Getting Started With AMPL

This is a basic tutorial for AMPL to get you started. This tutorial only covers the basics of syntaxes for linear programming using *cplex* solver.

- **Basic Syntax**
  - Case-sensitive
  - Every statement ends with a semi-colon ";"
  - Comments start with a pound symbol "#"
  - Block comments are enclosed in "/*" and "*/" i.e. /* multi-line comment */
- **Basic Files**
  - .mod: Contains elements of the actual model e.g. variables, sets, parameters, objective, constraints etc. In this example, we create a file called "ampl_intro_tutorial.mod" that is available on canvas.
  - .dat: Contains input data for the model. In this example, we create a file called "ampl_intro_tutorial.dat" that is available on canvas.
- **Importing solver:** You should start your model script (.mod) by importing the solver you need for your problem and also a *reset* statement –

```
reset;
option solver cplex;
```

  cplex is the solver to import. The reset statement resets all the input and output variables so when you rerun your script, you won't run into any unexpected errors due to previous values.
- **Defining sets:** Set can be defined using the following syntax –
  - In the mod file –

```
#Example of defining sets
set MONTH;
set YEAR;
```

  - In the dat file –

```
set MONTH := feb mar apr;
set YEAR := y2019 y2020 y2021;
```

Here MONTH and YEAR are sets of 3 months and 3 years respectively. The sets are defined in the model files and their values are stored in the data file. We will see this pattern when defining every input parameter.

- **Defining parameters:**
  - Single-valued parameters in mod file

```
#Example of defining simple, single-valued parameters
param type; #product types
param purchase_cost; #cost of purchasing pre-made products
param making_cost; #cost of making a product
param selling_price; #selling price of each unit
```

  - Single-valued parameters in dat file

```
param type := 5; #5 different product types
param purchase_cost := 10;
param making_cost := 15;
param selling_price := 25;
```

Notice the *type* parameter. It is a single valued parameter i.e. it has a value of 5. However, we will use it like a set as you will see in the next bullet points.

- Multi-valued parameters in mod file

```
#Example of multi-valued parameters that has values for each value of a pre-defined set
param demand {1 .. type}; #demand for each product type
param budget {YEAR}; #yearly budget
```

These parameters are sets because they have a value for each value in the set they are based on. Here, *{1 .. type}* is a set of 5 values *[1,2,3,4,5]* since *type* has a value of 5 and correspondingly, demand is a set of 5 values as you will find in the dat file displayed in the next bullet point.

- Multi-valued parameters in dat file

```
param demand :=
1       100
2       50
3       30
4       90
5       75;

param budget :=
y2019       500
y2020       200
y2021       750;
```

The actual values of the parameter are on the right column and the indices are on the left column.

- Multi-dimensional parameters in mod file

```
#Example of a 2 dimensional multi-valued parameter
#i.e. a parameter that depends on 2 pre-defined sets
param limit {MONTH, YEAR}; #maximum number of items that can be produced per term
```

The limit parameter has a value for each month and year i.e. it is a 2-dimensional set/array

- Multi-dimensional parameters in dat file

```
param limit:    y2019   y2020   y2021 :=
feb             75      50      100
mar             100     45      75
apr             60      40      90;
```

Month indices are on the left-most column and Year indices are the top-most row.

- **Defining decision variables along with range constraints**
  - Decision variables are only defined in the mod file
  - Single-valued decision variable with a single non-negativity constraint

```
#Example of a single-valued decision variable with a single constraint
var purchase >= 0; #Number of products to be purchased at the beginning
```

We could have simply defined it like a parameter i.e. *var purchase;*
But we included the non-negativity constraint along with the variable declaration to keep the code neat and simple. Almost every variable has a range of values it can take on so, it is easier to declare the variable along with its range constraint.

- Multi-valued decision variable with two range constraints

```
#Example of a multi-valued decision variable with multiple constraints
var x {m in MONTH,y in YEAR} >=0, <= limit[m, y]; #Number of products to be produced each month
```

Here again, we could have simply defined the multi-valued variables like we did multi-valued parameters e.g. *var x {MONTH, YEAR}.* But using aliases *m* and *y* helped us set the multi-valued constraint *limit[m, y]*.

- **Defining objective function**
  - Objective function is only defined in the mod file
  - Example of a maximizing objective function

```
#------------------------Objective Function-----------------------
#Example of a maximizing objective function
maximize profit: selling_price*(purchase + sum {m in MONTH,y in YEAR} x[m,y]) -
((purchase_cost*purchase) + (making_cost*sum {m in MONTH,y in YEAR} x[m,y]));
```

We use the key word *maximize* then, name the function *profit* and write out the function after a colon. Here we are maximizing profit by summing (*sum* keyword) the total number of products multiplied by selling cost and subtracting total number of products multiplied by production cost.
If we needed a minimizing objective function, we would simply use the *minimize* keyword.

- **Defining constraints**
  - Constraints are only defined in mod file

```
#------------------------Constraints------------------------
subject to yearly_budget {y in YEAR} : sum {m in MONTH} making_cost*x[m, y] <= budget[y]; #For
s.t. pre_made_limit : purchase_cost*purchase <= 100; #pre-purchase product cost can not be more
```

  - Constraints can be defined using the keywords *subject to* or *s.t.* Either of them is fine.
  - Constraints also have to be named. The first constraint *yearly_budget* is a set of constraints with indices (the indices would be the years). So we have a budget constraint for every year and they can be accessed with proper indices.
  - The second constraint *pre_made_limit* is a simple single constraint.
- **Connecting to data file**
  - In mod file

```
#------------------------Data------------------------
data ampl_intro_tutorial.dat;
```

Use the *data* keyword and specify the data file name along with appropriate file path. If the data file is not in the same folder and you need to specify a file path, it's safer to use quotations e.g. *data "/path/to/data/file/myData.dat";*

  - In dat file

```
data;
```

In the data file, start the file with the keyword *data*.

- **Other Commands: Solving, Displaying and Looping**
  - Use the *solve* command to solve the problem

```
solve; #solve the LP problem
```

After running the command..

```
 ampl: model ampl_intro_tutorial.mod;
CPLEX 20.1.0.0: optimal solution; objective 1116.666667
0 dual simplex iterations (0 in phase I)
```

  - Use the *display command* to display variables or parameters

```
#Example of simply displaying
display purchase; #Display the purchase variable
display budget; #Display the budget parameter
```

After running the command..

```
purchase = 10

budget [*] :=
y2019  500
y2020  200
y2021  750
;
```

  - Use the *expand* command to display constraints. For set of constraints (e.g. *yearly_budget*), they can be indexed

```
expand yearly_budget["y2020"]; #Show the yearly_budget constraint at y2020 index
```

After running the command..

```
subject to yearly_budget['y2020']:
        15*x['feb','y2020'] + 15*x['mar','y2020'] + 15*x['apr','y2020'] <= 200;
```

  - We can also print with formatting instead of just displaying

```
#Example of nicely printing
printf "-----Solution----- \n";
printf "Number of pre-made products: %d \n", purchase;
for {y in YEAR} {
    printf "Nmmber of products made in February of %s: %d \n", y, x['feb',y];
}
printf " \n ";
```

This example also shows the syntax of a for loop.

After running the command..

```
-----Solution-----
Number of pre-made products: 10
Nmmber of products made in February of y2019: 33
Nmmber of products made in February of y2020: 13
Nmmber of products made in February of y2021: 50
```

  - The display commands are useful for debugging and the print commands are useful for nicely presenting your final solution/answers. In the homework, please comment out the debugging codes and only keep the final solution.