Chapitre 5

Compléments sur le langage AMPL

5.1 Les commentaires

Voici deux façons d'inclure des commentaires dans un modèle AMPL.

```
/*
   Ceci est un commentaire rédigé sur
   plusieurs lignes.
*/

var x >= 0; # en euros/tonne
```

5.2 Les paramètres

5.2.1 Paramètres indicés par un même ensemble

Considérons l'exemple suivant.

```
set PROD;
param prix_achat {PROD};
param prix_vente {PROD};

data;
set PROD := A B F;
param prix_achat :=
    A  10
    B  37
    F  17;
param prix_vente :=
    A  12
    B  29
    F  20;
```

Comme *prix_achat* et *prix_vente* sont indicés par un même ensemble, il est possible de déclarer les deux d'un coup.


```
data;
set PROD := A B F;
param : prix_achat prix_vente :=
    A        10        12
    B        37        29
    F        17        20;
```

Il est même possible de déclarer en une fois non seulement les deux paramètres mais aussi l'ensemble *PROD*. Il suffit de procéder comme suit :

```
data;
param : PROD : prix_achat prix_vente :=
        A        10        12
        B        37        29
        F        17        20;
```

5.2.2 Paramètres indicés par deux ensembles

Le premier indice est l'indice de ligne. Le second est l'indice de colonne. Si on s'est trompé, on peut insérer « (tr) » entre l'identificateur du paramètre et le « : » dans la partie « data », indiquant ainsi qu'on donne la transposée de la matrice.

```
set PLATS;
set VITAMINES;
param apport {PLATS, VITAMINES};

data;

set PLATS := potjevleesch carbonnade waterzoi;
set VITAMINES := A C D;

param apport :
               A      C       D :=
potjevleesch   18     20      44
carbonnade      5     30      69
waterzoi       40      5      30;
```

5.2.3 Laisser des valeurs indéfinies

Il suffit de mettre un « . » à la place de la valeur indéfinie. En voici une utilisation lors de la définition d'une matrice symétrique.

```
param N integer >= 1;
param matsym {lig in 1 .. N, lig .. N} >= 0;

data;
param N := 3;
param matsym :
        1   2   3 :=
```

```
   1   4   5   8
   2   .   7  11
   3   .   .   2;
```

5.2.4 Paramètres indicés par trois ensembles

Voici un exemple de déclaration d'un paramètre *cube* à trois dimensions.

```
# fichier cube
set INDICES := 1 .. 2;
param cube {INDICES, INDICES, INDICES};

data;
param cube :=
   [*, *, 1] : 1    2 :=
             1   3    5
             2   4    6
   [*, *, 2] : 1    2 :=
             1   2   11
             2   0    7;
```

5.3 L'affichage

Le format d'affichage par défaut du paramètre *cube* (le format « liste ») n'est pas très agréable.

```
ampl: model cube;
display cube;
cube :=
1 1 1    3
1 1 2    2
1 2 1    5
1 2 2   11
2 1 1    4
2 1 2    0
2 2 1    6
2 2 2    7
;
```

On peut obtenir un affichage sous forme plus compacte en modifiant la valeur de la variable prédéfinie *display_1col*, qui contient la taille en dessous de laquelle les données sont affichées au format « liste ». Par défaut sa valeur est 20. Pour forcer l'affichage du cube sous forme plus compacte, il suffit donc d'affecter à *display_1col* une valeur inférieure au nombre d'éléments de *cube*. Par exemple zéro.

```
ampl: option display_1col 0;
ampl: display cube;
cube [1,*,*]
:   1   2     :=
```



```
1   3    2
2   5   11

 [2,*,*]
:   1   2    :=
1   4   0
2   6   7
;
```

5.4 Les ensembles

5.4.1 Les ensembles non ordonnés

On signale au passage l'existence de la fonction *card*.

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

5.4.2 Les intervalles

Les intervalles sont des ensembles de nombres ordonnés (on peut spécifier *by* « *pas* » si on le souhaite).

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

5.4.3 Les opérations ensemblistes

Les opérations ensemblistes permettent de définir des ensembles à partir d'ensembles A et B existants.

