Mathematics for Decisions

Introduction to AMPL and GMPL

Romeo Rizzi, Alice Raffaele, and Matteo Zavatteri

University of Verona

romeo.rizzi@univr.it, alice.raffaele@univr.it, matteo.zavatteri@univr.it

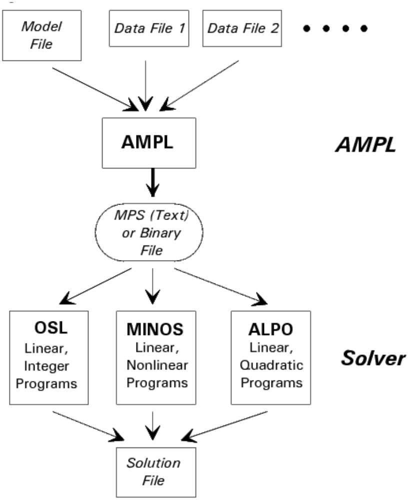
October 21, 2021

Overview

Introduction to

- **A***lgebraic* **M***odeling* **P***rogramming* **L***anguage* (**AMPL**) assists in formulating the optimization problem of interest within a mathematical formalism. It allows to:
  - use computers to design, experiment, refine and apply mathematical programming models without having to code;
  - solve instances without having to designing algorithms nor crack hard mathematical problems.
- **AMPL purpose**: translating an optimization problem into a form understandable by a generic solver.

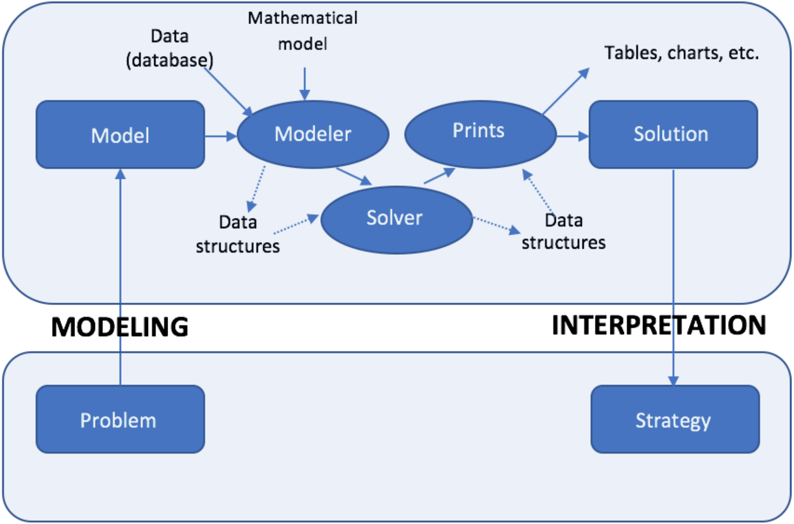
The strategic place of AMPL



What is a solver?

- Software that finds a solution to a mathematical problem given as input.
- Together with the solution, it also returns some information related to the mathematical model describing the problem.
- Solvers can be useful to optimize different classes of problems:
  - Linear Programming problems (LP);
  - Mixed-Integer Linear Programming (MILP);
  - Mixed-Integer Quadratic Programming (MIQP);
  - Constraint Programming (CP);
  - Quadratic Programming (QP);
  - Quadratically Constrained Programming (QCP);
  - Mixed-Integer Quadratically Constrained Programming (MIQCP);
  - Nonlinear Programming (NLP).

From problem to strategy



Advantages of AMPL

- AMPL was designed and implemented in 1985 by Robert Fourer, David M. Gay and Brian Kernighan.
- Modeling languages can help to make mathematical programming more economical and **reliable**.
- **Similarity** with algebraic modeling languages (based on the use of traditional mathematical notation).
- **Applicability** to several classes of problems.
- **Support** for different solvers.
- **Easiness** of syntax.
- **High generality** of set and indexing expressions.

