
# PROJET DU MODULE TP OPTIMISATION

**Travail présenté par :**

Mohamed Sy

Hafedh Ferchichi

Mohamed Aziz Tousli

**Encadré par :**

**M. Walid Benromdhane**

**Mme Yosra Gati**

**Année universitaire :**

2017 _ 2018

# I) But 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.

# II) Optimisation sans contrainte : La régression linéaire simple:

## 1) Etude du problème :

**1.**

$$E = \sum_{k=0}^{n} (yk - \hat{y}k)^2$$

La fonction d'erreur E dépend uniquement des variables $a_1$ et $a_0$, car :
$E = f(\hat{y}_i)$ avec $\hat{y}_i = a_1.x_i + a_0$.

**2.**

$$\text{Soit, } M = \begin{pmatrix} 1 & x1 \\ \vdots & \vdots \\ 1 & xn \end{pmatrix}, \; a = \begin{pmatrix} a0 \\ a1 \end{pmatrix} \text{ et } m = \begin{pmatrix} y1 \\ \dots \\ yn \end{pmatrix}$$

$$\text{Ma-m} = \begin{pmatrix} a0 + a1.x - y1 \\ \dots \\ a0 + a1.x - yn \end{pmatrix}$$

On trouve donc : $||Ma - m||^2 = \sum(Ma - m)(i)^2 = E(a)$

**3.**

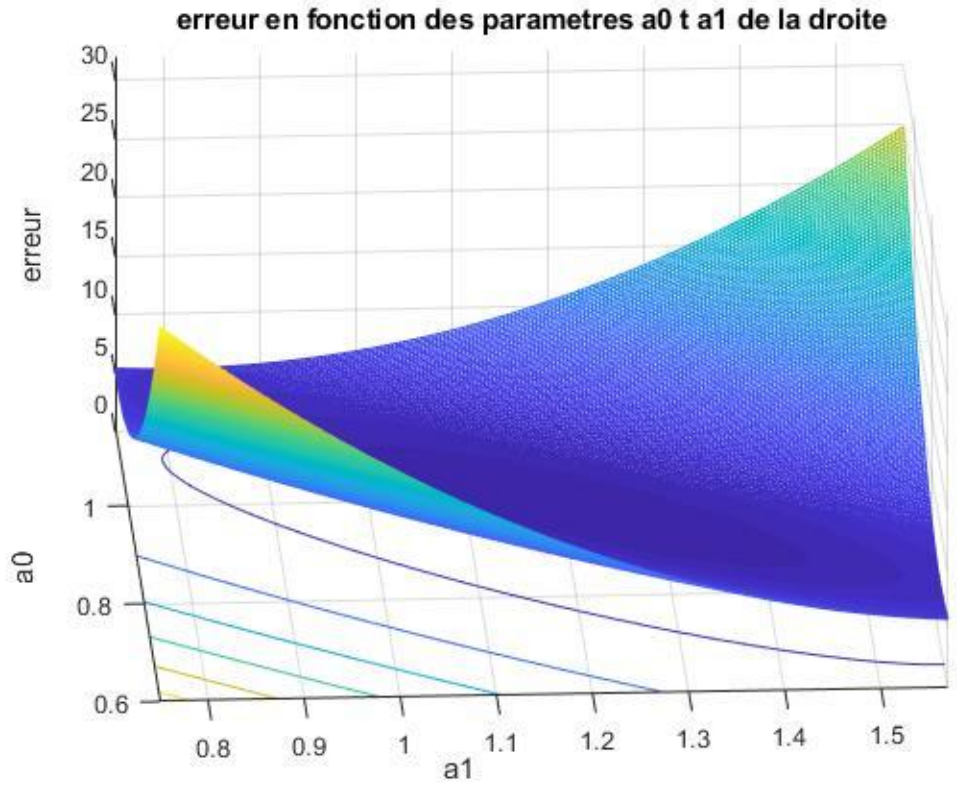
**a.** Voici la fonction d'erreur en MatLab:

```
function e=erreur(A,x,y)
M=[ones(101,1),x];
e=norm(M*A-y,2).^2;
end
```

**b.** Voici la fonction d'affichage en MatLab:

```
%La fonction affiche.m qui affiche l`erreur en 3D et son contour
function affc(x,y,xmin,xmax,ymin,ymax,n)
X=linspace(xmin,xmax,n);
Y=linspace(ymin,ymax,n);
l1=length(X);
l2=length(Y);
Z=zeros(l2,l1);
for i=1:l2
    for j=1:l1
        Z(i,j)=erreur([Y(i);X(j)],x,y);
    end
end
contour(X,Y,Z);
hold on;
```

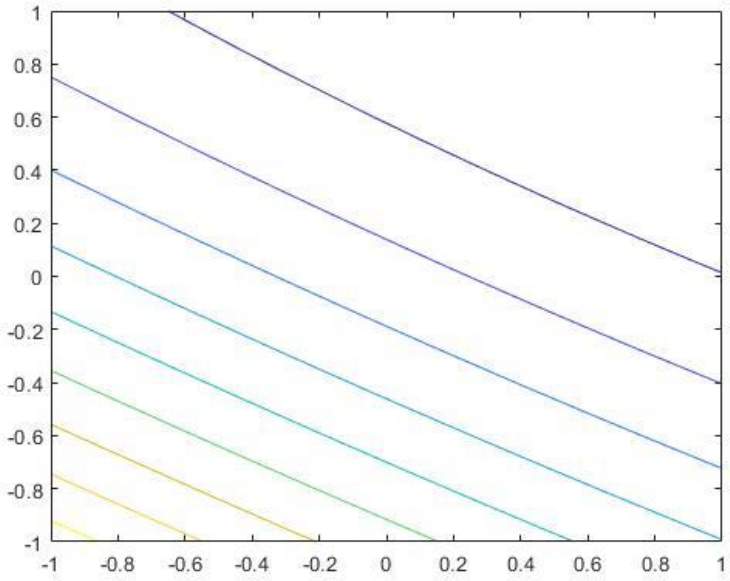
Voici la figure qu'on obtient après l'éxécution du code ci-dessus:



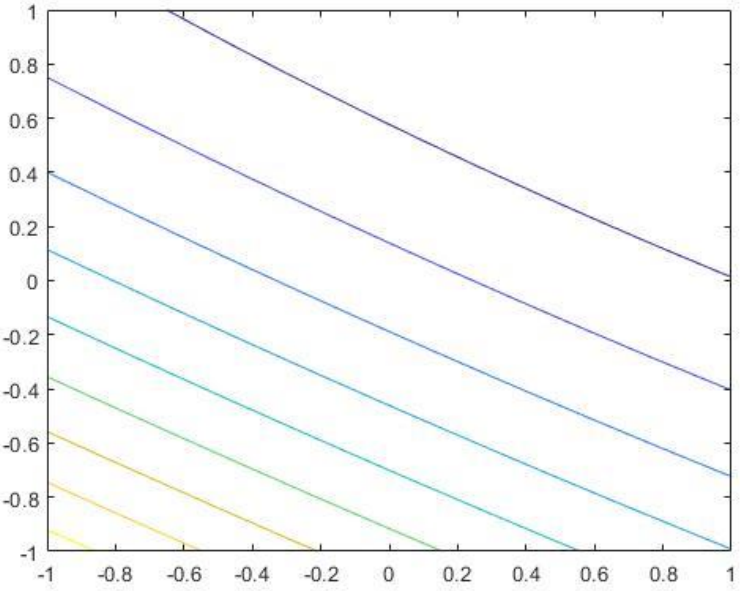

On remarque que E est une fonction convexe et admet un minimum $\epsilon$ [0,2]x[0,2].

