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

Devoir Maison : soutenance semaine du 23-27 mars

Partie - 1 *Optimisation en dimension 1 : méthode de la section dorée*

On se propose ici d'approfondir la compréhension de la méthode de la section dorée. Pour cela nous allons
- optimiser son algorithme (tel que proposé en TP1),
- lui proposer une implémentation alternative,

Partie - 1- 1 *Optimisation des coûts mémoire et CPU*

On se propose ici de réduire le coût de chaque itération de l'algorithme de la section dorée proposée dans le TP 1. *Le principe de la méthode est un peu semblable à celui de la dichotomie, mais ici, à chaque étape on calcule la valeur de la fonction en deux points de l'intervalle* $[a, b]$ *de départ, définis par :*

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

Afin de faciliter sa mise en oeuvre lors de la séance de TP, son algorithme a été donné sous la forme suivante :

Algorithme de la section dorée pour minimiser une fonction f sur l'intervalle $[a, b]$
Initialiser le compteur k à 0, l'erreur `err` *à $b - a$.*
Tant que `err` $> \varepsilon$ *(tolérance choisie) :*
- *calculer* $a' = a + \frac{b-a}{\tau^2}$ *et* $b' = a + \frac{b-a}{\tau}$ *avec* $\tau = \frac{1+\sqrt{5}}{2}$,
- *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.*

Q-1 : En remarquant que le coût de l'évaluation de la fonction à minimiser peut être élevé, modifier l'algorithme ainsi proposé afin qu'elle ne nécessite qu'une seule évaluation de la fonction f à chaque itération de l'algorithme.

Q-2 : Critiquer le test d'arrêt proposé. *On pourra considérer le cas extrême où* $a = \frac{\varepsilon}{4}, b = \frac{3\varepsilon}{4}$. En consultant la bibliographie, proposer un meilleur test d'arrêt.

Q-3 : Fournir alors pour sa mise en oeuvre une fonction de prototype

Code Listing 1: Section doree

```
% f       est la fonction dont on cherche le minimum
% [a,b]   est un intervalle contenant le minimum
% epsilon est le parametre du le test d'arret
% maxIter est le nombre maximal d'iterations (ou d'appel de la fonction f) autorise
% y       est le point de minimum obtenu
% nbiter  est le nombre d'iterations effectuees
function [y,nbiter] = section_doree(f, a, b, epsilon, maxIter)
```

Partie - 1- 2 *Implémentation récursive*

Nous savons que ce qui rend la récursion chère c'est en particulier l'existence des variables locales. En effet celles-ci sont emmagasinées et l'espace mémoire occupé (par elles) n'est libéré qu'après la dernière instruction de la pile des appels. Un algorithme récursif avec moins de variables locales sera donc *relativement* rapide.

Q-4 : Montrer que l'on peut modifier l'algorithme de la section dorée de sorte qu'elle :

— devienne récursive,

— et ne nécessite aucune variable locale à part ses arguments.

Q-5 : En déduire l'avantage de cet algorithme dans un langage de programmation où les arguments peuvent être passés par adresse (exemple du langage C) ou par référence (exemple du langage C++).

Q-6 : Fournir pour son implémentation, une fonction `Matlab` de prototype

Code Listing 2: Section dorée récursive

```
function [x, iter] = section_doree_recursive(a, b, f, epsilon)
  tau = (1+sqrt(5))/2;
  bp = a + (b-a)/tau; % b'
  ap = a + b - bp;    % a'
  fap = f(ap);        % f(a')
  fbp = f(bp);        % f(b')
  fa = f(a);          %
  fb = f(b);
  iter = 0;
  [x, iter] = detail_section_doree_recursive(a, ap, bp, b, fa, fap, fbp, fb, f, epsilon,iter);
end
```

où `detail_section_doree_recursive` est la fonction récursive à définir.

Q-7 : Comparer le présent algorithme à celui de l'exercice précédent. Considérer le problème de la fiche de TP 1.