A union B,	
A inter B,	
A diff B	dans A mais pas dans B,
A symdiff B	dans A ou B mais pas les deux.

```
set HUILES_VEGETALES;
set HUILES_ANIMALES;
set HUILES := HUILES_VEGETALES union HUILES_ANIMALES;
```

5.4.4 Désigner un élément ou une partie d'un ensemble

On dispose des opérateurs suivants

in	pour l'appartenance
within	pour l'inclusion.

```
# Fichier exemple
set NOEUDS;
param racine in NOEUDS;
set FEUILLES within NOEUDS := NOEUDS diff { racine };

data;
set NOEUDS := 3 17 24 45;
param racine := 17;
```

Exemple d'exécution

```
$ ampl
ampl: model exemple;
ampl: display NOEUDS;
set NOEUDS := 3 17 24 45;
ampl: display racine;
racine = 17
ampl: display FEUILLES;
set FEUILLES := 3 24 45;
```

5.4.5 Désigner un élément d'un ensemble de symboles

Implicitement

On peut vouloir manipuler des noeuds désignés par des symboles plutôt que des nombres. Il faut alors ajouter le qualificatif *symbolic* au paramètre *racine* : les paramètres en effet sont censés désigner des quantités numériques.

```
# Fichier exemple
set NOEUDS;
param racine symbolic in NOEUDS;
set FEUILLES within NOEUDS := NOEUDS diff { racine };

data;
set NOEUDS := A B C D
param racine := B;
```

On continue l'exemple précédent pour montrer un bel exemple de paramètre calculé. On suppose que les noeuds appartiennent à un arbre qu'on décrit grâce la fonction *prédecesseur*. Cette fonction est codée par un paramètre indicé par l'ensemble des noeuds moins la racine. On signale qu'un noeud ne peut pas être son propre prédecesseur. Enfin on affecte au paramètre *prof* la *profondeur* de chaque noeud.

```
param pred {n in NOEUDS diff {racine}} in NOEUDS diff {n};
param prof {n in NOEUDS} :=
    if n = racine then 0 else 1 + prof [pred [n]];
```


Explicitement

Il est parfois utile dans un modèle de manipuler explicitement un élément d'un ensemble. Cette opération est à éviter autant que faire se peut puisque les éléments des ensembles ne sont pas censés être connus dans le modèle. Il vaut mieux utiliser la méthode de la section précédente. Quand c'est nécessaire, il suffit de mettre des guillemets autour du symbole.

```
set PROD;
param prix {PROD} >= 0;
subject to contrainte_speciale :
    prix ["machin"] <= prix ["truc"];
```

Il est parfois utile aussi de donner la valeur d'un ensemble dans le modèle et non dans les données. Cette opération aussi est à éviter autant que faire se peut. Il suffit de mettre des guillemets autour des symboles.

```
set PROD := { "machin", "truc", "chose" };
```

5.4.6 L'opérateur « : »

Il est suivi d'une expression logique. Voici comment sélectionner l'ensemble des éléments positifs d'un ensemble de nombres.

```
set SIGNES;
set POSITIFS within SIGNES := { i in SIGNES : i >= 0 };
```

Voici un exemple de matrice triangulaire supérieure.

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

Voici deux déclarations possibles de l'ensemble des nombres premiers inférieurs ou égaux à N.

```
param N integer >= 0;
set premiers1 := { n in 2 .. N : forall {m in 2 .. n-1} n mod m <> 0};
set premiers2 := { n in 2 .. N : not exists {m in 2 .. n-1} n mod m = 0};
```

5.4.7 Ensembles ordonnés

On a vu les intervalles qui sont des ensembles de nombres ordonnés. On peut manipuler des ensembles ordonnés de symboles également. Voici une version « symbolique » de l'exemple de la section 2.5.2. On considère un ensemble d'employés de différents grades. Les nouveaux employés du grade g sont soit des anciens employés de ce grade soit d'anciens employés promus depuis le grade précédent. Les expressions conditionnelles permettent de gérer les cas particuliers du premier et du dernier grade.

```
set GRADES ordered;
var anciens {GRADES} >= 0;
var nouveaux {GRADES} >= 0;
# promus [g] = ceux qui passent de g à g+1
var promus {g in GRADES : g <> last (GRADES)} >= 0;
subject to calcule_nouveaux {g in GRADES} :
    nouveaux [g] = anciens [g] +
        (if g <> first (GRADES) then promus [prev (g)] else 0) -
        (if g <> last (GRADES) then promus [g] else 0);
```

Remarque : il est parfois nécessaire de préciser l'ensemble auquel appartient l'élément dont on cherche le successeur ou le prédecesseur. Voici un exemple.

```
set Ens ordered;
set Fns ordered within Ens;

data;
set Ens := A B C D E;
set Fns := B D E;
```

Quel est le prédecesseur de D ? La réponse dépend de l'ensemble considéré :

```
ampl: display prev ("D", Ens);
prev('D', Ens) = C

ampl: display prev ("D", Fns);
prev('D', Fns) = B
```

5.4.8 Ensembles circulaires

Il est également possible de définir des ensembles circulaires (ou cycliques). Il suffit de mettre le qualificatif *circular* au lieu de *ordered*. Le prédecesseur du premier élément est le dernier élément. Le successeur du dernier élément est le premier élément.