Example – The Diet Problem

A diet prescribes that, to fulfill baseline daily necessities, specific amounts of *calories*, *protein* and *calcium* have to be introduced in the body. This diet is composed of five basic foods (*bread*, *milk*, *eggs*, *meat*, *sweets*), while baseline amounts are 2000 calories, 50 g. of protein and 700 mg of calcium.
The following values are derived from dietary tables, indicating how many calories (in Kcal.) protein (in g.), calcium (in mg.) a portion of each food provides:

	Bread	**Milk**	**Eggs**	**Meat**	**Sweets**
Calories	150	120	160	230	450
Protein	4	8	15	14	4
Calcium	2	285	54	80	22

The following table represents the cost per portion and the number of daily portions available for each food:

	Bread	**Milk**	**Eggs**	**Meat**	**Sweets**
Cost	3	2	3	19	15
Max portions	4	7	2	3	2

Determine a minimum-cost diet that satisfies every prescription.

Formulation

- Let's start by assigning a variable to represent the number of portions of each food: x_1, x_2, x_3, x_4, x_5
- The objective function corresponds to the sum of the costs of every food type: $min\ 3x_1 + 2x_2 + 3x_3 + 19x_4 + 15x_5$
- Baseline daily necessities:
  - Calories: $150x_1 + 120x_2 + 160x_3 + 230x_4 + 450x_5 \geq 2000$
  - Protein: $4x_1 + 8x_2 + 15x_3 + 14x_4 + 4x_5 \geq 50$
  - Calcium: $2x_1 + 285x_2 + 54x_3 + 80x_4 + 22x_5 \geq 700$
- Maximum number of portions:
  - Bread: $0 \leq x_1 \leq 4$
  - Milk: $0 \leq x_2 \leq 7$
  - Eggs: $0 \leq x_3 \leq 2$
  - Meat: $0 \leq x_4 \leq 3$
  - Sweets: $0 \leq x_5 \leq 2$

What you must know to start

- Every AMPL project is composed of at least two files:
  - the **model**, where you describe the problem; the extension of this file is **.mod**.
  - the **data**, where you specify the problem instance, i.e. the values to be used for parameters, sets and variables defined in the model; the extension of this file is **.dat**.
- **Note:** it is a smart move to always keep the model and the data separate. By doing so, the model can be used again with several instances of data, without even having to modify even a single character!
- The **AMPL process** in six simple steps:
  1. Launch AMPL
  2. Load the model (.mod)
  3. Load the data (.dat)
  4. Specify the solver to use
  5. Solve the model
  6. Display solution and results

Something to know about AMPL syntax

- AMPL is **case-sensitive**
- Every instruction ends with **;**
- Before every definition, there must be a declaration
- Maximum row length is 255 characters (characters in excess simply ignored)
- Main keywords: **set**, **param**, **var**, **minimize/maximize**, **subject to / s.t.**

The model file: diet.mod

diet.mod

```
# SETS
set M; # set of nutrients
set N; # set of foods

# PARAMETERS
param amount_portion{M,N}; # amounts of nutrients for a portion of a given food
param cost{N}; # cost of a portion of a given food
param max_portions{N}; # maximum daily number of portions for a given food
param minimum_amount{M}; # minimum amounts of nutrients to get daily

# VARIABLES
var x{N}>=0; # number of portions for a given food
# OBJECTIVE FUNCTION
minimize total_cost: sum{i in N} cost[i] * x[i];
# CONSTRAINTS
subject to daily_necessities{i in M}: sum{j in N} amount_portion[i, j] * x[j] >=
minimum_amount[i];
subject to max_number{i in N}: x[i] <= max_portions[i];
```

- As you can see, there are no numbers specified in this file: this will let you keep the model separate from the instances of the problem → **Flexibility and modularity**: to solve a new instance of the problem, you will not have to re-define the model every time; only the data file will change.
- **Declaration in the .mod file, definition in the .dat file**

