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

Note 1 (Changement dans le déroulement)**.**

Pour la séance 3, on devra procéder comme suit :
- Achever la fiche TP2
- Puis entamer la présente fiche (TP3) avec pour objectif de l'achever avant la fin de la séance.

TP3 : Optimisation sous contrainte en dimension finie. Méthode de projection

Dans les problèmes de minimisation avec contrainte, il est parfois possible de déterminer l'opération de projection sur l'espace des contraintes. (*On rappelle que ce problème est aussi difficile que le problème de minimisation de départ*).

Mais lorsqu'on dispose de cet opérateur de projection, on peut ajuster les méthodes de descentes, et dériver alors des méthodes efficaces de résolution du problème de départ.

Nous allons considérer dans cette partie la méthode du gradient projeté et conformément au TP1, nous allons distinguer le cas à **pas fixe** du cas à **pas variables**.

Partie - 1 *Le problème modèle et sa discrétisation par éléments finis*

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

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

Dans cette fiche nous allons entre autres :
- tirer partie de la possibilité de définir un opérateur de projection sur l'espace des contraintes.
- Implémenter les algorithmes de descente avec projection.

On reprend le problème de du TP précédent, et pour être complet nous le décrivons de nouveau ici.

Soit f et g deux fonctions continues données sur $[0,1]$. On souhaite résoudre le problème d'obstacle suivant :

$$\begin{cases} \min\limits_{u \in K} J(u) \\ J(u) = \displaystyle\int_0^1 \left(\frac{1}{2}u'(x)^2 - f(x)u(x)\right) dx \\ K = \Big\{u \in H^1(]0,1[) : u(0) = a, u(1) = b,\ u(x) \geq g(x)\ \forall x \in [0,1]\Big\} \end{cases} \tag{2}$$

Ce problème est discrétisé par la méthode des éléments finis comme suit : on se donne un maillage de $[0,1]$ de sommets $0 = x_0 < x_1 < \ldots < x_i < \ldots < x_{N+1} = 1$. On pose $h_i = x_{i+1} - x_i, i = 0, \ldots, N+1$ et $h = \max\limits_{0 \leq i \leq N+1} h_i$, Et on introduit l'espace

$$K_h = \Big\{v \in C^0([0,1]) : v(0) = a, v(1) = b,\ v_{|[x_i,x_{i+1}]}(x) = v(x_i) + (x - x_i)\frac{v(x_{i+1}) - v(x_i)}{h_i}, i = 0, \ldots, N, \\ v(x_i) \geq g(x_i),\ i = 0, \ldots, N+1\Big\}. \tag{3}$$

On approche alors le problème (2) par le problème suivant :

$$\begin{cases} \min\limits_{u_h \in K_h} J(u_h) \\ J(v_h) = \displaystyle\int_0^1 \left(\frac{1}{2}v_h'(x)^2 - f(x)v_h(x)\right) dx \end{cases} \tag{4}$$

Q-1 : [**Explicitation du problème discret**]

Q-1-1 : Expliciter ce problème (4) en utilisant notamment la formule des trapèzes pour calculer les intégrales et montrer qu'il peut se mettre sous la forme suivante

$$\begin{cases} \min\limits_{u_N \in K_N} J_N(u_N) \\ K_N = \{v \in R^N : v_i \geq g(x_i) : \forall\ 1 \leq i \leq N\} \end{cases} \tag{5}$$

où J_N est à expliciter.(*On pourra se servir du TP1* .)

Q-1-2 : Montrer que ce problème admet une unique solution.

Q-2 : [**L'espace discret** K_N]

Q-2-1 : Montrer que K_N est un ensemble convexe ferme non vide.

Q-2-2 : Rappeler la définition de l'opérateur projection P_K sur un convexe fermé non vide K.

