Projet d'Optimisation Continue

Table of Contents

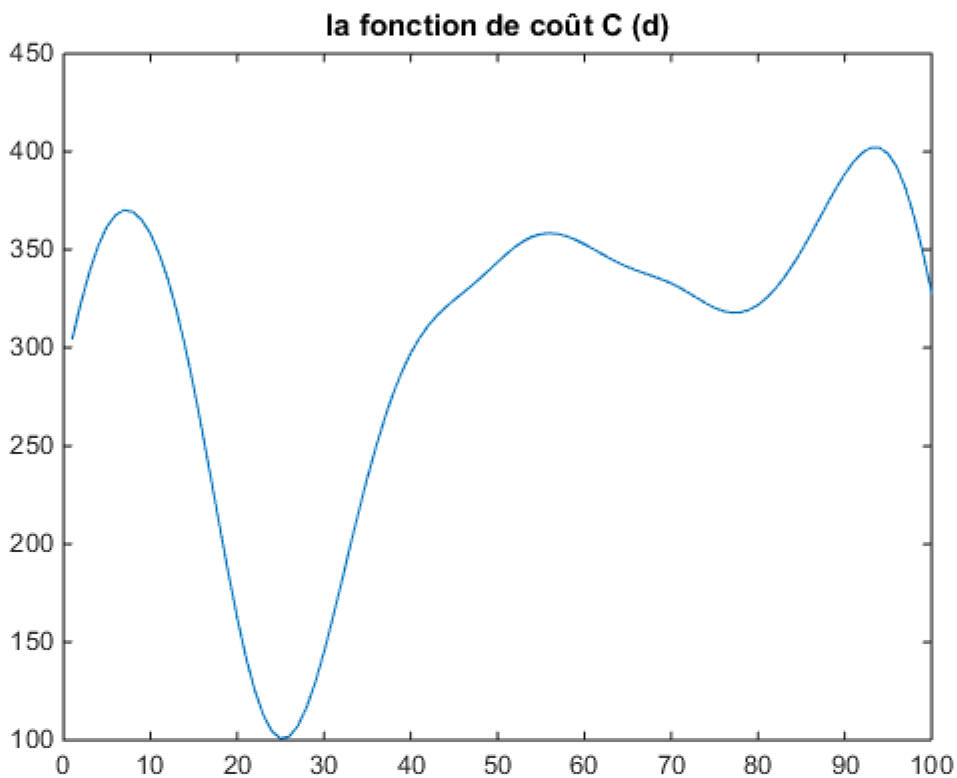
Groupe A: YAN Yutong & ZHANG Heng

```
close all;
clear;
warning('off','all');
load('data.mat');
```

Q2

```
t = 1:100;
d = 1:100;
result = Unparametre(t, sig_noisy, d);
figure()
plot(d,result)
title('la fonction de coût C (d)')

dbtype('Unparametre');

% Ici on applique un fonction 'Unparametre' qui sert à représenter la
% fonction C(d).
% Dans la fonction 'sum((y-exp(-(t-d).^2./(((1+s^2)^2)*2)).*a^2).^2)'
% retourne la valeur de la fonction C
```

```
function result = Unparametre(t, y, d)
    s = 2;
    a = 2;
    a2 = a^2;
    s2 = ((1+s^2)^2)*2;
    result = zeros(size(d));
    for i = 1:size(d,2)
        result(i) = sum((y-exp(-(t-d(i)).^2./s2).*a2).^2);
    end
end
```

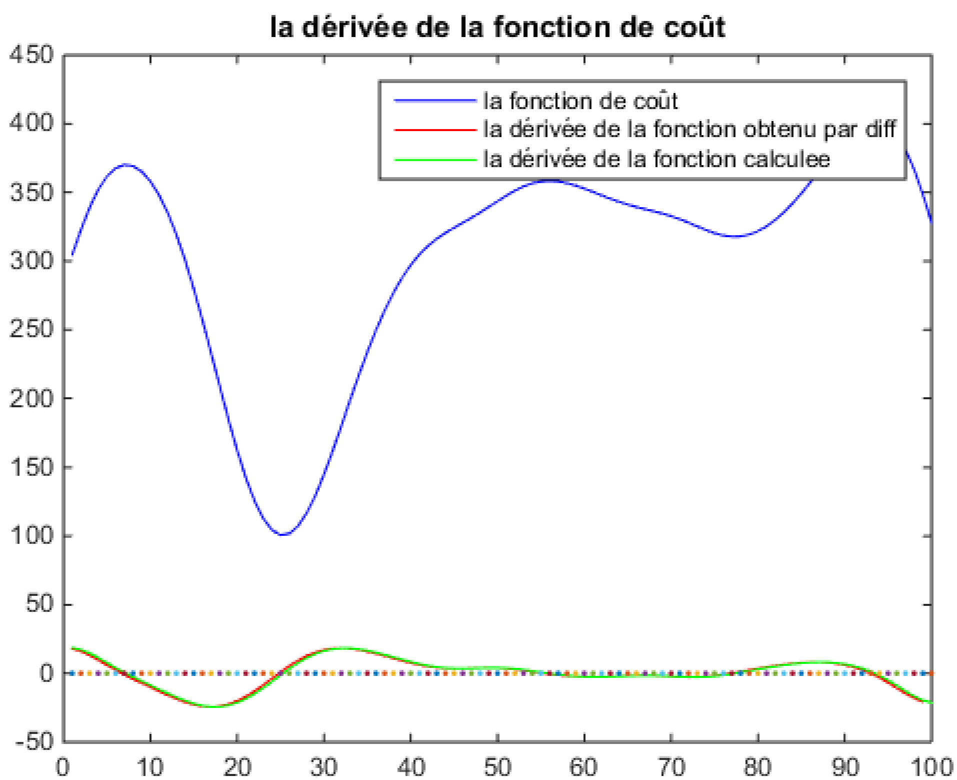


Q4

```
result_diff = diff(result);
reultat_UnparametreDerivee = UnparametreDerivee(t, sig_noisy, d);
figure()
plot(d,result,'b',d(:,1:length(result_diff)),result_diff,'r', d, reultat_Unparamet
title('la dérivée de la fonction de coût');
legend('la fonction de coût','la dérivée de la fonction obtenu par diff','la dériv

dbtype('UnparametreDerivee');

% Premièrement on applique la fonction diff() pour trouver la dérivée
% de la fonction C(d).
% Deuxièment on applique la fonction 'UnparamètreDerivee' qu'on a
```

```
% écrit pour calculer la dérivée.
% Finalement on affiche les résultats des deux fonctions. On peut
% voir qu'ils sont correspondants.
% Le calcul de la dérivée est présenté sur la rapport sur le papier
```

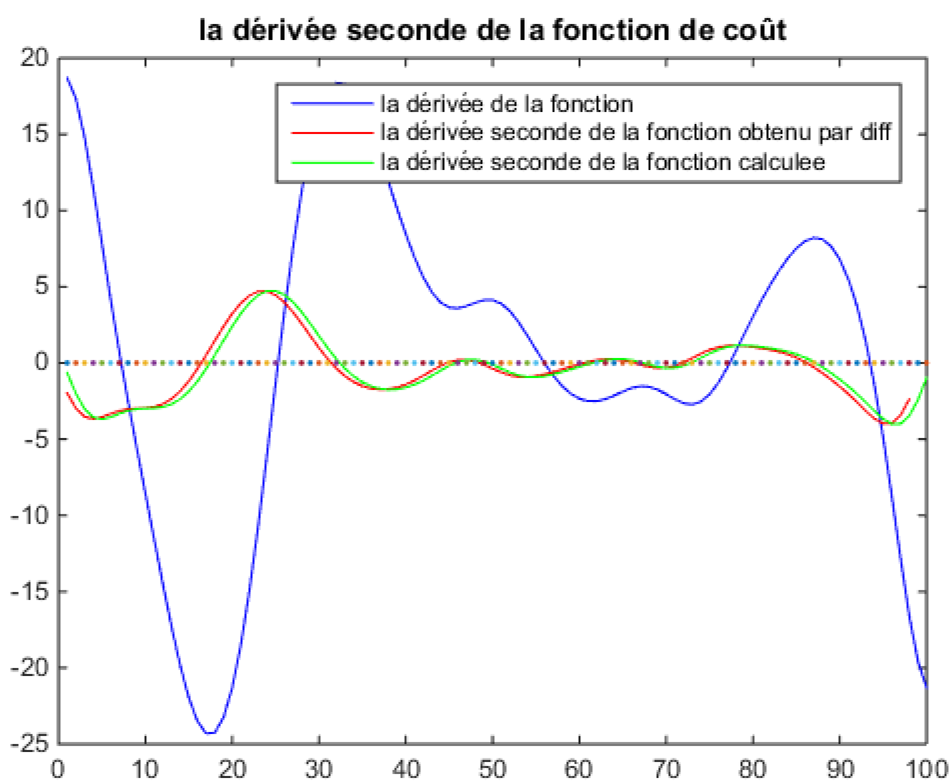
```
function result = UnparametreDerivee(t, y, d)
    s = 2;
    a = 2;
    a2 = a^2;
    s2 = ((1+s^2)^2)*2;
    s3 = s2./2;
    result = zeros(size(d));
    for i = 1:size(d,2)
        result(i) = sum(4*a2/s2.*(-y.*(t-d(i)).*exp(-(t-d(i)).^2./s2)+a2.*(t
    end
end
```



Q6

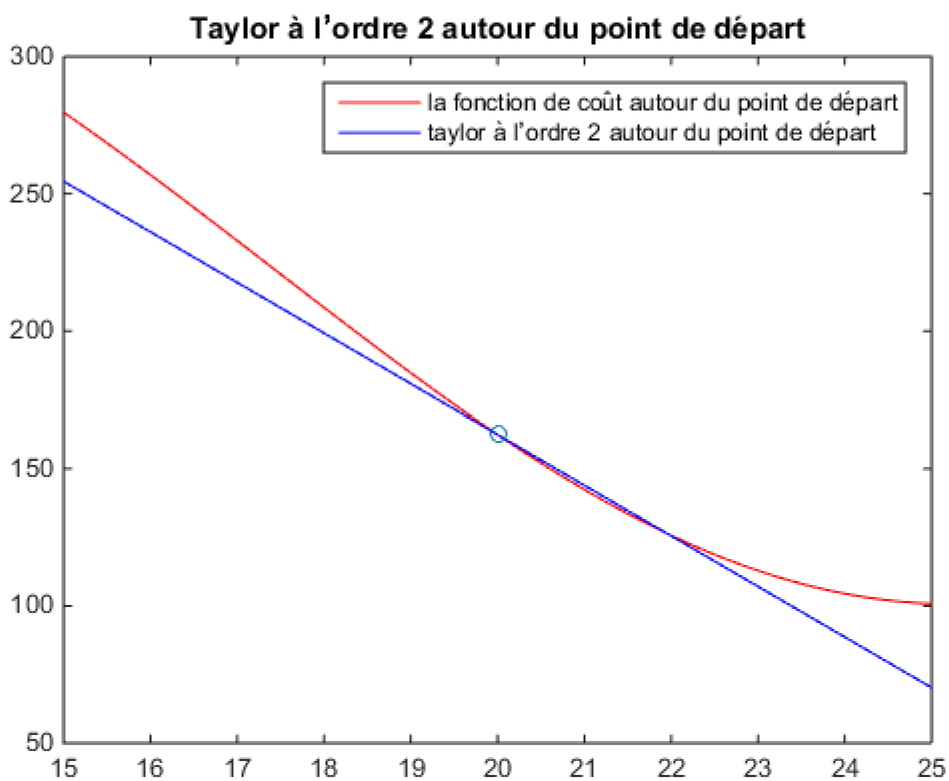
```
result_diff_diff = diff(result_diff);
resultat_UnparametreDeriveeSecond = UnparametreDeriveeSecond(t,sig_noisy, d);
figure()
plot(d,reultat_UnparametreDerivee,'b',d(:,1:length(result_diff_diff)),result_diff_
title('la dérivée seconde de la fonction de coût');
legend('la dérivée de la fonction','la dérivée seconde de la fonction obtenu par d
```

```
dbtype('UnparametreDeriveeSecond');

% Premièrement on applique la fonction diff() pour trouver la
% dérivée seconde de la fonction C(d).
% Deuxièment on applique la fonction 'UnparametreDeriveeSecond'
% qu'on a écrit pour calculer la dérivée seconde.
% Finalement on affiche les résultats des deux fonctions. On peut
% voir qu'ils sont correspondants.
% Le calcul de la dérivée seconde est présenté sur la rapport sur
% le papier


function result = UnparametreDeriveeSecond(t, y, d)
    s = 2;
    a = 2;
    a2 = a^2;
    s2 = ((1+s^2)^2)*2;
    s3 = (1+s^2)^2;
    result = zeros(size(d));
    for i = 1:size(d,2)
        p1 = -(t-d(i)).^2./s2;
        p2 = -(t-d(i)).^2./s3;
        result(i) = sum(2*a2/s3.*(((t-d(i)).^2./s3).*(-y.*exp(p1)+2.*a2.*exp
    end
end
```



