Rapport de projet

Olivier GLADING - Julien SALOMON

ENSTA - Promotion 2002

Optimisation convexe non différentiable

octobre 2001

Introduction

Le présent rapport propose l'étude par des exemples de diverses méthodes d'optimisation convexe non-différentiable. Dans un premier temps, il sera notamment procédé à la mise en œuvre d'un algorithme de type algorithme du gradient puis de la méthode des plans sécants. En second lieu, on utilisera une méthode de décomposition duale pour un problème de minimisation quadratique.

On trouvera donc dans ce rapport tout d'abord quelques rappels théoriques, puis la particularisation par des exemples et enfin, les résultats obtenus et leurs interprétations (on trouvera par ailleurs en annexe les sources SCILAB correspondantes).

À Yves CAUCHON, notre mentor et ami.

1 Étude d'une courbe

Nous étudierons la fonction $f : \mathbb{R} \to \mathbb{R}$ définie par :

$$f(x) = \max\left[P_1(x), P_2(x), P_3(x), P_4(x), P_5(x)\right]$$

avec

$$\begin{aligned}
P_1(x) &= 0.005\ x^2 - x \\
P_2(x) &= 0.05\ x^2 - 2\ x \\
P_3(x) &= 0.5\ x^2 - 3\ x \\
P_4(x) &= 0.1\ x^2 - 4\ x \\
P_5(x) &= 0.03\ (x-1)^4 - 3\ x
\end{aligned}$$

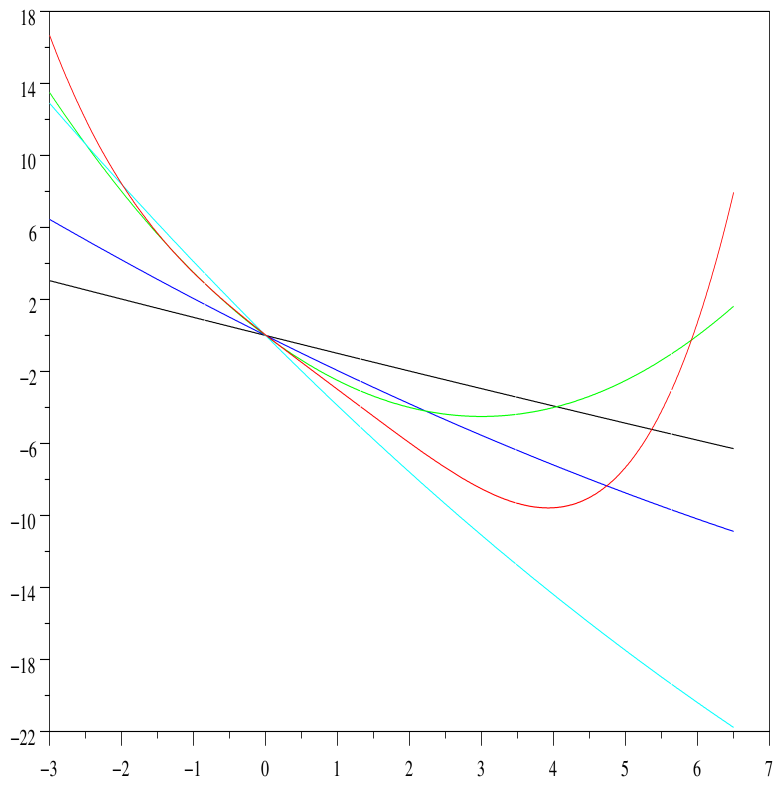

Figure 1 - Allure des polynômes $(P_i)_{i\in[1,5]}$

La fonction *simul.sci* (voir annexe 1) prend pour argument un réel x et donne en sortie la valeur explicite $f(x) = P_i(x)$ de f ainsi que celle de son gradient $g(x) = \nabla f(x) = P_i'(x)$ avec :

$$
\begin{aligned}
P_1'(x) &= 0.01\ x - 1 \\
P_2'(x) &= 0.1\ x - 2 \\
P_3'(x) &= x - 3 \\
P_4'(x) &= 0.2\ x - 4 \\
P_5'(x) &= 0.12\ (x-1)^2 - 3
\end{aligned}
$$

