Data Mining
TP7 - Introduction à l'optimisation

Rémi MUSSARD - Thomas ROBERT

Le but du TP est d'étudier différentes méthodes permettant de converger vers le minimum d'une fonction convexe de plusieurs variables.

Pour ces méthodes, on utile le gradient ou la matrice Hessienne afin de déterminer la direction de descente permettant de faire décroitre la fonction de coût. On avance dans cette direction d'une valeur définie par un pas qui peut être fixe ou variable. Enfin, on itère ce processus tant que le coût varie de façon non négligeable (nous avons choisi de fixer le seuil à 10^{-5}.

1 Fonctions utilisées

Pour faire ce TP, les fonctions suivantes ont été codées.

1.1 `moncritere.m`

Cette fonction permet de calculer la valeur du critère de coût en θ.

```
function cout = moncritere(a, b, c, theta)
    cout = exp(-0.1)*(exp(a'*theta) + exp(b'*theta) + exp(c'*theta));
end
```

1.2 `mongradient.m`

Cette fonction permet de calculer le gradient au critère de coût en θ.

```
function d = mongradient(a, b, c, theta)
    d = exp(-0.1)*(a*exp(a'*theta) + b*exp(b'*theta) + c*exp(c'*theta));
end
```

1.3 `monHessien.m`

Cette fonction permet de calculer la matrice Hessienne du critère de coût en θ.

```
function H = monHessien(a, b, c, theta)
    H = exp(-0.1)*(exp(a'*theta)*(a*a') + exp(b'*theta)*(b*b') + exp(c'*theta)*(c*c'));
end
```

1.4 `init_fig.m`

Cette fonction permet d'initialiser une figure avec les contours du gradient ainsi que θ_0.

```
function init_fig(theta0, Jmat, n, X, Y)

    % Create the surface plot using contour command
    figure;
    contour(X, Y, reshape(Jmat, n, n), 10, 'linewidth', 1.5);
    colorbar;

    % trace de theta0
    hold on
    h = plot(theta0(1,:), theta0(2,:), 'ro');
    set(h, 'MarkerSize', 8, 'markerfacecolor', 'r');
    text(theta0(1), theta0(2)+0.025, '\theta_0', 'fontsize', 15)
```

2 Préparation du script

Cette partie de code permet de préparer les données utilisées par chaque méthode.

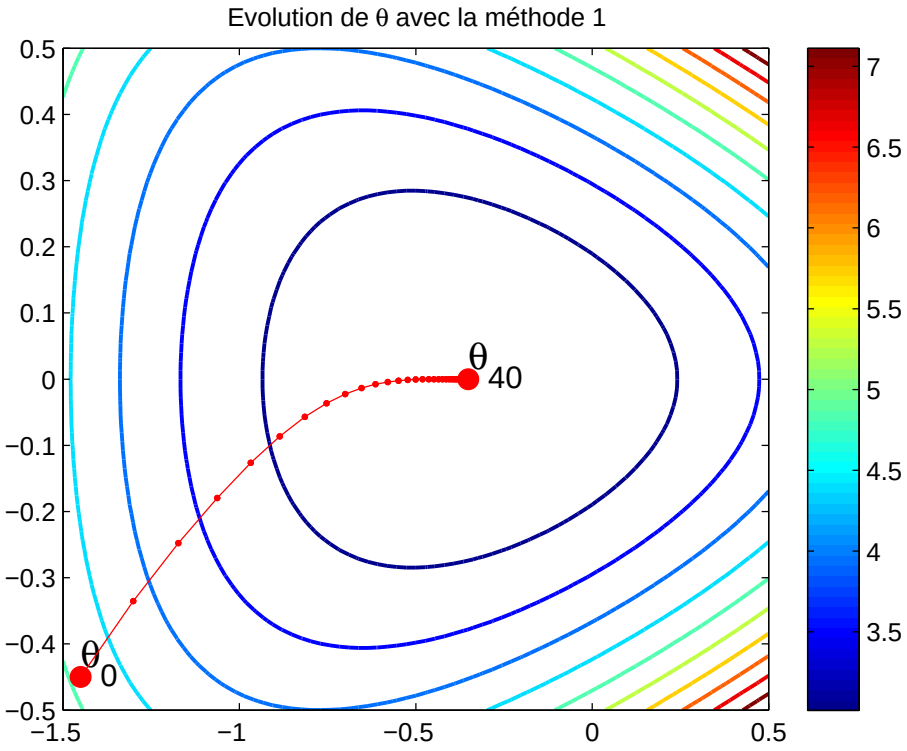
```
clear all
close all
clc

% parametres du probleme
a = [1; 3];
b = [1; -3];
c = [-1; 0];

% Create a grid of x and y points
n = 75;
[X, Y] = meshgrid(linspace(-1.5, 0.5, n), linspace(-0.5, 0.5, n));
ptx = reshape(X, n*n,1);
pty = reshape(Y, n*n,1);
pt = [ptx pty];

% Define the function J = f(\theta)
Jmat = exp(-0.1)*(exp(pt*a) + exp(pt*b) + exp(pt*c));

% solution initiale
theta0 = [-1.45; -0.45];
```