Partie - 2 *Optimisation sans contrainte : méthode de gradient conjugué projeté et application*

On rappelle que l'algorithme du gradient conjugué pour la détermination du minimum sans contrainte de

$$\mathcal{J}(x) = \frac{1}{2}(Ax, x) - (b, x)$$

où A est une matrice $n \times n$ symétrique définie positive et b un vecteur de $\mathbb{R}^n$ est donné par :

Note 1 (Algorithme du gradient conjugué).

$$\begin{array}{rcl} \text{Initialisation :} & x_0 = 0, r_0 = b - Ax_0, d_0 = r_0 & \\ \text{Itérations} & : k = 0, 1, \ldots & \\ \alpha_k & = & \|r_k\|^2/(d_k, Ad_k) \\ x_{k+1} & = & x_k + \alpha_k d_k \\ r_{k+1} & = & r_k - \alpha_k A d_k \\ \beta_{k+1} & = & \|r_{k+1}\|^2/\|r_k\|^2 \\ d_{k+1} & = & r_{k+1} + \beta_{k+1} d_k \end{array},$$

le test d'arrêt pouvant être $\|r_k\| \leq \varepsilon\|b\|$.

Partie - 2- 1 *Production de l'algorithme du gradient conjugué projeté*

On s'intéresse dans cet exercice à la recherche du minimum dans un espace affine E de la fonction

$$\mathcal{J}(x) = \frac{1}{2}(Ax, x) - (b, x)$$

où A est une matrice $n \times n$ symétrique positive, et b est un vecteur de $\mathbb{R}^n$. A est supposée définie positive sur l'espace vectorielle associé à E.

On supposera que E est défini par $E = \{x \in \mathbb{R}^n, Bx + c = 0\}$ où $c \in R^m$ et B est une matrice $m \times n$ avec B^T de rang $m < n$.

On désigne par P l'opérateur de projection orthogonale sur l'espace vectoriel associé à E.

Q-8 : Construction de l'opérateur de projection sur l'espace vectoriel associé à E

Q-8-1 : On suppose que les vecteurs lignes de B sont linéairement indépendants.
Montrer que la matrice BB^T est alors régulière.

Q-8-2 : On pose $Q = B^T(BB^T)^{-1}B$.
Montrer que $QQ = Q, Q^T = Q, Q(I - Q) = (I - Q)Q = 0$.
Montrer que Q est une opérateur de projection orthogonale sur $Im(B^T)$.
En déduire que $P = I - Q$.

Q-9 : Construction de l'algorithme

Soit x_0 un point qui vérifie $Bx_0 + c = 0$. On introduit le changement de variable :

$$x = x_0 + Py, \quad \text{et on pose} \qquad \mathcal{G}(y) = \mathcal{J}(x_0 + Py)$$

Q-9-1 : Soit $\bar{y}$ un point de minimum de la fonction $\mathcal{G}$. Montrer que le point correspondant

$$\bar{x} = x_0 + P\bar{y} \quad \text{réalise le minimum de} \quad \mathcal{J} \quad \text{sous la contrainte} \quad Bx + c = 0.$$

Q-9-2 : Montrer que l'algorithme du gradient conjugué appliqué à la recherche du minimum $\mathcal{G}$ est

Initialisation :	$y_0 = 0, r_0 = P(b - Ax_0), d_0 = r_0$	
Itérations	$: k = 0, 1, \ldots$	
α_k	$=$	$\|r_k\|^2/(d_k, (PAP)d_k)$
y_{k+1}	$=$	$y_k + \alpha_k d_k$
r_{k+1}	$=$	$r_k - \alpha_k (PAP)d_k$
β_{k+1}	$=$	$\|r_{k+1}\|^2/\|r_k\|^2$
d_{k+1}	$=$	$r_{k+1} + \beta_{k+1} d_k$

Q-10 : Simplification de l'algorithme

On souhaite réécrire l'algorithme précédent de sorte à faire intervenir directement x

Q-10-1 : Montrer que $Pd_k = d_k$ (*On pourra le faire par récurrence.*)

Q-10-2 : En déduire que l'algorithme peut s'écrire

Initialisation :	x_0 tel que $Bx_0 + c = 0, r_0 = b - Ax_0, z_0 = Pr_0, d_0 = z_0$	
Itérations	$: k = 0, 1, \ldots$	
α_k	$=$	$(z_k, z_k)/(d_k, Ad_k)$
x_{k+1}	$=$	$x_k + \alpha_k d_k$
r_{k+1}	$=$	$r_k - \alpha_k A d_k$
z_{k+1}	$=$	Pr_{k+1}
β_{k+1}	$=$	$\|z_{k+1}\|^2/\|z_k\|^2$
d_{k+1}	$=$	$z_{k+1} + \beta_{k+1} d_k$

Q-11 : Soit m_0 l'ordre de multiplicité de la valeur propre nulle de PAP. Montrer que l'algorithme précédent converge en au plus $n - m_0$ itérations. (*On pourra admettre cette question.*)

Q-12 : Implémentation
Programmer l'algorithme à l'aide d'une fonction `Matlab` dont on précisera les arguments, prévoir de passer la matrice de projection P en paramètre.