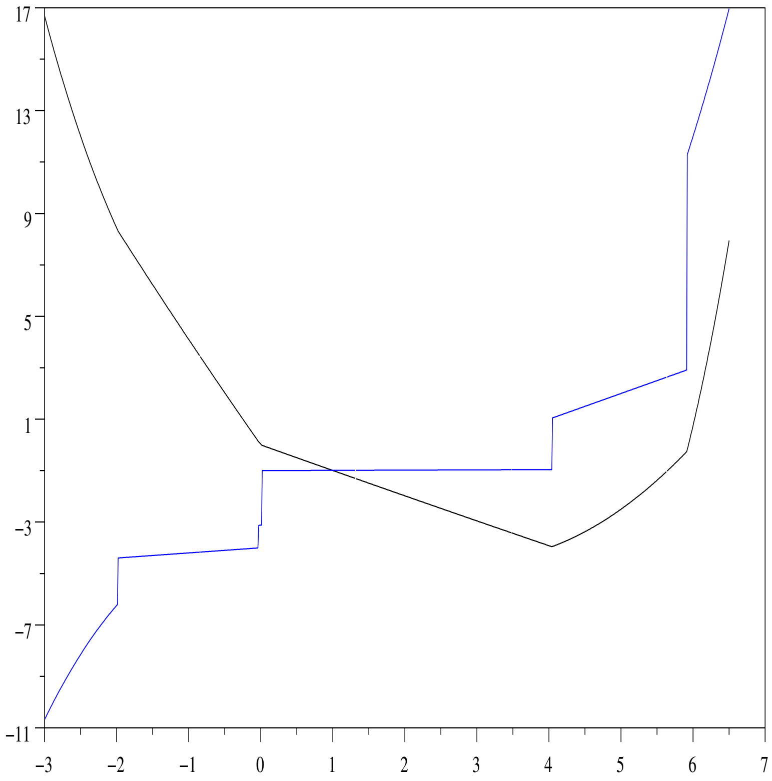

Figure 2 - Fonction f et sa dérivée

On utilise dans ce code la fonction maximum de SCILAB qui permet de choisir dans le cas précis de f la plus grande valeur en x choisie parmi les 5 polynômes. La valeur dérivée est alors associée.

Dans un second temps, à l'aide de la fonction plot2d de *SCILAB* la fonction *plot-simul.sci* (voir annexe 2) donne la représentation graphique de $f(x)$ (voir figure 2).

2 Calcul du sous-gradient

Dans cette partie, on procède au calcul du sous-gradient dans le cas de la fonction définie dans la partie précédente. La fonction *sous-gradient.sci* donne en sortie le sous-gradient, étant donnés en entrée :
- le point initial x_0 pour lequel commence l'algorithme
- le pas ϵ
- le nombre N d'itérations
- une valeur test $mode - it$ qui détermine si on on utilise la méthode classique à pas constant (mode-it = 1) ou la méthode à série divergente (deux cas : mode-it = 2 et 3).

Soit le problème d'optimisation

$$\min_{u \in \mathbb{R}^n} J(u)$$

avec J convexe continue et non différentiable.

Introduisons tout d'abord le sous-différentiel :

$$\partial J(u) = \{ r \in \mathbb{R}^n \mid \forall v \in \mathbb{R}^n, J(v) - J(u) \geq \ < r \ , v - u > \}$$

