TP-Projet ENSEEIHT
Méthodes numériques pour les problèmes d'optimisation

ABDELLATIF MAMOUN

Janvier 2020

Sommaire

1 Introduction

Ce projet consite à découvrir et résoudre des problèmes d'optimisation par des méthodes numériques. On se propose dans ce document d'étudier et expliquer dans un premier lieu des problèmes sans contraintes, et dans un second lieu, on envisage de résoudre des problèmes avec contraintes.

2 Optimisation sans contraintes

Dans cette partie, on s'interesse à résoudre des problèmes de genre :

$$\min_{x \in \mathbb{R}^n} f(x)$$

Où f est une fonction de C^2 sur $\mathbb{R}^n$. On s'interesse à résoudre ce problèmes via deux méthodes, voire algorithmes : `Algorithme de Newton` et `Algorithme de région de confiance`.

2.1 Algorithme de Newton

L'algorithme de Newton locale est une méthode qui permet de résoudre des problèmes sans contraintes. Son principe est de trouver une solution du problème suivant : $\nabla f(x) = 0$.
De ce fait, on utilisera bien le développement de Taylor du second ordre de la fonction au voisinage de l'itéré courant. Une documentation de cet algorithme est disponible via la commande sur shell : help NewtonLocale.

2.1.1 Tests

On se propose ici d'étudier la validité de notre algorithme via les tests suivants, et on travaille avec les paramètres d'encadrement suivant :
tolerance = $10^{-6} \times [1,1,1]$, et maxIter = 100.

Fonction	x0	xmin	f(xmin)	nbrIter	flag
f1	1 0 0	1 1 1	1.9722e-31	1	0
f1	10 3 -2.2	1 1 1	5.0191e-29	1	0
f2	-1.2 1	1 1	1.8527e-11	5	0
f2	2 0 10 0	1 1	1.3279e-07		3
f2	0 0.0050	0 2.5e+19	2.4998e+19	4	1

2.1.2 Interpretations

On remarque que la fonction f1 converge dans tous les cas durant la première itération. Ceci est dû au fait que la méthode de Newton Locale a une convergence quadratique dans les favorables cas, et c'est le cas ici pour la fonction *f1*.
Pourtant, la fonction *f2* ne converge pas dans tous les cas, surtout si la hissiène $\nabla^2 f$ n'est pas inversible en une itération, et si le point de départ est trop éloigné de la solution, et c'est le cas ici pour le point initial x_{23} =[0 ;0.0050].

2.2 Région de confiance - Partie 1

Dans cette partie, on se propose de résoudre notre problème d'optimisation via l'algorithme des régions de confiance. En fait, cet algorithme est optimal, et nous permet d'éviter des problèmes de convergence, comme au dessus. Ceci peut s'expliquer du fait qu'on contrôle un pas de confiance à chaque itération, ce qui permet de dériger et évoluer l'algorithme sans perturbation vers la solution. Une documentation est disponible via la commande help regionConfiance.
On se propose dans ce cas, de calculer le pas via la méthode de Cauchy, qui permet de trouver le pas dans le sens inverse du ∇f. On fournit ainsi une documentation via la commande au shell :help pasCauchy.

2.2.1 Pas de Cauchy

On s'intersse dans ce qui suit à réaliser des tests sur l'algorithme du `Pas de Cauchy`. Les protypes sont réalisés sur les paramètres suivants :

$$g_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad H_1 = \begin{bmatrix} 7 & 0 \\ 0 & 2 \end{bmatrix}$$

$$g_2 = \begin{bmatrix} 6 \\ 2 \end{bmatrix} \quad H_2 = \begin{bmatrix} 7 & 0 \\ 0 & 2 \end{bmatrix}$$

$$g_3 = \begin{bmatrix} -2 \\ 1 \end{bmatrix} \quad H_3 = \begin{bmatrix} -2 & 0 \\ 0 & 10 \end{bmatrix}$$

et on donne comme région confiée $delta = 10$. Les résultats sont comme suit :

```
>> testCauchy
Tests sur le calcul de pas cas de Cauchy :
-------------- Quadratique 1 --------------

s1 =

     0
     0

-------------- Quadratique 2 --------------

s2 =

   -0.9231
   -0.3077

-------------- Quadratique 3 --------------

s3 =

    5.0000
   -2.5000

>>
```

FIGURE 1 – Tests Pas de Cauchy.

2.2.2 Algorithme : Région de confiance

On se propose ici d'étudier la validité de notre algorithme via les tests suivants, et on travaille avec les paramètres initiaux suivants :
maxIter=50, tol = 10^{-3}[1,1,1], delta0=2, gamma1=0.5, gamma2=2, eta1=0.25 et eta2=0.75.

