Introduction to AMPL

Instructions to download a full version of AMPL (with a time-restricted license) have been sent by email. All chapters from the AMPL book are available as a PDF document from the above web page. Go through the first chapter in detail—most of the basics necessary to get started are given in Chapter 1.

1 AMPL Basics

AMPL is a modeling language which allows the user to represent optimization models in a compact and logical manner. The data (for instance, demand for each month, amount of raw material available, distance between cities, etc.) is handled separately from the optimization model itself (which consists of the decision variables, objective function, and constraints). Thus the user need not alter the original model each time a small change is made in the data. You need to create a model file (for example FarmerJones.mod.txt) and a data file (FarmerJones.dat.txt). The model file declares the *data parameters*, the variables, objective function, and the constraints in a symbolic fashion. All the numbers (actual data) are provided in the data file. Note the following facts.

- It is a good convention to name model files as `something.mod`, and data files as `something.dat` (AMPL will accept *any* name for the model and data files though). To help your computer identify the files as text files, it might be a good practice to add a ".txt" extension to the file names, e.g., `something.mod.txt`.
- Every declaration (of a parameter, variables, objective function or a constraint) ends with a **;** (semicolon). This is true for both the model and the data files.
- You can specify values for a parameter in the data file only after it is already declared in the model file. Else, AMPL will give an error.
- Before solving problems, you need to specify the solver. We will be using Gurobi as the solver. It is a good practice to start each AMPL session by typing `option solver gurobi;` at the `ampl:` prompt.

1.1 Example: The Farmer Jones problem

(Taken from *Introduction to Mathematical Programming* by Winston and Venkataramanan.)

Farmer Jones must decide how many acres of corn and wheat to plant this year. An acre of wheat yields 25 bushels of wheat and requires 10 hours of labor per week. An acre of corn yields 10 bushels of corn and requires 4 hours of labor per week. Wheat can be sold at $4 per bushel, and corn at $3 per bushel. Seven acres of land and 40 hours of labor per week are available. Government regulations require that at least 30 bushels of corn need to be produced in each week. Formulate and solve an LP which maximizes the total revenue that Farmer Jones makes.

The model and data files are given below. You can use any text editor to create the model and data files. For instance, **Notepad** works well in Windows. You could use MS Word or WordPad, but make sure you save the files as *text only* documents (which could be tricky, unfortunately). **Vi** or **Emacs** could be used in Unix/Linux machines.

Model file: FarmerJones.mod.txt

```
# AMPL model file for the Farmer Jones problem
# The LP is
#      max Z = 30 x1 + 100 x2         (total revenue)
#      s.t         x1 +     x2 <= 7   (land available)
#                 4 x1 +  10 x2 <= 40  (labor hrs)
#                10 x1           >= 30  (min corn)
#                   x1, x2       >= 0   (non-negativity)


set   Crops;                 # corn, wheat
param Yield    {j in Crops};  # yield per acre
param Labor    {j in Crops};  # labor hrs per acre
param Price    {j in Crops};  # selling price per bushel
param Min_Crop {j in Crops};  # min levels of corn and wheat (bushels)
param Max_Acres;          # total land available (acres)
param Max_Hours;        # total labor hrs available

var   x {j in Crops} >= 0; # x[corn], x[wheat] = acres of corn, wheat

maximize total_revenue: sum {j in Crops} Price[j]*Yield[j]*x[j];

s.t. land_constraint:  sum {j in Crops} x[j] <= Max_Acres;
s.t. labor_constraint: sum {j in Crops} Labor[j]*x[j] <= Max_Hours;
s.t. min_crop_constraint {j in Crops}: Yield[j]*x[j] >= Min_Crop[j];
```

Data file: FarmerJones.dat.txt

```
set Crops := corn wheat;

param  Yield :=
corn  10
wheat 25;

param  Labor :=
corn   4
wheat 10;

param  Price :=
corn   3
wheat  4;

param Min_Crop :=
corn  30
wheat  0;          # No minimum level specified for wheat

param Max_Acres :=  7;
param Max_Hours := 40;
```

