

AMPL

A Modeling Language for Mathematical Programming

Claudia D'Ambrosio

LIX, CNRS & École Polytechnique, France

MPRO — PMA

Algebraic modeling languages

- ▶ AMPL (and others): **algebraic modeling languages**

Algebraic modeling languages

- ▶ AMPL (and others): **algebraic modeling languages**
- ▶ **Syntax**: very similar to mathematical notation

Algebraic modeling languages

- ▶ AMPL (and others): **algebraic modeling languages**
- ▶ **Syntax**: very similar to mathematical notation
- ▶ **Model** mathematical optimization problems

Algebraic modeling languages

- ▶ AMPL (and others): **algebraic modeling languages**
- ▶ **Syntax**: very similar to mathematical notation
- ▶ **Model** mathematical optimization problems
- ▶ Develop **algorithms** based on mathematical optimization

Algebraic modeling languages

- ▶ AMPL (and others): **algebraic modeling languages**
- ▶ **Syntax**: very similar to mathematical notation
- ▶ **Model** mathematical optimization problems
- ▶ Develop **algorithms** based on mathematical optimization
- ▶ Linked to solvers for LP/MILP/MINLP problems

Algebraic modeling languages

- ▶ AMPL (and others): **algebraic modeling languages**
- ▶ **Syntax**: very similar to mathematical notation
- ▶ **Model** mathematical optimization problems
- ▶ Develop **algorithms** based on mathematical optimization
- ▶ Linked to solvers for LP/MILP/MINLP problems
- ▶ **AMPL book**:
www.ampl.com/BOOK/download.html

Algebraic modeling languages

- ▶ AMPL (and others): **algebraic modeling languages**
- ▶ **Syntax**: very similar to mathematical notation
- ▶ **Model** mathematical optimization problems
- ▶ Develop **algorithms** based on mathematical optimization
- ▶ Linked to solvers for LP/MILP/MINLP problems
- ▶ **AMPL book**:
`www.ampl.com/BOOK/download.html`
- ▶ **Quick-start guide**:
`https://www.lix.polytechnique.fr/~dambrosio/teaching/ampl-quick-start-guide\_dambrosio.pdf`

AMPL files

- ▶ Each problem instance is coded in AMPL using three files:

AMPL files

- ▶ Each problem instance is coded in AMPL using three files:
 - ▶ a **model file** (extension .mod): contains the mathematical formulation of the problem.

AMPL files

- ▶ Each problem instance is coded in AMPL using three files:
 - ▶ a **model file** (extension .mod): contains the mathematical formulation of the problem.
 - ▶ a **data file** (extension .dat): contains the numerical values of the problem parameters.

AMPL files

- ▶ Each problem instance is coded in AMPL using three files:
 - ▶ a **model file** (extension .mod): contains the mathematical formulation of the problem.
 - ▶ a **data file** (extension .dat): contains the numerical values of the problem parameters.
 - ▶ a **run file** (extension .run): specifies the solution algorithm (external and/or coded by the user in the AMPL language itself).

File .mod

- ▶ parameters, lines starting with the keyword `param`

File .mod

- ▶ parameters, lines starting with the keyword
param
- ▶ sets, lines starting with the keyword
set

File .mod

- ▶ parameters, lines starting with the keyword
`param`
- ▶ sets, lines starting with the keyword
`set`
- ▶ decision variables, lines starting with the keyword
`var`

File .mod

- ▶ parameters, lines starting with the keyword
`param`
- ▶ sets, lines starting with the keyword
`set`
- ▶ decision variables, lines starting with the keyword
`var`
- ▶ objective function(s), lines starting with the keyword
`minimize` or `maximize`

File .mod

- ▶ parameters, lines starting with the keyword
`param`
- ▶ sets, lines starting with the keyword
`set`
- ▶ decision variables, lines starting with the keyword
`var`
- ▶ objective function(s), lines starting with the keyword
`minimize` or `maximize`
- ▶ constraints, lines starting with the keyword
`subject to`

