

# Chapitre 5: Programmation linéaire

MAT7560 — Optimisation combinatoire

Alexandre Blondin Massé

Université du Québec à Montréal

Hiver 2019

# Plan

- 1 Généralités
- 2 Modélisation d'un problème
- 3 Transformations d'un problème
- 4 Algorithme du simplexe
- 5 Théorie de la dualité
- 6 Logiciels

# Généralités

# Formulation générale

$$\begin{array}{ll}\text{minimiser (ou maximiser)} & \sum_{j=1}^n c_j x_j \\ \text{sous les contraintes} & \sum_{j=1}^n a_{ij} x_j \square b_i, \quad i = 1, 2, \dots, m \\ & x_j \square 0, \quad j = 1, 2, \dots, n\end{array}$$

où  $\square$  peut être remplacé par  $\leq$ ,  $=$  ou  $\geq$

- Les  $x_j$  sont appelées **variables de décision**
- Les valeurs  $c_j$ ,  $a_{ij}$  et  $b_i$  sont des **constantes**

Notation **matricielle**

$$\begin{array}{ll}\text{minimiser (ou maximiser)} & \mathbf{c}^T \mathbf{x} \\ \text{sous les contraintes} & \mathbf{Ax} \square \mathbf{b} \\ & \mathbf{x} \square \mathbf{b}\end{array}$$

# Terminologie

- Les contraintes décrites par  $\mathbf{Ax} \leq \mathbf{b}$  sont appelées **contraintes fonctionnelles**
- Certaines des variables  $x_j$  peuvent être **non restreintes**
- Un vecteur  $\mathbf{x}$  qui satisfait toutes les contraintes est appelé **vecteur réalisable** ou **solution réalisable**
- La fonction  $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$  est appelée **fonction objectif** ou **fonction de coût**
- Une solution  $\mathbf{x}^*$  qui optimise la fonction objectif est appelée **solution optimale**
- La valeur  $\mathbf{c}^T \mathbf{x}^*$  est alors appelée **valeur optimale** ou **coût optimal**
- Si la valeur optimale est  $+\infty$  ou  $-\infty$ , on dit que la solution est **non bornée**

## Exemple: 2 variables

Soit le **programme linéaire** suivant:

$$\begin{array}{ll}\text{minimiser} & -x_1 - x_2 \\ \text{sous les contraintes} & x_1 + 2x_2 \leq 3 \\ & 2x_1 + x_2 \leq 3 \\ & x_1, x_2 \geq 0\end{array}$$

- Dessiner la région **réalisable**
- Calculer une solution **optimale**
- Que se passe-t-il si la fonction objectif est  $-x_1 + 2x_2$ ?

## Exemple: 3 variables

Considérons le **programme linéaire** suivant:

$$\begin{array}{ll}\text{minimiser} & -x_1 - x_2 - x_3 \\ \text{sous les contraintes} & x_i \leq 1, \ i = 1, 2, 3 \\ & x_1, x_2, x_3 \geq 0\end{array}$$

- Dessiner la région **réalisable**
- Calculer une solution **optimale**

# Différents scénarios (1/2)

Considérons le **programme linéaire** suivant:

$$\begin{array}{ll}\text{minimiser} & \mathbf{c}^T \mathbf{x} \\ \text{sous les contraintes} & -x_1 + x_2 \leq 1 \quad x_1, x_2 \geq 0\end{array}$$

- Si  $\mathbf{c} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ , alors l'**unique** solution optimale est  $\mathbf{x} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$
- Si  $\mathbf{c} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ , alors l'ensemble solution  $\mathbf{x} = \begin{bmatrix} 0 & x_2 \end{bmatrix}^T$ ,  $0 \leq x_2 \leq 1$  est **borné**
- Si  $\mathbf{c} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$ , alors l'ensemble solution  $\mathbf{x} = \begin{bmatrix} x_1 & 0 \end{bmatrix}^T$ ,  $x_1 \geq 0 \leq 1$  est **non borné**
- Si  $\mathbf{c} = \begin{bmatrix} -1 & -1 \end{bmatrix}^T$ , alors il n'y a **aucune** solution optimale et le coût optimal est **non borné**
- Si on **ajoute** la contrainte  $x_1 + x_2 \leq -2$ , alors il n'y a **aucune solution réalisable**



# Différents scénarios (2/2)

## Théorème

Soit  $P$  un problème de programmation linéaire.

