Fondements de la Recherche Opérationnelle

Prof. Christian Prins

Laboratoire d'Optimisation des Systèmes Industriels (LOSI)
Institut Charles Delaunay – UMR CNRS 6281

christian.prins@utt.fr

Objectifs du cours

Modéliser et résoudre des problèmes d'optimisation formulés sous forme de programmes mathématiques :

- Notion de programme mathématique
- Cas particulier des programmes linéaires
- Modèles génériques et démarche de modélisation
- Principes des logiciels de résolution (solveurs)
- Logiciels pour Excel : solveur d'Excel et Open Solver
- Logiciel avec langage de modélisation : cas de GUSEK.

TD : exercices de modélisation avec Excel et GUSEK.
Note : on ne voit pas les méthodes de résolution (cf. MT14).
Evaluation : devoir (50%) + examen final (50%).

Programmes mathématiques (PM)

Un PM est un modèle pour optimiser une fonction numérique de n variables (*fonction-objectif*) sous m contraintes. Notons :

- $\mathbb{x} = (x_1, x_2, \ldots, x_n)$ vecteur des n variables
- $S \subseteq \mathbb{R}^n$ domaine des variables, comme $S = \mathbb{R}^{n+}$
- $f(\mathbb{x})$ fonction à optimiser
- $g_i(\mathbb{x})$ fonction de contrainte numéro i

Tout programme mathématique peut s'écrire sous la forme :

$$\min \text{ ou } \max f(\mathbb{x})$$
$$\forall\, i \in \{1,2,..,m\} :\ g_i(\mathbb{x}) \leq, =, \geq 0$$
$$\mathbb{x} \in S$$

Programmes mathématiques

Remarques :

Les variables sont réelles, entières ou binaires (0-1).

L'objectif est unique (il existe des extensions multi-objectifs).

Pour éviter des problèmes numériques dans les logiciels :

- On suppose que f et les g_i sont continues et dérivables (une discontinuité peut causer une division par 0)
- Pas de contraintes < et > car les logiciels ont une précision de calcul limitée ("double x" en C = 15 chiffres significatifs). Ils ne peuvent pas distinguer entre $\leq$ et $<$ et entre $\geq$ et $>$.

Programmes mathématiques

Remarques (suite) :

Les contraintes simples (signe, intervalle) sont dans la définition de S, inutile de les mettre dans les $g_i(\mathbb{x})$.

Les logiciels mettent aussi à part les contraintes simples comme $x_1 \geq 0$ ou $2 \leq x_1 \leq 5$, car elles sont facilement gérées dans les algorithmes de résolution.

Notez qu'il existe d'autres types de modèles d'optimisation :

- les graphes (chemins optimaux etc.), cf. UV SY18 à l'UTT.
- la programmation dynamique, cf. UV MT14.

On peut parfois utiliser plusieurs types de modèles : certains problèmes de graphes peuvent se mettre sous forme de PM.

Programmes mathématiques

On appelle *solution réalisable* (SR) ou solution faisable une solution $\mathbb{x}$ qui satisfait toutes les contraintes.

Une SR $\mathbb{x}^*$ est *optimale* si aucune autre SR n'est meilleure. On dit que cette SR est un *optimum* (ce n'est pas $f(\mathbb{x}^*)$!).

Un PM peut avoir plusieurs optimums, et même une infinité.

Selon les contraintes et les valeurs des données, un PM peut aussi être *infaisable* (pas de SR) ou *ne pas avoir d'optimum fini* (rien ne limite la croissance ou décroissance de f).

Pour un problème réel, ces deux derniers cas indiquent une erreur dans le modèle ou dans les données.

Programmes mathématiques

Principaux types de programmes mathématiques :

- Programme linéaire (PL) : f et les g_i sont tous linéaires.
- PL à variables entières, dit "en nombres entiers" (PLNE).
- PL en 0-1 : cas particulier de PLNE à variables avec deux valeurs 0 ou 1 (variables dites *booléennes* ou *binaires*).
- PL mixte : on a à la fois des variables continues et entières.
- Programme non linéaire (PNL) : au moins une contrainte ou la fonction-objectif n'est plus linéaire (il suffit par exemple d'un produit de deux variables $x_1.x_2$).

Les PLNE et PL en 0-1 sont plus difficiles que les PL ordinaires.

Programmes linéaires (PL)

Exemple et ordre de conception

Une usine produit 2 ciments, rapportant 50$ et 70$/t. Pour 1 tonne de ciment 1, il faut 40 minutes de four et 20 minutes de broyage. Pour 1 tonne de ciment 2, 30 minutes et 30 minutes. Four et broyeur sont disponibles 6h et 8h par jour. Quelles quantités faire chaque jour pour maximiser le bénéfice ?

On définit d'abord les *variables* :

- x_1 et x_2 quantités de ciment 1 et 2 à fabriquer,
- domaine : réel (quantités continûment divisibles),
- signe : positif ou nul.

On écrit donc : $x_1, x_2 \geq 0$.

Programmes linéaires

Puis on formule les *contraintes* :

Les contraintes viennent souvent de ressources limitées :
argent, capacité des stocks, personnel, temps-machine etc.

On a ici deux ressources, un four et un broyeur.
Attention aux unités de temps mélangées : heures et minutes.

Capacités (disponibilités) : 360 et 480 minutes.
Une tonne de ciment 1 prend 40 min et 20 min.
Les x_1 et x_2 tonnes fabriquées vont prendre 40.x1 et 20.x2
Donc, contrainte de disponibilité du four : $40.x_1 + 30.x_2 \leq 360$.

Pour le broyeur, même raisonnement : $20.x_1 + 30.x_2 \leq 480$.

Programmes linéaires

On écrit enfin le *critère à optimiser*, ou *fonction-objectif* ou *fonction économique* ou *fonction de coût*.

En dernier car unique. Ici c'est le profit total : $50.x_1 + 70.x_2$
On ajoute Max car on veut le maximiser : Max $50.x_1 + 70.x_2$

Récapitulation du PL complet (noter l'ordre de *présentation*) :

Max $50.x_1 + 70.x_2$	$\rightarrow$ fonction-objectif, en premier
$40.x_1 + 30.x_2 \leq 360$	$\rightarrow$ contrainte de disponibilité du four
$20.x_1 + 30.x_2 \leq 480$	$\rightarrow$ contrainte de disponibilité du broyeur
$x_1, x_2 \geq 0$	$\rightarrow$ définition des variables, en dernier

Programmes linéaires

Comme tout est linéaire, un PL peut s'écrire :

$$(1) \quad \text{Max ou Min } z = \sum_{j=1}^{n} c_j x_j$$

$$(2) \quad \forall i = 1 \ldots m : \sum_{j=1}^{n} a_{ij} x_j \leq, = \text{ ou } \geq b_i$$

$$(3) \quad \forall j = 1 \ldots n : x_j \geq 0 \text{ (ou autres contraintes de domaine)}$$

- n nombre de variables, avec définition du domaine (3)
- m nombre de contraintes d'égalité ou d'inégalité (2)
- z fonction objectif à optimiser (1)
- c_j coefficient de coût ou de profit de la variable x_j
- a_{ij} coefficient de x_j dans la contrainte numéro i
- b_i second membre constant de la contrainte i.