Sets

```
# SETS
set M; # set of nutrients
set N; # set of foods
```

- Sets define the **domain** of the problem
- A set is a group of elements of the same type; it can be:
  - unordered: set a;
  - numerical: set a := 1 .. 10 **by** 2;
  - ordered: set a **ordered**;
  - circular: set a **circular**;
- Operations on sets: **union**, **diff**, **inter**, **symdiff**, **card**, **within**
- If a set A is ordered, there are additional operations: **first(A)**, **last(A)**, **next(i,A,j)**, **prev(i,A,j)**, **next(i,A)**, **prev(i,A)**, **ord0(i,A)**, **member(i,A)**
- You can use the keyword **cross** to define a set as the Cartesian product of other two sets: set a := X cross Y;

Parameters

```
# PARAMETERS
param amount_portion{M,N}; # amounts of nutrients for a portion of a given food
param cost{N}; # cost of a portion of a given food
param max_portions{N}; # maximum daily number of portions of a given food
param minimum_amount{M}; # minimum amounts of nutrients to get daily
```

- Parameters are the **values** of a problem
- A parameter can be:
  - a single scalar: param b;
  - a collection of values indexed by a set: param a {j in P};
  - **param cost**{**N**} indicates that there is a value associated to each item in the set N
- Every piece of information given in the problem is represented by a parameter.
- Parameters and other numerical values are the building blocks of the expressions for the model's objective function and constraints.

Variables

```
# VARIABLES
var x{N}>=0; # number of portions of a given food
```

- Variables are quantities that describe the **solution** and they are determined by the solver
- Syntactically, their declarations are the same as for parameters. They can be:
  - simple: var x;
  - a collection of unknown quantities indexed by a set: var x{N};
  - **var x**$\{$**N**$\}\geq$**0** indicates that there is a variable for every item in the set N and each one must be nonnegative

The objective function and the constraints

```
# OBJECTIVE FUNCTION
minimize total_cost: sum{i in N} cost[i] * x[i];
```

- The function to be optimized
- According to the requirements, the goal may be to **minimize** or **maximize** a certain expression

```
# CONSTRAINTS
subject to daily_necessities{i in M}: sum{j in N} amount_portion[i, j] * x[j] >=
minimum_amount[i];
subject to max_number{i in N}: x[i] <= max_portions[i];
```

- The objective function is followed by the list of constraints to be satisfied during the resolution of the problem
- Constraints are introduced by the keywords **subject to**
- The objective function and constraints must have names.

The data file

diet.dat

```
# SETS
set M := CALORIES PROTEIN  CALCIUM;
set N := BREAD MILK EGGS MEAT SWEETS;
# PARAMETERS
param amount_portion : BREAD MILK EGGS MEAT SWEETS :=
CALORIES 150 120 160 230 450
PROTEIN  4 8 15 14 4
CALCIUM 2 285 54 80 22;
param cost :=
BREAD 3
MILK 2
EGGS 3
MEAT 19
SWEETS 15;
param max_portions :=
BREAD 4
MILK 7
EGGS 2
MEAT 3
SWEETS 2;
param minimum_amount :=
CALORIES 2000
PROTEIN  50
CALCIUM 700;
```

Definition of sets and parameters

The symbol used for definition is **:=**

diet.dat

```
# SETS
set M := CALORIES PROTEIN  CALCIUM;
set N := BREAD MILK EGGS MEAT SWEETS;
# PARAMETERS
param amount_portion : BREAD MILK EGGS MEAT SWEETS :=
CALORIES 150 120 160 230 450
PROTEIN  4 8 15 14 4
CALCIUM 2 285 54 80 22;
param cost :=
BREAD 3
MILK 2
EGGS 3
MEAT 19
SWEETS 15;
param max_portions :=
BREAD 4
MILK 7
EGGS 2
MEAT 3
SWEETS 2;
param minimum_amount :=
CALORIES 2000
PROTEIN  50
CALCIUM 700;
```