Alors exactement un des scénarios suivants survient:

- Il existe une **unique** solution optimale
- Il existe **plusieurs** solutions optimales. L'ensemble des solutions peut être **borné** ou **non borné**
- Le coût optimal est  $+\infty$  ou  $-\infty$ , et **aucune** solution réalisable n'est optimale
- L'ensemble des solutions **réalisables** est vide, de sorte que le problème est **non réalisable**

## Remarques

- Il n'est pas toujours **évident** si un problème est réalisable ou non
- Ou s'il est **borné**

# Modélisation d'un problème

# Étapes

- Identifier d'abord les **variables** (inconnues) qui seront les **variables de décision** et les représenter par des **symboles**
- Identifier toutes les **contraintes** ou **restrictions** du problème et les exprimer comme des **équations** ou **inéquations** linéaires en fonction des variables de décision
- Identifier la **fonction objectif** et l'exprimer en fonction des variables de décision

# Le problème de la diète

- Une ferme utilise **au moins** 800 kg de nourriture par jour
- La nourriture est un **mélange** de maïs et de soya
- Il faut au moins 30% de protéines et au plus 5% de fibres
- Les **coûts** sont décrits dans le tableau ci-bas:

Céréale	Protéines (%)	Fibres (%)	Coût (\$/kg)
Maïs	0.09	0.02	0.30
Soya	0.60	0.06	0.90

- On cherche le **coût** minimal
- Formuler le problème comme un **programme linéaire**

# Solution

On définit les **variables** de décision:

- $x_1$ : quantité de maïs
- $x_2$ : quantité de soja

Contraintes:

- **Protéines**:  $0.09x_1 + 0.60x_2 \geq 0.3(x_1 + x_2)$
- **Fibres**:  $0.02x_1 + 0.06x_2 \leq 0.05(x_1 + x_2)$

Alors on obtient le programme **linéaire**

$$\begin{array}{ll} \text{minimiser} & 0.3x_1 + 0.9x_2 \\ \text{sous les contraintes} & \begin{array}{rcl} x_1 + x_2 & \geq & 800 \\ -0.21x_1 + 0.3x_2 & \geq & 0 \\ -0.03x_1 + 0.01x_2 & \leq & 0 \\ x_1, x_2 & \geq & 0 \end{array} \end{array}$$

# Construction d'horaires

- Dans une entreprise, le **nombre d'employés** nécessaires chaque jour varie selon le tableau suivant:

Jour	Nombre d'employés
Lundi	17
Mardi	13
Mercredi	15
Jeudi	19
Vendredi	14
Samedi	16
Dimanche	11

- Un travailleur à temps plein travaille **5 jours** consécutifs, suivi de **2 jours** de congé
- On souhaite **minimiser** le nombre d'employés à embaucher
- Formulation comme un **problème linéaire**?

# Solution

Soit  $x_j$  le nombre d'employés qui **commencent** le jour  $j$

Alors on cherche à

$$\begin{array}{ll} \text{minimiser} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\ \text{sous les contraintes} & \begin{array}{l} x_1 + x_4 + x_5 + x_6 + x_7 \geq 17 \\ x_1 + x_2 + x_5 + x_6 + x_7 \geq 13 \\ x_1 + x_2 + x_3 + x_6 + x_7 \geq 15 \\ x_1 + x_2 + x_3 + x_4 + x_7 \geq 19 \\ x_1 + x_2 + x_3 + x_4 + x_5 \geq 14 \\ x_2 + x_3 + x_4 + x_5 + x_6 \geq 16 \\ x_3 + x_4 + x_5 + x_6 + x_7 \geq 11 \\ x_j \geq 0 \\ x_j \in \mathbb{Z} \end{array} \end{array}$$