Programmes linéaires

On peut même définir un PL avec quelques tableaux :

- $x = (x_1, x_2, \ldots, x_n)^T$ vecteur des variables
- $b = (b_1, b_2, \ldots, b_m)^T$ vecteur des seconds membres
- $c = (c_1, c_2, \ldots, c_n)$ vecteur des coûts ou profits
- A matrice $m \times n$ matrice des coefficients a_{ij}

Et le formuler par calcul matriciel, par exemple :

Max ou Min $c.x$
$A.x \leq b$
$x \geq 0$

Programmes linéaires

L'hypothèse de linéarité semble restrictive, mais les PL sont très répandus en optimisation industrielle. Les PNL ne sont fréquents qu'en chimie, mécanique et physique.

Les variables sont souvent des activités (quantités à produire, à transporter…). Les contraintes ≤ traduisent des ressources limitées (capacité de production ou de stockage…). Les ≥ servent pour un stock minimal, des seuils de rentabilité etc.

La linéarité résulte :

- de la *proportionnalité* des coûts et des consommations de ressources aux intensités des activités,
- de *l'additivité* des consommations de ressources (pas d'interactions entre activités).

Programmes linéaires

Evidemment, ces hypothèses sont parfois fausses :

- Le prix de revient d'un article diminue avec la quantité produite (économie d'échelle) : pas de proportionnalité.
- Si un nouveau supermarché peut attirer 10000 clients mais que deux nouveaux supermarchés sont trop proches, ils vont se "voler" des clients : pas d'additivité.

Cependant, beaucoup de problèmes qui semblent non linéaires peuvent se "linéariser", au prix de variables ou des contraintes supplémentaires.

Les PL sont donc très importants en pratique, d'autant plus que des logiciels très puissants sont disponibles.

Résolution graphique des PL

Possible si n=2 ou 3 variables. Exemple :

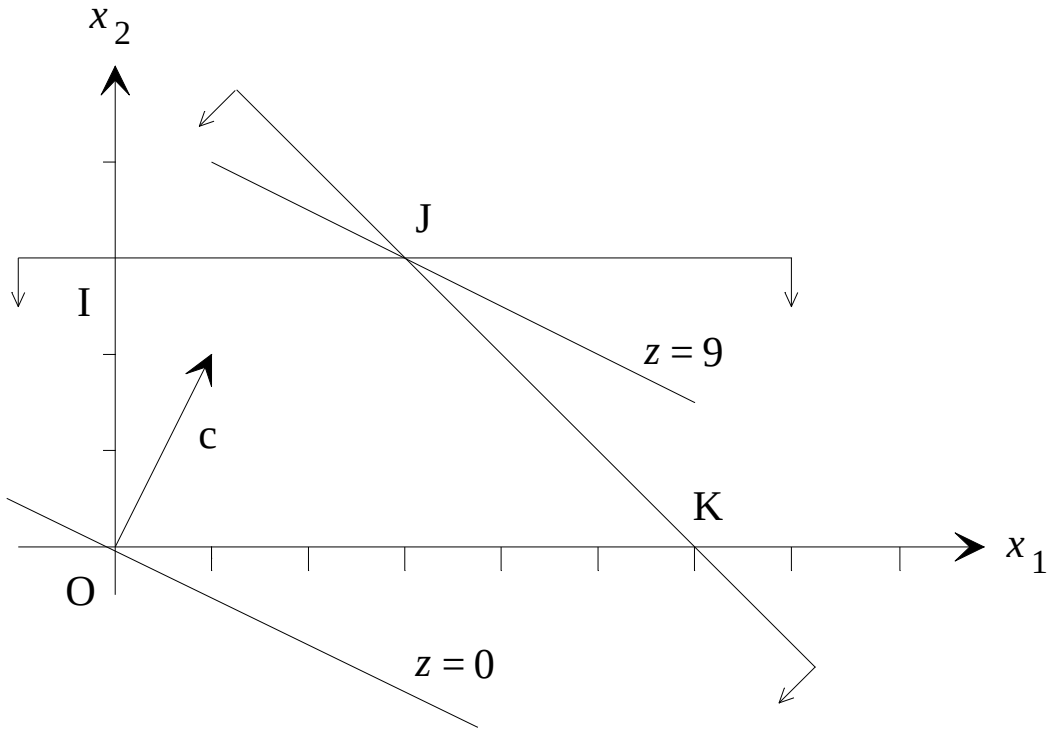
$$\begin{aligned} \text{Max } z = \quad & x_1 + 2 \cdot x_2 \\ & x_1 + \quad x_2 \leq 6 \\ & \quad\quad x_2 \leq 3 \\ & x_1, \quad\quad x_2 \geq 0 \end{aligned}$$

On a ici $m = 2$, $n = 2$, $c = (1,2)$, $x = (x_1, x_2)^T$, $b = (6,3)^T$ et

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Résolution graphique des PL

On trace 2 axes pour x_1 et x_2. Puis les droites d'équations $x_1+x_2 = 6$ et $x_2 = 3$. Les petites flèches donnent le demi-plan conservé. On obtient le *domaine des SR*, le polygone OIJK.



Résolution graphique des PL

Rappel. Pour une fonction $f(\mathbb{x})$ quelconque, $\nabla f(\mathbb{x})$ est le gradient ou vecteur des dérivées partielles. Par exemple, si $f(\mathbb{x}) = 2.x_1.x_2^3 + \sin(x_2)$, $\nabla f(\mathbb{x}) = (2.x_2^3, 6.x_1.x_2^2 + \cos(x_2))$.

Rappel. En tout point $\overline{\mathbb{x}}$ du domaine de définition de f, $\nabla f(\overline{\mathbb{x}})$ donne la direction de plus forte augmentation de f.

Pour un PL, $\nabla f(\mathbb{x}) = c$ pour tout $\mathbb{x}$: le gradient est constant!

Donc, en se déplaçant dans la direction du gradient c, on trouve le maximum : c'est le point J : $x_1 = 3$, $x_2 = 3$.

On note souvent avec des * la solution optimale et son coût : $x^* = (3,3)$, $z^* = 9$. En remplaçant les variables par leurs valeurs dans le PL, on peut vérifier le coût et les contraintes.

Propriétés des PL valables pour tout *n*

L'intersection des domaines définis par chaque contrainte forme un *polyèdre convexe* (un *polygone convexe* si n=2).