Voici le contour:

- Pour n=50:




- Pour n=100:



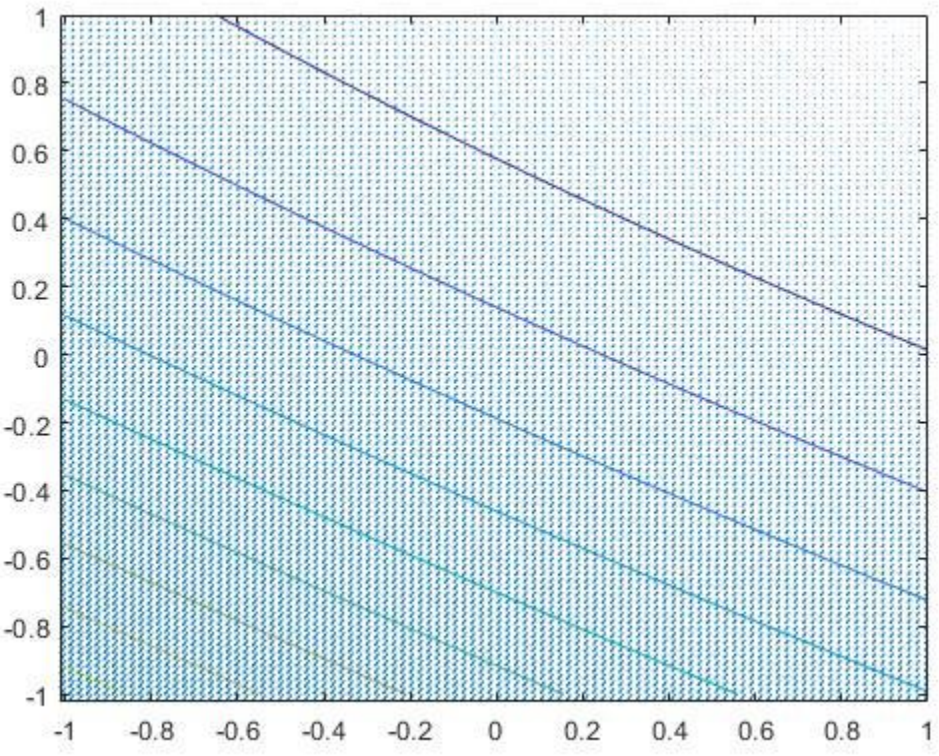

On remarque ici que le minimum est pour $a_0 \in [0,2]$ et $a_1 \in [0,2]$.

**c.**

Voici le code:

```
function affcg(x,y,xmin,xmax,ymin,ymax,n)
X=linspace(xmin,xmax,n);
Y=linspace(ymin,ymax,n);
l1=length(X);
l2=length(Y);
Z=zeros(l2,l1);
for i=1:l2
    for j=1:l1
        Z(i,j)=erreur([Y(i);X(j)],x,y);
    end
end
contour(X,Y,Z);
hold on;
[dfx,dfy]=gradient(Z,(ymax-ymin)./(n-1),(xmax-xmin)./(n-1));
quiver(X,Y,dfx,dfy);
hold off;
```

Et voici le graphe après éxécution du code:




Il n'existe pas des points critiques autre que le minimum.

**4.**

On a, $E(a) = ||Ma-m||^2$

$= < Ma-m , Ma-m > = < Ma , Ma > + ||m||^2 - 2* < Ma , m >$

$= < {}^tM.M.a , a > - < 2.{}^tM.m , a > + ||m||^2$

$= ½ < A.a , a > - < a , b > + ||m||^2$

$= F(a) + ||m||^2$

Donc minimiser E revient à minimiser F.

**5.**

$F(a+h) - F(a) = < A.a – b , h > + < h , h >$

Donc, $DF(a)(h) = < A.a – b , h >$

Ainsi, $\nabla(F)(a) = A.a – b$

Et, $\nabla^2(F)(a) = A.$

**6.**

F est minimale au point $\nabla(F)(x) = o$ donc pour $Aa-b=o$ .

# 2)Algorithmes de résolution :

- **Méthode de Newton :**

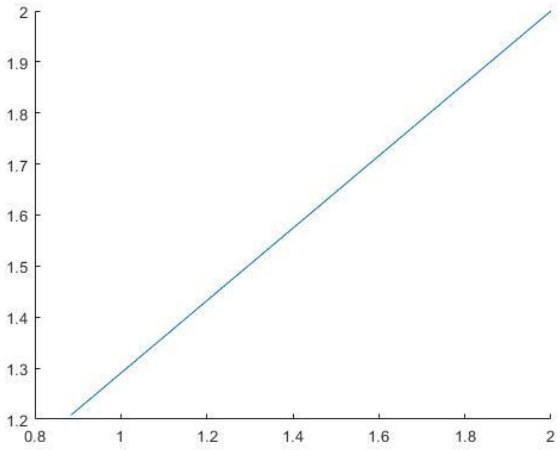
```
function R=Newton(x,y,x0,y0,epsilon)
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
R0=[x0;y0];
while (true)
    R=R0-(A^(-1))*(A*R0-b);
    if (norm(R-R0)<epsilon)
        break;
    end
    R0=R;
end
```

Le code de la fonction de SuiviNewton est:

```
function E=suivinewton(x,y,x0,y0,epsilon)
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
R0=[x0;y0];
hold on;
E=R0;
while (true)
    R=R0-(A^(-1))*(A*R0-b);
    E=horzcat(E,R);
    if (norm(R-R0)<epsilon)
        break;
    end
    R0=R;
end
X=E(1,:);
Y=E(2,:);
plot(X,Y,'-');
```

Et son éxécution nous donne:

```
>> Newton(x  , y , 1 ,2,0.0004)

ans =

    0.8844
    1.2083
```



- **Méthode de gradient à pas constant :**

```
function R=gradpasconst(x,y,x0,y0,p,epsilon)
R0=[x0;y0];
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
while (true)
    R=R0-p*(A*R0-b);
    if (norm(R-R0)<epsilon)
        break;
    end
    R0=R;
end
```