Q7

```
depart = randi([min(d)+5,max(d)-5]);
deplace = -5:0.1:5;
taylorOrdre2 = result(depart) + reultat_UnparametreDerivee(depart).*deplace +(resu
result_deplace = Unparametre(t, sig_noisy, deplace+depart);
figure();
plot(deplace+depart,result_deplace,'r', deplace+depart,taylorOrdre2,'b',depart, re
title('Taylor à l’ordre 2 autour du point de départ');
legend('la fonction de coût autour du point de départ','taylor à l’ordre 2 autour

% On applique la fonction randi pour obtenir un entier au hasard
% entre 5 et 95 comme le point de départ.
% Le déplacement autour du point de départ est de -5 à 5 avec
% pas de 0.1.
% On représente le Taylor à l'ordre 2 avec la fonction, sa dérivée
% et sa dérivée seconde.
% Puis on trouve la fonction de cout entre -5 et 5 pour mieux
% afficher le résultat.
```

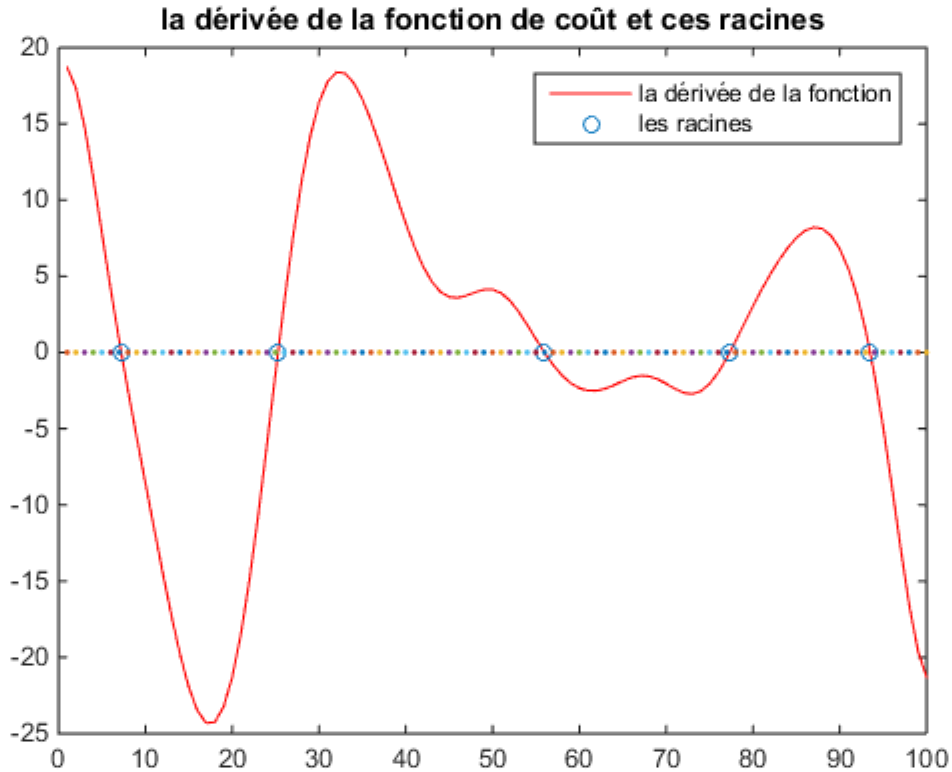


Q8

```
precision = 10^(-15);
depart_Newtons = [10,20,30,40,55,60,80,90];
racine = zeros(size(depart_Newtons));
```

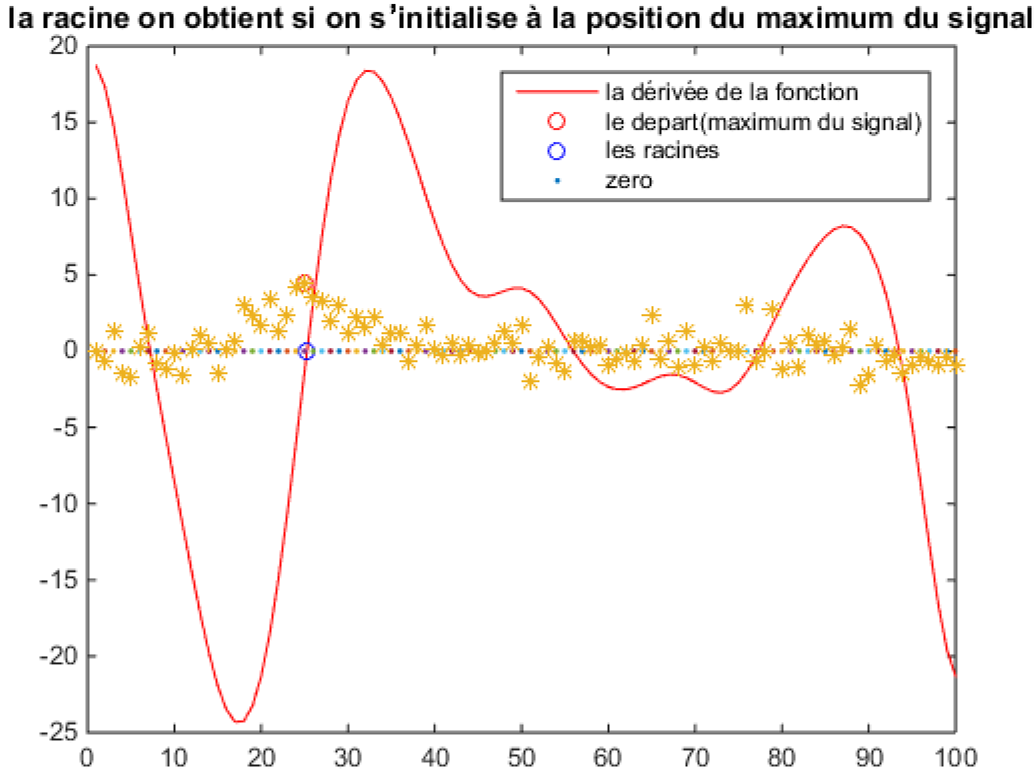
```
for i = 1:size(depart_Newtons,2)
depart_Newton = depart_Newtons(i);
x_Newton = depart_Newton - UnparametreDerivee(t, sig_noisy, depart_Newton)/Unparam
while(abs(x_Newton - depart_Newton) > precision)
depart_Newton = x_Newton;
x_Newton = depart_Newton - UnparametreDerivee(t, sig_noisy, depart_Newton)/Unparam
end
racine(i) = x_Newton;
end
figure(5)
plot(d, reultat_UnparametreDerivee, 'r',racine, UnparametreDerivee(t, sig_noisy, r
title('la dérivée de la fonction de coût et ces racines')
legend('la dérivée de la fonction','les racines')
fprintf('les racines : ');
disp(racine);

% On a premièrement défini la précision de la méthode Newton:
% 10^(-15).
% Puis on a choisi 10,20,30,40,55,60,80,90 comme les points de départ.
% Après on applique la méthode Newton pour chaque point de départ.
% Dans la méthode Newton, il y a une itération pour approcher
% les racines.
% Finalement on affiche tous les racines qu'on a trouvé sur la figure.
```

```
les racines :   Columns 1 through 7

    7.2171   25.2786   25.2786   77.3195   55.9628       NaN   77.3195

  Column 8

   93.5087
```



Q9

```
for i = 1:size(sig_noisy,2)
    if(sig_noisy(i) == max(sig_noisy))
        break
    end
end
depart_Newton = i;
x_Newton = depart_Newton - UnparametreDerivee(t, sig_noisy,depart_Newton)/Unparame
while(abs(x_Newton - depart_Newton) > precision)
    depart_Newton = x_Newton;
    x_Newton = depart_Newton - UnparametreDerivee(t, sig_noisy,depart_Newton)/Unpa
end
plot(d, reultat_UnparametreDerivee, 'r',i,sig_noisy(i),'ro',x_Newton,UnparametreDe
title('la racine on obtient si on s’initialise à la position du maximum du signal'
legend('la dérivée de la fonction','le depart(maximum du signal)','les racines','z

% Premièrement on trouve l'index de la maximum du signal bruité,
% qui est(25,4.3879).
% On utilise ce point comme le point de départ de la méthode Newton.
% Et on retrouve une racine 25.2786 qui est le plus proche de ce point
% de départ.
% Finalement on affiche la fonction dérivée, le point de départ et le
% racine qu'on a trouvé pour mieux observer le résultat.
```