File .mod

```
param n > 0;
```

File .mod

```
param n > 0;
```

```
param w{1..n} > 0;
```

File .mod

```
param n > 0;
```

```
param w{1..n} > 0;
```

```
param n > 0;  
param m > 0;  
param a{1..n, 1..m};
```

File .mod

```
set VARS ordered := 1..n;
```

File .mod

```
set VARS ordered := 1..n;
```

```
param w{VARS} > 0;
```

File .mod

```
set VARS ordered := 1..n;
```

```
param w{VARS} > 0;
```

```
param w{N} > 0;  
param p{j in VARS} <= 10*w[j];
```

File .mod

Decision variables:

```
var x{j in 1..n} >= 0, <= 1, binary;
```


File .mod

Decision variables:

```
var x{j in 1..n} >= 0, <= 1, binary;
```

Objective function:

```
maximize total_profit:  
    sum{j in VARS} (a[j]+b[j]*x[j]+c[j]*x[j]**2+d[j]*x[j]**3);
```

File .mod

```
subject to capacity_constraint:  
    sum{j in VARS} w[j]*x[j] <= c;
```

File .mod

```
subject to capacity_constraint:  
    sum{j in VARS} w[j]*x[j] <= c;
```

```
subject to random_constraint{j in 2..n}:  
    w[j]*x[j] - w[j-1]*x[j-1] <= 1;
```

Continuous knapsack problem: File .mod

```
param N > 0;           # number of objects
set VARS ordered := {1..N};
param U {j in VARS} > 0;
param a {j in VARS} > 0;
param b {j in VARS} > 0;
param c {j in VARS} > 0;
param d {j in VARS} > 0;
param C > 0;  # knapsack capacity

var x {j in VARS} >= 0, <= U[j]; # variables

maximize Total_Profit: # objective function
    sum {j in VARS} (a[j]+b[j]*x[j]+c[j]*x[j]**2+d
        [j]*x[j]**3);

subject to KP_constraint: # constraint
    sum{j in VARS} x[j] <= C;
```

Continuous knapsack problem: File .dat

```
# Author: Claudia D'AMBROSIO  
# Date: 20200106  
# nlkp.dat
```

```
param N := 10;  
param C := 546.000000;
```

```
param: a :=  
1      0.172274  
2      0.134944  
3      0.101030  
4      0.163588  
5      0.152350  
6      0.196601  
7      0.181208  
8      0.126588  
9      0.184087  
10     0.187434  
;
```

```
param: b :=  
1      78.770199  
2      77.468892  
3      93.324757  
4      96.180080  
5      55.137398  
6      40.101851  
7      36.007819  
8      5.317250  
9      9.964929  
10     60.265707  
;
```

```
param: c :=  
1      3.062328  
2      43.280130  
3      52.983122  
4      62.101010  
5      58.531125  
6      47.574366  
7      53.101406  
8      6.902601  
9      16.985577  
10     62.576610  
;  
param: d :=  
1      -81.876165  
2      -56.455229  
3      -56.428945  
4      -43.813029  
5      -28.246895  
6      -81.108142  
7      -37.839956  
8      -1.138258  
9      -80.968161  
10     -11.536015  
;
```

Continuous knapsack problem: File .dat

```
param: U :=  
1      100.000000  
2      100.000000  
3      100.000000  
4      100.000000  
5      100.000000  
6      100.000000  
7      100.000000  
8      100.000000  
9      100.000000  
10     100.000000  
;
```

Continuous knapsack problem: File .run

```
# Author: Claudia D'Ambrosio
# Date: 20190121
# nlkp.run

reset;
reset data;

model nlkp.mod;

data "../dat/nlkp.dat";

option solver "baron";
option baron_options "prfreq=100 outlev=1";

solve > nlkp.out;
```