Rappel. Un ensemble S de IR^n est *convexe* si pour tout couple de points A et B dans S, le segment [A,B] est contenu dans S.

L'optimum x^* est atteint en un *sommet* du polyèdre.

Donc, bien que le polyèdre contienne un nombre infini de SR, il suffit de tester les sommets du polyèdre, en nombre fini, pour trouver l'optimum!

Cet énorme avantage de la PL est exploité par un algorithme très efficace, l'algorithme du simplexe.

Cas particuliers de PL

En supprimant $x_1 + x_2 \leq 6$, le domaine des solutions réalisables serait *non borné*, ainsi que l'optimum.

Si en plus $c = (-1,2)$, on obtient le point I : donc l'optimum peut être fini même si le domaine des SR est non borné.

Si on ajoute la contrainte $x_1 - x_2 \leq -4$ dans le PL de départ, le polyèdre devient *vide* et il n'y a aucune solution réalisable.

Enfin, si on maximisait $z = x_1 + x_2$, il y aurait *plusieurs optima* : tous les points de l'arête JK.

Les algorithmes d'optimisation ne trouvent qu'un optimum. L'algorithme du simplexe trouvera donc J ou bien K.

Cas particuliers de PL

Pour un problème réel (exemple, un plan de production), un domaine vide est anormal car il y a au moins une solution : le plan *actuel* de l'entreprise. De plus, l'optimum est *borné en pratique* à cause des *ressources limitées*.

L'infaisabilité vient souvent d'un problème trop contraint ou d'une "inversion de contrainte" ($\geq$ au lieu de $\leq$).

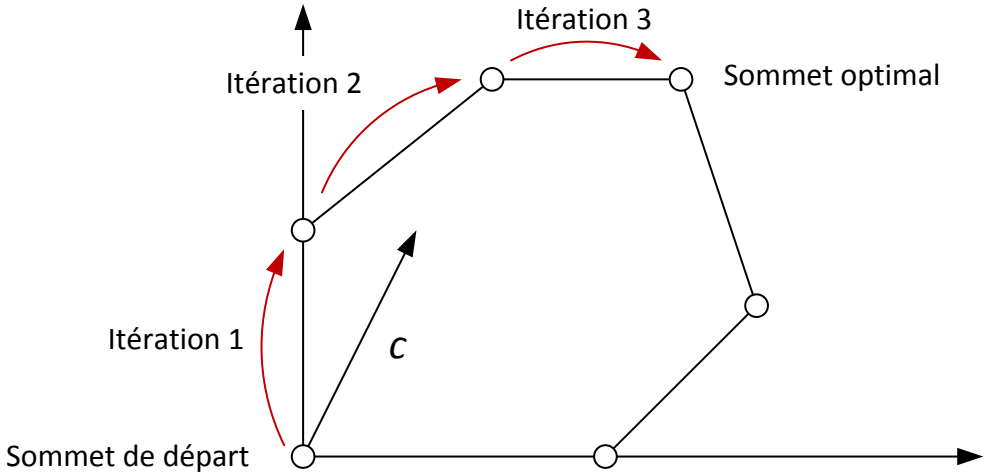
L'oubli d'une contrainte peut donner un optimum non borné.

Une *erreur de saisie* du modèle dans un logiciel peut aussi produire ces phénomènes. Il faut alors revoir la formulation.

Difficulté des PL

En fait, les PL sont faciles à résoudre : les contraintes forment un polyèdre et l'optimum se trouve parmi les sommets.

L'algorithme du simplexe utilisé dans les logiciels commerciaux a déjà résolu des PL à 1 million de variables. Il consiste à visiter des sommets adjacents en cherchant à augmenter la fonction-objectif (en maximisation).



Algorithme du simplexe

Bases et solutions de base

Soit un PL en forme standard avec un système $A.x = b$.
Rappel : A matrice $m \times n$, x vecteur $n \times 1$, b vecteur $m \times 1$.

On suppose qu'il n'y a pas de contrainte redondante : le *rang* de A vaut m et $m \leq n$.

On appelle *base* de A, ou *matrice de base*, toute sous-matrice carrée inversible $m \times m$ de A.

Soit une base B : on peut écrire $A = (B \mid N)$ et $x = (x_B \mid x_N)$:

- N : *matrice hors-base* $m \times (n\text{-}m)$ des colonnes hors-base.
- x_B : m *variables de base* associées aux colonnes de B.
- x_N : $n\text{-}m$ autres variables, appelées *variables hors-base*.

Algorithme du simplexe

Bases et solutions de base (fin)

Pour une base *B* choisie, on peut écrire le PL en exprimant les variables de base en fonction des hors-base :

$$A \cdot x = b \Leftrightarrow B \cdot x_B + N \cdot x_N = b \Leftrightarrow x_B = B^{-1} \cdot b - B^{-1} \cdot N \cdot x_N$$

$$z = c \cdot x = c_B \cdot x_B + c_N \cdot x_N = c_B \cdot B^{-1} \cdot b + \left(c_N - c_B \cdot B^{-1} \cdot N\right) \cdot x_N$$

Solution évidente si $x_N = 0$: $x_B = B^{-1}.b$ et $z = c_B.B^{-1}.b$.
C'est la *solution de base* (SB) associée à la base *B*.

Elle peut violer des contraintes de positivité :
on a une *solution de base réalisable* (SBR) si en plus $x_B \geq 0$.

Algorithme du simplexe

Exemple d'énumération des bases

Mettons en forme standard le PL de la résolution géométrique, avec des *variables d'écart* x_3 et x_4 (une par contrainte) :

$$\begin{aligned} \text{Max } z = \; & x_1 + 2 \cdot x_2 \\ & x_1 + x_2 + x_3 = 6 \\ & x_2 + x_4 = 3 \\ & x_1, \; x_2, \; x_3, \; x_4 \geq 0 \end{aligned}$$

En notant a_i la colonne i de A, la matrice 2×4 du PL est :

$$(a_1, a_2, a_3, a_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Algorithme du simplexe

Exemple d'énumération des bases (suite)

Toutes les sous-matrices 2 × 2 sont inversibles, sauf (a_1, a_3).
Les solutions de base associées sont réalisables, sauf B_4.