More points to note:

- Comments are given using the symbol #. Everything after a # in a line are ignored by AMPL.
- Symbolic *parameters* are declared for data, whose actual values are specified in the data file. Any parameter is declared using the keyword `param`.
- Variables are declared using the keyword `var`.
- The objective function starts with a `maximize` or a `minimize`, followed by a name, and then a colon (**:**). The actual expression of the objective function then follows.
- Each (set of) constraint(s) begins with the keyword `subject to` (or `s.t.` in newer versions) followed by a constraint name, possible indexing, and then a colon (**:**). The expression for the constraint(s) follows.
- Each constraint and objective function must have a unique name.
- Nonnegativity and other sign restrictions are declared along with the variable declarations. If no sign restrictions are provided, the variable(s) are considered *unrestricted in sign (urs)*.

1.2 Running AMPL, Output from AMPL session

To start an AMPL session in Windows, double-click on the executable names `sw.exe`. A **s**crollable **w**indow will open with the prompt `sw:`. Type `ampl` and press enter to get the `ampl:` prompt. In Unix/Linux machines, run the ampl executable to get the `ampl:` prompt. Here are the commands and output from an AMPL session to solve the Farmer Jones LP.

```
ampl: option solver gurobi;
ampl: model FarmerJones.mod.txt; data FarmerJones.dat.txt;
ampl:  expand land_constraint;
subject to land_constraint:
 x['corn'] + x['wheat'] <= 7;

ampl: expand min_crop_constraint;
subject to min_crop_constraint['corn']:
 10*x['corn'] >= 30;

subject to min_crop_constraint['wheat']:
 25*x['wheat'] >= 0;

ampl: solve; display x;
Gurobi 10.0.0: optimal solution; objective 370
2 simplex iterations
x [*] :=
 corn  3
wheat  2.8
;
ampl: display total_revenue;
total_revenue = 370
```

If there is any error in the model file, AMPL will point it out. You will have to make the appropriate corrections in your `.mod` file (model file) and save it. In order to re-read the model file, you first need to give the command `reset;` at the `ampl:` prompt. Then say `model FarmerJones.mod;` again. If there was no error in the model file, but there is an error in the data file, you could reset just the data part by typing `reset data;`. This commands leaves the model file in tact. The modified data file could then be read in using the `data` command.

The command `display` can be used to display the value(s) of a (set of) variables or the objective function. In the above LP, if you give the command `display land_constraint;` after solving the LP, AMPL will display the value of the dual variable corresponding to the constraint (which is 0 in this case). The command `expand` is used to display the actual expression of a constraint or an objective function.

1.3 Another Example: Inventory problem

(Taken from *Introduction to Mathematical Programming* by Winston and Venkataramanan.)

A customer requires 50, 65, 100, and 70 units of a commodity during the next four months (no backlogging is allowed). Production costs are 5,8,4, and 7 dollars per unit during these months. The storage cost from one month to the next is $2 per unit (assessed on ending inventory). Each unit at the end of month 4 could be sold at $6. Use LP to minimize the net cost incurred by the customer in meeting the demands for the next four months.

Model file: InvProb.mod.txt

```
# AMPL model file for the inventory model
# (WV-IMP problem 1 from page 104)
#
#   min  z =  5 x1 + 8 x2 + 4 x3 + 7 x4 + 2 (s1+s2+s3) - 6 s4 (net cost)
#
#   s.t. s1 = x1     -  50  (inventory month 1)
#  s2 = x2+s1  -  65  (inventory month 2)
#  s3 = x3+s2  - 100  (inventory month 3)
#  s4 = x4+s3  -  70  (inventory month 4)
#        all vars >= 0      (non-negativity)


param n;                        # No. of months
param Demand   {i in 1..n};    # demand for each month
param Cost     {i in 1..n};    # production cost for each month
param Store_Cost;           # storage cost (same for each month)
param Price;                    # selling price (at the end of month n)

var   x {j in 1..n} >= 0;      # No. units made in month j
var   s {j in 0..n} >= 0;      # inventory at the end of month j
         # s[0] - inventory at the start of month 1 = 0
```

```
minimize net_cost:
      sum {k in 1..n} Cost[k]*x[k] + Store_Cost*(sum{j in 0..n-1} s[j])
      - Price*s[n];

s.t. inventory_balance {i in 1..n}: s[i] = x[i] + s[i-1] - Demand[i];
s.t. set_initial_inventory: s[0] = 0;
```