Partie - 2- 2 *Application à un problème concret*

On considère un problème du type $-\varepsilon\Delta u = f,$ dans $\Omega, \quad \frac{\partial u}{\partial n} = 0 \quad$ sur $\quad \partial\Omega$. On s'intéresse pour simplifier à la version mono-dimensionnelle donnée par

$$\left\{ \begin{array}{rcll} -\varepsilon u''(x) & = & f(x) & \text{dans }]0,1[, \\ u'(1) & = & 0 & , \\ u'(0) & = & 0 & \end{array} \right. \tag{1}$$

où f est une fonction continue sur l'intervalle $[0, 1]$ et $\varepsilon > 0$ est un réel.

Q-13 : Montrer que si $\int_0^1 f(x)\,dx \neq 0$ alors le problème (1) n'admettra pas de solution.

Dans la suite on suppose que $\int_0^1 f(x)\,dx = 0$. On pourra par exemple prendre $f(x) = \frac{1}{2} - x$.

Q-14 : Montrer que si une fonction u est solution de (1), il en sera de même de $u + c, \quad \forall c \in \mathbb{R}$

Q-15 : Montrer que (1) a une unique solution si l'on impose à cette solution d'être à moyenne nulle, c'est-à-dire

$$\int_0^1 u(x)\,dx = 0. \tag{2}$$

(*On pourra déterminer analytiquement cette solution en résolvant le problème.*)

Ainsi, le problème que l'on considère dans la suite est le problème (1)-(2) qu'on choisit de discrétiser par une méthode des différences finies.

On commence par recouvrir $[0,1]$ d'une grille uniforme de pas $h = \frac{1}{n-1}$, pour un $n > 1 \in \mathbb{N}$. Ceci génère les noeuds

$$\mathcal{X}_n = \{x_i, x_i = (i-1) \times h, i \in \{1, \dots, n\}\}\,.$$

On écrit ensuite que l'équation est vérifiée en chaque noeud interne, soit

$$\left\{\begin{array}{rcll} -\varepsilon u''(x_i) & = & f(x_i) & \forall i \in \{2, \dots, n-1\}, \\ u'(x_1) & = & 0, & \\ u'(x_n) & = & 0, & \\ \int_0^1 u(x)\,dx & = & 0 & . \end{array}\right. \tag{3}$$

On remplace enfin les opérateurs différentiels par des différences divisées et on approche l'intégrale par une formula de quadrature. On choisit la formule des trapèzes pour approcher l'intégrale sur chaque intervalle
(i.e $\int_{x_i}^{x_i+h} u(x)\,dx = \frac{h}{2}(u(x_i) + u(x_i+h)) + \mathcal{O}(h^3)$)

Q-15-1 : Montrer que u solution de (3) vérifie :

$$\left\{\begin{array}{rcll} \varepsilon\dfrac{-u(x_{i-1}) + 2u(x_i) - u(x_{i+1})}{h^2} + \mathcal{O}(h^2) & = & f(x_i), \quad \forall i = 2, \dots, n-1 & \\ \dfrac{u(x_2) - u(x_1)}{h} + \mathcal{O}(h) & = & 0, & \\ \dfrac{u(x_n) - u(x_{n-1})}{h} + \mathcal{O}(h) & = & 0, & \\ h\left(\dfrac{1}{2}u(x_1) + \displaystyle\sum_{i=2}^{n-1} u(x_i) + \dfrac{1}{2}u(x_n)\right) + \mathcal{O}(h^2) & = & 0. & \end{array}\right. \tag{4}$$

Pour tout $i = 1, \dots, n$, on désigne par u_i la valeur approchée de $u(x_i)$ lorsqu'on se débarrasse des restes dans les formules (4). De même on définit $f_i = f(x_i), \quad \forall i = 2, \dots, n-1$ et $f_1 = 0, f_n = 0$.