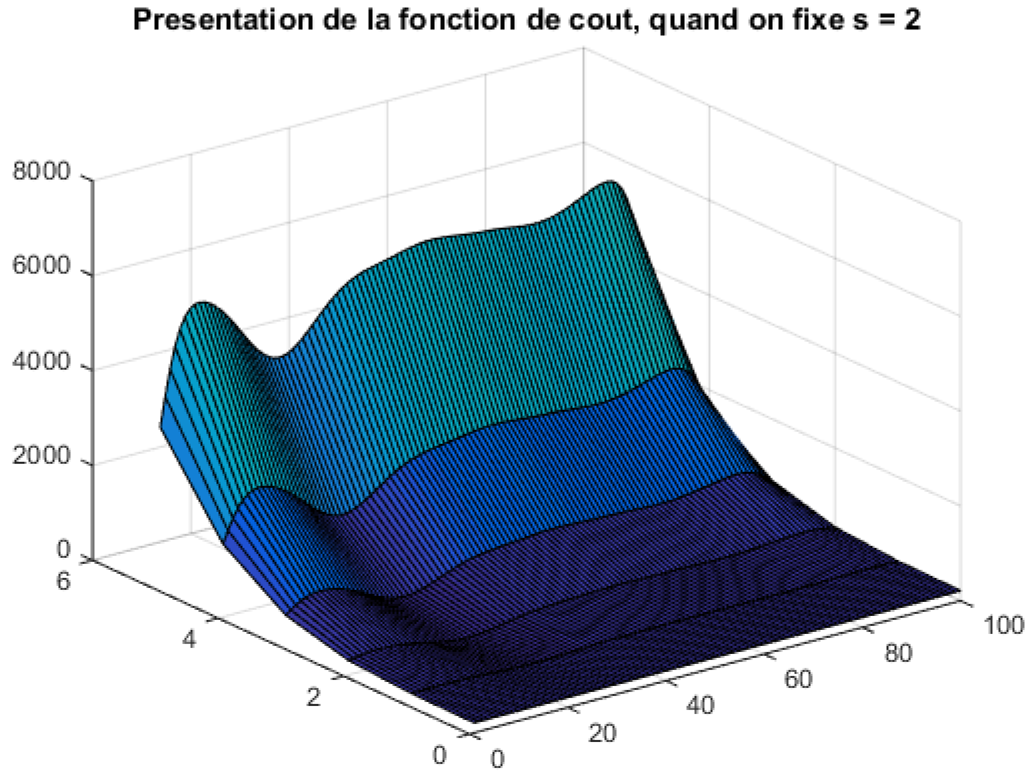


Q10

```
% Presentation de la fonction de cout

t = 1:100;
```

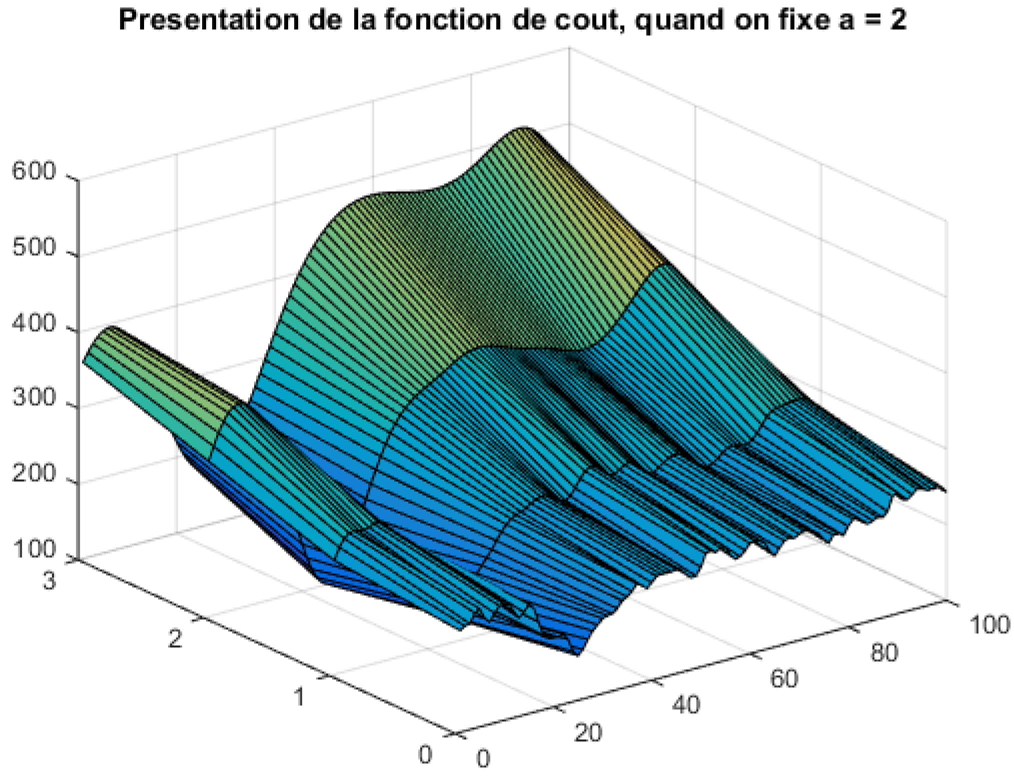
Presentation de la fonction de cout, quand on fixe s = 2

```
d = 1:100;
a = 0:5;
s = 2;
result = zeros(size(d, 2), size(a, 2));
for i = 1:size(d,2)
    for j = 1:size(a,2)
        result(i,j) = Troisparametre(t,sig_noisy,[d(i),a(j),s]);
    end
end
figure(1);
surf(d,a,result');
title('Presentation de la fonction de cout, quand on fixe s = 2');
```



Presentation de la fonction de cout, quand on fixe a = 2

```
d = 1:100;
a = 2;
s = 0:3;
result = zeros(size(d, 2), size(s, 2));
for i = 1:size(d,2)
    for j = 1:size(s,2)
        result(i,j) = Troisparametre(t,sig_noisy,[d(i),a,s(j)]);
    end
end
figure(2);
surf(d,s,result');
title('Presentation de la fonction de cout, quand on fixe a = 2');
```

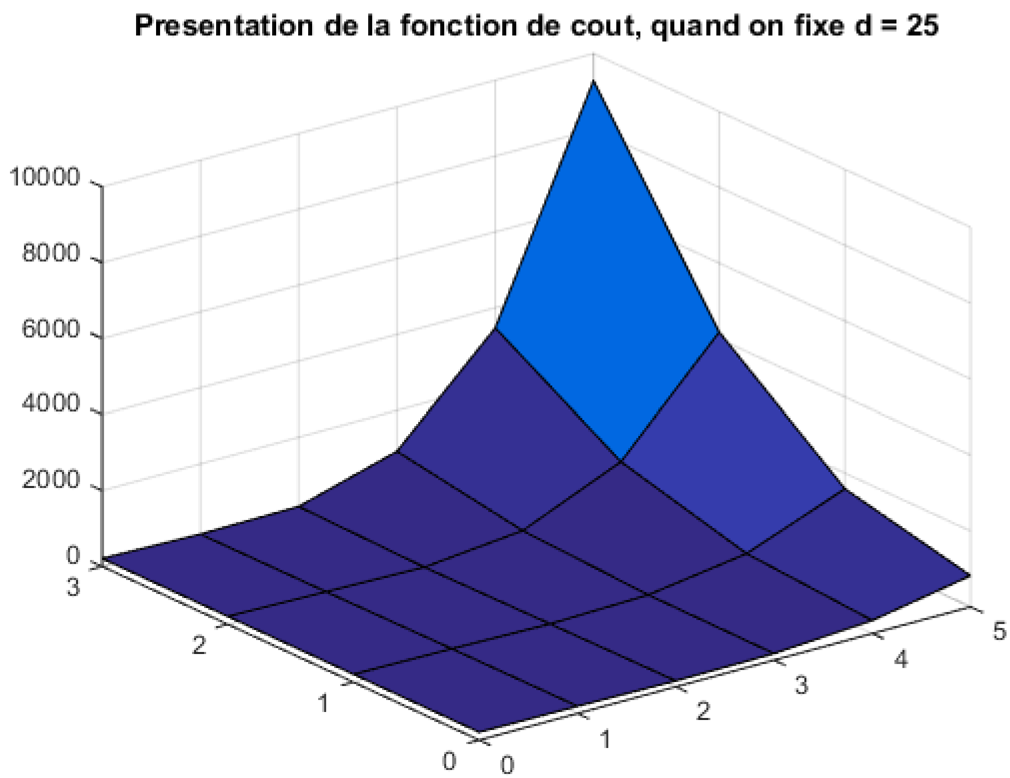


Presentation de la fonction de cout, quand on fixe d = 25

```
d = 25;
a = 0:5;
s = 0:3;
result = zeros(size(a, 2), size(s, 2));
for i = 1:size(a,2)
    for j = 1:size(s,2)
        result(i,j) = Troisparametre(t,sig_noisy,[d,a(i),s(j)]);
    end
end
figure(3);
surf(a,s,result');
title('Presentation de la fonction de cout, quand on fixe d = 25');

dbtype Troisparametre.m;
```

```
function result = Troisparametre(t, y, x)
    d = x(1);
    a = x(2);
    s = x(3);
    a2 = a.^2;
    s2 = ((1+s.^2).^2).*2;
```

```
        result = sum((y-exp(-(t-d).^2./s2).*a2).^2);
    end
```



Q11

```
% variation de a,d,s en meme temps
d = 24:0.1:26;
a = 1:0.01:3;
s = 1:0.01:3;

% initialisation de l'algo
minimum = inf;
dd = 0;
aa = 0;
ss = 0;

% Iteration pour trouver le minimum et le theta correspontant
for i = 1:size(d,2)
    for j = 1:size(a,2)
        for k = 1:size(s, 2)
            if(minimum > Troisparametre(t,sig_noisy,[d(i),a(j),s(k)]))
                minimum = Troisparametre(t,sig_noisy,[d(i),a(j),s(k)]);
                dd = d(i);aa = a(j);ss = s(k);
            end
        end
    end
```

```
end

% Specification du resultat
fprintf('Le minimum = %f\n', minimum);
fprintf('d = %f\n', dd);
fprintf('a = %f\n', aa);
fprintf('s = %f\n', ss);
```

```
Le minimum = 98.516376
d = 25.400000
a = 1.850000
s = 2.090000
```

