© *J.-B.A.K. <jean-baptiste.apoung@math.u-psud.fr>*

TP 1 : Prise en main de quelques méthodes numériques élémentaires en calcul des variations : cas d'un problème d'obstacle.

On souhaite résoudre numérique un problème d'optimisation donné sous la forme :

$$\text{Minimiser } J(\mathbf{u}) \text{ sur } K = \{\mathbf{v} \in \mathbb{R}^N : \varphi_1(\mathbf{v}) \leq 0, \dots, \varphi_d(\mathbf{v}) \leq 0\}. \tag{1}$$

La fiche est formée de deux parties :

- parie 1 : On présente un problème concret dont le problème discret associé rentre dans ce formalisme.
- parie 2 : On présente deux méthodes de résolution assez classiques de part leur simplicité de mise en oeuvre :
  - Méthode de gradient projeté à pas fixes (et donc optimal).
  - Méthode de gradient projeté à pas variables (projeté). Dans cette dernière un choix exprès est fait de résoudre finement le problème de recherche du pas optimal afin d'explorer et implémenter quelques méthodes d'optimisation sans contraintes (ici en dimension 1) : méthode de la section d'orée et méthode de Newton.

Note 1 (Méthodes encore plus efficaces pour le problème considéré)**.**

Pour ce type de problème on rappelle néanmoins qu'il existe des algorithmes encore plus performants que l'on peut s'amuser à explorer. Sans être exhaustif citons : la méthode de **pénalisation**, la méthode du **Lagrangien augmenté**, la méthode d'**Uzawa**... On peut aussi dans l'extension de cette fiche, envisager une méthode du gradient conjugué non-linéaire avec un préconditionneur incorporant la projection.

Partie - 1 *Le problème et sa discrétisation par différences finies*

Soit f et g deux fonctions continues données sur $[0,1]$. On souhaite résoudre le problème d'obstacle suivant : trouver $u : [0,1] \to \mathbb{R}$ telle que

$$\left.\begin{array}{rcl} -u''(x) & \geq & f(x), \\ u(x) & \geq & g(x), \\ (-u''(x) - f(x))(u(x) - g(x)) & = & 0, \end{array}\right\} \quad \text{sur }]0,1[, \text{et } u(0) = u(1) = 0. \tag{2}$$

La première équation traduit une concavité maximale de la fonction u. La deuxième équation représente l'obstacle : on veut que la solution u soit au dessus de g. La troisième équation traduit le fait que l'on a au moins égalité dans une des deux équations précédentes : soit on résout $-u''(x) = f(x)$, soit $u(x) = g(x)$, et u est sur l'obstacle.

On discrétise le problème par différences finies. On introduit une subdivision uniforme $x_i = ih$ de $[0,1]$, où $h = \frac{1}{N+1}$ désigne le pas en espace du maillage et où $i \in \{0, \dots, N+1\}$. On cherche alors à résoudre le problème suivant :

$$\left.\begin{array}{rcl} \dfrac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} & \geq & f(x_i), \\ u_i & \geq & g(x_i), \\ \left(\dfrac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} - f(x_i)\right)(u_i - g(x_i)) & = & 0, \end{array}\right\} \quad \text{pour } i = 1 \dots N, \text{ et } u_0 = u_{N-1} = 0. \tag{3}$$

On note J_N la fonctionnelle définie sur $\mathbb{R}^N$ par

$$J_N(\mathbf{u}) = \frac{1}{2}(A_N\mathbf{u}, \mathbf{u}) - (\mathbf{f}_N, \mathbf{u}).$$

où A_N et $\mathbf{f}_N$ sont donnés par :

$$A_N = \begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix} \quad \text{et } \mathbf{f}_N = h^2 \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{pmatrix}.$$

Q-1 : Montrer l'équivalence suivante :

$$\left(\mathbf{u}_N = \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} \text{ est solution de (3)} \right) \Leftrightarrow \left(\begin{array}{l} \mathbf{u}_N \text{ minimise } J_N(\mathbf{u}) \\ \text{sur } K_N = \{\mathbf{v} = (v_i)_{i=1\ldots N} : v_i \geq g_i \ \forall\, i\}. \end{array} \right) \tag{4}$$

Q-2 : Préciser alors les quantités : $J, K, \varphi_i, i = 1, \ldots d$, et d du problème (1) dans ce cas.

