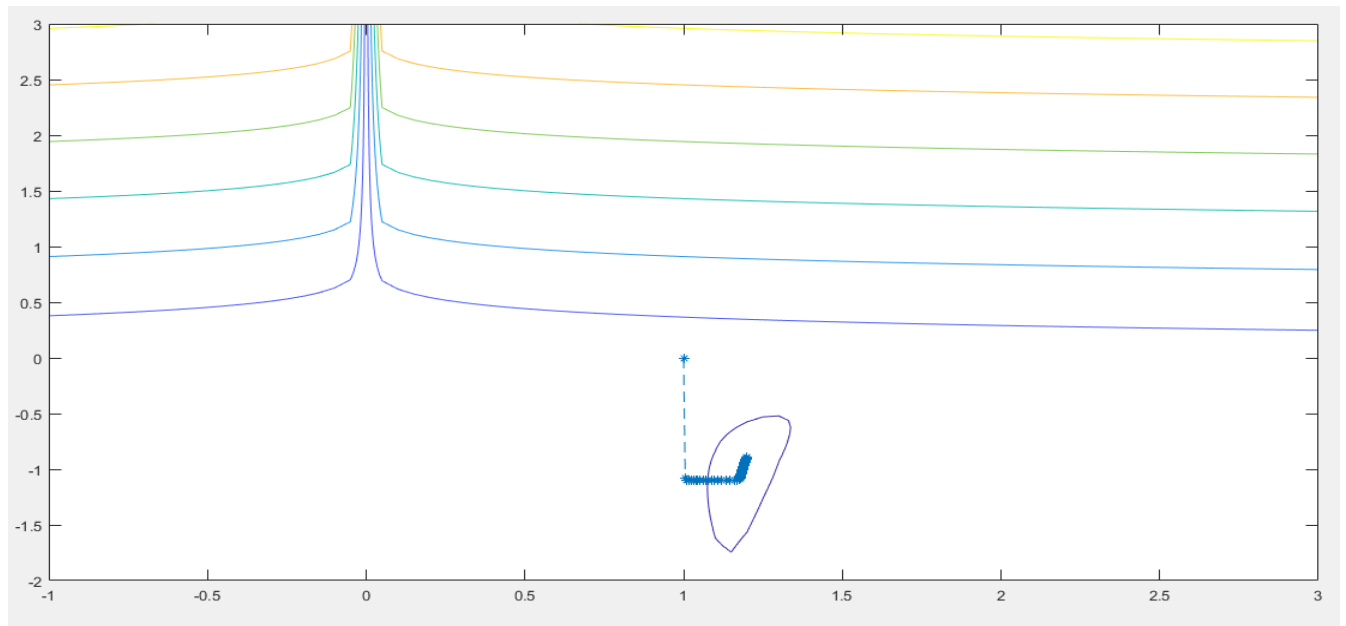
Méthode du gradient à pas fixe :

Test pour un pas 0.0005 , $x_0=[1 ;0]$ et une précision de 0.001.



