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

TP 2 : Optimisation sous contrainte, un problème d'obstacle.

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

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

La fiche est formée de trois parties :

— parie 1 : On présente un problème concret dont le problème discret associé rentre dans ce formalise.
— parie 2 : On présente deux méthodes de résolution assez classiques de part leur simplicité de mise en oeuvre.
— parie 3 : Un travail à rendre est proposé avec pour objectif un approfondissement ou une synthèse des acquis.

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.

Comme dans le TP 1, 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 $i \in \{0, \ldots, N+1\}$. On cherche alors à résoudre le problème suivant :

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

On note à nouveau J_N la fonctionnelle définie 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.
```

On adaptera simplement le programme **`gradient_fixe.m`** du TP 1

Q-5 : **Validations 1/2.** Créer un script **`script_TP3.m`** et tester la fonction **`gradient_projete.m`** pour $f(x) = 1, g(x) = \max(1.5 - 20(x - 0.6)^2, 0), N = 2, 5, 20, 50, 100, \varepsilon = 10^{-5}$, et ρ choisi de façon optimale, 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.
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.

Q-6 : **Validations 2/2.** Reprendre la question précédente pour $f(x) = \pi^2 \sin(\pi x)$.

Partie - 2- 2 *Méthode de pénalisation*

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-7 : **Mise en oeuvre.** En adaptant la fonction **`gradient_fixe.m`** du TP1, implémenter cet algorithme à travers une fonction une fonction **`penalisation.m`**

Code Listing 2: 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-8 : **Validations.** Tester cette fonction pour les valeurs numériques de la question **Q-**5, 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 1 (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 !

Partie - 3 *Devoir maison : à rendre*

Note 2 (Recommandations)**.**

Cette partie porte sur la résolution du problème décrit dans la Partie-1, par une méthode d'Uzawa et une version ajustée de la méthode de pénalisation.
Le problème dans sa mise en oeuvre est expressément décrit de manière succincte afin que le **choix des paramètres des fonctions vous incombe**. En effet ceci sera pris en compte dans l'évaluation. Seront aussi pris en compte :

- La structuration des fonctions via une décomposition fonctionnelle que vous jugerez optimale pour alléger la lecture et faciliter la ré-utilisabilité.
- Le choix des arguments d'entrée et de sortie ainsi que leur position et leur nombre, permettant de mener convenablement et de manière pertinente les expériences demandées.
- Un effort de recherche bibliographie, si l'interprétation rigoureuse des résultats obtenus le réclame.

Partie - 3- 1 *Méthode d'Uzawa*

On souhaite ici résoudre le problème (1) par la méthode d'Uzawa. Pour $\lambda = (\lambda_1, \ldots, \lambda_d)$, on note

$$L_\lambda(\mathbf{u}) = J(\mathbf{u}) + \sum_{i=1}^{d} \lambda_i \varphi_i(\mathbf{u}).$$

La méthode d'Uzawa consiste à remplacer le problème (1) par une suite de problèmes de minimisation sans contrainte portant sur les L_λ.
Pour un point de départ $\mathbf{u}^0$, un choix initial de $\lambda^0 = (\lambda_1^0, \ldots, \lambda_d^0)$, une tolérance ε, et un pas ρ donné, cet algorithme s'écrit de la façon suivante :

Méthode d'Uzawa
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 $\mathbf{u}^{k+1}$ le minimiseur de $\mathbf{v} \mapsto L_{\lambda^k}(\mathbf{v})$ (voir Note 4),
- poser $\lambda^{k+1} = (\max(\lambda_i^k + \rho\varphi_i(\mathbf{u}^{k+1}), 0))_{i=1\ldots d}$,
