

## TP : Solveurs de programmes linéaires

Il existe beaucoup de solveurs de programmes linéaires. L'un des plus connus (car plus performant) est peut-être CPLEX, développé par Robert E. Bixby de l'université de Rice à Houston au Texas, en 1988. CPLEX est racheté en 1997 par l'entreprise française ILOG puis à son tour racheté en 2009 par IBM (mais nous n'avons pas la licence). R. E. Bixby a depuis créé un autre solveur commercial, Gurobi, en 2008 (Gurobi vient des noms de famille des fondateurs : Gu, Rothberg, Bixby).

Nous allons utiliser plusieurs autres logiciels dans ce TP.

### Exercice 1 (Utilisation du tableur Calc de LibreOffice).

Le tableur Calc intègre un solveur de programmes linéaires, `lp_solve`. Utilisons-le sur l'exemple des courgettes et des navets du TD1, en rentrant les données suivantes :

	A	B	C	D	E
1		courgettes	navets		
2					
3	profit	4	5	$=B3*B\$2+C3*C\$2$	
4	engrais A	2	1	$=B4*B\$2+C4*C\$2$	8
5	engrais B	1	2	$=B5*B\$2+C5*C\$2$	7
6	Anti-P		1	$=B6*B\$2+C6*C\$2$	3

Le but est d'obtenir les valeurs optimales des variables dans les cases B2 et C2. Le profit optimal s'affichera dans la case D3.

Pour lancer le solveur, il faut aller dans le menu **Outils** → **Solveur**, entrer la case correspondant à la fonction objectif dans le champ "cellule cible" (ici c'est  $D\$3$ ), les cases contenant les variables dans le champ "par modification des cellules" ( $B\$2$  et  $C\$2$ ), et spécifier les contraintes dans les champs "conditions de limitation" (par exemple la première contrainte sera  $D\$4 \leq E\$4$ ).

Jouer avec le solveur.

*Un solveur est également disponible dans Excel, mais il faut d'abord installer l'add-in gratuit "solveur". Le fonctionnement est similaire, le solveur est accessible dans le menu **Données** → **Analyse**. On peut également y produire un "rapport de sensibilité", pour évaluer l'impact de changements de coefficients dans les contraintes, comme le "coût marginal" d'une contrainte.*

### Exercice 2 (Utilisation de GLPK).

Le GNU Linear Programming Kit (GLPK), dont la première version fut développée par Andrew O. Makhorin du Moscow Aviation Institute en 2000, est une suite logicielle pour la programmation linéaire sous GNU. On peut décrire un programme linéaire sous divers formats standards, par exemple, le format `lp`.

Voici le contenu d'un fichier `test.lp` :

```

Maximize
    nom_objectif: x1 + 2 x2 - x3
Subject To
    nom_contrainte_1: x1 + 3 x2 <= 12
    nom_contrainte_2: - x1 + 4 x3 >= 40
Bounds
    x2 <= 10
End

```

qui représente le programme linéaire suivant (noter que les variables sont non-négatives par défaut) :

maximiser	$x_1$	+	$2x_2$	-	$x_3$	
tel que	$x_1$	+	$3x_2$			$\leq 12$
	$-x_1$			+	$4x_3$	$\geq 40$
			$x_2$			$\leq 10$
	$x_1$					$\geq 0$
			$x_2$			$\geq 0$
					$x_3$	$\geq 0$

On peut ensuite exécuter le solveur avec la commande : `glpsol --lp test.lp -o solution.txt`, qui écrit la solution et un rapport dans le fichier `solution.txt`. La valeur optimale de la fonction objectif est dans la ligne “objective”. Les valeurs des variables sont dans la colonne “activity” du deuxième tableau. (L’option `--ranges sensitivity.txt` écrit un rapport de sensibilité dans le fichier `sensitivity.txt`.)