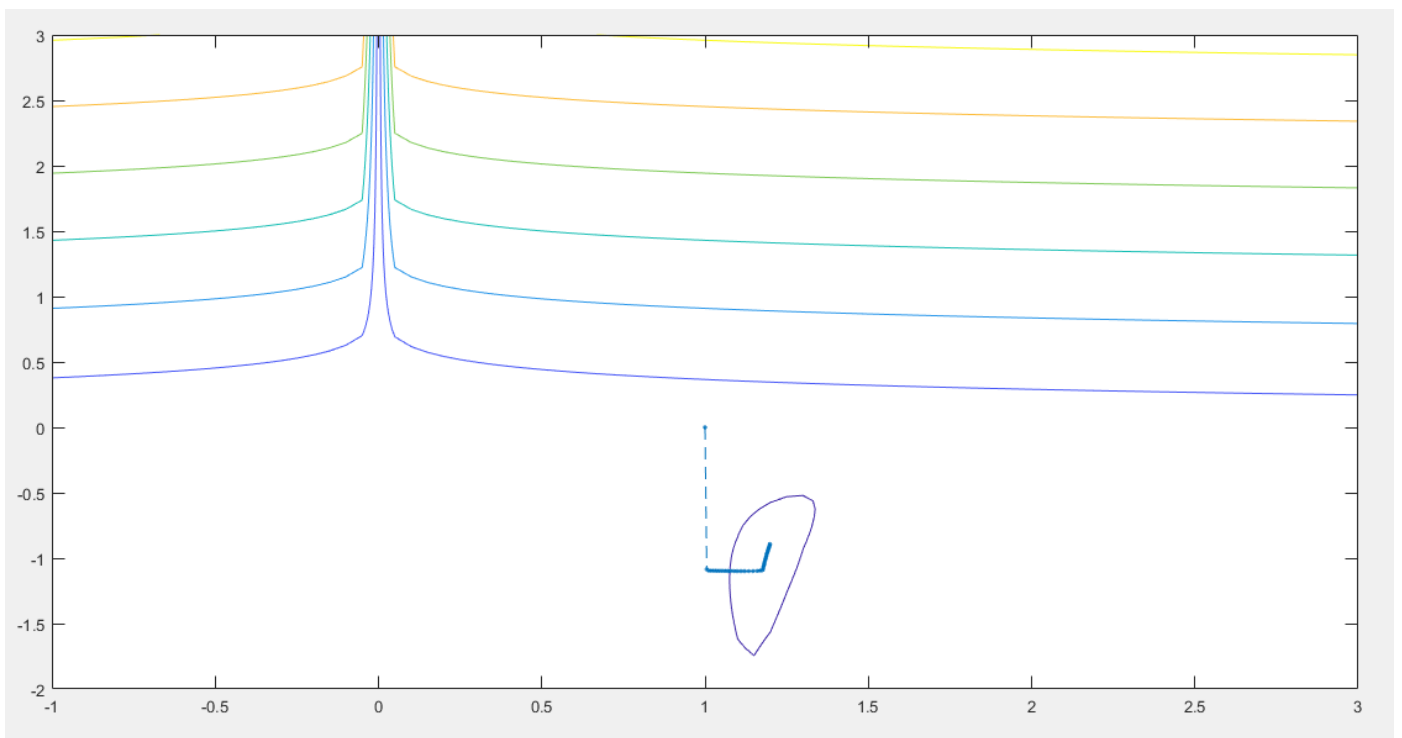
Méthode d'inertie :

Test pour $\eta = 0.1, p = 0.0005, x_0 = [1; 0], v_0 = \text{grad}(x_0)$ et une précision de 0.0001.



Méthode accélérée de Nesterov :

Test pour $\eta = 0.1, p = 0.0005, x_0 = [1; 0], v_0 = \text{grad}(x_0)$ et une précision de 0.0001.

