

nom :

Master Maths 1 2010-2011 Optimisation

TP noté : méthode du gradient conjugué et méthode de Newton

Déroulement : suite à votre travail, vous devez rendre cet énoncé sur lequel vous devez écrire votre nom ainsi que les réponses aux questions posées : des trous sont prévus à cet effet. Néanmoins il est conseillé de rédiger préalablement au brouillon.

Recommandation : BIEN LIRE L'ENONCE!

Il est interdit d'utiliser INTERNET sauf mention explicite de ma part !

1 Le problème

On considère la fonction :

$$f(x) = x_1^4 + (x_1 + x_2)^2 + (e^{x_2} - 1)^2$$

On va minimiser cette fonction en utilisant la méthode du gradient conjugué puis la méthode de Newton. Il est à peu près clair que $x^* = [0; 0]$ est le minimum de f .

Avant de passer aux questions, créez-vous un sous répertoire puis copier un fichier de mon répertoire principal : dans un terminal rentrez les commandes suivantes :

```
mkdir OptimTPnote  
cd OptimTPnote  
cp ~/pincon/examlib.sci .  
scilab &
```

Dans scilab un `exec examlib.sci`; lui permettra de connaître les fonctions contenues dans le fichier!

Partie 1

- (1 point) dans un nouveau fichier de nom `examtp.sci` (ou autre...) écrire le code scilab permettant d'évaluer cette fonction en un point $x \in \mathbb{R}^2$. L'entête de cette fonction doit être `function y = f(x)`. Recopier votre fonction dans l'espace ci-dessous.

```
function y = f(x)
```

```
endfunction
```

2. (1 point) calculer les dérivées partielles de la fonction f (d'abord au brouillon puis proprement sur cette feuille) :

$$\frac{\partial f}{\partial x_1}(x) =$$

$$\frac{\partial f}{\partial x_2}(x) =$$

3. (1 point) à la suite de votre fonction scilab **f**, écrire une fonction scilab calculant le gradient de f (sous la forme d'un vecteur colonne (*)). L'entête de cette fonction doit être **function y = grd_f(x)**. Recopier votre fonction dans l'espace ci-dessous.

```
function y = grd_f(x)
```

```
endfunction
```

(*) Aide : si vous définissez le vecteur **y** par ses composantes (cad **y(1) = ...** puis **y(2) = ...**) vous obtiendrez bien un vecteur colonne et pas un vecteur ligne.

4. (1 point) Il est possible que votre gradient soit faux (erreur dans le calcul et ou erreur dans le codage). Vous allez le vérifier en utilisant la fonction **derivative** de scilab qui permet de calculer un gradient par différences finies. Ecrire ci-dessous les valeurs que vous observez (rmqs : 1/ derivative renvoie le gradient comme un vecteur ligne, c'est normal d'où la transposition pour calculer la norme de la différence!, 2/ avec le format d'affichage par défaut (peu précis) vous devez obtenir les mêmes "nombres" pour les 2 méthodes d'où l'intérêt de calculer la norme de la différence).

```
x = [2;3]
```

```
y1 = grd_f(x) donne :
```

```
y2 = derivative(f,x) donne :
```

```
|| y1 - y2 || = norm(y1 - y2') donne :
```

```
x = [-2;1]
```

```
y1 = grd_f(x) donne :
```

```
y2 = derivative(f,x) donne :
```

```
|| y1 - y2 || = norm(y1 - y2') donne :
```

Si la différence en norme est inférieure à 10^{-7} on considérera que votre gradient est bon. Dans le cas contraire, retournez aux questions précédentes.

5. (2 point) écrire un script qui utilise la fonction `gradient_conjugué` du fichier `examlib.sci` pour trouver le minimum de f ; on utilisera d'abord le point de départ $x_0 = [1;1]$ puis le point de départ $x_0 = [-1;3]$ avec une tolérance sur la norme du gradient (paramètre `tol`) de 10^{-8} (on pourra prendre aussi `pas=0.1` et `itermax=20`). Noter ci dessous les résultats obtenus :

(a) en partant de : $x_0 = [1;1]$, j'obtiens :

```
norm(xopt - x^*) = norm(xopt) =
norm(gopt) =
iter =
```

(b) en partant de : $x_0 = [-1;3]$, j'obtiens :

```
norm(xopt - x^*) = norm(xopt) =
norm(gopt) =
iter =
```

6. (1 point) Calculer la matrice hessienne de f (d'abord au brouillon puis noter votre résultat ci-dessous) :

$H_f(x) =$

7. (1 point) A la suite de votre fonction scilab `grd_f`, écrire une fonction scilab qui, étant donné x , calcule la matrice Hessienne de f en x . Recopier votre fonction dans l'espace ci-dessous.

```
function H = hess_f(x)
```

```
endfunction
```

8. (1 point) Comme pour le gradient, vérifier votre Hessienne grâce à la fonction scilab `derivative` (il suffit de l'utiliser sur la fonction `grd_f`) :

avec $x = [2;3]$

$H_1 = \text{hess_f}(x)$ donne :

en utilisant `derivative` j'obtiens $H_2 =$:

$\max(\text{abs}(H_1 - H_2)) =$

avec $x = [-2;1]$

$H_1 = \text{hess_f}(x)$ donne :

en utilisant derivative j'obtiens H2 = :

$\max(\text{abs}(H1-H2)) =$

Si l'erreur (mesurée par l'écart maximum en valeur absolue entre les deux matrices) est inférieure à 10^{-6} on considèrera que votre fonction `hess_f` est correcte (sinon retour aux questions précédentes).

