TP 1

Programmation linéaire

Le but de ce TD est de résoudre des problèmes de programmation linéaire en utilisant plusieurs solveurs. Les solveurs disponibles sont Clp/CBC et la boite à outils "Optimisation" de MATLAB.

1. **Solveur MATLAB**

   Les principales fonctions disponibles dans la *toolbox* de MATLAB et qui vous seront utiles, sont les suivantes :

   • `linprog` : Programmation linéaire (Linear programming).

   ```
   [x, fval] = linprog(c,A,b,Aeq,beq,lb,ub,x0)
   ```

   résout le problème de programmmation linéaire suivant par l'algorithme du simplexe :

   $$\min_{\mathbf{x}} \mathbf{c}^\top \mathbf{x} \quad \begin{cases} A\mathbf{x} \leq \mathbf{b} \\ A_{\text{eq}}\mathbf{x} = \mathbf{b}_{\text{eq}} \\ \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \end{cases}$$

   — `lb` et `ub` sont des *vecteurs*. Il faut mettre `ub=[]` s'il n'y aucune contrainte de bornes supérieures (ou `ub[i]=Inf` si la composante `i` est sans borne supérieure).

   — En option, `x0` est une solution réalisable initiale (`x0=[]` s'il n'y a pas d'initialisation spécifique).

   — Par défault, la fonction `linprog` utilise une méthode de *points intérieurs*. Pour utiliser la méthode du simplexe (*dual-simplexe* en fait), il faut changer les options :

   ```
   options = optimset('Algorithm','dual-simplex');
   x = linprog(..., options);
   ```

   — Pour obtenir plus d'informations :

   ```
   options = optimset(...,'Display','iter');
   [x,fval, exitflag, output] = linprog(...,options);
   ```

   • `intlinprog` : Programmation linéaire en variables entières.

   ```
   [x, fval] = intlinprog(c,intcon,A,b,Aeq,beq,lb,ub)
   ```

   résout le problème de programmation linéaire en variables binaires :

   $$\min_{\mathbf{x}} \mathbf{c}^\top \mathbf{x} \quad \begin{cases} A\mathbf{x} \leq \mathbf{b} \\ A_{\text{eq}}\mathbf{x} = \mathbf{b}_{\text{eq}} \\ \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \\ \mathbf{x}(\texttt{intcon}) \text{ entiers} \geq 0. \end{cases}$$

   Dans la fonction `intlinprog`, on précise le tableau d'indices `intcon` des composantes de $\mathbf{x}$ qui sont entières.

   Pour résoudre un problème en variables *binaires*, il faut définir la contrainte entière sur les variables avec des bornes $\mathbf{lb} = \mathbf{0}$ et $\mathbf{ub} = \mathbf{1}$.

2. **Solveurs Clp/CBC** [1] de la suite *Coin-or*.

Clp (Coin-or linear programming) et CBC (Coin-or Branch and Cut) sont des packages *open-source* de la suite *Coin-or* permettant de résoudre des problèmes de programmation linéaire en variables réelles ou entières (MIP, mixed integer programming). Clp est un solveur pour variables réelles et CBC est un solveur MIP. Vous pouvez télécharger les binaires exécutables (pour windows/linux/OsX) aux adresses :

`https://www.coin-or.org/download/binary/Cbc`

Le package Cbc contient aussi le solveur Clp. La version la plus récente à ce jour est la *Cbc-2.10.5*. Des binaires pour Windows (64bits) et Linux (64bits) sont également disponibles sur ARCHE.

Pour utiliser Clp ou CBC en ligne de commande [2] (*Terminal* dans Linux, *Command Prompt* dans Windows), il faut tout d'abord décrire le problème à résoudre dans un fichier au format LP [3](voir annexe "LP Format").