On note U^* l'ensemble des minima de J, u^* un élément quelconque de U^* et $J^* = J(u^*)$. Ci-dessous nous présentons deux algorithmes basés sur le sous-gradient qui utilisent toutes deux une suite à série divergente : ils permettent de calculer une suite (u^k) qui converge vers u^*. La convergence vers 0 de la distance à l'optimum est donnée par

$$d(u^k, U^*) = \min_{u^* \in U^*} \|u^k - u^*\|.$$

Donnons quelques résultats classiques de schémas numériques convergents :

a) pour une suite ε_k $(dans)$ $\mathbb{R}^{\mathbb{N}}$ à série divergente, on a la convergence de l'algorithme suivant (algorithme de sous-gradient normalisé) :

$$\begin{cases} u^0 \in \mathbb{R}^n \\ u^{k+1} = u^k - \varepsilon^k \frac{g^k}{\|g^k\|} \end{cases} \qquad \text{avec } g^k \in \partial J(u^k).$$

b) pour une suite ε_k $(dans)$ $\mathbb{R}^{\mathbb{N}}$ à série divergente, mais convergente dans $\mathcal{L}^2$ (σ-suite) : l'algorithme suivant converge (algorithme du sous-gradient) :

$$\begin{cases} u^0 \in \mathbb{R}^n \\ u^{k+1} = u^k - \varepsilon^k g^k \end{cases} \qquad \text{avec } g^k \in \partial J(u^k).$$

Seul ce deuxième résultat théorique sera appliqué à la fonction définie dans la première partie.

2.1 Algorithme à pas constant

Dans un premier temps, il est légitime (comme pour le cas différentiable) de vouloir d'abord utiliser un algorithme à pas constant utilisant le schéma numérique suivant :

$$x^{k+1} = x^k - \varepsilon g^k$$

où g^k désigne la valeur du gradient de f en x^k, ε étant fixé a priori.

Cet algorithme est mis en œuvre dans la fonction *sous-gradient* pour *mode-it* $= 1$ (annexe 3). Les résultats sont présentés en section 2.3.

Il semblerait que l'utilisation d'un tel algorithme ne soit pas très judicieux dans le cas non-différentiable. En effet, la condition d'optimalité $\nabla J(u^*) = 0$ n'a pas de sens et, d'autre part, si (u^k) est une suite convergeant vers u^*, la limite de $\nabla J(u^k)$ quand $u^k \to u^*$ n'a aucune chance d'être nulle.

Cela dit, quand bien même l'algorithme précédent ne convergerait pas, il serait possible, a posteriori, de réduire le pas si on constate que l'algorithme oscille trop.

2.2 Algorithme à série divergente

Le pas prend la forme $\varepsilon^k = \varepsilon\theta^k$, où θ^k est le terme général d'une série divergente (le point initial de l'algorithme x^1 est choisi dans l'intervalle $[-7, 7]$). Nous allons considérer deux cas : suite à série divergente puis suite à série divergente mais appartenant à $\mathcal{L}^2$.

a) $\theta^k = \frac{1}{\sqrt{k}}$

Cet algorithme est présenté dans la fonction *sous-gradient* pour *mode-it* $= 3$ (annexe 3) et les résultats en section 2.3.

b) $\theta^k = \frac{1}{k}$

On présente cet algorithme dans la fonction *sous-gradient* pour *mode-it* $= 2$ (annexe 3), les résultats associés en section 2.3.

x^0	ε	$N_{iterations}$	Algorithme 1	Algorithme 2	Algorithme 3
8	0.1	100	4.083	4.04	4.048
6	0.1	100	4.112	4.18	4.031
4	0.1	100	4.01	4.0398	4.033
2	0.1	100	4.06	2.51	3.85
0	0.1	100	3.95	0.72	2.05
2	0.1	1000	3.96	2.73	4.039
0	0.1	1000	4.06	0.95	4.042
-2	0.1	1000	4.07 ± 0.1	0.325	4.039
-6	0.1	1000	4.01 ± 0.1	0.18	4.038
3	1	1000	4.0 ± 1	4.04	4.04 ± 0.02

Figure 3 - Résultats pour les trois schémas numériques

2.3 Résultats

Les résultats obtenus sont présentés ci-après en Figure 3. Les trois algorithmes ont été testés pour *mode-it* $= 1$ (pas constant), 2 ($\theta_k = 1/k$) et 3 ($\theta_k = 1/\sqrt{k}$).

