M2ISN, RO: Introduction à AMPL

Bernhard Beckermann,
Labo Painlevé (AN-EDP), ULille
Bernhard.Beckermann@univ-lille.fr

Villeneuve d'Ascq, 12 septembre 2025

Introduction à AMPL

Le document ci–dessous constitue une courte et incomplète introduction au logiciel AMPL. Des versions ”étudiants” gratuites de ce logiciel sont disponibles à l'URL `http://www.ampl.com`, ainsi que des multiples manuels. La lecture du livre Robert Fourer, David M. Gay, and Brian W. Kernighan, AMPL : A Modeling Language For Mathematical Programming. est fortement conseillée.
AMPL est

- un langage de modélisation pour les problèmes (non) linéaires et discrets ;
- une interface (*DOS, Windows, Linux, kestrel, web, chez nous* monampl *interface entre TclTk et AMPL*) faisant appel à des nombreux solvers sous forme de boîte noire (*minos,cplex,lpsolve,bmftp,...*)

Philosophie de AMPL :

- créer des modèles génériques
- séparer le problème mathématique des données.
- choisir les noms parlant pour les variables et paramètres

Pour un problème mathématique nommé 'toto' on construit

- un fichier `toto.mod` comportant
  - Déclaration d'ensembles d'indices : `set nom_d_ensemble`
  - Déclaration des données numériques : `param nom_de_donnee`
  - Déclaration des inconnues : `var nom_de_variable`
  - L'objectif : `minimize, maximize`
  - Les contraintes : `subject to nom_de_contrainte`
- un fichier `toto.dat` comportant
  - Le contenu des ensembles
  - La valeur actuelle des paramètres
- éventuellement un fichier `toto.run` comportant
  - chargement de modèle/données : `model toto.mod; data toto.dat;`
  - Choix de la boîte noire : `option solver cplex;`
  - La résolution : `solve;`
  - L'affichage des variables/paramètres (les bornes, variables d'écart) : `display nom_de_variable; display quoi, quoi.lb, quoi.ub; display nom_de_contrainte.slack;`
  - avant de reprendre un deuxième exemple : `reset;`

L'exemple du boulanger

Un boulanger veut produire des rouleaux de pâte brisée et de pâte feuilletée. La production d'une unité de rouleaux (en milliers) nécessite une certaine quantité de beurre, sel et farine (en tonnes, cf. tableau ci–dessous). On cherche à maximiser le revenu qu'il est possible de tirer des stocks.

	beurre	sel	farine	prix de vente
pâte brisée	1	0	2	4
pâte feuilletée	2	1	1	5
quantité en stock	7	3	8	

On introduira

- 2 ensembles : `PATES` et matières premières `MATP`.
- 3 paramètres : le prix de vente (`prix` indicé par `PATES`), la quantité nécessaire pour produire une unité de rouleaux (`qte`, indicée par `PATES` et `MATP`) et le stock (`stock` indicé par `MATP`).
- 1 variable : le nombre d'unités de rouleaux à produire (`a_prod`, indicée par `PATES`).

Problème mathématique :

$$\begin{cases} \max \sum_{p \in PATES} prix[p] * a\_prod[p], \ \forall p \in PATES : a\_prod[p] \geqslant 0, \\ \forall m \in MATP : \sum_{k \in PATES} qte[p, m] * a\_prod[p] \leqslant stock[m] \end{cases}$$

```
# Fichier mod\‘ele (boulanger.mod)
set PATES; # les differentes pates a produire
set MATP;  # les differentes matieres premieres a utiliser
param prix{PATES} >= 0;       # benefices unitaires
param qte{PATES, MATP} >= 0; # matieres premieres utilisees par roule
param stock{MATP} >= 0;       # matieres premieres en stock
var a_prod{PATES} >= 0;       # la variable a ajuster
maximize revenu :
   sum{p in PATES} prix[p] * a_prod[p];
subject to stocks_limites {m in MATP} :
   sum{p in PATES} qte[p,m] * a_prod[p] <= stock[m];

# Fichier de donn\’ees (boulanger.dat)
set PATES := brisee feuilletee;
set MATP := beurre sel farine;
param stock := beurre 7000   sel 3000   farine 8000;
param qte : beurre   sel   farine :=
   brisee     1       0       2
   feuilletee 2       1       1;
param prix := brisee 4   feuilletee 5;

# Fichier executable (boulanger.run)
model boulanger.mod;
data boulanger.dat;
option solver lpsolve;
solve;
display a_prod; display revenu;
```

L'interface MONAMPL

Interface écrit en *TclTk* pour accéder à AMPL.

