Feuille de travaux pratiques n° 1
Utilisation d'un solveur de Programmation Linéaire

Les solveurs de Programmation Linéaire en variables mixtes sont des outils très répandus. Une simple recherche sur internet permet d'en trouver un grand nombre aussi bien commerciaux (CPLEX, XPRESS-MP...) que libres (lp-Solve, GLPK, COIN OR...). D'ailleurs, des solveurs de Programmation Linéaire sont de plus en plus souvent implémentés dans des tableurs (Excel, Open Office 3.0), les rendant faciles d'accès et d'utilisation.

Les solveurs sont des bibliothèques de fonctions à appeler en écrivant un programme dans un langage de programmation (C, C++, ...). Cela reste encore aujourd'hui la principale manière de les utiliser quand il s'agit d'intégrer leur usage dans un logiciel, ou d'utiliser ces solveurs comme routine dans un algorithme plus complexe. Cependant, si on souhaite simplement tester une modélisation ou résoudre un programme linéaire, il est plus rapide et plus simple d'utiliser un langage de modélisation algébrique.

Dans le cadre de ces TPs, nous utiliserons essentiellement le solveur GLPK, et le langage de modélisation associé GNU MathProg. Nous utiliserons également lp-solve, puisqu'il s'agit du solveur utilisé par Open Office Calc. Par conséquent, ce dernier solveur est a priori installé par défaut dans toute distribution de linux. Ce n'est pas nécessairement le cas de GLPK, mais on l'installe facilement.

1 Open Office Calc

1.1 Informations générales

Le solveur utilisé par Open Office Calc est en lp-solve. La façon de l'utiliser est calquée sur le solveur de Excel. Par rapport à Excel, Il manque juste la possibilité d'obtenir un rapport de résolution contenant les résultats d'une analyse de sensibilité. Cependant comme ces résultats sont fournis par le solveur lp-Solve, on peut espérer les obtenir dans une prochaine mise-à-jour.

1.2 Saisir un problème et le résoudre

Nous considérons l'exemple suivant :

– Une entreprise consacre ses ressources à la fabrication de deux peintures P1 et P2 dont les prix de ventes respectifs sont 3000 € et 2000 € par millier de litres de peinture. La demande du mois prochain n'excédera pas respectivement 3000 litres pour la peinture P2 et 3500 litres pour la peinture P1. La fabrication des deux produits requiert 2 composant C1 et C2 de la manière suivante en milliers de litres :

	P1	P2	Disponibilité
C1	2	1	6
C2	1	1	4

Nous souhaitons déterminer le plan de production maximisant le profit.

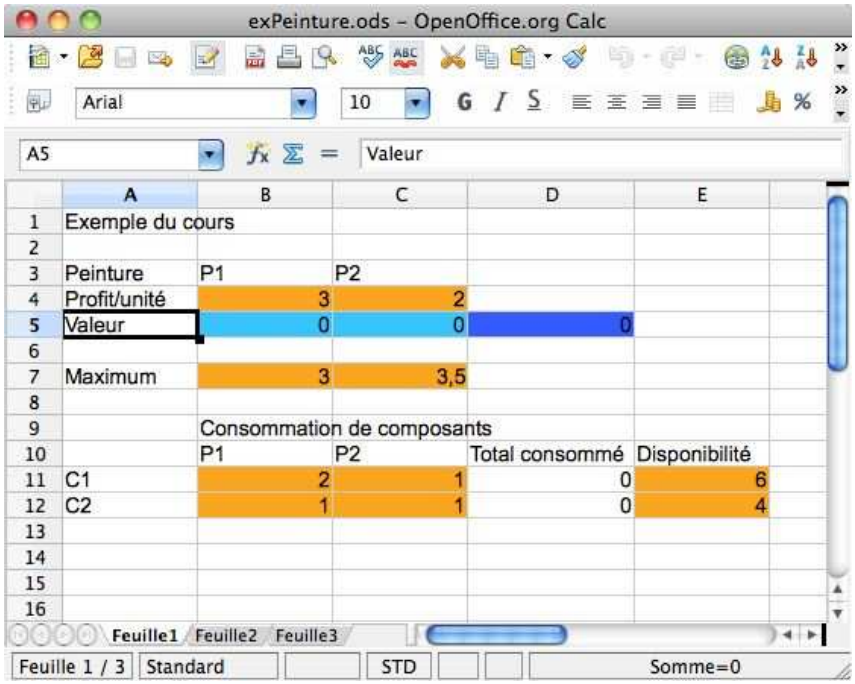
– La modélisation de ce problème s'écrit :

