Modelling and solving optimization problems with AMPL: tutorial
Material for the PGE 305 class: continuous optimization.

Laurent Pfeiffer[*]

2021-2022

Contents

[*]Inria-Saclay and CentraleSupélec, Université Paris-Saclay. E-mail : laurent.pfeiffer@inria.fr. This script is a slightly modified version of the one written by Anna Désilles in 2018-2019.

1 Describing an optimization problem

1.1 Generalities

Modelling a problem with AMPL can be done in two steps:

- Creation of a model: the decision variables, the parameters, the cost function, and the constraints are declared.
- Initialization: particular numerical values are assigned to the parameters. The set of those values is called **instance of the model**.

Example. Let us consider a least-square model formulated so as to identify the coefficients of a second-order polynomial function, using a set of measurements:

$$(x_i, y_i),\ i = 1, \dots, N$$

. We look for the coefficients $(a \geq 0, b \geq 0, c)$ of some function:

$$f(x) = ax^2 + bx + c$$

minimizing the criterion

$$J(a, b, c) = \sum_{i=1}^{N} (f(x_i) - y_i)^2.$$

This description corresponds to the mathematical expression of the model. It depends here on three parameters: the number of available measurements N, the vector $\{x_i,\ i = 1, \dots, N\}$ and the vector $\{y_i,\ i = 1, \dots, N\}$, representing the measurements. The solution to the problem depends of course on the parameters.

The AMPL command prompt has two main working modes, corresponding to the two description steps indicated above:

model This mode allows to declare the different elements of the problem (variable, parameters, cost, constraints). This is the default mode. It is employed with the command `model`, followed by the name of the file to be loaded:

```
model modeleR.txt;
```

data This mode allows to create an instance of a model. It is used with the key word `data;` followed by the name of the file to be loaded:

```
data dataSet.txt;
```

The following commands allow to solve the problem and to display the results:

- `solve`: allows to solve the loaded problem. Write:

```
solve;
```

- `display`: allows to display the values of the variables or parameters. For example:

```
display x,y;
```

allows to display the decision variables x and y.

- `reset`: reset the memory of the command prompt
- `quit`: exit the command prompt.

Remak. AMPL distinguishes lower-case and upper-case letters. Line breaks are ignored. The character ; is mandatory at the end of each instruction. If it is forgotten, an error message will be generated by the console.

1.2 Decision variables

Decision variables are declared with the key word `var`

```
var name1;
var name2;
```

Variables declared in that way are real-valued, we will later see how to declare vector and matrix-valued variables.

1.3 Parameters

The key word `param` allows to define the parameters of the problem. It is possible to assign a value to a parameter directly in the model file:

```
param n:=4;
```

Alternatively, a parameter can be declared in the model file:

```
param n;
```

one can then assign a value to it in the data file:

```
param n:= 4;
```

1.4 Cost function

The cost function is defined with the key words `maximize` and `minimize` :

```
minimize nameFunction : expression;
```

Example. Consider the minimization problem of the Rosenbrock function:

$$\min_{x,y\in\mathbb{R}^2} (1-x)^2 + 100(y-x^2)^2$$

The global solution is $x_0 = (1,1)$. The problem can be defined with AMPL as follows:

```
# Optimization variables
var x;
var y;

# Cost function
minimize J_Rosenbrock: 100*(y - x^2)^2 + (1-x)^2;

# Initialization of the value of the optimization variables (optional):
let x:=-1.;
let y:=1.1;
```

1.5 Constraintes

The key-word `s.t.` must be used to describe the constraints of the problem:

```
s.t. name1 : expression1;
s.t. name2 : expression2;
...
```

The constraint $x \geq 0$ can be programmed as follows:

```
s.t. g: x >= 0;
```

The constraint is called `g`.

2 Mathematical operations

2.1 Sets

A set can be declared with the key word `set`:

```
set A;
set time;
```

A simple way to define a set is to enumerate its elements in curly brackets:

```
set A= { 2.0, 3.0, 8.0, 6.0};
```

One can use the following syntax to define a sequence of uniformly spaced numbers:

```
start .. end by step
```

For example

```
set time = 1..10 by 2;
```

Le default step is 1.0:

```
set indices = 1..5;
```

Note the following set operations:

- `A inter B` : intersection
- `A union B` : union
- `A diff B` : difference
- `A symdyff B` : symmetric difference

2.2 Vectors and matrices

In order to declare a variable as vector or a matrix, the following syntax must be used:

```
var nameVariable{firstIndex..lastIndex};
```

The indices can be positive or negative. For example, the code

```
var x{0..5};
var y{-3..3};
```

declares a vector x in $\mathbb{R}^6$ and a vector in $\mathbb{R}^7$. One can access to the components of x with the commands:

```
x[0], x[1], x[2], x[3], x[4], x[5]
```

It is also possible to declare a vector x with indices in a set A (that needs to be defined before):

```
var x{A};
```

One can define a matrix `nameMatrix` as follows:

```
var nameMatrix{0..3, -3..3}
```

and access to the element $(0, -2)$ with `matrice[0,-2]`.