5.4.9 Sous–ensembles d'un ensemble ordonné

On peut demander qu'un sous–ensemble d'un ensemble ordonné A soit ordonné suivant le même ordre que A ou suivant l'ordre inverse de celui de A.

```
set A ordered;
set B within A ordered by A;
set C within A ordered by reversed A;
```

La même possibilité existe pour les ensembles circulaires. Il suffit d'utiliser les expressions *circular by* et *circular by reversed*.

5.4.10 La fonction ord

Il permet d'obtenir le numéro d'ordre d'un élément dans un ensemble ordonné. L'ensemble des sommets de numéro d'ordre inférieur à la racine.

```
set NOEUDS ordered;
param racine symbolic in NOEUDS;
set NOEUDS_INF_RACINE within NOEUDS :=
    { n in NOEUDS : ord (n) < ord (racine, NOEUDS) };
```

5.4.11 Produits cartésiens

On les a déjà rencontrés dans le cadre de paramètres à deux dimensions. Voici un exemple de modélisation de carte routière. Un couple (x, y) appartient à *ROUTES* s'il existe une route de x vers y. Pour les routes à double sens, il faut stocker à la fois (x, y) et (y, x).

```
set CARREFOURS;
set ROUTES within { CARREFOURS, CARREFOURS };
param longueur {ROUTES} >= 0;

data;
set CARREFOURS := A B C D;
set ROUTES := (B,A) (B,C) (A,D) (A,C) (C,A);
param : longueur :=
    B A   8
    B C   3
    C A   2
    A C   2
    A D   5;
```

On améliore la déclaration comme suit. On profite de la possibilité de définir d'un seul coup l'ensemble *ROUTES* et le paramètre *longueur*. On vérifie aussi que la carte est cohérente : d'une part une route relie deux carrefours différents, d'autre part si (x, y) et (y, x) appartiennent à *ROUTES* alors ces deux routes ont même longueur.

```
set CARREFOURS;
set ROUTES within { x in CARREFOURS, y in CARREFOURS : x <> y };
param longueur {ROUTES} >= 0;

check : forall { (x,y) in ROUTES : (y,x) in ROUTES }
    longueur [x,y] = longueur [y,x];

data;
set CARREFOURS := A B C D;
param : ROUTES : longueur :=
    B A   8
    B C   3
    C A   2
    A C   2
    A D   5;
```

5.4.12 Ensembles d'ensembles

Le langage AMPL permet de définir des ensembles indicés par d'autres ensembles. Il y a un ensemble de clients et un ensemble de fournisseurs pour chaque entreprise. À partir de ces ensembles, on crée l'ensemble des clients de toutes les entreprises ainsi que, pour chaque entreprise, l'ensemble de tous les couples (fournisseur, client).

```
set ENTREPRISES;
set CLIENTS {ENTREPRISES};
set FOURNISSEURS {ENTREPRISES};

set TOUS_LES_CLIENTS := union {e in ENTREPRISES} CLIENTS [e];

set COUPLES {e in ENTREPRISES} := {FOURNISSEURS [e], CLIENTS [e]};

data;
set ENTREPRISES := e1 e2;
set CLIENTS [e1] := c1 c2;
set CLIENTS [e2] := c1 c3;
set FOURNISSEURS [e1] := f1 f2 f3;
set FOURNISSEURS [e2] := f4;
```

Voici, pour fixer les idées, ce qu'on obtient à l'exécution.

```
ampl: display CLIENTS;
set CLIENTS[e1] := c1 c2;
set CLIENTS[e2] := c1 c3;

ampl: display TOUS_LES_CLIENTS;
set TOUS_LES_CLIENTS := c1 c2 c3;

ampl: display COUPLES;
set COUPLES[e1] := (f1,c1) (f1,c2) (f2,c1) (f2,c2) (f3,c1) (f3,c2);
set COUPLES[e2] := (f4,c1) (f4,c3);
```

5.5 Les opérateurs arithmétiques et logiques

Le tableau suivant donne les opérateurs par priorité croissante. Il n'y a aucune difficulté à appliquer ces opérateurs sur des paramètres soit dans le corps des contraintes soit pour définir des paramètres calculés. Lorsqu'on les applique à des variables, on court le risque d'obtenir des expressions non linéaires.

L'opérateur `less` retourne la différence de ses deux opérandes si elle est positive et zéro sinon. L'opérateur `div` retourne le quotient de la division euclidienne de ses deux opérandes.