Partie - 2 *Deux exemples classiques de méthode de résolution*

Nous explorons ici deux méthodes de résolution du problème décrit précédemment. Les méthodes présentées dans cette partie voient leur importance simplement à travers la facilité de leur mise en oeuvre.

Partie - 2- 1 *Méthode de gradient projeté*

On rappelle l'algorithme du gradient projeté à pas fixe pour minimiser une fonctionnelle $J : \mathbb{R}^N \to \mathbb{R}$ sur un ensemble K, pour un point de départ $\mathbf{u}^0$, un pas ρ et un test d'arrêt ε préalablement définis :

Méthode du gradient projeté à pas fixe
Initialiser le résidu r^0 à 1 et le compteur k à 0.
Tant que le résidu est plus grand que ε et que le compteur n'est pas trop grand :
- calculer la descente $\mathbf{w}^k = -\nabla J(\mathbf{u}^k)$,
- poser $\mathbf{u}^{k+1} = P_K(\mathbf{u}^k + \rho \mathbf{w}^k)$, où P_K désigne la projection sur K,
- calculer le résidu : $r^{k+1} = ||\mathbf{u}^{k+1} - \mathbf{u}^k||$,
- incrémenter le compteur.

Q-3 : Montrer que la projection sur K_N est donnée par :

$$P_{K_N}(\mathbf{u}) = (\max(u_i, g_i))_{i=1\ldots N}.$$

Ecrire un programme **`projK.m`** qui prend en argument un point $\mathbf{u}$ et $\mathbf{g}_N = (g(x_i))_{i=1\ldots N}$, et qui renvoie $P_{K_N}(\mathbf{u})$.

Q-4 : **Mise en oeuvre.** Implémenter cet algorithme à travers une fonction de prototype :

Code Listing 1: Fichier gradient_projete.m

```
function [u,iter] = gradient_projete(J, DJ, gN, u0, rho, epsilon, iterMax, store)
% ENTREES
%  J       : fonctionnelle à minimiser.
%  DJ      : le gradient de la fonctionnelle à minimiser.
%  gN      : vecteur identifiant le convex K i.e  K={v : v[i] >= gN[i],i=1...d}
%  u0      : valeur initiale.
%  rho     : pas fixe.
%  epsilon : test d'arrêt.
%  iterMax : nombre maximal d'itérations autorisées.
%  store   : paramètre controllant le type de stockage dans u. Il prend les valeurs 0 ou 1.
% SORTIES
%   u      : dernier terme de la suite des itérés uk si store = 0 ou tous les termes si store = 1.
%   iter   : nombre d'itérations effectuées.
```

Les questions qui suivent permettront de valider l'implémentation et de comprendre certaines propriétés de la méthode. On créera un script **`scriptTP1_fixe.m`** pour répondre à ces questions. On prendra $f(x) = 1, g(x) = \max(1.5 - 20(x-0.6)^2, 0)$.

Q-5 : **Validations 1/3 : cas $N = 2$.**

Q-5-1 : Tracer sur une même figure les courbes de niveaux de J_2 ainsi que le champ de vecteurs ∇J_2 sur le pavé $[-10, 10] \times [-10, 10]$. On utilisera les fonctions **`contour`** et **`quiver`**.

Q-5-2 : Calculer les itérations $\mathbf{u}^k = (u_1^k, u_2^k)$ données par l'algorithme de gradient projeté à pas fixe, et tracer sur la même figure que précédemment la ligne qui relie les $\mathbf{u}^k$. On prendra $\mathbf{u}^0 = (8, 4)$, $\rho = 0.1$ et $\varepsilon = 10^{-5}$.

