AMPL is not Excel, and an FAQ that I get is: “Why not just use Excel?”.

Indeed, for small LPs (or integer programs) Excel is very convenient. For larger and more complex LPs, however, AMPL is much better. For example, we will see soon an inventory problem with two *kinds* of variables: the first kind is for amount of stuff purchased; the second kind is for amount of stuff stored from one month to another month. And there are 5–6 variables of each kind. This problem would already be pretty hard to set up in Excel. In the real world, there are LPs with 20 *kinds* of variables, and there may be dozens of variables of each kind. For example, purchasing, inventory, subcontracting, etc. Those kinds of LPs would be very hard to set up in Excel.

2 Basic AMPL

2.1 Giapetto problem as covered in class

Recall the problem from class.

$$\begin{array}{rl} \max & 3x_1 + 2x_2 \\ s.t. & x_1 + x_2 \leq 80 \text{ (carpentry)} \\ & 2x_1 + x_2 \leq 100 \text{ (finishing)} \\ & x_1 \leq 40 \text{ (soldier demand)} \\ & x_1, x_2 \geq 0 \end{array}$$

The AMPL file that codes this LP looks like this:

```
File: giapetto.mod

var x1 >=0; # number of soldiers
var x2 >=0; # number of trains

maximize profit: 3*x1 + 2*x2;

subject to carpentry: x1 + x2 <= 80;
subject to finishing: 2*x1+x2 <= 100;
subject to soldierdemand: x1 <= 40;
```

Main points:

1. Things are in the usual order: variables, objective, and constraints.
2. The following are AMPL key words: “var”, “maximize”, “minimize”, “subject to”.
3. The words “profit”,, “carpentry”, etc. were chosen by the user.
4. We should keep in mind:
   (a) multiplication sign.
   (b) semicolon at the end: many of the coding mistakes are just forgetting the semicolon.
   (c) colon, in the definition of the objective and constraints.
5. Everything after the # sign is comment.

```
AMPL run looks like this:

ampl: model giapetto.mod;
ampl: option solver cplex;
ampl: solve;
CPLEX 20.1.0.0: optimal solution; objective 180
2 dual simplex iterations (1 in phase I)
ampl: display x1, x2;
x1 = 20
x2 = 60

ampl:
```

Main points:

1. AMPL is not really a "programming language" like Java or C, although it can be used as one (for example, we can add two vectors useing AMPL).
2. AMPL is a modeling language: it passes the LP to a solver, which implements the *simplex method*, a nice algorithm to solve LPs.
3. The company's name whose implementation we use is CPLEX, that is why we have to use the "option solver cplex" command. This just tells AMPL to use the specialized solver of CPLEX, which is specifically tailored to solve *linear* programs. (We can just type this one line automatically).
4. The action really begins after we typed "solve". Then the simplex method is run, and it delivers the solution, which we can then display.
5. If we solve a problem, and we then want to solve another one, we need to type "reset". That tells AMPL to get ready for a new problem.
6. If we make a mistake, then we need to:
   (a) Fix the mistake; then
   (b) Type "reset", and restart the whole process.

2.2 Ad problem as covered in class

```
File: ads. mod

var x1 >=0; # number of comedy ads
var x2 >=0; # number of football ads

minimize cost: 50*x1+100*x2;

subject to men: 2*x1 + 12*x2 >= 24;
subject to women: 7*x1+2*x2 >= 28;
```

3 An exercise – making candy

Let us get our hands dirty, and do a little exercise!

Tar Heel Candy, Inc. has 1500 lb of chocolate, 230 lb of nuts, and 150 lb of fruit available. A box of the Special Mix requires 3 lb of chocolate, 1 lb each of nuts and fruit, and it brings \$10 profit. A box of the Regular Mix requires 4 lb of chocolate, 0.5 lb of nuts, and no fruit, and brings \$6 profit. A box of the Purist Mix requires 5 lb of chocolate, no nuts or fruit, and it brings in \$4.

How many boxes of each should we produce to maximize our profit? Hint: the optimal profit is \$2788.

4 Advanced AMPL: separate model and data file

4.1 The caffeine problem

To get started, let us implement the caffeine problem as we covered it in class.

First, let us recap it.

We have 3 types of food available: Mountain Dew, Latte, and Coke.

The caffeine and sugar content as well as the price of each is given below:

	Mt Dew, 1 bottle	Latte, 1 cup	Coke, 1 bottle
Caffeine (g)	110	200	60
Sugar (g)	25	15	30
Price (\$)	1.5	2	1.2

If I want to have at least 1000 mg of caffeine and 100 g of sugar in my diet, how should I satisfy these needs at minimum cost?

The linear program is:

$$\begin{array}{rrcl} \min & 1.5x_1 + 2x_2 + 1.2x_3 & & \\ s.t. & 110x_1 + 200x_2 + 60x_3 & \geq & 1000 \\ & 25x_1 + 15x_2 + 30x_3 & \geq & 100 \\ & \text{all } x_i \geq 0. & & \end{array} \tag{1}$$

The AMPL code is below:

```
File: caff-simple.mod

var x1 >=0; # amount of Mt. Dew
var x2 >=0; # amount of Latte
var x3 >= 0; # amount of Coke

minimize mycost: 1.5*x1 + 2*x2 + 1.2*x3;

subject to caffeine: 110*x1 + 200*x2 + 60*x3 >= 1000;
subject to    sugar: 25*x1+15*x2+30*x3 >= 100;
```

Now suppose we introduce more foods: cup of Earl Grey tea, diet Coke, Pepsi, etc. In that case, we will need to rewrite the entire problem for 4, 5 etc foods, which is inconvenient. Instead, we will write a separate model file, which will contain the essential structure of the problem, and a separate data file which will have the data.