Q-15-2 : Montrer qu'on obtient le problème

$$\left\{\begin{array}{rcll} u_1 - u_2 & = & 0, & \\ -u_{i-1} + 2u_i - u_{i+1} & = & \frac{h^2}{\varepsilon} f_i, \quad \forall i = 2, \dots, n-1 & \\ -u_{n-1} + u_n & = & 0, & \\ u_1 + \left(\displaystyle\sum_{i=2}^{n-1} 2u_i\right) + u_n & = & 0. & \end{array}\right. \tag{5}$$

Q-15-3 : Montrer que l'équation (5) se met sous la forme $\left\{\begin{array}{rcl} Ax & = & b, \\ Bx & = & 0, \end{array}\right.$
où A est une matrice carrée de taille n et B est une matrice $1 \times n$, x et b sont des vecteurs de taille n, tous donnés par :

$$A = \begin{pmatrix} 1 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & & 0 \\ 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 1 \end{pmatrix}, \quad x = \begin{pmatrix} u_1 \\ u_2 \\ \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix}, \quad b = \frac{h^2}{\varepsilon}\begin{pmatrix} f_1 \\ f_2 \\ \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix}, \quad B^T = \begin{pmatrix} 1 \\ 2 \\ \\ \vdots \\ 2 \\ 1 \end{pmatrix}$$

On rappelle qu'on a posé $f_1 = f_n = 0$. Vérifier qu'on a effectivement $Bb = 0$.

Q-16 : Montrer que ce problème rentre dans le formalisme de l'exercice précédent.

Q-17 : **Implémentation :** On prendra dans les tests $f(x) = \frac{1}{2} - x, \varepsilon = 1$

Q-17-1 : Écrire un script `Matlab` qui construit les matrices A, B et le vecteur b et qui teste l'algorithme du gradient conjugué projeté de l'exercice précédent.

Q-17-2 : Comparer la solution obtenue avec celle obtenue en résolvant le système linéaire $Ax = b$ avec `Matlab` et avec celle obtenue par la commande suivante de `Matlab`: `u = [A;B]\[b;0]` . Commentez.

Partie - 3 *Optimisation sous contrainte : méthode du Lagrangien augmenté*

On veut résoudre un problème de minimisation sous contrainte de la forme :

$$\text{Minimiser } J(\mathbf{u}) \text{ sur } K = \{\mathbf{v} \in \mathbb{R}^N : \varphi_1(\mathbf{v}) \leq 0, \ldots, \varphi_d(\mathbf{v}) \leq 0\}. \tag{6}$$

Afin de tirer partie des méthodes établies pour la minimisation sans contrainte, une classe de méthodes de résolution du problème (6) repose sur le principe suivant : construire une suite de problèmes de minimisation sans contrainte dont la suite des solutions convergera vers la solution du problème (6).

Dans cette classe de méthode on distingue la **méthode de pénalisation** (vue et TP) et la méthode du **Lagrangien augmenté**. Dans ces deux méthodes, la fonctionnelle à minimiser sans contrainte est donnée par :

$$L_{\lambda,\eta}(\mathbf{u}) = J(\mathbf{u}) + \sum_{i=1}^{d} \Psi(\varphi_i(\mathbf{u}), \lambda_i, \eta) \tag{7}$$

où $\lambda = (\lambda_1, \ldots, \lambda_d)$ et $\Psi(t, \sigma, \eta)$ est une fonction de trois variables réelles bien choisie, introduite ici pour unifier la présentation des deux méthodes.

Note 2 (Choix de Ψ dans la méthode de pénalisation)**.**

La méthode de pénalisation est très répandue. L'une des variantes les plus utilisées est la *pénalisation (externe) quadratique* qui correspondrait à prendre

$$\Psi(t, \sigma, \eta) = \frac{1}{\eta} \max(t, 0)^2 = \begin{cases} \dfrac{t^2}{\eta} & \text{si} \quad t \geq 0 \\ 0 & \text{sinon} \end{cases} \tag{8}$$

On observera bien, qu' ici, $\Psi(t, \sigma, \eta)$ ne dépend pas de σ !

Note 3 (Choix de Ψ dans la méthode du Lagrangien augmenté)**.**

La méthode du Lagrangien augmenté est utilisée en majeure partie lorsque les contraintes sont toutes de type égalité. Les variables d'écart (en anglais *slack variables*) sont donc généralement introduites pour transformer les contraintes d'inégalité en contraintes d'égalité. Mais lorsque toutes les contraintes sont des inégalités, comme dans le problème considéré ici, on peut dériver une méthode du Lagrangien augmenté sans recourir aux variables d'écart. Cela revient à prendre

$$\Psi(t, \sigma, \eta) = \begin{cases} \sigma t + \dfrac{t^2}{2\eta} & \text{si} \quad t + \eta\sigma \geq 0 \\ -\dfrac{\eta}{2}\sigma^2 & \text{sinon} \end{cases} \tag{9}$$

L'algorithme de résolution peut aussi être unifié de la manière suivante :

Méthode de pénalisation ou du Lagrangien augmentée
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 $\mathbf{v} \mapsto L_{\lambda^k,\eta_k}(\mathbf{v})$: 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.