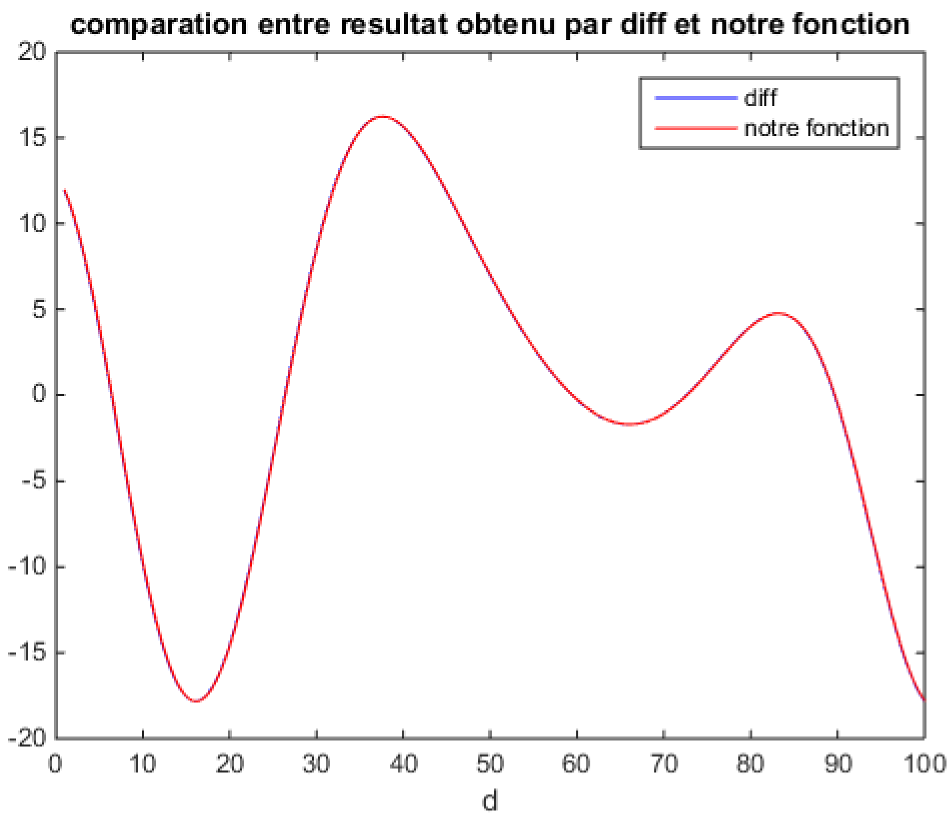
Q14

d

```
% initialisation de l'algo
t = 1:100;
h = 0.1;
d = 1:0.1:100;
a = 2;
s = 3;
result = zeros(size(d, 2),1);
diffResult = zeros(size(d, 2),1);

% iteration pour calculer la gradient de fonction de cout
for i = 1:size(d,2)
    result(i) = Troisparametre(t,sig_noisy,[d(i),a,s]);
    diffResult(i) = GradientDeFonctionDeCoutD(t,sig_noisy,[d(i), a, s]);
end

% calculer la gradient de fonction de cout en appliquant 'diff'
diffResult2 = diff(result)/h;

% Presentation de la comparation entre resultat obtenu
% par 'diff' et notre fonction
figure;
plot(d(1,1:end-1), diffResult2,'b',d, diffResult,'r');
xlabel('d');
legend('diff','notre fonction');
title('comparation entre resultat obtenu par diff et notre fonction');

diffResult2 = [diffResult2; diffResult(end)];
fprintf('Le erreur relative maximale est %f\n', max(abs((diffResult2 - diffResult)
dbtype GradientDeFonctionDeCoutD.m;
```

```
Le erreur relative maximale est 11.621487

      function result = GradientDeFonctionDeCoutD(t, y, x)
          d = x(1);
          a = x(2);
```

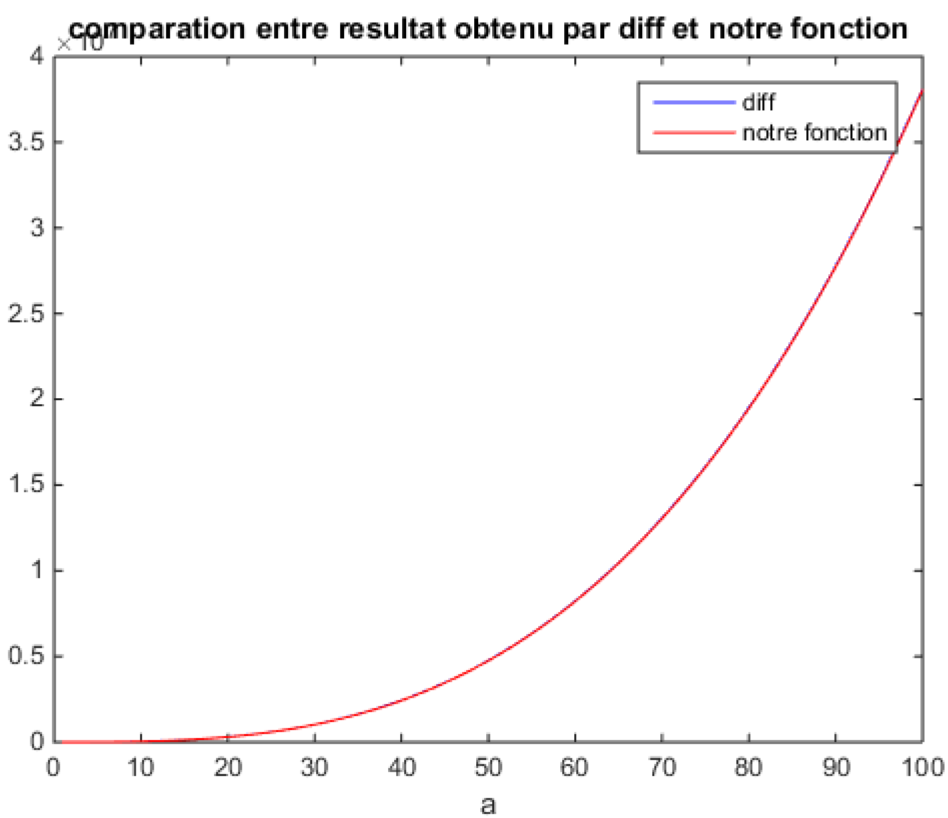
```
        s = x(3);
        a2 = a.^2;
        s2 = ((1+s^2)^2)*2;
        s3 = s2./2;
        result = sum(4*a2/s2.*(-y.*(t-d).*exp(-(t-d).^2./s2)+a2.*(t-d).*exp(-(t-
    end
```



a

```
% initialisation de l'algo
t = 1:100;
d = 25.4000;
h = 0.1;
a = 1:0.1:100;
s = 2.0900;
result = zeros(size(a, 2),1);
diffResult = zeros(size(a, 2),1);

% iteration pour calculer la gradient de fonction de cout
for i = 1:size(a,2)
    result(i) = Troisparametre(t,sig_noisy,[d,a(i),s]);
    diffResult(i) = GradientDeFonctionDeCoutA(t,sig_noisy,[d,a(i),s]);
end

% calculer la gradient de fonction de cout en appliquant 'diff'
diffResult2 = diff(result)/h;
```

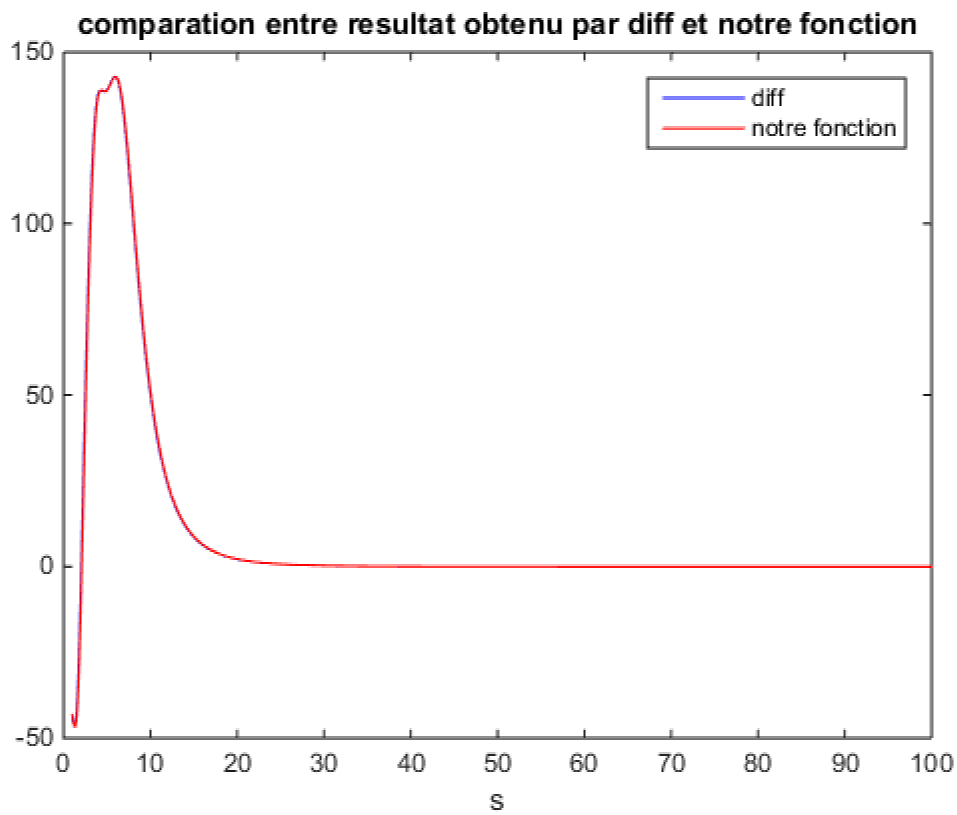
```
% Presentation de la comparation entre resultat obtenu
% par 'diff' et notre fonction
figure;
plot(a(1,1:end-1), diffResult2,'b',a, diffResult,'r');
xlabel('a');
legend('diff','notre fonction');
title('comparation entre resultat obtenu par diff et notre fonction');

diffResult2 = [diffResult2; diffResult(end)];
fprintf('Le erreur relative maximale est %f\n', max(abs((diffResult2 - diffResult)

dbtype GradientDeFonctionDeCoutA.m;

Le erreur relative maximale est 67.522569

function result = GradientDeFonctionDeCoutA(t, y, x)
    d = x(1);
    a = x(2);
    s = x(3);
    a2 = a^2;
    s2 = ((1+s^2)^2)*2;
    result = -4.*sum(a.*(y-a2.*exp(-(t-d).^2./s2)).*exp(-(t-d).^2./s2));
end
```



s

```
% initialisation de l'algo
t = 1:100;
d = 25.4000;
a = 1.8500;
h = 0.1;
s = 1:0.1:100;
result = zeros(size(s, 2),1);
diffResult = zeros(size(s, 2),1);

% iteration pour calculer la gradient de fonction de cout
for i = 1:size(s,2)
    result(i) = Troisparametre(t,sig_noisy,[d,a,s(i)]);
    diffResult(i) = GradientDeFonctionDeCoutS(t,sig_noisy,[d,a,s(i)]);
end

% calculer la gradient de fonction de cout en appliquant 'diff'
diffResult2 = diff(result)/h;

% Presentation de la comparation entre resultat obtenu
% par 'diff' et notre fonction
figure;
plot(s(1,1:end-1), diffResult2,'b',s, diffResult,'r');
xlabel('s');
legend('diff','notre fonction');
title('comparation entre resultat obtenu par diff et notre fonction');

diffResult2 = [diffResult2; diffResult(end)];
fprintf('Le erreur relative maximale est %f \n', max(abs((diffResult2 - diffResult

dbtype GradientDeFonctionDeCoutS.m;
```

```
Le erreur relative maximale est 1.287471

      function result = GradientDeFonctionDeCoutS(t, y, x)
          d = x(1);
          a = x(2);
          s = x(3);
          a2 = a^2;
          s2 = ((1+s^2)^2)*2;
          result = -4.*sum(a2.*s.*(y-a2.*exp(-(t-d).^2./s2)).*exp(-(t-d).^2./s2).*
      end
```