$$\begin{array}{rcrcl} \max\ z & = & 3x_1 + 2x_2 & & \\ & s.c. & 2x_1 + x_2 & \leq & 6 \\ & & x_1 + x_2 & \leq & 4 \\ & & x_2 & \leq & 3 \\ & & x_1 & \leq & 3,5 \\ & & x_1, x_2 & \geq & 0 \end{array}$$

où x_1, x_2 sont les variables de décision représentent respectivement les milliers de litres de peinture P1 et P2 à produire

Nous utilisons maintenant le solveur de Open Office Calc sur cet exemple. Afin de bien le suivre, il peut être utile de télécharger le fichier `exPeinture.ods` sur madoc.

Les coefficients de la fonction objectif (habituellement notés c_j), des membres de gauche (a_{ij}) et de droite des contraintes (b_i) sont saisis dans des cellules (en orange dans l'exemple). Des cellules sont ensuite désignées (pour le moment, uniquement pour nous) comme étant les variables du problème (en bleu clair dans l'exemple). Il est inutile de compléter ces cellules, elles le seront lors de la résolution. Une cellule est ensuite désignée pour représenter la fonction objectif (en bleu foncé dans l'exemple).



Cette dernière cellule doit être complétée par une formule dépendant bien sûr des variables de décision. Dans l'exemple, on a $B4*B5+C4*C5$. De même, les membres de gauche des contraintes sont définis par des formules. Par exemple, pour la consommation totale du composant C1, on a $B\$5 * B11 + C\$5 * C11$. On peut remarquer sans surprise sur cet exemple, que la disposition des cellules a une importance, non seulement pour la lisibilité de la modélisation, mais aussi afin d'avoir recours au copier-coller (par exemple pour la formule de la consommation du composant C2).

Tout est maintenant en place pour la résolution. Il reste à appeler le solveur via le menu "Outils→Solveur". Un formulaire apparaît afin de désigner la cellule-cible (la fonction objectif), les variables et les contraintes. Les options sont importantes pour spécifier le type des variables (non-négatives ici). Une fois ce formulaire complété, le résultat est obtenu en cliquant sur "Résoudre".

Les cellules variables (et les autres en dépendant) sont modifiées, permettant ainsi de lire une solution optimale et la valeur optimale.

2 GLPK - GNU MathProg

2.1 Informations générales

Nous utiliserons dans un premier temps le solveur GLPK via le langage de modélisation algèbrique qui est fourni avec celui-ci : GNU MathProg. S'il est possible de saisir des modélisations en suivant divers formats dans un fichier texte, les langages de modélisation permettent un peu plus que cela. En particulier, il est possible d'évaluer des expressions, d'exécuter des boucles. Enfin, il sera également possible d'apporter de la généricité au modèle en le déclarant implicitement sans ses données.

2.2 Utilisation de GNU MathProg

Nous considérons l'exemple suivant :

– Un géant de l'industrie pharmaceutique a le choix de produire 4 sortes de médicament pour soigner le mal de rein. La fabrication de chaque médicament engendre un profit mais produit également des déchets toxiques dans des pourcentages variables de la quantité fabriquée. Le tableau ci-dessous indique la quantité en kg de chaque déchet produit par la fabrication de 100 kg de chacun des médicaments ainsi que le profit par kg de médicament.