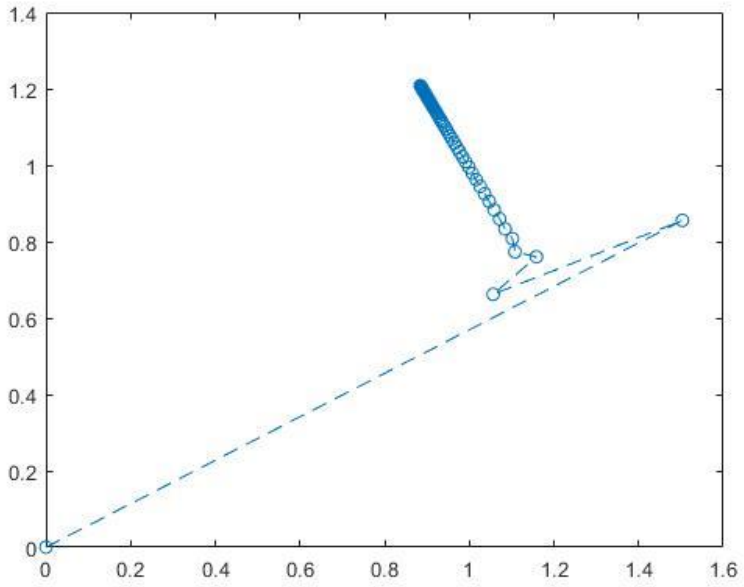
Le code de la fonction de SuiviGradientapasconstant est:

```
function E=suivigradpasconst(x,y,x0,y0,p,epsilon)
R0=[x0;y0];
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
E=R0;
while (true)
    R=R0-p*(A*R0-b);
    E=horzcat(E,R);
    if (norm(R-R0)<epsilon)
        break;
    end
    R0=R;
end
X=E(1,:);
Y=E(2,:);
plot(X,Y,'--o');
```

L'éxécution du code donne:

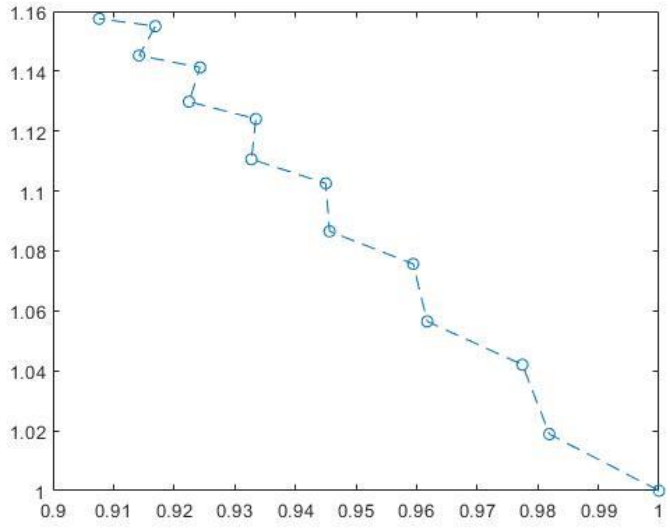
- Pour $x_0=y_0=0$, $\varepsilon=0.00001$, et $p=0.001$

```
>> suivigradpasconst(x  , y , 0,0,0.01,0.00001)

ans =

  Columns 1 through 12

         0    1.5034    1.0564    1.1584    1.1080    1.1018    1.0846    1.0717    1.0588    1.0471    1.0360    1.0258
         0    0.8554    0.6622    0.7601    0.7734    0.8076    0.8334    0.8592    0.8827    0.9048    0.9253    0.9445

  Columns 13 through 24

    1.0162    1.0073    0.9990    0.9912    0.9840    0.9772    0.9709    0.9651    0.9596    0.9545    0.9498    0.9453
    0.9623    0.9790    0.9945    1.0090    1.0225    1.0351    1.0468    1.0577    1.0679    1.0774    1.0863    1.0945
```




- Pour $x_0=y_0=1$, $\varepsilon=0.00001$, et $p=0.01561725$

```
>> suivigradpasconst(x  , y , 1,1,0.01561725,0.01)

ans =

  Columns 1 through 12

    1.0000    0.9819    0.9775    0.9618    0.9594    0.9456    0.9450    0.9328    0.9335    0.9224    0.9243    0.9142
    1.0000    1.0189    1.0421    1.0565    1.0757    1.0866    1.1026    1.1106    1.1241    1.1299    1.1413    1.1452

  Columns 13 through 14

    0.9169    0.9076
    1.1550    1.1575
```



- **Méthode de gradient à pas optimal :**

```
function al=gradopt(x,y,x0,y0,epsilon)
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
a0=[x0;y0];
r=A*a0-b;
ro=dot(r,r)/dot(A*r,r);
while (true)
    al=a0-ro*(A*a0-b);
    if (norm(al-a0)<epsilon)
        break;
    end
    a0=al;
    r=A*a0-b;
    ro=dot(r,r)/dot(A*r,r);
end
```

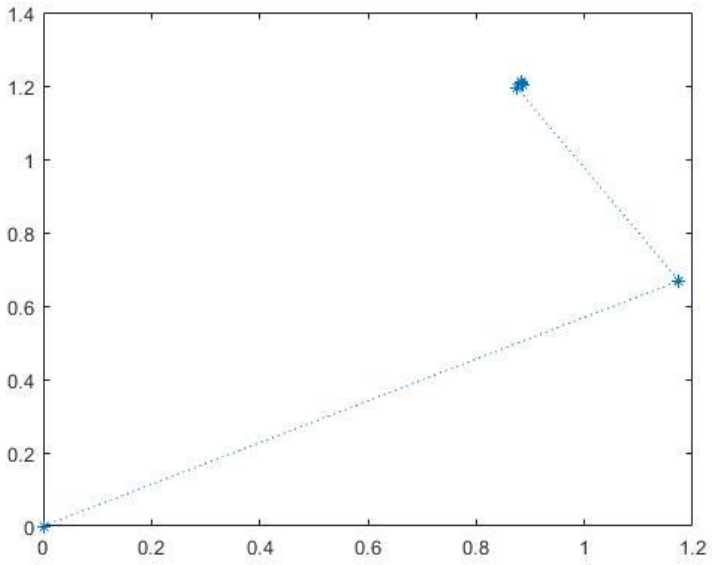
Le code de la fonction de SuiviGradientapasoptimal est:

```
function E=suivigradopt(x,y,x0,y0,epsilon)
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
a0=[x0;y0];
E=a0;
r=A*a0-b;
ro=dot(r,r)/dot(A*r,r);
while (true)
    al=a0-ro*(A*a0-b);
    E=horzcat(E,al);
    if (norm(al-a0)<epsilon)
        break;
    end
    a0=al;
    r=A*a0-b;
    ro=dot(r,r)/dot(A*r,r);
end
X=E(1,:);
Y=E(2,:);
plot(X,Y,':*');
```

L'éxécution du code donne:

- Pour $x_0=y_0=0$, $\varepsilon=0.000001$

```
>> suivigradopt(x  , y ,0,0,0.000001)

ans =

         0    1.1747    0.8749    0.8875    0.8843    0.8844    0.8844    0.8844    0.8844
         0    0.6684    1.1953    1.2025    1.2082    1.2082    1.2083    1.2083    1.2083
```