Objectif :

- Editer/afficher facilement des fichiers **.mod* et **.dat*
- Création automatique des fichiers **.run* (batchfile pour AMPL)
- Création de tableaux/graphes pour valoriser les résultats.

Comment résoudre un problème ?

- Ouvrir/editer/sauver les fichiers **.mod et *.dat* ;
- Choisir un solveur : (CPLEX=entiers, minos=non linéaire) ;
- *Make :* créer un fichier exécutable AMPL **.run*, variables à afficher ;
- *Run :* lancer AMPL, afficher les résultats ;
- Pour afficher les résultats à l'aide d'un graphe/tableau, voir plus loin.

Exemples :

- Le boulanger ou steel (optimisation linéaire simple)
- hamilton2 (voyageur de commerce avec un circuit)
- netmax3 (transporter une quantité maximum d'un bien du sommet a vers g)

Le menu de `monampl`

- *File/Open [buttons]* ouvre des fichiers *.mod,*.dat,*.log,*.run déjà existants
- *File/Save [button]* sauvegarde les fichiers *.mod,*.dat,*.log,*.run à l'endroit actuel
- *File/Save as* sauvegarde les fichiers *.mod,*.dat,*.log,*.run à un nouvel endroit
- *File/Project* permet de mémoriser 4 fichiers (pas forcement du même nom) et l'espace de travail.
- *File/Quit [button]* sauvegarde et sort du programme
- *Edit* les commandes classiques d'édition
- *View* des commandes classiques d'affichage, coloriage des mots clefs (on colorie les ensembles/variables/contraintes si le modèle est correct), taille des caractères.
- *AMPL/Debug [button]* : vérification du modèle, messages d'erreur plus explicites.
- *AMPL/Make [button]* crée fichier executable *.run.
- *AMPL/Make .. web* rend le fichier *.run adapté pour une résolution par une interface web (sans limitations), lance internet explorer (voir le fichier *.run pour plus de détails).
- *AMPL/Run [button]* exécute le fichier *.run, affiche les résultats
- *AMPL/Graph* et *AMPL/Table* et *AMPL/Make Post...* : voir les pages suivantes
- *AMPL/Solver* choisit la boîte noire AMPL (*cplex* : entiers, *minos* : nonlineaire, ...)
- *Help* quelques pages d'aide sur AMPL et sur l'interface.

Dans les fenêtres on peut utiliser les commandes classiques de Windows pour éditer : *ctrl x = shift suppr* pour couper, *ctrl v = shift inser* pour coller, essayez aussi *ctrl* $\rightarrow$, *shift* $\rightarrow$ *ctrl shift* $\rightarrow$... La plupart des commandes possède une touche raccourcie $<$ *ctrl* $>$, voir menu

Comment créer un graphe ? Si le modèle correspond à un problème d'optimisation sur un graphe et si on a coché la case "AMPL/show graph", il est possible d'afficher le graphe. Pour cela on doit fixer

- les ensembles correspondant aux sommets (ici *set INTER*) et aux arcs (ici *set ROADS*) ;
- *optionnel :* les valeurs/couleurs des sommets (variables/paramètres indicés par *INTER*) :
- *optionnel :* les valeurs/couleurs des arcs (variables/paramètres indicés par *ROADS*) ;
- *optionnel :* les coordonnées des sommets (sinon positions aléatoires).

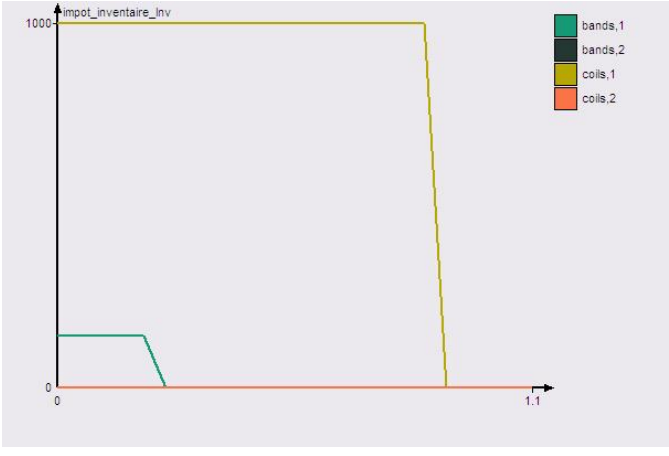
Ces données sont déclarées et sauvegardées dans le fichier **.mod* correspondant, ou toute ligne doit commencer par une chaîne particuliere de caractères et un espace. Voici l'exemple netmax3 :

```
###graph: vertexset  INTER  # obl: set
###graph: vertexxpos  xpos # obl: unknown index vertexset
###graph: vertexypos  ypos # obl: unknown index vertexset
###graph: vertexcolor  # opt: unknown, index vertexset
###graph: vertexvalue  # opt: unknown, index vertexset
###graph: edgeset   ROADS   # obl: set subset vertexset x vertexset
###graph: edgecolor  cap  # opt: unknown, index edgeset
###graph: edgevalue  traffic  # opt: unknown, index edgeset
```