Il s'agit d'un programme linéaire en **nombres entiers**

# Transformations d'un problème



# Deux formes

## Forme compacte

minimiser  $\mathbf{c}^T \mathbf{x}$   
sous les contraintes  $\mathbf{Ax} \geq \mathbf{b}$

## Forme standard

minimiser  $\mathbf{c}^T \mathbf{x}$   
sous les contraintes  $\mathbf{Ax} = \mathbf{b}$

# Équivalences de problèmes

## Équivalence

Deux programmes linéaires sont dits **équivalents** si une solution optimale de l'un permet de **construire** une solution de l'autre.

## Observations

- **Tout** programme linéaire peut être transformé en forme **compacte**
- **Tout** programme linéaire peut être transformé en forme **standard**

## Intérêt

- La forme compacte ( $\leq$ ) simplifie les **démonstrations**
- La forme standard ( $=$ ) est plus utile pour des raisons de **calculs** (algorithme du **simplexe**)

# Transformation sous forme compacte

Considérons le problème suivant:

$$\begin{array}{ll}\text{minimiser} & 2x_1 - x_2 + 4x_3 \\ \text{sous les contraintes} & x_1 + x_2 + x_4 \geq 2 \\ & 3x_2 - x_3 = 5 \\ & x_3 + x_4 \leq 3 \\ & x_1 \geq 0 \\ & x_3 \leq 0\end{array}$$

- Le réécrire sous forme **compacte**

# Transformation sous forme standard

- **Étape 1:** éliminer les variables **néglatives** et les variables **non restreintes**
  - Remplacer  $x_j \leq 0$  par  $\bar{x}_j = -x_j$ , où  $\bar{x}_j \geq 0$
  - Remplacer les variables **non restreintes**  $x_j$  par  $x_j^+ - x_j^-$ , avec  $x_j^+ \geq 0$  et  $x_j^- \geq 0$
- **Étape 2:** éliminer les **inéquations**
  - Remplacer les inéquations  $\leq$  par une équation en introduisant une variable **libre** (*slack variable*)
  - Par exemple  $x_1 + 2x_2 \leq 3$  devient  $x_1 + 2x_2 + s_1 = 3$ , avec  $s_1 \geq 0$
  - Remplacer les inéquations  $\geq$  par une équation en introduisant une variable **de surplus** (*surplus variable*)
  - Par exemple  $3x_1 + 4x_2 \geq 1$  devient  $3x_1 + 4x_2 - s_1 = 1$ , avec  $s_1 \geq 0$

# Exemple

Transformons le problème suivant sous forme **standard**

$$\begin{array}{ll}\text{minimiser} & 2x_1 - x_2 + 4x_3 \\ \text{sous les contraintes} & x_1 + x_2 + x_4 \geq 2 \\ & 3x_2 - x_3 = 5 \\ & x_3 + x_4 \leq 3 \\ & x_1 \geq 0 \\ & x_3 \leq 0\end{array}$$

# Solution

minimiser  $2x_1 - x_2^+ + x_2^- - 4x_3'$   
sous les contraintes

$$\begin{aligned}x_1 + x_2^+ - x_2^- + x_4^+ - x_4^- - s_1 &= 2 \\ 3x_2^+ - 3x_2^- + x_3' &= 5 \\ -x_3' + x_4^+ - x_4^- + s_2 &= 3 \\ x_1, x_2^+, x_2^-, x_3', x_4^+, x_4^-, s_1, s_2 &\geq 0\end{aligned}$$

