Recherche Opérationnelle (TEOST)

Chapitre 1 : Optimisation linéaire

J.-F. Scheid
Institut Elie Cartan de Lorraine/Télécom Nancy
Université de Lorraine

2026-27 (version du 21/09/26)

Table des matières

1. Notations et rappels d'algèbre linéaire
2. Introduction
3. Méthode du simplexe
4. Dualité
   - Introduction et définitions
   - Théorèmes de dualité
   - Théorème des écarts complémentaires
   - Méthodes *primal-dual*
   - Méthode du *simplexe dual*
5. Quelques solveurs de PL

I) Notations et rappels d'algèbre linéaire

Quelques rappels sur les vecteurs et matrices.

On note $\mathrm{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ un vecteur (en colonne) de $\mathbb{R}^n$.

• On fait la distinction entre vecteur *ligne* et vecteur *colonne*. Les vecteurs sont tous des vecteurs *colonne* par défaut. Le symbole de transposition $\top$ permet d'obtenir un vecteur ligne : $\mathrm{x}^\top = (x_1, \cdots, x_n)$.

• *Produit scalaire* entre 2 vecteurs x et y :

$$\mathrm{x}^T \mathrm{y} = x_1 y_1 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

On notera parfois $(\mathrm{x} \,|\, \mathrm{y}) := \mathrm{x}^\top \mathrm{y}$

- Matrice A de taille (m, n) ;

$$A = (a_{ij})_{\substack{1 \le i \le m \\ 1 \le j \le n}} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

- *Produit matrice x vecteur*. Soit $\mathrm{x} \in \mathbb{R}^n$. Le vecteur $\mathrm{y} = A\mathrm{x} \in \mathbb{R}^m$ est donné par

$$y_i = (A\mathrm{x})_i = (a_{i1}, \cdots, a_{in}) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \sum_{j=1}^{n} a_{ij} x_i$$

- *Produit matrice x matrice*. Soit B une matrice de taille (n, p). La matrice $C = AB$ de taille (m, p) est donnée par

$$c_{ij} = (a_{i1}, \cdots, a_{in}) \begin{pmatrix} b_{1j} \\ \vdots \\ b_{nj} \end{pmatrix} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

- *Transposition d'un produit*. On a $(AB)^\top = B^\top A^\top$.

• *Système linéaire et inéquations linéaires*. Soit $\mathsf{b} \in \mathbb{R}^m$. On a

$$A\mathsf{x} = \mathsf{b} \iff \begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n & = & b_1 \\ \vdots & \vdots & \vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n & = & b_m \end{cases} \tag{1}$$

Un vecteur $\mathsf{x} \in \mathbb{R}^n$ vérifiant (1) est dit *solution du système linéaire*.

On écrira l'inéquation vectorielle

$$A\mathsf{x} \leq \mathsf{b} \tag{2}$$

si $(A\mathsf{x})_i \leq b_i$ pour tout i.

• *Matrice inversible*. Soit A une matrice carrée de taille (n, n).
A est dite *inversible* s'il existe une matrice notée A^{-1} telle que $AA^{-1} = A^{-1}A = I_n$ où I_n désigne la matrice identité (des 1 sur la diagonale, 0 partout ailleurs).

A est inversible $\iff$ les colonnes de A sont linéairement indépendantes

$\iff$ le système linéaire $A\mathsf{x} = b$ admet une unique solution x donnée par $\mathsf{x} = A^{-1}\mathsf{b}$.

• *Produit par blocs.* Soit A de taille (m, n). On décompose en *blocs horizontaux* la matrice

$$A = (A_1 \ A_2)$$

où A_1 est de taille (m, n_1) et A_2 de taille (m, n_2) (même nombre de lignes que A) avec $n = n_1 + n_2$. On a

$$Ax = A_1x_1 + A_2x_2 \tag{3}$$

pour $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^n$ et $x_1 \in \mathbb{R}^{n_1}$, $x_2 \in \mathbb{R}^{n_2}$.

I) Introduction

Soient deux vecteurs $\mathrm{b} = (b_1, \cdots, b_m)^\top \in \mathbb{R}^m$, $\mathrm{c} = (c_1, \cdots, c_n)^\top \in \mathbb{R}^n$ et une matrice rectangulaire A de taille (m, n).

On considère le problème de programmation linéaire suivant (encore appelé programme linéaire et en abrégé PL) :

$$\max_{\mathrm{x} \in \mathbb{R}^n} F(\mathrm{x}) = \mathrm{c}^\top \mathrm{x} = c_1 x_1 + \cdots + c_n x_n$$
$$\text{sous les contraintes}$$
$$\begin{cases} A\mathrm{x} \leq \mathrm{b} \\ \mathrm{x} \geq 0 \end{cases}$$

Contraintes *inégalités* $A\mathrm{x} \leq \mathrm{b}$: PL sous **forme canonique pure**.

Contraintes *égalités* $A\mathrm{x} = \mathrm{b}$: PL est sous **forme standard**.

Proposition

Tout PL sous forme standard s'écrit de façon équivalent en un PL sous forme canonique pure et inversement.

1. Soit un PL sous forme canonique pure, montrons qu'il peut s'écrire en un PL sous forme standard. On a

$$A\mathrm{x} \leq \mathrm{b} \Leftrightarrow A\mathrm{x} + \mathrm{e} = \mathrm{b},\ \mathrm{e} \geq 0$$

où $\mathrm{e} = (e_1, \cdots, e_m)^\top \in \mathbb{R}^m$ sont appelées les **variables d'écart**. Ainsi,

$$\begin{cases} A\mathrm{x} \leq \mathrm{b} \\ \mathrm{x} \geq 0 \end{cases} \Leftrightarrow \begin{cases} \tilde{A}\tilde{\mathrm{x}} = \mathrm{b} \\ \tilde{\mathrm{x}} \geq 0 \end{cases}$$

avec la matrice $\tilde{A} = (A \mid I_m)$ de taille $(m, n+m)$, I_m désignant la matrice identité d'ordre m et $\tilde{\mathrm{x}} = \begin{pmatrix} \mathrm{x} \\ \mathrm{e} \end{pmatrix} \in \mathbb{R}^{n+m}$.

2. Pour la réciproque, il suffit d'écrire l'égalité (dans les contraintes du problème sous forme standard) comme 2 inégalités $\leq$ et $\geq$.