param amount_portion{M,N}

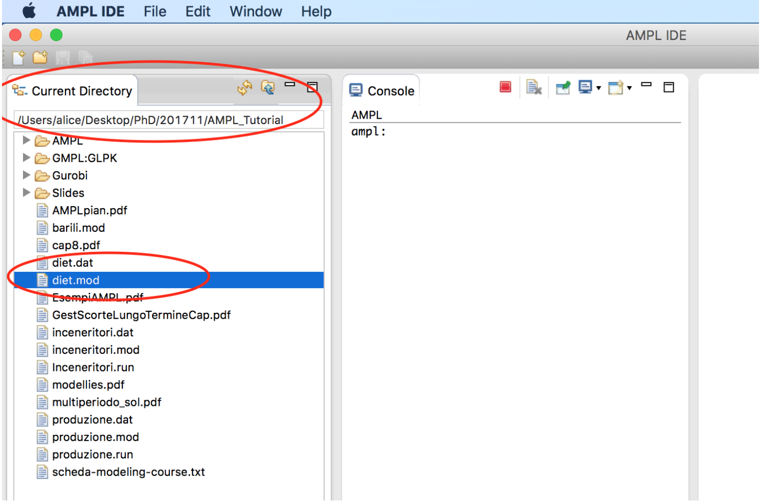
param cost{N}

param max_portions{N}

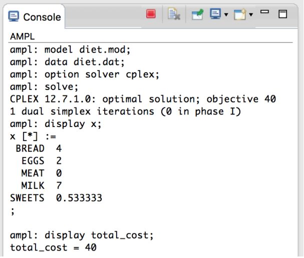
param minimum_amount{M}

How to solve the problem

1. Launch the AMPL IDE and go to the directory where you have the files:



2. Load the model: **model diet.mod;**
3. Load the data: **data diet.dat;**
4. Choose the solver: **option solver cplex;**
5. Solve the problem: **solve;**
6. Look at the results: **display x;**



```
Console
AMPL
ampl: model diet.mod;
ampl: data diet.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.7.1.0: optimal solution; objective 40
1 dual simplex iterations (0 in phase I)
ampl: display x;
x [*] :=
 BREAD  4
  EGGS  2
  MEAT  0
  MILK  7
SWEETS  0.533333
;

ampl: display total_cost;
total_cost = 40
```

The optimal solution is given by 4 portions of bread, 2 of eggs, 0 of meat, 7 of milk and 0.533 of sweets; the objective function is equal to 40.

AMPL-Gurobi

- Another way: solving the problem using AMPL and calling Gurobi as solver.
- We can use again the .mod and .dat files because they do not need edits.
- The only thing to be changed is the solver and its options:
  - **option solver gurobi;**
  - **option gurobi_options 'presolve 2';** to tell Gurobi to be "aggressive" (default value -1; 0 disabled; 1 "conservative").
  - **option gurobi_options $gurobi_options 'mipgap 0.01';** to set the MIP gap (the solver will terminate when the gap between the lower and upper objective bound is less than MIPGap times the absolute value of the upper bound).

- Here is the code:

Console

AMPL

```
ampl: model diet.mod;
ampl: data diet.dat;
ampl: option solver gurobi;
ampl: option gurobi_options 'presolve 2';
ampl: option gurobi_options $gurobi_options 'mipgap 0.01';
ampl: solve;
Gurobi 7.5.0: presolve 2
mipgap 0.01
Gurobi 7.5.0: optimal solution; objective 40
1 simplex iterations
ampl: display x;
x [*] :=
 BREAD  4
  EGGS  2
  MEAT  0
  MILK  7
SWEETS  0.533333
;

ampl: display total_cost;
total_cost = 40
```