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

La résolution du problème de minimisation sans contrainte peut se faire avec les fonctions `fminsearch`, `fminunc ...` de `Matlab`. **Cependant un plus grand nombre de points sera accordé à ceux qui utiliseront les outils développés en TPs.**

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 :
$$\eta_{k+1} = \frac{\eta_k}{100}. \tag{10}$$
2. **Multiplicateur de Lagrange dans la méthode du Lagrange augmenté** :
   La mise à jour du multiplicateur de Lagrange est faite dans le cadre de la formule (9) par :
$$\lambda_i^{k+1} = \begin{cases} 0 & \text{si} \quad g_i(\mathbf{u}^k) + \frac{\lambda_i^k}{\eta_k} \leq 0, \\ \lambda_i^k + \dfrac{g_i(\mathbf{u}^k)}{\eta_k} & \text{sinon.} \end{cases}, \ i = 1, \ldots, d. \tag{11}$$
3. **Multiplicateur de Lagrange dans la méthode pénalisation** :
   La méthode de pénalisation ne dépend pas du multiplicateur de Lagrange. Néanmoins nous fournissons la formule suivante de mise à jour, basée sur l'estimation du multiplicateur de Lagrange :
$$\lambda_i^{k+1} = \frac{\partial \Psi}{\partial t}(g_i(\mathbf{u}^k), \lambda_i^k, \eta_k) = \begin{cases} \dfrac{2}{\eta_k} g_i(\mathbf{u}^k) & \text{si} \quad g_i(\mathbf{u}^k) \geq 0 \\ 0 & \text{sinon} \end{cases}, \ i = 1, \ldots, d. \tag{12}$$

Dans la suite, on prendra :

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \in \mathbb{R}^2 \quad \text{et} \quad J(\mathbf{u}) = (u_1 - 0.5)^2 + (u_2 - 0.5)^2,$$

$$d = 4 \quad \text{et} \quad \begin{cases} \varphi_1(\mathbf{v}) = v_2 - v_1^2, \\ \varphi_2(\mathbf{v}) = \frac{v_1^2}{2} - v_2, \\ \varphi_3(\mathbf{v}) = v_1 - 2v_2^2 + 1, \\ \varphi_4(\mathbf{v}) = v_2^2 - v_2 - 0.5. \end{cases}$$

Q-18 : Écrire les fonctions `J.m` et `DJ.m` qui prennent en argument $\mathbf{u}$ de composante (u_1, u_2) et qui retournent respectivement $J(\mathbf{u})$ et $\nabla J(\mathbf{u})$;

Q-19 : Écrire les fonctions `g.m` et `Dg.m` qui prennent en arguments $\mathbf{u}$ de composante (u_1, u_2) et un entier i et qui retournent respectivement $g_i(\mathbf{u})$ et $\nabla g_i(\mathbf{u})$;

Q-20 : Écrire les fonctions `Psi.m, DPsi.m` qui prennent en arguments trois réels t, σ, η et qui retournent respectivement $\Psi(t, \sigma, \eta)$ et $\dfrac{\partial \Psi(t, \sigma, \eta)}{\partial t}$.

Q-21 : Calculer le gradient de $L_{\lambda,\eta}$. Puis écrire les fonctions `L.m` et `gradL.m` qui prennent en arguments $\mathbf{u}$, λ et η et qui renvoient respectivement $L_{\lambda,\eta}(\mathbf{u})$ et $\nabla L_{\lambda,\eta}(\mathbf{u})$.

Q-22 : Écrire une fonction `mettreAjourParametres.m` qui prend en arguments $\lambda^k, \eta_k, \mathbf{u}^k$ et qui retourne $\lambda^{k+1}, \eta_{k+1}$ selon les formules de mise à jour ci-dessus (voir Note 5).