Exemple. On considère le PL du problème de production donné dans l'Introduction générale. Sous forme canonique pure, il s'écrit

$$\max_{(x_1,x_2)} \left[F(x_1,x_2) = 6x_1 + 4x_2\right]. \quad \begin{cases} 3x_1 + 9x_2 \leq 81 \\ 4x_1 + 5x_2 \leq 55 \\ 2x_1 + x_2 \leq 20 \\ x_1, x_2 \geq 0 \end{cases} \tag{4}$$

Ecrivons-le sous forme standard en introduisant 3 variables d'écarts e_1, e_2 et e_3 :

$$\max_{(x_1,x_2,e_1,e_2,e_3)} \left[F(x_1,x_2,e_1,e_2,e_3) = 6x_1 + 4x_2\right]. \quad \begin{cases} 3x_1 + 9x_2 + e_1 = 81 \\ 4x_1 + 5x_2 + e_2 = 55 \\ 2x_1 + x_2 + e_3 = 20 \\ x_1, x_2, e_1, e_2, e_3 \geq 0 \end{cases} \tag{5}$$

Remarque. La positivité des variables assure l'équivalence des deux formes. En fait, on peut toujours se ramener au cas de variables positives $x \geq 0$:

- Si une variable x_i a une borne inférieure $l \leq x_i$, on introduit une nouvelle variable $y_i = x_i - l \geq 0$.
- S'il n'y a pas de borne inférieure sur x_i, on pose $x_i = y_i - z_i$ avec deux nouvelles variables $y_i \geq 0$, $z_i \geq 0$.

III) Méthode du simplexe (Dantzig 1947)

On considère un PL sous *forme standard*

$$\max_{x \in \mathbb{R}^n} F(x) = c^\top x = c_1 x_1 + \cdots c_n x_n$$
$$\begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

où A est une matrice de taille (m, n).

Hypothèse de rang plein [1] : On suppose que $\text{rang}(A) = m \leq n$.

Sous cette hypothèse, le système $Ax = b$ admet toujours des solutions [2].
L'hypothèse de rang plein n'est pas restrictive car si $\text{rang}(A) < m$ le système $Ax = b$ n'a pas de solution *en général*.

---

1. Le rang de A est le nombre maximal de lignes de A linéairement indépendantes (= nombre de colonnes de A linéairement indépendantes).
2. si $m < n$, le système $Ax = b$ admet une infinité de solutions ; si $m = n$, la matrice A est inversible (solution unique $x = A^{-1}b$, il n'y a rien à maximiser).

On note

$$\mathcal{D}_R = \{\mathrm{x} \in \mathbb{R}^n,\ A\mathrm{x} = \mathrm{b},\ \mathrm{x} \geq 0\} \tag{6}$$

l'ensemble des **solutions réalisables**. L'ensemble $\mathcal{D}_R$ est un polyèdre [3] convexe, fermé.

Définition d'une solution de base

Soit $B \subset \{1, \cdots, n\}$ un ensemble d'indices avec $\mathrm{card}(B) = m$ tel que la matrice carrée A_B formée des colonnes A^j, $j \in B$, est **inversible**.

- On dit que l'ensemble B est une **base** et que
  - les variables $\mathrm{x}_B = (x_j,\ j \in B)$ sont les **variables de base**.
  - les variables $\mathrm{x}_H = (x_j,\ j \notin B)$ sont les **variables hors-base**.

  On notera $H = \{j \in \{1, \cdots, n\}, j \notin B\}$ l'ensemble des indices correspondants aux variables hors-base.
- On dit que $\mathrm{x} = \begin{pmatrix} \mathrm{x}_B \\ \mathrm{x}_H \end{pmatrix}$ est **solution de base** associée à la base B si $Ax = b$ et $\mathrm{x}_H = 0$.

3. Un polyèdre Q de $\mathbb{R}^n$ est défini par $Q = \{\mathrm{x} \in \mathbb{R}^n,\ M\mathrm{x} \leq \mathrm{b}\}$ où M est une matrice de taille (m, n).

A une renumérotation près (permutation) des colonnes de A, on peut toujours écrire les décompositions par blocs :

$$A = (A_B|\ A_H)\,, \quad x = \begin{pmatrix} x_B \\ x_H \end{pmatrix}$$

où A_H est la matrice formée des colonnes $A^j,\ j \in H$.

Le système $Ax = b$ est équivalent à

$$A_B x_B + A_H x_H = b.$$

et on obtient

$$\boxed{x_B = A_B^{-1}(b - A_H x_H)} \tag{7}$$

Solution de base réalisable

Si $x \in \mathcal{D}_R$ est une *solution de base réalisable* alors $x_H = 0$ et x_B est déterminé par $x_B = A_B^{-1} b \geq 0$.

Caractérisation de l'optimum.

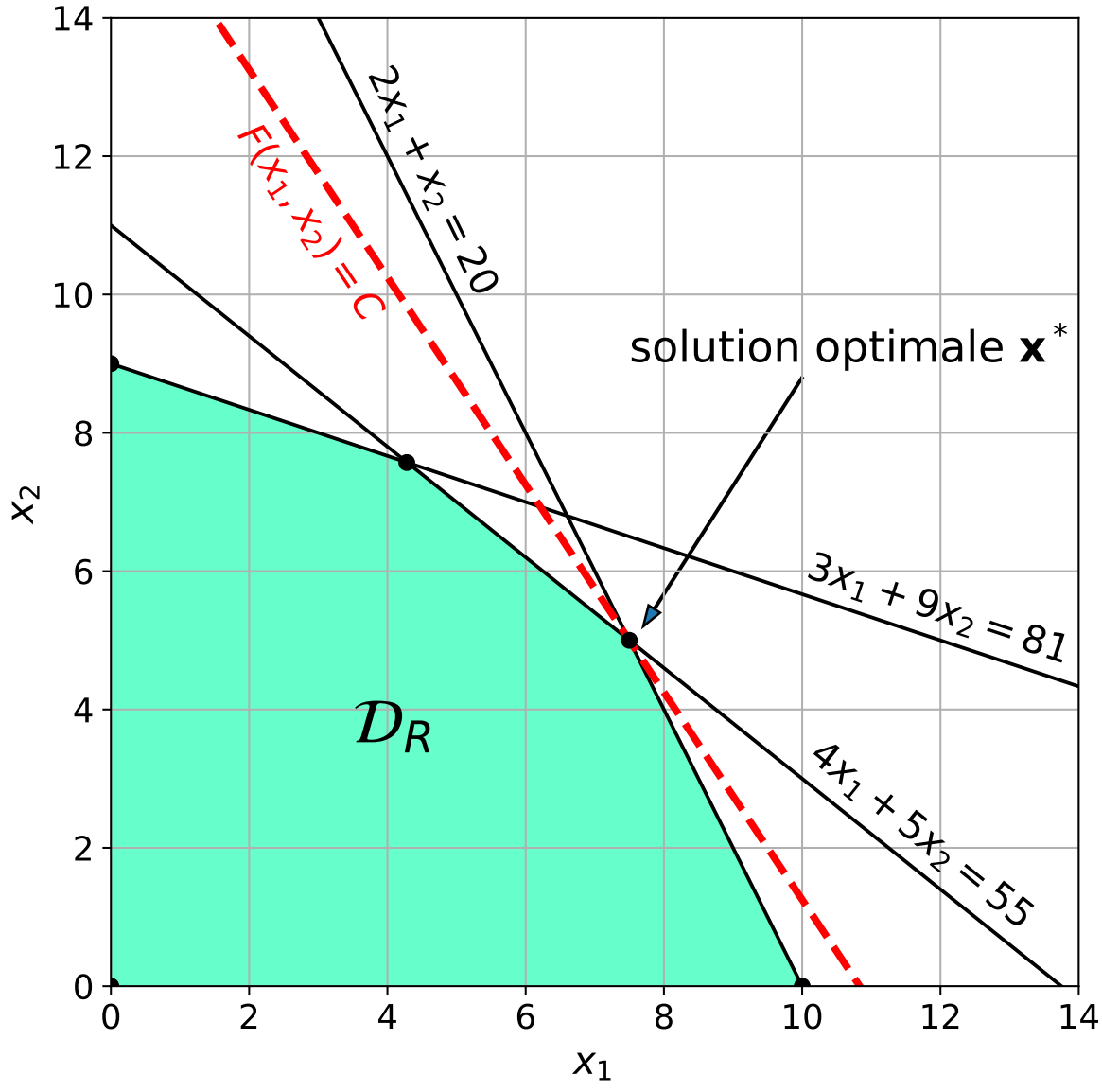
• **Un exemple**. On considère le PL suivant (problème de production)