- calculer le résidu $r^{k+1} = ||\mathbf{u}^{k+1} - \mathbf{u}^k||$,
- incrémenter le compteur.

Dans la suite on considère le problème de la section précédente (voir aussi Partie-2-1).

Q-9 : On souhaite utiliser la méthode du gradient à pas fixe pour calculer à chaque étape le minimiseur de $\mathbf{v} \mapsto L_{\lambda^k}(\mathbf{v})$. Écrire une fonction **`gradient_fixe_lambda.m`** qui prend en argument λ, ρ, ε et un point de départ $\mathbf{u}_0$, et qui retourne le minimiseur de $\mathbf{v} \mapsto L_\lambda(\mathbf{v})$.

Q-10 : Créer un script **`script_uzawa.m`** et résoudre le problème de minimisation sous contrainte $\min_{\mathbf{u}\in K} J(\mathbf{u})$ à l'aide de l'algorithme d'Uzawa. On prendra $\mathbf{u}^0$ et λ^0 nuls.

Q-11 : Que se passe-t-il si l'on augmente le pas ρ ?

Q-12 : Peut-on modifier l'algorithme d'Uzawa de sorte à faire varier ρ ? Y' a-t-il intérêt à le faire ?

Partie - 3- 2 *Méthode de pénalisation*

Note 3.

— La fin de l'exercice sur la méthode de pénalisation de la Partie-2 a mis en évidence des comportements différents suivant les valeurs du paramètre de pénalisation.
— Il est donc judicieux de mettre en place un algorithme itératif permettant d'ajuster le paramètre le pénalisation à chaque itération.
— On se propose ici de mettre en oeuvre un tel algorithme et de le comparer avec la méthode d'Uzawa sous la main. Mais pour que cette comparaison soit être possible, l'algorithme dans son implémentation devra retourner à chaque itération une approximation du multiplicateur de Lagrange.

On procède comme dans la méthode d'Uzawa. Pour $\eta \in \mathbb{R}^*$, on note

$$L_\eta(\mathbf{u}) = J(\mathbf{u}) + \frac{1}{\eta}\sum_{i=1}^{d}\Big(\max(\varphi_i(\mathbf{u}),0)\Big)^2.$$

La méthode de pénalisation avec ajustement du paramètre de pénalisation s'écrit alors :

Méthode de pénalisation
Initialiser le résidu r^0 à 1, le compteur k à 0 et η_0 à 1.
Tant que le résidu est plus grand que ε et que le compteur n'est pas trop grand :
— calculer $\mathbf{u}^{k+1}$ le minimiseur (*approché*) de $\mathbf{v} \mapsto L_{\eta_k}(\mathbf{v})$: voir Note 4
— mettre à jour le multiplicateur de Lagrange λ^{k+1} : voir Note 5,
— choisir un nouveau paramètre de pénalisation $\eta_{k+1} < \eta_k$: voir Note 5,
— calculer le résidu $r^{k+1} = ||\mathbf{u}^{k+1} - \mathbf{u}^k||$,
— incrémenter le compteur.

Q-13 **:** Modifier la fonction **`penalisation.m`** de la question **Q-**7, pour mettre en oeuvre cet algorithme.

Q-14 **:** Créer un script **`script_penalisation.m`** et résoudre le problème de la **Partie-**1 (voir **Q-**8).

Q-15 **:** Comparer avec la méthode d'Uzawa pour le problème considéré. On insistera sur la qualité de la solution, la difficulté à obtenir la solution qu'on pourra mesurer par les nombres d'itérations et les temps mis pour la résolution. On commentera les valeurs approchées du multiplicateur de Lagrange obtenu.

Note 4 (Résolution du problème sans contrainte)**.**

La résolution du problème de minimisation sans contrainte peut se faire soit à l'aide des fonctions **`fminsearch, fminunc ...`** de **`Matlab`**, soit en faisant recours à l'une des méthodes vues au précédent TP.

Note 5 (Mise à jour des paramètres : η_{k+1} et $\lambda^{k+1} = (\lambda_1^{k+1}, \ldots, \lambda_d^{k+1})$)**.**

1. **Paramètre de pénalisation** :
   La mise à jour du nouveau paramètre de pénalisation peut se faire de la manière suivante :

$$\eta_{k+1} = \frac{\eta_k}{100}. \tag{5}$$

2. **Multiplicateur de Lagrange dans la méthode pénalisation** :
   La méthode de pénalisation ne dépend pas du multiplicateur de Lagrange. On peut néanmoins estimer le multiplicateur de Lagrange par la formule suivante :

$$\lambda_i^{k+1} = \frac{2}{\eta_k} \max(\varphi_i(\mathbf{u}^k), 0) \quad i = 1, \ldots, d. \tag{6}$$