9. (1 point) Soit une fonction f dérivable. Quelle est la condition pour qu'un vecteur d soit une direction de descente pour f au point x (cad tel qu'il existe $\eta > 0$ tel que $f(x + td) < f(x)$, $\forall t \in]0, \eta[$). Cette condition est (l'écrire juste en dessous) :

2 Partie 2 : la méthode de Newton

Nous avons vu en cours qu'une condition nécessaire d'optimalité est :

$$\nabla f(x^*) = 0$$

Une autre idée pour minimiser une fonction f serait donc de trouver un zéro de son gradient, ce qui suggère d'utiliser la méthode de Newton sur le gradient de f . Par Taylor on peut écrire :

$$0 = \nabla f(x^*) = \nabla f(x^{(k)}) + Hf(x^{(k)})(x^* - x^{(k)}) + o(x^* - x^{(k)})$$

et si $x^{(k)}$ est assez près de l'optimum x^* alors on doit pouvoir négliger le reste. Par contre même dans ce cas on ne va pas obtenir exactement x^* mais une approximation notée $x^{(k+1)}$, ce qui nous donne l'itération de Newton :

$$x^{(k+1)} = x^{(k)} - Hf(x^{(k)})^{-1} \nabla f(x^{(k)})$$

pour laquelle la matrice Hessienne $Hf(x^{(k)})$ doit être inversible. On peut ré-interpréter cette méthode différemment. Pour cela poser $d^k = -Hf(x^{(k)})^{-1} \nabla f(x^{(k)})$, on remarque alors que l'itération de Newton s'écrit :

$$x^{(k+1)} = x^{(k)} + \tau d^k \text{ avec } \tau = 1$$

Résoudre maintenant la question suivante :

Question 10 : (2 points) sous les hypothèses $Hf(x^{(k)})$ définie positive et $\nabla f(x^{(k)}) \neq 0$, montrer que d^k est une direction de descente pour f en $x^{(k)}$. Ecrire votre "démonstration" dans l'espace ci-dessous (après l'avoir écrite au brouillon...).

La condition " $Hf(x^{(k)})$ définie positive" sera au moins vérifiée près d'un minimum local "régulier" (on entend par là que $\nabla f(x^*) = 0$ et $Hf(x^*)$ définie positive). Ceci suggère d'utiliser la stratégie suivante :

1. étant donné $x^{(k)}$, on calcule $g^k = \nabla f(x^{(k)})$ et $H_k = H_f(x^{(k)})$;
2. on calcule d^k par résolution du système linéaire $H_k d^k = -g^k$
3. si cette résolution échoue ou si d^k n'est pas une direction de descente, on fait une itération de gradient à pas optimal qui permet d'obtenir un $x^{(k+1)}$
4. sinon, on tente l'itération $x^{new} = x^{(k)} + \tau d^k$ avec au départ $\tau = 1$ (le pas de Newton), et si $f(x^{new}) \geq f(x^{(k)})$ on recommence avec un pas plus petit, par exemple $\tau \leftarrow \tau/2$ et ceci jusqu'à ce que la condition de décroissance sur f soit satisfaite ; lorsque c'est le cas, on pose $x^{(k+1)} = x^{new}$
5. aller en 1

Dans le cas de la fonction particulière f que nous utilisons dans ce TP, on peut montrer que sa matrice Hessienne est toujours définie positive, ce qui simplifie la stratégie précédente puisque l'étape 3 n'est plus nécessaire.

Question 11 : (3 points) écrire cet algorithme (l'algorithme simplifié constitué des étapes 1,2 et 4) en pseudo code en rajoutant le test d'arrêt usuel sur le gradient. Dans les logiciels comme matlab, octave, scilab, pour résoudre un système linéaire $Ax = b$, on utilise $\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$, notation qu'on pourra reprendre dans le pseudo-code (rmq : le pseudo-code ne suit pas de règles très précises, il s'agit d'écrire l'algorithme de manière à faciliter le passage au langage informatique utilisé (ici scilab). Ecrire votre pseudo-code dans l'espace ci-dessous :

Question 12 : (3 points) coder cet algorithme en langage scilab par une fonction d'entête :

```
function [xopt, fopt, gopt, iter] = newton(x0, f, grdf, hessf, tol, itermax, verb)
```

Recopier votre code dans l'espace si-dessous :

Question 13 : (2 points) tester votre code sur la fonction f en partant des mêmes points que pour le gradient conjugué. Noter ci dessous les résultats obtenus :

1. en partant de : $x_0 = [1; 1]$, j'obtiens :

```
norm(xopt - x^*) = norm(xopt) =  
norm(gopt) =  
iter =
```

2. en partant de : $x_0 = [-1; 3]$, j'obtiens :

```
norm(xopt - x^*) = norm(xopt) =  
norm(gopt) =  
iter =
```

Rmq : sur ces deux exemples la méthode de Newton utilise un peu plus d'itérations. Cependant si on part plus près de l'optimum x^* elle en fera moins.