

Master Maths 1 2006-2007

Optimisation

TP3 : Méthodes de gradient 1

Dans ce TP, on commence par programmer la méthode du gradient à pas fixe et ensuite, la méthode du gradient à pas optimal mais sur une fonction quadratique (car on connaît la formule pour le pas!). On complètera ce deuxième code la prochaine fois avec une minimisation unidimensionnelle de sorte à avoir un code valable pour tout type de fonction.

1 Cas test

Pour commencer on considèrera le cas test suivant :

$$f_1(x) = \frac{1}{2}x^\top Ax + b^\top x, \quad \text{avec } A = \begin{bmatrix} 0.06 & 0.12 \\ 0.12 & 0.98 \end{bmatrix} \text{ et } b = \begin{bmatrix} 0.18 \\ 1.1 \end{bmatrix}$$

dont l'optimum est $\bar{x} = [1;1]$. D'autres tests (de vrais tests non linéaires) seront fournis ultérieurement. Néanmoins le cas quadratique est déjà instructif : pourquoi ?

Ecrire dans un fichier `testquad.sci` la fonction f_1 en langage scilab, ainsi qu'une autre fonction donnant son gradient. Les entêtes seront :

```
function y = f1(x)
    // x doit être un vecteur 2x1 (ou encore une matrice 2xm)
    A = [ 0.06 , 0.12 ; 0.12 , 0.98 ]; b = [0.18 ; 1.1];
    y = ....
endfunction
function df = grd_f1(x)
    // x doit être un vecteur 2x1 (ou encore une matrice 2xm)
    A = [ 0.06 , 0.12 ; 0.12 , 0.98 ]; b = [0.18 ; 1.1];
    df = ....
endfunction
```

2 Le gradient à pas fixe

Coder cette méthode comme une fonction scilab d'entête :

```
function [xopt, fopt, gopt, iter] = gradient_pas_fixe(x0, f, grdf, ...
                                                    tau, tol, itermax, verb)

// x0 : point de départ
// f : fonction à optimiser
// grdf : gradient de cette fonction
// tau : le pas
// tol : la tolérance : on sort lorsque || grdf(x) || <= tol
// itermax : le nb max d'itérations
// verb : booléen (si vrai alors on affiche la valeur de f, de son
//          gradient et du x courant à la fin de chaque itération)
```

En sortie, `xopt` est la valeur optimale trouvée pour x , `fopt` et `gopt` la valeur de la fonction et du gradient pour cette valeur et `iter` est le nombre d'itérations.

Tester avec par exemple : `x0 = [3;4]`, `itermax = 400`, `tau = 0.5` et `tol = 1e-5`; par la suite, expérimentez avec différentes valeurs de `tau`, et de `x0`. Dans cette méthode appliquée sur une fonction quadratique on connaît la valeur optimale pour le pas (fixe) :

$$\tau_{opt} = \frac{2}{\lambda_1 + \lambda_2}$$

Calculez cette valeur¹ et testez là.

Pour les affichages on pourra utiliser la fonction suivante :

```
function print_vector(x,str)
    n = length(x)
    printf("\n %s = [ %g",str, x(1))
    for i=2:n, printf(",%g",x(i)), end
    printf("]")
endfunction
```

3 Le gradient à pas optimal (sur une fonction quadratique)

Retrouver la formule donnant le pas optimal dans le cas d'une fonction quadratique.

Coder cette méthode comme une fonction scilab d'entête :

```
function [xopt, fopt, gopt, iter] = gradient_pas_opt(x0, A, b, tol, ...
                                                    itermax, verb)

// x0 : point de départ
// A, b : matrice définissant une fonction quadratique
// tol : la tolérance : on sort lorsque la norme du gradient est <= à tol
// itermax : le nb max d'itérations
// verb : booléen (si vrai alors on affiche la valeur de f, de son
//          gradient et du x courant à la fin de chaque itération)
```

Expérimenter avec les mêmes paramètres (`x0`, `tol` et `itermax`).

¹la fonction `spec` de scilab permet de calculer les valeurs propres d'une matrice.

Master Maths 1 2005-2006

Optimisation

TP4 : Méthodes de gradient 2

On complète le dernier TP *Méthodes de gradient 1*. Un code scilab pour la minimisation unidimensionnelle sera fourni pour gagner du temps, vous pouvez donc passer la section suivante et la lire ultérieurement pour comprendre le code fourni.

1 L'algorithme de la section dorée

Il s'agit de coder en scilab cet algorithme robuste de recherche unidimensionnelle d'un minimum. Il ne faut pas oublier que la fonction de départ est à plusieurs variables et que l'on cherche un minimum dans une certaine direction d . On définit donc :

$$g(t) = f(x_0 + td)$$

Et on suppose que l'on a déjà encadré le minimum dans un intervalle $[\lambda, \mu]$ (ce problème d'encadrement est traité plus loin). L'algorithme s'écrit alors (cf cours) :

```
 $\lambda_1 = \lambda; \mu_1 = \mu$ 
pour  $i = 1, 2, \dots$ 
   $\lambda' = \lambda_i + (\mu_i - \lambda_i)/\tau^2; \mu' = \lambda_i + (\mu_i - \lambda_i)/\tau$ 
  si  $g(\lambda') < g(\mu')$  alors
     $\lambda_{i+1} = \lambda_i; \mu_{i+1} = \mu'$ 
  sinon
     $\lambda_{i+1} = \lambda'; \mu_{i+1} = \mu_i$ 
  fin si
fin pour
```

Où $\tau = \frac{\sqrt{5}+1}{2}$ est le *nombre d'or*. Avant d'aller plus loin, il faut réécrire l'algorithme d'une manière plus proche d'un code informatique. En effet on peut s'apercevoir qu'à la deuxième itération, l'une des deux valeurs de λ' ou μ' correspond à celle de l'itération précédente et il serait bête de recalculer ce point une deuxième fois et surtout de ré-évaluer la fonction en ce point !