Q-2-3 : Montrer alors que P_{K_N} est défini par

$$\forall v \in \mathbb{R}^N, \quad (P_{K_N}(v))_i = \max(g_i, v_i) \quad \forall i = 1, \ldots, N \tag{6}$$

Q-3 : [**Généralisation**]

Montrer de manière générale que si $K = \{v \in \mathbb{R}^N : a_i \leq v_i \leq b_i\}$ on a

$$\forall v \in \mathbb{R}^N, \quad (P_{K_N}(v))_i = \max(a_i, \min(v_i, b_i)) \quad \forall i = 1, \ldots, N \tag{7}$$

Et que cette définition est valable même si l'on a $a_i = -\infty$ *ou* $b_i = +\infty$ *pour certains* $1 \leq i \leq N$

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-4 :

Ecrire un programme **`projK.py`** 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-5 : **Mise en oeuvre.** Implémenter cet algorithme à travers une fonction de prototype :

Code Listing 1 – Fichier gradient_projete_fixe.py

```
def gradient_projete_fixe(J, DJ, gN, u0, rho, epsilon, iterMax, store):
""" ENTREES
  J       : fonctionnelle a minimiser.
  DJ      : le gradient de la fonctionnelle à minimiser.
  gN      : vecteur identifiant le convexe K i.e  K={v : v[i] >= gN[i],i=1...d}
  u0      : valeur initiale.
  rho     : pas fixe.
  epsilon : test d'arret.
  iterMax : nombre maximal d'iterations autorisees.
  store   : parametre controllant le type de stockage dans u. Il prend les valeurs 0 ou 1.
 SORTIES
   u      : dernier terme de la suite des iteres uk si store = 0 ou tous les termes si store = 1.
   iter   : nombre d'iterations effectuees.
"""
```

Les questions qui suivent permettront de valider l'implémentation et de comprendre certaines propriétés de la méthode. On créera un script **`scriptTP3_fixe.py`** pour répondre à ces questions. On prendra $f(x) = 1, g(x) = \max(1.5 - 20(x-0.6)^2, 0)$.

Q-6 : **Validations 1/3 : cas $N = 2$.**

Q-6-1 : Tracer sur une même figure les courbes de niveaux de J_2 ainsi que le champ de vecteurs ∇J_2 sur le pavé $[-10, 10] \times [-10, 10]$. (*voir* ***TP1***).

Q-6-2 : 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-7 : **Validations 2/3 : cas N quelconque.**

Q-7-1 : Pour $N = 2, 5, 20, 50, 100$. Afficher à l'aide de la fonction **`print`** 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-7-2 : Reprendre l'expérience précédente pour $\rho = 0.5$, puis $\rho = 1$. Que constate-t-on ? Peut-on choisir le pas ρ arbitrairement ?

Q-8 : **Validations 3/3.**

Q-8-1 : Reprendre les questions **Q**-6 **Q**-7-1 avec ρ optimal c'est-à-dire

$$\rho = \frac{2}{\lambda_1 + \lambda_N} \tag{8}$$

où λ_1 et λ_N sont respectivement la plus petite et la plus grande valeur propre de A_N (associée au gradient de J_N).

Q-8-2 : 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-9 : Mise en oeuvre.

Q-9-1 : Minimisation mono-dimensionnelle associée.

Pour déterminer le pas optimal ρ_k fournir trois approches comme décrites dans le Listing 2 :

Code Listing 2 – Détermination du pas optimal

```
def optimal_alpha_analytique(J, DJ, uk, wk, rho0):
# Par un calcul explicite tenant compte du ècaratre quadratique de J
#_____________________________________________________________________
def optimal_alpha_newton(J, DJ, uk, wk, rho0):
# Par la émthode de Newton
#_____________________________________________________________________
def optimal_alpha_section_doree(J, DJ, uk, wk, rho0):
# Par la methode de la section doree  (voir Annexe TP1)
#_____________________________________________________________________
# function [rho] = optimal_alpha_Wolfe(J, DJ, uk, wk, rho0) <-- si l'on veut
#_____________________________________________________________________
# Chacune de ces fonctions prend en arguments:
# J la fonctionnnelle a minimiser, DJ son gradient, uk et wk les solution
# et direction de descente courrantes et enfin rho0 le pas de édpart.
# Elles retournent chacune rho: le pas optimal comme decrit dans l'algorithme
```