$$\max_{(x_1,x_2)} \left[F(x_1,x_2) = 6x_1 + 4x_2\right].$$

sous les contraintes :

$$\begin{cases} 3x_1 + 9x_2 \leq 81 \\ 4x_1 + 5x_2 \leq 55 \\ 2x_1 + x_2 \leq 20 \\ x_1, x_2 \geq 0 \end{cases} \tag{8}$$

On peut résoudre graphiquement ce PL en dessinant tout d'abord l'ensemble $\mathcal{D}_R$ des solutions réalisables puis en faisant varier la constante C de la droite d'équation $F(x_1,x_2) = C$. Partant d'une grandeur valeur positive, on diminue C jusqu'à ce que la droite vienne "toucher" l'ensemble $\mathcal{D}_R$. La solution optimale est atteint en un sommet $\mathrm{x}^* = \begin{pmatrix} 7.5 \\ 5 \end{pmatrix}$.



☛ La solution optimale est atteinte en un sommet du polyèdre.

Le problème (8) s'écrit sous forme standard (variables d'écart e_1, e_2, e_3) :

$$\max_{x} \left[F(x) = c^\top x \right] . \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases} \tag{9}$$

avec

$$x = \begin{pmatrix} x_1 \\ x_2 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix}, \; c = \begin{pmatrix} 6 \\ 4 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \; A = \begin{pmatrix} 3 & 9 & 1 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 81 \\ 55 \\ 20 \end{pmatrix} .$$

On a $rang(A) = 3$ et il y a 5 solutions de base réalisables $x = \begin{pmatrix} x_B \\ x_H \end{pmatrix}$.

Par exemple :

- Base $B = \{3, 4, 5\}$ avec $A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$; solution de base

  réalisable $\mathrm{x}_B = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 81 \\ 55 \\ 20 \end{pmatrix}$, $\mathrm{x}_H = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

- Base $B = \{1, 2, 3\}$ avec $A_B = \begin{pmatrix} 3 & 9 & 1 \\ 4 & 5 & 0 \\ 2 & 1 & 0 \end{pmatrix}$; solution de base

  réalisable $\mathrm{x}_B = \begin{pmatrix} x_1 \\ x_2 \\ e_1 \end{pmatrix}$ avec $\mathrm{x}_B = A_B^{-1}\mathrm{b}$, $\mathrm{x}_H = \begin{pmatrix} e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

Caractérisation de l'optimum.

• **Sommet d'un polyèdre.** Un point $x \in \mathcal{D}_R$ est un **sommet** s'il n'existe pas de points y, $z \in \mathcal{D}_R$, $y \neq z$ tels que $x = \lambda y + (1-\lambda)z$ avec $0 < \lambda < 1$.

Théorème

1. x est une solution de base réalisable si et seulement si x est un sommet de $\mathcal{D}_R$.
2. Le maximum de la fonction objectif F sur $\mathcal{D}_R$, s'il existe, est atteint en au moins un sommet de $\mathcal{D}_R$.

Il suffit donc de se restreindre aux solutions de base réalisables. Au plus C_n^m solutions de base réalisables (toutes ne sont pas réalisables) avec un coût de $\mathcal{O}(m^3)$ opérations pour déterminer la solution du système $A_B x_B = b$ (méthode directe de type Gauss/factorisation LU).

$\Rightarrow$ nombre d'opérations d'ordre $\mathcal{O}(m^3 C_n^m)$, ce qui devient vite énorme : pas d'exploration *exhaustive* possible des solutions.

Avec la **méthode du simplexe**, on va explorer seulement les sommets qui permettent d'augmenter F et on évitera ainsi d'explorer *tous* les sommets.

Algorithme du simplexe.

On suppose qu'on dispose d'une solution de base réalisable $\underline{x}$ associée à une base B. On veut trouver une autre base B^* et $\underline{x}^*$ solution de base réalisable "voisine" telle que $\underline{x}^*$ est meilleure que $\underline{x}$, au sens où :

$$F(\underline{x}^*) > F(\underline{x}) \tag{10}$$

Principe de la méthode du simplexe : faire rentrer une variable hors-base dans la nouvelle base (variable *entrante*) et faire sortir à la place une variable de base (variable *sortante*).

•**Variable entrante.** On exprime F en fonction des variables hors-base.

Proposition *(Coûts réduits)*

Pour tout $\mathrm{x} \in \mathcal{D}_R = \{\mathrm{x} \in \mathbb{R}^n,\ A\mathrm{x} = \mathrm{b},\ \mathrm{x} \geq 0\}$, on a

$$F(\mathrm{x}) = F(\underline{x}) + \mathrm{L}_H^\top \mathrm{x}_H \quad \text{avec} \quad \mathrm{L}_H^\top = \mathrm{c}_H^\top - \mathrm{c}_B^\top A_B^{-1} A_H. \tag{11}$$

L_H est le vecteur des *coûts réduits*.

Choix de la variable entrante

La variable entrante $x_e \in \mathbb{R}$ est la variable hors-base (une composante de x_H) qui a le coût réduit (coefficient de L_H) **positif le plus grand possible**.

Si les coûts réduits sont tous négatifs ($L_H \leq 0$), la méthode du simplexe s'arrête (fin normale) : la solution de base réalisable $\underline{x}$ est optimale.