- **Méthode de gradient conjugué:**

```
function al=gradconj(x,y,x0,y0,epsilon)
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
a0=[x0;y0];
r=A*a0-b;
d=r;
ro = dot(r,r)/dot(A*r,r);
while (true)
    al=a0-ro*d;
    if (norm(a0-al)<epsilon)
        break;
    end
    beta=(norm(A*al-b)/norm(r)).^2;
    r=A*al-b;
    d=r+beta*d;
    a0=al;
    ro=dot(r,r)/dot(A*r,r);
end
```

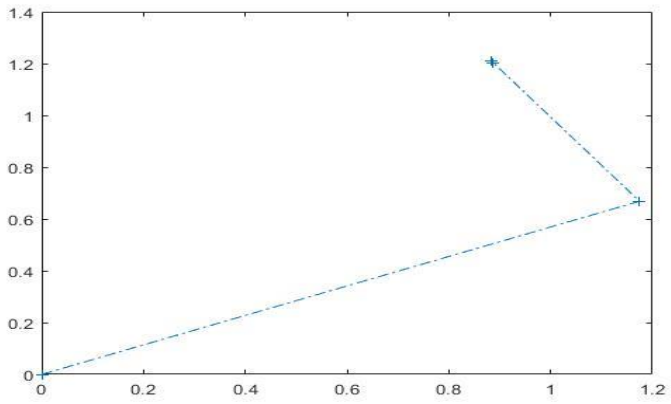
Le code de la fonction de SuiviGradientaconjugué est:

```
function A=suivigradconj(x,y,x0,y0,epsilon)
M=[ones(101,1),x];
b=M.'*y;
A=M.'*M;
a0=[x0;y0];
E=a0;
r=A*a0-b;
d=r;
ro = dot(r,r)/dot(A*r,r);
while (true)
    al=a0-ro*d;
    E=horzcat(E,al);
    if (norm(a0-al)<epsilon)
        break;
    end
    beta=(norm(A*al-b)/norm(r)).^2;
    r=A*al-b;
    d=r+beta*d;
    a0=al;
    ro=dot(r,r)/dot(A*r,r);
end
X=E(1,:);
Y=E(2,:);
plot(X,Y,'-.+');
```

L'éxécution du code donne:

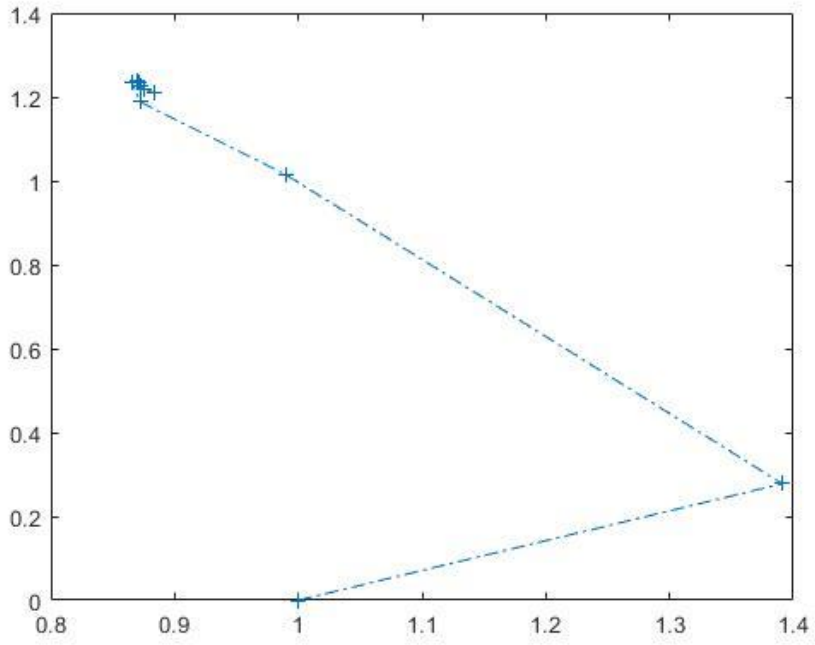
- Pour $x_o=y_o=0$, $\varepsilon=0.0001$

```
>> suivigradconj(x  , y ,0,0,0.0001)

ans =

  101.0000   50.5000
   50.5000   33.8350
```



- Pour $x_0=1$, $y_0=0$, $\varepsilon=0.001$

```
>> suivigradconj(x  , y ,1,0,0.001)

ans =

  101.0000   50.5000
   50.5000   33.8350
```



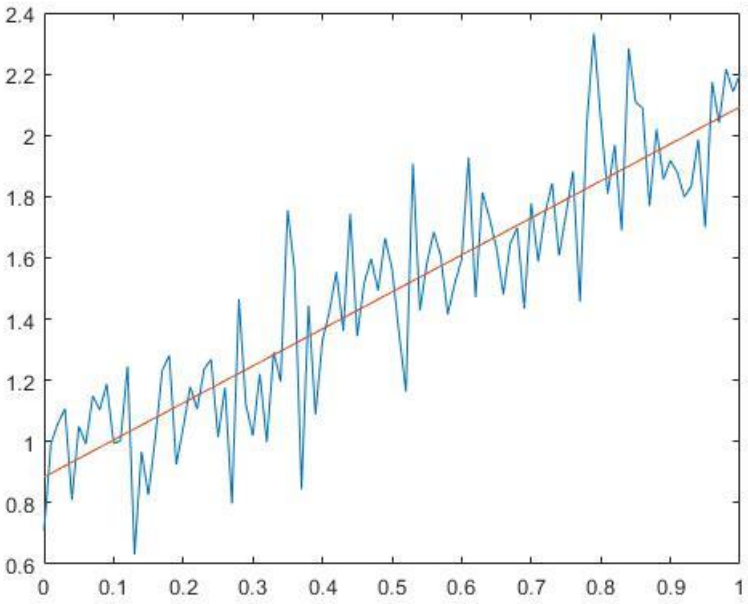

## 3.

Le nuage des points et la droite des regression optimale :

Le script est :

```
>>  plot(x,[ones(101,1),x]*[0.8844;1.2083])
>> plot(x,y)
>> hold on
>>  plot(x,[ones(101,1),x]*[0.8844;1.2083])
>>
```

Apres l'execution on a :



# III) Optimisation avec contrainte : La régression linéaire multiple :

## Etude du probleme :

1) Matriciellement, on a ; **$\tilde{y}$=X.a**

y = [yi] :

2) E(a)=llXa-yll²

$\nabla$ (E(a))=2 $X^t$(Xa-y) ; donc par concequent la Hessienne est de :

Hess(E(a))=2$^T$(X)*X : Or la Hessienne est symétrique positive

En effet <$X^t$Xh,h>=<Xh,Xh> >=0 pour tout h

Donc notre fonction est convexe (cf cours ) << **f convexe si et seulement si la hessiene de f est definie positive** >>.