Q-6 : **Validations 2/3 : cas N quelconque.**

Q-6-1 : Pour $N = 2, 5, 20, 50, 100$. Afficher à l'aide de la fonction **`fprintf`** le nombre d'itérations ainsi que le temps de calcul pour chaque N. Tracer sur une même figure les solutions approchées $\mathbf{u}_N$, ainsi que le graphe de la fonction g. On prendra $\rho = 0.1$, $\varepsilon = 10^{-5}$.

Q-6-2 : Reprendre l'expérience précédente pour $\rho = 0.5$, puis $\rho = 1$. Que constate-t-on ? Peut-on choisir le pas ρ arbitrairement ?

Q-7 : **Validations 3/3.**

Q-7-1 : Reprendre les questions **Q**-5 **Q**-6-1 avec ρ optimal c'est-à-dire

$$\rho = \frac{2}{\lambda_1 + \lambda_N}$$

où λ_1 et λ_N sont respectivement la plus petite et la plus grande valeur propre de A_N.

Q-7-2 : Reprendre la question précédente pour $f(x) = \pi^2 \sin(\pi x)$.

Partie - 2- 2 *Méthode de gradient (projeté) à pas variable*

L'inconvénient dans la méthode du gradient projeté est cette nécessité pour obtenir une convergence en un temps *raisonnable*, de recourir à un pas fixe optimal, nécessitant ainsi des informations à priori sur le spectre de la matrice A. Un moyen d'éviter cela tout en gardant le même ordre de convergence (observation faite dans le cadre sans projection) est de faire varier le pas de descente dans la méthode du gradient.

Méthode du gradient (projeté) à pas variable
Initialiser le résidu r^0 à 1 et le compteur k à 0.
Tant que le résidu est plus grand que ε et que le compteur n'est pas trop grand :
- calculer la descente $\mathbf{w}^k = -\nabla J(\mathbf{u}^k)$,
- calculer $\rho^k \geq 0$ qui minimise $\rho \mapsto J(\mathbf{u}^k + \rho \mathbf{w}^k)$,
- poser $\mathbf{u}^{k+1} = P_K(\mathbf{u}^k + \rho^k \mathbf{w}^k)$, où P_K désigne la projection sur K,
- calculer le résidu $r^{k+1} = ||\mathbf{u}^{k+1} - \mathbf{u}^k||$,
- incrémenter le compteur.

Note 2 (Autre appellation : **méthode de descente**).

- On reconnaît sous cette formulation la variante projetée d'une méthode couramment identifiée en optimisation sous le nom **méthode de descente**.
- On rappelle que dans ce cas, il n'est pas nécessaire de résoudre de manière **exacte** le problème de la détermination de ρ^k. Il suffit en effet, pour assurer la convergence de la méthode de descente, de choisir ρ^k de sorte à satisfaire une condition de décroissance suffisante couplée à une condition de courbure permettant d'éliminer les très petites valeurs de ρ^k (se renseigner sur la règle **de Wolfe**).
- Nous allons néanmoins essayer de déterminer finement ρ^k, afin de tester quelques méthodes de minimisation sans contrainte en une dimension d'espace.

Q-8 : Mise en oeuvre.

Q-8-1 : Minimisation mono-dimensionnelle associée.

