Data Mining
TP8 - Optimisation sans contraintes

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 gradient est supérieur à un seuil fixé à 10^{-3}.

1 Ce qu'on veut trouver

Afin de calculer le gradient et la Hessienne, on calcule les dérivées partielles d'ordre 1 et 2 par rapport à θ_1 et θ_2. On obtient :

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} = & -2(1-\theta_1) - 400\theta_1(\theta_2 - \theta_1^2)^2 \\ \frac{\partial J}{\partial \theta_2} = & 200(\theta_2 - \theta_1^2)^2 \end{bmatrix}$$

$$H = \begin{bmatrix} \frac{\partial^2 J}{\partial \theta_1^2} = 2 - 400(\theta_2 - \theta_1^2)^2 + 800\theta_1^2 & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2} = -400\theta_1 \\ \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2} = -400\theta_1 & \frac{\partial^2 J}{\partial \theta_2^2} = 200 \end{bmatrix}$$

On peut également calcule analytiquement l'extremum de cette fonction, il suffit d'annuler les dérivées, ce qui permet d'obtenir un extremum en $\theta^* = (1,1)^\top$.

Le calcul de la Hessienne permet de voir qu'il s'agit bien d'un minimum puisqu'elle est définie positive :

$$H(\theta^*) = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$$

2 Préparation

```
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 = 100;
[X, Y] = meshgrid(linspace(-1.25, 2.5, n), linspace(-1.75, 2, n));
ptx = reshape(X, n*n,1);
pty = reshape(Y, n*n,1);
pt = [ptx pty];
```

```
16
17 % Define the function J = f(\theta)
18 Jmat = (1-pt(:,1)).^2 + 100*(pt(:, 2) - pt(:, 1).^2).^2;
19
20 % solution initiale
21 theta0 = [-1; 0];
```

3 Fonction `init_fig`

Afin d'afficher une figure avec les lignes de niveaux, on utilisera la fonction init_fig.

```
1  function init_fig(theta0, Jmat, n, X, Y)
2
3      % Create the surface plot using contour command
4      figure;
5      set(gcf, 'PaperUnits', 'points');
6      set(gcf, 'PaperPosition', [0 0 750*1.7 500*1.7]);
7      set(gcf, 'Position', [0 0 750*1.7 500*1.7]);
8      contour(X, Y, reshape(Jmat, n, n), [40:-5:0 1 0], 'linewidth', 1.5);
9      colorbar
10     axis tight
11
12     % trace de theta0
13     hold on
14     h = plot(theta0(1,:), theta0(2,:), 'ro');
15     set(h, 'MarkerSize', 8, 'markerfacecolor', 'r');
16     text(theta0(1), theta0(2)+0.025, '\theta_0', 'fontsize', 15)
```