• **Format LP : un exemple.** Considérons le PL suivant.

$$(P_0)\begin{cases} \max_{\mathbf{x}\in\mathbb{R}^2}\,[F(\mathbf{x}) = 1700x_1 + 3200x_2] \\ \begin{array}{rcrcl} & & 3x_2 & \leq & 39 \\ 1.5x_1 & + & 4x_2 & \leq & 60 \\ 2x_1 & + & 3x_2 & \leq & 57 \\ 3x_1 & & & \leq & 57 \end{array} \\ x_1,\, x_2 \geq 0 \end{cases}$$

Voici ce que doit contenir le fichier `mon_pb.lp` de description du problème (P_0) au format LP :

```
Maximize
      F : 1700 x1 + 3200 x2
Subject To
      C1 : 3 x2 <=  39
      C2 : 1.5 x1 + 4 x2 <= 60
      C3 : 2 x1 + 3 x2 <= 57
      C4 : 3 x1 <= 57
Generals
      x1 x2
End
```

On remarquera qu'il n'y a pas besoin de préciser la positivité des variables, elles le sont par défaut. Pour préciser des bornes sur une variable, il faut utiliser le mot clef `Bounds`. Par exemple, si on avait les contraintes de bornes $1 \leq x_1 \leq 2$, il faudrait rajouter dans le fichier `mon_pb.lp` (avant le `End`) :

```
Bounds
      1  <= x1 <= 2
```

• **Exécution de Clp/CBC.**

Une fois le fichier de description `mon_pb.lp` créé, on résout le problème en utilisant l'exécutable `cbc` avec la commande :

```
clp mon_pb.lp solve solu fichier_resultats.txt
```

Le résultat est sauvegardé dans le fichier ASCII nommé `fichier_resultats.txt`. La syntaxe pour exécuter CBC pour des variables entières, est la même.

On peut également utiliser le mode interactif en exécutant (toujours en ligne de commande), la commande `clp` (ou bien la commande `cbc`), simplement sans paramètre. On se retrouve alors dans l'environnement Clp (ou CBC) :

1. `https://github.com/coin-or/`
2. Pour plus de détails, voir `http://www.decom.ufop.br/haroldo/files/cbcCommandLine.pdf`
3. `https://www.gurobi.com/documentation/9.5/refman/lp_format.html`

```
Coin LP version 1.17.3, build Feb 17 2021
Clp takes input from arguments ( - switches to stdin)
Enter ? for list of commands or help
Clp:
```

Puis :

```
1 Clp: import mon_pb.lp
2 Clp: solve
3 Clp: solu fichier_resultats.txt
```

Remarque. Pour un problème de *minimisation*, il faut le préciser avec le mot clef/commande `min` (idem dans l'environnement Clp :) : `clp exo1.lp` **`min`** `solve solu fichier_resultats.txt`

Exercice 1. Résoudre (P_0) en utilisant :

1. `MATLAB`, après avoir écrit (P_0) sous forme matricielle.
2. `Clp` en ligne de commande avec un fichier de description au format LP.

Exercice 2. *Un problème nutritionnel*

Marc se demande combien il doit dépenser d'argent en nourriture pour subvenir à ses besoins quotidiens en énergie (2000 kcal), en protéine (55g) et en calcium (800mg). Il choisit 6 aliments qui semblent être des sources bon marché d'éléments nutritifs. Les données nutritionnelles de ces aliments sont reportées dans le tableau ci-dessous.

aliments	taille des portions	énergie (kcal)	protéine (g)	calcium (mg)	prix par portion (centimes d'€)
flocons d'avoine	28 g	110	4	2	11
poulet	100 g	205	32	12	140
oeuf	2 gros	160	13	54	100
lait entier	237 ml	160	8	285	52
tarte aux cerises	170 g	420	4	22	132
porc aux haricots	260 g	260	14	80	105

Sur les conseils d'un nutritionniste, Marc décide de limiter le nombre de portions de chacun des 6 aliments pour éviter par exemple de prendre 10 portions de porc aux haricots qui couvriraient tous ses besoins pour un prix de 10,5 €. Les limitations sont les suivantes :

flocons d'avoine	:	4 portions au plus par jour
poulet	:	3 portions au plus par jour
oeuf	:	2 portions au plus par jour
lait entier	:	8 portions au plus par jour
tarte aux cerises	:	2 portions au plus par jour
porc aux haricots	:	2 portions au plus par jour

Marc veut connaitre le nombre de portions (les quantités) de chacun des 6 aliments nécessaires pour couvrir tous ses besoins en énergie, en protéines et en calcium, en dépensant le moins d'argent possible (en centimes d'€) et tout en respectant les limitations sur les quantités mentionnées ci-dessus.

1. Modéliser ce problème sous la forme d'un programme linéaire.
2. Le résoudre en utilisant Clp-CBC.
3. Ecrire les contraintes sous la forme matricielle $A\mathbf{x} \leq \mathbf{b},\ \mathbf{0} \leq \mathbf{x} \leq \mathbf{d}$ en précisant ce que valent la matrice A et les vecteurs $\mathbf{b}$ et $\mathbf{d}$. Résoudre le problème en utilisant MATLAB.