# Fonctions convexes par morceau

## Définition

Une fonction de la forme

$$\max_{i=1,\dots,m} (\mathbf{c}_i^T \mathbf{x} + d_i)$$

est appelée **convexe par morceau**

## Exemple 1

- $f(x) = \max\{2x, 1 - x, 1 + x\}$
- Dessiner le graphe de  $f$

## Exemple 2

Montrer que  $f(x) = |x|$  est convexe par morceau

# Optimisation de fonctions convexes par morceau

## Proposition

Le problème

$$\begin{array}{ll} \text{minimiser} & \max_{i=1,\dots,m} (\mathbf{c}_i^T \mathbf{x} + d_i) \\ \text{sous les contraintes} & \mathbf{Ax} \geq \mathbf{b} \end{array}$$

est équivalent au problème

$$\begin{array}{ll} \text{minimiser} & y \\ \text{sous les contraintes} & y \geq \mathbf{c}_i^T \mathbf{x} + d_i, \quad i = 1, \dots, m \\ & \mathbf{Ax} \geq \mathbf{b} \end{array}$$



# Exemple

Transformer le programme suivant en programme linéaire

$$\begin{array}{ll} \text{minimiser} & \max\{3x_1 - x_2, x_2 + 2x_3\} \\ \text{sous les contraintes} & 2x_1 + 3x_2 \leq 5 \\ & x_2 - 2x_3 \leq 6 \\ & x_3 \leq 7 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

# Solution

On trouve

$$\begin{array}{ll}\text{minimiser} & x_4 \\ \text{sous les contraintes} & 3x_1 - x_2 - x_4 \geq 0 \\ & x_2 + 2x_3 - x_4 \geq 0 \\ & 2x_1 + 3x_2 \leq 5 \\ & x_2 - 2x_3 \leq 6 \\ & x_3 \leq 7 \\ & x_1, x_2, x_3 \geq 0\end{array}$$

On pourrait aussi le transformer sous forme **compacte** ou **standard** ensuite!

# Algorithme du simplexe

# Idée générale

- On part avec une solution **réalisable**  $x$
- On calcule différentes statistiques
- On choisit la dimension la plus **prometteuse**
- On modifie la solution selon cette **dimension**
- On répète jusqu'à ce qu'on ne puisse rien **améliorer**

## Difficultés

- Comment trouver une **solution initiale**?
- Comment choisir la dimension **prometteuse**?

# Exemple

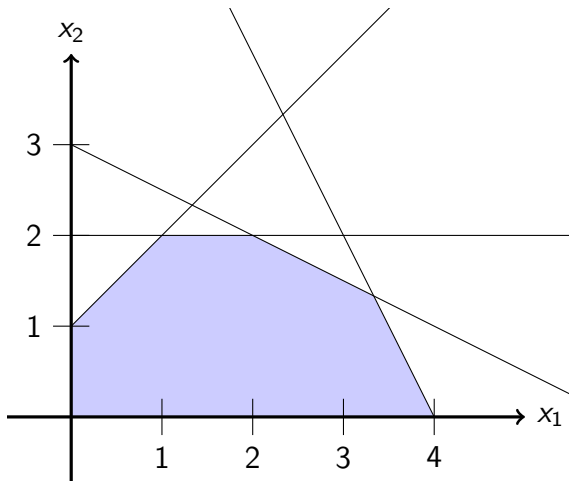
Montrons comment résoudre le problème suivant

$$\begin{array}{ll}\text{minimiser} & -3x_1 - 2x_2 \\ \text{sous les contraintes} & x_1 + 2x_2 \leq 6 \\ & 2x_1 + x_2 \leq 8 \\ & -x_1 + x_2 \leq 1 \\ & x_2 \leq 2 \\ & x_1, x_2 \geq 0\end{array}$$

Étapes à suivre:

- Transformer le problème sous forme **standard**
- Trouver une solution réalisable **initiale**
- L'améliorer **successivement**