$$B_1 = (a_1, a_2) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$B_2 = (a_1, a_4) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_1 \\ x_4 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \end{pmatrix}$$

$$B_3 = (a_2, a_3) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

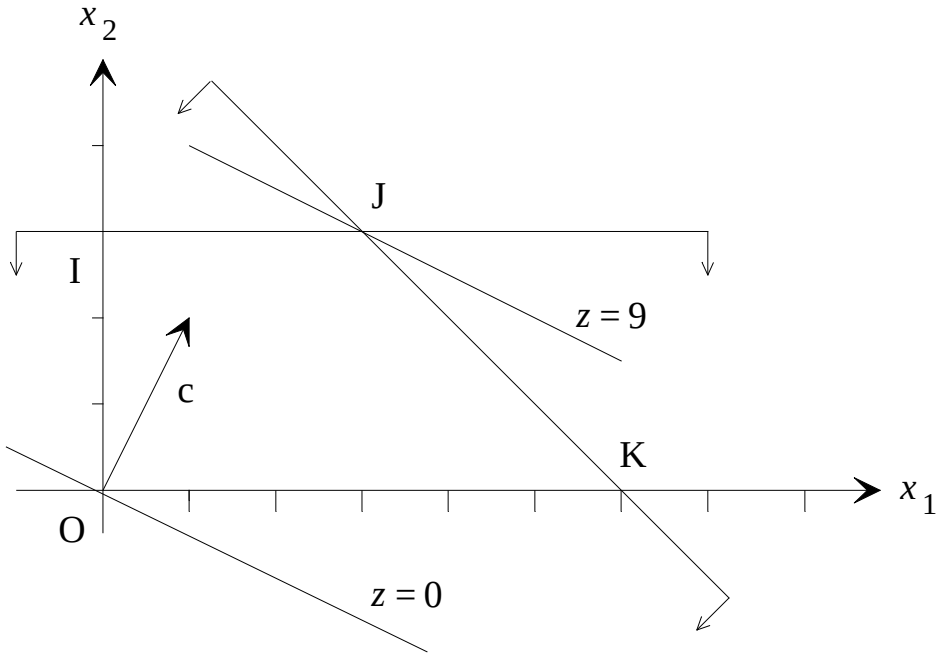
$$B_4 = (a_2, a_4) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_2 \\ x_4 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ -3 \end{pmatrix}$$

$$B_5 = (a_3, a_4) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \end{pmatrix}$$

Algorithme du simplexe

Exemple d'énumération des bases (suite)

La forme standard a un polyèdre à 4 dimensions mais celui de la forme canonique (résolution graphique) est une projection à 2 dimensions, sans les variables d'écart. *On constate que les SBR correspondent aux sommets J, K, I et O du polyèdre*.



Algorithme du simplexe

Calcul sans notation matricielle

On exprime les *m* variables de base en fonction des autres.
Par exemple, pour la SBR associée à $B_1 = (a_1,a_2)$ ou point J, on part du système $A \cdot x = b$:

$$\begin{aligned} x_1 + x_2 + x_3 &= 6 \\ x_2 + x_4 &= 3 \end{aligned}$$

On exprime les variables de base x_1, x_2 en fonction des autres

$$\begin{aligned} x_1 &= 6 - x_2 - x_3 = 3 - x_3 + x_4 \\ x_2 &= 3 - x_4 \end{aligned}$$

En mettant les variables hors-base x_3 et x_4 à 0,
on retrouve la SBR $x_1 = 3$ et $x_2 = 3$, correspondant au point J.

Algorithme du simplexe

Opération de pivotage

Les bases pour 2 sommets adjacents du polyèdre ont *m*-1 variables communes. On peut passer d'une base à l'autre sans inversion de matrice : opération appelée *pivotage*.

Pivotage de B_5 (x_3 et x_4, point O) à B_3 (x_2 et x_3, point I).

$$\begin{aligned} x_3 &= 6 - x_1 - x_2 \\ x_4 &= 3 - x_2 \end{aligned} \Rightarrow \begin{aligned} x_3 &= 6 - x_1 - (3 - x_4) \\ x_2 &= 3 - x_4 \end{aligned} \Rightarrow \begin{aligned} x_3 &= 3 - x_1 + x_4 \\ x_2 &= 3 - x_4 \end{aligned}$$

On doit remplacer x_4 par x_2 dans la base :

a) x_2 passe dans le 1er membre de l'équation pour x_4.
b) on élimine x_2 dans les autres équations.

Algorithme du simplexe

Algorithme du simplexe (Dantzig, 1947)

Cet algorithme construit une suite de SBR de profit croissant (maximisation), jusqu'à ce qu'il n'y ait plus de gain possible.

Géométriquement, on visite une suite de sommets adjacents du polyèdre.

Le passage d'une base à l'autre se fait par pivotage.

Souvent, on part d'un PL avec contraintes $\leq$ et $b \geq 0$.
En ajoutant les variables d'écart, on obtient une base initiale évidente (matrice identité) correspondant au point O (x=0).

Algorithme du simplexe

PL de la résolution géométrique, mis en forme standard :

$$
\begin{aligned}
\text{Max } z = \; & x_1 + 2 \cdot x_2 \\
& x_1 + x_2 + x_3 = 6 \\
& x_2 + x_4 = 3 \\
& x_1, \; x_2, \; x_3, \; x_4 \geq 0
\end{aligned}
$$

On exprime x_3 et x_4 en fonction de x_1 et x_2, même dans la fonction-objectif qui est traitée comme une pseudo-équation :

$$
\begin{aligned}
x_3 &= 6 - x_1 - x_2 \\
x_4 &= 3 - x_2 \\
z &= 0 + x_1 + 2x_2
\end{aligned}
$$

SBR initiale associée. On met les variables hors-base à 0 : x = (0, 0, 6, 3) et z = 0. Point O de la résolution géométrique.

Algorithme du simplexe

Pour changer de SBR, on inspecte les coefficients des variables hors-base (*profits marginaux* ou *profits réduits*) dans z.

Critère heuristique : la variable hors-base x_e choisie pour entrer en base (*variable entrante*) est celle de coût réduit > 0 maximum (en maximisation).

On augmente x_e jusqu'à ce qu'une variable de base x_s s'annule. On élimine x_s de la base (*variable sortante*).

Géométriquement, on passe ainsi à un sommet adjacent du polyèdre.

Algorithme du simplexe

$$x_3 = 6 - x_1 - x_2$$
$$x_4 = 3 - x_2$$
$$z = 0 + x_1 + 2x_2$$

x_e est la variable hors-base de profit réduit maximal : x_2.

Contrainte 1: x_2 peut augmenter jusqu'à 6 et x_3 s'annule.
Contrainte 2: x_2 peut augmenter jusqu'à 3 et x_4 s'annule.

x_s est la variable qui s'annule en premier : x_4.

x_2 ne peut pas dépasser 3 sinon x_4 devient négative !

Algorithme du simplexe

Les nouvelles variables de base sont donc x_3 et x_2, et on doit les écrire, ainsi que z, uniquement en fonction des nouvelles variables hors-base x_1 et x_4. On obtient :

$$x_2 = 3 - x_4$$
$$x_3 = 6 - x_1 - x_2 = 6 - x_1 - (3 - x_4) = 3 - x_1 + x_4$$
$$z = 0 + x_1 + 2 \cdot (3 - x_4) = 6 + x_1 - 2x_4$$

La SBR actuelle se lit en mettant à 0 les variables hors base : $x_2 = 3$, $x_3 = 3$, $x_1 = x_4 = 0$, avec un profit $z = 6$ (point I).