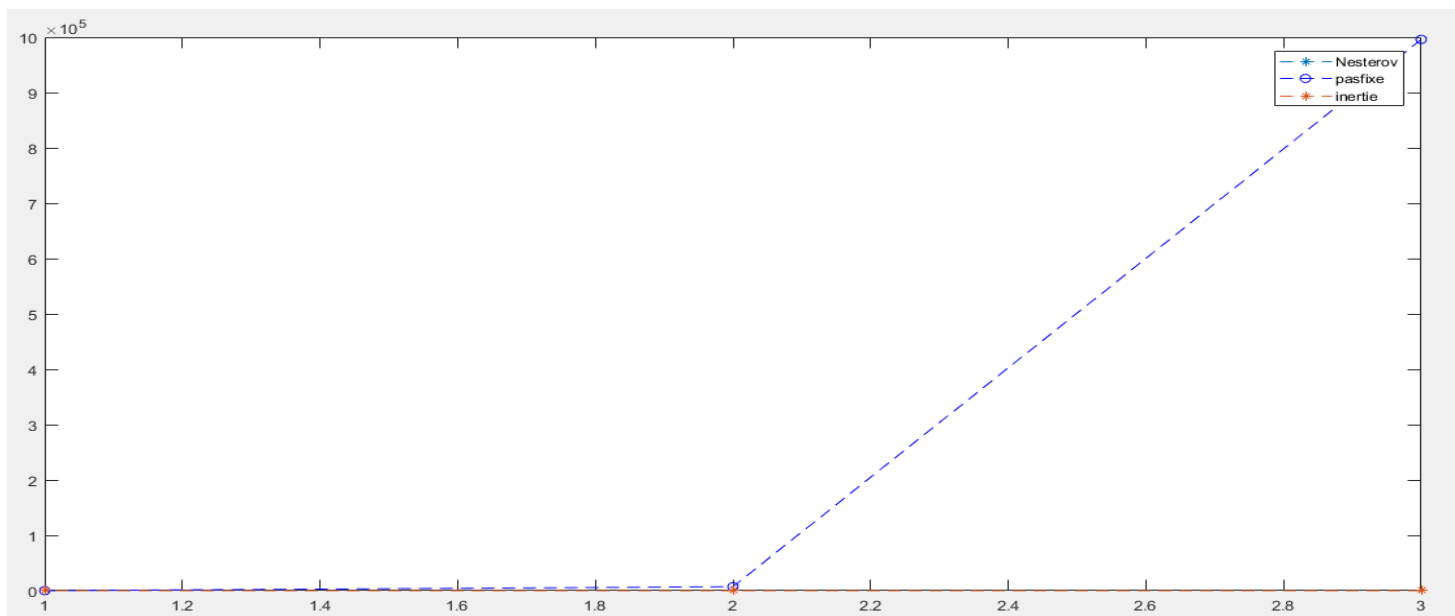


6.b.ii

```

epsilon=[0.000001 0.00001 0.0001 0.001 0.01 0.1];
table1=[]; %tableau contenant les valeurs d'itérations pour la methode de Nesterov
table2=[]; %tableau contenant les valeurs d'itérations pour la methode de gradient à
pas fixe
table3=[]; %tableau contenant les valeurs d'itérations pour la methode d'inertie
for i=1:6
table1(i)=grad_Nesterov(t,y,[1;0],grad_erreur(t,y,1,0),0.00035,0.09,epsilon(i));
table2(i)=gradpasfix(t,y,[1;0],0.01,epsilon(i));
table3(i)=inertie(t,y,[1;0],grad_erreur(t,y,1,0),0.00035,0.09,epsilon(i));
end
disp(table1);
disp(table2);
disp(table3);
plot(-log(epsilon)/log(10),table1,'--*'); %construction de nbre d'itérations de la
methode de Nesterov en fonction de log10(epsilon)
holdon;
plot(-log(epsilon)/log(10),table2,'b--o') %construction de nbre d'itérations de la
methode de gradient à pas fixe en fonction de log10(epsilon)
plot(-log(epsilon)/log(10),table3,'--o') %construction de nbre d'itérations de la