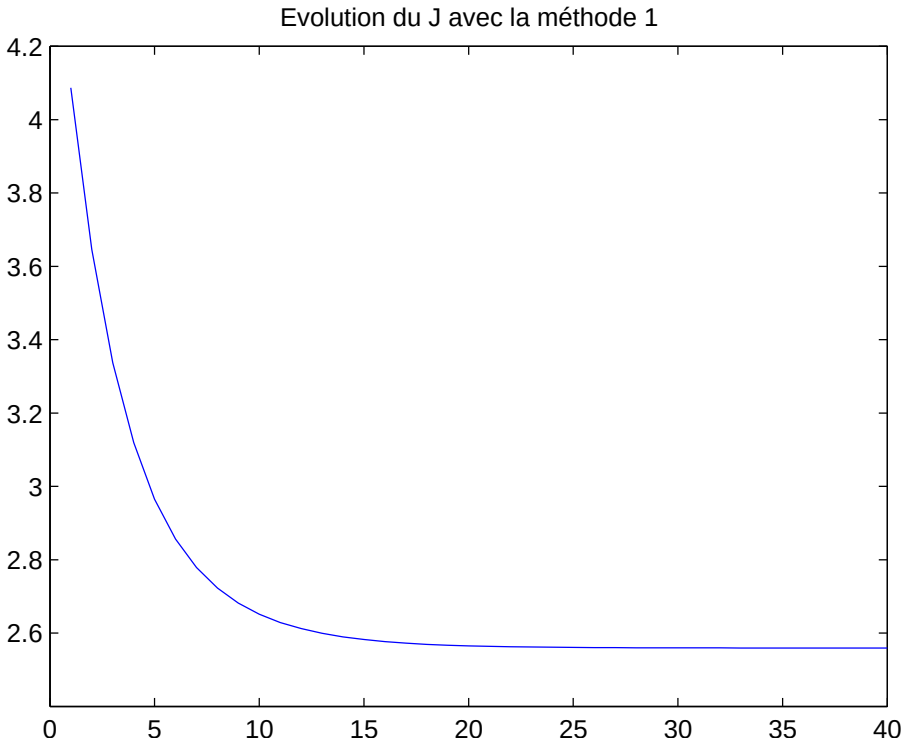
3 Première version avec α constant

On réalise un premier code avec un pas α constant. Ce pas est défini par des essais successifs pour trouver un pas adapté au problème, c'est à dire convergeant à une vitesse satisfaisante : ni trop rapide, ni trop lent.