Data file: InvProb.dat.txt

```
param n := 4;
param Demand :=
1   50
2   65
3  100
4   70;
param Cost :=
1   5
2   8
3   4
4   7;
param Store_Cost := 2;
param Price := 6;
```

Notice how all the inventory balance constraints are represented in a single line (under inventory_balance). Here is the output from AMPL where the above LP is solved.

```
ampl: reset; model InvProb.mod.txt; data InvProb.dat.txt;
ampl: solve; display x,s;
Gurobi 10.0.0: optimal solution; objective 1525
0 simplex iterations
:    x    s     :=
0    .     0
1   115   65
2     0    0
3   170   70
4     0    0
;
```

1.4 An integer programming example

Binary and integer variables can be declared by adding the keyword `binary` or `integer` to the variable declaration. This keyword is included (usually) after possible bounds (separated by commas) before the final semi-colon (**;**). Here is an example. (Taken from *Introduction to Mathematical Programming* by Winston and Venkataramanan.)

Because of excess pollution on the Momiss river, the state of Momiss is going to build pollution control stations. Three sites (1,2, and 3) are under consideration. Momiss wants to control the levels of two pollutants (1 and 2). The state legislature requires that at least 80,000 tons of pollutant 1 and at least 50,000 tons of pollutant 2 be removed from the river. The relevant data for the problem is shown in the table. Formulate and solve an IP to minimize the cost of meeting the state legislature's goals.

			Amt removed per ton of water	
site	station building cost ($)	cost of treating 1 ton of water ($)	pollutant 1	pollutant 2
1	100,000	20	0.40	0.30
2	60,000	30	0.25	0.20
3	40,000	40	0.20	0.25

Model File: MomissIP.mod.txt

```
# model file for Momiss river MIP - problem 2 from page 502
param n_Sites;
param n_Polut;
param Cost_Treat {j in 1..n_Sites};
param Cost_Build {j in 1..n_Sites};
param Polut_Removed {i in 1..n_Polut, j in 1..n_Sites};
param Min_Polut {i in 1..n_Polut};

var  x {j in 1..n_Sites} >= 0;
var  y {j in 1..n_Sites} binary;
#var  y {j in 1..n_Sites} >=0, <=1, integer; # this declaration works too

param M;

minimize total_cost: sum {j in 1..n_Sites} Cost_Treat[j]*x[j] +
       sum {j in 1..n_Sites} Cost_Build[j]*y[j];

s.t. min_polut_removed {i in 1..n_Polut}:
      sum {j in 1..n_Sites} Polut_Removed[i,j]*x[j] >= Min_Polut[i];

s.t. forcing_cons {j in 1..n_Sites}: x[j] <= M*y[j];
```

Data File: MomissIP.dat.txt

```
# data file for  Momiss river MIP - problem 2 from page 502
param n_Sites := 3;
param n_Polut := 2;

param Cost_Treat :=
1  20
2  30
3  40;

param Cost_Build :=
1  100000
2   60000
3   40000;

param Polut_Removed :   1     2    3 :=
1       0.40  0.25 0.20
2       0.30  0.20 0.25 ;

param Min_Polut :=
1  80000
2  50000;

param M := 400000;
```

Output from AMPL session

```
ampl: reset; model MomissIP.mod.txt; data MomissIP.dat.txt;
ampl: solve; display x,y;
Gurobi 10.0.0: optimal solution; objective 4100000
0 simplex iterations
1 branching nodes
:     x     y    :=
1   2e+05   1
2       0   0
3       0   0
;
```

2 For more

You should read the AMPL book to learn more about AMPL. A whole suite of examples are also available as part of the Ampl package (also for free download).