ECOLE D'INGÉNIEUR ENSEEIHT

Optimisation numérique
Rapport de projet

Florian GARIBAL

9 février 2017

Sommaire

Introduction

Ce rapport résume les différentes étapes du projet réalisé dans le cadre de l'UE Optimisation numérique. En effet il nous était demandé de produire une étude numérique à travers l'implémentation d'algorithmes de minimisation de fonctions. Ce projet était divisé en deux parties, la première partie concernait l'optimisation sans contrainte et la deuxième partie l'optimisation avec contrainte. Dans la première partie nous avons implémenté les algorithmes de Newton en utilisant la méthode des régions de confiance, conçu dans un premier temps avec le pas de Cauchy puis avec Moré-Sorensen. La deuxième partie consistait à implémenter la méthode du Lagrangien augmenté.

Chapitre 1

Optimisation sans contrainte

Le but de cette partie est la résolution de $\min_{x \in \mathbf{R}} f(x)$ avec f de classe C^2 sur $\mathbf{R^n}$.

1.1 Algorithme de Newton local

Dans cette première partie, on utilise le développement de Taylor du second ordre de la fonction au voisinage de l'itéré courant. En effet le but est de trouver le minimum de ce développement à chaque itération.
Afin de décider quand on arrête de chercher une solution plus précise, j'ai mis en palce quatre critères d'arrêts :

— Le nombre maximum d'itérations ;
— $\frac{\|\nabla_x\|}{\|\nabla_{x0}+\epsilon\|} \leq \epsilon$
— $\frac{\|x_{k1}-x_k\|}{\|x_k\|} \leq \epsilon$
— $\frac{\|f(x_{k1})-f(x_k)\|}{\|f(x_k)\|} \leq \epsilon$

Afin de vérifier que mes fonctions, notamment le calcul du gradient et de la hessienne, n'est pas appelé de trop nombreuses fois, j'ai mis en place des variables globales qui comptent le nombre d'appels à chaque fonctions.
De ce fait en éxécutant les tests suivants je peux savoir d'une part pour quelle raison s'est arrêté l'itération à l'aide des *flag* mais aussi combien de fois elle a appelé toutes mes fonctions.

1.1.1 Tests

On pose :
$f_1 : (x_1, x_2, x_3) \mapsto 2(x_1 + x_2 + x_3 - 3)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2$
$f_2 : (x_1, x_2) \mapsto 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

— Minimiser f_1 avec $x_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$: 1 itération ; flag 2

$x_{min} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, f(x_{min}) = 9,8608 * 10^{-32}$

Nombres d'évaluations des fonctions :

- $f_1 : 3$
- $gradient\_f_1 : 3$
- $hessienne\_f_1 : 2$

Interprétation :
Ce calcul de minimum converge en une seule itération car f_1 est une forme quadratique donc son développement de Taylor est exactement égal à f_1.

— Minimiser f_1, $x_0 = \begin{bmatrix} 10 \\ 3 \\ -2.2 \end{bmatrix}$: 5 itérations ; flag = 2

$x_{min} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $f(x_{min}) = 4.9698 * 10^{-29}$

Nombres d'évaluations des fonctions :

- f_1 : 3
- $gradient\_f_1$: 3
- $hessienne\_f_1$: 2

— Minimiser f_2, $x_0 = \begin{bmatrix} -1.2 \\ 1 \end{bmatrix}$: 3 itérations ; flag = 2

$x_{min} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $f(x_{min}) = 1.8527 * 10^{-11}$

Nombres d'évaluations des fonctions :

- f_1 : 11
- $gradient\_f_2$: 7
- $hessienne\_f_2$: 6

— Minimiser f_2, $x_0 = \begin{bmatrix} 10 \\ 0 \end{bmatrix}$: 3 itérations ; flag = 2

$x_{min} = \begin{bmatrix} 1.0004 \\ 1.0007 \end{bmatrix}$, $f(x_{min}) = 1.3279 * 10^{-7}$