Pour un nombre d'itérations faible, on remarque que les trois algorithmes fonctionnent, pour peu que l'on situe le point initial au-delà de 4 (en réalité, au-delà de la valeur du minimum qui est de l'ordre de 4.04). En revanche, pour un point initial situé en-dessous de 4, les algorithmes 2 et 3 donnent des résultats médiocres. Cela s'explique en observant la courbe de la dérivée, qui dans l'intervalle $[0\,;4]$ prend des valeurs très faibles, qui contribuent de fait peu à la progression de l'algorithme. En augmentant le nombre d'itérations, on remarque que l'on assure une bonne convergence pour l'algorithme 3, tandis qu'une telle modification est inefficace pour l'algorithme 2.

À noter le fait que l'algorithme 1 nous fournit une convergence relative : en effet le résultat oscille au bout d'un certain temps autour de la valeur cherchée (environ 4.04), avec une erreur relative de l'ordre du pas ε choisi (pour $\varepsilon = 1$, oscillations grossières ; on remarque également que pour un tel ε, l'algorithme 3 a tendance á osciller légèrement, mais dans une moindre mesure).

3 Méthode des plans sécants

On désire minimiser la fonctionnelle $J\ : \mathbb{R} \to \mathbb{R}$, convexe sur le sous-ensemble convexe compact U. La méthode des plans sécants est basée sur le fait que si l'on dispose de k triplets $(x^k, J(x^k), r^k)$ avec $r^k \in \partial J(x^k)$ (ou dans le cas différentiable $r^k = \nabla J(x^k)$), alors il est possible de construire une approximation polyédrique de l'épigraphe de J à l'aide du graphe de la fonction suivante :

$$J^k(x) = \max_{i \in [0,k]} \left(J(x^i) + r^i.(x - x^i)\right)$$

La figure 4 montre un exemple d'une telle démarche. Cette approximation peut être utilisée pour déterminer le minimum de J : en effet, si epi J^k est une bonne approximation de epi J, alors le minimum de J^k est une bonne approximation de J. Notons x^k ce mimimum approché. En connaissant $J(x^{k+1})$ et r^{k+1}, il est alors possible d'affiner l'approximation de l'épigraphe de J.

Cette méthode se met donc simplement sous forme algorithmique :

À l'étape 0 : on fixe un compact U (réputé contenir le mimimum de J) et le point initial x^0 dans U.

À l'étape k : on dispose de k triplets $(x^k, J(x^k), r^k)$ et on calcule :

$$x^{k+1} = \min_{x \in U} \left[J^k(x) = \max_{i \in [0,k]} \left(J(x^i) + r^i.(x - x^i)\right)\right]$$

Tant que le critère de convergence (fixé par l'utilisateur) n'est pas satisfait, k devient $k+1$.

La fonction *plans-secants.sci* (annexe 4) prend en argument :
- le point initial x^0
- la précision souhaitée *tolf*
- le nombre maximal d'itérations *maxit*

Elle donne en sortie :
- les x^k fournis par l'algorithme (avec $k \leq$ *maxit*)
- l'approximation souhaitée.

Cette fonction utilise la commande préintégrée linpro, qui permet de résoudre le problème suivant :

$$\begin{cases} \min\left(< p, u >\right) \\ \text{sous la contrainte :} \\ C.u \leq b \\ c_i \leq u \leq c_s \end{cases}$$