Pour déterminer le pas optimal ρ_k fournir trois approches comme décrites dans le Listing 4 :

Code Listing 2: Détermination du pas optimal

```
function [rho] = optimal_alpha_analytique(J, DJ, uk, wk, rho0)
% Par un calcul explicite tenant compte du caratère quadratique de J
%_____________________________________________________________________
 function [rho] = optimal_alpha_newton(J, DJ, uk, wk, rho0)
% Par la méthode de Newton
%_____________________________________________________________________
 function [rho] = optimal_alpha_section_doree(J, DJ, uk, wk, rho0)
% Par la méthode de la section dorée vue en cours (voir Annexe)
%_____________________________________________________________________
% function [rho] = optimal_alpha_Wolfe(J, DJ, uk, wk, rho0) <-- si l'on veut
%_____________________________________________________________________
% Chacune de ces fonctions prend en arguments:
% J la fonctionnnelle à minimiser, DJ son gradient, uk et wk les solution
% et direction de descente courrantes et enfin rho0 le pas de départ.
% Elles retournent chacune rho: le pas optimal comme décrit dans l'algorithme
```

Q-8-2 : Fonction principale. Fournir une fonction

Code Listing 3: Fichier gradient_optimal.m

```
function [u,iter,rhoL] = gradient_optimal(J, DJ, u0, rho, epsilon, iterMax, store, optim1D)
 % ENTREES
 %  J       : fonctionnelle à minimiser.
 %  DJ      : le gradient de la fonctionnelle à minimiser.
 %  u0      : valeur initiale.
 %  rho     : valeur initiale du pas (n'est pas forcément utilisé).
 %  epsilon : test d'arrêt.
 %  iterMax : nombre maximal d'itérations autorisées.
 %  store   : paramètre controllant le type de stockage dans u. Il prend les valeurs 0 ou 1.
 %  optim1D : entier (à votre convenance) identifiant le type de méthode pour déterminer le pas optimal.
 % SORTIES
 %   u      : dernier terme de la suite des itérés uk si store = 0 ou tous les termes si store = 1.
 %   iter   : nombre d'itérations effectuées.
 %   rhoL   : liste des pas optimaux générés par les itérations.
```

On créera un script **`scriptTP1_optimal.m`** pour la validation.

Q-9 : **Validations : cas $N = 2$.**
Reprendre les expériences effectuées dans la méthode du gradient à pas fixe.

Q-10 : **Validations : cas N quelconque.**

Q-10-1 : Reprendre les questions **Q-5** et **Q-6-1**.

Q-10-2 : Modifier la fonction **`gradient_optimal.m`** de sorte qu'elle retourne aussi les directions de descente w^k que l'on rangera dans les colonnes d'une matrice W.

Q-10-3 : Pour le pas optimal obtenu de manière analytique, et pour $N = 30$,
a) Comparer les pas obtenus avec la valeur optimale du pas de la question **Q-7**.
b) Calculer $W' * W$ et commenter le résultat (W' est la transposée de W).

Note 3 (Pour ceux qui arrivent ici avant la fin de la séance)**.**

Tournez la page pour un exercice supplémentaire.

Partie - 2- 3 *Méthode de pénalisation (Seulement pour ceux qui auront été très rapides dans les exercices précédents)*

La méthode de pénalisation consiste à remplacer le problème de minimisation sous contrainte (4) en une suite de problèmes de minimisation sans contrainte qui converge vers (4).

Soit $\eta > 0$ donné, et J_N^η défini par :

$$J_N^\eta(\mathbf{u}) = J_N(\mathbf{u} + \frac{1}{\eta}\sum_{i=1}^{N}(\max(g_i - u_i, 0))^2.$$

Le minimiseur $\mathbf{u}_N^\eta$ de J_N^η sur $\mathbb{R}^N$ converge vers $\mathbf{u}_N$ quand η tend vers 0.
On admet que le gradient de J_N^η est donné par :

$$\nabla J_N^\eta(\mathbf{u} = A_N\mathbf{u} - \mathbf{f}_N - \left(\frac{2}{\eta}\text{signe}(g_i - u_i)\max(g_i - u_i, 0)\right)_{i=1,\ldots,N}$$

où la fonction signe est définie par : $\text{signe}(x) = \begin{cases} -1 & \text{si} \quad x < 0 \\ 0 & \text{si} \quad x = 0 \\ 1 & \text{si} \quad x > 0 \end{cases}$