methode d'inertie en fonction de log10(epsilon)
legend('Nesterov','pasfixe','inertie')

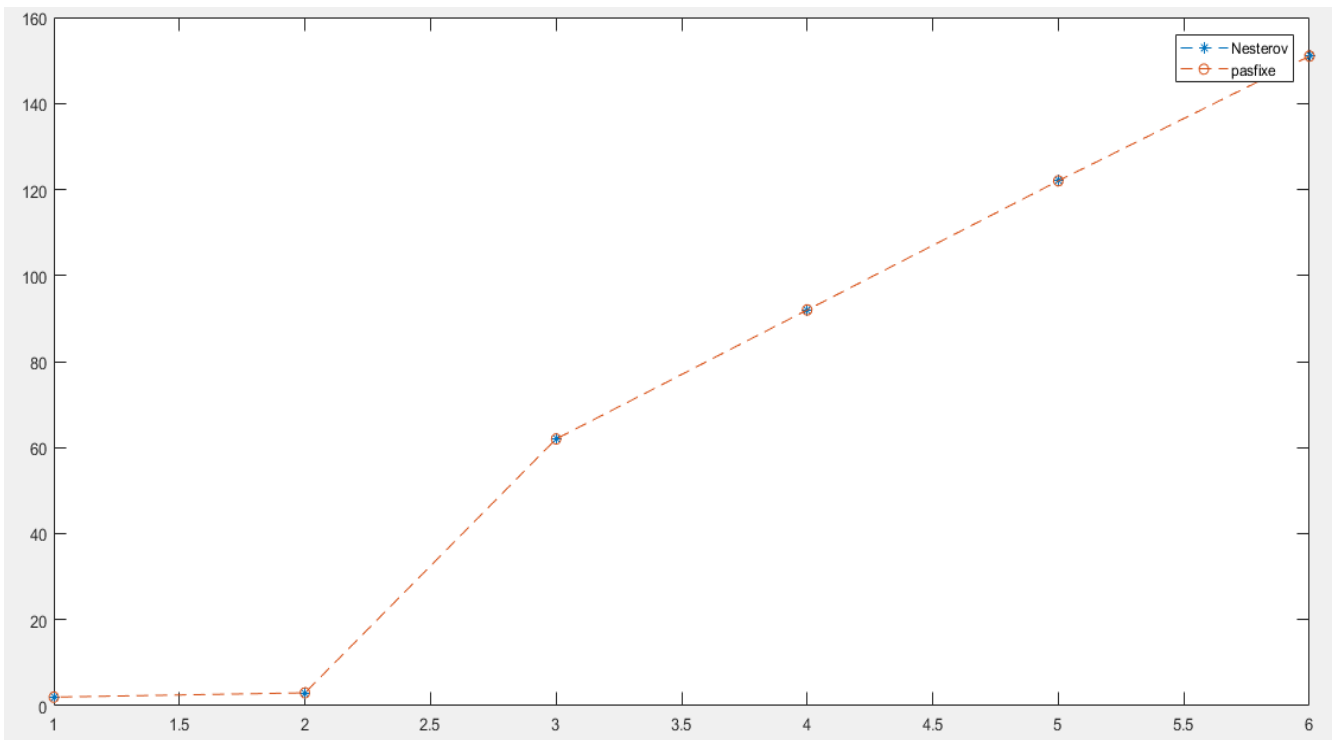
```



On trace maintenant les 3 courbes au même temps :

On remarque que le nombre d'itérations pour la méthode du gradient à pas fixe est très importante par rapport aux nombres des deux autres méthodes de plus la variation des nombres d'itérations des méthodes d'inertie et de Nesterov ne peut pas être identifier donc on doit tracer ces variations pour les identifier.

On trace les deux courbes de la méthode de Nesterov et d'inertie on obtient :



Ainsi on conclue que les méthodes de Nesterov et d'inertie possèdent exactement le même nombre d'itérations pour les même paramètres.

NB :Pour visualiser les courbes d'itérations il faut mettre en commentaires les commandes liées à plot dans les fonctions grad_Nesterov, gradient à pas fixe et inertie dans le programme afin d'éviter les duplications des courbes d'itérations avec celle des trajectoires.

