Méthodes d’optimisation

Programmation linéaire - Glpk

Introduction à la programmation linéaire

En recherche opérationnelle (RO), modéliser un problème consiste à identifier :

- Les **variables** intrinsèques (inconnues)
- Les différentes **contraintes** auxquelles sont soumises ces variables
- L'**objectif** visé (optimisation)

Dans un problème de programmation linéaire (**PL**) les contraintes et l'objectif sont des fonctions **linéaires** des variables. On parle aussi de *programme linéaire*.

Introduction à la programmation linéaire

Exemple d'un problème de production

Une usine fabrique deux produits P1 et P2 nécessitant des ressources d'équipement, de main d'œuvre et de matières premières disponibles en quantité limitée.

	P1	P2	disponibilité
Équipement	3	9	81
Main d'œuvre	4	5	55
Matière première	2	1	20

P1 et P2 rapportent à la vente 6 euros et 4 euros par unité

Quelles quantités (non entières) de produits P1 et P2 doit produire l'usine pour maximiser le bénéfice total venant de la vente des deux produits ?

Introduction à la programmation linéaire

Le problème de production se modélise sous la forme d'un *programme linéaire* :

$$\max_{x1,x2} \left[F(x1,x2) = 6x1 + 4x2 \right]$$

sous les contraintes

$$\begin{cases} 3x1 + 9\,x2 \le 81 \\ 4x1 + 5x2 \le 55 \\ 2x1 + x2 \le 20 \end{cases}$$

$$x1, x2 \ge 0$$

Introduction à la programmation linéaire

Le programme linéaire (1.1)-(1.2) s'écrit sous forme canonique matricielle :

$$\begin{cases} A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq 0 \\ \text{maximiser } Z(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \end{cases}$$

avec

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n_+,\ \mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n,\ \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m \text{ et } A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m\times n}.$$

Où n représente le nombre de variables
m représente le nombre de contraintes

Introduction à la programmation linéaire

$$\begin{cases} A.x \leq b \\ X \geq 0 \\ \text{Maximiser } Z(x) = c^T.\, X \end{cases}$$

Avec

$X = \begin{pmatrix} x1 \\ x2 \end{pmatrix}$ $c = \begin{pmatrix} 6 \\ 4 \end{pmatrix}$ $b = \begin{pmatrix} 81 \\ 55 \\ 20 \end{pmatrix}$ et $A = \begin{pmatrix} 3 & 9 \\ 4 & 5 \\ 2 & 1 \end{pmatrix}$

n = 2 et m =3

Structuration

La modélisation d'un problème d'optimisation se divise en deux parties :

- La section Modèle contient toutes les déclarations, les paramètres calculables et les définitions des contraintes et de l'objectif.
- La section Données contient toutes les données fixes (valeurs des paramètres, du contenu des ensembles).

Structuration

Les deux sections peuvent être déclarées :