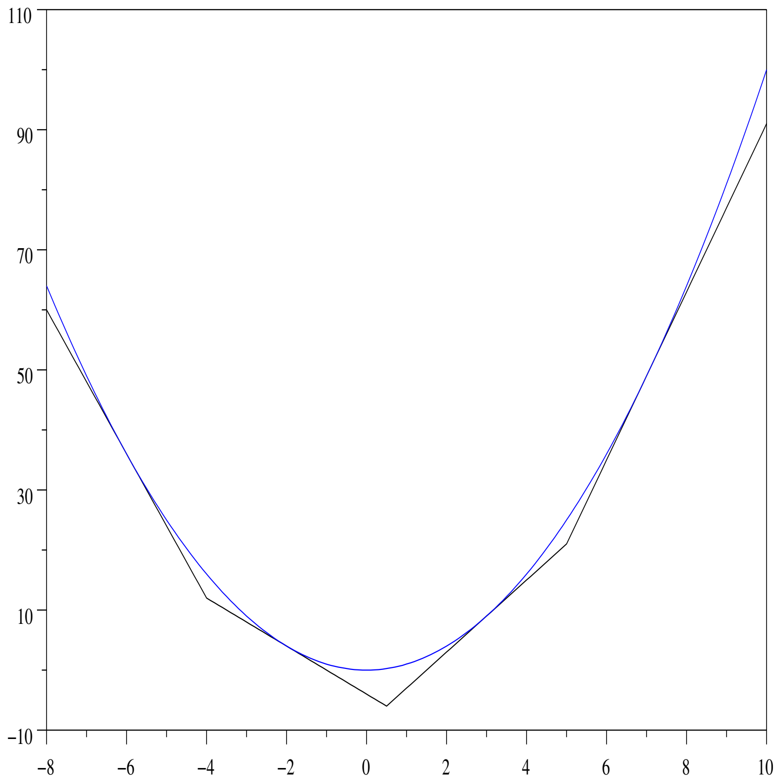

Figure 4 - Approximation polyédrique de l'épigraphe d'une fonctionnelle J

À l'étape k, on traduit la condition $\max_k J^k(x)$ en écrivant que pour tout $u = (X\ Y)'$, on a :

$$Y \geq J(x^k) + r^k(X - x^k)$$

En écrivant :

$$\begin{cases} C = \left[C\ ;\ [-1\ r^k]\right] \\ b = [b\ ;\ r^k.x^k - J(x^k)] \end{cases}$$

la condition $C.u \leq b$ équivaut à ce choix de max.
La commande *linpro* retourne alors les valeurs x^k et $J(x^k)$.

La convergence constatée semble rapide (une dizaine d'itérations pour un critère d'arrêt de l'ordre de 0.01), pour visiblement tout x^0 arbitrairement grand - cette méthode ne semble pas permettre d'oscillations. Comme pour l'autre méthode, on trouve bien une valeur de l'ordre de 4.04.

4 Minimisation sous contrainte

On considère le problème quadratique suivant :

$$\begin{cases} \min\left(p_1^2 + 2\ p_2^2 + 10\ p_1 + 2\ p_2\right) \\ \text{sous les contraintes :} \\ p_1 + p_2 = 20 \\ 0 \leq p_1 \leq 12 \\ 0 \leq p_2 \leq 12 \end{cases}$$

4.1 Résolution directe

Le logiciel SCILAB permet la résolution de problèmes quadratiques par une fonction pré-intégrée, en l'occurrence la fonction quapro.

Cette fonction quapro permet de résoudre des problèmes de la forme suivante :

$$\begin{cases} \min\left(\frac{1}{2}p'.Q.p + r'.p\right) \\ \text{sous les contraintes :} \\ C.p \leq b \\ c_i \leq p \leq c_s \end{cases}$$

Pour le cas qui nous intéresse les valeurs prises en compte par quapro sont données par :

$$p = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

$$Q = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix} \text{ et } r = \begin{pmatrix} 10 \\ 2 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} \text{ et } b = \begin{pmatrix} 20 \\ -20 \end{pmatrix}$$

$$c_i = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ et } c_s = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

Est également prise en compte une valeur arbitraire p^0 du vecteur p qui correspond à l'initialisation du vecteur quapro.

La fonction quapro donne comme solution du problème de minimisation le vecteur $(p_1, p_2) = (12, 8)$. Dans la mesure où la fonction de deux variables à minimiser peut s'exprimer uniquement en fonction de p_1 par exemple, sous la forme $\varphi(p_1)$, on peut tester graphiquement la solution trouvée en traçant cette fonction (voir figure 5 ci-dessous).