A vous de jouer :

- Résoudre le problème du régime alimentaire à 5 variables présenté dans le cours magistral (voir ci-dessous).
- Jouer avec les contraintes et les prix des denrées, pour obtenir un régime qui vous paraît intéressant et/ou cohérent. Observer l’influence des différentes contraintes sur la solution.

aliment	prix (€/kg)	protéines (g/kg)	vitamine C (mg/kg)	fer (mg/kg)
Ananas	3.1	5	478	3
Banane	2.1	10	70	12
Carotte	1.6	7.8	20	2.4
Datte	8.7	25	4	10
Endive	3.8	13	65	8

Soient  $a, b, c, d, e$  les quantités d’ananas, bananes, carottes, dattes, endives (en kg). On obtient le programme linéaire suivant :

$$\begin{array}{ll}
 \text{minimiser :} & 3.1a + 2.1b + 1.6c + 8.7d + 3.8e \\
 \text{tel que :} & \begin{array}{l}
 5a + 10b + 7.8c + 25d + 13e \geq 56 \\
 478a + 70b + 20c + 4d + 65e \geq 110 \\
 3a + 12b + 2.4c + 10d + 8e \geq 2 \\
 a \geq 0 \\
 b \geq 0 \\
 c \geq 0 \\
 d \geq 0 \\
 e \geq 0
 \end{array}
 \end{array}$$

### Exercice 3 (Utilisation de Sagemath).

Sagemath est très pratique pour manipuler des graphes, et s'interface avec des solveurs de PL. Pour le lancer, écrire la commande `sage`, puis, dans la console de sagemath, écrire `notebook()`. Cela lance une interface (basée sur `jupyter notebook`) dans votre navigateur. Vous pouvez ensuite créer une nouvelle feuille de calcul en cliquant sur `New worksheet`. Ensuite on code en Python.

On peut définir et résoudre un PL dans Sagemath de la façon suivante :

```
p = MixedIntegerLinearProgram()           #on initialise un PL
V = p.new_variable(real=True, nonnegative=True) #V est un dictionnaire qui va contenir les
                                              variables (toutes positives ou nulles) du PL
x, y, z = V["x"], V["y"], V["z"]          #on initialise des variables dans le
                                              dictionnaire. C'est optionnel, on peut aussi
                                              manipuler directement V[0], V["toto"] etc.

p.set_objective( x + y + 3*z )             #fonction objectif
#p.set_objective( V["x"] + V["y"] + 3*V["z"] ) #version équivalente
p.add_constraint( x + 2*y <= 4 )           #contrainte 1
p.add_constraint( 5*z - y == 8 )          #contrainte 2
p.show()                                  #affiche toutes les variables et
                                              contraintes du PL

opt = p.solve()                           #résoudre le PL
print(opt, p.get_values(V))               #afficher la valeur optimale du PL et les
                                              valeurs des variables
```

Cela correspond au programme linéaire suivant :

maximiser	$x$	+	$y$	+	$3z$	
tel que	$x$	+	$2y$			$\leq 4$
			$-y$	+	$5z$	$= 8$
	$x$					$\geq 0$
			$y$			$\geq 0$
					$z$	$\geq 0$

On peut aussi faire une somme sur une liste  $L$  de variables du PL  $p$  (dans une contrainte ou dans la fonction objectif) de la façon suivante :

```
p.set_objective ( p.sum(x for x in L) )
```

Beaucoup de fonctions de base sur les graphes sont implémentées dans Sagemath (voir <https://doc.sagemath.org/html/en/reference/graphs/index.html>). Il existe aussi beaucoup d'algorithmes avancés qui y sont implémentés. (D'ailleurs, souvent, la méthode utilisée est la PL.)