Algorithme du simplexe

Pour augmenter le profit, on peut seulement augmenter x_1, seule variable hors-base de profit réduit > 0.

x_1 peut augmenter jusqu'à 3 et remplace dans la base x_3, qui s'annule. On écrit donc x_1, x_2 et z en fonction de x_3 et x_4 :

$$x_1 = 3 - x_3 + x_4$$

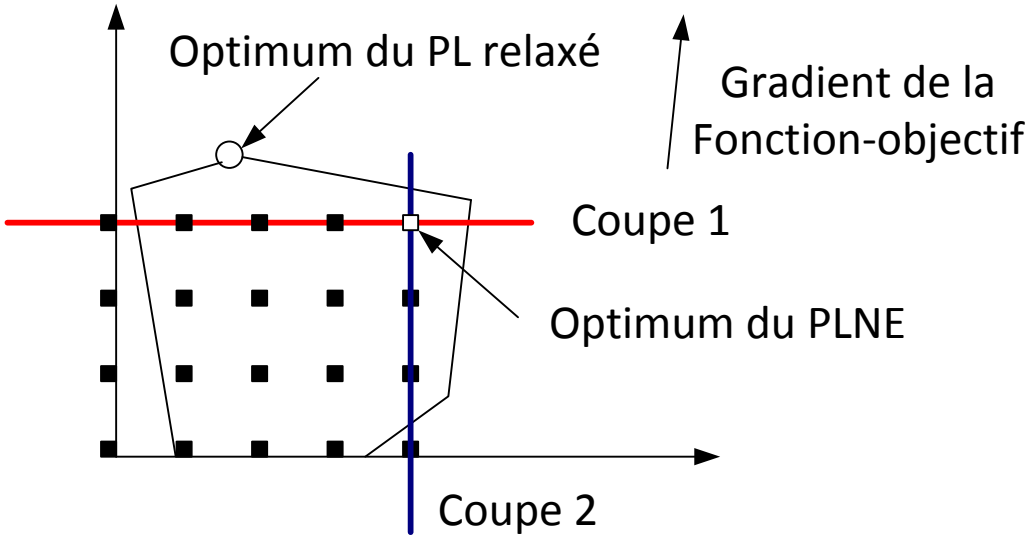
$$x_2 = 3 - x_4$$

$$z = 6 + (3 - x_3 + x_4) - 2x_4 = 9 - x_3 - x_4$$

La SBR associée est x = (3,3,0,0), de profit 9 (point J).
Optimum, car les variables hors-base ont un profit réduit < 0.

Difficulté des PLNE

On appelle *PL relaxé* (PLR) le PLNE sans la contrainte $\mathbb{x} \in \mathbb{N}^n$.
Les SR du PLNE sont des points à coordonnées entières.
Si l'optimum du PLR (un sommet du polyèdre) n'a pas de coordonnées entières, il faut voir à l'intérieur du polyèdre : donc, l'algorithme du simplexe est inutilisable!



Difficulté des PLNE

La résolution est difficile si $n > 2$ ou 3 (pas de graphique):

- Arrondir l'optimum du PL relaxé ne marche pas!
- On peut avoir un polyèdre non vide mais sans SR entière!

Les algorithmes disponibles n'utilisent aucune variable entière. Ils résolvent d'abord le PL relaxé : s'il a des coordonnées entières, c'est aussi l'optimum du PLNE.

Il existe des PLNE dont les polyèdres ont des sommets à coordonnées entières : ils ne sont pas plus difficiles que les PL simples car il suffit de résoudre leur PL relaxé.

Le *problème de transport* est un exemple célèbre.

Difficulté des PLNE

Si le PL relaxé admet un optimum non entier, on ajoute une contrainte appelée *coupe*, qui "rogne" le polyèdre mais sans enlever des solutions entières. Puis on résout le PL obtenu.

Dans l'exemple, on peut trouver l'optimum en ajoutant seulement deux coupes et en résolvant trois PL.

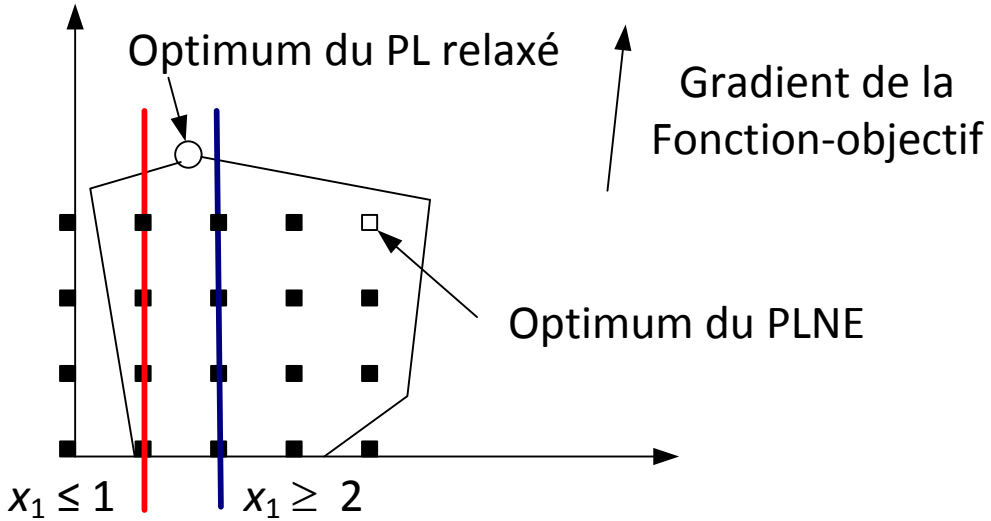
Si n est grand, trouver des coupes puissantes n'est pas évident, car on ne peut pas visualiser le polyèdre.

De plus, on doit parfois ajouter un grand nombre de coupes, avec un PL à résoudre à chaque fois : la résolution peut être très longue à partir de quelques centaines de variables.

Les PLNE sont donc plus difficiles à résoudre que les PL.

Difficulté des PLNE

Algorithme de Dakin. Résoudre le PLR, choisir une variable non entière x et séparer en 2 PL : un avec la coupe $x \leq \lfloor x \rfloor$, l'autre avec $x \geq \lceil x \rceil$. Ci-dessous, $x_1^* \approx 1.5$ donc $x_1 \leq 1$ et $x_1 \geq 2$.



En continuant à séparer, on obtient une arborescence de PL. Elle peut être énorme et causer des problèmes de mémoire.

Difficulté des PL en 0-1

Les PL en 0-1 sont des cas particuliers de PLNE, mais ils sont plus faciles car chaque variable ne vaut que 0 ou 1.