Nombres d'évaluations des fonctions :

- f_1 : 7
- $gradient\_f_2$: 5
- $hessienne\_f_2$: 4

— Minimiser f_2, $x_0 = \begin{bmatrix} 0 \\ \frac{1}{200} + \frac{1}{10^{12}} \end{bmatrix}$: 2 itérations ; flag = 3

$x_{min} = \begin{bmatrix} -0.0000 \\ 2.5000 \end{bmatrix}$, $f(x_{min}) = 2.499999587298197 * 10^{+19}$

Nombres d'évaluations des fonctions :

- f_1 : 5
- $gradient\_f_2$: 4
- $hessienne\_f_2$: 3

Interprétation :

Le calcul de minimum pour la fonction f_2 peut ne pas converger selon le x_0 choisi car la fonction comporte plusieurs points où le gradient s'annule et qui ne sont pas forcément des minima. De ce fait la solution trouvée n'est pas forcément la bonne selon le x_0 choisi.

1.2 Région de confiance - Partie 1

Cette méthode permet d'éviter les problèmes que nous avons obtenu dans la partie précédente à savoir que selon le point initial la convergence global n'était pas garantie. Afin d'implémenter cette méthode nous avons d'abord implémenté le pas de Cauchy afin de rendre plus facile la résolution de ce problème.

1.2.1 Pas de Cauchy

Le pas de Cauchy permet de trouver une solution une solution approchée à un problème ce qui accélère la recherche de solutions dans les regions de confiance. Pour cela on restreint au sous ensemble engendré par le vecteur g_k.

Pour tester, on considère des fonctions quadratique de la forme $q(s) = s^T g + \frac{1}{2} s^T H s$ où l'on fait varier g et H.

- $g = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ et $H = \begin{bmatrix} 7 & 0 \\ 0 & 2 \end{bmatrix}$; Solution non trouvée
- $g = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$ et $H = \begin{bmatrix} 7 & 0 \\ 0 & 2 \end{bmatrix}$; $s = \begin{bmatrix} -0.9231 \\ -0.3077 \end{bmatrix}$
- $g = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$ et $H = \begin{bmatrix} 7 & 0 \\ 0 & 2 \end{bmatrix}$; $s = \begin{bmatrix} 2.6833 \\ -1.3416 \end{bmatrix}$

1.2.2 Tests

Ensuite on devait intégrer le pas de cauchy dans la méthode des régions de confiance. On réutilise les fonctions de la partie sur newton local :
$f_1 : (x_1, x_2, x_3) \mapsto 2(x_1 + x_2 + x_3 - 3)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2$
$f_2 : (x_1, x_2) \mapsto 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

Les résultats complets des tests sont disponibles avec la comparaison de convergence de l'algorithme de Moré Sorensen.

Interprétation :

• La relation qui lie la fonction test f_1 et son modèle de Taylor à l'ordre 2 est une relation d'égalité. En effet f_1 est égal à son développement de Taylor à l'ordre 2.

1.3 Région de confiance - Partie 2

Cette deuxième partie consiste à réimplémenter l'algorithme des régions de confiance en utilisant une autre méthode que le pas de Cauchy pour approximer la solution. En effet le pas de Cauchy ne garantit pas une convergence rapide en général. On va donc implémenter l'algorithme de Moré-Sorensen. Pour cela, on aura besoin de résoudre des équations de la forme $\phi(\lambda) = 0$ où ϕ sera une fonction **non linéaire** à variables réelle.