Q-23 : Écrire une fonction `minLSansContrainte.m` qui prend en argument $\mathbf{u}^0, \lambda, \eta, \varepsilon$ ainsi qu'un nombre maximum d'itérations et qui retourne le minimiseur de $\mathbf{v} \mapsto L_{\lambda,\eta}(\mathbf{v})$, ainsi que le nombre d'iterations effectuées.
(*Elle fera donc appel aux fonctions* `L.m` *et* `gradL.m`).

Q-24 : Écrire une fonction `PenaOuLagrangienAugmenteLIM.m` qui implémente les méthodes de pénalisation ou du lagrangien augmenté données ci-dessus :

- Elle retournera la solution $\mathbf{u}$, la valeur approchée du multiplcateur de Lagrange λ, la dernière valeur du paramètre de pénalisation η ainsi que le nombre d'itérations effectuées.
- Elle prendra pour arguments $\mathbf{u}^0, \lambda^0, \eta_0$, ε et fera appel de manière implicite aux fonctions $L.m$ et $dradL.m$ etc. *On mettra par conséquent dans le fichier un commentaire prévenant toute modification de l'un de ces fichiers.* **Remarque :** Pour vous épargner ces commentaires vous pouvez aussi passer ces fonctions en arguments de `PenaOuLagrangienAugmenteLIM.m`.

Q-25 : Créer un script `scriptLagrangienAugmente.m` et résoudre le problème minimisation sous contrainte $\min_{\mathbf{u}\in K} J(\mathbf{u})$ à l'aide de l'algorithme . On prendra $\varepsilon = 10^{-7}$, $\mathbf{u}^0 = (0,0)^T$ et $\lambda^0 = (0,0,0,0)^T$. Tracer sur une même figure les lignes de niveau 0 des φ_i, les lignes de niveau de J, ainsi que la solution $\mathbf{u}^*$ obtenue. On utilisera les fonctions `meshgrid` et `contour`, et on se placera sur $[-1,2] \times [-1,2]$. On commentera les figures obtenues dans les cas de la méthode de pénalisation et de la méthode du Lagrangien augmenté. Comparer ces méthodes dans le cadre du problème considéré.

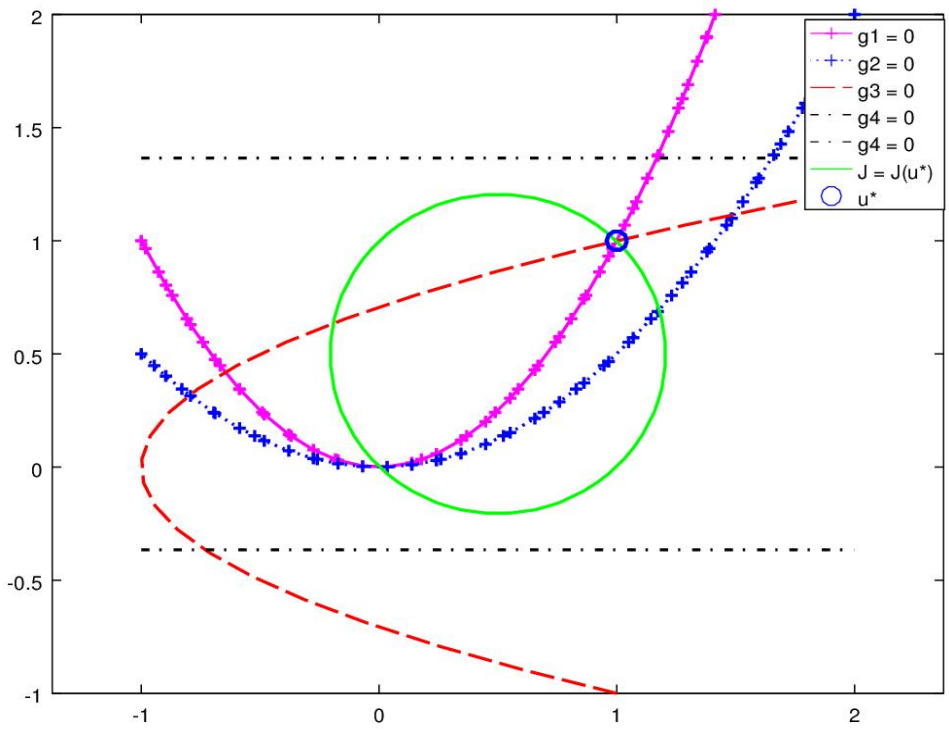

FIGURE 1: Exemple de sortie graphique de la **Partie- 3**