7.a

On effectue les tests pour des paramètres :

$\eta = 0.09, p = 0.00035, x_0 = [-2; -1], v_0 = \text{grad}(x_0)$ et une précision de 0.000001.

```
-----
>> inertie(t,y,[-2;-1],grad_erreur(t,y,-2,-1),0.00035,0.09,0.000001);

x =

    1.1991
   -0.8919

Elapsed time is 0.127681 seconds.
```

```
>> grad_Nesterov(t,y,[-2;-1],grad_erreur(t,y,-2,-1),0.00035,0.09,0.000001)

x =

    1.1991
   -0.8919

Elapsed time is 0.127334 seconds.

ans =

    6530

^^
```

```
>> gradpasfix(t,y,[-2;-1],0.00035,0.000001)

x =

    1.1991
   -0.8919

Elapsed time is 0.127436 seconds.

ans =

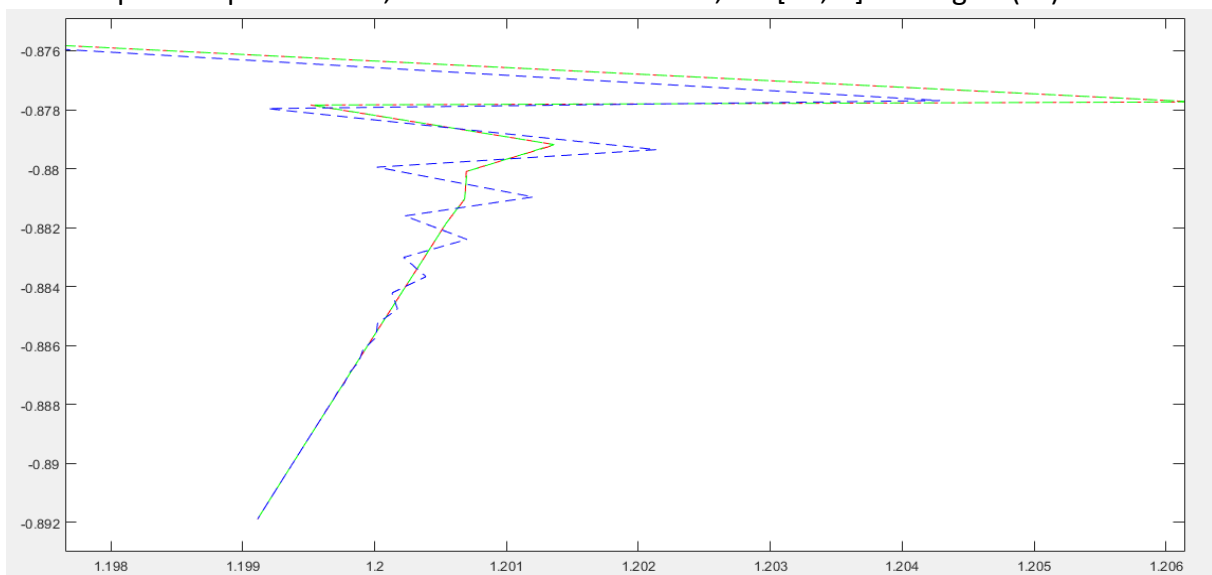
    7454
```

On distingue que les nombres d'itérations des deux premières méthodes (inertie et accéléré de Nesterov) sont égales ainsi ces deux méthodes sont plus rapides que la méthode du gradient. Mais de point de vue temps d'exécution la méthode de Nesterov est la plus rapide et c'est parce qu'elle possède un nombre d'opérations élémentaires inférieurs à celle de la méthode d'inertie et la méthode du gradient à pas fixe.

Ainsi par temps d'exécution on trouve : 1- gradient accéléré de Nesterov

2- gradient à pas fixe 3-méthode d'inertie.

7.b Test pour un pas 0.00035, une tolérance 0.000001 , $x_0 = [-2 ; -1]$ et $v_0 = \text{grad}(x_0)$