On peut saisir ces variables, déplacer les sommets, sauvegarder **.ps/jpg*

Comment créer un tableau ? Si le modèle a été correctement écrit (sans messages d'erreur) et lancé, les noms des ensembles/variables/contraintes apparaissent en couleur rouge. Cliquer avec le bouton droite de la souris sur une telle variable. A part de pouvoir afficher leur contenu (display) ou l'ajouter dans le fichier *.run, on peut aussi lire une quantité indexée dans un tableau (également pour les paramètres).

impot_inventaire_Inv	bands,1	bands,2	coils,1	coils,2
0	142.857	0	1000	0
0.05	142.857	0	1000	0
0.1	142.857	0	1000	0
0.15000000000000002	142.857	0	1000	0
0.2	142.857	0	1000	0
0.25	0	0	1000	0
0.30000000000000004	0	0	1000	0
0.35000000000000003	0	0	1000	0
0.4	0	0	1000	0
0.45	0	0	1000	0
0.5	0	0	1000	0
0.55	0	0	1000	0
0.6000000000000001	0	0	1000	0
0.65	0	0	1000	0
0.7000000000000001	0	0	1000	0
0.75	0	0	1000	0
0.8	0	0	1000	0
0.8500000000000001	0	0	1000	0
0.9	0	0	0	0
0.9500000000000001	0	0	0	0
1	0	0	0	0
1.05	0	0	0	0
1.1	0	0	0	0



Ensuite on peut editer et sauvegarder ce tableau (dans *.dat, dans le "clipboard" pour l'exporter sous excel, ou sous format *.jpg), ou le transformer en graphique.

L'exemple de l'usine sidérurgique

On traitera un problème d'une usine sidérurgique : son laminoir prend en entrée des brames d'acier et produit deux sortes de produits semi-finis (*PROD*), sous forme de ruban (*bands*) ou de fil (*coils*). Si le laminoir produit du ruban, sa production est de 200t/h, s'il produit du fil il en produit 140t/h (*rate*). L'expérience dit que l'usine ne peut pas vendre plus que 6000t par semaine de ruban et de 4000t/semaine du fil (*market*). Le bénéfice par tonne de ruban est de 25F, et de 30F/t pour le fil (*profit*). Cherchons le plan de production pour 40 heures disponibles (*avail*) maximisant les bénéfices.

- 1 ensemble : `PROD`
- 4 paramètres : `rate, market, profit` indicées par `PROD` et `avail`.
- 1 variable : `Make` indicée par `PROD`, le nombre de produits semi-finis à produire (en tonnes)

Problème mathématique :

$$\begin{cases} \max \sum_{p \in PROD} profit[p] * Make[p] \\ \forall p \in PROD : 0 \leqslant Make[p] \leqslant market[p] \\ \sum_{k \in PROD} Make[p]/rate[p] \leqslant avail \end{cases}$$

Fichiers de l'exemple steel.*

```
# fichier steel.mod
set PROD;                       # products
param rate {PROD} > 0;          # tons produced per hour
param avail >=0;                # hours available in week
param profit {PROD};            # profit per ton
param market {PROD} >= 0;       # limit on tons sold in week
var Make {p in PROD} >= 0, <= market[p]; # tons produced

maximize total_profit: sum {p in PROD} profit[p] * Make[p];
    # Objective: total profits from all products
subject to Time: sum {p in PROD} (1/rate[p]) * Make[p] <= avail;
    # Constraint: total of hours used by all products may not
    # exceed hours available

# fichier steel.dat
set PROD := bands coils;
param:    rate  profit  market :=
 bands    200    25     6000
 coils    140    30     4000 ;
param avail := 40;
```

(steel.mod)

Variation : ajouter un produit

Quel modification faut-il adapter si on peut produire aussi de plaques (*plates*) sachant que le le laminoir peut produire 160t/h de plaques, que les bénéfices sont de 29F/t et qu'on ne peut pas vendre plus que 3500 tonnes par semaine de plaques.