4 Première version avec α constant

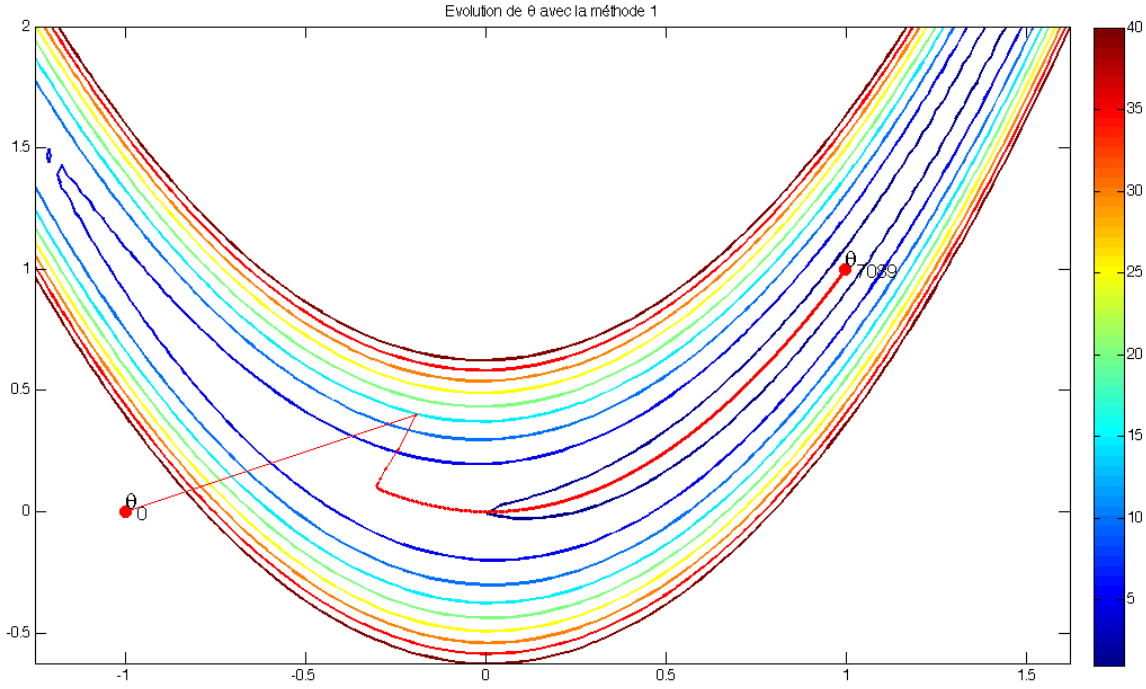
On réalise un premier code avec un pas α constant. Ce pas est défini par des essais successifs pour trouver un pas adapté au problème, c'est à dire convergeant à une vitesse satisfaisante : ni trop rapide, ni trop lent.

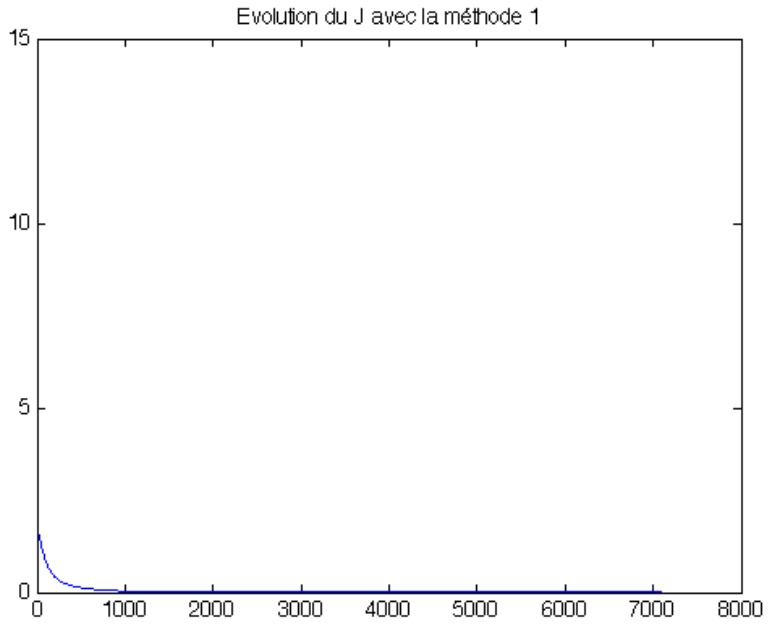
Notons qu'en réalité, le pas d'avancement à chaque itération n'est pas constant car on ne normalise pas le gradient. Ainsi, le pas d'avancement réel dépend de la valeur du gradient, et est donc d'autant plus grand que la fonction augmente rapidement selon la direction de descente.

On remarque que la fonction présente une zone où le coût ne varie quasiment pas, donnant un gradient de valeur faible, et nous faisant donc faire de très petits pas vers le minimum, rendant la convergence très lente.

```
1  % pas alpha fixé
2  alpha =  0.002;
3
4  % variables pour la boucle et initialisation
5  Jlist = [];
6  grad = mongradient(theta0);
7  theta_old = theta0;
8  theta = theta0;
9  i = 1;
10
11 % initialiser la figure
12 close all;
13 init_fig(theta0, Jmat, n, X, Y);
14
15 % tant qu'on a pas convergé, on itère
16 while norm(grad) > 1e-3 && i < 15000
17
18     % calculs du nouveau theta
19     grad = mongradient(theta);              % calcul du gradient
20     direction = -grad;                      % direction de descente
21     theta_old = theta;                      % sauvegarde ancien theta pour affichage
22     theta = theta + alpha * direction;      % MAJ du point en cours
23
24     % 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
    Jlist = [Jlist moncritere(theta)];
    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');
```





5 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 voit que la méthode met toujours longtemps à converger car la direction de descente ne suit pas vraiment une direction optimale qui serait au centre de la ligne de niveau, c'est à dire la direction curviligne qui suivrait le « creux » de la fonction.