Q15

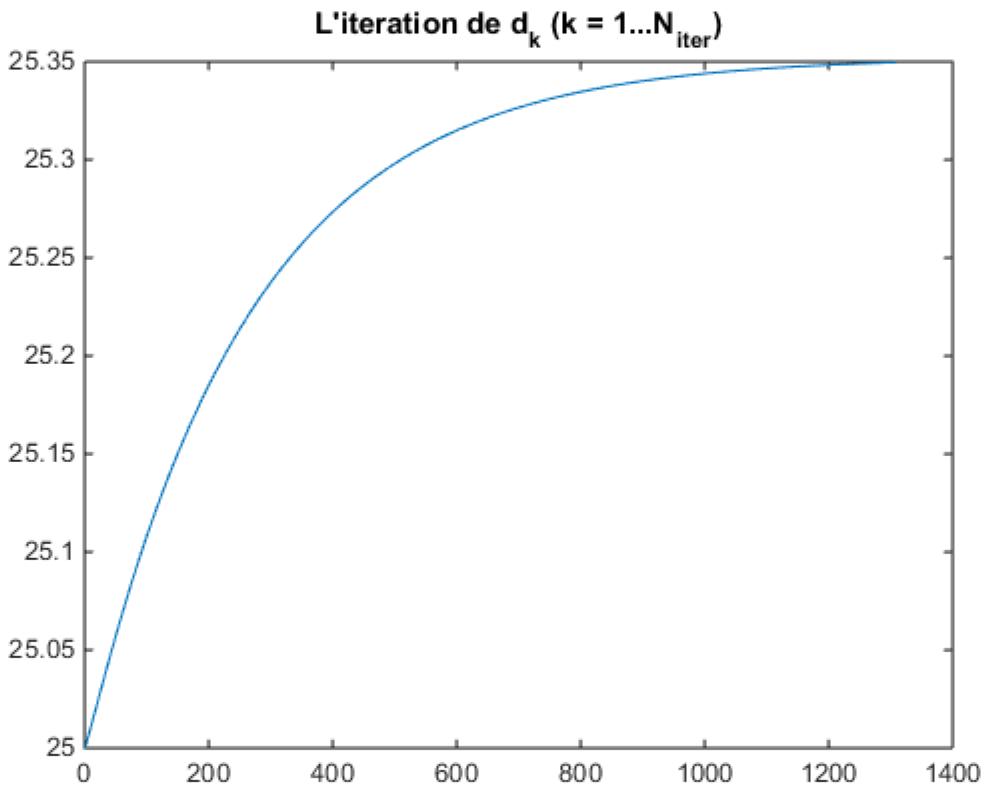
```
% Dans cette exercice on va appliquer
% la methode de des plus fortes pentes

% configuration initiale
epsilon = 10^-2;
x0 = [25; 2; 2];

% variables pour conserver le resultat
xk = x0;
xkList = [];
xkList = [xkList, xk];

% tant que la norme du gradient sera superieure a 10^-2
while norm(GradientDeFonctionDeCout(t, sig_noisy, xk)) > epsilon

    % calcul de dk

    dk = -GradientDeFonctionDeCout(t, sig_noisy, xk);

    % calcul de alpha

    alphal = 0;
    alphar = inf;
```

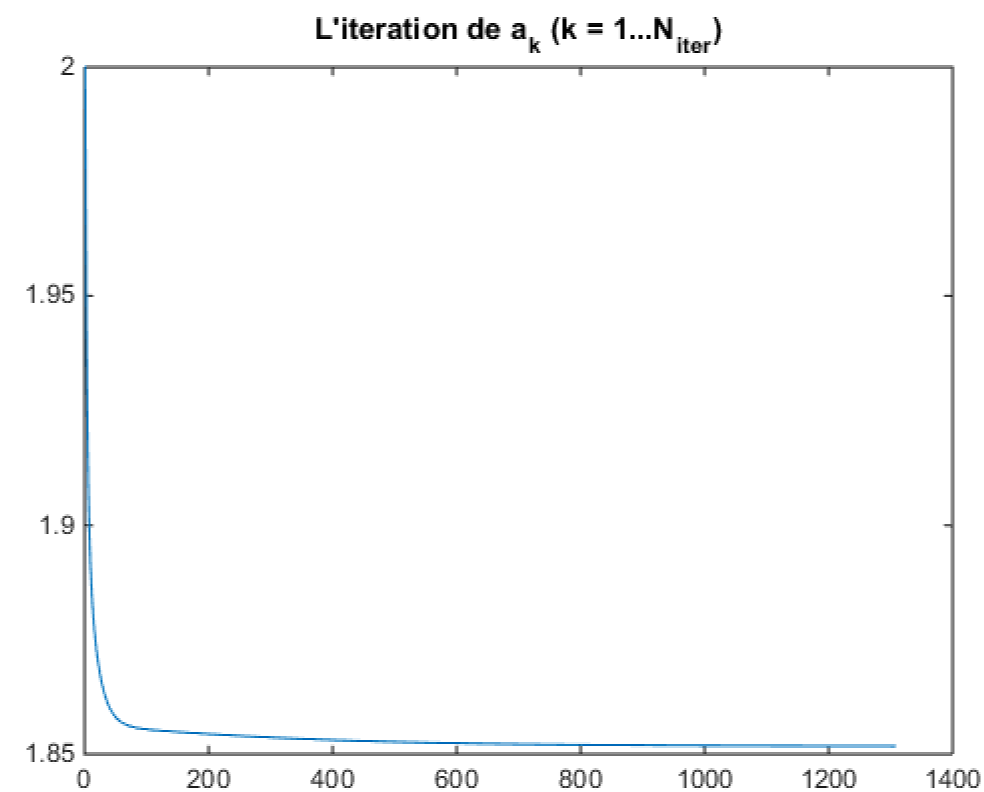
```
    alphai = 10^-3;
    beta1 = 10^-3;
    beta2 = 0.99;
    lambda = 20;
    alphak = alphai;

    while 1
        gamma = -beta1.*(GradientDeFonctionDeCout(t, sig_noisy, xk))'*dk;
        if(Troisparametre(t,sig_noisy,(xk+alphai.*dk)) > (Troisparametre(t,sig_noi
            alphar = alphai;
            alphai = (alphal+alphar)/2;
            continue;
        elseif (((GradientDeFonctionDeCout(t, sig_noisy, xk+alphai*dk)'*dk))/(Troi
            alphal = alphai;
            if alphar < inf
                alphai = (alphal+alphar)/2;
            else
                alphai = lambda*alphai;
            end
            continue;
        else
            break;
        end
    end
    alphak = alphai;

    % mis a jour xk
    xk = xk + alphak*dk;

    % mis a jour xkList

    xkList = [xkList, xk];

end

xkInf = xk;

% Presentation de l'iteration de theta_k (k = 1...N_{iter})
figure;
plot(1:size(xkList,2), xkList(1,:));
title('L''iteration de d_k (k = 1...N_{iter})');
figure;
plot(1:size(xkList,2), xkList(2,:));
title('L''iteration de a_k (k = 1...N_{iter})');
figure;
plot(1:size(xkList,2), xkList(3,:));
title('L''iteration de s_k (k = 1...N_{iter})');

% Presentation de l'ecart entre deux iteres successifs
xkEcart = [];
for i = 2:size(xkList,2)
    xkEcart = [xkEcart, norm(xkList(:,i)-xkList(:,i-1))];
end
figure;
```

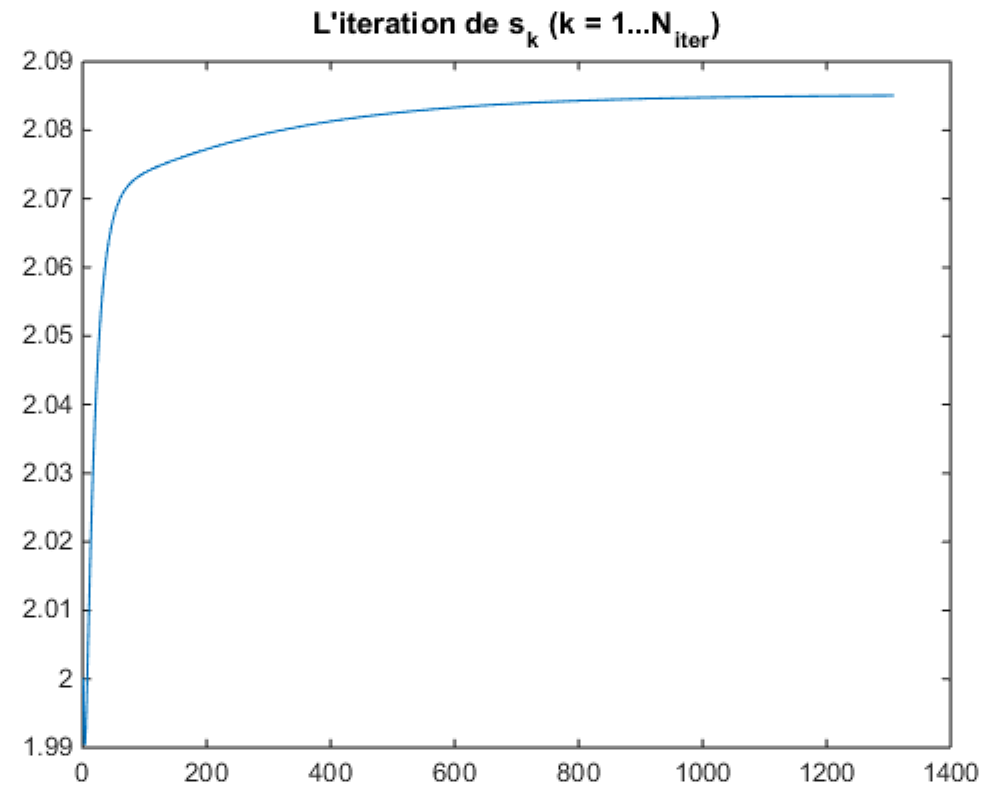
```
plot(1:size(xkEcart,2), xkEcart);
title('L''ecart entre deux iteres successifs, || \theta_k - \theta_{k-1} ||');

% presentation de l'ecart a l'optimum theta_inf
xkEcartInf = [];
for i = 1:size(xkList,2)
    xkEcartInf = [xkEcartInf, norm(xkList(:,i)-xkInf)];
end
figure;
plot(1:size(xkEcartInf,2), xkEcartInf);
title('L''ecart a l''optimum \theta_inf,  || \theta_k - \theta_{inf} ||');

% Presentation de l'ecart en terme de fonctions de cout
CEcart = [];
for i = 1:size(xkList,2)
    CEcart = [CEcart, norm(Troisparametre(t,sig_noisy,xkList(:,i))-Troisparametre(
end
figure;
plot(1:size(CEcart,2), CEcart);
title('L''ecart en terme de fonctions de cout,  || C(\theta_k) - C(\theta_{inf}) |

% la norme infinie du gradient
fprintf('la norme infinie du gradient est: %f\n', norm(GradientDeFonctionDeCout(t,

% l'optimum theta_inf
fprintf('l''optimum theta sont: d = %f, a = %f, s = %f\n', xkInf);

dbtype GradientDeFonctionDeCout.m;
```