Voir steel2.mod

Variation : ajouter une autre contrainte

On suppose que l'usine a pris un engagement (*commitment*) de produire au moins 1000 t/semaine de fil, 500 t/semaine de rubans et 750 t/semaine de plaques. Écrire les modifications AMPL.

Voir steel3.mod

Variation : ajouter une autre phase de production

Supposons qu'il y a une phase de production supplémentaire, plus précisément une première phase de recuit (*reheat*) de l'acier (avant la phase déjà connue de laminage). Dans cette phase de recuit on peut traiter 200t/h de chaque produit, mais la machine est seulement disponible pendant une période de 35h. Écrire les modifications AMPL.

Voir steel4.mod

Variation : épuiser le temps disponible

On souhaite modifier le modèle de telle sorte que le temps disponible soit égal au temps utilisé. Obtient–on une autre solution ?

Voir steel4.mod

Variation : restreindre/maximiser le poids total

Que faut-il ajouter au modèle pour que le poids total des produits soit inférieur à un nouveau paramètre `poids_max` ? Résoudre le nouveau programme pour une limite de 6500 tonnes et justifier l'impact sur la solution.

Voir steel5.mod

Comment changer le modèle pour maximiser le nombre total de tonnes produits ? Impact pour notre exemple ?

Variation : imposer des quotas

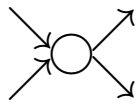
Partant de steel.mod, supposons qu'à la place des bornes inférieures sur les chiffres de production x_k on désire imposer que chaque produit représente au moins une certaine fraction s_k du total des tonnes produites, i.e., $x_k / \sum_k x_k$ est borné inférieurement (attention à garder un programme linéaire). Si les quota sont $0,4$ pour le ruban et $0,1$ pour le fil, quelle est la nouvelle solution ? Et pour les quota $0,5$ pour le ruban et $0,1$ pour le fil ? Justifier.

Voir steel6.mod

Multi-période : L'usine sur plusieurs semaines

unités produites au cours de la semaine t

unités en stock à la fin de la semaine $t-1$



unités vendues au cours de la semaine

unités en stock à la fin de la semaine

Dans notre usine sidérurgique, on se permet de produire sur plusieurs semaines (entre semaine 1 et semaine T), mais au lieu de vendre la production immédiatement, on dispose de la possibilité de mettre le produit dans un entrepôt. Considérons les variables

- *Make[p,t]* la production en tonnes du produit p au cours de la semaine t
- *Sell[p,t]* la vente en tonnes du produit p au cours de la semaine t
- *Inv[p,t]* la quantité en tonnes du produit p à la fin de la semaine t dans l'entrepôt.
- *Inv[p,0]* est donné (par paramètre *inv0[p]*).

On obtient alors le bilan

$$\forall p \in PROD\ \forall t \in \{1...T\} : \quad Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t]$$

Trois groupes de paramètres

- *rate, inv0, prodcost, invcost* dépendant seulement du produit : la production par heure (en t/h), le stock au début (en t), les frais unitaires de production (en F/t) et les frais unitaires de stockage (en F/t)
- *market, revenue* dépendant du produit et de la semaine : les bornes supérieures pour la production (en t), les bénéfices unitaires du produit vendu (en F/t)
- *avail* dépendant de la semaine : le temps de disponibilité du laminoir (en h).

Ceci donne lieu au modèle suivant (voir steelT.mod)

```
set PROD;                              # products
param T > 0;                           # number of weeks
param rate {PROD} > 0;                 # tons per hour produced
param inv0 {PROD} >= 0;                # initial inventory
param avail {1..T} >= 0;               # hours available in week
param market {PROD,1..T} >=0;          # limit on tons sold in week
param prodcost {PROD} >= 0;            # cost per ton produced
param invcost {PROD} >= 0;             # carrying cost/ton of inventory
param revenue {PROD,1..T} >= 0;        # revenue per ton sold
```

```
 var Make {PROD,1..T} >= 0;       # tons produced
 var Inv {PROD,0..T} >= 0;        # tons inventoried
 var Sell {p in PROD, t in 1..T} >= 0, <= market[p,t]; # tons sold

maximize total_profit:
   sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] -
      prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t]);
                # Total revenue less costs in all weeks

subject to time {t in 1..T}:
   sum {p in PROD} (1/rate[p]) * Make[p,t] <= avail[t];
                # Total of hours used by all products
                # may not exceed hours available, in each week
subject to initial {p in PROD}:  Inv[p,0] = inv0[p];
                # Initial inventory must equal given value
subject to balance {p in PROD, t in 1..T}:
   Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];
                # Tons produced and taken from inventory
                # must equal tons sold and put into inventory
```