# Région réalisable



# Transformation sous forme standard

- On introduit des **variables**  $s_1, s_2, s_3$  et  $s_4$

$$\begin{array}{llllllll} \text{minimiser} & -3x_1 & -2x_2 & +0s_1 & +0s_2 & +0s_3 & +0s_4 & \\ \text{s. c.} & x_1 & +2x_2 & +s_1 & & & & =6 \\ & 2x_1 & +x_2 & & +s_2 & & & =8 \\ & -x_1 & +x_2 & & & +s_3 & & =1 \\ & & x_2 & & & & +s_4 & =2 \\ & x_1, x_2, s_1, s_2, s_3, s_4 & \geq 0 & & & & & \end{array}$$

- Ensuite, on effectue la trace à l'aide de **tableaux**

# Étape 1

- On initialise le tableau avec les valeurs du problème
- **Colonne** pivot: minimise  $\bar{c}$
- **Ligne** pivot: minimise le ratio positif solution/colonne pivot

Base	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	$s_4$	Solution	Ratio
$\bar{c}$	-3	-2	0	0	0	0	0	
$s_1$	1	2	1	0	0	0	6	6
$s_2$	2	1	0	1	0	0	8	4
$s_3$	-1	1	0	0	1	0	1	
$s_4$	0	1	0	0	0	1	2	

- Puis on fait des opérations de **ligne** pour que la colonne pivot devienne 0 partout sauf sur la ligne pivot, où la valeur est 1



## Étape 2

Base	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	$s_4$	Solution	Ratio
$\bar{c}$	0	$-1/2$	0	$3/2$	0	0	12	
$s_1$	0	$3/2$	1	$-1/2$	0	0	2	$4/3$
$x_1$	1	$1/2$	0	$1/2$	0	0	4	8
$s_3$	0	$3/2$	0	$1/2$	1	0	5	$10/3$
$s_4$	0	1	0	0	0	1	2	2

- Puis on répète...

## Étape 3

Base	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	$s_4$	Solution
$\bar{c}$	0	0	$1/3$	$4/3$	0	0	$12\frac{2}{3}$
$x_2$	0	1	$2/3$	$-1/3$	0	0	$4/3$
$x_1$	1	0	$-1/3$	$2/3$	0	0	$10/3$
$s_3$	0	0	-1	1	1	0	3
$s_4$	0	0	$-2/3$	$1/3$	0	1	$2/3$

- On arrête car toutes les valeurs potentielles dans  $\bar{c}$  sont non négatives
- Ainsi, on obtient comme valeur optimale  $12\frac{2}{3}$
- Une solution optimale est  $x_1 = 10/3$  et  $x_2 = 4/3$

# Exercice

Appliquer la méthode du simplexe au programme suivant:

$$\begin{array}{ll}\text{minimiser} & -5x_1 + 4x_2 - 6x_3 - 8x_4 \\ \text{s. c.} & x_1 + 7x_2 + 3x_3 + 7x_4 \leq 46 \\ & 3x_1 - 2x_2 + x_3 + 2x_4 \leq 8 \\ & 2x_1 + 3x_2 - x_3 + x_4 \leq 10 \\ & x_1, x_2, x_3, x_4 \geq 0\end{array}$$

# Description générale

- On initialise une **base** avec les colonnes  $\mathbf{A}_{B(1)}, \mathbf{A}_{B(2)}, \dots, \mathbf{A}_{B(m)}$
- On calcule les **coûts réduits**  $\bar{c}_j = c_j - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_j$  pour toutes les variables **hors base**
- Si tous les coûts sont **non négatifs**, on a atteint l'optimal
- Sinon, on choisit un indice  $j$  tel que  $\bar{c}_j < 0$
- La variable  $x_j$  est appelée **variable entrante**
- On calcule  $\mathbf{u} = \mathbf{A}_B^{-1} \mathbf{A}_j$
- Si aucune composante de  $\mathbf{u}$  n'est positive, le problème est **non borné**
- Sinon, soit  $\theta = \min\{x_{B(i)}/u_i : u_i > 0\}$
- Soit  $\ell$  tel que  $\theta = x_{B(\ell)}/u_\ell$
- Alors  $x_{B(\ell)}$  est appelé **variable sortante**
- On remplace  $\mathbf{A}_{B(\ell)}$  par  $\mathbf{A}_j$  dans la base en mettant à jour