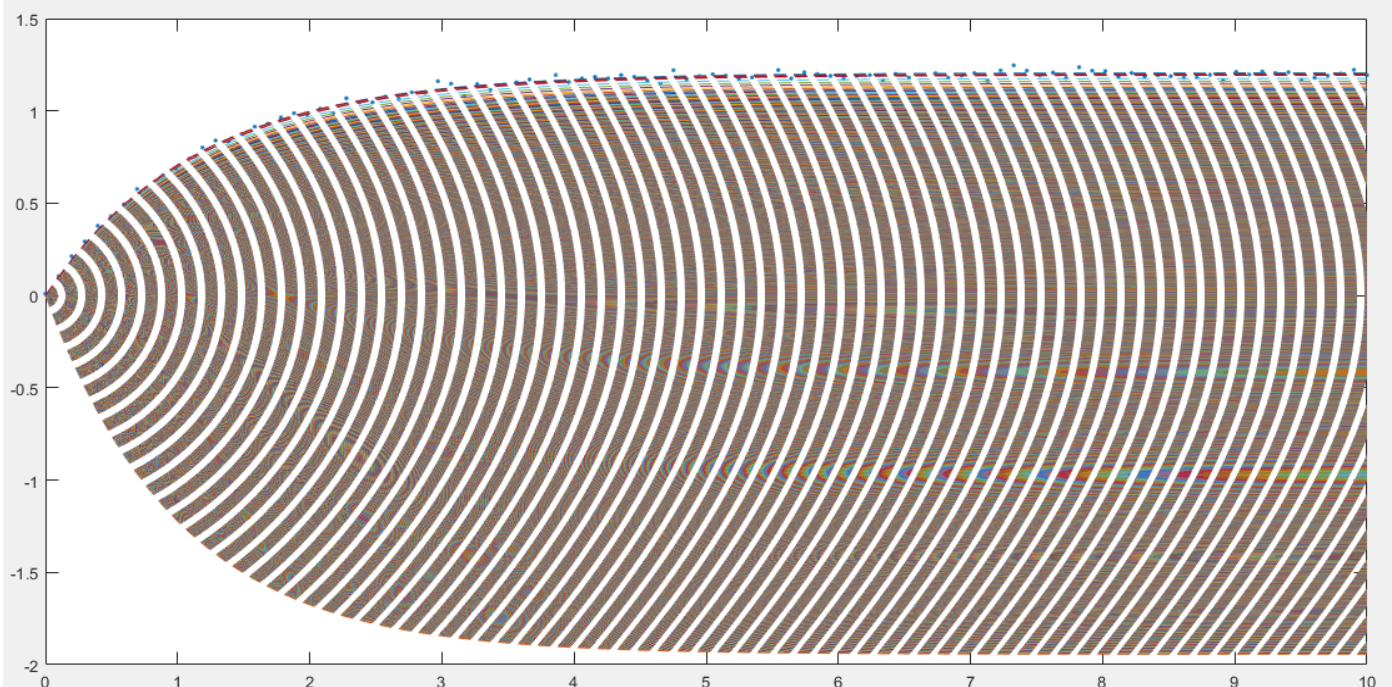


7-b- Afin d'aboutir à la solution la méthode de gradient à pas fixe prend un chemin plus long que les méthodes de Nesterov et d'inertie qui prennent exactement le même chemin et c'est pertinent puisque les deux dernières méthodes ont le même nombre d'itération.

7.c.

On trace l'animation en utilisant la méthode de Nesterov.

```
function [iter]=courbefinale(t,y,x0,v0,r,n,eps)
plot(t,y, '.');hold on;
v=n*v0+r*grad_erreur(t,y,x0(1),x0(2));
x=x0-v;
l=[x0,x];
tableY=[];
for i=1:1:102
tableY(i)=x(1)*(1-exp(x(2)*t(i)));
end
plot(t,tableY, '--');
while (norm(x-x0)>eps)
x0=x;
v=n*v+r*grad_erreur(t,y,x(1)-r*v(1),x(2)-r*v(2));
x=x-v;
l=[l,x];
for i=1:1:102
tableY(i)=x(1)*(1-exp(x(2)*t(i)));
end
plot(t,tableY, '--');
end
```



$\eta = 0.09, p = 0.00035, x_0 = [-2; -1], v_0 = \text{grad}(x_0)$ et une précision de 0.000001.

ZOOM :

