

Résolution d'un problème d'optimisation différentiable

Marc BOURQUI

Victor CONSTANTIN
Florian SIMOND

Ian SCHORI

January 3, 2013

Énoncé du problème

Trouver (une approximation de) la solution du problème suivant en appliquant le théorème de la plus forte pente:

$$\min_{x \in \mathbb{R}^2} (x_1 - 2)^4 + (x_1 - 2)^2 x_2^2 + (x_2 + 1)^2 \quad (1)$$

Réponses aux questions

- (a) Implémenter la méthode de plus forte pente (Algorithme 11.3) à l'aide du logiciel MATLAB. Déterminer la taille du pas en appliquant la recherche linéaire, Algorithme 11.2 (les deux conditions de Wolfe).

Listing 1: pfp.m

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %
3 % Methodes de descente pour l'optimisation non lineaire %
4 % sans contraintes %
5 % %
6 % BOURQUI Marc %
7 % CONSTANTIN Victor %
8 % SCHORI Ian %
9 % SIMOND Floriant %
10 % %
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12 function [x, fx, iterations] = pfp(f, x0, epsilon, useRL, ...
    showDetails)
13
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 % Interface %
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17
18 % nom de la fonction a minimiser, qui est specifiee dans ...
    le fichier 'f.m'
19 % et qui est declaree sous forme de string
20 fct = f;
21
22 % point initial
23 x = x0;
24
25 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26 % Parametres %
27 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28
29 % pour le critère d'arrêt
30 maxIter = 200 ;
31
32 % initialisation du nombre d'iterations
```

```

33 i=1 ;
34
35 % initialisation de la matrice qui stocke tous les iterés
36 % un iteré = une colonne de cette matrice
37
38 stock(:,i) = x0;
39
40 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41 % Boucle principale %
42 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
43
44
45 % Critere d'arret: x a atteint la precision demandée OU nb ...
    iterations max atteint
46
47 while ( normGradient(fct,stock(:,i)) >= epsilon ) && ( i ...
    <= maxIter )
48     % mise a jour du nombre d'iterations
49     i = i+1;
50
51     prev = stock(:,i-1);
52     if showDetails
53         fprintf('Iteration number %d : x = [%f, %f]\n', i, ...
            prev(1), prev(2));
54     end
55     % calcul et stockage de la valeur du nouveau x
56     stock(:,i) = pfpInnerLoop(fct, prev, useRL);
57
58 end
59
60 % Calcul de la taille de la matrice contenant tous les x
61 taille = size(stock,2);
62
63 % Evaluation de la fonction en chaque point
64 for i=1:taille
65     valeurstock(i)=feval(fct,stock(:,i));
66 end
67
68
69 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
70 % Affichage des résultats %
71 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
72 if showDetails
73     disp('Valeur de la suite des x :') ;
74     stock '
75 end
76 disp('*****')
77 disp(['Nombre d'iterations : ' ...
    num2str(i-1)])
78 disp(['Valeur de la fonction a l'optimum : ' ...
    num2str(feval(fct,stock(:,i)))] ) ;
79 disp('Valeur de l'optimum : ')
80 xOptim = stock(:,i) '
81 disp('*****')

```

```

82
83 if showDetails
84     % passage au module de visualisation de la fonction et ...
85     % des resultats
86     visual3d(fct , stock , valeurstock);
87 end
88 %sprintf('Nombre de fois que la boucle a ete parcourue : ...
89 %d',i)
90
91 x = xOptim;
92 fx = feval(fct , stock(:,i));
93 iterations = i-1;
94 %clear;
95 end

```

Pour **pfp.m**, nous avons réutilisé la structure du corrigé de la série 3. Nous l'avons adapté pour y résoudre l'algorithme de plus forte pente, selon l'algorithme 11.3. La fonction, son gradient et sa hessienne sont placés dans un fichier que nous avons nommé **f.m**. De plus, nous avons ajouté un booléan **useRL** qui permet de sélectionner la méthode de détermination du pas (recherche linéaire ou le pas calculé en (b)).

Listing 2: pfpInnerLoop.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                                                                 %
3  % Calcul d'un itéré et du pas soit en utilisant %
4  % la recherche linéaire soit la formule de Cauchy %
5  %                                                                 %
6  % BOURQUI Marc %
7  % CONSTANTIN Victor %
8  % SCHORI Ian %
9  % SIMOND Floriant %
10 %                                                                 %
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12
13 function x = pfpInnerLoop(f , x0 , useRL)
14
15     alpha = 1;
16     x = x0;
17
18     [fx , gfx] = feval(f , x);
19     d = -gfx;
20
21     % Calcul du pas
22     if useRL
23         % Avec la recherche linéaire
24         beta1 = 0.5;
25         beta2 = 0.75;

```

```

26         lambda = 2;
27         alpha = rechercheLineaire(f, fx, gfx, x, alpha, ...
                                   beta1, beta2, lambda);
28     else
29         %Soit on peut utiliser la fonction dans b) pour ...
           calculer le pas
30         alpha = taillepasCauchy(f,x);
31     end
32     x = x + alpha * d;
33 end

```

Effectue une itération de l'algorithme en utilisant la méthode de calcul du pas spécifiée. Le choix est effectué à l'aide du booléen useRL qui permet de choisir entre la recherche linéaire et la méthode indiquée au point (b).