# Théorie de la dualité

# Idée générale

- Problème **primal**  $P$ : on optimise dans une direction
- En cherchant l'**infimum** d'un ensemble  $\mathcal{S}_P$
- Problème **dual**  $D$ : on optimise dans la direction contraire
- En cherchant le **supremum** de l'ensemble

$$\mathcal{S}_D = \{y \mid y \leq x, \text{ pour tout } x \in \mathcal{S}_P\}$$

- En particulier, on a la relation

$$\inf(\mathcal{S}_P) = \sup(\mathcal{S}_D)$$

c'est-à-dire que les solutions admissibles se **rencontrent**

# Motivation

Deux principales motivations:

- La résolution du problème dual  $D$  est parfois **plus facile** que la résolution du problème primal  $P$
- Pour toute solution  $S \in \mathcal{S}_P$  et pour toute solution  $S' \in \mathcal{S}_D$ , le nombre

$$|f(S) - f(S')|$$

est un **majorant** de l'**erreur**

$$f(S) - \inf(\mathcal{S}_D)$$

- Il est donc possible d'estimer l'**erreur** en lorsqu'on doit résoudre des problèmes linéaires en **nombres entiers**, qui sont difficiles

# Définition

Considérons le problème (**primal**) suivant sous forme standard:

$$\begin{array}{ll}\text{minimiser} & \mathbf{c}^T \mathbf{x} \\ \text{sous les contraintes} & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}\end{array}$$

Alors le problème **dual** associé est donné par

$$\begin{array}{ll}\text{maximiser} & \mathbf{y}^T \mathbf{b} \\ \text{sous les contraintes} & \mathbf{y}^T \mathbf{A} \leq \mathbf{c}^T\end{array}$$



# Remarques et conséquences

- Le problème dual n'est ni sous forme **standard**, ni sous forme **compacte**, mais il peut facilement être transformé
- Chaque contrainte **fonctionnelle** est « remplacée » par une **variable** dans le problème dual
- De la même façon, chaque **variable** introduit une **contrainte** correspondante dans le dual

## Théorème (dual d'un dual)

Soit  $P$  un programme linéaire,  $P'$  son problème dual associé et  $P''$  le dual associé au problème  $P'$ . Alors  $P$  et  $P''$  sont **équivalents**.

# Lien entre primal et dual

## Théorème (primal/dual)

Soit  $P$  un problème de programmation linéaire et  $D$  son dual associé. Alors il existe un programme linéaire qui est équivalent à  $D$  obtenu en de la façon suivante

- On remplace « **maximiser** » par « **minimiser** » (ou inversement);
- Les vecteurs  $\mathbf{b}$  et  $\mathbf{c}$  sont **échangés**
- Chaque **contrainte** de la forme  $\geq$  (resp.  $\leq$ ,  $=$ ) est remplacée par une **variable** ayant la contrainte  $\geq 0$  (resp.  $\leq 0$ , aucune contrainte);
- Chaque **variable** de la forme  $\geq 0$  (resp.  $\leq 0$ , aucune contrainte) est remplacée par une **contrainte** de la forme  $\leq$  (resp.  $\geq$ ,  $=$ ).

# Exemple

Considérons le problème

$$\begin{array}{ll}\text{minimiser} & x_1 + 2x_2 + 3x_3 \\ \text{sous les contraintes} & -x_1 + 3x_2 = 5 \\ & 2x_1 - x_2 + 3x_3 \geq 6 \\ & x_3 \leq 4 \\ & x_1 \geq 0 \\ & x_2 \leq 0\end{array}$$

- ① Écrire le problème dual en utilisant le théorème précédent
- ② Vérifier que le dual du dual est bien équivalent au primal en utilisant la définition et non le théorème