On considère donc une initialisation partant de 3 points a, b, c tels que¹ :

$$(*) \begin{cases} a < b < c \\ \frac{c-a}{b-a} = \tau \text{ (cas d) ou } \tau^2 \text{ (cas g)} \\ g(a) > g(b) \\ g(c) > g(b) \end{cases}$$

par exemple en prenant² $a = \lambda, c = \mu$ et $b = \lambda'$ ou μ' . Puis on calcule δ tel que :

$$\begin{aligned} \delta &= a + (c - a)/\tau^2 & \text{si } b &= a + (c - a)/\tau & \text{(cas d)} \\ \delta &= a + (c - a)/\tau & \text{si } b &= a + (c - a)/\tau^2 & \text{(cas g).} \end{aligned}$$

¹(cas d) signifie que b est plutôt sur la droite de l'intervalle $[a, c]$ et (cas g) qu'il est plutôt sur la gauche

²nous verrons ultérieurement que la recherche d'un triplet (a, b, c) encadrant le minimum peut se faire de manière à obtenir naturellement les conditions (*)

Question 1 : Montrer que $\delta = c + a - b$ dans les deux cas.

Vous avez du comprendre que b et δ jouent le rôle de λ' et μ' , on obtient ainsi l'algorithme :

```

 $g_a = g(a); g_b = g(b); g_c = g(c)$ 
tant que "non convergence"
     $\delta = c + a - b$ 
     $g_\delta = g(\delta);$ 
    si  $g_\delta < g_b$  alors
        si  $\delta < b$  alors  $c = b; g_c = g_b$  sinon  $a = b; g_a = g_b$  fin si
         $b = \delta; g_b = g_\delta$ 
    sinon
        si  $\delta < b$  alors  $a = \delta; g_a = g_\delta$  sinon  $c = \delta; g_c = g_\delta$  fin si
    fin si
fin tant que
    
```

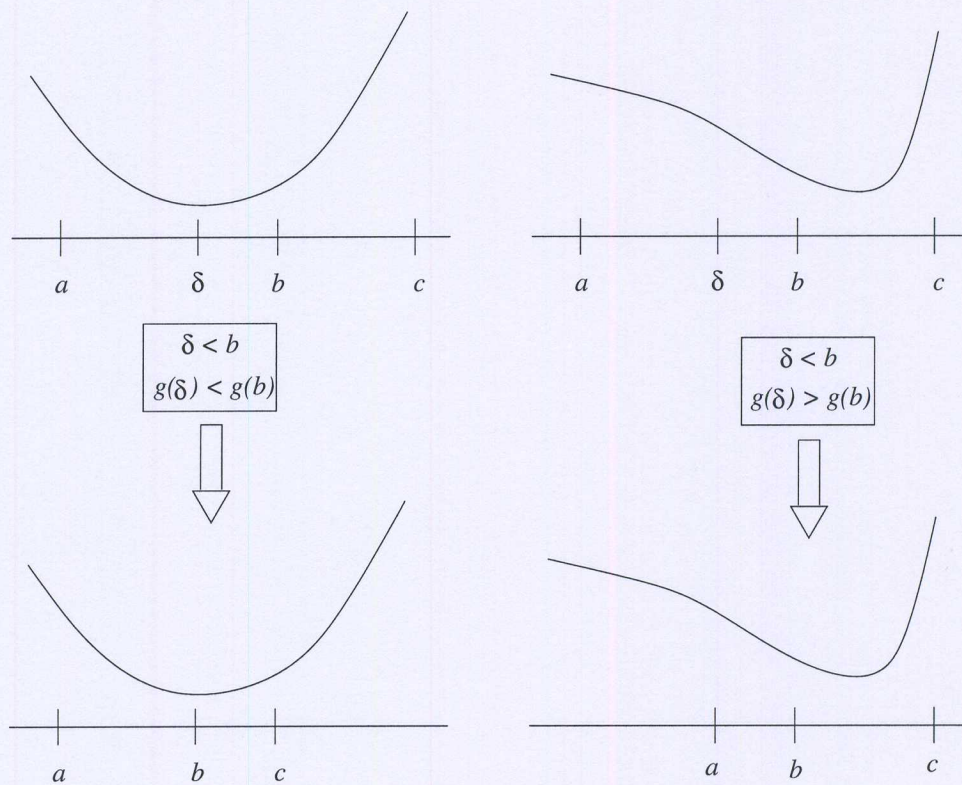


FIG. 1 – Illustrations pour la section dorée...

Test d'arrêt

Il n'est pas facile de donner un test d'arrêt général... Si on a une idée de la précision que l'on cherche, disons tol , il suffit a priori de sortir de l'algorithme lorsque $c - a \leq tol$. Néanmoins il faut