	M_1	M_2	M_3	M_4
T_1	20	20	10	40
T_2	10	30	20	0
T_3	20	40	30	10
Profit	15	60	4	20

Le gouvernement a limité la production des déchets toxiques : T1, T2,T3 à 21 kg, 6 kg, et 14 kg par semaine. Nous souhaitons déterminer le plan de production adopter de façon à respecter les restrictions gouvernementales tout en maximisant le profit.

– La modélisation de ce problème s'écrit :

$$\begin{array}{rcll} \max\ z & = & 15x_1 + 60x_2 + 4x_3 + 20x_4 & \\ & s.c. & 20x_1 + 20x_2 + 10x_3 + 40x_4 & \leq\ 21 \\ & & 10x_1 + 30x_2 + 20x_3 & \leq\ 6 \\ & & 20x_1 + 40x_2 + 30x_3 + 10x_4 & \leq\ 14 \\ & & x_1, x_2, x_3, x_4 & \geq\ 0 \end{array}$$

où x_i sont des variables de décision représentant les quantités (en centaines de kg) du médicament M_i à fabriquer ($i = 1, \ldots, 4$)

Une première solution pour saisir ce problème est de le faire explicitement en remplissant un fichier dont l'extension doit obligatoirement être `.mod` (par exemple `medoc1.mod`).

```
# Declaration de variables non-negatives

var x1 >= 0;
var x2 >= 0;
var x3 >= 0;
var x4 >= 0;

# Fonction objectif

maximize profit : 15*x1 + 20*x2 + 4*x3 + 20*x4;

# Contraintes

s.t. Toxine1 : 20*x1 + 20*x2 + 10*x3 + 40*x4 <= 21;
s.t. Toxine2 : 10*x1 + 30*x2 + 20*x3 <= 6;
s.t. Toxine3 : 20*x1 + 40*x2 + 30*x3 + 10*x4 <= 14;

# Resolution (qui est ajoutee en fin de fichier si on ne le precise pas)

solve;

# Affichage des resultats
display : x1, x2, x3, x4; # affichage "standard" des variables
printf : "x1 = %f, x2 = %f, x3 = %f, x4 = %f\n",x1,x2,x3,x4; # affichage au "format C"
display : 15*x1 + 20*x2 + 4*x3 + 20*x4; # affichage de la valeur optimale

end;
```

Les variables de décision sont déclarées avec le mot-clé `var`, leur type peut ensuite être spécifié (par exemple `integer`, `binary`) . On peut également préciser à la déclaration des bornes sur les variables, cela permet en particulier de préciser que les variables sont ici non-négatives. Nous pouvons ensuite déclarer la fonction objectif, nous lui donnons ici le nom `profit`. Puis les contraintes sont déclarées en étant précédée du mot-clé `subject to` ou pour faire plus court `s.t.`. Les résultats de la résolution peuvent finalement être affichés à l'aide des mots-clés `display` et `printf`.

Pour interpréter le fichier `medoc1.mod`, il faut taper dans un terminal : `glpsol --model medoc.mod`. L'utilisation de GNU MathProg apporte assez peu sur cet exemple. Heureusement, il est possible d'être plus habile, par exemple, il est possible d'utiliser un tableau pour les variables de décision, l'exemple devient :

```
# Declaration des ensembles utilises

param tailleM; # nombre de medicaments
set M := 1..tailleM; # ensemble des indices des medicaments

# Declaration de variables non-negatives sous la forme
# d'un tableau de variables indicees sur les medicaments
```

```
var x{M} >= 0;

# Fonction objectif

maximize profit : 15*x[1] + 20*x[2] + 4*x[3] + 20*x[4];

# Contraintes

s.t. Toxine1 : 20*x[1] + 20*x[2] + 10*x[3] + 40*x[4] <= 21;
s.t. Toxine2 : 10*x[1] + 30*x[2] + 20*x[3] <= 6;
s.t. Toxine3 : 20*x[1] + 40*x[2] + 30*x[3] + 10*x[4] <= 14;

# Resolution (qui est ajoutee en fin de fichier si on ne le precise pas)
solve;

# Affichage des resultats
display : x; # affichage "standard" des variables
display : 15*x[1] + 20*x[2] + 4*x[3] + 20*x[4]; # affichage de la valeur optimale

# Donnee n'apparaissant pas explicitement dans la modelisation

data;

param tailleM := 4;

end;
```