Q-9-2 : Fonction principale. Fournir une fonction

Code Listing 3 – Fichier gradient_projete_optimal.py

```
def gradient_projete_optimal(J, DJ, gN, u0, rho, epsilon, iterMax, store, typeOptim1D):
 """ ENTREES
    J       : fonctionnelle e minimiser.
    DJ      : le gradient de la fonctionnelle à minimiser.
    gN      : vecteur identifiant le convexe K i.e  K={v : v[i] >= gN[i],i=1...d}
    u0      : valeur initiale.
    rho     : valeur initiale du pas (n'est pas forcement utilise).
    epsilon : test d'arret.
    iterMax : nombre maximal d'iterations autorisees.
    store   : parametre controllant le type de stockage dans u. Il prend les valeurs 0 ou 1.
    typeOptim1D : entier (a votre convenance) identifiant le type de methode de determination du pas ↩
        optimal.
   SORTIES
     u      : dernier terme de la suite des iteres uk si store = 0 ou tous les termes si store = 1.
     iter   : nombre d'iterations effectuees.
     rhoL   : liste des pas optimaux generes par les iterations.
 """
```

On créera un script **`scriptTP3_optimal.py`** pour la validation.

Q-10 : **Validations : cas $N = 2$.**
Reprendre les expériences effectuées dans le cas du pas fixe.

Q-11 : **Validations : cas N quelconque.**

Q-11-1 : Reprendre les questions **Q-6** et **Q-7-1**.

Q-11-2 : Modifier la fonction **`gradient_projete_optimal.py`** de sorte qu'elle retourne aussi les directions de descente w^k que l'on rangera dans les colonnes d'une matrice W.

Q-11-3 : 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-8**.
b) Calculer $W' * W$ et commenter le résultat (W' est la transposée de W).

Partie - 2- 3 *Comparaison avec la méthode de pénalisation*

On se place dans la cas de la méthode du gradient à pas fixe (mais optimal). Le paramètre pas fixe optimal est donné dans le cas quadratique par la formule 8.

Le petit listing 4 ci-dessous calcul ce pas optimal pour une fonctionnelle quadratique J. Il suffit pour cela de lui fournir la jacobienne DJ de J et la longueur n de v dans $J(v)$.

Code Listing 4 – Détermination du pas optimal

```
def alphaFixeOptimal(DJ,n):
""" Attention fonction assez chere en ressources. Ne pas l'utiliser dans un code de production  """
    a = np.eye(n)
    b = DJ(np.zeros(n))
    for i in range(n):
        a[:,i] = DJ(a[:,i]) - b
    lam = np.linalg.eig(a)[0]
    return 2.0/(np.min(lam) + np.max(lam))
```

Dans la suite on prendra $f(x) = 1$, $g(x) = \max(1.5 - 20(x - 0.6)^2, 0)$, $N = 50$ et dans le cas de la méthode de pénalisation, le paramètre de pénalisation prendra les valeurs $\eta = 10^5, 10^4, 10^3, 10^2, 10, 1, 0.1$.

Q-12 : Pour les valeurs du paramètre de pénalisation ci-dessus, représenter sur une même figure, l'obstacle et les solutions obtenues par la méthode de pénalisation et la méthode du gradient projeté, lorsque la méthode du gradient à pas fixe optimal est utilisée. *On comparera la valeur du pas optimal pour chaque méthode.*

Q-13 : Comparer pour ces deux méthodes les temps d'exécution.
On pourra aussi en modifiant le listing 4 comparer les conditionnements de la jacobienne.

Q-14 : Quelles observations faites-vous de cette analyse ?