3°)

**HessE(a) est positive donc E(a) est convexe**

et par concequent E(a) admet au moins un minimum

Minimiser E(a) revient a chercher le min parmis les points critiques de E(a)

d'ou , **$\nabla$E(a)=0**

**ie :-2*X$^T$y +2*X$^T$X*a =0**

Si on se place dans le cas où $X^TX$ inversible alors $a=(X^tX)^{-1}X^ty$ donc on trouve les paramètres qui minimise l'erreur.

On peut aussi proceder par la methode de Gauss Seidel pour resoudre le probleme .

4°)

```
function Z = Question4partie2()
load partie2 ;
A=transpose(X) ;
Z=inv(A,X)*(A,y) ;
plot(y) ;
hold on ;
plot(Z*X) ;
```

, l'execution donne :

```
ans =

    1.4565
    1.5982
    1.5982
    0.5096
    0.1849
    0.6462
    0.6462
```

# 2)Methode de penalisation :

## 2-1)

soit **f(a)=||Xa-y||²+λ||a||²**

Cherchons **minf(a)**

Determinons les points critiques : **∇f(a)=0 -2.XT.y+2(XT.X+λI).a=0** avec I : la matrice identité

Comme XT.X est semi définie positive (XT.X+λI) est symétrique définie positive (ses valeurs propres sont >0 car λ>0)

(XT.X+λI) est inversible et, D'où la pénalisation permet d'éviter

l'explosion des coefficients.

## 2-2-3 : La variation des coefficients en fonction de lambda :

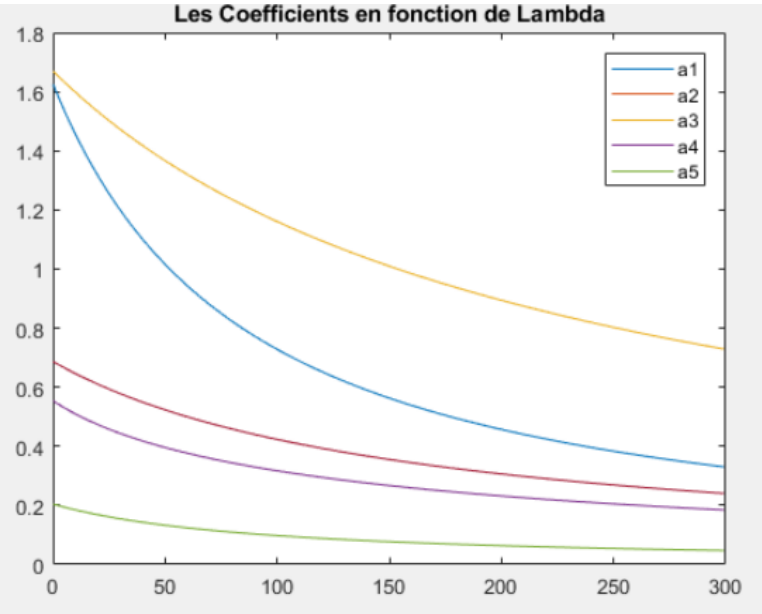
Le programe ci-dessous traduit le script de la variations des coefficients en fonction de lambda :

```
function z=Varia_Coef2()
load partie2;
a=[1,1,1,1,1,1,1];
m1=[];m2=[];m3=[];
m4=[];m5=[];m6=[];m7=[];
lam=[];
k=300;
p=0.1
for lambda=p:1:k
        lam=[lam,lambda];
        b=Newton_partie2(a',lambda,0.00001);
        m1=[m1,b(1)];
end
for lambda=p:1:k
        b=Newton_partie2(a',lambda,0.00001);
```

```
end
for lambda=p:1:k
        b=Newton_partie2(a',lambda,0.00001);
        m2=[m2,b(2)];
end
for lambda=p:1:k
        b=Newton_partie2(a',lambda,0.00001);
        m3=[m3,b(3)];
end
for lambda=p:1:k
        b=Newton_partie2(a',lambda,0.00001);
        m4=[m4,b(4)];
end
for lambda=p:1:k
        b=Newton_partie2(a',lambda,0.00001);
        m5=[m5,b(5)];
```

```
end
for lambda=p:1:k
        b=Newton_partie2(a',lambda,0.00001);
        m6=[m6,b(6)];
end
for lambda=p:1:k
        b=Newton_partie2(a',lambda,0.00001);
        m7=[m7,b(7)];
end
 plot(lam,m1,lam,m2,lam,m3,lam,m4,lam,m5,lam,m6,lam,m7);
 legend({'a1','a2','a3','a4','a5'});
title('Les Coefficients en fonction de Lambda');
end
```

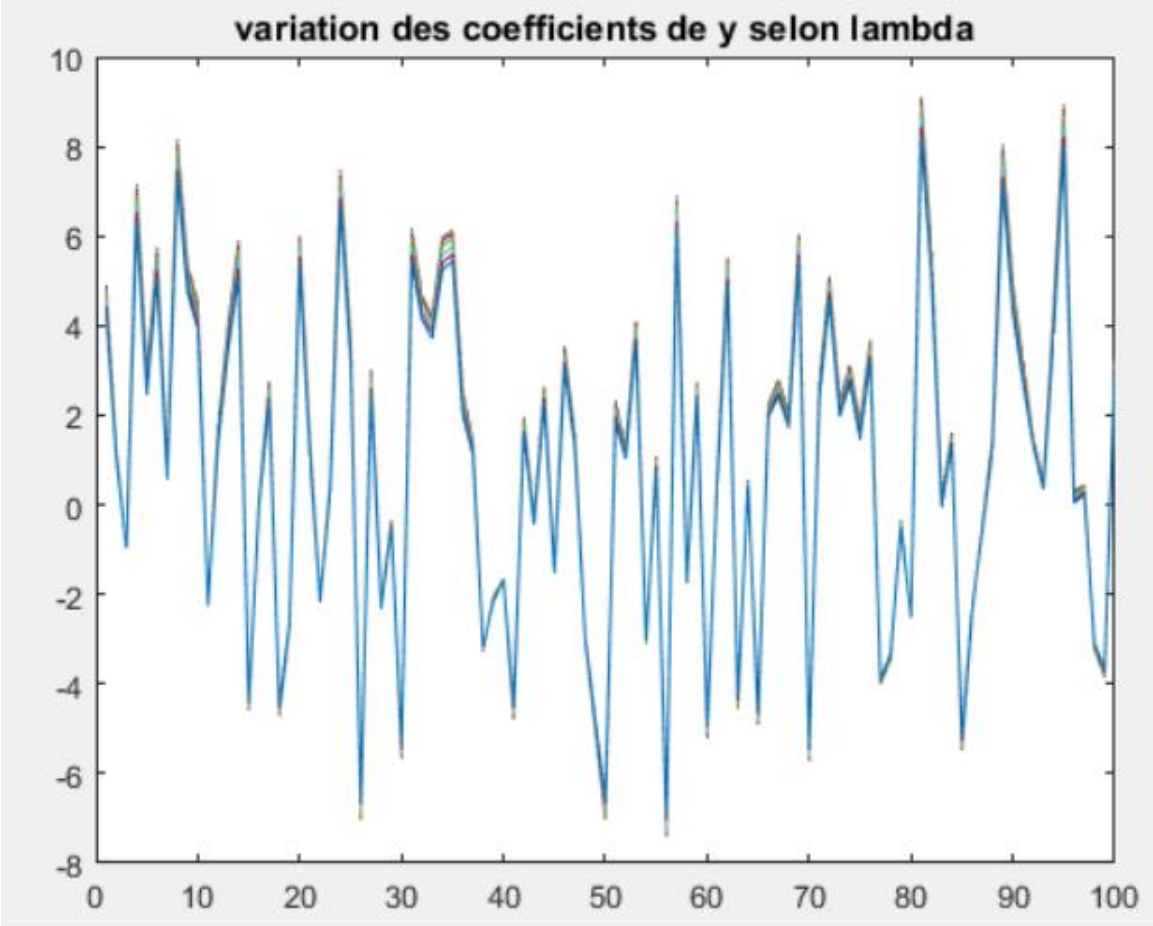
Voici les courbes qu'on obtient :