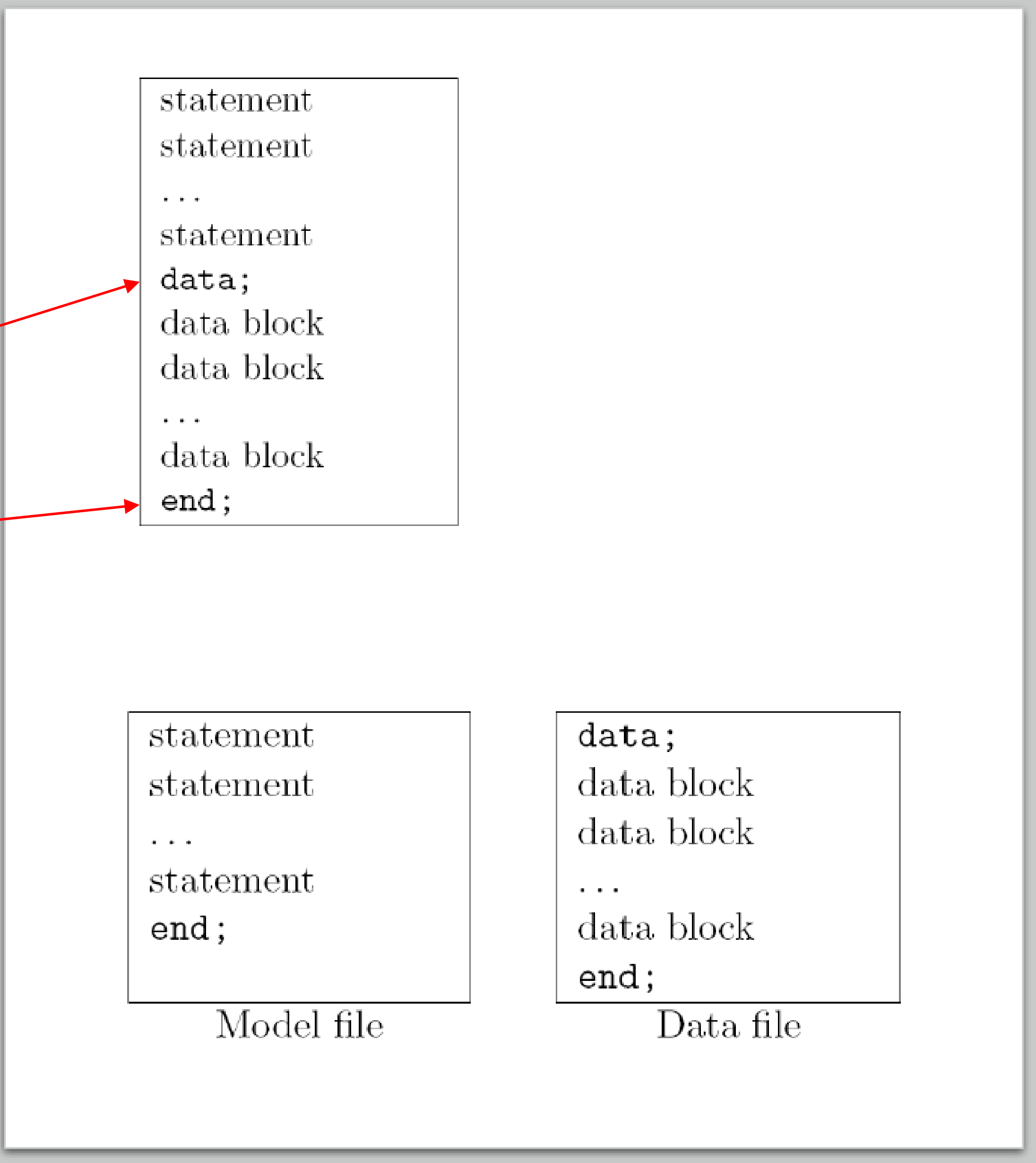
Dans un même fichier composé comme suit :

Il est obligatoire de séparer les deux parties en plaçant la partie de données entre data; et end;.

Le fichier devra être sauvegardé avec l'extension (*.mod).

Dans 2 fichiers séparés :

Dans ce cas, le modèle doit être sauvé avec l'extension (*.mod) et les données avec l'extension (*.dat).



Données

Définition des ensembles

Forme générale	`set` *name* , *record* , . . . , *record* ; `set` [*symbol*, . . . , *symbol*] , *record* , . . . , *record* ;
Avec	*name* est le nom symbolique de l'ensemble *symbol*, . . . , *symbol* sont les indices qui spécifient un élément particulier de l'ensemble *record* , . . . , *record* sont les différentes entrées de données
Records	:= est facultatif mais permet une meilleure lisibilité (*slice*) spécifie un n-uplet *simple-data* définit un ensemble dans un format simple : *matrix data* définit un ensemble dans un format matriciel `(tr)` : *matrix data* définit un ensemble sous la forme de la transposée d'une matrice

- set month := "Jan" "Fev" "Mar" "Avr" "Mai" "Jui";
- set A[3,'Mar'] := (1,2) (2,3) (4,2) (3,1) (2,2) (4,4) (3,4);
- set A[3,'Mar'] : 1 2 3 4 :=
    - 1 - + - -
    - 2 - + + -
    - 3 + - - +
    - 4 - + - + ;
- set B := (1,2,3) (1,3,2) (2,3,1) (2,1,3) (1,2,2) (1,1,1) (2,1,1);

```
set Objets;
var x{i in Objets}>=0, integer;
minimize Obj : sum{i in Objets} x[i];
s.t. rest {i in Objets} : x[i]>=7;

data;
set Objets:= Animals Plantes Personnes;
end;
```