Q-11 : **Mise en oeuvre.** En adaptant la fonction **`gradient_fixe.m`** voir annexe, implémenter cet algorithme à travers une fonction une fonction **`penalisation.m`**

Code Listing 4: Fichier penalisation.m

```
function [u,iter] = penalisation(J, DJ, gN, eta, u0, rho, epsilon, iterMax, store)
% ENTREES
%  J       : fonctionnelle à minimiser.
%  DJ      : le gradient de la fonctionnelle à minimiser.
%  gN      : vecteur identifiant le convex K i.e  K={v : v[i] >= gN[i],i=1...d}
%  eta     : parametre de penalisation destiné à tender vers 0
%  u0      : valeur initiale.
%  rho     : pas fixe.
%  epsilon : test d'arrêt.
%  iterMax : nombre maximal d'itérations autorisées.
%  store   : paramètre controllant le type de stockage dans u. Il prend les valeurs 0 ou 1.
% SORTIES
%   u      : dernier terme de la suite des itérés uk si store = 0 ou tous les termes si store = 1.
%   iter   : nombre d'itérations effectuées.
```

Q-12 : **Validations.** Tester cette fonction pour les valeurs numériques de la question **Q-6**, en prenant : $N = 50, \rho = 0.1$ et $\eta = 10^5, 10^4, 10^3, 10^2, 10, 1, 0.1$. Afficher à l'aide de la fonction **`fprintf`** le nombre d'itérations ainsi que le temps de calcul pour chaque η. Tracer sur une même figure les iso-solutions approchées $\mathbf{u}_N^\eta$, ainsi que le graphe de la fonction g et le graphe de la solution de $-u''(x) = 1$. Que constatez-vous pour η grand ? Pour η petit ?

Note 4 (Indications).

On rappelle que si l'on dispose des deux fonctions

```
function [v] = penalite(u,gN,etha)
```

```
function [v] = grad_penalite(u,gN,eta)
```

dont le premier implémente le terme de pénalisation et le second son gradient, alors le listing suivant implémente la méthode du gradient pour le problème pénalisé

```
Jf = @(U) (J(U) + penalite(U,gN,eta)) ;
DJf = @(U) (DJ(U) + grad_penalite(U,gN,eta)) ;
[u, iter] = gradient_fixe(Jf,DJf,u0,rho,epsilon,iterMax,store);
```

Il faut remarquer que dans le cas du problème considéré, la pénalisation apporte un terme quadratique !

Utilitaires pour l'optimisation sans contrainte en dimension 1.

Méthode de la section dorée

Cette méthode est valable uniquement pour une fonction :
- *à valeurs réelles,*
- *dont on connaît un intervalle $[a, b]$ sur lequel elle admet un unique minimum local.*

Le principe de la méthode est un peu semblable à celui de la dichotomie, mais à chaque étape on calcule la valeur de la fonction en deux points de l'intervalle $[a, b]$ de départ, définis par :

$$a' = a + \frac{b-a}{\tau^2} \quad \text{et} \quad b' = a + \frac{b-a}{\tau} \quad \text{avec} \quad \tau = \frac{1+\sqrt{5}}{2}.$$

Algorithme de la section dorée pour minimiser une fonction f
Initialiser le compteur k à 0, l'erreur `err` à $b - a$.
Tant que `err` $> \varepsilon$ (tolérance choisie) :
- *calculer a' et b',*
- *évaluer $f(a')$ et $f(b')$,*
- *si $f(a') > f(b')$: poser $a = a'$*
  si $f(a') < f(b')$: poser $b = b'$
  si $f(a') = f(b')$: poser $a = a'$ et $b = b'$,
- *calculer la nouvelle erreur `err` $= b - a$,*
- *incrémenter le compteur.*