•**Variable sortante.** On maintient la relation $Ax = b$ avec $x \geq 0$.

$$\begin{aligned} Ax = b &\Leftrightarrow A_B x_B + A^e x_e = b \text{ où } A^e \text{ est la } e\text{-ième colonne de } A \\ &\Leftrightarrow x_B = A_B^{-1}(b - A^e x_e) \\ &\Leftrightarrow x_B = \underline{x}_B - A_B^{-1} A^e x_e \\ &\Leftrightarrow x_B = \underline{x}_B - z x_e \quad \text{avec} \quad z = A_B^{-1} A^e \in \mathbb{R}^m. \end{aligned}$$

Choix de la variable sortante

On doit avoir $x_B = \underline{x}_B - z x_e \geq 0$. La variable sortante x_s est celle *qui s'annule la première* lorsque x_e augmente.

Remarque.

Si $z \leq 0$ (i.e. toutes les composantes de z sont négatives) alors on a $x_B \geq 0$ quelque soit $x_e > 0$. La fonction F n'est donc pas bornée ($\max F = +\infty$) : arrêt de la méthode du simplexe.

Algorithme général.

1. Calcul des variables de base réalisables. Etant donné $A = (A_B \,|\, A_H)$, on calcule $\underline{x}_B = A_B^{-1} b \geq 0$.
2. Calcul des coûts réduits $L_H^\top = c_H^\top - c_B^\top A_B^{-1} A_H$. Si $L_H \leq 0$ alors $\underline{x}_B$ est optimale $\rightarrow$ arrêt.
3. Détermination de la variable entrante x_e.
4. Détermination de la variable sortante x_s (arrêt possible si $z \leq 0$: fonction objectif non majorée).
5. Nouvelle base B^* et matrice A_{B^*} obtenue en remplaçant la colonne A^s par A^e. Calcul de $A_{B^*}^{-1}$ et retour en 1.

Plusieurs méthodes de mise en œuvre de la méthode du simplexe qui diffèrent par la façon d'effectuer les différents calculs (solution de base réalisable x_B, coûts réduits L_H, etc.) : méthode des dictionnaires, méthodes des tableaux, simplexe révisé.

Méthode des dictionnaires

- On exprime les variables de base x_B ainsi que F en fonction des variables hors-base x_H. On obtient un système linéaire qu'on appelle **dictionnaire**.
- On choisit la variable entrante x_e comme la composante de x_H qui fait le plus augmenter F (coefficient du coût réduit L_H le plus grand).
- On détermine la variable sortante x_s en maintenant les conditions de positivité sur les variables de base $x_B \geq 0$ quand on augmente $x_e \nearrow$ à partir de 0.

Exemple du problème de production (8).

Solution de base réalisable initiale :
$x_1 = 0,\ x_2 = 0,\ e_1 = 81,\ e_2 = 55,\ e_3 = 20$ avec $F = 0$.

Convergence de la méthode des dictionnaires en 3 étapes. A la dernière étape, on obtient le dictionnaire :

$$\begin{array}{|l|}\hline x_2 = 5 - \frac{1}{3}e_2 + \frac{2}{3}e_3 \\ x_1 = \frac{15}{2} + \frac{1}{6}e_2 - \frac{5}{6}e_3 \\ e_1 = \frac{27}{2} + \frac{5}{2}e_2 - \frac{7}{2}e_3 \\ \hline F = 65 - \frac{1}{3}e_2 - \frac{7}{3}e_3 \\ \hline \end{array}$$

Coûts réduits $\mathrm{L}_H^\top = \left(-\frac{1}{3}, -\frac{7}{3}\right) < 0 \Rightarrow$ l'optimum est atteint et la solution optimale est

$$x_1^* = \frac{15}{2},\ x_2^* = 5,\ e_1^* = \frac{27}{2},\ e_2^* = 0,\ e_3^* = 0 \text{ avec } \max F = 65.$$

IV) Dualité

1) Introduction et définitions

L'exemple du problème de production (cf Introduction générale). On suppose qu'un acheteur se présente pour acheter toutes les ressources de l'entreprise. Il propose les prix unitaires y_1, y_2 et y_3 pour chacune des 3 ressources (équipement, énergie, matière première). On suppose que :

- l'entreprise acceptera de lui vendre toutes ses ressources si elle obtient pour chaque produit P_1 et P_2 un bénéfice plus grand que les prix unitaires de vente.
- l'acheteur veut minimiser ses dépenses.

Quels prix y_1, y_2 et y_3 doit-il proposer ?

Le problème se modélise par le *problème dual* :

$$\min_{(y_1,y_2,y_3)} \left[G(y_1,y_2,y_3) = 81y_1 + 55y_2 + 20y_3\right].$$
$$\begin{cases} 3y_1 + 4y_2 + 2y_3 \geq 6 \\ 9y_1 + 5y_2 + y_3 \geq 4 \\ y_1, y_2, y_3 \geq 0 \end{cases} \tag{12}$$

Définitions

Au programme linéaire primal

$$(PL) \quad \max_{x \in \mathbb{R}^n} \left[F(x) = c^\top x \right] \quad \begin{cases} Ax \leq b \\ x \geq 0 \end{cases}$$

où A est une matrice de taille $m \times n$, on associe le programme linéaire dual

$$(PLD) \quad \min_{y \in \mathbb{R}^m} \left[G(y) = b^\top y \right] \quad \begin{cases} A^\top y \geq c \\ y \geq 0 \end{cases}$$

La notion de dualité peut s'introduire avec le Lagrangien

$$\mathcal{L}(x, y) = c^\top x + (b - Ax)^\top y \tag{13}$$

associé au PL sous forme standard (contraintes $Ax = b$) qui peut s'écrire

$$\boxed{\text{(primal)} : \qquad \max_x \left\{ \min_y \mathcal{L}(x, y),\ x \geq 0 \right\}.} \tag{14}$$

On obtient le problème dual en échangeant le max et le min :

$$\boxed{\text{(dual)} : \qquad \min_y \left\{ \max_x \left(\mathcal{L}(x, y),\ x \geq 0 \right) \right\}.} \tag{15}$$

De façon générale, on a la définition suivante lorsque le problème primal est sous forme *canonique mixte* :

Primal	Dual
$\max_{x \in \mathbb{R}^n} \left[F(x) = c^\top x \right]$	$\min_{y \in \mathbb{R}^m} \left[G(y) = b^\top y \right]$
$\forall i \in I_1,\ \sum_{j=1}^{n} a_{ij} x_j \leq b_i$	$\forall i \in I_1,\ y_i \geq 0$
$\forall i \in I_2,\ \sum_{j=1}^{n} a_{ij} x_j = b_i$	$\forall i \in I_2,\ y_i$ de signe quelconque
$\forall j \in J_1,\ x_j \geq 0$	$\forall j \in J_1,\ \sum_{i=1}^{m} a_{ij} y_i \geq c_j$
$\forall j \in J_2,\ x_j$ de signe quelconque	$\forall j \in J_2,\ \sum_{i=1}^{m} a_{ij} y_i = c_j$

avec $I_1 \cup I_2 = \{1, \cdots, m\}$, $I_1 \cap I_2 = \emptyset$ et $J_1 \cup J_2 = \{1, \cdots, n\}$, $J_1 \cap J_2 = \emptyset$.

