Utilisation du solveur linéaire GLPK et du solveur linéaire et quadratique convexe XPress

ECMA

Semaine du 12 décembre 2011

Dans ce TP, nous présentons l'utilisation de deux solveurs de programmation mathématique. Le premier appelé GNU Linear Programming Kit (GLPK) est un solveur qui permet de résoudre des programmes linéaires en variables mixtes entières, c'est à dire des problèmes dont la fonction à optimiser est linéaire, et où les variables peuvent être soit entières, soit réelles. GLPK est un solveur que l'on peut utiliser gratuitement, sans limite de taille des problèmes résolus. Le deuxième, XPress, est un logiciel propriétaire, qui permet de résoudre des programmes dont la fonction à optimiser est convexe, c'est à dire soit linéaire, soit quadratique et convexe. Comme pour GLPK, les variables du problème peuvent être soit entières, soit réelles. Le logiciel Xpress étant propriétaire, il n'est possible d'utiliser qu'une version étudiante de Xpress, qui ne peut pas résoudre des problèmes plus grand que 500 variables et 500 contraintes.

1 Utilisation du solveur linéaire GLPK

Avant de commencer :

1. Télécharger le solveur linéaire GNU Linear Programming Kit (GLPK) à l'adresse suivante : `http://sourceforge.net/projects/winglpk/`.
2. Modifier la variable d'environnement `PATH` pour que le système sache où trouver l'exécutable `glpsol.exe` au moment de l'appel. Pour cela, aller dans :
   `Panneau de configuration -> Système -> paramètres avancés -> variable d'environnement`
   et ajouter le chemin absolu du répertoire contenant `glpsol.exe`. Exemple :
   `C:\Users\lambert\Desktop\glpk-4.47\ w64\;`
3. Le logiciel s'utilise ensuite sous l'éditeur de commande. Vérifiez que vous avez bien accès au logiciel GLPK, par exemple en tapant la commande
   `glpsol -help`
4. les documentations de GLPK et de son langage de modélisation de GMPL se trouvent dans le répertoire `doc`.

Exercice 1 On veut modéliser et résoudre le problème :

$$(P_1)\begin{cases} Max & z = f(x_1, x_2) = 3x_1 + 5x_2 \\ s.c. & x_1 \leqslant 4 \\ & x_2 \leqslant 6 \\ & 3x_1 + 2x_2 \leqslant 18 \\ & x_1 \geqslant 0\,;\, x_2 \geqslant 0 \end{cases}$$

Pour ce faire, vous pouvez créer le fichier texte suivant, qui contient le problème à résoudre au format lp. Vous pouvez nommer ce fichier ex1.lp

```
\* Probleme: exemple1 *\
Maximize
 z: + 3 x1 + 5 x2
```

```
Subject To
 c1:  x1 <= 4
 c2:  x2 <= 6
 c3:  3 x1 + 2 x2 <= 18

End
```

1. Éditer ce fichier par votre éditeur de texte favori
2. Vous pouvez ensuite résoudre le contenu de ce fichier par la commande :
   `glpsol -cpxlp ex1.lp`
   et si vous souhaitez de plus obtenir la solution dans un fichier nommé ex1.sol, vous pouvez utiliser :
   `glpsol -cpxlp ex1.lp -output ex1.sol`
   Observez ce qui se passe et le contenu du fichier résultat.

Exercice 2 — *Généralisation* On veut modéliser et résoudre le problème du sac à dos :

$$(P_2)\begin{cases} Max & z = \sum_{i=1}^{n} c_i x_i \\ s.c. & \sum_{i=1}^{n} a_i x_i \leqslant b \\ & x_i \in \{0,1\} \quad i = 1, \ldots, n \end{cases}$$

1. Écrire un fichier qui représente ce modèle. Le nom de fichier sera suffixé par `.mod` (par exemple `SacADos.mod`).

```
#modele de sac a dos
#donnees
param n ; #nombre d'objets
param C{i in 1..n}; #utilité de l'objet i
param A{i in 1..n}; #poids de l'objet i
param B; #capacité du sac

#variables
var x{1..n} binary;

#objectif
maximize f :sum {i in 1..n} C[i]*x[i] ;

#contraintes
subject to
capacite : sum{i in 1..n} A[i]*x[i] <= B ;

printf "------Debut de la resolution -----\n";
solve;

printf "------Fin de la resolution -----\n";
display x;

end;
```

2. Écrire un fichier qui contient des données pour ce modèle. Le nom de fichier sera suffixé par `.dat` (par exemple `SacADos.dat`).

```
#un fichier de donnees pour le probleme de sac a dos
data;
param n := 5;

param C := 1 12
```

```
            2 15
            3 5
            4 16
            5 17;

param A := 1 2
           2 6
           3 1
           4 7
           5 8;

param B := 20;

end;
```

3. Résoudre le programme mathématique obtenu par juxtaposition du modèle et des données avec la commande suivante :
   `glpsol -model SacADos.mod -data SacADos.dat`
4. Déterminer la valeur optimale de la relaxation continue du précédent problème.
5. L'intérêt est qu'on peut changer les données. Écrire un autre fichier de données SacADos2.dat avec par exemple $n = 10$ et des données que vous choisissez et exécutez :
   `glpsol -model SacADos.mod -data SacADos2.dat`
6. On peut également demander que le programme mathématique obtenu soit écrit dans un fichier au format lp avec la commande suivante :
   `glpsol -model SacADos.mod -data SacADos.dat -wcpxlp SacADos.lp`
   Exécutez cette commande et regardez le contenu du fichier texte SacADos.lp

Exercice 3 — *Le Sudoku* Proposez un modèle et un fichier de données pour trouver la solution de la grille de Sudoku ci-dessous. Affichez le résultat.