Pour créer et manipuler un graphe étiqueté dans Sagemath on peut écrire :<sup>1</sup>

```
G = DiGraph(3)           #Crée un graphe orienté à 3 sommets (nommés 0,1,2 par défaut)
G.add_vertex(name="toto") #ajoute un sommet nommé "toto"
G.add_edge(1,2,6)         #Ajoute l'arc 1->2 avec l'étiquette (label) "6"
for e in G.edges():       #boucle sur la liste des arcs de G
    print(e[0],e[1],e[2]) #affiche l'origine, la destination, et l'étiquette de l'arc e
for v in G.vertices():    #boucle sur la liste des sommets de G
    print(G.neighbors_out(v)) #affiche la liste des voisins sortants de v
    print(G.neighbors_in(v)) #affiche la liste des voisins entrants de v
    print(G.neighbors(v))    #affiche la liste de tous les voisins de v
```

1. Voir <https://doc.sagemath.org/html/en/reference/graphs/index.html> pour plus de fonctions, si nécessaire.

```

print(G.incoming_edges(v)) #affiche la liste des arcs entrant dans v
print(G.outgoing_edges(v)) #affiche la liste des arcs sortant de v
G.show(edge_labels=True)   #dessine le graphe en affichant les étiquettes

```

À vous de jouer :

- Dans Sagemath, créer le graphe orienté étiqueté qui correspond au réseau de la figure 1.
- Créer le programme linéaire qui résout le problème de flot pour aller de  $s$  à  $d$  (méthode vue en cours). Résoudre le PL et afficher la solution. Pour manipuler une variable correspondant à un arc  $a$  entre "sommet1" et "sommet2" dans le dictionnaire de variables  $V$ , vous pouvez par exemple écrire directement :  
`V[("sommet1","sommet2")]`  
ou  
`V["sommet1sommet2"]`  
ou encore :  
`V[a]`  
vous pouvez en effet choisir librement les clés du dictionnaire de variables, à vous de voir ce qui est plus pratique pour vous.
- Faites de même pour le graphe de la figure 2, pour un flot qui va d'Elphy à Santenago.
- Écrire une fonction générique `def flot(G,s,t)` qui prend en entrée un graphe étiqueté  $G$  (les étiquettes sont les capacités initiales), le sommet source  $s$  et le sommet destination  $t$ . Dans cette fonction, on définit et on résout le PL associé au problème du flot dans  $G$ . La fonction renvoie la solution optimale sous forme d'un couple (graphe étiqueté, valeur optimale du flot), où les étiquettes des arcs représentent les valeurs de flot pour ces arcs.

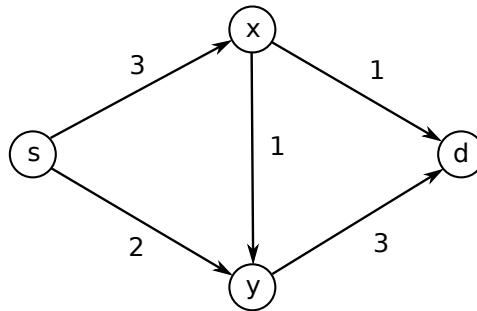


FIGURE 1 – Un petit graphe

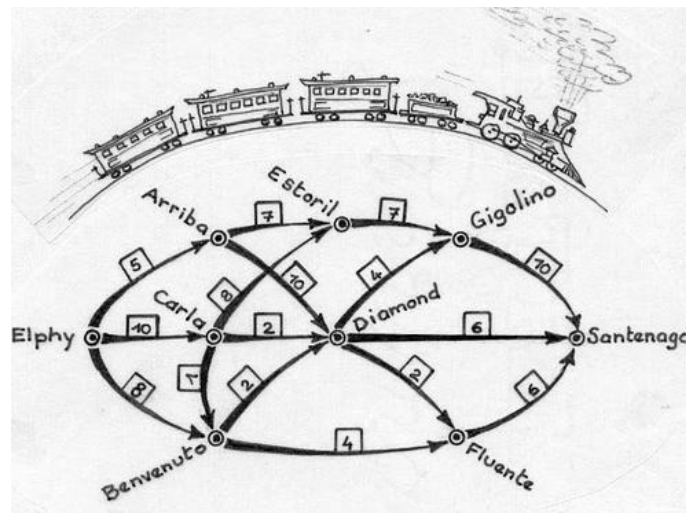


FIGURE 2 – Un graphe moyen