Precisely, let us first write the caffeine problem as

$$\begin{array}{ll} \min & c^T x \\ s.t. & Ax \geq b \\ & x \geq 0, \end{array} \tag{2}$$

and specify the data as:

$$A = \begin{pmatrix} 110 & 200 & 60 \\ 25 & 15 & 30 \end{pmatrix}, \; c^T = \begin{pmatrix} 1.5 & 2 & 1.2 \end{pmatrix}, b = \begin{pmatrix} 1000 \\ 100 \end{pmatrix}. \tag{3}$$

When we create a separate model and separate data file in AMPL, we do the same thing: the model file codes the information in (2) and the data file codes the information in (3). It looks like this:

```
File: caff.mod

param n;  # Number of  foods
param m; # Number of nutrients

param A {i in 1..m, j in 1..n}; # A[i,j]=amount of nutrient i in food j
param b {i in 1..m};                #  Demand for nutrient i
param c {j in 1..n};                  # Cost of food j

var x {j in 1..n} >=0;

minimize cost:  sum {j in 1..n} c[j] * x[j];

subject to cons {i in 1..m}:
sum {j in 1..n} A[i,j]*x[j] >= b[i];


File: caff.dat

param n :=3;  # n foods: Mt Dew, latte, Coke
param m :=2; # 2 nutrients: caffeine and sugar


param A:   1 2 3 :=
1       110 200 60
2       25   15   30;

# food 1 has 110 mg of nutrient 1 in it, food 1 has 25 g of nutrient 2 in it, etc.
param b :=
1  1000
2 100;
```

```
param c :=
1  1.5
2 2
3  1.2;
```

Main points:

1. The matrix is written in a not–too-nice format. The 1,2,3 on top means there are 3 columns; the 1,2 next to it means there are 2 nutrients. Don't forget the ":=" sign.

2. The constraints in the model file say:

$$\sum_{j=1}^{n} A_{ij}x_j \geq b_i \text{ for } i = 1, \ldots, m.$$

   For example, when $i = 1$, we get the first constraint:

$$A_{11}x_1 + A_{12}x_2 + A_{13}x_3 \geq b_1$$

   With numbers, we get

$$110x_1 + 200x_2 + 60x_3 \geq 100.$$

Exercise: What does the 2nd constraint look like in (2)? Once you plug in the numbers given in (3), what is the constraint?

AMPL run looks like this:

```
ampl: model caff.mod;
ampl: data caff.dat;
ampl: solve;
CPLEX 20.1.0.0: optimal solution; objective 10.58823529
2 dual simplex iterations (0 in phase I)
ampl: display x;
x [*] :=
1  0
2  4.70588
3  0.980392
;
```

Main points:

1. When we write "param", it means we are declaring *parameters.* Parameters are *data*; they must be declared in the model file, but their values should be only specified in the dat file.

2. If we make a mistake in the model file, we need to fix it, then type “reset” and then again load the model file.

3. Suppose the model file is correct, but we make a mistake in the data file. Then we need to type “reset data” and then load *the data file only.* That is, AMPL will remember the correctly loaded model file.

4.2 Computing Fibonacci numbers

Recall that the Fibonacci sequence is defined as $F_1 = F_2 = 1, F_i = F_{i-1} + F_{i-2}$ for $i \geq 3$.

The following AMPL code computes Fibonacci numbers.

Of course, doing such basic math is not the main point of AMPL. But this nicely illustrates the separation of model and data file.

```
File:  fib.mod

param n;  # Number of variables

var f {j in 1..n};

maximize whatever: f[n];

subject to init1: f[1]=1;
subject to init2: f[2]=1;

subject to cons {i in 3..n}:
f[i-2]+f[i-1]=f[i];
```

File fib.dat

```
param n :=10;
```

—

AMPL run:

```
ampl: model fib.mod;
ampl: data fib.dat;
ampl: solve;
Solution determined by presolve;
objective whatever = 55.
```

---

5 How to write good AMPL code

This is a rough breakdown of steps on how to write correct AMPL code.

1. Write the LP on paper. Specify what is the data, and what are variables, objective and constraints. That is, do not start writng the AMPL code.

2. Walk before you run: Write an AMPL code with *only* a mod file. (Like giapetto.mod, ads.mod, caff-simple.mod).

3. In many case, we can write the LP on paper, as

$$\begin{array}{rl} \min & c^T x \\ s.t. & Ax \geq b \\ & x \geq 0, \end{array} \tag{4}$$

   OR, "min" can be replaced by "max", the $\geq$ sign with $\leq$, etc. It depends on the problem.

4. Write down on paper what is A, b, c. That is, write the elements of A, b, c.

5. Do some spot checking: what is the first, second, etc. constraint in your LP with numbers?

6. Write the mod file. Make sure it is in the same form as (4).

7. Write the dat file. The "param" part of the dat file should contain what you specified as data. Make sure to put the right A, b, c into it. (What you just wrote on paper).

   Some other problems, like the KWOil problem are not convenient to write in this form with the A, b, c. That is because that has two kinds of variables, the x_i and the y_i. Still, even those LPs you should write on paper first.

8. Run it.

9. Display the solution. Check whether it makes sense.

6 Another exercise

Revisit the candy problem from Section 3 and write a sepaarte model and data file to implement that code. Then run it.