Code Listing 5: Fichier methode_section_doree_recursive.m

```
 function [x, iter] = methode_section_doree_recursive(a, b, f, epsilon)
  tau = (1+sqrt(5))/2;
  k =  0;
  err =  b - a;
  delta = 1e-36;
  bp = a + (b-a)/tau;
  ap = a+ b - bp;
  fap = f(ap);
  fbp = f(bp);
  fa = f(a);
  fb = f(b);
  iter = 0;
  [x, iter] = util_methode_section_doree_recursive(a,ap,bp,b, fa,fap,fbp, fb,f,epsilon,iter);
end
% ---------------------- utilitaire ------------------------

function [x, k] = util_methode_section_doree_recursive(a, ap, bp, b,fa,fap,fbp,fb, f,epsilon,ki)
   if( (b-a) < (epsilon*(abs(ap) + abs(bp) )))
       x = (b+a)/2;
       k = ki;
   else
   if( fap  >  fbp)
    ki = ki+1;
    [x,k] = util_methode_section_doree_recursive(ap,bp,ap + b -bp,b, fap,fbp,f(ap+b-bp), fb, f,epsilon,ki);
   else
    ki = ki + 1;
    [x,k] = util_methode_section_doree_recursive(a,a+bp-ap,ap,bp, fa,f(a+bp-ap),fap, fbp,f,epsilon,ki);
   end
  end
end
```

Méthode de Newton en dimension 1

Cette méthode est valable pour des fonctions dont on connaît une approximation des zéros.

L'algorithme de Newton-Raphson permet de trouver un point en lequel une fonction f s'annule, connaissant une approximation x^0 de ce point :

Algorithme de Newton-Raphson en dimension 1
Initialiser le compteur `it` *à* 0, *l'erreur* `err` *à* 1.
Tant que `it` < `itmax` *et que* `err` $> \varepsilon$ *(tolérance choisie) :*

- *calculer le prochain candidat pour le zéro de* f *:* $x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$,
- *calculer l'erreur* `err` $= |f(x^k)|$,
- *incrémenter le compteur.*

Code Listing 6: Fichier methode_newton.m

```
function [x, iter] = methode_newton(f, fp, x0, epsilon, itermax)
% ENTREES:
% f  : fonction donc on cherche une le zero
% fp : la dérivée de f
% x0 : solution initiale
% epsilon : tolérance
% itermax : nombe maximal d'itérations autorisées
% SORTIES:
%   x : solution la valeur approchée du zero de f
%   iter: le numbre d'itérations effectuées
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 iter =  0;
 err  = 1.;
 x = x0;
 fx = f(x);
 while ( (iter < itermax) && (err > epsilon))
   fpx = fp(x);
   delta = fx / fpx;
   x  = x - delta;
   fx = f(x);
   err  = min(abs(delta)/(abs(x)+1.), abs(fx));
   iter = iter + 1;
 end
end
```

On insère ici une implémentation de la méthode du gradient (non projeté) à pas fixe

Code Listing 7: Fichier gradient_fixe.m

```
function [u,iter] = gradient_fixe(J, DJ, gN, eta, u0, rho, epsilon, iterMax, store)
% ENTREES
%  J       : fonctionnelle à minimiser.
%  DJ      : le gradient de la fonctionnelle à minimiser.
%  u0      : valeur initiale.
%  rho     : pas fixe.
%  epsilon : test d'arrêt.
%  iterMax : nombre maximal d'itérations autorisées.
%  store   : paramètre controllant le type de stockage dans u. Il prend les valeurs 0 ou 1.
% SORTIES
%   u      : dernier terme de la suite des itérés uk si store = 0 ou tous les termes si store = 1.
%   iter   : nombre d'itérations effectuées.
function [u,iter] = gradient_fixe(J, DJ, u0,rho, epsilon, iterMax,store)
 err = 1;
 iter = 0;
 u = u0;
 un = u0;
 while( err > epsilon && iter < iterMax)
   w = rho * DJ(un);
   un = un - w;
   if (store ~= 0)
   u = [u,un];
   else
   u = un;
   end
   err = norm(w);
   iter = iter + 1;
 end
end
```