1 TD 12 : Optimisation sous contrainte

(correction page ??)

Abordé lors de cette séance	
programmation	calcul matriciel
algorithme	optimisation sous contrainte

Première demi-heure : Premier problème .

On veut optimiser le problème suivant :

$$\begin{cases} \min_{x,y} x^2 + y^2 - xy + y \\ \text{sous contrainte } x + 2y = 1 \end{cases} \tag{1}$$

Utiliser le module `cvxopt` [1] pour résoudre ce problème. Quelques fonctions utiles :

```
from cvxopt import solvers, matrix
m = matrix( [ [2.0, 1.1] ] )  # mettre des réels (float) et non des entiers
                              # cvxopt ne fait pas de conversion implicite
t = m.T                       # la transposée
t.size                        # affiche les dimensions de la matrice
```

La documentation `cvxopt` de la fonction est parfois peu explicite. Il ne faut pas hésiter à regarder les exemples d'abord. Le plus intéressant pour le cas qui nous intéresse est celui-ci [2] :

```
from cvxopt import solvers, matrix, spdiag, log

def acent(A, b):
    m, n = A.size
    def F(x=None, z=None):
        if x is None: return 0, matrix(1.0, (n,1))
        if min(x) <= 0.0: return None
        f = -sum(log(x))
        Df = -(x**-1).T
        if z is None: return f, Df
        H = spdiag(z[0] * x**-2)
        return f, Df, H
    return solvers.cp(F, A=A, b=b)['x']
```

Cet exemple résoud le problème de minimisation suivant :

$$\begin{cases} \min_X -\sum_{i=1}^{n} \ln x_i \\ \text{sous contrainte } Ax = b \end{cases} \tag{2}$$

1. la fonction `cp` : http://cvxopt.org/userguide/solvers.html#problems-with-nonlinear-objectives
2. tiré de la page : http://cvxopt.org/userguide/solvers.html#problems-with-nonlinear-objectives

Remarque 0.1 : numpy et cvxopt
Les deux modules `numpy` et `cvxopt` n'utilisent pas les mêmes matrices bien qu'elles portent le même nom dans les deux modules. Les fonctions de `cvxopt` ne fonctionnent qu'avec les matrices de ce module. Il ne faut pas oublier de convertir la matrice quand elle est décrite par une autre classe.

```
m = cvxopt.matrix( a convertir )
```

Seconde et troisième demi-heure : L'algorithme de Arrow-Hurwicz

On cherche à résoudre le même problème avec l'algorithme de Arrow-Hurwicz [3] et les notations suivantes :

$$\begin{cases} \min_U J(U) = u_1^2 + u_1^2 - u_1 u_2 + u_2 \\ \text{sous contrainte } \theta(U) = u_0 + 2u_1 - 1 = 0 \end{cases} \tag{3}$$

L'algorithme est le suivant :

1. On choisit $\epsilon > 0$, $\rho > 0$, des vecteurs aléatoires $U_0 \in \mathbb{R}^2$ et $P_0 \in \mathbb{R}^d$ (d est le nombre de contraintes).
2. A l'itération k, on met à jour :

$$U_{t+1} = U_t - \epsilon(\nabla J(U_t) + [\nabla \theta(U_t)]' P_t \tag{4}$$
$$P_{t+1} = P_t + \rho\theta(U_{t+1}) \tag{5}$$

3. On retourne à l'étape précédente jusqu'à ce que la suite (U_k) n'évolue plus.

Le coefficient P_t correspond au coefficient de Lagrange pour un Lagrangien défini comme suit :

$$L(U, P) = J(U) + P'\theta(U) \tag{6}$$

Implémenter cet algorithme et vérifier qu'il converge vers la même solution.

Quatrième demi-heure : Le lagrangien augmenté

On reprend l'algorithme précédent mais appliqué à la fonction :

$$A(U) = J(U) + \frac{c}{2}\theta^2(U) \tag{7}$$

L'objectif est ici de remplacer la fonction J par la fonction A (ou Lagrangien augmenté) dans l'algorithme précédent et de vérifier qu'il converge en un nombre moindre d'itérations lorsqu'on choisit les mêmes conditions initiales.

3. Tiré de *Introduction à l'optimisation*, de Jean-Christophe Culioli, 2nd édition, éditions Ellipses

Pour aller plus loin ou pour ceux qui ont fini plus tôt

Prolongement 1

On souhaite maintenant ajouter une contrainte d'inégalité : $Du - d \leqslant 0$. L'idée est de créer une matrice Θ définie par bloc :

$$\Theta = \begin{pmatrix} \theta \\ D \end{pmatrix} \ , \ \Pi = \begin{pmatrix} P \\ D \end{pmatrix} \ , \ \mathbb{R} = diag \begin{pmatrix} 1 \\ r_i \end{pmatrix} \tag{8}$$

On considère que Θ est une matrice. On étend ce découpage en bloc au formule de mise à jour de l'algorithme de Arrow-Hurwicz :

$$U_{t+1} = U_t - \epsilon(\nabla J(U_t) + [\nabla \Theta(U_t)]' \, \Pi_t \tag{9}$$
$$\Pi_{t+1} = \mathbb{R} \, (\Pi_t + \rho \theta(U_{t+1})) \tag{10}$$

Les coefficients r_i valent 0 ou 1 selon que la contrainte d'inégalité i est respectée ou non. On applique la modification pour le problème :

$$\begin{cases} \min_U J(U) = u_1^2 + u_1^2 - u_1 u_2 + u_2 \\ \text{sous contrainte } \theta(U) = u_0 + 2u_1 - 1 = 0 \text{ et } u_1 \geqslant 0.3 \end{cases} \tag{11}$$

Prolongement 2

On s'intéresse à un problème différent :

$$\begin{cases} \min_X C'X \\ \text{sous contrainte } AX \leqslant B \end{cases} \tag{12}$$

Une façon de résoudre ce problème est l'algorithme de Karmarkar [4]. On optimise maintenant une fonction linéaire et non une fonction quadratique. Par conséquent, la solution est forcément sur un bord : des contraintes sont activées.

Remarques

Vous pouvez découvrir un autre exemple d'optimisation dynamique au travers de ce blog : *Optimisation sous contraintes appliquée au calcul du report des voix* [5].

4. http://fr.wikipedia.org/wiki/Algorithme_de_Karmarkar
5. http://www.xavierdupre.fr/blog/2013-12-07_nojs.html