Notation standard	Notation alternative	type des opérandes	type du résultat
`if - then - else`		logique, arithmétique	arithmétique
`or`	`||`	logique	logique
`exists forall`		logique	logique
`and`	`&&`	logique	logique
`not` (unaire)	`!`	logique	logique
`< <= = <> > >=`	`< <= == != > >=`	arithmétique	logique
`in not in`		objet, ensemble	logique
`+ - less`		arithmétique	arithmétique
`sum prod min max`		arithmétique	arithmétique
`* / div mod`		arithmétique	arithmétique
`+ -` (unaires)		arithmétique	arithmétique
`^`	`**`	arithmétique	arithmétique

Voici un tableau des principales fonctions mathématiques.

`abs(x)`	valeur absolue
`ceil(x)`	plus petit entier supérieur ou égal
`exp(x)`	exponentielle e^x
`floor(x)`	plus grand entier inférieur ou égal
`log(x)`	logarithme neperien $\ln x$
`log10(x)`	logarithme en base 10
`sqrt(x)`	racine carrée

5.6 Quelques commandes de l'interprète AMPL

La commande « *model nom-de-fichier* » permet de charger un modèle en mémoire. Il est parfois utile d'enregistrer dans des fichiers différents le modèle et ses données (par exemple dans le cas où on souhaite traiter différents jeux de données). Dans ce cas, on charge en mémoire le modèle avec la commande « *model nom-de-fichier* » et les données avec la commande « *data nom-de-fichier* ».

La commande « *reset* » réinitialise la mémoire de l'interprète. Elle est utile en particulier lorsqu'on souhaite utiliser la commande « *model* » plusieurs fois de suite (lors de la mise au point des modèles).

```
ampl: model conges.ampl;
ampl: model conges.ampl;

conges.ampl, line 1 (offset 4):
        PRODUITS is already defined
context:  set  >>> PRODUITS; <<<

...
```

```
conges.ampl, line 13 (offset 390):
        stock_final_impose is already defined
context:  param  >>> stock_final_impose  <<< >= 0;

Bailing out after 10 warnings.
ampl: reset;
ampl: model conges.ampl;
```

La commande « *option solver nom-de-solveur* » spécifie le solveur à utiliser.

La commande « *solve* » permet de résoudre un modèle chargé en mémoire.

La commande « *display* » permet d'afficher les valeurs des variables, les valeurs marginales des contraintes etc … Elle permet d'afficher plusieurs données en même temps. Sur l'exemple, on affiche une variable et ses bornes inférieure et supérieure. Ensuite, on affiche une la valeur du corps d'une contrainte et sa valeur marginale.

```
mpl: model conges.ampl;
ampl: option solver cplex;
ampl: solve;
CPLEX 8.0.0: optimal solution; objective 10231.17857
9 dual simplex iterations (1 in phase I)
ampl: display profit_mensuel.lb, profit_mensuel, profit_mensuel.ub;
:   profit_mensuel.lb profit_mensuel profit_mensuel.ub    :=
Jan      -Infinity        1984            Infinity
Fev      -Infinity        1519.68         Infinity
Mar      -Infinity        1487            Infinity
Avr      -Infinity         800            Infinity
Mai      -Infinity        1855            Infinity
Jun      -Infinity        2585.5          Infinity
;
ampl: display limitation_de_la_vente.body, limitation_de_la_vente;
:      limitation_de_la_vente.body limitation_de_la_vente    :=
Jan P1             55                       7
Jan P2             50                       1.8
...
Jun P5             95                       9.2
Jun P6             55                       2.4
Jun P7             34.5                     0
```

La contrainte *limitation_de_la_vente* est indicée par des mois et des produits. Il est possible de n'afficher que les valeurs marginales du mois de mai en utilisant la syntaxe suivante.

```
ampl: display {p in PRODUITS} limitation_de_la_vente ["Mai", p];
limitation_de_la_vente['Mai',p] [*] :=
P1  10
P2   6
P3   8
P4   4
P5  11
P6   9
P7   3
;
```


La commande *let* permet de modifier la valeur d'un paramètre sous l'interprète. Dans l'exemple suivant, *prix* est une variable, *delta* et *prix_approximatif* sont des paramètres.

```
ampl: model elasticite3.ampl;
ampl: option solver cplex;
ampl: solve;
CPLEX 8.0.0: optimal solution; objective 1992470.886
79 dual simplex iterations (0 in phase I)
ampl: let {p in PRODUITS_LAITIERS} prix_approximatif[p] := prix[p];
ampl: let delta := delta / 2; solve;
CPLEX 8.0.0: optimal solution; objective 1993220.536
37 dual simplex iterations (2 in phase I)
```