7			2		9			1
4	9	1		3		8	6	2
	8						9	
5			6		3			8
3	6	2				9	5	4
8			9		4			3
	5						1	
6	7	8		4		2	3	9
1			3		6			5

Exercice 4 — *Le problème p-médian* Le problème p-médian est le suivant. Étant donnés :

1. n : nombre de sites clients
2. m : nombre de sites potentiels pour des dépôts
3. p : nombre de dépôts à ouvrir
4. d_{ij} : distance entre le site client i et le site dépôt j

On veut ouvrir p dépôts et affecter chaque client au dépôt ouvert qui lui est le plus proche, de façon à minimiser la somme des distances entre les clients et les dépôts auxquels ils sont affectés.
Proposez un modèle pour ce problème et résoudre les instances présentes à l'adresse :
`http://cedric.cnam.fr/~lamberta/MPRO/ECMA/`
On montrera à l'occasion de cet exercice l'importance du choix d'une "bonne" modélisation.

2 Utilisation du solveur linéaire et quadratique convexe XPress

Avant de commencer :

1. Télécharger le solveur linéaire et quadratique XPress version étudiante à l'adresse suivante : `http://optimization.fico.com/student-version-of-fico-xpress.html`.
2. Installer le logiciel XPress et lancer le.
3. Télécharger la documentation du langage de modélisation Mosel de Xpress à l'adresse suivante :
   `http://cedric.cnam.fr/~lamberta/MPRO/ECMA/`

Exercice 5 On veut modéliser et résoudre la problème (P_1). Pour ce faire, vous pouvez créer le fichier suivant dans l'éditeur de texte de Xpress.

1. Compiler et exécuter le code suivant :

```
model Premier_Exemple
 uses "mmxprs"          ! utilise le solveur Xpress-Optimizer

 declarations
  x1, x2: mpvar              ! variables
 end-declarations

 Z:=  3*x1 + 5*x2        !fonction objectif
 x1 <=  4                !trois contraintes
 x2 <= 6
 3*x1+ 2* x2 <= 18

 maximize (Z)

 writeln("Solution optimale de cout : ", getobjval) ! affichage de Z
 writeln("valeur de x1 ", getsol(x1)) ! affichage de la valeur de x1
 writeln("valeur de x2 ", getsol(x2)) ! affichage de la valeur de x2
end-model
```

2. Il est également possible de résoudre ce problème en utilisant un tableau :

```
model Deuxième_Exemple
 uses "mmxprs"        ! utilise le solveur Xpress-Optimizer

 declarations
  x: array (1..2) of  mpvar              ! variables
 end-declarations

 Z:=  3*x(1) + 5*x(2)        !fonction objectif
 x(1) <=  4                !trois contraintes
 x(2) <= 6
 3*x(1)+ 2* x(2) <= 18

 maximize (Z)

 writeln("Solution: ", getobjval) ! affichage de la valeur de Z
 writeln("valeur de x(1) ", getsol(x(1)))
 writeln("valeur de x(2) ", getsol(x(2)))
end-model
```

Exercice 6 — *Généralisation* On veut modéliser et résoudre le problème du sac à dos (P_2).

1. Écrire un fichier qui représente ce modèle.

```
model sac_a_dos
 uses "mmxprs"          ! utilise le solveur Xpress-Optimizer

 !déclaration des paramètres, ici n
 parameters
  n = 5
 end-parameters

 declarations
  ! les donnees
  B : real;
  C : array (1..n) of real;
  A : array (1..n) of real;

  ! les variables
  x: array (1..n) of  mpvar
 end-declarations

 initializations from 'sac.dat'
  B C A
 end-initializations

 ! le modèle
 u:=  sum (i in 1..n) C(i)*x(i)        !fonction objectif
 ! les contraintes
 sum (i in 1..n) A(i)*x(i) <= B
 forall(i in 1..n) x(i) <= 1

 ! résolution du problème
 maximize (u)

 writeln("Solution: ", getobjval) ! affichage de la valeur de u
 forall (i in 1..n)
  writeln("valeur de x(",i,")= ", getsol(x(i)))
end-model
```

2. Écrire un fichier qui contient des données pour ce modèle.

```
B : 5
C : [30 20 40 10 20]
A : [1 2 4 1 1 ]
```

3. Compiler et exécuter.

Exercice 7 — *Un programme quadratique* On veut modéliser et résoudre le problème suivant :

$$(QP)\begin{cases} Max & z = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j \\ s.c. & \sum_{i=1}^{n} a_i x_i \leqslant b \\ & x_i \in \{0,1\} \quad i = 1,\ldots,n \end{cases}$$

1. Écrire un fichier qui représente ce modèle.
2. Écrire un fichier qui contient des données pour ce modèle.
3. Compiler et exécuter.
4. Résoudre les instances présentes à l'adresse :
   `http://cedric.cnam.fr/~lamberta/MPRO/ECMA/`