1.3.1 Algorithme de Newton pour les équations non linéaires

1.3.1.1 Tests

On pose :

$\phi_1(\lambda) = \|s(\lambda)\|^2 - \delta^2$

$\phi_2(\lambda) = \frac{1}{\|s(\lambda)\|^2} - \frac{1}{\delta^2}$

Interprétation :

- Les conditions d'arrêts sont les suivantes :
  - **Flag 1** : $|f(\lambda)| \leq \epsilon$
  - **Flag 2** : $nbIter \geq maxIter$
- Si l'utilisateur ne fournit pas de couple $(\lambda_{min}, \lambda_{max})$, on peut les déterminer à l'aide d'un suite du type $\lambda_{min_k} = -\lambda_1 + 2^k$ en ayant comme critère d'arrêt $s(\lambda_{min}^2 < \delta^2$.
  On peut faire de même avec λ_{max}.

Résultats des tests :

- $\|s(\lambda)\|^2 = \frac{4}{(\lambda+2)^2} + \frac{36}{(\lambda+14)^2}$ et $\delta = 0.5$
  - $res = 3.4965$
  - Nombre d'itérations : 6
  - Flag : 1
- $\|s(\lambda)\|^2 = \frac{4}{(\lambda-38)^2} + \frac{400}{(\lambda+20)^2}$ et $\delta \in \{0.2, 0.7\}$
  - $res = -70.0536$
  - Nombre d'itérations : 26
  - Flag : 1

1.3.2 Algorithme de Moré-Sorensen

1.3.2.1 Tests

Afin de tester la validité de notre algorithme, un étalon nous était fourni. En effet ce code étant fonctionnel, nous pouvions comparer l'exactitude de nos résultats et ainsi débugger notre fonction.
Pour quatres tests sur six, ma fonction renvoie exactement le bon résultat. En revanche pour les deux autres tests j'ai une erreur de l'ordre de 10^-3 à peu près. J'ai essayé de voir pourquoi mais je n'ai pas trouvé à quel moment cette erreur peut avoir lieu.
Ci dessous les comparaisons de résultats avec l'étalon.

		Etalon	**Ma fonction**
Test 1	$s*$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$
	$\lambda*$	0	0
Test 2	$s*$	$\begin{bmatrix} -0.8571 \\ -1.0000 \end{bmatrix}$	$\begin{bmatrix} -0.8571 \\ -1.0000 \end{bmatrix}$
	$\lambda*$	0	0
Test 3	$s*$	$\begin{bmatrix} \mathbf{2.000} \\ -0.0769 \end{bmatrix}$	$\begin{bmatrix} \mathbf{1.9985} \\ -0.0769 \end{bmatrix}$
	$\lambda*$	**3**	**3.0007**
Test 4	$s*$	$\begin{bmatrix} 2 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 0 \end{bmatrix}$
	$\lambda*$	2	2
Test 5	$s*$	$\begin{bmatrix} 1.2606 \\ -1.5527 \end{bmatrix}$	$\begin{bmatrix} 1.2606 \\ -1.5527 \end{bmatrix}$
	$\lambda*$	1.8035	1.8035
Test 5	$s*$	$\begin{bmatrix} -0.1053 \\ \mathbf{1.9972} \end{bmatrix}$	$\begin{bmatrix} -0.1053 \\ \mathbf{1.9889} \end{bmatrix}$
	$\lambda*$	15	15

1.3.3 Intégration dans l'algorithme des régions de confiance

L'algorithme de Moré Sorensen étant une alternative à l'algorithme du pas de Cauchy, il est nécessaire de comparer leur rapidité de convergence ainsi que leur précision.
Ci dessous la comparaison des résultats selon l'algorithme d'approximation de solution utilisé :
$f_1 : (x_1, x_2, x_3) \mapsto 2(x_1 + x_2 + x_3 - 3)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2$
$f_2 : (x_1, x_2) \mapsto 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

• La décroissance avec le Pas de Cauchy est toujours moins bonne que la convergence de l'algorithme de Moré-Sorensen.