4-) le script pour les courbes de Y :

```
load('partie2.mat');%appel a Partie2.mat
L=[0.1 .5 1 2 5 10 15 20]; %les valeurs de lamda choisis ( elle peut être modifiée )

for i = 1:length(L) % parcours a chaque valeur de lambda dans L
    a=inv(X.'*X+L(i)*eye(7))*X.'*y; %le vecteur a d'apres la formule de penalisation
    b=X*a; % le vecteur y^
    for k =1:100
        B(i,k)=b(k);
    end
end
plot(B');% afficher la variation des coefficients de y^ selon lambda ainsi que les valeur de y
title('variation des coefficients de y^ selon lambda');
hold on;
plot(y) %afficher les coeficients de y^ et y dans la meme courbe
```

L'execution du programme donne :



On constate que plus lambda duminie plus y est optimal donc il vaut meiux prendre les petits valeurs de lambda , pour see rapprocher de la solution optimal ! d'ailleures la question precedente nous donne des valeurs ecartés de a !

# IV) Méthode de gradient accéléré :

## 1) Système de premier ordre :

**1.**

On a:

$$E(a, b) = \log\left( \sum_{i=1}^{n} [\, y_i - a.( 1-e^{bti} ) \,]^2 \right)$$

On aura donc:

$$(E)=\frac{1}{\sum_{i=1}^{n} [\, yi - a*( 1-e^{bti} ) \,]^2} \begin{pmatrix} \sum_{i=1}^{n} 2\left(yi - a\left( 1 - e^{bti} \right)\right)\left(e^{bti} - 1\right) \\ \sum_{i=1}^{n} 2\left(yi - a * \left( 1 - e^{bti} \right)\right) * a * ti * e^{bti} \end{pmatrix}$$

**2.**

Voici les fonctions Erreur.m et ∇Erreur.m écrites sur MatLab:

```
function e=erreur(x)
a=x(1);b=x(2);
load partie3.mat;
n=length(y);
y2=f(a,b);
x=norm(y-y2,2)^2;
e=log(x);

end
```

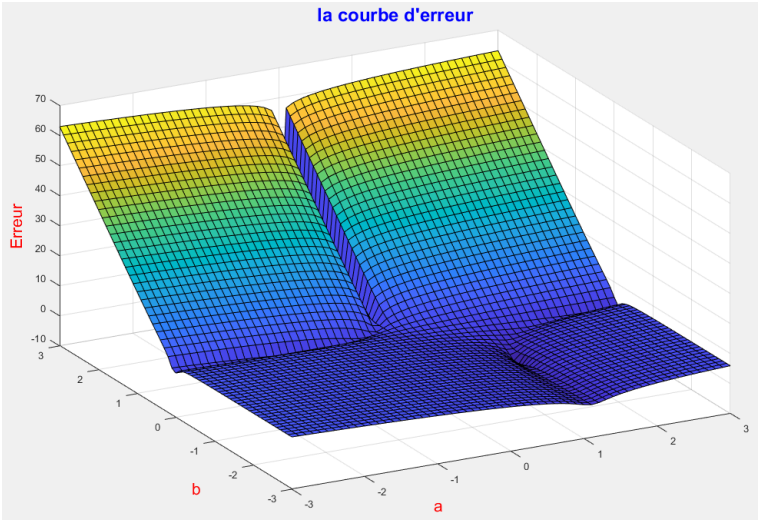
```
function [grad]=grad_erreur(x)
a=x(1);b=x(2);
load partie3.mat;
n=length(y);
y2=f(a,b);

norme=0;
s1=0;
s2=0;

for i=1:n
    norme=norme+(y(i)-a*(1-exp(b*t(i))))^2;
    com=2*(y(i)-a*(1-exp(b*t(i))));
    s1=s1-(1-exp(b*t(i)))*com;
    s2=s2+a*t(i)*exp(b*t(i))*com;
end
if (norme==0)
    disp('norme nulle');
    return;
end
grad=[s1/norme;s2/norme];
end
```

## 3.

Voici le traçage des courbes:

- En 3D:

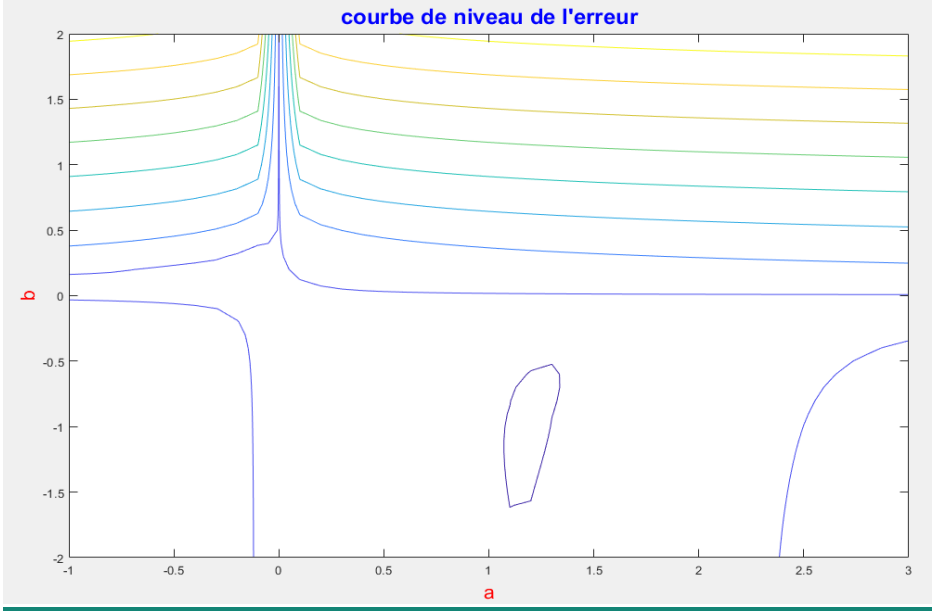
```
function tracageErreur()
    [X,Y]=meshgrid(-3:0.1:3,-3:0.1:3);
    Z=arrayfun(@erreur3,X,Y);
    surf(X,Y,Z);
    t=xlabel('a'); t.Color="red"; t.FontSize=16;
    t=ylabel('b'); t.Color="red"; t.FontSize=16;
    t=zlabel('Erreur'); t.Color="red"; t.FontSize=16;
    t=title("la courbe d'erreur"); t.Color="blue"; t.FontSize=18;
end
```