# Théorème de dualité faible

Considérons la paire de problème primal-dual suivante:

$$\begin{array}{ll}\text{minimiser} & \mathbf{c}^T \mathbf{x} \\ \text{sous les contraintes} & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}\end{array}$$

et

$$\begin{array}{ll}\text{maximiser} & \mathbf{y}^T \mathbf{b} \\ \text{sous les contraintes} & \mathbf{y}^T \mathbf{A} \leq \mathbf{c}^T\end{array}$$

Alors

$$\mathbf{y}^T \mathbf{b} \leq \mathbf{c}^T \mathbf{x}$$

# Théorème de dualité forte

Considérons la paire de problème primal-dual suivante:

minimiser  $\mathbf{c}^T \mathbf{x}$

sous les contraintes  $\mathbf{Ax} = \mathbf{b}$   
 $\mathbf{x} \geq \mathbf{0}$

maximiser  $\mathbf{y}^T \mathbf{b}$

sous les contraintes  $\mathbf{y}^T \mathbf{A} \leq \mathbf{c}^T$

Si un des deux problèmes admet une solution **optimale**, alors l'autre problème aussi admet une solution optimale et leurs **valeurs optimales** respectives coïncident.

# Logiciels

# Solveurs disponibles

## Libres

- **Coin-or Linear Programming (CLP)**: écrit en C++ en **2000**, sous license EPL (Eclipse Public License), interfaces R, Java et Matlab disponibles.
- **GNU Linear Programming Kit (GLPK)**: écrit en C, aussi en **2000**, sous license GPLv3 (GNU Public License version 3).

## Propriétaires

- **CPLEX**: disponible commercialement depuis 1988, créé par Robert E. Bixby, acquis par ILOG en 1997 et ensuite par IBM en 2009. Licence propriétaire, mais gratuit pour étudiants et enseignants en milieu académique. Peut être utilisé en C, C++, Java et Python.

## Exemple

Considérons le problème de programmation linéaire suivant:

$$\begin{array}{ll}\text{maximiser} & x_1 + 2x_2 + 3x_3 \\ \text{sous les contraintes} & -x_1 + x_2 + x_3 \leq 20 \\ & x_1 - 3x_2 + x_3 \leq 30 \\ & 0 \leq x_1 \leq 40 \\ & 0 \leq x_2 \\ & 0 \leq x_3\end{array}$$

(Tiré de la documentation de Cplex)



# Interpréteur Cplex

```
CPLEX> enter example
Enter new problem '['end on a separate line terminates]:
maximize x1 + 2 x2 + 3 x3
subject to -x1 + x2 + x3 <= 20
           x1 - 3 x2 + x3 <=30

bounds
0 <= x1 <= 40
0 <= x2
0 <= x3
end
CPLEX> optimize
[...]
CPLEX> display solution variables x1-x3
[...]
```

# Bibliothèque C++

```
#include <ilcplex/ilcplex.h>
ILOSTLBEGIN

int main (int argc, char **argv) {
    IloEnv env;
    try {
        IloModel model(env);
        IloNumVarArray vars(env);
        vars.add(IloNumVar(env, 0.0, 40.0));
        vars.add(IloNumVar(env));
        vars.add(IloNumVar(env));
        model.add(IloMaximize(env, vars[0] + 2 * vars[1] + 3 * vars[2]));
        model.add(- vars[0] + vars[1] + vars[2] <= 20);
        model.add( vars[0] - 3 * vars[1] + vars[2] <= 30);
        IloCplex cplex(model);
        if (!cplex.solve()) {
            env.error() << "Failed to optimize LP." << endl;
            throw(-1);
        }
        IloNumArray vals(env);
        env.out() << "Solution status = " << cplex.getStatus() << endl;
        env.out() << "Solution value = " << cplex.getObjValue() << endl;
        cplex.getValues(vals, vars);
        env.out() << "Values = " << vals << endl;
    }
    catch (IloException& e) {
        cerr << "Concert exception caught: " << e << endl;
    }
    catch (...) {cerr << "Unknown exception caught" << endl;}
    env.end();
    return 0;
}
```