Fonction	x0	xmin	f(xmin)	nbrIter	flag
f1	1 0 0	1.003 1 0.9975	1.7525e-05	16	0
f1	10 3 -2.2	1.0124 0.9986 0.9847	4.1919e-04	19	0
f2	-1.2 1 2 10 0 0	-0.9213 0.8620 1 1	3.7089 18.7338	51	3 7
f2	0 0.0050	0.6614 0.4414	0.1135	51	3

2.2.3 Interpretations

- La fonction f1 est égale à son modèle de Taylor à l'ordre 2. On garantit alors la convergence de cet algorithme. De plus, l'algorithme de Newton serait plus rapide que RC-Pas de Cauchy sur cette fonction.
- L'algorithme RC-Pas de Cauchy peut-être amélioré, en jouant sur les paramètres tolerance, et maxIter. Par les multiplicateurs γ_1, γ_2, η_1, et η_2 ne changent pas la convergence et les circonstances de la convergence de notre algorithme.
- On remarque aussi que la diminution de la tolerance (ordre $10^{-6} par exemple) permet de garantir de plus en plus$

2.3 Région de Confiance - Partie 2

On se propose d'étudier dans ce cas l'algorithme de région de Confiance. Comme étant un cadre générique, on s'interesse à le réaliser, en calculant le pas via la méthode du `gradient conjugué tronqué`. une documentation du dernier algorithme est disponible via la commande help gradConjugueTronque.

2.3.1 Algorithme : Gradient Conjugué Tronqué

On s'intersse dans ce qui suit à réaliser des tests sur l'algorithme du `Gradient Conjugué Tronqué`. Les protypes sont réalisés sur les paramètres suivants :

$$g_4 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad H_4 = \begin{bmatrix} -2 & 0 \\ 0 & 10 \end{bmatrix}$$

$$g_5 = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad H_5 = \begin{bmatrix} 4 & 6 \\ 6 & 5 \end{bmatrix}$$

$$g_6 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \quad H_6 = \begin{bmatrix} 4 & 0 \\ 0 & -15 \end{bmatrix}$$

et on donne comme région confiée $delta = 1$. On obtiendra commme résultat :

```
>> testgradConjTronque
Tests sur le calcul de la méthode Gradient Conjugué Tronqué :
-------------- Quadratique 4 --------------

s4 =

     0
     0

-------------- Quadratique 5 --------------

s5 =

   -1.0000
   -0.0000

-------------- Quadratique 6 --------------

s6 =

   -0.5000
         0
```

FIGURE 2 – Tests Lagrangien augmenté tronqué.

2.3.2 Algorithme : Région de confiance

On se propose ici d'étudier la validité de notre algorithme via les tests suivants, et on travaille avec les paramètres initiaux suivants :
maxIter=50, tol = 10^{-3}[1,1,1], delta0=2, gamma1=0.5, gamma2=2, eta1=0.25 et eta2=0.75.

Fonction	x0	xmin	f(xmin)	nbrIter	flag
f1	1 0 0	1 1 1	2.4652e-31	2	0
f1	10 3 -2.2	1 1 1	2.4652e-30	4	0
f2	-1.2 1	0.9 0.8064	0.0104	26	0
2 0	10 0	3.9624 15.7004	8.7757		5
f2	0 0.0050	1 1	1.7826e-13	17	0

2.3.3 Interpretations

- On peut dire que la méthode converge bien dans f_1. Ceci, est dû au fait que f_1 est quadratique. L'algorithme est à peu près de même complexité que la méthode de Newton. - On garantit dans la majorité des cas la convergence de l'algorithme. Pourtant l'améliorer en dimuniant la tolerance.

2.3.4 Comparaison avec RC-Pas de Cauchy

L'algorithme de RC-Gragient conjugué est plus efficace que RC-Cauchy. En fait, le premier algorithme permet de trouver des solutions exactes en peu nombres d'itérations, voire en durée de temps. Ceci est dû au fait que le pas de Cauchy cherche son coefficient directeur sur un intervalle ce qui parfois

mène à une perte de direction exacte. Pourtant, la méthode du langrangien tronqué permet de la localiser sur une sphère, ce qui assure la récupération du bon pas.

3 Optimisation avec contraintes

Dans cette partie, on s'interesse à résoudre des problèmes de genre :

$$\min_{x \in \mathbb{R}^n} f(x) \qquad sous la contrainte \; x \in \mathrm{C}$$

Où f est une fonction de C^2 sur $\mathbb{R}^n$. On s'interesse à résoudre ce type de problèmes via la méthode du `Lagrangien Augmenté`. La méthode du langrangien augmenté permet de résoudre un problème avec contraintes avec pénalisation. Il s'agit ici d'essayer de résoudre un tel problème en une suite de résoudre de problèmes sans contraintes.

3.1 Interpretations

Malheureusement, on n'a pas pu fonctionner cet algorithme parfaitement. En fait, j'ai trouvé des problèmes qui sont relatifs aux paramètres d'encadrement, et un peu de calcul (hissiène du lagrangien proposé, résolu). Par contre, j'ai constaté quelques remarques, qui peuvent être utiles sur l'évolution de la convergence du problème :
- Le résultat peut être exact si on donne un λ proche à sa valeur exacte.
- mu doit être grand pour qu'on s'approche de la solution. On peut aussi augmenter τ pour en réaliser.

4 Conclusion

Ce projet nous permet de bien découvrir le côté numérique de la résolution des problèmes d'optimisation, qui sont utiles surtout pour des grands et coûteux problèmes. On a pu réaliser dans un premier lieu la réalisation des problèmes sans contraintes, et on a conclu que l'algorithme *Région de confiance avec le lagrangien conjugué tronqué* est optimal. On a pu aussi découvrir comment résoudre des problèmes avec contraintes, via la méthode de Lagrangien augmenté.