• Les avantages et les inconvénients de chacune des méthodes sont les suivants :

— **Pas de Cauchy :**
  — Avantages : Converge plus rapidement que Moré-Sorensen, en moins d'itérations, sur des x_0 bien choisi
  — Inconvénients : Converge mal si x_0 pas bien choisi

— **Moré-Sorensen :**
  — Avantages : Converge en très peu d'itérations
  — Inconvénients : Nombre de calculs très important

		Pas de Cauchy	**Moré Sorensen**
Test 1 :	nbIter	49	3
$minf_1(x)$	x_{min}	$\begin{bmatrix}1\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}$
$x_0 = \begin{bmatrix}1\\0\\0\end{bmatrix}$	$f(x_{min})$	$2.9316 * 10^{-15}$	0
Test 2 :	nbIter	53	6
$minf_1(x)$	x_{min}	$\begin{bmatrix}1\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}$
$x_0 = \begin{bmatrix}10\\3\\-2.2\end{bmatrix}$	$f(x_{min})$	$3.1371 * 10^{-14}$	$9.8608 * 10^{-32}$
Test 3 :	nbIter	11994	30
$minf_2(x)$	x_{min}	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$
$x_0 = \begin{bmatrix}-1.2\\1\end{bmatrix}$	$f(x_{min})$	$1.0234 * 10^{-11}$	0
Test 4 :	nbIter	12	36
$minf_2(x)$	x_{min}	$\begin{bmatrix}0.9950\\0.9899\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$
$x_0 = \begin{bmatrix}10\\0\end{bmatrix}$	$f(x_{min})$	$2.567 * 10^{-5}$	0
Test 5 :	nbIter	49	3
$minf_2(x)$	x_{min}	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$
$x_0 = \begin{bmatrix}0\\\frac{1}{200} + \frac{1}{10^12}\end{bmatrix}$	$f(x_{min})$	$1.1179 * 10^{-15}$	$4.9304 * 10^{-30}$

Chapitre 2

Optimisation avec contraintes

Le but de cette partie est de résoudre les problèmes de la forme :
$\min_{x \in \mathbb{R}^n} f(x)$ sous la contrainte : $x \in C$ où C est un ensemble non vide de $\mathbb{R}^n$

2.1 Méthode du Lagrangien augmenté

J'ai implémenté la méthode du Lagrangien augmenté mais je n'ai pas eu le temps de tester et débugger si besoin. Le programme implémenté est donné en annexe.

Conclusion

Ce projet m'a permis d'implémenter et de comparer les différentes méthodes permettant de minimiser une fonction. En effet j'ai pu mettre en place la résolution de problème à l'aide de l'algorithme de **Newton Local**. Ensuite j'ai développé l'algorithme du **Pas de Cauchy** que j'ai intégrer à la méthode des **régions de confiance**. Cependant cette méthode (Pas de Cauchy) ne permettait pas de converger efficacement dans tous les cas et de ce fait j'ai implémenté l'algorithme de **Moré-Sorensen** qui, lui, converge beaucoup plus rapidement et efficacement vers la solution. Pour finir j'ai essayé d'implémenter le **Lagrangien augmenté** mais je n'ai pas réussi à le développer correctement de sorte à ce qu'il me renvoie des résultats corrects.

Ce projet fut très intéressant bien que très compliqué et très long (étalé sur une longue période). En effet le sujet et les explications en TPs étaient très complémentaires et permettaient de bien mettre en place les algorithmes et méthodes demandées. En revanche, le projet étant étalé sur plus de 2 mois, ce fut très compliqué de s'organiser, de se rendre compte de l'avancement et de la charge de travail qu'il restait à faire.

