POLYTECH SORBONNE

Optimisation continue
MAIN, année 2019
Séance TP N° 1
Septembre 2019

Objectifs

- ☞ Prise en main de GLPK
- ☞ Prise en main de GMPL
- ☞ Application à la projection en L_∞

Problème [Optimisation linéaire à l'aide de GLPK]

Actuellement, il existe différents logiciels dédiés à la résolution de problèmes d'optimisation linéaires. Il faut en distinguer deux classes. La classe des logiciels libres et celles des logiciels propriétaires. Ces dernières sont, en général, payant mais des licences académiques (gratuites) sont disponibles. Cependant, il faut avoir à l'esprit le but derrière de telles initiatives. S'agissant des logiciels libres, en plus de leur gratuité, vous disposez du code source. Ce qui n'est pas le cas d'un logiciel propriétaire (quel que soit la licence). Disposer du code permet, dans certains cas, des adaptations des méthodes proposées pour résoudre des problèmes nouveaux.

Nous utiliserons, pour notre part, le logiciel libre **glpk** (GNU Linear Programming Kit). Différentes raisons motive ce choix. La première est, comme mentionné ci-dessus, l'aspect libre de ce logiciel. La seconde est qu'il propose un langage de modélisation, **gmpl**, très utile. Les langages de modélisation sont précieux, ils permettent un gain de temps et peuvent être utilisés par des non spécialistes (à condition de savoir ce qu'ils font !). Enfin, une derrière, est la diversité. Autrement dit, pour ne pas se retrouvez à utiliser le même logiciel pour tous vos cours !

Pour ce TP vous aurez besoin, entre autre, des documents suivants :

- ☞ **Documentations** : vous aurez besoin des documentations de **glpk** et **gmpl**. Vous pouvez les téléverser à l'adresse suivante : `https://www.gnu.org/software/glpk/`
- ☞ **Fichiers exemples** : `exemple_glpk.c`, `exemple_gmpl.mod` et `exemple_gmpl.dat`. Ils sont disponibles sur le site du cours.

Partie A : Prise en main de glpk

Dans cette première partie, vous utilisez le solveur [1] **glpk** pour résoudre le programme suivant :

$$\begin{array}{ll} \max & 3x_1 + 5x_2 + 7x_3 \\ s.c. & \\ & 2x_1 + 2x_2 + x_3 \leq 4 \\ & x_1 + 3x_2 + x_3 \leq 3 \\ & 4x_1 + x_2 + 2x_3 \leq 5 \\ & x_1, x_2, x_3 \geq 0 \end{array} \tag{1}$$

Voici la commande vous permettant de compiler le fichier `exemple-glpk.c` :

☞ `gcc -Wall exemple-glpk.c -lglpk -o exe`

1. Le fichier `exemple-glpk.c` montre : (i) comment modéliser un problème d'optimisation linéaire (pour plus de détails voir la documentation de glpk), (ii) les fonctions nécessaires pour récupérer des informations élémentaires sur la solution trouvée (valeur de la solution, la solution, ...).

1. Le code source est écrit en langage C. Cependant, vérifiez s'il n'existe pas sur le Web une interface de **glpk** écrite dans votre langage favori.

Modifier le fichier `exemple-glpk.c` afin de résoudre le problème d'optimisation linéaire suivant :

$$\begin{array}{ll} \max & 19x_1 + 6x_2 + 13x_3 + 52x_4 + 8x_5 \\ s.c. & \\ & 30 \leq x_1 + 3x_2 + x_3 + x_4 + x_5 \leq 100 \\ & 2 \leq 2x_1 + x_2 \leq 40 \\ & 25 \leq x_3 + 3x_4 + x_5 \leq 80 \\ & 5 \leq x_1, x_4, x_5 \leq 30 \\ & 0 \leq x_2, x_3 \leq 20 \end{array} \tag{2}$$

Partie B : Langage de modélisation gmpl

Dans cette deuxième partie, vous allez utiliser le langage de modélisation **gmpl**. Ce langage vous permettra de modéliser des problèmes d'optimisation linéaire avec des notations assez proches des notations mathématiques usuelles.

Cette seconde approche offre l'avantage d'être indépendante du langage de programmation dans lequel est écrit le solveur [2], le langage C dans le cas de **glpk**. Signalons que **gmpl** n'est utilisable qu'avec des problèmes d'optimisation linéaire. En revanche, sa syntaxe est similaire à un autre langage de modélisation (non libre) assez utilisé pour modéliser des problèmes d'optimisation assez généraux [3].

1. Lisez le contenu du ficher `exemple-gmpl.mod` et essayez d'en interpréter le contenu. Ce fichier contient le modèle mathématique, écrit en **gmpl**, d'un problème d'optimisation linéaire (lequel ?), notez l'extension `.mod` du fichier. Les valeurs des paramètres de ce modèle sont donnés dans un fichier à part. Dans le cas de l'exemple, il s'agit du fichier `exemple-gmpl.dat` (notez l'extension `.dat`). La donnée du modèle et les valeurs de ses paramètres définie un problème bien précis (on parle d'instance). Voici la commande pour le résoudre à l'aide de **glpk** :

   ☞ `glpsol -m exemple-gmpl.mod -d exemple-gmpl.dat`

2. Modifier le fichier `exemple-gmpl.mod` afin de modéliser la famille de problèmes d'optimisation linéaire suivants :

$$\begin{array}{ll} \max & \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\ s.c. & \\ & \sum_{j=1}^{n} x_{ij} = 1, \quad i \in \{1, \ldots, m\}, \\ & \sum_{i=1}^{m} x_{ij} = 1, \quad j \in \{1, \ldots, n\}, \\ & x_{ij} \geq 0, \quad i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}. \end{array} \tag{3}$$

3. Résoudre le programme pour une instance (ou plusieurs) du programme (3).
4. Dans votre modèle, changer en *binary* le type des variables puis résoudre le nouveau modèle. Que remarquez-vous ?

Partie C : Projection en norme L_∞

Nous terminons ce *court TP* [4] avec une petite modélisation du problème de la projection en norme uniforme, c'est-à-dire $\|.\|_\infty$ sur un polyèdre.

La projection en norme uniforme (en dimension finie) sur un polyèdre, sans perte de généralités, est le problème d'optimisation suivant :

$$\begin{array}{ll} \min & \|x - a\|_\infty \\ s.c. & \\ & Ax \leq b, \\ & x \geq 0, \end{array} \tag{4}$$

où

☞ a un vecteur appartenant à $\mathbb{R}^n$ donné ;

☞ A est une matrice appartenant à $\mathcal{M}_{m \times n}(\mathbb{R})$;

2. Nous devrions dire *application*.
3. Il s'agit du langage **AMPL**. Nous en reparlerons le moment venu.
4. Inutile de vous plaindre ! Nous n'avons peut-être pas le même critère de mesure !

☞ b un vecteur appartenant à $\mathbb{R}^m$.

Pour rappel :

$$\|x\|_\infty = \max\left\{|x_j| : j = 1, \ldots, n\right\}.$$

Enfin, signalons que le problème (4) est, en apparence, un problème d'optimisation non linéaire.

1. Modéliser le problème (4) en tant que problème d'optimisation linéaire.
2. Résoudre le problème (4) pour quelques instances générées aléatoirement.