On peut même les résoudre en énumérant tous les vecteurs $\mathbb{x}$ possibles, mais il y en a 2^n. Pour $n = 20$, $2^{20} \approx 10^6$. Pour $n = 30$, $2^{30} \approx 10^9$: le temps de calcul croît de manière exponentielle!

En pratique, on fait comme dans la méthode de Dakin : résolution du PLR, choix d'une variable fractionnaire x et test des deux cas d'arrondi $x \leq 0$ et $x \geq 1$. En fait, $x = 0$ ou $x = 1$.

x est donc remplacée par sa valeur 0 ou 1 : cette variable disparaît et on obtient deux PL plus petits. La méthode est donc plus simple et plus rapide que pour un PLNE.

Difficulté des PNL

Les PNL sont les PM les plus durs. Même si on suppose que f et les g_i sont continues et dérivables, la fonction-objectif et les contraintes peuvent avoir des expressions compliquées.

Le domaine des SR peut avoir n'importe quelle forme, par exemple une région délimitée par des arcs de courbe.

La direction d'amélioration donnée par le gradient $\nabla f(\mathbb{x})$ de la fonction-objectif change selon le point considéré.

La fonction-objectif peut atteindre son optimum sur un sommet du domaine des SR, sur un bord ou à l'intérieur.

En général, on a plusieurs optima locaux.

Les logiciels ne trouvent en général qu'un seul optimum local.

Difficulté des PNL

Un bon cas – Objectif et contraintes convexes :

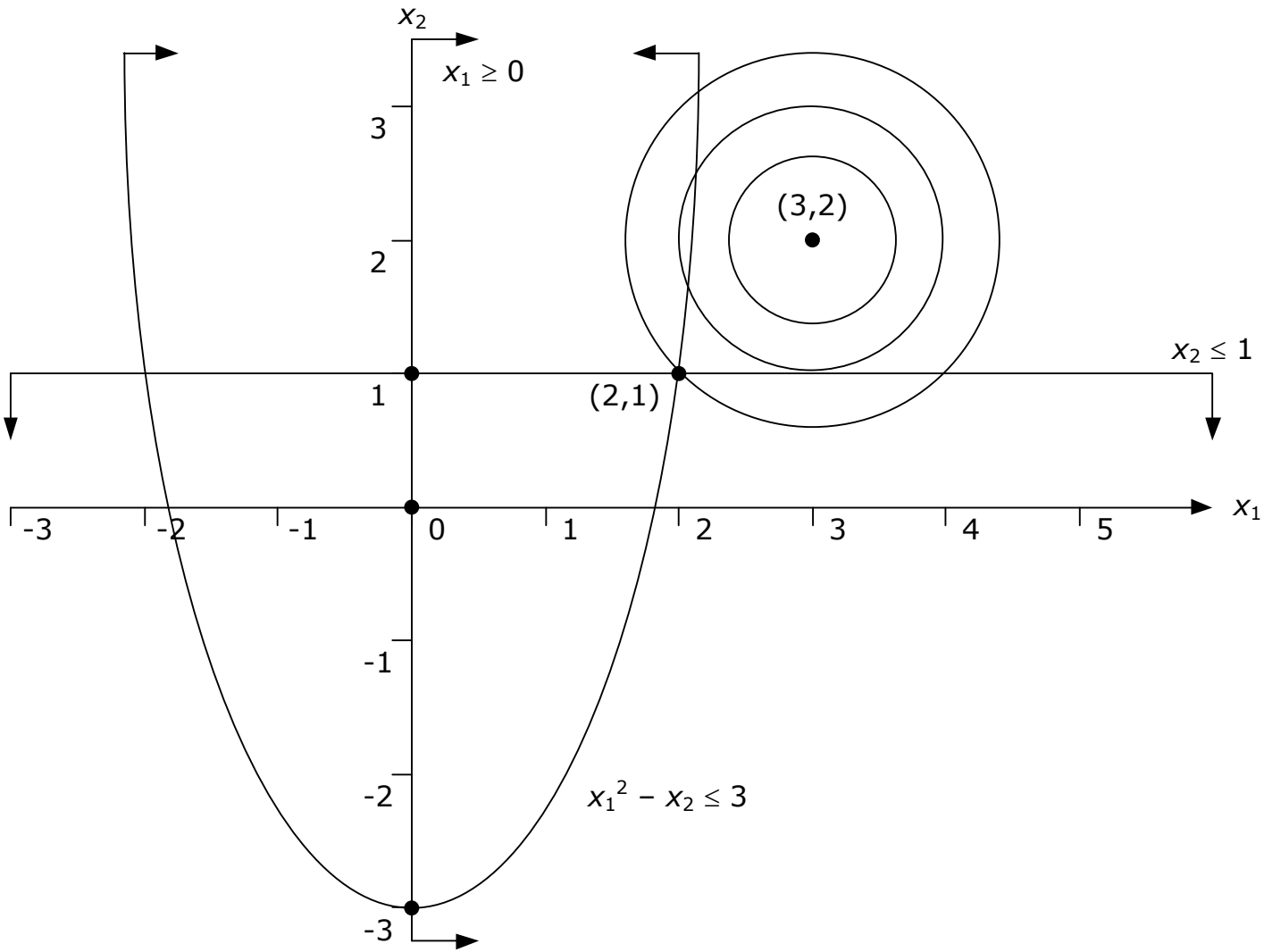
(1) $\min\ f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 2)^2$
(2) $x_1^2 - x_2 \leq 3$
(3) $x_2 \leq 1$
(4) $x_1 \geq 0$

Résolution graphique (slide suivante) :

- les courbes de niveau de f sont des cercles de centre (3,2)
- la contrainte (2) définit une région à frontière parabolique.
- (3) et (4) sont linéaires et forment une bande horizontale.

On doit trouver le plus petit cercle de centre (3,2) ayant une intersection avec le domaine des SR : point (2,1), de coût 2.

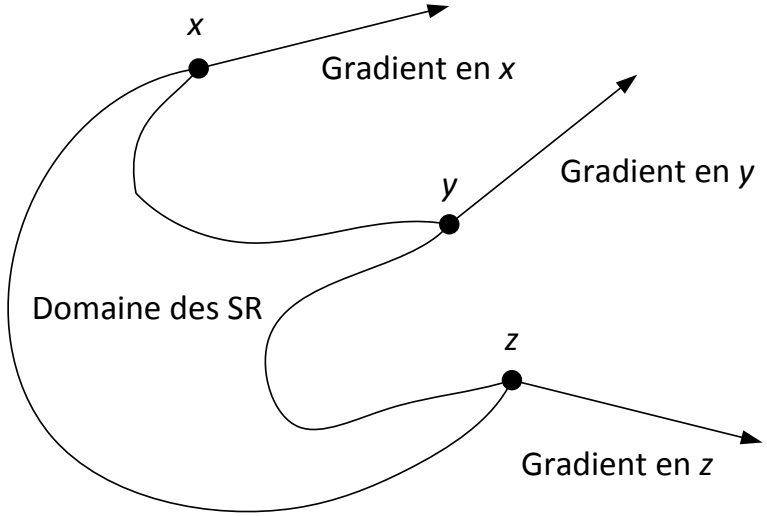
Difficulté des PNL



Difficulté des PNL

L'exemple est un "PL quadratique" : la non linéarité est limitée à des produits de 2 variables. Ces PNL sont plus faciles que les autres. De plus, son objectif et ses contraintes sont convexes : si on trouve un optimum local, il est aussi global.