Données

Définition des paramètres

Forme générale	`param` *name* , *record* , . . . , *record* ; `param` *name* `default` *value* , *record* , . . . , *record* ; `param` : *tabbing-data* ; `param default` *value* : *tabbing-data* ;
Avec	*name* est le nom symbolique du paramètre *value* est une valeur par défaut du paramètre *record* , . . . , *record* sont les différentes entrées de données *tabbing-data* représente l'entrée des données sous la forme d'un tableau
Records	`:=` est facultatif mais permet une meilleure lisibilité `[` *slice* `]` spécifie un n-uplet *plain-data* définit un ensemble dans un format plain $(indice_1, \ldots, indice_n, valeur)$ *: tabular data* définit un ensemble dans un tableau `(tr)` *: tabular data* définit un paramètre sous la forme de la transposée d'une matrice

Données

Définition des paramètres

```
param j := 3
param A := 1 1 2 3 3 8 4 6 ;
# A[3]*3 existe et vaut 24
param B : 1 2 3 4 :=
          1 2 3 4 5
          2 5 4 9 3
          3 4 5 9 2;
# B[2,4]*B[1,3]*2 existe et vaut 3*4*2=24
param C : PAR TOU MAR :=
          PAR 0 700 850
          TOU 700 0 500
          MAR 850 500 0;
# C['PAR','TOU']+C['MAR','TOU'] existe et vaut 700+500=1 200
```

```
data;
param T:=2;
param b := 1 81 2 55 3 20;
param A : 1 2 :=
          1 3 9
          2 4 5
          3 2 1;
param c := 1 6 2 4;
end;
```

Règles de codage du modèle

- **Les chaînes de caractères**

La chaîne de caractères est en fait une séquence de caractères enfermés par ‘ ou " (les 2 formes sont équivalentes). Pour introduire ces mêmes caractères dans une chaîne de caractères, il faut les doubler.

- **Les commentaires**

Les commentaires peuvent être sur une ligne et dans ce cas commencent après le caractère spécial # et finissent à la fin de la ligne. Ils peuvent être sur plusieurs lignes et sont inscrits entre /* et */.

```
var a := 4; #Definition de la variable a
var b := 5; /* Definition de
la variable b */
```

Règles de codage du modèle

Définition des ensembles

Forme générale	`set` *name* {*domain* } , *attrib* , . . . , *attrib* ;
Avec	*name* est le nom symbolique de l'ensemble *domain* est optionnel et définit les dimensions et/ou l'ensemble de définition de l'ensemble *attrib*, . . . , *attrib* est une série d'attributs optionnels
Attributs	`dimen` *n* spécifie la dimension des n-uplets de l'ensemble `within` *expression* qui contraint tous les éléments de l'ensemble à être dans un ensemble plus grand `:=` *expression* qui assigne une valeur fixée ou calculée à l'ensemble `default` *expression* qui spécifie une valeur par défaut à l'ensemble ou à un de ses éléments quand aucune information n'est disponible

- set noeuds;
- set arcs within (noeuds cross noeuds);

```
set Objets;
var x{i in Objets}>=0, integer;
minimize Obj : sum{i in Objets} x[i];
s.t. rest {i in Objets} : x[i]>=7;

data;
set Objets:= Animals Plantes Personnes;
end;
```

Règles de codage du modèle

Déclaration des paramètres

Forme générale	`param` *name* {*domain* } , *attrib* , . . . , *attrib* ;
Avec	*name* est le nom symbolique du paramètre *domain* est optionnel et définit les dimensions et/ou l'ensemble de définition du paramètre *attrib*, . . . , *attrib* est une série d'attributs optionnels
Attributs optionnels	`integer` pour spécifier que le paramètre est entier `binary` pour spécifier que le paramètre est binaire (0 ou 1) `symbolic` pour spécifier que le paramètre est symbolique *relation avec* `< <= = == >= > <> !=` pour obliger le paramètre à vérifier des conditions formulées via ces opérateurs (sinon erreur !) `in` *expression* pour obliger le paramètre à être dans un certain ensemble `:=` *expression* qui assigne une valeur fixée ou calculée au paramètre `default` *expression* qui spécifie une valeur par défaut au paramètre quand aucune information n'est disponible

- param I := 2;
- param units{I};
- param N := 20, integer, >=0, <=100;
- param A {n in 0..N, k in 0..n} := if k=0 or k=n then 1 else A[n-1,k-1]+A[n-1,k];

```
param T; # T représente le nombre d'articles
param b{i in 1..3};
param A{i in 1..3, j in 1..T};
param c{j in 1..T};
```