Cas particuliers primal/dual

Proposition

On a les correspondances suivantes entre problème primal et dual :

Primal	Dual
(PL_1) $\max_{x\in\mathbb{R}^n}\left[F(x)=c^\top x\right]$ $\begin{cases} Ax=b \\ x\geq 0 \end{cases}$	(PLD_1) $\min_{y\in\mathbb{R}^m}\left[G(y)=b^\top y\right]$ $\begin{cases} A^\top y\geq c \\ y \text{ de signes quelconques} \end{cases}$
(PL_2) $\max_{x\in\mathbb{R}^n}\left[F(x)=c^\top x\right]$ $\begin{cases} Ax=b \\ x \text{ de signes quelconques} \end{cases}$	(PLD_2) $\min_{y\in\mathbb{R}^m}\left[G(y)=b^\top y\right]$ $\begin{cases} A^\top y=c \\ y \text{ de signes quelconques} \end{cases}$

2) Théorèmes de dualité

Remarquons tout d'abord le résultat suivant :

Proposition

Le dual du dual est le primal

On s'intéresse à présent au lien entre les solutions de programmes linéaires en dualité.

Théorème faible de dualité

Soit x une solution réalisable d'un (PL) sous forme canonique (pure ou mixte) et y une solution réalisable du problème dual (PLD) associé.
Alors :

1. $F(x) \leq G(y)$
2. Si $F(x) = G(y)$ alors x et y sont des solutions optimales de (PL) et (PLD) respectivement.

Démonstration de la dualité faible (cas d'un PL sous forme canonique pure).

1. On a d'une part $Ax \leq b$, $x \geq 0$ et d'autre part $A^\top y \geq c$, $y \geq 0$. Par conséquent,
$$F(x) = c^\top x \leq (A^\top y)^\top x = y^\top \underbrace{Ax}_{\leq b} \leq y^\top b = G(y) \text{ car } y \geq 0$$
2. Soient x^* et y^* des solutions réalisables de (PL) et (PLD) respectivement telles que $F(x^*) = G(y^*)$. D'après 1., pour toute solution réalisable x de (PL), on a $F(x) \leq G(y^*) = F(x^*)$ donc x^* est une solution réalisable optimale qui réalise le maximum de F. Idem pour y^*. □

Théorème fort de dualité

Si le problème primal (PL) admet une solution réalisable optimale x^* alors le problème dual (PLD) associé admet lui aussi une solution réalisable optimale y^* et on a
$$F(x^*) = G(y^*).$$

Il y a 3 cas possibles (et seulement 3) pour le problème primal (PL) :

(1) il existe (au moins) une solution optimale.

(2) l'ensemble $\mathcal{D}_R$ des solutions réalisables n'est pas borné et l'optimum est infini.

(3) pas de solution réalisable ($\mathcal{D}_R = \emptyset$).

Les mêmes situations se retrouvent pour le problème dual.

Théorème

Etant donnés un problème primal (PL) et son dual (PLD), une et une seule des trois situations suivantes peut se produire.

(a) les deux problèmes possèdent chacun des solutions optimales (à l'optimum, les coûts sont égaux).

(b) un des problèmes possède une solution réalisable avec un optimum infini, l'autre n'a pas de solution.

(c) aucun des deux problèmes ne possède de solution réalisable.

Remarque. Si l'un des problèmes possède une solution réalisable et l'autre n'a pas de solution réalisable alors le premier a un coût non-borné (optimum infini). Ainsi la condition d'optimum infini dans (b) n'est pas nécessaire. En effet, si (PL) admet une solution réalisable avec un optimum fini alors d'après (a) le dual (PLD) a aussi une solution réalisable.

En résumé. Il y a donc 3 situations (au lieu de 9) :

		Dual		
		(1) solution optimale	*(2) optimum infini*	*(3) non-réalisable*
Primal	*(1) solution optimale*	(a)	impossible	impossible
	(2) optimum infini	impossible	impossible	(b)
	(3) non-réalisable	impossible	(b)	(c)

3) Théorème des écarts complémentaires.

Cas (a) : le primal et le dual possèdent chacun des solutions optimales (optimum fini).

☛ On peut alors calculer l'une à partir de l'autre.

Théorème des écarts complémentaires (TEC)

Soient x et y des solutions réalisables respectivement du problème primal (PL) sous forme canonique mixte et du problème dual (PLD). Alors x et y sont des solutions réalisables optimales si et seulement si

$$\bullet\ \forall i \in I_1,\ \sum_{j=1}^{n} a_{ij}x_j = b_i \quad \textbf{ou}\ y_i = 0 \tag{16}$$

$$\bullet\ \forall j \in J_1,\ \sum_{i=1}^{m} a_{ij}y_i = c_j \quad \textbf{ou}\ x_j = 0 \tag{17}$$

On peut interpréter ce résultat de la façon suivante :

- Si une contrainte est satisfaite en tant qu'inégalité stricte dans (PL) (resp. dans (PLD)) alors la variable correspondante de (PLD) (resp. de (PL)) est nulle.
- Si la valeur d'une variable dans (PL) ou (PLD) est strictement positive alors la contrainte correspondante de l'autre programme est une égalité.

Démonstration de la condition nécessaire (cas d'un PL sous forme canonique pure). Soient x et y des solutions réalisables *optimales* de (PL) et (PLD) respectivement. On a donc $Ax \leq b$, $x \geq 0$ et $A^\top y \geq c$, $y \geq 0$. En introduisant les variables d'écart e et ε respectivement pour (PL) et (PLD), on a

$$\begin{array}{l} Ax + e = b \\ x \geq 0,\ e \geq 0 \end{array} \quad \text{et} \quad \begin{array}{l} A^\top y - \varepsilon = c \\ y \geq 0,\ \varepsilon \geq 0 \end{array}$$

Dans ces conditions,

$$F(\mathrm{x}) = \mathrm{c}^\top \mathrm{x} = (A^\top \mathrm{y} - \varepsilon)^\top \mathrm{x} = \mathrm{y}^\top A\mathrm{x} - \varepsilon^\top \mathrm{x}$$
$$G(\mathrm{y}) = \mathrm{b}^\top \mathrm{y} = (A\mathrm{x} + \mathrm{e})^\top \mathrm{y} = (A\mathrm{x})^\top \mathrm{y} + \mathrm{e}^\top \mathrm{y} = \mathrm{y}^\top A\mathrm{x} + \mathrm{e}^\top \mathrm{y}.$$

Or d'après le Théorème de la dualité forte, $F(\mathrm{x}) = G(\mathrm{y})$ donc on obtient

$$\varepsilon^\top \mathrm{x} + \mathrm{e}^\top \mathrm{y} = 0. \tag{18}$$

Puisque $\mathrm{x} \geq 0$ et $\mathrm{y} \geq 0$, on a nécessairement

$$\begin{cases} \varepsilon_i x_i = 0, & \forall i \\ e_j y_j = 0, & \forall j \end{cases}$$

On obtient ainsi les relations, parfois appelées *relations d'exclusion* :

$$\begin{cases} \text{Si } \varepsilon_i \neq 0 \text{ alors } x_i = 0 \\ \text{Si } x_i \neq 0 \text{ alors } \varepsilon_i = 0, \end{cases} \qquad \begin{cases} \text{Si } e_j \neq 0 \text{ alors } y_j = 0 \\ \text{Si } y_j \neq 0 \text{ alors } e_j = 0. \end{cases}$$

La réciproque (condition suffisante) se démontre à partir du Théorème faible de dualité. □

Utilisation pratique du TEC.