Notons qu'en réalité, le pas d'avancement à chaque itération n'est pas constant car on ne normalise pas le gradient. Ainsi, le pas d'avancement réel dépend de la valeur du gradient, et est donc d'autant plus grand que la fonction augmente rapidement selon la direction de descente.

```
% pas alpha fixé
alpha =  0.05;

% variables pour la boucle et initialisation
Jlist = [];
J = moncritere(a, b, c, theta0);
Jprec = J + 1;
theta_old = theta0;
theta = theta0;
i = 1;

% initialiser la figure
init_fig(theta0, Jmat, n, X, Y);

% tant qu'on a pas convergé, on itère
while abs(J - Jprec) > 1e-5 && i < 200

    % calculs du nouveau theta
    grad = mongradient(a, b, c, theta);     % calcul du gradient
    direction = -grad;                      % direction de descente
    theta_old = theta;                      % sauvegarde ancien theta pour affichage
    theta = theta + alpha * direction;      % MAJ du point en cours

    % trace du theta courant
    h = plot([theta_old(1) theta(1)], [theta_old(2) theta(2)], '-ro');
    set(h, 'MarkerSize', 2, 'markerfacecolor', 'r');

    % calcul de J
    Jprec = J;
    J = moncritere(a, b, c, theta);
    Jlist = [Jlist J];
    i = i + 1;
end

% theta final
h = plot(theta(1,:), theta(2,:), 'ro');
set(h, 'MarkerSize', 8, 'markerfacecolor', 'r');
text(theta(1,1), theta(2,1)+0.025, ['\theta_{' int2str(i-1) '}'], 'fontsize', 15)
title('Evolution de \theta avec la méthode 1');

% affichage évolution J
figure;
plot(Jlist);
title('Evolution du J avec la méthode 1');
```


Evolution de θ avec la méthode 1
θ 40
θ 0
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−1.5
−1
−0.5
0
0.5
7
6.5
6
5.5
5
4.5
4
3.5


Evolution du J avec la méthode 1
4.2
4
3.8
3.6
3.4
3.2
3
2.8
2.6
0
5
10
15
20
25
30
35
40

4 Deuxième version avec α variable

Cette fois, on décide d'appliquer une règle permettant de faire varier α afin de converger plus vite.

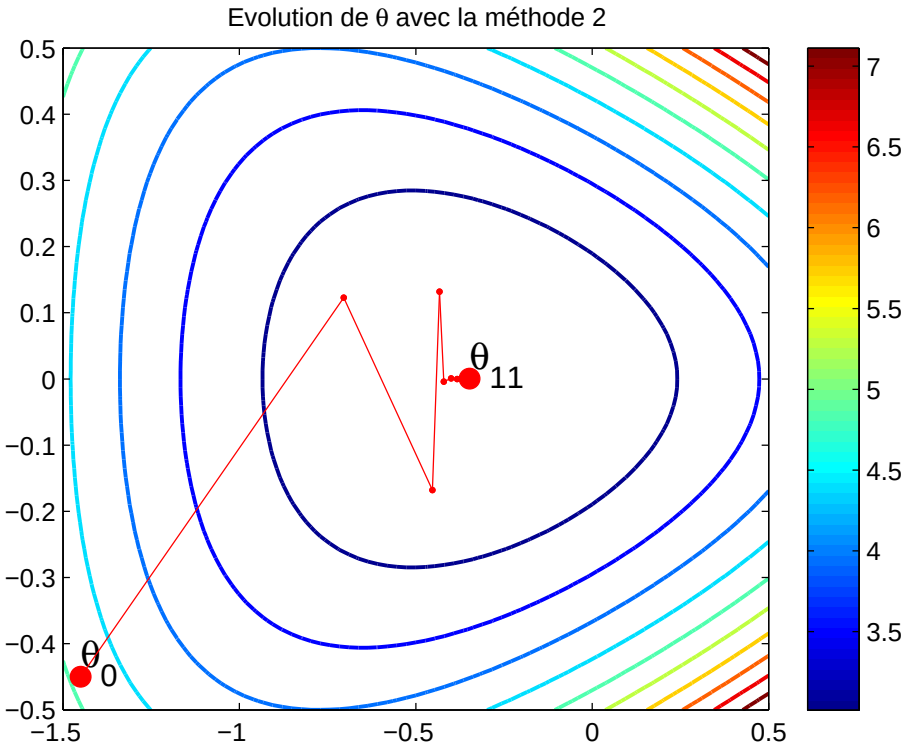
A chaque itération, si le α que l'on a nous permet de converger, on le multiplie par 1,15 afin d'essayer de converger plus vite à l'itération suivante.