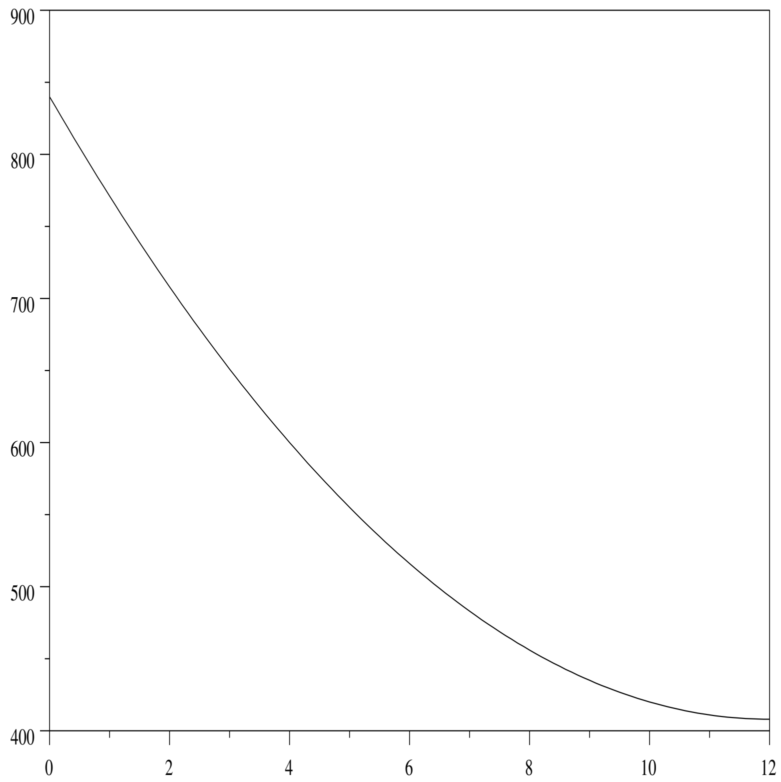

Figure 5 - Allure de φ avec $\varphi(p_1) = p_1^2 + 2p_2^2 + 10p_1 + 2p_2$ et $p_1 + p_2 = 20$

4.2 Résolution par décomposition

Rappelons les données du problème :

$$\begin{cases} \min\left(p_1^2 + 2\ p_2^2 + 10\ p_1 + 2\ p_2\right) \\ p_1 + p_2 = 20 \\ 0 \leq p_1 \leq 12 \\ 0 \leq p_2 \leq 12 \end{cases}$$

Ce problème peut s'écrire sous forme compacte selon :

$$\left\{\begin{array}{l}\min_{p\in U} J(p)\\ U=\{p \mid \phi_1(p)\leq 0,\ \phi_2(p)\leq 0,\ \phi_3(p)\leq 0\}\end{array}\right.$$

avec :

$$J : p=\begin{pmatrix} p_1\\ p_2\end{pmatrix} \longmapsto J(p)=p_1^2+2p_2^2+10p_1+2p_2$$

et par ailleurs :

$$\left\{\begin{array}{l}\phi_1(p)=C.p-b\\ \phi_2(p)=-p\\ \phi_3(p)=p+d\end{array}\right.$$

avec :

$$C=\begin{pmatrix} 1 & 1\\ -1 & -1\end{pmatrix} \text{ et } b=\begin{pmatrix} 20\\ -20\end{pmatrix}$$

et

$$d=\begin{pmatrix} -12\\ -12\end{pmatrix}$$

Le lagrangien pour ce problème convexe différentiable s'écrit :

$$L(p,\Lambda)=J(p)+\Sigma_{i=1}^{3}\lambda_i.\phi_i(p) \quad \text{avec } \Lambda=(\lambda_1,\lambda_2,\lambda_3)$$

En toute rigueur, l'étude qui suit devrait prendre en compte les trois contraintes imposées par les fonctions $(\phi_i)_{i\in\{1,2,3\}}$. Cependant, une telle approche semble lourde à mettre en œuvre ; il suffira de s'intéresser uniquement à la contrainte ϕ_1, et de restreindre l'intervalle d'étude de p (ϕ_2 et ϕ_3 correspondant à des intervalles pour p_1 et p_2), donc de λ. Ainsi, le lagrangien, du fait de cette assertion prend-il la forme suivante :

$$\left\{\begin{array}{l}L(p,\lambda)=J(p)+\lambda.\varphi(p) \quad \text{avec :}\\ \varphi(p)=p_1+p_2-20\end{array}\right.$$