La dualité et le TEC permettent souvent de vérifier si une solution réalisable x d'un (PL) est optimale ou non :

- on vérifie que x est une solution réalisable de (PL).
- si on peut, on détermine y par le TEC si on obtient suffisamment d'équations pour y.
- on vérifie que y est une solution *réalisable* du dual (PLD) ou pas en testant les contraintes non utilisées par le TEC.

Lorsque (PL) et (PLD) ont des solutions réalisables optimales x^* et y^* respectivement, on a :

- $\sum_{j=1}^{n} a_{ij} x_j^* < b_i \Rightarrow y_i^* = 0$
- $\sum_{i=1}^{m} a_{ij} y_i^* > c_j \Rightarrow x_j^* = 0$

et

$$\bullet\ y_i^* > 0 \ \Rightarrow \ \sum_{j=1}^{n} a_{ij} x_j^* = b_i$$

$$\bullet\ x_j^* > 0 \ \Rightarrow \ \sum_{i=1}^{m} a_{ij} y_i^* = c_j$$

Exemples. 1) Problème dual du problème de production

$$\min_{\mathrm{y}} \left[G(\mathrm{y}) = 81y_1 + 55y_2 + 20y_3 \right]$$

$$\begin{cases} 3y_1 + 4y_2 + 2y_3 \geq 6 \\ 9y_1 + 5y_2 + 1y_3 \geq 4 \\ y_1,\ y_2,\ y_3 \geq 0 \end{cases}$$

On veut vérifier que la solution

$$\mathrm{x} = (x_1, x_2, e_1, e_2, e_3)^\top = (15/2,\ 5,\ 27/2,\ 0,\ 0)^\top$$

du primal (PL) est optimale. La solution x est bien réalisable.

Le TEC donne :

$$\begin{aligned} e_1 = 27/2 > 0 &\overset{\text{TEC}}{\Longrightarrow} y_1 = 0 \\ x_1 = 15/2 > 0 &\overset{\text{TEC}}{\Longrightarrow} 3y_1 + 4y_2 + 2y_3 = 6 \ (\varepsilon_1 = 0) \\ x_2 = 5 > 0 &\overset{\text{TEC}}{\Longrightarrow} 9y_1 + 5y_2 + y_3 = 4 \ (\varepsilon_2 = 0) \\ e_2 = e_3 = 0 & \end{aligned}$$

En résolvant le système pour y, on obtient la solution optimale du problème dual :

$$y_1 = 0, \quad y_2 = 1/3, \quad y_3 = 7/3.$$

On vérifie que y est bien réalisable pour (PLD) donc x est optimale (et y aussi).

2) *Exercice* : soit le PL

$$\max_{x} [F(x) = 2x_1 - x_2 + x_3]$$

$$\begin{cases} x_1 + x_2 \leq 5 \\ x_2 - x_3 \geq 1 \\ x_1 + x_3 = 3 \\ x_1, \, x_2, \, x_3 \geq 0 \end{cases}$$

On veut savoir si $x = (3, 1, 0)^\top$ est une solution optimale.

On veut savoir si $\mathrm{x} = (3, 1, 0)^\top$ est une solution optimale. La solution x est bien réalisable. Sous forme standard, le problème s'écrit

$$\max_{\mathrm{x}} [F(\mathrm{x}) = 2x_1 - x_2 + x_3]$$
$$\begin{cases} x_1 + x_2 + e_1 = 5 \\ x_2 - x_3 - e_2 = 1 \\ x_1 + x_3 = 3 \\ x_1,\ x_2,\ x_3,\ e_1,\ e_2 \geq 0 \end{cases}$$

Le dual (PLD) s'écrit

$$\min_{\mathrm{y}} [G(\mathrm{y}) = 5y_1 + y_2 + 3y_3]$$
$$\begin{cases} y_1 + y_3 \geq 2 \\ y_1 + y_2 \geq -1 \\ -y_2 + y_3 \geq 1 \\ y_1 \geq 0,\ \ y_2 \leq 0,\ \ y_3 \text{ de signe quelconque} \end{cases}$$

La solution (réalisable) de PL sous forme standard est
$\mathrm{x} = (x_1, x_2, x_3, e_1, e_2)^\top = (3, 1, 0, 1, 0)^\top$

Le TEC donne

$$\begin{cases} 2-(y_1+y_3)=0 \\ -1-(y_1+y_3)=0 \\ y_1=0 \end{cases} \Leftrightarrow \begin{cases} y_1=0 \\ y_2=-1 \\ y_3=2 \end{cases}$$

On vérifie que y est bien une solution réalisable du dual (PLD) et donc que x est optimale (et y aussi).

4) Méthodes *primal-dual*.

Les conditions d'optimalité d'un problème primal sous forme standard conduisent - grâce au TEC - au problème suivant :

$$(PD) \quad \begin{cases} Ax = b \\ A^\top y - \varepsilon = c \\ \varepsilon^\top x = 0 \\ x \geq 0, \varepsilon \geq 0, \\ y \text{ de signes qcq} \end{cases}$$

dont les inconnues sont (x, y, ε).

On peut résoudre directement (PD) pour déterminer les solutions optimales x^* (primale) et y^* (duale) : on parle alors de méthodes *primal-dual*.

Le problème (PD) est non-linéaire à cause de la présence du produit $\varepsilon^\top x$. On peut utiliser une *méthode de descente* sur les variables duales prenant en compte la positivité ou bien des *méthodes intérieures* (avec Newton modifiée pour la positivité)

5) Méthode du *simplexe dual*.

☛ Dans certaines situations, il peut être avantageux de résoudre le problème dual plutôt que le problème primal. Par exemple, si le nombre de contraintes est beaucoup plus grand que le nombre de variables, on aura intérêt à résoudre par simplexe le problème dual car on sait que le coût de la méthode du simplexe dépend essentiellement du nombre de contraintes et dans une moindre mesure du nombre de variables.

☛ La méthode du simplexe dual ne consiste pas à résoudre directe le problème dual mais à appliquer la méthode du simplexe au problème primal en travaillant avec des solutions de base qui ne sont pas nécessairement *positives* (donc pas nécessairement réalisables).