Vous pourrez considérer les cas de portions entières et non-entières.

Exercice 3. *Stockage.*

On dispose de m disques durs sur lesquels on veut stocker n fichiers de taille $c_1, \cdots, c_n$. Les disques durs ont tous la même capacité de stockage C (en GigaOctets). Tous les fichiers doivent être stockés. Pour qu'un stockage soit admissible il faut que la somme des tailles des fichiers stockés sur un disque soit inférieure ou égale à C. De plus, un fichier n'est stocké qu'une seule fois sur un seul disque.

Déterminer un plan de stockage consiste alors à déterminer sur quels disques stocker les fichiers. On cherche à déterminer le plan de stockage optimal de façon à minimiser le nombre de disques utilisés.

Modélisation. (*Rappel, cf TD*)

On introduit les variables suivantes :

- les variables binaires y_j pour indiquer si le disque j est utilisé ($y_j = 1$) ou non ($y_j = 0$).
- les variables binaires x_{ij} pour indiquer si le fichier i est stocké sur le disque j ($x_{ij} = 1$) ou non ($x_{ij} = 0$).

Le problème du stockage optimal se modélise alors par le programme linéaire suivant :

$$(P)\begin{cases} \min \displaystyle\sum_{j=1}^{m} y_j \\ \displaystyle\sum_{i=1}^{n} c_i x_{ij} \leq C y_j, \quad \forall j = 1, \cdots, m \\ \displaystyle\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i = 1, \cdots, n \\ y_j \in \{0,1\}, \quad \forall j = 1, \cdots, m \\ x_{ij} \in \{0,1\}, \quad \forall i = 1, \cdots, n \end{cases}$$

Forme matricielle. Les variables x_{ij} et y_j sont regroupées dans un seul vecteur $\mathbf{x}$ de $(n+1)m$ composantes. On choisit un regroupement *colonne par colonne* :

$$\mathbf{x} = (x_{11}, x_{21}, \cdots, x_{n1} \,|\, x_{12}, x_{22}, \cdots, x_{n2} \,|\, \cdots \,|\, x_{1m}, \cdots, x_{nm} \,|\, y_1, \cdots, y_m)^\top$$

Le problème (P) s'écrit matriciellement sous la forme suivante

$$(PL)\begin{cases} \min\limits_{\mathbf{x}} F(\mathbf{x}) = \mathbf{p}^\top \mathbf{x} \\ A\mathbf{x} \leq \mathbf{b} \\ B\mathbf{x} = \mathbf{d} \\ \mathbf{x} \in \{0,\, 1\} \end{cases}$$

- La matrice A est de taille $m \times (n+1)m$ (c'est-à-dire m lignes et $(n+1)m$ colonnes) et elle a la structure suivante :

$$A = \left(\begin{array}{ccc|ccc|c|ccc|ccc} c_1 & \cdots & c_n & & 0 & & & & 0 & & -C & & 0 \\ & 0 & & c_1 & \cdots & c_n & & & 0 & & & -C & \\ & & & & & & \ddots & & & & & & \ddots \\ & 0 & & & 0 & & & c_1 & \cdots & c_n & 0 & & -C \end{array}\right)$$

  Le dernier bloc diagonal avec les $-C$ est de taille $m \times m$. Les autres blocs (m blocs) contiennent le vecteur $\mathbf{c} = (c_1, \cdots, c_n)$ et sont tous de taille $m \times n$.

- La matrice B est de taille $n \times (n+1)m$ (c'est-à-dire n lignes et $(n+1)m$ colonnes) et elle a la structure suivante :

$$B = \left(\begin{array}{ccc|ccc|c|ccc|c} 1 & & & 1 & & & & 1 & & & \\ & \ddots & & & \ddots & & \cdots & & \ddots & & 0 \\ & & 1 & & & 1 & & & & 1 & \end{array}\right)$$

  La matrice B est composée de m blocs de matrice identité de taille $n \times n$ et d'un bloc de taille $n \times m$ ne contenant que des 0.