Nous remarquons dans cet exemple l'apparition d'un nouveau bloc `data;` dans lequel des données sont séparées de la modélisation, on parle alors de modélisation implicite. Un ensemble (`set`) est défini pour spécifier les indices du tableau de variables. Remarquons que les indices ne commencent pas nécessairement à 1 et pourraient même ne pas être entier, on pourrait par exemple avoir des noms de villes. Cet ensemble dépend d'un paramètre (le nombre de médicaments) qui n'est pas encore défini, cela est réalisé dans le bloc `data`. L'utilisation d'un ensemble donne l'avantage de pouvoir utiliser des boucles pour parcourir cet ensemble.

En utilisant plusieurs ensembles, nous pouvons mieux séparer les données de la modélisation.

```
# Declaration des ensembles utilises

param tailleM; # nombre de medicaments
set M := 1..tailleM; # ensemble des indices des medicaments

param tailleT; # nombre de toxines produites
set T := 1..tailleT; # ensemble des indices des toxines

param obj{M}; # coefficients de la fonction objectif
param coeffcontr{T,M}; # coefficients des membres de gauche des contraintes
param mdroite{T}; # coefficients des membres de droite

# Declaration de variables non-negatives sous la forme
# d'un tableau de variables indicees sur les medicaments

var x{M} >= 0;

# Fonction objectif

maximize profit : sum{j in M} obj[j]*x[j];

# Contraintes
```

```
s.t. Toxine{i in T} : sum{j in M} coeffcontr[i,j]*x[j] <= mdroite[i];

# Resolution (qui est ajoutee en fin de fichier si on ne le precise pas)
solve;

# Affichage des resultats
display : x; # affichage "standard" des variables
display : sum{j in M} obj[j]*x[j]; # affichage de la valeur optimale

# Donnee n'apparaissant pas explicitement dans la modelisation

data;

param tailleM := 4;

param tailleT := 3;

param obj := 1 15 2 20 3 4 4 20; # tableau [15, 20 10 40],
                                 # les indices doivent etre rappeles

param coeffcontr : 1  2  3  4 :=
                 1 20 20 10 40
                 2 10 30 20 0
                 3 20 40 30 10;

param mdroite := 1 21
                 2 6
                 3 14;

end;
```

Deux ensembles sont déclarées, `M` et `T`, correspondant respectivement aux ensembles de médicaments et de toxines. Ces deux ensembles nous permettent de déclarer sans les définir immédiatement le vecteur des coefficients de la fonction objectif (`obj`) et les coefficients des membres de gauche (`coeffcontr`) et des membres de droite (`mdroite`) des contraintes. Des boucles peuvent cette fois être utilisées pour définir la fonction objectif et les contraintes. Cela est réalisé en utilisant des accolades, et en précisant un indice parcourant un ensemble.

Nous utilisons en particulier une boucle pour réaliser des sommes sur un ensemble d'indices. Par exemple `sum{j in M}` signifie $\sum_{j \in M}$ ou encore $\sum_{j=1}^{tailleM}$. Cette écriture s'impose ici naturellement pour écrire la fonction objectif et chacune des contraintes. De plus, l'écriture de chacune des contraintes étant similaire, nous pouvons également les écrire dans une boucle. Cela est réalisé en précisant un ensemble d'indices à parcourir après le nom de la (série de) contrainte(s). Il est courant de regrouper des contraintes similaires lorsqu'on écrit une modélisation sur papier, ce regroupement peut aussi s'écrire en utilisant un langage de modélisation.

Nous pouvons enfin remarquer que les données sont ici totalement séparées de la modélisation. Il est alors préférable pour rendre le modèle réutilisable de placer les données dans un autre fichier dont l'extension est `.dat` (par exemple `medoc.dat`). Pour interpréter les deux fichiers, on doit alors taper dans un terminal : `glpsol --model medoc.mod --data medoc.dat`.