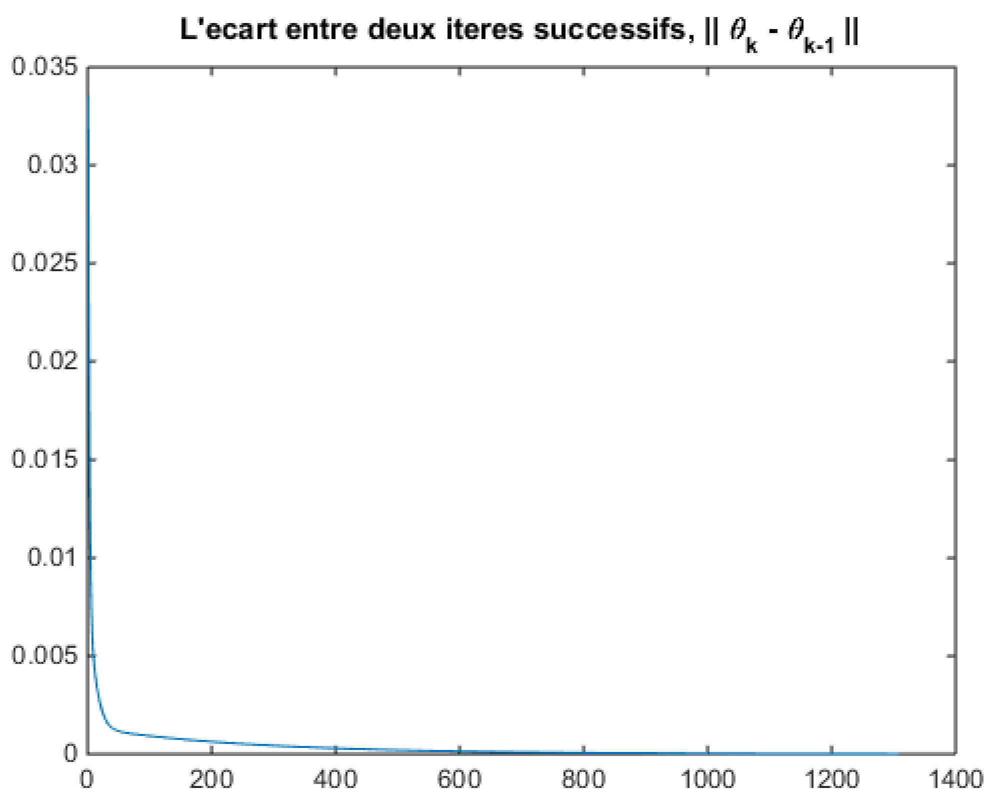
```
la norme infinie du gradient est: 0.009983
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054
```

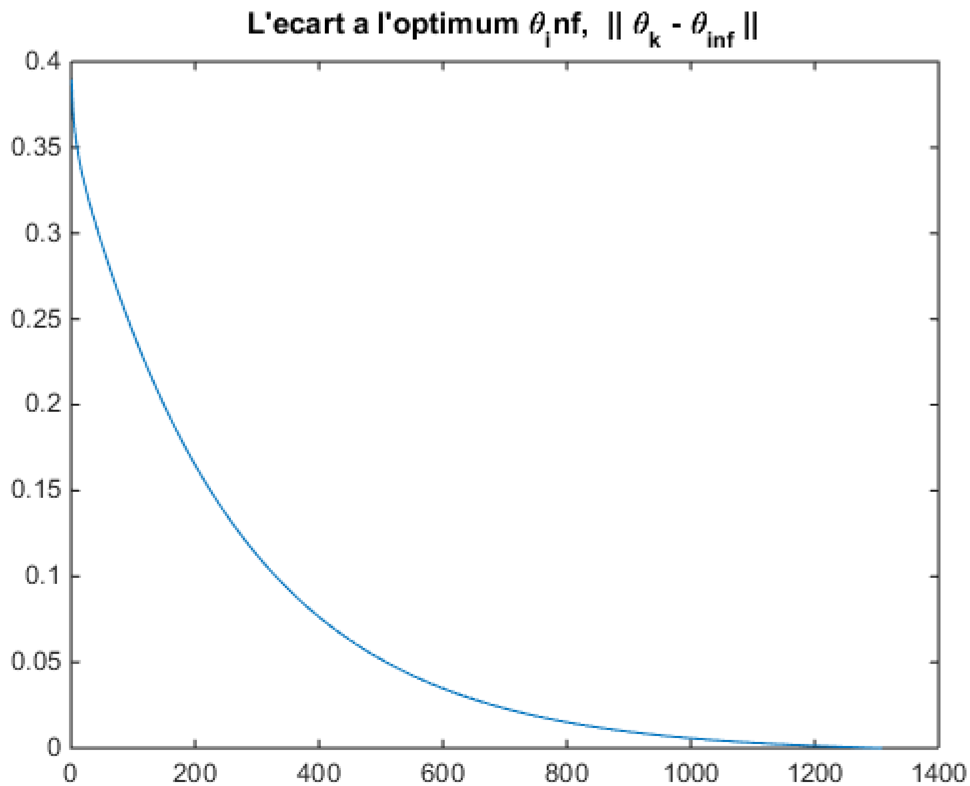
```
function result = GradientDeFonctionDeCout(t, sig_noisy, xk)
    result = [GradientDeFonctionDeCoutD(t,sig_noisy,xk); GradientDeFonctionD
end
```

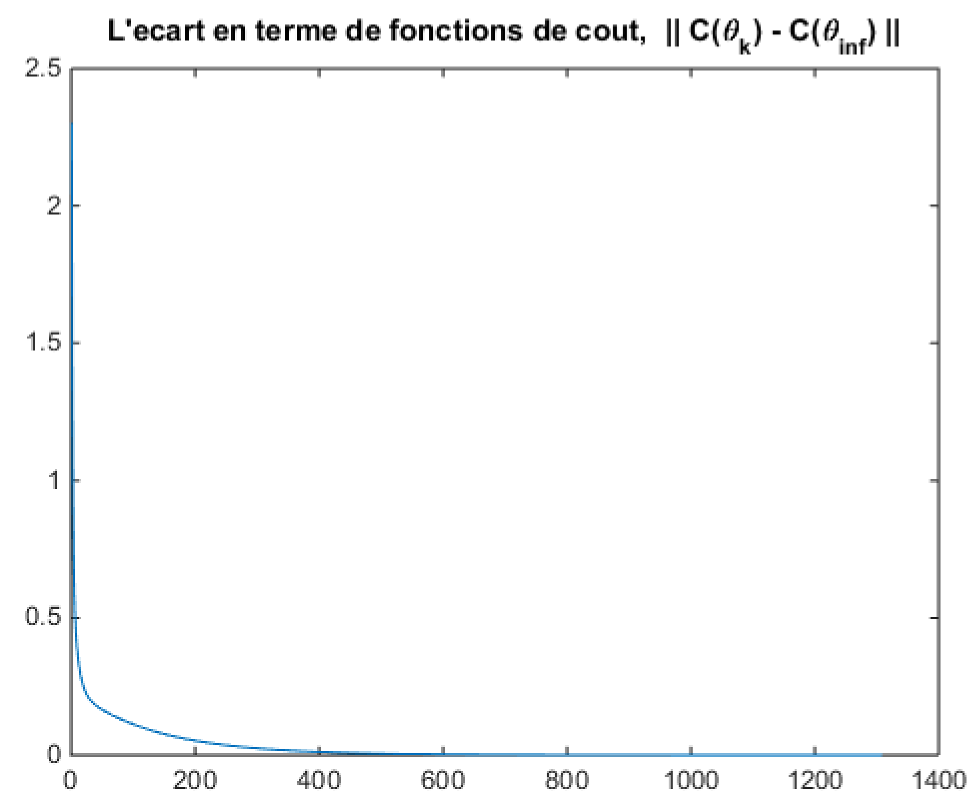

L'iteration de d_k (k = 1...N_iter)
25.35
25.3
25.25
25.2
25.15
25.1
25.05
25
0
200
400
600
800
1000
1200
1400


L'iteration de a_k (k = 1...N_iter)
2
1.95
1.9
1.85
0
200
400
600
800
1000
1200
1400


L'iteration de s_k (k = 1...N_{iter})
2.09
2.08
2.07
2.06
2.05
2.04
2.03
2.02
2.01
2
1.99
0
200
400
600
800
1000
1200
1400


L'ecart entre deux iteres successifs, $|| \theta_k - \theta_{k-1} ||$
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
200
400
600
800
1000
1200
1400


L'ecart a l'optimum θ_inf, || θ_k - θ_inf ||
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
200
400
600
800
1000
1200
1400


L'ecart en terme de fonctions de cout, || C(θ_k) - C(θ_inf) ||
2.5
2
1.5
1
0.5
0
0
200
400
600
800
1000
1200
1400

Q16

```
% Dans cette exercice on va varier les parametres de la
% fonction de recherche lineaire ainsi que le choix du point de depart
```

Quand on change le point du depart:

```
x0Choix = [23:27;0:4;0:4];

for i = 1:size(x0Choix,2)

    % configuration initiale
    epsilon = 10^-2;
    x0 = x0Choix(:,i);

    % variables pour conserver le resultat
    xk = x0;
    xkList = [];
    xkList = [xkList, xk];

    % tant que la norme du gradient sera superieure a 10^-2
    while norm(GradientDeFonctionDeCout(t, sig_noisy, xk)) > epsilon

        % calcul de dk

        dk = -GradientDeFonctionDeCout(t, sig_noisy, xk);

        % calcul de alpha

        alphal = 0;
        alphar = inf;
        alphai = 10^-3;
        beta1 = 10^-3;
        beta2 = 0.99;
        lambda = 20;
        alphak = alphai;

        while 1
            gamma = -beta1.*(GradientDeFonctionDeCout(t, sig_noisy, xk))'*dk;
            if(Troisparametre(t,sig_noisy,(xk+alphai.*dk)) > (Troisparametre(t,sig
                alphar = alphai;
                alphai = (alphal+alphar)/2;
                continue;
            elseif (((GradientDeFonctionDeCout(t, sig_noisy, xk+alphai*dk)'*dk))/(
                alphal = alphai;
                if alphar < inf
                    alphai = (alphal+alphar)/2;
                else
                    alphai = lambda*alphai;
                end
            else
                break;
```

```
            end
        end
        alphak = alphai;

        % mis a jour xk
        xk = xk + alphak*dk;

        % mis a jour xkList

        xkList = [xkList, xk];

    end

    xkInf = xk;

    % l'optimum \theta_inf
    fprintf('Quand le point de depart est (%d, %d, %d)\n', x0);
    fprintf('l''optimum theta sont: d = %f, a = %f, s = %f\n', xkInf);

end

% Conclusion:
% Le point du depart est tres important pour qu'on puisse trouver
% le theta optimal car si on est trop loin de le point optimal,
% on n'arrive pas a trouver le bon resultat.

Quand le point de depart est (23, 0, 0)
l'optimum theta sont: d = 23.000000, a = 0.000000, s = 0.000000
Quand le point de depart est (24, 1, 1)
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054
Quand le point de depart est (25, 2, 2)
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054
Quand le point de depart est (26, 3, 3)
l'optimum theta sont: d = 25.354999, a = 1.851657, s = 2.085328
Quand le point de depart est (27, 4, 4)
l'optimum theta sont: d = 25.354996, a = -1.851657, s = 2.085328
```