Vector-valued and matrix-valued paramaters are declared in the same fashion, except that the key word `param` must be used in place of `var`. We will see later on how to initialise such parameters.

2.3 Usual operations

Variables, parameters, and constants can be used in mathematical expressions. The standard arithmetic operations are executed according to the standard syntaxic rules used in most programming languages (C, pascal, Java, Python, ...):

```
*   /   +   -
```

For the power operator, the characters ^ and ** can be used. Most standard mathematical functions can be employed:

```
abs(x)     sin(x)    cos(x)         exp(x)
log(x)     log10(x)  min(x,y,...)   max(x,y,...)
round(x,n) round(x)  precision(x,n) trunc(x)
trunc(x,n) sqrt(x)   floor(x)
```

as well as some more specific ones:

```
Beta(a,b)   Cauchy()   Exponential()   Gamma(a)
Irand224()  Normal()   Poisson()       Uniform(m,n)
```

The number π can be obtained with:

```
param pi := 4*atan(1);
```

2.4 Mathematical expression involving sets

The syntax to be employed to formulate expressions such as "for all indices j in J" is the following:

```
{j in J}
```

For example, with `time` defined by

```
set time= 1..100 by 10
```

we can employ

```
{j in time}
```

to indicate that the index j runs over the set `time`.

This kind of syntax can be used in various situations.

1. *Sums and products.* Example:

   ```
     sum {j in 1..4} x[j]
   ```

   or

   ```
   prod {j in 1..4} x[j]
   ```

2. *Constraints.* Example:

   ```
     s.t. constraintesLin {j in 1..3}: sum{i in 1..3} (i+j)*x[j] <=2*j
   ```

defines the constraint

$$\sum_{i=1}^{3}(i+j)x_j \leq 2j, \quad \forall j = 1, 2, 3.$$

3. *Initialization of a vector, a matrix.* The vectorial parameter $a \in \mathbb{R}^4$, defined by $a_i = i$, can be initialized in that way:

   ```
   param a{i in 1..4} := i;
   ```

   Same thing for a matricial parameter $M \in \mathbb{R}^{4\times 3}$ defined by $M_{ij} = i + j$:

   ```
   param M{i in 1..4, j in 1..3} := i+j;
   ```

4. *Initialization of optimization variables.* Example:

   ```
   let {j in 1..3} x[j]:=j;
   ```

It is possible to define a subset of a given set with the help of a logical expression:

```
{index in set : logicalExpression}
```

For example:

```
sum{j in time : x[j]>0} x[j]
```

2.5 Logical expressions

The following key words can be used:

- `if expr1 then expre2 else expr 3` : conditional assignment; the operand of `if` must be a logical value (true/false), those of `then` and `else` are real-valued;
- `or` or `||` for the logical "or" ; the operands and the result are logical values
- `and` or `&&` for the logical "and"; the two operands and the result are logical values
- `exists` et `forall` : l'unique opérande et le résultat sont à valeur logiques;
- `not` or `!` : logical negation ; the operand and the result are logical values.

The classical comparison operators produce logical expressions:

```
<    >    <=    >=    <>    =
```

Remak. The operator `=` stands for a logical expression (and not an assignment).

Note that `in` also produces a logical expression. For example

```
i in 1:4 or x[i]<0
```

is true if i is between 1 and 4 (included) or if $x[i]$ is negative. The expression

```
exists {i in 1..10} x[i] > 30
```

is the logical value corresponding to

$$\exists i \in \{1, \ldots, 10\} \text{ such that } \; x_i > 30$$

Finally, the conditional assignment can be useful to program functions defined in a "piecewise" fashion. Consider for example:

$$\sum_{i=1}^{N} f(x_i)$$

where

$$f(x) = \begin{cases} 3 & \text{si } x \leq -2.0 \\ 3 + 2(x+2) & \text{sinon} \end{cases}$$

This can be done with the AMPL instruction

```
sum{i in 1..N} (if x[i]<=2.0 then 3.0 else 3.0+2.0*(x[i]+2.0) )
```