- En utilisant contour:

```
function tracageContourErreur()
    [X,Y]=meshgrid(-3:0.1:3,-3:0.1:3);
    Z=arrayfun(@erreur3,X,Y);
    contour(X,Y,Z);
    t=xlabel('a'); t.Color="red"; t.FontSize=16;
    t=ylabel('b'); t.Color="red"; t.FontSize=16;
    t=zlabel('Erreur'); t.Color="red"; t.FontSize=16;
    t=title("courbe de niveau de l'erreur"); t.Color="blue"; t.FontSize=18;
end
```

L’execution du code donne :




3 –a ) Non elle n’est pas convexe car $< grad(E)(x) - grad(E)(y), x - y > < 0$

On peut le montrer en passant par la hessiene de E(x) et en montrant qu’elle n’est pas definie positive .

5 )

a ) Methode gradient a pas fixe :

```
function [x] = gradpasfixe(r,epsilon)
x0=[-2;-1];
x=x0-r*grad_erreur(x0);
iteration=0;

while (norm(x-x0,2)>epsilon)&&(iteration<100001)
    x0=x;
    x=x0-r*grad_erreur(x0);
    iteration=iteration+1
    x
end

x(3)=iteration;
end
```

b) Methode d'inertie :

```
function [x] = ine(mu,r,eps)
x0=[-2;-1];
v0=x0;
v=mu*v0+r*grad_erreur(x0);
x=x0-v;
iteration=0;

while (norm(x-x0,2)>eps)&&(iteration<100002)
    x0=x;v0=v;
    v=mu*v0+r*grad_erreur(x0);
    x=x0-v;
    iteration=iteration+1;
end
x(3)=iteration;
end
```

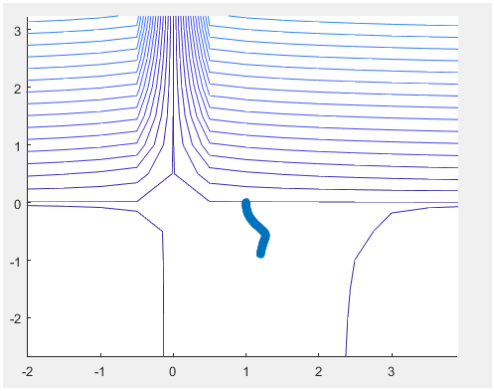
C) Methode gradient acceleré de Nestrov :

```
function [x] = gradaccnesterov(mu,r,eps)
x0=[-2;-1];
v0=x0;

v=mu*v0+r*grad_erreur(x0);
x=x0-v;
iteration=0;
while (norm(x-x0,2)>eps)&&(iteration<10000)
    x0=x;
    v0=v;
    v=mu*v0+r*grad_erreur(x0-mu*v0);
    x=x0-v;
    iteration=iteration+1;
end
x(3)=iteration;
end
```

6 )

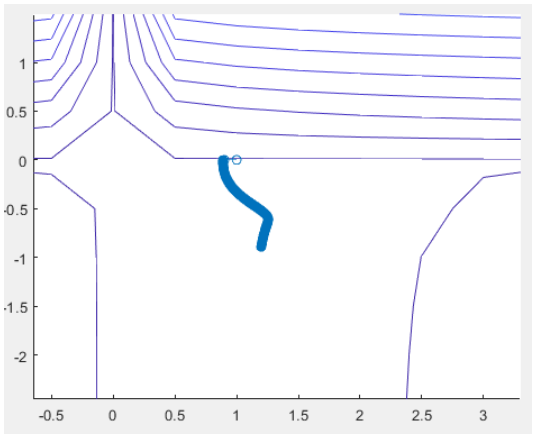
**Methode de gradient a pas fixe :**

```
X=gradpasfix([1;0],0.0001,0.00001,t,y);
x1=X(1:1,:);
x2=X(2:2,:);
hold on
contour(A,B,M,50);
plot(x1,x2,"--o");
hold off
```

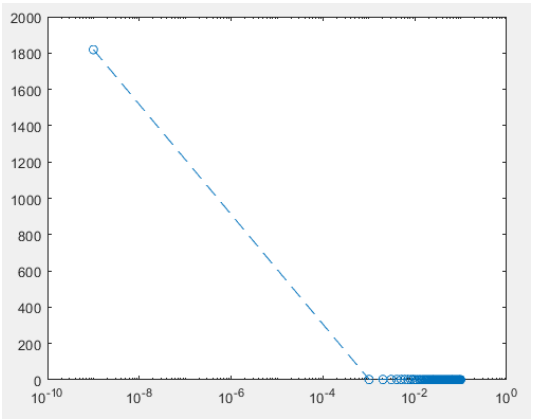



**Methode d'inertie :**

```
X=inertie([1;0],[1;0],0.0001,.1,0.00001,t,y);
x1=X(1:1,:);
x2=X(2:2,:);
hold on
contour(A,B,M,50);
plot(x1,x2,"--o");
```



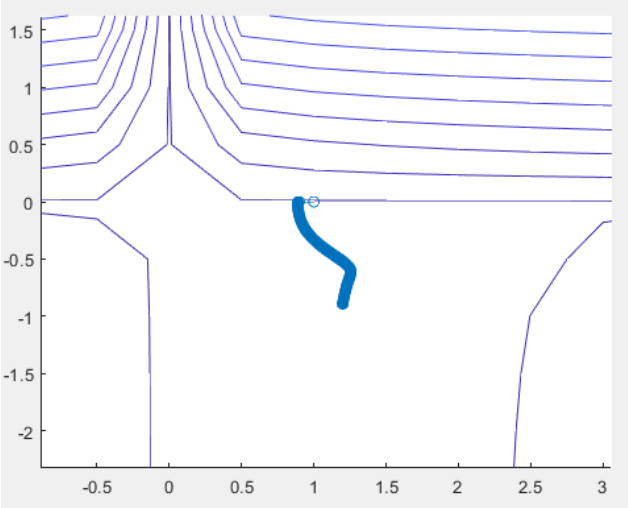

Iteration = f(epsilon)



