Résolution d'un problème d'optimisation différentiable

Marc Bourqui Victor Constantin Ian Schori
Floriant Simond

January 3, 2013

Énoncé du problème

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

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

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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                               %
% Methodes de descente pour l'optimisation non lineaire         %
% sans contraintes                                              %
%                                                               %
% BOURQUI Marc                                                  %
% CONSTANTIN Victor                                             %
% SCHORI Ian                                                    %
% SIMOND Floriant                                               %
%                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function x = pfp(f, x0, epsilon, useRL)

%%%%%%%%%%%%%%%%%%
% Interface      %
%%%%%%%%%%%%%%%%%%

% nom de la fonction a minimiser, qui est specifiee dans le fichier 'f.m'
% et qui est declaree sous forme de string
fct = f;

% point initial
x = x0;

%%%%%%%%%%%%%%%%%%%%
% Parametres       %
%%%%%%%%%%%%%%%%%%%%

% pour le critère d'arrêt
maxIter = 200    ;

% initialisation du nombre d'iterations
i=1 ;
```

```
% initialisation de la matrice qui stocke tous les iterés
% un iteré = une colonne de cette matrice

stock(:,i) = x0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Boucle principale            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Critere d'arret: x a ateint la precision demandée OU nb ...
    iterations max ateint

while ( normGradient(fct,stock(:,i)) >= epsilon ) && ( i < ...
    maxIter )
    % mise a jour du nombre d'iterations
    i = i+1;

    prev = stock(:,i-1);
    fprintf('Iteration number %d : x = [%f, %f]\n', i, ...
        prev(1), prev(2));
    % calcul et stockage de la valeur du nouveau x
    stock(:,i) = pfpInnerLoop(fct, prev, useRL);

end

% Calcul de la taille de la matrice contenant tous les x
taille = size(stock,2);

% Evaluation de la fonction en chaque point
for i=1:taille
    valeurstock(i)=feval(fct,stock(:,i));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Affichage des résultats  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Valeur de la suite des x :') ;
stock'

disp('****************************************')
disp(['Nombre d''iterations :                     ' ...
    num2str(i-1)])
disp(['Valeur de la fonction a l''optimum : ' ...
    num2str(feval(fct,stock(:,i)))] ) ;
disp('Valeur de l''optimum : ')
xOptim = stock(:,i)'
disp('****************************************')

% passage au module de visualisation de la fonction et des ...
    resultats
```

```
visual3d(fct, stock, valeurstock);

sprintf('Nombre de fois que la boucle a ete parcourue : ...
    %d',i)

clear;
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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                          %
% Calcul d'un itéré et du pas soit en utilisant            %
% la recherche linéaire soit la formule de Cauchy          %
%                                                          %
% BOURQUI Marc                                             %
% CONSTANTIN Victor                                        %
% SCHORI Ian                                               %
% SIMOND Floriant                                          %
%                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x = pfpInnerLoop(f, x0, useRL)

    alpha = 1;
    x = x0;

    [fx, gfx] = feval(f, x);
    d = -gfx;

    % Calcul du pas
    if useRL
        % Avec la recherche linéaire
        beta1 = 0.5;
        beta2 = 0.75;
        lambda = 2;
        alpha = rl(f, fx, gfx, x, alpha, beta1, beta2, ...
            lambda);
    else
        %Soit on peut utiliser la fonction dans b) pour ...
            calculer le pas
        alpha = tp(f,x);
    end
```

```
    x = x + alpha * d;
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: rl.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Implémente la recherche linéaire %
%                                  %
% BOURQUI Marc                     %
% CONSTANTIN Victor                %
% SCHORI Ian                       %
% SIMOND Floriant                  %
%                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function alpha = rl(f, fx, gfx, x, alpha0, beta1, beta2, ...
    lambda)
    alpha = alpha0;
    alphal = 0;
    alphar = inf;

    [fxad, fgxad] = feval(f, x + alpha * -gfx);

    while (fxad > fx + alpha * beta1 * gfx' * -gfx) || ...
        (fgxad' * -gfx < beta2 * gfx' * -gfx)
        if fxad > fx + alpha * beta1 * gfx' * -gfx
            alphar = alpha;
            alpha = (alphal + alphar)/2;
        elseif fgxad' * -gfx < beta2 * gfx' * -gfx
            alphal = alpha;
            if alphar < inf
                alpha = (alphal + alphar)/2;
            else
                alpha = lambda * alpha;
            end
        end

        [fxad, fgxad] = feval(f, x + alpha * -gfx);
    end
end
```

Implémente la recherche linéaire d'après l'algorithme 11.2. fx est la fonction évaluée en x, et gfx 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 fx et gfx en ajoutant `[fx, gfx] ...`

= feval(f, x); avant la boucle while.

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

$$\alpha_k = \frac{\bigtriangledown f(x_k)^T \bigtriangledown f(x_k)}{\bigtriangledown f(x_k)^T \bigtriangledown^2 f(x_k) \bigtriangledown f(x_k)} \tag{2}$$

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

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

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