— Les vecteurs $\mathbf{p}$, $\mathbf{b}$ et $\mathbf{d}$ sont donnés par

$$\begin{aligned}\mathbf{b} &= (0, \cdots, 0)^\top \in \mathbb{R}^m \\ \mathbf{d} &= (1, \cdots, 1)^\top \in \mathbb{R}^n \\ \mathbf{p} &= (\underbrace{0, \cdots, 0}_{nm}, \underbrace{1, \cdots, 1}_{m}) \in \mathbb{R}^{(n+1)m}\end{aligned}$$

Travail demandé.

1. Ecrire un script MATLAB pour construire les matrices A et B à partir du vecteur des tailles des fichiers
$$\mathbf{c} = (c_1,\, c_2,\, \cdots, c_n)^\top .$$
2. Compléter votre script pour résoudre (PL) en utilisant la fonction `intlinprog`.
3. Une fois le vecteur solution $\mathbf{x}$ obtenu, on veut retrouver les numéros des fichiers stockés sur un disque donné. Plus précisément, si on note les variables
$$(z_1, \cdots, z_k, \cdots, z_N) = (x_{11}, x_{21}, \cdots, x_{n1} \,|\, x_{12}, x_{22}, \cdots, x_{n2} \,|\, \cdots \,|\, x_{1m}, \cdots, x_{nm})$$
avec $N = nm$, on veut trouver (i, j) tel que $z_k = x_{ij}$. On connait k (avec $z_k = 1$) et on veut retrouver le couple (i, j). Utiliser les fonctions `reshape` et `find` pour retrouver (i, j) à partir de k tel que $z_k = 1$.
4. Tester les différentes parties du script avec les données suivantes : $n = 3$, $m = 4$, $C = 4$ et $\mathbf{c} = [1,\, 2,\, 3]$.
5. Résoudre (P) avec les données numériques suivantes : $n = 11$, $m = 4$ et $C = 8$.
Les tailles des fichiers sont $\mathbf{c} = (1,\, 3,\, 2,\, 1,\, 2,\, 4,\, 2,\, 3,\, 5,\, 2,\, 2)^\top$.
Utiliser la fonction `printkey` qui vous est fournie[4] pour afficher la distribution trouvée des fichiers sur les disques durs.
6. *Méthode heuristique.* Une alternative possible à (PL) est de ne pas chercher une solution optimale mais de se contenter d'une solution "proche". Une méthode consiste à stocker les fichiers par taille décroissante : on commence par stocker le plus gros fichier, puis le deuxième plus gros et ensuite de suite ... On rajoute un disque dur au fur et à mesure lorsqu'il n'y a plus de place sur les disques déjà utilisés.
Ecrire un script MATLAB correspondant à cette méthode. Comparer avec la solution optimale trouver précédemment.

Exercice 4. *Voyageur de commerce.*

La modélisation du problème du voyageur de commerce sous forme de programme linéaire fait intervenir les variables

$$x_{ij} = \begin{cases} 1 & \text{si on va de la ville } i \text{ à la ville } j \\ 0 & \text{sinon.} \end{cases}$$

ainsi que les variables u_i pour $1 \leq i \leq n$, représentant la position de la ville i dans la tournée du voyageur. On suppose que la ville 1 est la ville de départ, on a donc $u_1 = 1$ et par ailleurs $u_i \in \{2, \cdots, n\}$ pour $i \in \{2, \cdots, n\}$. Le problème s'écrit sous la forme suivante (Miller-Tucker-Zemlin, 1960) :

$$\begin{aligned}&\min \sum_{i,j=1}^{n} d_{ij} x_{ij} \\ &\sum_{j=1}^{n} x_{ij} = 1, \ \forall i \in \{1, \cdots, n\} \\ &\sum_{i=1}^{n} x_{ij} = 1, \ \forall j \in \{1, \cdots, n\} \\ &u_i - u_j + 1 \leq (n-1)(1 - x_{ij}), \quad \forall i, j, \ i \neq 1, j \neq 1 \\ &x_{ij} \in \{0, 1\}, \quad 1 \leq i, j \leq n, \qquad u_i \in \{2, \cdots, n\}, \quad 2 \leq i \leq n.\end{aligned}$$