Si l'objectif ou certaines contraintes ne sont pas convexes, on a en général plusieurs optima locaux.



Difficulté des PNL

Les algorithmes pour la PNL sont nombreux et compliqués.

Par exemple, en partant d'une solution réalisable x^0 :

- on calcule le gradient $\nabla f(x^0)$
- on cherche un minimum x^1 sur la droite-support du gradient
- attention à ne pas sortir du domaine des SR!
- on recommence à partir du point obtenu x^1
- on stoppe à l'itération k si $\left\|x^k - x^{k-1}\right\| < \varepsilon$ (précision fixée).

Les difficultés numériques sont fréquentes, même pour trouver un point initial, et on n'aboutit qu'à un optimum local.

Pour tenter d'avoir l'optimum global, on peut diviser le domaine des SR en mailles et résoudre un PNL par maille.

Démarche de modélisation

Avant d'utiliser un logiciel, il faut modéliser le problème à traiter sous forme de programme mathématique (PM).

Comme les PNL sont très difficiles, il faut essayer de trouver (si possible) un modèle linéaire (PL, PLNE, PL en 0-1).

Le modèle doit être le plus générique possible, c'est-à-dire :

- tout doit être paramétré
- les données et variables doivent être groupées en tableaux
- les contraintes de même nature doivent être regroupées.

On obtient ainsi un modèle :

- indépendant des données
- lisible, compact, facile à programmer et à modifier.

Démarche de modélisation

PL non générique pour les deux ciments (slide 8) :

Max $50.x_1 + 70.x_2$ → profit total, à maximiser
$40.x_1 + 30.x_2 \leq 360$ → contrainte de disponibilité du four
$20.x_1 + 30.x_2 \leq 480$ → contrainte de disponibilité du broyeur
$x_1, x_2 \geq 0$ → définition des variables

Le PL change si par exemple :

- on modifie des données (profits, disponibilité etc.)
- on ajoute d'autres types de ciments
- on ajoute d'autres ressources (unité d'ensachage).

Il devient énorme pour un grand nombre de ciments et de ressources.

Démarche de modélisation

Méthode pour arriver à un modèle générique :

1. Analyser et structurer les données disponibles

a) Identifier chaque entité (ensemble d'objets de même nature), choisir un symbole pour son cardinal et un indice :

- ensemble de $n = 2$ types de ciments, indicés par j
- ensemble de $m = 2$ ressources, indicées par i.

b) Données dépendant d'une seule entité, avec leurs unités :

- chaque ciment j a un profit par tonne p_j (en euros)
- chaque ressource i a une disponibilité b_i (en heures).

On obtient un ensemble de vecteurs (tableaux à 1 dimension).

Démarche de modélisation

c) Données dépendant de plusieurs entités (matrices) :

- le ciment j nécessite a_{ij} heures par tonne de ressource i.

d) Paramétrer les données non associées à un ensemble :

On n'en a pas ici. Par exemple, il vaut mieux définir une constante tva au lieu de mettre des 20% dans le modèle.

2. Définir les variables et les grouper en tableaux

Préciser les domaines et les unités. Ici les variables sont suggérées : quelles quantités fabriquer pour chaque ciment? On définit donc une variable $x_j \geq 0$ pour la quantité de ciment j (en tonnes), ce qui donne un vecteur x.

Démarche de modélisation

3. Définir les contraintes et les grouper (quantificateur)

On a un seul type de contraintes : les capacités des ressources. Pour la ressource i :

$$\sum_{j=1}^{n} a_{ij}.x_j \leq b_i$$

Noter qu'avec la somme, l'expression ne s'allonge pas avec le nombre de ciments. On groupe ces contraintes avec un ∀ :

$$\forall\, i = 1 \ldots m : \sum_{j=1}^{n} a_{ij}.x_j \leq b_i$$

Si on n'arrive pas à exprimer une contrainte, on a oublié des données ou des variables : revenir aux étapes précédentes.

Démarche de modélisation

4. Définir la fonction-objectif et le sens de l'optimisation

Ici l'objectif à maximiser est le profit total en euros. Là encore on utilise une somme pour avoir une formule courte :

$$\max z = \sum_{j=1}^{n} c_{ij} . x_j$$

5. Récapituler le PM complet dans l'ordre classique

C'est-à-dire : objectif, contraintes, définition des variables.

Sur un PL complexe, il est normal de faire plusieurs itérations de la méthode car on oublie toujours quelque chose.

Le modèle mathématique obtenu est assez facile à traduire dans n'importe quel logiciel d'optimisation

Logiciels – Les solveurs

Un *solveur* est un ensemble d'algorithmes d'optimisation, programmés sous forme d'une bibliothèque de fonctions à appeler dans des programmes en C, C++, Java, Matlab etc.

Il faut écrire un programme appelant et donner le PM à résoudre sous forme numérique (non générique). Certains solveurs utilisent Excel comme programme appelant.

Utiliser un solveur est la méthode la plus rapide pour résoudre un PM. On l'utilise dans certains logiciels de gestion de production et de transport.

Par contre, le temps de développement est assez long et le risque d'erreur (gestion du PM numérique) non négligeable.

Logiciels – Les solveurs

On distingue des solveurs de PL-PLNE-PL en 0-1, exemples :

- Cplex, Gurobi, Xpress (les plus performants, mais chers)
- Open Solver pour Excel (gratuit)
- GLPK (ressemble à Cplex mais gratuit)
- le solveur linéaire utilisé dans le modeleur LINGO
- le solveur linéaire de Matlab
- le solveur linéaire d'Excel (max $n = 300$ et $m = 100$).

Et des solveurs spécifiques pour les PNL, exemples :

- Knitro (le plus performant, mais cher)
- Lancelot
- Minos
- les solveurs non linéaire de LINGO et d'Excel.

Les langages de modélisation

Plus récents que les solveurs, ils permettent d'écrire des PM génériques : l'utilisateur peut définir des tableaux de données et de variables, et écrire des équations sous forme symbolique (avec des sommes sur des indices).