Other commands

```
reset;  
reset data;
```


Other commands

```
reset;  
reset data;
```

```
option solver gurobi;  
option gurobi_options "outlev 1";  
solve;
```

Other commands

```
model myMILPmodel.mod;  
data myInstance.dat;  
option solver cplex;  
option relax_integrality 1; # relaxing the  
    integrality requirements on all the decision  
    variables  
solve;  
option relax_integrality 0; # restoring the  
    integrality requirements on all the decision  
    variables  
solve;
```

Other commands

```
display n, c;  
display N;  
display w, p;
```

Other commands

```
display n, c;  
display N;  
display w, p;
```

```
n = 7  
c = 19  
  
set N := 1 2 3 4 5 6 7;  
  
:      w      p      :=  
1      11      10  
2       6       3  
3       6       4  
4       5       5  
5       5       6  
6       4       7  
7       1       2  
;
```

Other commands

```
display x;  
display cost;
```

Other commands

```
display x;  
display cost;
```

```
x [*] :=  
1  0.363636  
2  0  
3  0  
4  1  
5  1  
6  1  
7  1  
;  
  
cost = 23.6364
```

Other commands

```
display capacity_constraint;
```

Other commands

```
display capacity_constraint;
```

```
capacity_constraint = 0.909091
```


Other commands

```
expand capacity_constraint;
```

Other commands

```
expand capacity_constraint;
```

```
subject to capacity_constraint:  
    11*x[1] + 6*x[2] + 6*x[3] + 5*x[4] + 5*x[5] + 4*x[6] + x[7] <= 19;
```

Other commands

```
printf "param n := %d;\n", n;  
printf "\n";
```

```
# param c  
printf "param c :";  
for {j in VARS} {  
    printf "\t%d", j;  
}  
printf "\t:=\n";  
for {i in VARS} {  
    printf "\t%d", i;  
    for {j in VARS} {  
        printf "\t%d", c[i,j];  
    }  
    printf "\n";  
}  
printf ";\n\n";
```

Other commands

```
for {j in 1..n} {  
  ...  
}
```

Other commands

```
for {j in 1..n} {  
  ...  
}
```

```
repeat {  
  . . .  
}  
until x[n] > 0;
```

Non linear knapsack problem

File nlkp.out obtained:

```
BARON 18.11.12 (2018.11.12): prfreq=100 outlev=1
=====
...
=====
Preprocessing found feasible solution with value 1.60010400000
Doing local search
Preprocessing found feasible solution with value 1093.96317285
Solving bounding LP
Starting multi-start local search
Done with local search
=====
Iteration Open nodes Time (s) Lower bound Upper bound
1 1 0.15 1093.96 11340.2
100 16 1.62 1093.96 1121.50
200 30 2.04 1093.96 1094.05
300 35 2.19 1093.96 1094.00
400 31 2.38 1093.96 1093.97
500 28 2.55 1093.96 1093.97
600 13 2.67 1093.96 1093.97
631 0 2.70 1093.96 1093.96
Cleaning up
*** Normal completion ***
Wall clock time: 3.00
Total CPU time used: 2.70
Total no. of BaR iterations: 631
Best solution found at node: -1
Max. no. of nodes in memory: 37
All done
=====
BARON 18.11.12 (2018.11.12): 631 iterations, optimal within tolerances.
Objective 1093.963173
```

Profit = 1093.96

References

- ▶ Modeling languages like **ampl**: ampl.com
or **gams**: www.gams.com
- ▶ Open source solvers like coin-or **ipopt/bonmin/couenne**:
www.coin-or.org/projects/
and **scip**: scip.zib.de
- ▶ **NEOS Server**, State-of-the-Art Solvers for Numerical
Optimization: www.neos-server.org/neos/
- ▶ **MINLP benchmarks** like minlp.org
www.gamsworld.org/minlp/minlplib2/html/
wiki.mcs.anl.gov/leyffer/index.php/MacMINLP