Quand on change les parametres de la fonction de recherche lineaire:

```
beta1List = [0.4;0.1;10^-3;10^-4;10^-5];
beta2List = [0.5;0.8;0.99;0.999;0.9999];

for i = 1:size(beta1List)

    % configuration initiale
    epsilon = 10^-2;
    x0 = [25; 2; 2];

 alphal = 0;
        alphar = inf;
        alphai = 10^-3;
```

```
    beta1 = beta1List(i);
    beta2 = beta2List(i);
    lambda = 20;

% variables pour conserver le resultat
xk = x0;
xkList = [];
xkList = [xkList, xk];

% tant que la norme du gradient sera superieure a 10^-2
while norm(GradientDeFonctionDeCout(t, sig_noisy, xk)) > epsilon

    % calcul de dk

    dk = -GradientDeFonctionDeCout(t, sig_noisy, xk);

    % calcul de alpha


    alphak = alphai;

    while 1
        gamma = -beta1.*(GradientDeFonctionDeCout(t, sig_noisy, xk))'*dk;
        if(Troisparametre(t,sig_noisy,(xk+alphai.*dk)) > (Troisparametre(t,sig
            alphar = alphai;
            alphai = (alphal+alphar)/2;
            continue;
        elseif (((GradientDeFonctionDeCout(t, sig_noisy, xk+alphai*dk)'*dk))/(
            alphal = alphai;
            if alphar < inf
                alphai = (alphal+alphar)/2;
            else
                alphai = lambda*alphai;
            end
        else
            break;
        end
    end
    alphak = alphai;

    % mis a jour xk
    xk = xk + alphak*dk;

    % mis a jour xkList

    xkList = [xkList, xk];

end

xkInf = xk;

% l'optimum \theta_inf

fprintf('Quand beta1 = %f, beta2 = %f \n', beta1List(i), beta2List(i));
```

```
    fprintf('l''optimum theta sont: d = %f, a = %f, s = %f\n', xkInf);

end

% Conclusion:
% L'algo n'est pas tres sensible a la variation des parametres
% beta1 et beta2.
% Mais on risque de rater a trouver le bon resultat si on est
% trop loin que les parametres optimales.
% Remarque: beta1 < beta2
```

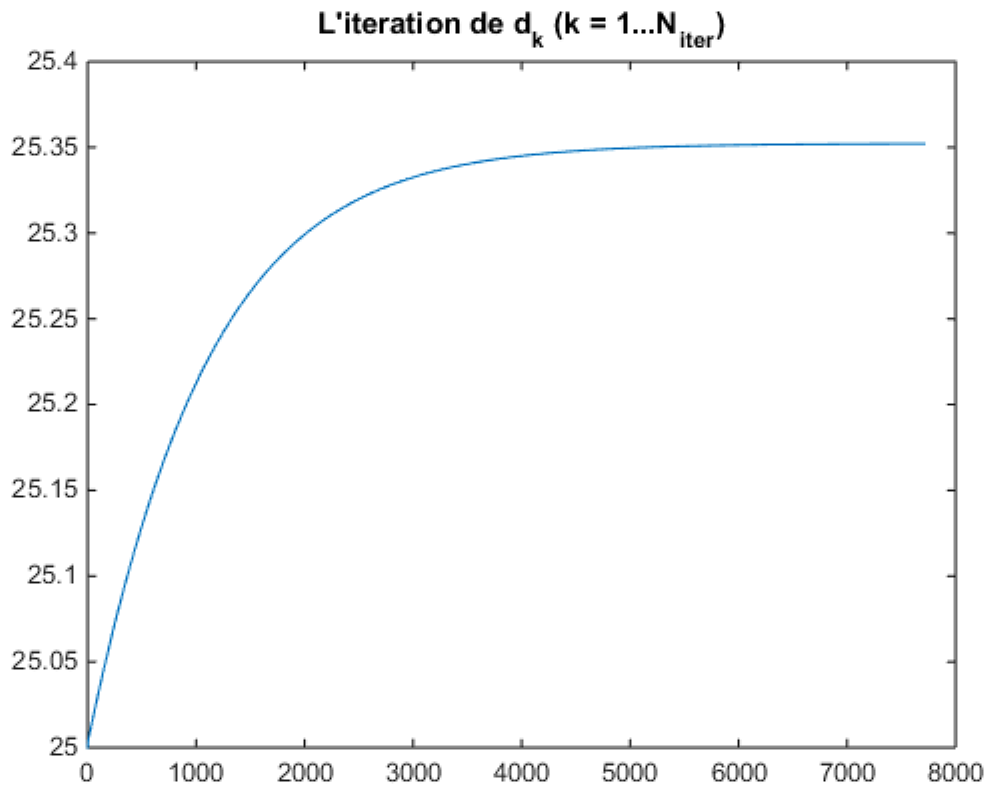
Quand beta1 = 0.400000, beta2 = 0.500000
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054
Quand beta1 = 0.100000, beta2 = 0.800000
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054
Quand beta1 = 0.001000, beta2 = 0.990000
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054
Quand beta1 = 0.000100, beta2 = 0.999000
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054
Quand beta1 = 0.000010, beta2 = 0.999900
l'optimum theta sont: d = 25.349633, a = 1.851757, s = 2.085054

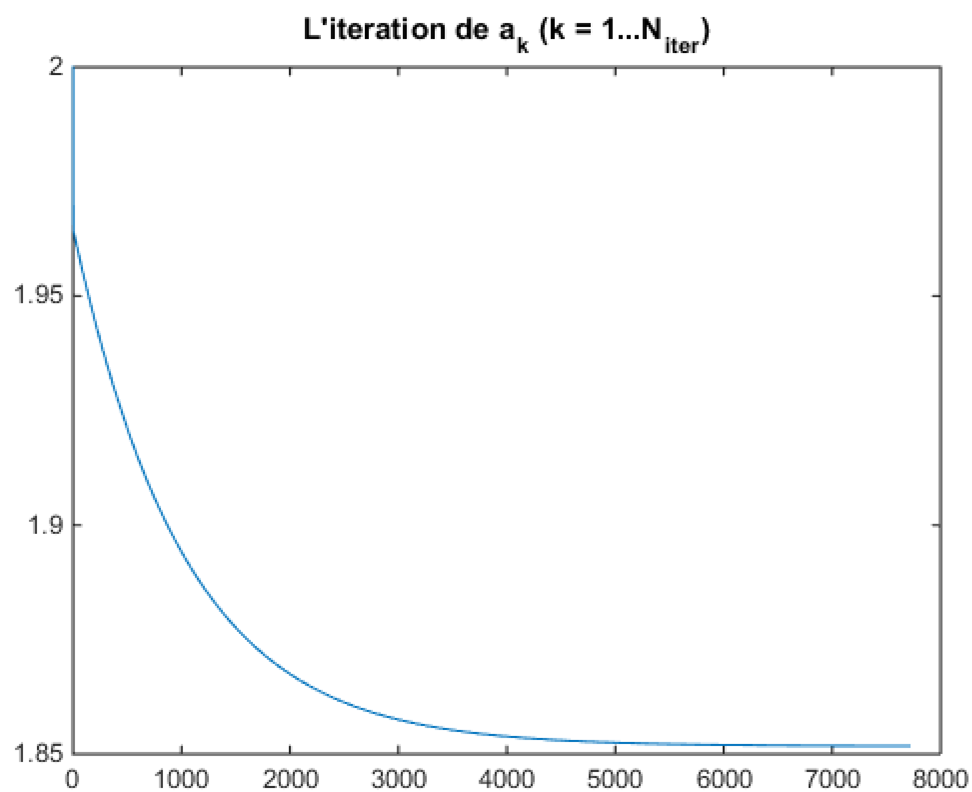
Q17

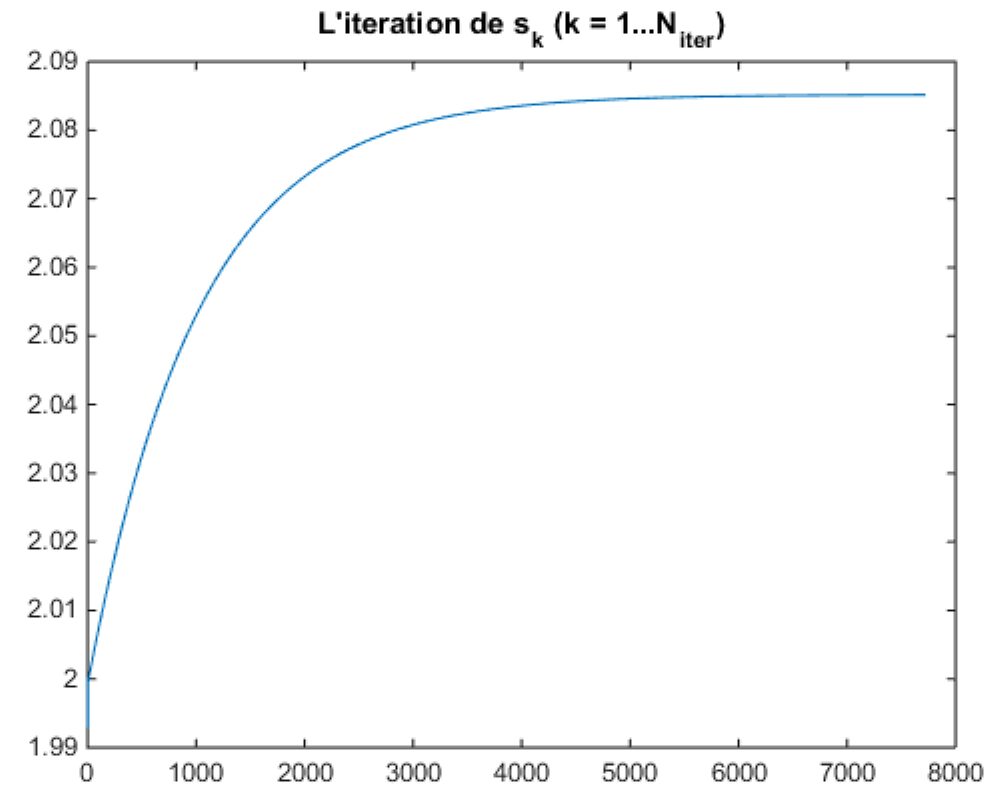
```
% Dans cette exercice on va appliquer la methode de quasi-Newton

% configuration initiale
epsilon = 10^-2;
x0 = [25; 2; 2];
I = eye(3);
H0 = I;

xk = x0;
Hk = H0;
xkList = [];
xkList = [xkList, xk];

while norm(GradientDeFonctionDeCout(t, sig_noisy, xk)) > epsilon

    % calcul de dk

    dk = -Hk*GradientDeFonctionDeCout(t, sig_noisy, xk);

    % calcul de alpha

    alphal = 0;
    alphar = inf;
    alphai = 10^-3;
    beta1 = 10^-3;
    beta2 = 0.99;
    lambda = 20;
    alphak = alphai;
```