2.6 Calculated parameters and variables

Sometimes, the definition of the cost function or the definition of the constraints may require quite intricate arithmetic and logical expressions. Even though in principle, any syntaxicly correct expression can be used, it is sometimes better to use shorter expressions, for a readability purpose. Consider for example the cost function

$$J(a, b, c) = \sum_{i=1}^{N} (f(x_i) - y_i)^2$$

où

$$f(x) = ax^2 + bx + c.$$

A direct approach for programming J would be:

```
# Declaration of the parameters
param N >= 1; # number of points
param x {j in 1..N}= 0.5*j;
param y {1..N}; # corresponding values

# Declaration of the optimization variables
var a;
var b;
var c;

# Declaration of the cost
minimize J:sum{i in 1..N} (a*x[i]^2 +b*x[i]+c - y[i])^2;
```

Alternatively, one can use so-called "calculated" variables. Mathematically, they are quantities defined in function of other variables and/or parameters of the model. The syntax to be employed is essentially the same as the one used to assign values to parameters. The operator to be used is the logical operator `=`. If the expression in the definition depends only on parameters, one uses the key word `param`:

```
param nomParam=expression;
```

if the expression contains optimization variable, one uses the key word `var` :

```
    var nomFunc=expression;
```

In the previous example, the model can be written as follows:

```
# Declaration of the parameters
param N >= 1; # number of points
param x {j in 1..N}= 0.5*j;
param y {1..N}; # values

# Declaration of the optimization variables
var a;
var b;
var c;
var f{i in 1..N} = a*x[i]^2 +b*x[i]+c;

# Declaration of the cost
minimize J:sum{i in 1..N} (f[i] - y[i])^2;
```

Remak. The inequality character employed here must not be mistaken with the character `:=` for assigning values to parameters.

3 Working with data

The `data;` allows to initialize an instance of an optimization problem, that is, to assign numerical values to all declared parameters of the problem. The used operator is `:=`. It is of course mandatory to initialize these values before to try to solve the problem (with `solve`). If some declared parameter has no assigned value, an error message will be generated.

It is possible to declare default values for some parameters, whenever useful or meaningful, with the key word `default`, when declaring the parameter. For example:

```
param N>=0, default 10;
```

This defines a parameter N, which must be nonnegative, whose default value is 10, if no other value is assigned to it.

3.1 Initialization of indexed parameters

For vector-valued parameters, the assignment can be done by enumerating the pairs `indice valeur`. Suppose that the parameters N, x, and A have been declared in a model file as follows:

```
param N;
param x{1..N}>0;
param A{1..N, 1..N};
```

Note the imposed restriction on x: if a negative value is assigned to any of the component of x, an error message will appear. The parameters N and x can be initialized in that way (in the `data` file):

```
data;
param N:=3;
param x:=
         1 0.1
         2 0.2
         3 0.3;
```

or equivalently

```
data;
param N:=3;
param x:= 1 0.1  2 0.2  3 0.3;
```

since line breaks are ignored. It is also possible to assign the same value to all components of the vector, in the `data` file:

```
param x default 1.0;
```

For matrices, the assignment can be done by enumerating the triplets `line column value`. For example:

```
data;
param N:=3;
param A:=
         1 1 0.1
         1 2 0.0
         1 3 0.0
         2 1 0.0
         2 2 0.2
         2 3 0.0
         3 1 0.0
         3 2 0.0
         3 3 0.3;
```

The following syntax allows to define the matrix line by line:

```
data;
param N:=3;
param A:=
         [1, *]  1 0.1 2 0.0 3 0.0
         [2, *]  1 0.0 2 0.2 3 0.0
         [3, *]  1 0.0 2 0.0 3 0.3;
```

Another possible syntax:

```
data;
param N:=3;
param A :  1     2     3 :=
      1   0.1   0.0   0.0
      2   0.0   0.2   0.0
      3   0.0   0.0   0.3;
```

A default value can be employed in that way:

```
data;
param N:=3;
param A  default 0.0 :  1    2   3 :=
                   1   0.1   .   .
                   2   .    0.2  .
                   3   .      .  0.3;
```

The default value will be employed for the components marked with a dot.

3.2 Modifying partially the parameters and the variables of the problem

Sometimes, it is necessary to solve several times the same model, with minor modifications of the parameters and/or of the constraints. This can be done without re-writing a full `model` file and/or a full `data` file.

Commandes "reset", "update" The command `reset data` allows to re-initialize one or several parameters of a problem. It must be followed by a `data` section, where new values are assigned to the mentioned parameters. For example:

```
reset data A;

data;
 param A  default 0.0 :  1    2   3 :=
                   1   0.5   .   .
                   2   .    0.5  .
                   3   .      .  0.5;
```

In this example, the original value of A is first erased, then a new value is assigned to it. The command `update data` does essentially the same thing except that if new values are not assigned to the mentioned variables, then the old ones are preserved.