# Bibliothèque C (1/4)

```
#include <ilcplex/cplex.h>
#include <stdlib.h>
#include <string.h>

#define NUMROWS 2
#define NUMCOLS 3
#define NUMNZ 6

int main(int argc, char **argv) {
    int status = 0;
    CPXENVptr env = NULL;
    CPXLPptr lp = NULL;

    double obj[NUMCOLS];
    double lb[NUMCOLS];
    double ub[NUMCOLS];
    double x[NUMCOLS];
    int rmatbeg[NUMROWS];
    int rmatind[NUMNZ];
    double rmatval[NUMNZ];
    double rhs[NUMROWS];
    char sense[NUMROWS];

    int solstat;
    double objval;
```

# Bibliothèque C (2/4)

```
env = CPXopenCPLEX(&status);
if (env == NULL) {
    char errmsg[1024];
    fprintf(stderr, "Could not open CPLEX environment.\n");
    CPXgeterrorstring(env, status, errmsg);
    fprintf(stderr, "%s", errmsg);
    goto TERMINATE;
}
lp = CPXcreateprob(env, &status, "lpex1");
if (lp == NULL) {
    fprintf(stderr, "Failed to create LP.\n");
    goto TERMINATE;
}

CPXchgobjsen(env, lp, CPX_MAX);

obj[0] = 1.0;  obj[1] = 2.0;          obj[2] = 3.0;
lb[0] = 0.0;   lb[1] = 0.0;          lb[2] = 0.0;
ub[0] = 40.0;  ub[1] = CPX_INFBOUND; ub[2] = CPX_INFBOUND;

status = CPXnewcols(env, lp, NUMCOLS, obj, lb, ub, NULL, NULL);
if (status) {
    fprintf(stderr, "Failed to populate problem.\n");
    goto TERMINATE;
}
```

# Bibliothèque C (3/4)

```
rmatbeg[0] = 0;
rmatind[0] = 0;    rmatind[1] = 1;    rmatind[2] = 2;    sense[0] = 'L';
rmatval[0] = -1.0; rmatval[1] = 1.0;  rmatval[2] = 1.0;    rhs[0] = 20.0;

rmatbeg[1] = 3;
rmatind[3] = 0;    rmatind[4] = 1;    rmatind[5] = 2;    sense[1] = 'L';
rmatval[3] = 1.0;  rmatval[4] = -3.0; rmatval[5] = 1.0;    rhs[1] = 30.0;

status = CPXaddrows(env, lp, 0, NUMROWS, NUMNZ, rhs, sense, rmatbeg,
                    rmatind, rmatval, NULL, NULL);
if (status) {
    fprintf(stderr, "Failed to populate problem.\n");
    goto TERMINATE;
}

status = CPXlpopt(env, lp);
if (status) {
    fprintf(stderr, "Failed to optimize LP.\n");
    goto TERMINATE;
}

status = CPXsolution(env, lp, &solstat, &objval, x, NULL, NULL, NULL);
if (status) {
    fprintf(stderr, "Failed to obtain solution.\n");
    goto TERMINATE;
}
```

# Bibliothèque C (4/4)

```
printf("\nSolution status = %d\n", solstat);  
printf("Solution value = %f\n", objval);  
printf("Solution          = [%f, %f, %f]\n\n", x[0], x[1], x[2]);
```

TERMINATE:

```
if (lp != NULL) {  
    status = CPXfreeprob(env, &lp);  
    if (status) {  
        fprintf(stderr, "CPXfreeprob failed, error code %d.\n", status);  
    }  
}  
  
if (env != NULL) {  
    status = CPXcloseCPLEX(&env);  
    if (status) {  
        char errmsg[1024];  
        fprintf(stderr, "Could not close CPLEX environment.\n");  
        CPXgeterrorstring(env, status, errmsg);  
        fprintf(stderr, "%s", errmsg);  
    }  
}  
  
return status;  
}
```