Principaux langages :

- AMPL, reconnu par de nombreux logiciels
- GMPL, sous-ensemble de AMPL utilisé avec le solveur GLPK
- OPL Studio, utilisé pour Cplex
- MOSEL, utilisé pour Xpress
- GAMS
- MPL
- LINGO.

Les langages de modélisation

Exemple de traduction de contraintes en AMPL et GMPL :

$\forall\, i = 1 \ldots m : \sum_{j=1}^{n} a_{ij}.x_j \leq b_i$ (ressources pour les ciments)

```
ressource {i in 1..m}: sum {j in 1..n} a[i,j]*x[j] <= b[i];
```

On remarque que :

- le langage utilise du texte simple disponible sur les claviers (pas de lettres grecques, de petites lettres en indice etc.).
- le quantificateur devient un nom de contraintes, indicé.

Un PM après traduction a plus de lignes, car il faut aider le logiciel en définissant les dimensions des tableaux, le type de leurs éléments (entiers, réels…), les variables etc.

Les langages de modélisation

A partir du modèle écrit dans un langage et des tableaux de données (pouvant être mis dans des fichiers), un programme compilateur (très complexe) analyse la syntaxe et génère le PM numérique dans un fichier.

Il faut ensuite un solveur pour résoudre le PM obtenu.

Les langages de modélisation permettent de développer et de tester rapidement des modèles mathématiques.

Certains langages sont la propriété d'une société de logiciels et ils ne peuvent appeler que le solveur de la société (OPL Studio appelle Cplex). Les compilateurs de GAMS et AMPL, par exemple, peuvent appeler différents solveurs.

Les modeleurs

Ces logiciels très puissants sont des environnements de développement de programmes mathématiques.

Ils combinent :

- un éditeur pour saisir les modèles dans un langage de modélisation et les sauver dans des fichiers
- un compilateur pour analyser la syntaxe du modèle et générer le PM numérique dans un fichier temporaire
- un solveur pour résoudre le PM numérique
- les résultats sont récupérés dans une fenêtre de l'éditeur.

Ces logiciels ont parfois le même nom que le langage de modélisation (GAMS, OPL Studio…).

Les modeleurs

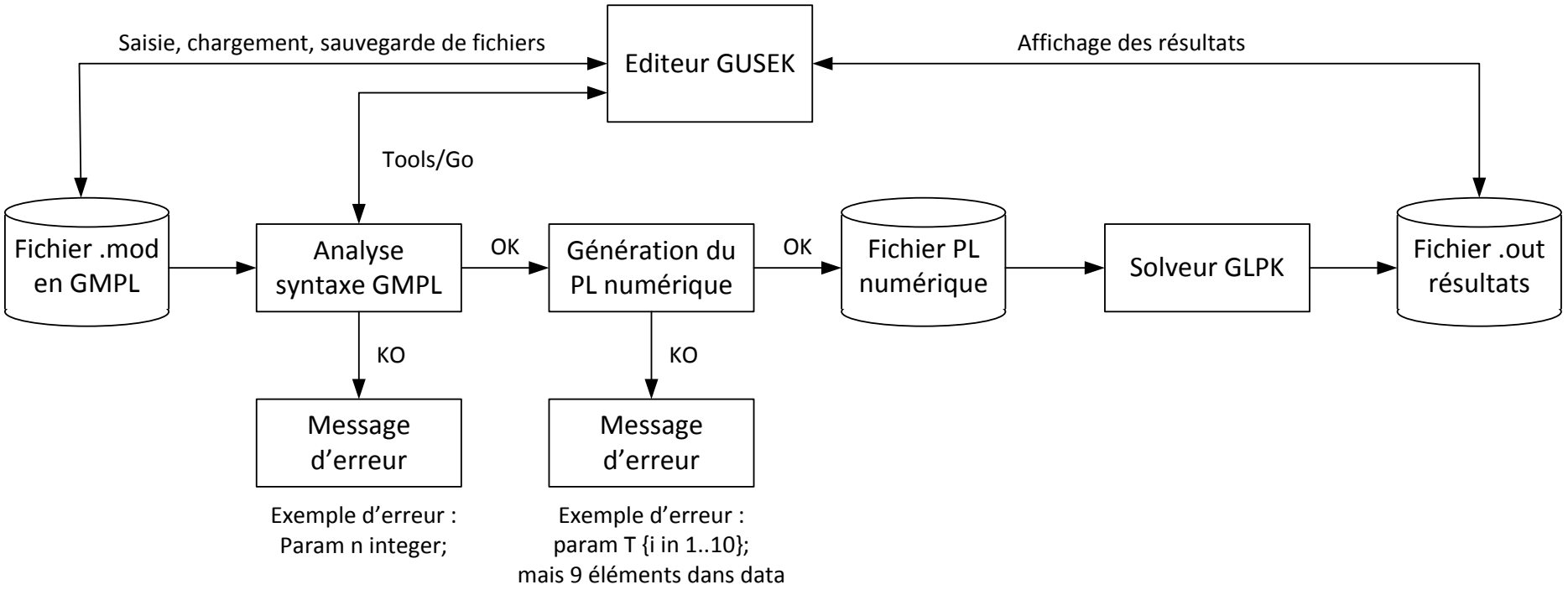
Principaux modeleurs :

- GAMS
- LINGO
- MPL
- Xpress-IVE (langage Mosel, solveur Xpress)
- OPL Studio (solveur Cplex)
- GUSEK (langage GMPL et solveur GLPK).

Modeleurs basés sur Excel (les équations sont des formules en Excel et les tableaux deviennent des plages de cellules) :

- Excel avec son solveur bridé (max $n = 300$ et $m = 100$)
- Excel avec le solveur Open Solver
- AIMMS, environnement très riche basé sur Excel.

Exemple – Architecture de GUSEK



Modeleurs gratuits et non bridés

Les seuls modeleurs gratuits et non limités sont actuellement GUSEK et Open Solver. Mais ils ne font pas de PNL.

Pour GUSEK, voir http://gusek.sourceforge.net
J'ai mis l'exécutable et une notice d'utilisation sur MOODLE.

Pour Open Solver, voir http://www.opensolver.org
Ce solveur prend la place du solveur d'Excel et s'utilise de la même façon (même interface).

A ma connaissance, il n'y a pas encore de produit équivalent pour la PNL.

Les autres modeleurs

Les autres modeleurs sont chers, environ 6000 €.

Cependant, il existe des versions d'évaluation gratuites, bridées en taille de PL ou limitées dans le temps.

Sous conditions, les étudiants et les universitaires peuvent souvent obtenir des versions non limitées, gratuites ou à tarif réduit, à des fins d'enseignement et de recherche.

Ainsi, OPL Studio et Cplex sont gratuits pour les enseignants-chercheurs des universités et les étudiants de doctorat.

Pour 1000 €, on peut avoir une licence de Xpress pour la recherche + 12 licences limitées en taille pour une salle de TP.