Command "let" The command `let` also allows to change the value of some parameters, similarly to `reset data` and `update data`. It should employed for a reduced number of parameters, since it does not allow to read new values in files. The syntax, for a scalar parameter, is simple:

```
 let T:=10;
```

For a vector-valued parameter, one can use for example:

```
 let {i in 1..5} x[i]:=0.5*i;
```

The command `let` can be used to initialize the optimization variables (before the resolution of the problem). They are otherwise initialized to zero. Depending on the model, they can speed up the numerical resolution or can avoid that the algorithm converges to a local minimizer.

Commands "fix", "unfix", "drop", "restore" The commands `reset data`, `update data` and `let` allow to modify partially the data of a model. It is also possible to modify the model itself, by freezing the value of some optimization variables or by ignoring some constraints.

The command `fix` allows to fix the value of some variables. Consider a model, involving some decision variable c. Its value can be fixed to 0 with the syntax:

```
fix c:= 0;
```

The command `unfix` allows to cancel the effect of `fix` :

```
unfix c;
```

The command `drop` (followed by the name of a constraint) allows to ignore the mentioned constraint.

The command `restore`, followed by the name of the constraint, allows to restore it.

3.3 Reading data in a non-formated file

The values of parameters can be extracted from a file. Consider for example a text file `data.txt` containing

```
3
0.1 0.2 0.3
```

The values of N and x (assuming they have been declared previously) can be initialized as follows:

```
read N, x[1], x[2], x[3] < data.txt;
```

Note the following syntax, particularly useful for large scale data:

```
read N, {i in 1..N} x[i]  < data.txt;
```

4 Exploiting the results

One may need to access to the optimization variables, constraints, or cost function, of the problem. This can be done with suffices. For optimization variables, they are:

- `.lb` : the lower bound of the variable
- `.ub` : the upper bound of the variable
- `.val` : the actual value.

The suffix `.dual` can be used to access the Lagrange multiplier associated with some given constraint. The command `display` allows to display the numerical value of any variable or constraint.

4.1 Writing results in a file

The syntax is

```
printf [index:] "format", listVariables
> fileName;
```

Example:

```
printf {i in 0..N-1}: "%10f %10f \n",i*h, v[i]
> vitesse.dat;
```

5 A few script commands

5.1 Commands "include" and "commands"

The commands `include` and `commands` allow to load in the console some lists of instructions, such as the commands `model` and `data`. The command

```
include fileName
```

is replaced by the content of the file `fileName` as it is. The command

```
commands fileName;
```

does the same thing, but the file `fileName` must contain an AMPL instruction (ended by `;`).

5.2 Loops commands and conditional instructions

The command

```
for {i in 1..3} commands fileName.txt;
```

loads the file `fileName` three times.

The command

```
repeat while a<=1.5 commands fileName.txt
```

loads the file `fileName` as long as $a \leq 1.5$.

6 Debugging your program

Here is a list of common mistakes.

1. The model and data files have not been saved.

2. A model is loaded while the previous has not been erased (with the `reset` command).

3. The character `;` is missing at one or several places.

4. Some optimization variables and some parameters have not been declared.

5. Some optimization variables have been declared without the key word `var`.

6. Some parameters have been declared without the key word `param`.

7. Some parameters have been initialized without the key word `param`.

8. Misuse of the characters `:` (for the cost function and constraints), `=` (equality constraints, calculated variables), and `:=` (initialization of variables).

9. For parameterized constraints, the index set must be put before `:`, for example:

   ```
   g {i in 1..n}: x[i] >= 0;
   ```

10. Misuse of parentheses and brackets. The characters `(` and `)` are used for prioritizing mathematical operations, the characters `{` and `}` are used for defining sets (in the declaration of the dimension of parameters and optimization variables, in parametrized constraints, in sums). The characters `[` and `]` are used to access to the component of a vector or a matrix.

11. Misuse of parentheses. The following commands are not understood by AMPL:

    ```
    sum ( {i in 1..n} x[i] );
    g: (x >= 0);
    ```

12. The character `*`, necessary for multiplications, is missing.

13. Use of a strict inequality constraint.

14. A vector-valued parameter (or optimization variable) has been declared as real-valued.

15. Undefined parameters. AMPL does not understand:

    ```
    param x{1..n};
    param n;
    ```

    The parameter n must be declared before x.