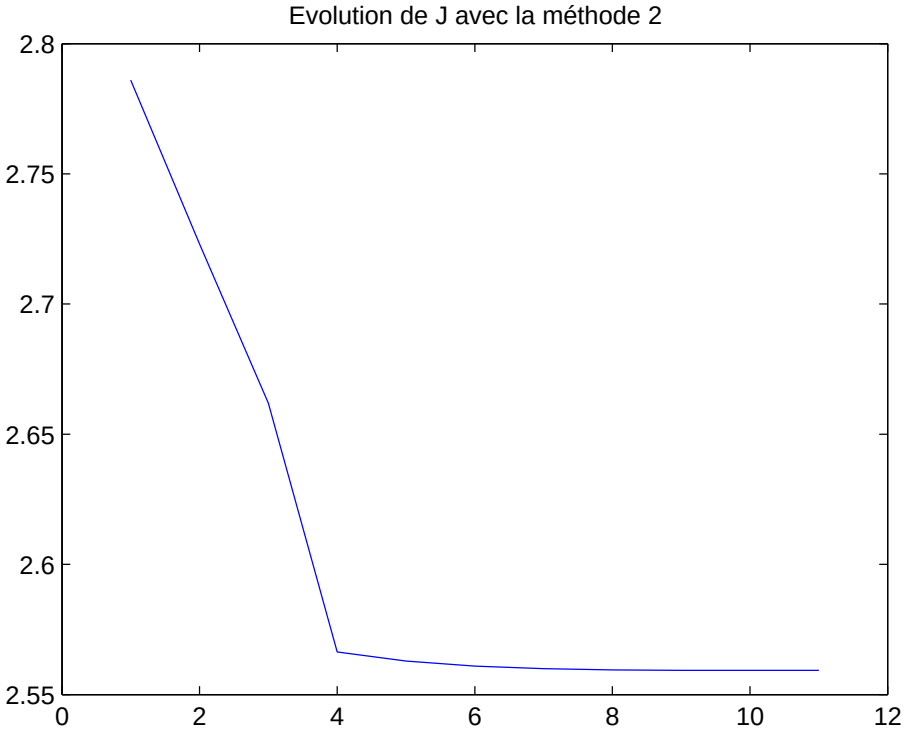
Sinon, si le α que l'on avait fait augmenter le coût, alors on annule les calculs réalisés et on divise α par 2.