Ces solutions de base sont en revanche *optimales* au sens où les coûts réduits associés sont toujours ≤ 0. On dit alors que ces solutions de base sont *dual-réalisables*. On va alors déterminer des variables entrante et sortante pour forcer les solutions de base à devenir positives.

☛ La méthode de simplexe dual est bien adaptée lorsqu'on rajoute une contrainte et qu'on ne veut pas recalculer la solution optimale en partant de rien mais au contraire en utilisant la solution optimale précédente obtenue avant l'ajout de la nouvelle contrainte. C'est le cas par exemple de la **méthode des coupes** pour la programmation linéaire en **nombres entiers** (voir Chap. 2).

IV) Quelques solveurs de PL

La plupart des solveurs résolvent des problèmes de programmation linéaire avec des variables entières/réelles : *Mixed Integer Linear Programming* (MIP).

Certains résolvent aussi les problèmes avec une fonction objectif quadratique (QP, *Quadratic Programming*) même non-linéaire, des contraintes quadratiques (QCP, *Quadratically Constrained Programming*) et tous les mélanges possibles : MIQP, MIQCP, ...

- Gurobi (depuis 2008). Code commercial, licence gratuite bridée (limitation de la taille des problèmes), licence éducation gratuite. C'est «le» solveur actuel (en 2025).
- CPLEX (IBM). Code commercial, licence éducation gratuite. Une référence aussi («la» référence ... avant Gurobi).
- SCIP, solveur universitaire non-commercial très prometteur..., développé au *Zuse Institute Berlin* (ZIB).
- HiGHS (high performance software for linear optimization), un solveur LP/MIP/QP aussi prometteur et opensource.

- COIN-OR (COmputational INfrastructure for Operations Research) Optimization Suite, une suite de logiciels *OpenSource*. En particulier, il contient :
  - Clp (Coin-or linear programming), un solveur PL.
  - CBC (Coin-or branch and cut), un solveur MILP.
- GLPK (GNU Linear Programming Kit). Code *OpenSource*, pour résoudre des problèmes de petite taille car très vite «poussif» quand les problèmes deviennent gros. Solveur PL/MILP.
- LPSOLVE. Code *OpenSource* PL/MILP.
- AMPL (A Mathematical Programming Language). C'est un langage de modélisation algébrique (décrire formellement un problème en vue de sa résolution numérique) qui ne résout pas directement les problèmes mais prend en charge des dizaines de solveurs (CBC, CPLEX, Gurobi ... et aussi des solveurs nonlinéaires).

et aussi ...

- MATLAB : fonctions `linprog` pour les PL et `intlinprog` pour les MILP ; interfaces pour Gurobi, CPLEX, CBC, Clp, GLPK.
- EXCEL intègre désormais un solveur LP.
- Python : interfaces pour Gurobi, SCIP, GLPK (module PyGLPK).

Formats de données standards.

Tous ces solveurs utilisent les formats de données les plus standards *MPS* et *LP*. Pour une description complète de ces formats (et d'autres !), vous pourrez consulter :

`https://docs.gurobi.com/projects/optimizer/en/current/reference/fileformats.html`

Exemples d'utilisation de solveurs. On considère le PL suivant.

$$(1) \quad \begin{cases} \max F(x) = 1700x_1 + 3200x_2 \\ 3x_2 \leq 39 \\ 1.5x_1 + 4x_2 \leq 60 \\ 2x_1 + 3x_2 \leq 57 \\ 3x_1 \leq 57 \\ x_1, x_2 \geq 0 \end{cases}$$

1. **Format de données LP** ; écrire dans le fichier `exo1.lp` :

```
maximize
        F : 1700 x1 + 3200 x2
subject to
        M1 : 3 x2 <= 39
        M2 : 1.5 x1 + 4 x2 <= 60
        M3 : 2 x1 + 3 x2 <= 57
        M4 : 3 x1 <= 57
end
```

Par défaut, les variables sont *réelles* (continues) et *positives*.

- Si les variables sont *entières*, il faut ajouter :

  ```
  int
      x1 x2
  ```

  ou bien si elles sont *binaires* :

  ```
  binary
      x1 x2
  ```

- Si les variables sont *bornées* (inf. et/ou sup.), il faut rajouter (par ex.) :

  ```
  bounds
      1 <= x1 <=15
      2 <= x2
  ```

(a) Exécution **directe** *de GUROBI.*

La commande d'exécution (dans un terminal linux ou dans l'invite de commandes windows) s'écrit :

```
gurobi_cl ResultFile=exo1.sol exo1.lp
```

Le fichier de résultat `exo1.sol` contient alors la solution optimale :

```
# Solution for model F
# Objective value = 5.4857142857142855e+04
x1 1.3714285714285714e+01
x2 9.8571428571428577e+00
```

Remarque. Sous Linux/MacOs, il y a aussi la possibilité d'exécuter GUROBI directement en ligne de commande (dans un terminal) : lancer simplement `gurobi.cl` pour ouvrir l'environnement *Gurobi Interactive Shell* (prompt "gurobi>").

Vous pouvez alors utiliser les commandes disponibles :

```
gurobi> m=read("exo1.lp")
gurobi> m.optimize()
```

(`help()` pour obtenir les différentes commandes).

(b) Utilisation de Gurobi à partir de python : `gurobipy`

Le solveur Gurobi est utilisé via un module python appelé `gurobipy`.

Un PL peut être défini directement à partir de tableaux `numpy` et aussi de matrices creuses (plusieurs types définis dans `scipy.sparse`)[4].

Voici un script possible pour résoudre le problème (1) (**LIRE les commentaires !**) :