Par ailleurs, il est à noter que la fonctionnelle J peut se décomposer en deux parties, l'une s'occupant exclusivement de la variable p_1, l'autre de la variable p_2. En effet :

$$J(p) = J_1(p_1) + J_2(p_2) \quad \text{avec :}$$
$$J_1(p_1) = p_1^2 + 10p_1$$
$$J_2(p_2) = 2p_2^2 + 2p_2$$

Naturellement, il faut aussi dualiser la contrainte couplante $\varphi(p) = 0$. Le lagrangien se décompose de fait selon :

$$\begin{cases} L_1(p_1, \lambda) = J_1(p_1) + \lambda(p_1 - 10) \\ L_2(p_2, \lambda) = J_2(p_2) + \lambda(p_2 - 10) \end{cases}$$

Par annulation du gradient du lagrangien, on accède aux expressions de $p_1(\lambda)$ et $p_2(\lambda)$:

$$\frac{\partial L_1}{\partial p_1} = 2p_1 + 10 + \lambda = 0$$
$$\frac{\partial L_2}{\partial p_2} = 4p_2 + 2 + \lambda = 0$$

soit :

$$\begin{cases} p_1^* = -\frac{1}{2}(\lambda + 10) \\ p_2^* = -\frac{1}{4}(\lambda + 2) \end{cases}$$

Les conditions d'encadrement sur p_1 et p_2 donnent respectivement :

$$-34 \leq \lambda \leq -10$$
$$-50 \leq \lambda \leq -2$$

Soit au final $\lambda \in [-34, -10]$. Redonnons l'expression du lagrangien en fonction de p_1 et p_2 :

$$L(p_1, p_2, \lambda) = p_1^2 + 2p_2^2 + +10p_1 + 2p_2 + \lambda(p_1 + p_2 - 20)$$

qui donne en injectant les expressions de p_1 et p_2 :

$$L^*(\lambda) = -\frac{3}{8}\lambda^2 - \frac{51}{2}\lambda - \frac{51}{2}$$

On veut maximiser L^* en λ, ce qui équivaut à minimiser son opposée, à savoir :

$$f(\lambda) = \frac{3}{8}\lambda^2 + \frac{51}{2}\lambda + \frac{51}{2} \quad \text{pour} \quad \lambda \in [-34, -10]$$

La fonction *simul2.sci* (annexe 5) prend pour argument une valeur de la variable duale λ et donne en sortie la valeur de $f(\lambda)$, opposée de la fonction duale pour ce λ, ainsi que la dérivée de f en λ :

$$g(\lambda) = f'(\lambda) = \frac{3}{4}\lambda + \frac{51}{2} \quad \text{(noter que cette dérivée s'annule en } \lambda = -34\text{)}$$

Pour minimiser f (donc maximiser la fonction duale), nous allons utiliser la fonction *sous-gradient2.sci* (annexe 6), rigoureusement identique à la fonction *sous-gradient* à ceci près qu'elle fait appel à la fonction *simul2.sci* (et non plus *simul*).

Les trois modes d'itération (pas constant, σ-suite et série divergente) de la fonction *sous-gradient2* convergent toutes trois vers la valeur $\lambda = -34$, soit $p_1 = 12$ et $p_2 = 8$. On peut noter toutefois que l'algorithme à pas constant semble le plus rapide dans ce cas précis (ce n'est pas surprenant puisque le lagrangien est parfaitement différentiable).

De même nous pouvons utiliser la fonction *plans-secants2.sci* (annexe 7) qui est une variante de la fonction *plans-secants*, appliquée au présent problème. On obtient également le résultat $\lambda = -34$. C'est manifestement la méthode la plus rapide en terme d'itérations.

5 Annexes

1 - Fonction simul.sci

2 - Fonction plot-simul.sci

3 - Fonction sous-gradient.sci

4 - Fonction plans-secants.sci

5 - Fonction simul2.sci

6 - Fonction sous-gradient2.sci

7 - Fonction plans-secants2.sci