Listing 3: rechercheLineaire.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Implémente la recherche linéaire %
3  %                                     %
4  % BOURQUI Marc                       %
5  % CONSTANTIN Victor                 %
6  % SCHORI Ian                       %
7  % SIMOND Floriant                   %
8  %                                     %
9  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10
11 function alpha = rechercheLineaire(f, fx, gfx, x, alpha0, ...
    beta1, beta2, lambda)
12     alpha = alpha0;
13     alphas = 0;
14     alphas = inf;
15
16     [fxad, fgxad] = feval(f, x + alpha * -gfx);
17
18     while (fxad > fx + alpha * beta1 * gfx' * -gfx) || ...
           (fgxad' * -gfx < beta2 * gfx' * -gfx)
19         if fxad > fx + alpha * beta1 * gfx' * -gfx
20             alphas = alphas;
21             alpha = (alphas + alphas)/2;
22         elseif fgxad' * -gfx < beta2 * gfx' * -gfx
23             alphas = alphas;
24             if alphas < inf
25                 alpha = (alphas + alphas)/2;
26             else
27                 alpha = lambda * alpha;
28             end
29         end
30
31     [fxad, fgxad] = feval(f, x + alpha * -gfx);

```

```

32     end
33 end

```

Implémente la recherche linéaire d'après l'algorithme 11.2. f_x est la fonction évaluée en x , et g_x est son gradient en x . Nous avons choisi de les passer en paramètres pour ne pas devoir les recalculer. Mais pour plus de modularité, on peut déterminer f_x et g_x en ajoutant $[f_x, g_x]$... = `feval`(f, x); avant la boucle `while`.

- (b) Implémenter une fonction qui donne la taille du pas suivant:

$$\alpha_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_k)^T \nabla^2 f(x_k) \nabla f(x_k)} \quad (2)$$

Quelle est la nature de ce pas? D'où cette formule vient-elle?

Listing 4: `taillepasCauchy.m`

```

1 function pas = taillepasCauchy(f, x)
2     [~, fgx, fghx] = feval(f, x);
3     top = fgx' * fgx;
4     bottom = fgx' * fghx * fgx;
5
6     pas = top / bottom;
7
8 end

```

f n'est pas une fonction linéaire, on peut l'approximer par un modèle quadratique. On va chercher à minimiser ce modèle dans la direction de la plus forte descente. Le point qui minimise ce modèle est le point de Cauchy qui se détermine de la manière suivante :

$$x_C = x_k - \alpha_C \nabla f(x_k) \quad (3)$$

où

$$\alpha_C = \underset{\alpha \in \mathbb{R}_0^+}{\operatorname{argmin}} m_{x_k}(x_k - \alpha \nabla f(x_k)) \quad (4)$$

Sachant f convexe, α_c peut être calculé par (2).

- (c) Comparer le comportement de l'algorithme en utilisant les pas (a) et (b).

k	x_0		$x_{optim,pfp}$		$f(x_{optim,pfp})$	n_{pfp}	$x_{optim,qN}$		$f(x_{optim,qN})$	n_{qN}
1	0	0	2	-1	1.7309e-14	8	2.0004	-1.0001	1.3149e-07	10
2	1	0	2	-1	5.6969e-14	5	1.9999	-1.0001	1.36e-08	7
3	1	1	2	-1	1.5584e-16	7	1.9999	-1.0003	7.2551e-08	8
4	-1	1	2	-1	1.2354e-13	9	2.0002	-0.99974	1.2364e-07	18
5	-1	-1	2	-1	1.7055e-14	9	2.0002	-0.99977	9.1992e-08	18
6	2	-1	2	-1	0	0	2	-1	0	1
7	2	2	2	-1	0	1	2	-1	0	1
8	-2	2	2	-1	6.4214e-17	11	2.0001	-1.0001	1.016e-08	16
9	-2	-2	2	-1	9.5404e-13	10	2.0002	-1.0001	3.4725e-08	16
10	3	-2	2	-1	1.0111e-11	20	1.9998	-1.0002	6.2453e-08	8

- (d) Comparer la methode de plus forte pente et la methode quasi-Newton (qui est déjà implementée – Série 3).

k	x_0		$x_{optim,pfp}$		$f(x_{optim,pfp})$	n_{pfp}	$x_{optim,qN}$		$f(x_{optim,qN})$	n_{qN}
1	0	0	2	-1	1.7309e-14	8	2	-1	2.1765e-11	11
2	1	0	2	-1	5.6969e-14	5	2	-1	2.427e-12	10
3	1	1	2	-1	1.5584e-16	7	2	-1	1.5015e-11	9
4	-1	1	2	-1	1.2354e-13	9	2	-1	1.2389e-11	12
5	-1	-1	2	-1	1.7055e-14	9	2	-1	1.1427e-11	12
6	2	-1	2	-1	0	0	2	-1	0	0
7	2	2	2	-1	0	1	2	-1	0	1
8	-2	2	2	-1	6.4214e-17	11	2	-1	1.7744e-11	12
9	-2	-2	2	-1	9.5404e-13	10	2	-1	8.7017e-12	12
10	3	-2	2	-1	1.0111e-11	20	2	-1	1.4096e-11	12

D'une manière générale, la méthode de la plus forte pente est plus efficace que la méthode de quasi-Newton lorsque la distance entre le point de départ et la solution est petite.