Introduction to AMPL (Math 364, Fall 2024)

Details of how to download and start up AMPL are given in http://www.ampl.com. Download and install AMPL in your laptop or PC (you could install it on multiple machines if you have access). The bundle for this class (see email sent on Sep 10) will come with the preferred solver Gurobi. The entire AMPL book is available from the above web page. Go through the first chapter in detail.

1 AMPL Basics

AMPL is a modeling language that 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) and a data file (FarmerJones.dat). 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.

- It is a good convention to name model files as `something.mod`, and data files as `something.dat` (although, AMPL will accept *any* name for the model and data files).
- Every declaration (of a parameter, variables, objective function or a constraint) ends with a **;** (semi-colon). This is true for both the model and the data file.
- You can specify values for a parameter in the data file only if the parameter is already declared in the model file.
- Before solving problems, you need to specify the solver. We will be using Gurobi as the solver. As a good practice, 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, but you must save the files as *text only* documents. Standalone editors such as **Vi** or **emacs** could be used as well.

Model file: FarmerJones.mod

```
# Model file for Farmer Jones LP using a set of crops
# 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, etc.
param Yield {j in Crops};   # Yield[j] is the yield per acre of crop j
param LaborHrs {Crops};     # {j in Crops} or {Crops} work equally;
param SellPrice {Crops};    # {j in Crops} for emphasis
param MinCrop {j in Crops}; # MinCrop['corn'] = 30, for instance

param Land;                 # Total land available in acres
param TotalLaborHrs;

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

maximize Total_Revenue:  sum {j in Crops} SellPrice[j]*Yield[j]*x[j];

subject to Land_Available: sum {j in Crops} x[j] <= Land;
subject to Labor_Hrs_Limit: sum {j in Crops} LaborHrs[j]*x[j] <= TotalLaborHrs;
subject to Min_Crop_Limit {j in Crops}: Yield[j]*x[j] >= MinCrop[j];
```

Data file: FarmerJones.dat

```
set Crops := corn wheat;

param Yield :=
corn  10
wheat 25;

param LaborHrs :=
corn   4
wheat 10;

param SellPrice :=
corn   3
wheat  4;

param MinCrop :=
corn   30
wheat   0; # no min production level for wheat

param Land := 7;                # Total land available in acres
param TotalLaborHrs := 40;
```

More points to note:

- Comments can be included using the symbol #. Everything after a # in a line are ignored.
- 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` followed by a constraint name, possible indexing, and then a colon (**:**). The expression for the constraint(s) follows.
- Each constraint (set) and objective function must have a unique name.
- Non-negativity 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 named `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.

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

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

subject to Min_Crop_Limit['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 as before.

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_Available;` 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
# 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];

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

Data file: InvProb.dat.txt

```
# Data file for Inventory planning LP from Lecture 6

param n := 4;           # No. of months

param Demand :=
1  50
2  65
3 100
4  70;

param Cost :=
1 5
2 8
3 4
4 7 ;


param Store_Cost := 2;  # same storage cost for each month
param Price := 6;  # selling price (at end of month n)
```

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
ampl: option solver gurobi;
ampl: 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
;
ampl:
```

2 For more

You should read the AMPL book to learn more about AMPL. Many examples are made available on the AMPL web page, as well as the AMPL bundle you download for the class.