PS : il vaut mieux définir des ensembles ordonnés au lieu des intervalles...(voir steelTbis)

Astuces sous AMPL

- **Nommer des quantités au moyen de variables supplémentaires :** Le soucis premier dans l'écriture d'un modèle AMPL doit être la lisibilité (les boîtes noires prévoient une optimisation du code). Par conséquent, une quantité souvent utilisée comme la production totale mérite parfois un propre nom (aussi variable).
- **Paramètres calculés :** Parfois certaines quantités sont calculés en fonction des données, à titre d'exemple les distances (vol d'oiseau) entre villes sachant leur coordonnées sur un plan

```
set VILLES;
param xpos{VILLES} >=0, <= 10;  param ypos{VILLES} >=0, <= 10;
param distance{(i,j) in VILLES cross VILLES} :=
   sqrt((xpos[i]-xpos[j])**2 + (ypos[i]-ypos[j])**2);
```

  Ici paramètres complètement déterminés, pas de variables.
- **Nombres entiers/binaires et non linéarité :** Par défaut, les variables AMPL sont de type *réel*. On peut imposer le type *entier*

```
var jours >=0, integer;
```

  (ou *binaire* par le qualificatif `binary`). Pour traiter des tels variables

```
option solver cplex;
```

  Le solveur par défaut, *minos*, ne résout pas les programmes linéaires en nombres entiers (mais il sait traiter problèmes non linéaires).

- **Nombre d'éléments :**

```
set PROD; param nb_prod := card(PROD);
```

- **Les intervalles :** ensembles de nombres ordonnés (on peut spécifier *by "pas"* si on le souhaite)

```
param N >= 1; set ANNEES := 1 .. N;
```

- **Opérations sur les ensembles :**

```
set A; set B;
set C:= A union B; set D := A inter B;  # aussi diff, symdiff
```

- **Désigner un élément ou une partie d'un ensemble :**

```
set NOEUDS;
param racine symbolic in NOEUDS;   # racine n'aura pas de valeu
set FEUILLES within NOEUDS := NOEUDS diff { racine };
```

- **L'opérateur " : " :** appartenance conditionnelle, p.e., matrice triangulaire supérieure

```
param N integer >= 0;
param matrice {j in 1 .. N, k in 1 .. N : j <= k};
```

- **Fixer un ordre dans un ensemble :**

```
set GRADES ordered;
first(GRADES); last(GRADES); prev(g,GRADES);
```

Compléments sur les fichiers de données

- **Paramètres indicés par un même ensemble :** Plusieurs tableaux avec même indiçage peuvent être définis simultanément (voir gauche), même avec leur ensemble d'indices (voir droite).

```
set PROD := A B F;
param : achat vente :=
   A     10    12
   B     37    29
   F     17    20;
```

```
param : PROD : achat :=
          A       10
          B       37
          F       20;
```

- **Laisser des valeurs indéfinies :** Il suffit de mettre un " . " à la place de la valeur indéfinie.
- **Paramètres à trois dimensions :**

```
# dans fichier.mod: param cube {INDICES,INDICES, INDICES};
set INDICES := 1 .. 2;
param cube :=
  [*, *, 1] : 1    2 :=
          1   3    5
          2   4    6
```

```
[*, *, 2] : 1    2 :=
          1   2   11
          2   0    7;
```

Comment créer des fichiers excel/openoffice

On suppose que le modèle comporte un paramètre `lambda`, et on souhaite faire des simulations pour plusieurs valeurs de λ, et créer un tableau comportant par ligne la valeur de `lambda`, et la valeur optimale `profit` associée. **Solution 1 :** Créer le fichier `exo2c.run` suivant

```
reset;
model ../sci2ampl/exercices/exo1c.mod;
data ../sci2ampl/exercices/exo1c.dat;
option solver cplex;
set valeurs := -30 .. 30 by 1;
param objectif{valeurs};
for {i in valeurs} {
   let lambda:=i; solve;
   let objectif[i]:=profit;
}
table ECRIRE OUT "../sci2ampl/exercices/exo1c.tab" :
   [valeurs], objectif;
write table ECRIRE;
```

Il reste à ouvrir le fichier `exo1c.tab`, remplacer les points par des virgules, et copier le tableau dans une feuille excel. Ensuite, sous openoffice, il faut marquer la partie que l'on veut transformer en graphique, et choisir [Insertion/Diagramme]. **Solution 2 :** voir la post optimisation...