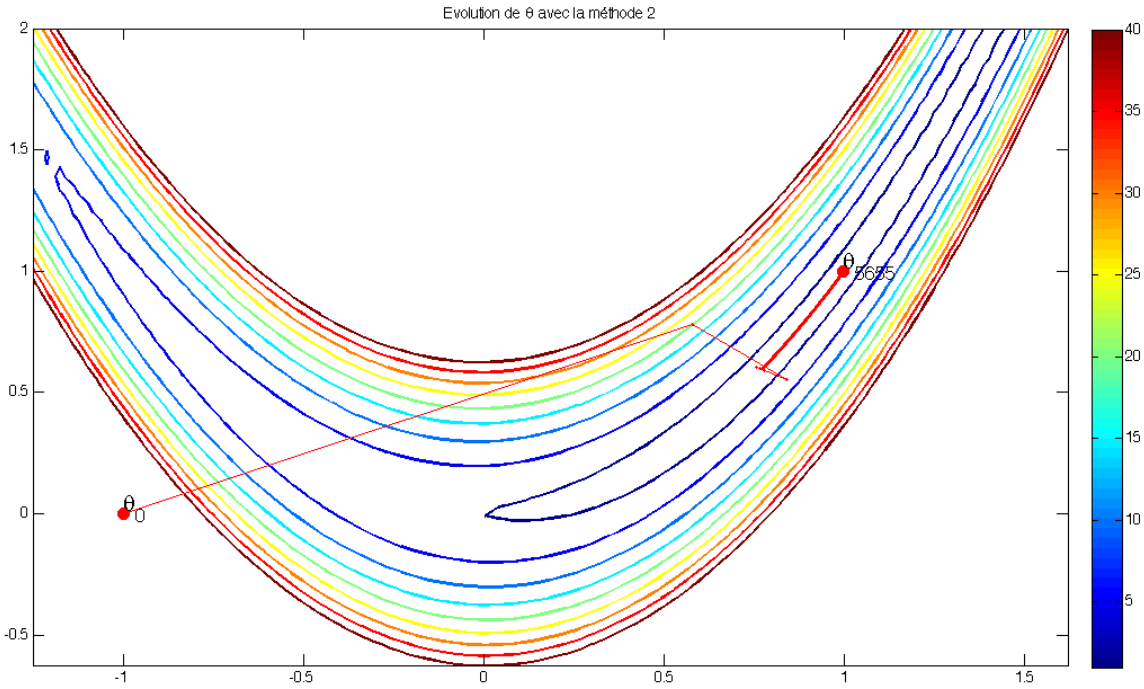
A la place, on oscille autour de cette direction optimale en suivant des directions qui ne sont pas optimales (souvent environ à 45° entre la direction optimale et sa perpendiculaire).