4. à récupérer sur Arche

On pose $\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{u} \end{pmatrix}$ avec $\mathbf{x} = (x_{11}, x_{12}, \cdots, x_{1n} \,|\, x_{21}, x_{22}, \cdots, x_{2n} \,|\, \cdots \,|\, x_{n1}, \cdots, x_{nn})^\top$ et $\mathbf{u} = (u_2, \cdots, u_n)^\top$.
Le problème (P) s'écrit matriciellement sous la forme suivante

$$(PL) \begin{cases} \min\limits_{\mathbf{z}} F(\mathbf{z}) = \mathbf{d}^\top \mathbf{z} \\ A\mathbf{z} = \mathbf{b} \\ B\mathbf{z} \leq \mathbf{c} \\ \mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{u} \end{pmatrix}, \ \mathbf{x} \in \{0,\, 1\}^{n^2}, \mathbf{u} \in \{2, \cdots, n\}^{n-1} \end{cases}$$

— La matrice A est de taille $(2n) \times (n^2 + n - 1)$:

$$A = \left(\begin{array}{ccc|ccc|c|ccc|c} 1 & \dots & 1 & 0 & \cdots & 0 & & & 0 & & 0_{n,n-1} \\ & & & 1 & \cdots & 1 & & & & & \\ & 0 & & & 0 & & \ddots & & & & \\ & & & & & & & 1 & \cdots & 1 & \\ \hline 1 & & & 1 & & & & 1 & & & \\ & \ddots & & & \ddots & & \cdots & & \ddots & & 0_{n,n-1} \\ & & 1 & & & 1 & & & & 1 & \end{array}\right)$$

où $0_{n,n-1}$ est la matrice de taille $n \times (n-1)$ composée de 0.

— La matrice B est de taille $(n-1)^2 \times (n^2 + n - 1)$ c'est-à-dire $(n-1)^2$ lignes et $n^2 + n - 1$ colonnes :

$$B = \left(\begin{array}{c|c|c|c|c|c} 0_{n-1,n} & D & 0_{n-1,n} & \cdots & 0_{n-1,n} & T_1 \\ \hline 0_{n-1,n} & 0_{n-1,n} & D & \cdots & 0_{n-1,n} & T_2 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline 0_{n-1,n} & 0_{n-1,n} & 0_{n-1,n} & \cdots & D & T_{n-1} \end{array}\right)$$

où $0_{n-1,n}$ désigne la matrice de taille $(n-1) \times n$ composée de 0. La matrice D est aussi de taille $(n-1) \times n$ et les T_k sont des matrices carrées de taille $(n-1) \times (n-1)$ avec

$$D = \begin{pmatrix} 0 & (n-1) & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & (n-1) \end{pmatrix}, \quad T_k = \begin{pmatrix} -1 & & & 1 & & & \\ & \ddots & & \vdots & & 0 & \\ & & -1 & 1 & & & \\ & & & 0 & & & \\ & & & 1 & -1 & & \\ & 0 & & \vdots & & \ddots & \\ & & & 1 & & & -1 \end{pmatrix}$$

$$\uparrow$$
colonne k

— Les vecteurs $\mathbf{b}$, $\mathbf{c}$ et $\mathbf{d}$ sont donnés par

$$\mathbf{b} = (1, \cdots, 1)^\top \in \mathbb{R}^{2n}$$
$$\mathbf{c} = (n-2, \cdots, n-2)^\top \in \mathbb{R}^{(n-1)^2}$$
$$\mathbf{d} = \big(d_{11}, d_{12}, \cdots, d_{1n} \,|\, \cdots \,|\, d_{n1}, \cdots, d_{nn} \,|\, \underbrace{0, \cdots, 0}_{n-1}\big)^\top \in \mathbb{R}^{n^2+n-1}$$

Travail demandé.

1. Construire les matrices A et B en MATLAB.
2. Ecrire un script MATLAB pour résoudre le programme linéaire (PL).
3. Tester votre code avec la donnée des distances :

$$\mathcal{D} = (d_{ij}) = \begin{pmatrix} 0 & 9 & 3 & 10 & 4 \\ 10 & 0 & 21 & 9 & 8 \\ 7 & 40 & 0 & 8 & 5 \\ 9 & 4 & 3 & 0 & 5 \\ 10 & 11 & 6 & 10 & 0 \end{pmatrix}$$

Pour cette donnée, la distance minimale vaut 32 avec deux parcours optimaux : $(1, 5, 3, 4, 2, 1)$ et $(1, 3, 5, 4, 2, 1)$.

Tester ensuite votre code en engendrant aléatoirement le vecteur $\mathbf{d}$ des distances (valeurs entières avec la fonction `randi`). Tester avec $n = 10$, puis $n = 20$.