La documentation complète est disponible ici

`https://docs.gurobi.com/projects/optimizer/en/current/reference/python.html`

4. C'est un des rares solveurs à proposer ces fonctionnalités qui évitent de perdre du temps à définir le modèle en rentrant les contraintes une par une (lorsqu'il y a beaucoup de contraintes, cela peut prendre du temps quand on utilise un langage interprété comme python).

```
import numpy as np
import gurobipy as gp

# on définit le vecteur c de la fct objectif
c = np.array([1700, 3200])

# on définit la matrice A et le vecteur b des contraintes
# d'inégalité
A = np.array([ [  0, 3],
               [1.5, 4],
               [  2, 3],
               [  3, 0]])
b = np.array([39, 60, 57, 57])

# On définit maintenant le modèle pour gurobi.
# 1/ Il faut d'abord instancier un objet de type ''Modèle Gurobi''
m = gp.Model()

# 2/ On définit les variables du modèle (le type est ''MVar Gurobi''),
# il faut donner le nombre de variables (i.e. autant de variables qu'il
# y a d'éléments dans c). Pas mal de réglages possibles (type des
# variables, bornes inf et/ou sup, donner un nom, etc.). Par défaut,
# les variables sont continues et bornées inférieurement par 0.
x = m.addMVar(len(c))
```

```
# 3/ On définit la fonction objectif, exactement comme on ferait pour
# définir le produit scalaire entre deux vecteurs numpy, c'est à dire
# avec l'opérateur @ (attention cependant si c est bien un tableau
# numpy, x joue le rôle d'une sorte de variable formelle). Dans notre
# cas il faut préciser qu'on veut maximiser (le défaut est de minimiser)
# GRB est un sous-module de gurobipy définissant toutes les constantes.
m.setObjective(c @ x, gp.GRB.MAXIMIZE)

# 4/ Les contraintes se définissent aussi naturellement que la fonction
# objectif, c'est à dire avec l'opérateur @ (bien sûr pour des contraintes
# d'égalité il faudrait utiliser A @ x == b et on pourrait aussi utiliser
# A @ x >= b), on peut aussi ajouter autant de contraintes que l'on veut.
m.addConstr( A @ x <= b)

# 5/ reste plus qu'à résoudre, ce qui s'obtient avec :
m.optimize()
```

```
# 6/ après optimisation, on peut :

# 6-a/ obtenir le statut avec :
m.Status

# qui retourne 2 pour OPTIMAL, 3 pour INFEASIBLE, 4 pour INF_OR_UNBD
#  et 5 pour UNBOUNDED mais d'autres possibilités existent comme 1 pour
# LOADED (le modèle n'a pas encore été optimisé) ou encore comme 9 pour
# TIME_LIMIT (il est possible de définir un temps max à ne pas dépasser).

# 6-b/ obtenir la solution (si le statut est OPTIMAL) sous forme de liste
# avec :
x_opt = m.X
# ou encore np.array(m.X) pour l'avoir sous forme de tableau numpy

# 6-c/ la valeur de la fonction objectif avec
f_opt = m.ObjVal
```

Remarques et compléments.

- **Sur les options de la méthode** `addMVar(nb_vars)`.

  L'argument imposé est le nombre de variables. Par défaut, les variables sont continues et inférieurement bornées par 0. Il existe (entre autres) les arguments optionnels suivants :

  - `lb=, ub=` bornes inférieures et supérieures ;
  - `vtype=` type des variables : `'C'`, `'I'`, `'B'`.

  Pour chaque argument, vous pouvez vous contenter d'un scalaire qui sera valable pour toutes les composantes, sinon il faut un *itérable* comme une liste. Pour le problème précédent, on aurait pu utiliser :

```
A = np.array([[1.5, 4], [2, 3]])
b = np.array([60, 57])
# instantiation du modèle
m = gp.Model()
# on donne les bornes sup lors de la définition des variables
x = m.addMVar(len(c), ub=[19,13])
```

Si vous avez certaines variables bornées supérieurement *et pas d'autres*, vous pouvez utiliser le nombre flottant spécial *Inf*, qui s'obtient via `np.inf` ou encore `float('inf')`, ainsi pour 3 variables avec la première majorée par 2 et les 2 suivantes non majorées, on utilisera

```
ub=[2, np.inf, np.inf]
```

- Lors de l'ajout de contraintes avec la méthode `addConstr`, si le second membre est constant (vecteur de 0, de 1, etc.), vous pouvez utiliser un scalaire, par exemple $Ax \geq 0$ peut s'écrire :

```
m.addConstr( A @ x >= 0 )
```

- Les variables gurobi peuvent être manipulées comme des tableaux `numpy`, on peut donc désigner une partie des variables à l'aide d'une liste d'indices ou de tranches.

Exemples :

```
n = 40
m = gp.Model()  # un modèle gurobi
X = m.addMVar(n)  # un ‘‘vecteur’’ (de variables gurobi)
                  # à n composantes
X[::2] # toutes les composantes paires
X[1::2] # toutes les composantes impaires
X[:10] # les 10 premières composantes
X[-10:] # les 10 dernières composantes
X[[1,5,7]] # les composantes d'indices 1, 5 et 7
```

Ces possibilités peuvent être utilisées lorsqu'on définit la fonction objectif et les contraintes.

Exemples.

- si la fonction objectif ne dépend que des m dernières variables, on pourra utiliser :

```
# modele est un modèle gurobi
# c un ndarray 1d de profil (m,)
# (coefs s'appliquant aux m dernières variables)
modele.setObjective( c @ X[-m:], gp.GRB.MINIMIZE)
```

- Si les p contraintes $Ax <= b$ ne dépendent que des m premières variables, on pourra se contenter de former un tableau A de profil (p, m) (au lieu d'un profil (p, n)) et écrire alors :

```
# modele est un modèle gurobi
# A un ndarray 2d de profil (p,m)
# b un ndarray 1d de profil (p,)
modele.addConstr( A @ X[:m] <= b )
```

- Le solveur peut travailler sur le problème dual. Ainsi dans certains cas, il ne peut pas savoir si le primal est non borné ou infaisable (domaine $\mathcal{D}_R = \emptyset$) d'où parfois la réponse 4 (`INF_OR_UNBD`) concernant le statut de l'optimisation. Si on veut savoir de quoi il en retourne, il est possible de refaire une optimisation du modèle en précisant :

```
m = gp.Model()
# pour savoir si notre problème est infaisable ou non borné:
m.Params.DualReductions = 0
```

- Il est possible et fortement conseillé pour les gros problèmes d'utiliser des matrices **creuses** pour définir la ou les matrices des contraintes. Seuls les coefficients non-nuls des matrices sont stockés. Il existe plusieurs formats creux plus ou moins efficaces en fonction des opérations à effectuer sur ces matrices (insertion/suppression d'éléments, multiplications,...)
- **Alternative aux formulations matricielles** : `quicksum`

  Plutôt que d'écrire l'objectif et les contraintes sous la forme $c^\top x$ et $Ax = 0$, on peut utiliser la fonction gurobi `quicksum` qui permet de manipuler directement les variables dans des sommes. On peut également définir des variables indicées dans un *dictionnaire* python où les clés sont les indices.

  Exemple. Variables binaires x_{ij} avec $i \in [\![1, n]\!]$, $j \in [\![1, m]\!]$ vérifiant les contraintes

$$\sum_{j=1}^{m} x_{ij} = 1, \ \forall i \in [\![1, n]\!]$$

```
# gurobipy est importé avec : import gurobipy as gp
# model est un modèle gurobi

# définition des variables binaires x[i,j]
x = {}   # x est un dictionnaire
for i in range(n):
    for j in range(m):
        x[(i, j)]=model.addVar(vtype=gp.GRB.BINARY, name=f"x_{i}_{j}")

model.update()    # mise à jour du modèle

# contraintes
for i in range(n):
     model.addConstr(gp.quicksum(x[(i, j)] for j in range(m)) == 1)
```

- gurobi s'arrête lorsqu'il a détecté une solution optimale mais il peut essayer de trouver l'ensemble des solutions pour un MIP :

```
# recherche de n solutions
space.setParam("PoolSolutions", n)
space.setParam("PoolSearchMode", 2) # pour avoir les meilleures sol.
```