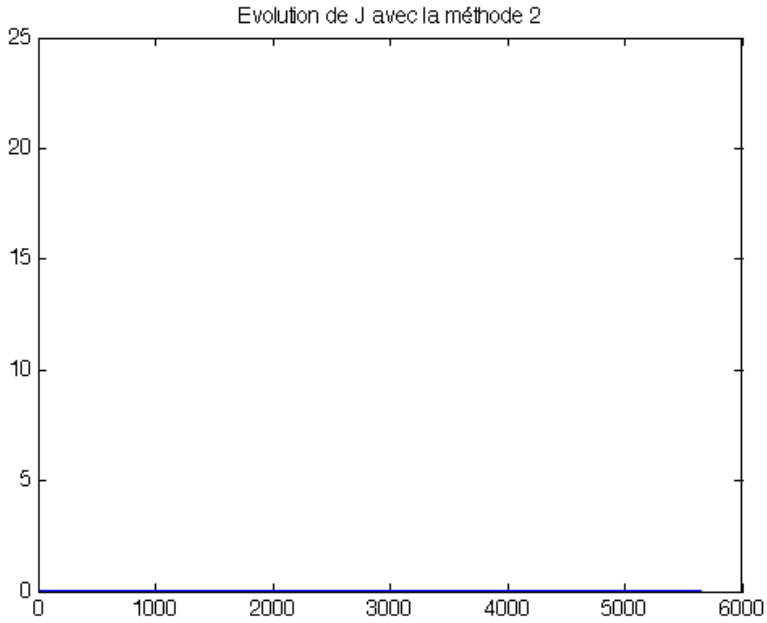
Le fait que cette direction ne sont pas optimale fait qu'il est impossible d'avancer beaucoup dans cette direction sans faire réaugmenter le coût J. Ainsi, le pas ne peux pas vraiment augmenter.

Ce problème de direction vient du fait que le gradient nous donne une approximation de premier ordre. On essaye descend donc forcément en ligne droite ce qui n'est pas optimal.

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

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

% tant qu'on a pas convergé, on itère
```

```
while norm(grad) > 1e-3 && i < 15000 && j < 10000

    % calculs
    grad = mongradient(theta);                  % calcul du gradient
    direction = -grad;                          % direction de descente
    theta_new = theta + alpha * direction;      % MAJ de theta
    J_new = moncritere(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*1.3;

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


Evolution de θ avec la méthode 2
θ0
θ3635


Evolution de J avec la méthode 2

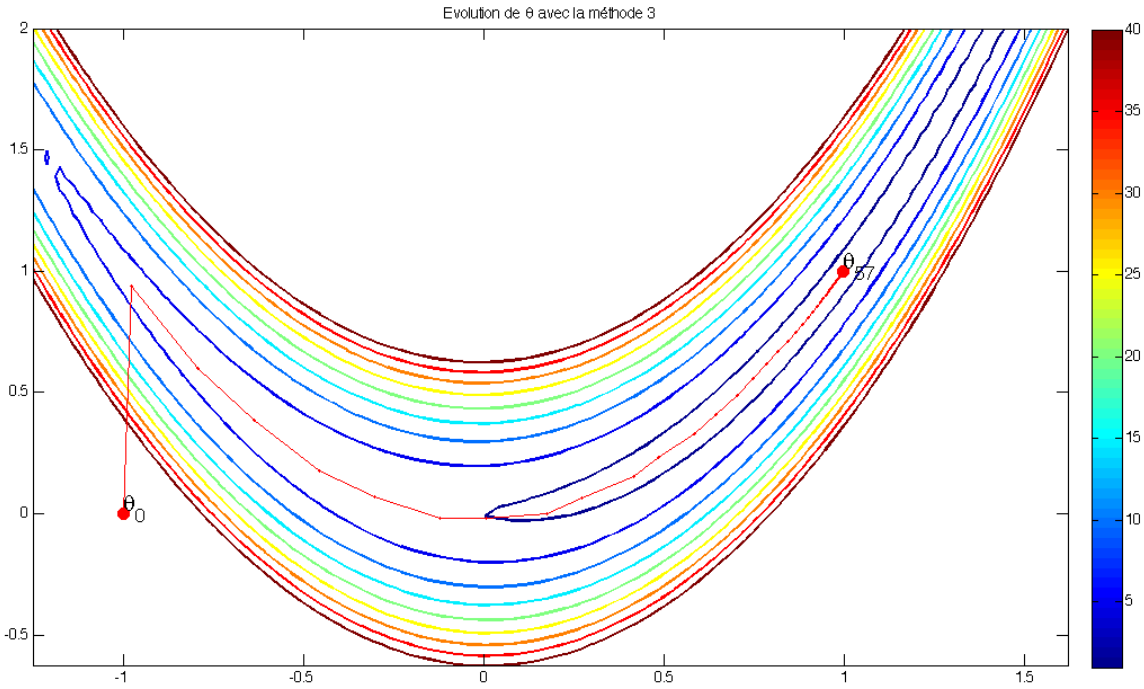
6 Troisième version avec la matrice Hessienne