## c) Méthode du gradient accélère de Nesterov :

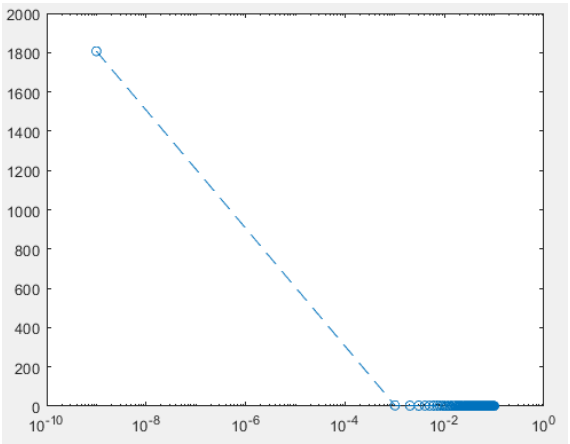
```
X=gradaccele([1;0],[1;0],0.0001,0.1,0.00001,t,y)
x1=X(1:1,:);
x2=X(2:2,:);
hold on
contour(A,B,M,50);
plot(x1,x2,"--o");
```

l'execution du programme donne ceci :




nombre d'iteration = f(epsilon)

```
N=zeros(1,100);
epsilon=linspace(0.000000001,0.1,100);
for i=1:100
    %N(i)=gradpasfix([1;0],0.0001,epsilon(i),t,y);
    %N(i)=inertie([1;0],[1;0],0.0001,0.01,epsilon(i),t,y);
    N(i)=gradaccele([1;0],[1;0],0.0001,0.1,epsilon(i),t,y);
end
semilogx(epsilon,N,"--o");
```



7 ) a ) **Pour le point de départ $x^0$=[-2,-1] et les paramètre ($v^0$=grad($x^0$) ; η=0.09 ; ρ =0.00035 ; ε =$10^{-6}$ ) .**

```
function algorithme_le_plus_rapide()
    % initialisation des paramètres:
    X0=[-2,-1]; V0=grad_erreur(X0); m=0.09; roo=0.00035; epsilon=0.000001;
    X1=gradPasFixe(X0,roo,epsilon);
    X2=methode_inertie(X0,V0,m,roo,epsilon);
    X3=gradAccelereNesterov(X0,V0,m,roo,epsilon);
    nombreIteration_gradPasFixe=length(X1);
    nombreIteration_methode_inertie=length(X2);
    nombreIteration_gradAccelereNesterov=length(X3);
    disp("le nombre d'itération de la méthode de gradient à pas fixe est: ");
    disp(nombreIteration_gradPasFixe);
    disp("le nombre d'itération de la méthode d'_inertie est: ");
    disp(nombreIteration_methode_inertie);
    disp("le nombre d'itération de la méthode de gradient accéléré est: ");
    disp(nombreIteration_gradAccelereNesterov);


end
```

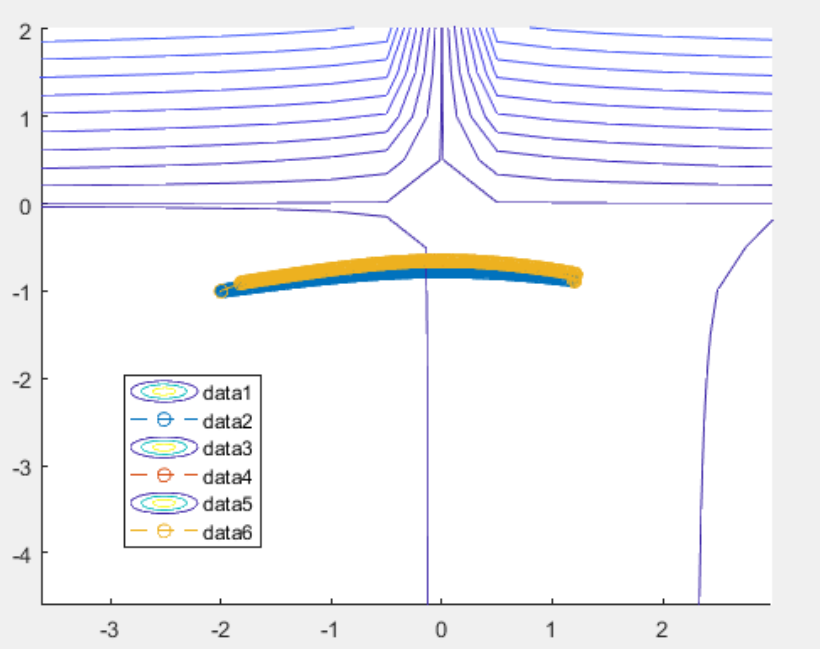
L'execution du code :

```
>> algorithme_le_plus_rapide()
le nombre d'itération de la méthode de gradient à pas fixe est:
        7454

le nombre d'itération de la méthode d'_inertie est:
        6530

le nombre d'itération de la méthode de gradient accéléré est:
        6530
```

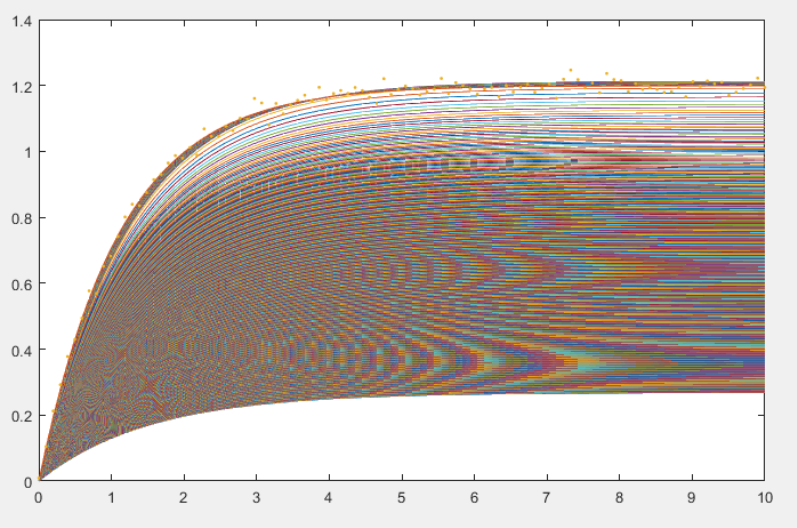
**On voit bien que le notre d'iteration du methode de gradpasfix est superieur a celle de la methode d'inertie et gradient accelere donc  ces deux dernieres sont plus rapides** ,

7-b)

Ici la courbe en bleu est celle du methode gradient a pas fixe on peut remarquer que les deux courbes inertie et nestrov sont quasiment confondus



7 – c ) en faisant plusieurs iterations de la methode d'inertie on obtient cette courbe ci dessous :




```
load partie3.mat;
X=inertie([-2;-1],[-2;-1],0.0004,.09,0.00001,t,y);
for i=4700:5310
    Y=zeros(102,1);
    for j=1:102
        Y(j)=X(1,i)*(1-exp(X(2,i)*t(j)));
    end
    plot(t,Y);
    hold on;
end
plot(t,y,'.');
```

Conclusion :

Suite a ce projet on peut conclure qu'il existe plusieurs methodes pour resoudre les problemes de regression  on on a mis en evidence que si le probleme est d'optimiser sans contraintes alors la methode de newton l'emporte (c'est la meilleure ) et s'il s'agit de sous contraine (la methode de gradient accelerée Nestrov , et la methode d'inertie l'emportent sur la methode de grad a pas fixe ) .