On constate que cette méthode permet de converger beaucoup plus rapidement en nous permettant de partir avec un α relativement grand sans avoir peur de diverger.

```
% initialisation des variables
alpha =  1;
Jlist = [];
J = moncritere(a, b, c, theta0);
Jprec = J+1e-3;
theta = theta0;
theta_old = theta0;
i = 1;
j = 1;

% initialiser la figure
init_fig(theta0, Jmat, n, X, Y);

% tant qu'on a pas convergé, on itère
while abs(J - Jprec) > 1e-5 && i < 300 && j < 300

    % calculs
    grad = mongradient(a, b, c, theta);          % calcul du gradient
    direction = -grad;                           % direction de descente
    theta_new = theta + alpha * direction;       % MAJ de theta

    J_new = moncritere(a, b, c, theta_new);      % calcul de J

    % si on améliore J avec le calcul réalisé
    if(J - J_new > 0)

        % augmentation de alpha pour l'itération suivante
        alpha = alpha*1.15;

        % enregistrement de ce qui a été fait
        Jprec = J;
        J = J_new;
        Jlist = [Jlist J];
        theta_old = theta;
        theta = theta_new;

        % affichage theta
        h = plot([theta_old(1) theta(1)], [theta_old(2) theta(2)], '-ro');
        set(h, 'MarkerSize', 2, 'markerfacecolor', 'r');

        j = j + 1;

    % sinon si on a augmenté J, on enregistre rien et on diminue alpha
    else
        alpha = alpha/2;
    end

    i = i + 1;
end

% theta final
h = plot(theta(1,:), theta(2,:), 'ro');
set(h, 'MarkerSize', 8, 'markerfacecolor', 'r');
text(theta(1,1), theta(2,1)+0.025, ['\theta_{' int2str(j-1) '}'], 'fontsize', 15)
title('Evolution de \theta avec la méthode 2');

% affichage évolution J
figure;
```

```
59 plot(Jlist);
60 title('Evolution de J avec la méthode 2');
```