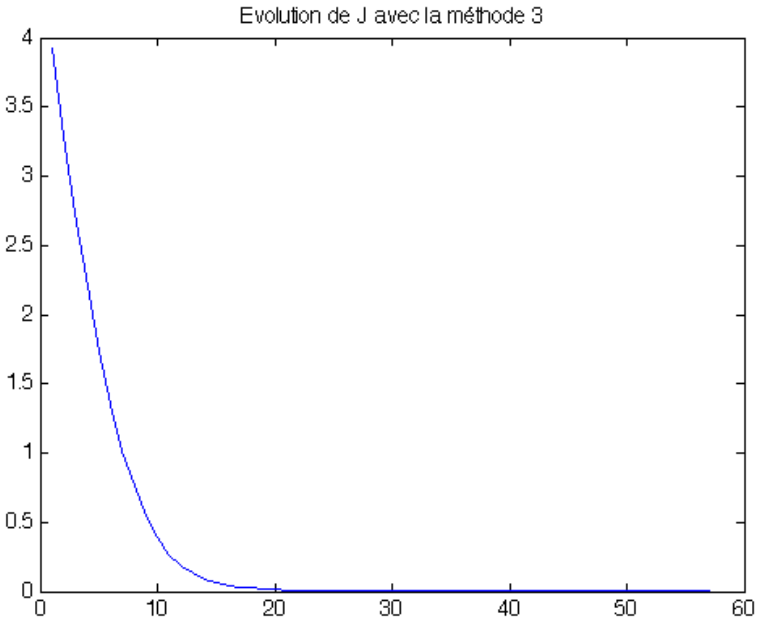
Cette fois, on calcule la matrice Hessienne afin d'augmenter la vitesse de convergence. On calcule donc (via la matrice Hessienne) les dérivées secondes de la fonction étudiée, permettant de converger beaucoup plus rapidement.

Le problème est que notre matrice Hessienne n'est pas définie positive, ou du moins pas toujours bien conditionnée. On utilise donc la méthode de *quasi-Newton* en utilisant au lieu de la matrice H une matrice B proche de H mais définie positive. Dans notre cas, on se contente de reconditionner la matrice H en avant $\lambda = 3$ sur la diagonale.

On remarque qu'avec cette méthode, on converge beaucoup plus rapidement qu'avec les méthodes précédentes.

```
% initialisation des variables
lambda = 3;
Jlist = [];
J = moncritere(theta0);
Jprec = J+1e-3;
grad = mongradient(theta0);
theta = theta0;
i = 1;

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

% tant qu'on a pas convergé, on itére
while norm(grad) > 1e-3 && i < 3000
%for i=1:2
    % calculs
    grad = mongradient(theta);              % calcul du gradient
    H = monHessien(theta);                  % calcul de la matrice Hesienne
    B = H + lambda * eye(size(H));          % calcul d'une matrice proche de H def pos
    direction = -B\grad;                    % direction de descente
    theta_old = theta;                      % sauvegarde ancien theta
    theta = theta + direction;              % MAJ theta

    % calcul de J
    Jprec = J;
    J = moncritere(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');
```


Evolution de θ avec la méthode 3
θ0
θ57


Evolution de J avec la méthode 3

7 Conclusion

Après avoir épuré le code afin de ne conserver que le code permettant de calculer le θ final pour chacune des méthodes (suppression des affichages, de la sauvegarde des J, etc.), on mesure combien de temps met chacune des méthodes pour converger.

On trouve les résultats suivants :

- Méthode α fixé : 0.29 secondes
- Méthode α variable : 0.48 secondes
- Méthode quasi-Newton : 0.023 secondes

On voit que dans ce cas, la méthode quasi-Newton est particulièrement efficace comparé aux autre. Ce phénomène aurait probablement tendance à s'atténuer avec d'autres fonctions, on si nous étions dans un espace de plus grande dimension que $\mathbb{R}^2$, en effet, la méthode de Newton a une complicité en $O(n^3)$ rendant l'algorithme très gourmand et peu performant en grande dimension.