Annexes

2.2 Lagrangien augmenté

2.1.1 Fonctions auxiliaires

Calcul du gradient :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Fonction grad_lagrangien_generic qui calcule le gradient du lagrangien générique
%% gradf           :
%% jacobc          :
%% c               :
%% xk              :
%% lambdak         :
%% muk             :
%%%%%%%%%%%%%%% SORTIES
%% y               :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [y] = grad_lagrangien_generic(gradf, jacobc, c, xk, lambdak, muk)
y = gradf(xk) + jacobc(xk)'*(lambdak + muk*c(xk));
end
```

Calcul de la Hessienne :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Fonction hess_lagrangien_generic qui calcule la hessienne du lagrangien générique
%% hessf           :
%% jacobc          :
%% hess            :
%% xk              :
%% muk             :
%%%%%%%%%%%%%%% SORTIES
%% y               :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [y] = hess_lagrangien_generic(hessf, jacobc, hessc, xk, muk)
    y = hessf(xk) + muk*jacobc(xk)'*jacobc(xk) + hessc(xk, lambdak+mukc(xk));
end
```

Fonction de résolution de problème sans contraintes :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Fonction min_lagrangien qui implémente la méthode des régions de confiance
%% f              :
%% c              :
%% xk             :
%% lambda         :
%% muk            :
%%%%%%%%%%%%%%% SORTIES
%% y              :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [y] = min_lagrangien(f, c, xk, lambdak, muk)
    y = f(xk) + lambdak'*c(x) + (1/2)*muk*norm(c(x))^2;
end
```

2.1.2 Algorithme du Lagrangien augmenté

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Fonction lagrangien_augmente qui implémente la méthode du lagrangien augmenté
%% f              :
%% c              :
%% gradient       :
%% jacobienne     :
%% hessienne      :
%% mu0            :
%% tho            :
%% ethaChapo0     : 0.1258925
%% alpha          :
%% betha          :
%% x0             :
%% lambda0        :
%% delta0         :
%% deltaMax       :
%% n1             :
%% n2             :
%% gamma1         :
%% gamma2         :
%% nbMaxIter         :
%% choixMethode : Méthode choisie
%%                     choixMethode = 0 -> Newton local
%%                     choixMethode = 1 -> Région de confiance
%%%%%%%%%%%%%%% SORTIES
%% x              :
%% lambda         :
%% mu             :
%% flag           :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x, lambda, mu, flag] = lagrangien_augmente(f, c, gradient, jacobienne, hessienne, mu0, tho, e

nbIter  = 0;
epsilon0 = 1/mu0;
etha0 = etaChapo0/(mu0^alpha);

xk = x0;
lambdak = lambda0;
muk = mu0;
ethak = etha0;
```

```
epsilonk = 1/muk;

continuer = true;

% Tant que pas convergence
while continuer

    % Construction du Lagrangien augmenté
    f = @(x)Lagrangien_generic(f, c, xk, lambdak, muk);
    grad = @(x)grad_Lagrangien_generic(gradf, jacobc, c, xk, lambdak, muk);
    hess = @(x)hess_Lagrangien_generic(hessf, jacobc, hessc, xk, muk);

    % Résolution selon méthode choisie
    % choixMethode = 0 -> Newton local
    % choixMethode = 1 -> Région de confiance
    if choixMethode == 0
        [xk, flag] = newton_local(f, xk, grad, hess, epsilonk, nbMaxIter);

    elseif choixMethode == 1
        % J'utilise la méthode de Moré Sorensen car elle converge plus rapidement
        [xk, useless, flag] = region_confiance(xk, deltaMax, delta0, nbMaxIter, gamma1, gamma2, n1, n2,
    end

    % Critères d'arrêts
    if norm(c(ck)) < epsilon2
        flag = 1;
        continuer = false;

    elseif nbIter >= nbMaxIter
        flag = 2;
        continuer = false;
    else
        if norm(c(xk)) < ethak
            % MAJ multiplicateurs
            lambdak = lambdak + muk*c(xk);
            %muk = muk
            epsilonk = epsilonk/muk;
            ethak = ethak/(muk^betha);
            nbIter  = nbIter  + 1;
        else
            % MAJ pénalités
            %lambdak = lambdak
            muk = tho*muk;
            epsilonk = epsilon0/muk;
            ethak = etaChapo0/(muk^alpha);
            nbIter  = nbIter  + 1;
        end
    end
end
end
```