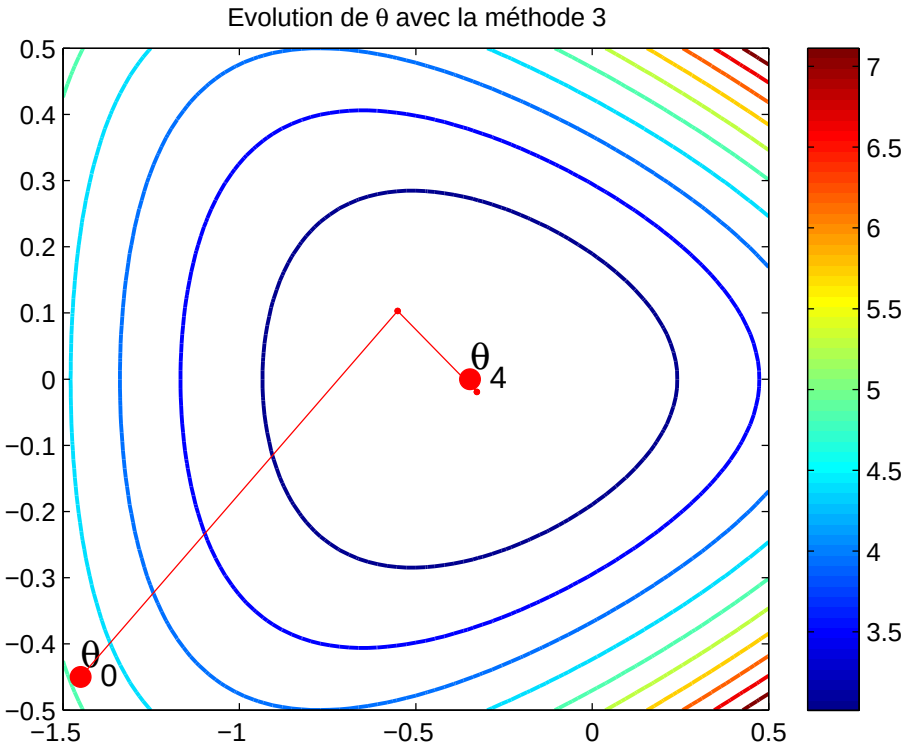


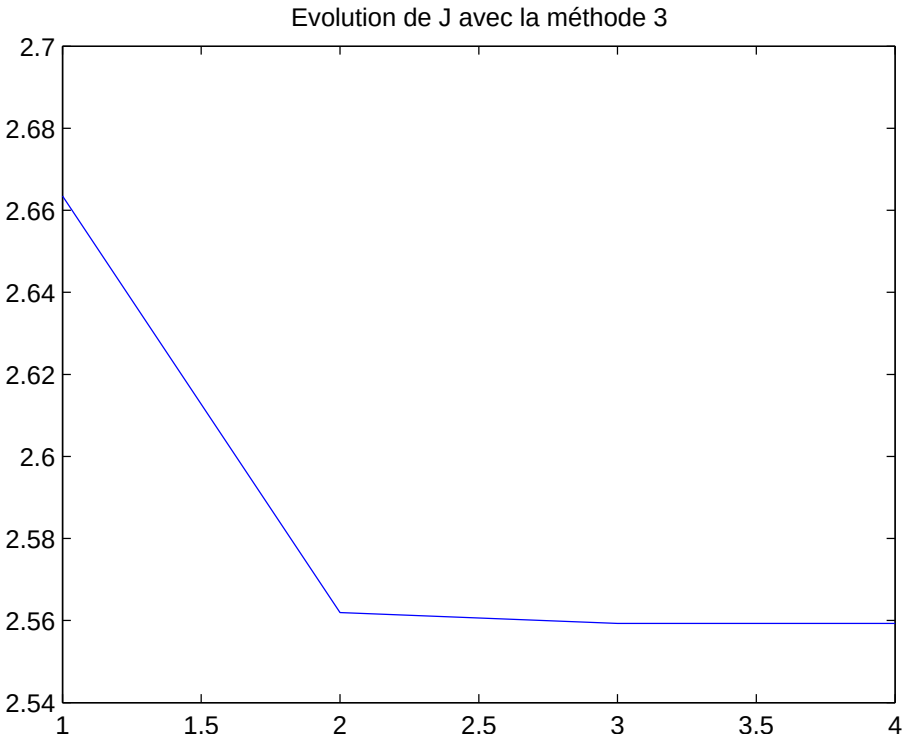


5 Troisième version avec la matrice Hessienne

Cette fois, on calcule la matrice Hessienne afin d'augmenter la vitesse de convergence. On calcule donc (via la matrice Hessienne) les dérivées secondes de la fonction étudiée, permettant de converger beaucoup plus rapidement.

```
% initialisation des variables
Jlist = [];
J = moncritere(a, b, c, theta0);
Jprec = J+1e-3;
theta = theta0;
i = 1;

% initialiser la figure
init_fig(theta0, Jmat, n, X, Y);

% tant qu'on a pas convergé, on itère
while abs(J - Jprec) > 1e-5 && i < 200

    % calculs
    grad = mongradient(a, b, c, theta);     % calcul du gradient
    H = monHessien(a, b, c, theta);         % calcul de la matrice Hesienne
    direction = -H\grad;                    % direction de descente
    theta_old = theta;                      % sauvegarde ancien theta
    theta = theta + direction;              % MAJ theta

    % calcul de J
    Jprec = J;
    J = moncritere(a, b, c, theta);
    Jlist = [Jlist J];

    % trace du theta courant
    h = plot([theta_old(1) theta(1)], [theta_old(2) theta(2)], '-ro');
    set(h, 'MarkerSize', 2, 'markerfacecolor', 'r');

    % inc i
    i = i + 1;
end

% theta final
h = plot(theta(1,:), theta(2,:), 'ro');
set(h, 'MarkerSize', 8, 'markerfacecolor', 'r');
text(theta(1,1), theta(2,1)+0.025, ['\theta_{' int2str(i-1) '}'], 'fontsize', 15)
title('Evolution de \theta avec la méthode 3');

% affichage évolution J
figure;
plot(Jlist);
title('Evolution de J avec la méthode 3');
```





6 Conclusion

Après avoir mesure les temps de calcul de chacune des méthodes proposées ci-dessus, il semblerait que la méthode 2 soit bien plus rapide que la 1, ce qui semble logique.

Par contre, il semble légitime de se demander si le surplus de calcul demandé par la méthode 3 est compensé

par la diminution du nombre d'itérations à réaliser pour converger. Dans notre cas, cela semble être le cas.