Règles de codage du modèle

Définition des variables

Forme générale	var *name* {*domain* } , *attrib* , . . . , *attrib* ;
Avec	*name* est le nom symbolique de la variable *domain* est optionnel et définit les dimensions et/ou l'ensemble de définition de la variable *attrib*, . . . , *attrib* est une série d'attributs optionnels
Attributs optionnels	integer pour contraindre la variable à être entière binary pour contraindre la variable à être binaire (0 ou 1) >= *expression* spécifie une borne inférieure à la variable <= *expression* spécifie une borne supérieure à la variable = (ou ==) *expression* spécifie une valeur fixée à la variable

```
var x = 0;

var y{I,J};

var A{n in I}, integer, >= b[n], <= c[n];

var z{i in I, j in J} >= i+j;
```

```
var x{i in 1..T}, integer;
```

Règles de codage du modèle

Définition des contraintes

Forme générale	`subject to` *name* {*domain* } : *expression* , = *expression* ; `subject to` *name* {*domain* } : *expression* , <= *expression* ; `subject to` *name* {*domain* } : *expression* , >= *expression* ; `subject to` *name* {*domain* } : *expression* , <= *expression* , <= *expression* ; `subject to` *name* {*domain* } : *expression* , >= *expression* , >= *expression* ;
Avec	*name* est le nom symbolique de la contrainte *domain* est optionnel et définit le nombre de contraintes de ce type *expressions* sont des expressions linéaires pour calculer les composants de la contrainte (la virgule est facultative)
Remarque	Le mot clé `subject to` peut être réduit à `subj to` ou `s.t.` ou même être supprimé.

subject to C1 {i in 1..3} : sum{j in 1..T} A[i,j]*x[j] <= b[i];

- s.t. C1 : x + y + z >= 0 ;
- subject to C2 {t in 1..T, i in 1..I} : x[t] + y[t] <= sqrt[2]*i ;
- subj to C3 {t in Ens1, r in Ens2} : sum{k in 1..t} x[k] + y[r] <= 2;

Règles de codage du modèle

Définition de la fonction objectif

Forme générale	`minimize` *name* {*domain* } : *expression* ; `maximize` *name* {*domain* } : *expression* ;
Avec	*name* est le nom symbolique de l'objectif *domain* est optionnel et définit l'ensemble de définition de l'objectif *expression* est une expression linéaire qui définit l'objectif

- minimize obj : x + 1.5*(y+z) ;
- maximize profit_total : sum{p in 1..P} profit[p] * produits[p] ;

maximize obj : sum{i in 1..T} c[i]*x[i];

Règles de codage du modèle

Résolution et affichage

Solve lance la résolution du problème d'optimisation

Forme générale	solve ;
Remarque	`solve` est facultatif et ne peut être utilisé qu'une seule fois. S'il n'est pas mentionné, GLPK résoud tout de même le modèle en considérant qu'il est placé en fin du modèle.

Display permet d'évaluer des expressions et d'écrire leurs valeurs sur l'output standard.

Forme générale	`display` {*domain* } : *item*, ..., *item* ;
Avec	*domain* est optionnel et définit l'ensemble d'affichage *item*, ..., *item* sont les éléments à afficher

Printf permet d'évaluer des expressions et d'écrire leurs valeurs sur l'output standard selon un formalisme choisi.

Forme générale	`printf` {*domain* } : *format*, *expression*, ..., *expression* ;
Avec	*domain* est optionnel et définit l'ensemble d'affichage *format* spécifie le format d'affichage *expression*, ..., *expression* est la liste des éléments à formater et à afficher

%d : entier – %f : réel (float) – %e : réel sous format xxx.yyy e+X
%E : réelsous format xxx.yyy E+X
%g : format le plus court entre %f et %e
%G : format le plus court entre %f et %E
%s : chaîne de caractères
De même, on rappelle que \n permet de passer à la ligne, \t d'introduire une tabulation, \b un backspace et \v un Tab vertical.

Compilation

Commandes de base

Supposons dans un premier temps que tout est incorpore dans un fichier modele.mod. Dans une fenêtre de commande, on se place dans le répertoire où est situé le fichier. L'appel au solveur est effectué à l'aide de la commande :

glpsol --model modele.mod

Dans le cas général (et conseillé !) où les données et le modèle sont séparés en 2 fichiers modele.mod et donnees.dat, l’exécution s’effectue avec :

glpsol --model modele.mod --data donnees.dat