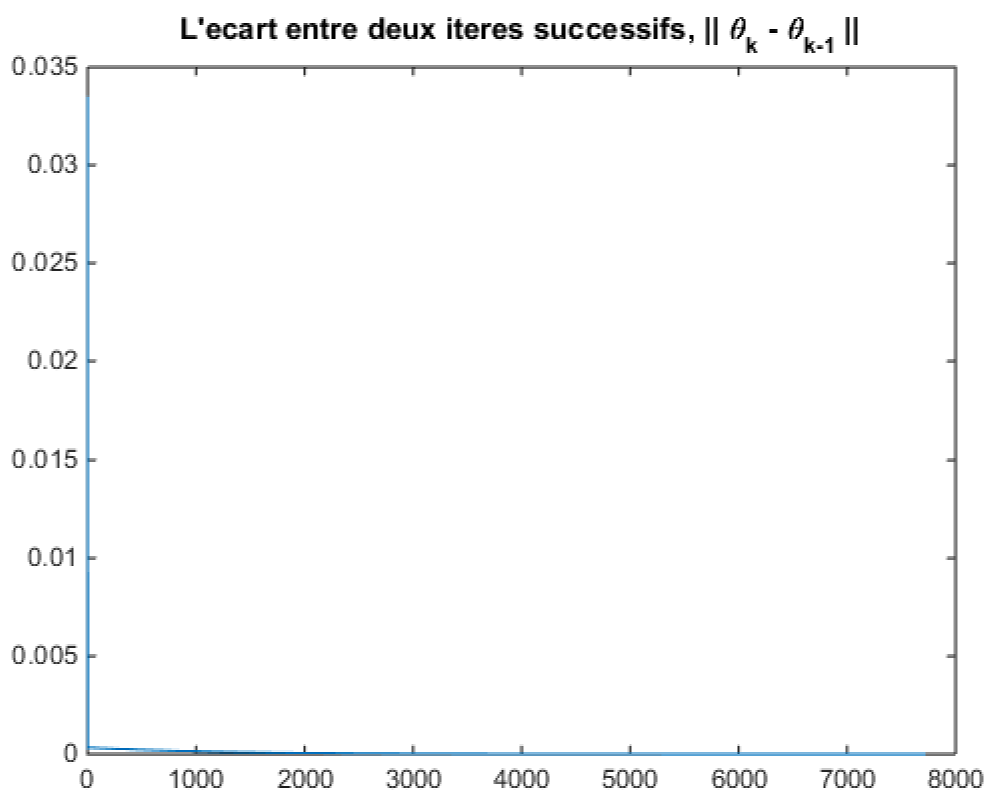
```
        while 1
            gamma = -beta1.*(GradientDeFonctionDeCout(t, sig_noisy, xk))'*dk;
            if(Troisparametre(t,sig_noisy,(xk+alphai.*dk)) > (Troisparametre(t,sig_noi
                alphar = alphai;
                alphai = (alphal+alphar)/2;
                continue;
            elseif (((GradientDeFonctionDeCout(t, sig_noisy, xk+alphai*dk)'*dk))/(Troi
                alphal = alphai;
                if alphar < inf
                    alphai = (alphal+alphar)/2;
                else
                    alphai = lambda*alphai;
                end
                continue;
            else
                break;
            end
        end
        alphak = alphai;

        % mis a jour xk

        xkOld = xk;
        xk = xk + alphak*dk;

        %  mis a jour Hk

        yk1 = GradientDeFonctionDeCout(t, sig_noisy, xk) - GradientDeFonctionDeCout(t,
        dk1 = xk - xkOld;
        Hk = (I-(dk1*(yk1'))/((dk1')*yk1))*Hk*(I-(yk1*(dk1'))/((dk1')*yk1)) + (dk1*(dk

        % mis a jour xkList

        xkList = [xkList, xk];

    end

    xkInf = xk;

    % Presentation de l'iteration de \theta_k (k = 1...N_{iter})
    figure;
    plot(1:size(xkList,2), xkList(1,:));
    title('L''iteration de d_k (k = 1...N_{iter})');
    figure;
    plot(1:size(xkList,2), xkList(2,:));
    title('L''iteration de a_k (k = 1...N_{iter})');
    figure;
    plot(1:size(xkList,2), xkList(3,:));
    title('L''iteration de s_k (k = 1...N_{iter})');

    % Presentation de l'ecart entre deux iteres successifs
    xkEcart = [];
    for i = 2:size(xkList,2)
        xkEcart = [xkEcart, norm(xkList(:,i)-xkList(:,i-1))];
```

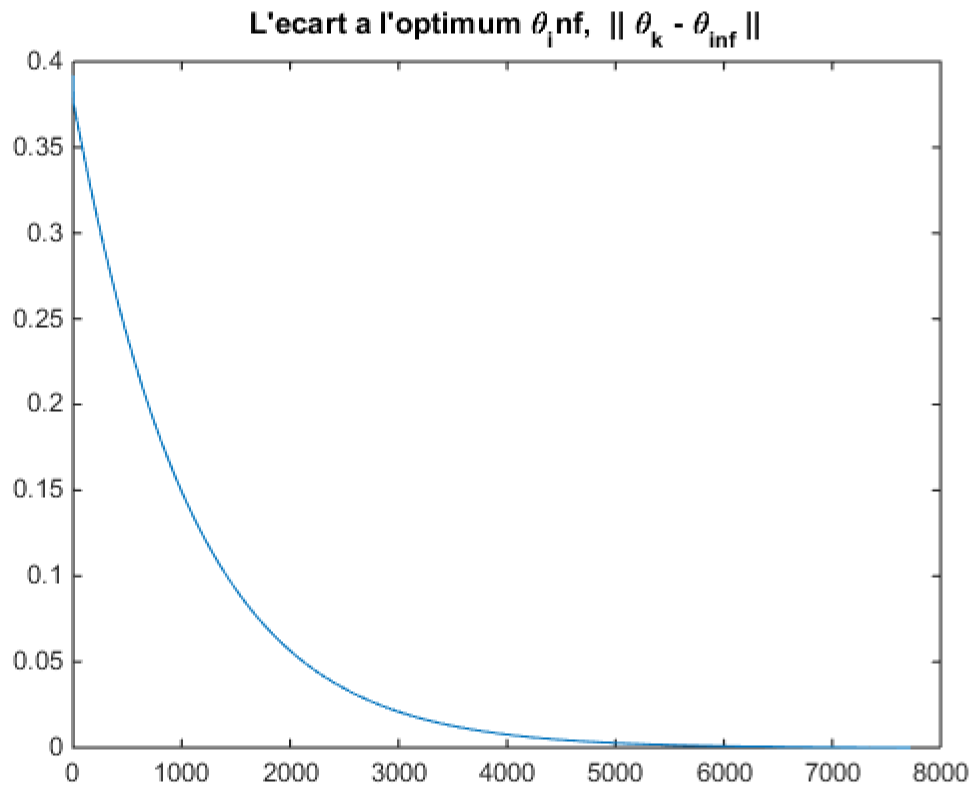
```
end
figure;
plot(1:size(xkEcart,2), xkEcart);
title('L''ecart entre deux iteres successifs, || \theta_k - \theta_{k-1} ||');

% presentation de l'ecart a l'optimum \theta_inf
xkEcartInf = [];
for i = 1:size(xkList,2)
    xkEcartInf = [xkEcartInf, norm(xkList(:,i)-xkInf)];
end
figure;
plot(1:size(xkEcartInf,2), xkEcartInf);
title('L''ecart a l''optimum \theta_inf,  || \theta_k - \theta_{inf} ||');

% Presentation de l'ecart en terme de fonctions de cout
CEcart = [];
for i = 1:size(xkList,2)
    CEcart = [CEcart, norm(Troisparametre(t,sig_noisy,xkList(:,i))-Troisparametre(
end
figure;
plot(1:size(CEcart,2), CEcart);
title('L''ecart en terme de fonctions de cout,  || C(\theta_k) - C(\theta_{inf}) |

% la norme infinie du gradient
fprintf('la norme infinie du gradient est: %f\n', norm(GradientDeFonctionDeCout(t,

% l'optimum \theta_inf
fprintf('l''optimum theta sont: d = %f, a = %f, s = %f', xkInf);

la norme infinie du gradient est: 0.009994
l'optimum theta sont: d = 25.352135, a = 1.851760, s = 2.085151
```

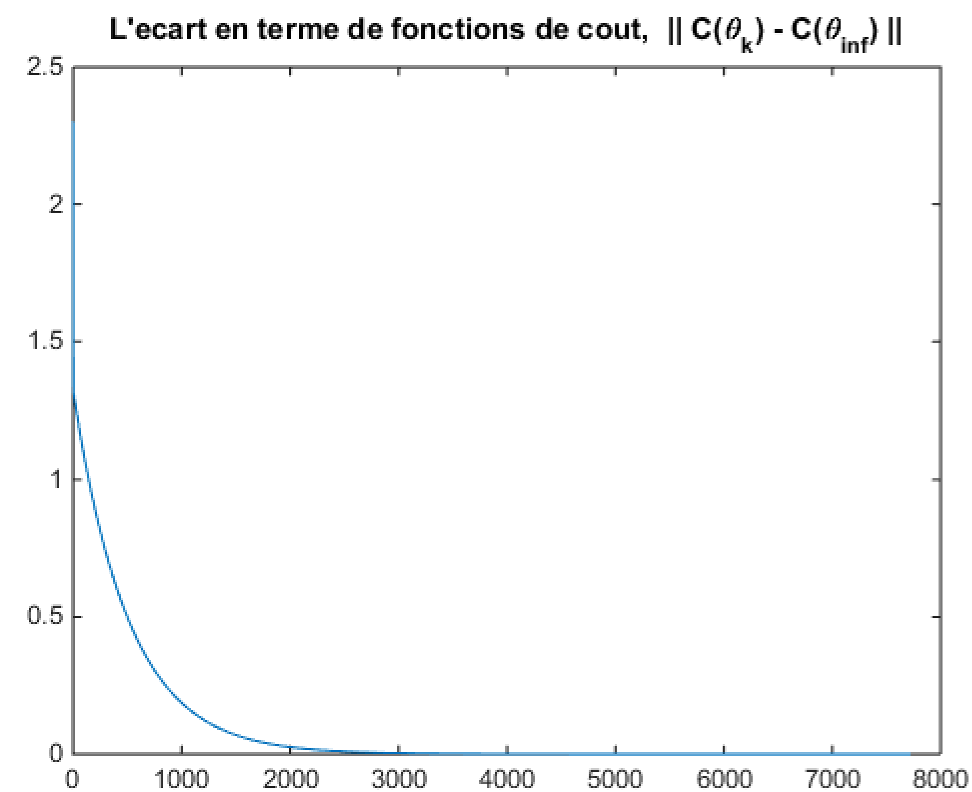

L'iteration de d$_k$ (k = 1...N$_{iter}$)
25.4
25.35
25.3
25.25
25.2
25.15
25.1
25.05
25
0
1000
2000
3000
4000
5000
6000
7000
8000


L'iteration de a$_k$ (k = 1...N$_{iter}$)
2
1.95
1.9
1.85
0
1000
2000
3000
4000
5000
6000
7000
8000


L'iteration de s_k (k = 1...N_{iter})
2.09
2.08
2.07
2.06
2.05
2.04
2.03
2.02
2.01
2
1.99
0
1000
2000
3000
4000
5000
6000
7000
8000


L'ecart entre deux iteres successifs, || θ_k - θ_{k-1} ||
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
1000
2000
3000
4000
5000
6000
7000
8000


L'ecart a l'optimum $\theta_i$nf, || θ_k - θ_{inf} ||
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
1000
2000
3000
4000
5000
6000
7000
8000


L'ecart en terme de fonctions de cout, || C(θ_k) - C(θ_{inf}) ||
2.5
2
1.5
1
0.5
0
0
1000
2000
3000
4000
5000
6000
7000
8000

Q18

```
% Ici on applique la fonction 'fminunc' pour determiner le minimum
fun = @(x)Troisparametre(t,sig_noisy,x);
x0 = [25, 2, 2];
[x,fval] = fminunc(fun,x0);
fprintf('En appliquant la fonction ''fminunc'',\n');
fprintf('on obtient le resultat suivant: d = %f, a = %f, s =  %f\n', x);
```

```
Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the function tolerance.



En appliquant la fonction 'fminunc',
on obtient le resultat suivant: d = 25.352314, a = 1.851707, s =  2.085191
```

Published with MATLAB® R2014b