Introduction to GLPK

- GLPK (i.e., GNU Linear Programming Kit) is a software package developed in ANSI C by Andrew Makhorin (Department for Applied Informatics, Moscow Aviation Institute) to solve large-scale LP and MIP problems.
- GLPK is a library included in the GNU Project.
- The package contains several components:
  - For LP problems: revised simplex method and primal-dual interior point method;
  - For IP problems: Branch-and-bound method and cut routines;
  - Others: translator for GNU MathProg modelling language; API; stand-alone LP/MIP solver **glpsol**

glpsol and GMPL

- Since GLPK is a library, it needs a client software that calls GLPK APIs → The default client is the glpsol solver (GNU Linear Program Solver).
- glpsol takes a **GMPL** model and the related data as inputs and it outputs a solution.
- **GMPL** (i.e., GNU Math Programming Language) is a modelling language intended to describe linear mathematical programming models:
  - Like in AMPL, the model is described through **model objects** such as sets, parameters, variables, constraints and objectives;
  - The model description is composed of two parts:
    - the model section
    - the data section
  - GMPL syntax is basically the same as AMPL

The Diet Problem with GMPL and glpsol

- You can use just one .mod file containing both model and data section, divided by the keyword **data;** or you can divide, as before, model and instance in two separate files, as you like.

diet-gmpl.mod

```
/* THE DIET PROBLEM */
set M;
/* nutrients */

set N;
/* foods */

param minimum_amount{M};
/* minimum amounts of nutrients to get daily */

param amount_portion{M,N};
/* amounts of nutrients for a portion of a given food */

param cost{N};
/* cost of a portion of a given food */

param max_portions{N};
/* maximum daily number of portions of a given food */

var x{n in N} >= 0;
/* number of portions of a given food */

s.t. daily_necessities{i in M}: sum{j in N} amount_portion[i, j] * x[j] >=
minimum_amount[i];
s.t. max_number{i in N}: x[i] <= max_portions[i];
/* constraints */

minimize total_cost: sum{i in N} cost[i] * x[i];
/* total cost (euro) */
```

- The last instruction is **end;**

diet-gmpl.mod

```
data;

set M := CALORIES PROTEIN CALCIUM;
set N := BREAD MILK EGGS MEAT SWEETS;

param amount_portion : BREAD MILK EGGS MEAT SWEETS :=
CALORIES 150 120 160 230 450
PROTEIN 4 8 15 14 4
CALCIUM 2 285 54 80 22;

param cost :=
BREAD 3
MILK 2
EGGS 3
MEAT 19
SWEETS 15;

param max_portions :=
BREAD 4
MILK 7
EGGS 2
MEAT 3
SWEETS 2;

param minimum_amount :=
CALORIES 2000
PROTEIN 50
CALCIUM 700;

end;
```

How to solve the problem with glpsol

1. Open a new terminal window, go to the folder where you have your **diet-gmpl.mod** file
2. Type **glpsol –model diet-gmpl.mod –output diet-gmpl.sol** to solve the problem and write the results on a .sol file

AMPL_Tutorial — -bash — 99×31

```
Last login: Fri Nov 10 17:52:31 on ttys000
MBP-di-Alice:~ alice$ cd Desktop/PhD/201711/AMPL_Tutorial/
MBP-di-Alice:AMPL_Tutorial alice$ glpsol --model diet-gmpl.mod --output diet-gmpl.sol
GLPSOL: GLPK LP/MIP Solver, v4.63
Parameter(s) specified in the command line:
 --model diet-gmpl.mod --output diet-gmpl.sol
Reading model section from diet-gmpl.mod...
Reading data section from diet-gmpl.mod...
59 lines were read
Generating daily_necessities...
Generating max_number...
Generating total_cost...
Model has been successfully generated
GLPK Simplex Optimizer, v4.63
9 rows, 5 columns, 25 non-zeros
Preprocessing...
3 rows, 5 columns, 15 non-zeros
Scaling...
 A: min|aij| =  2.000e+00  max|aij| =  4.500e+02  ratio =  2.250e+02
GM: min|aij| =  2.737e-01  max|aij| =  3.653e+00  ratio =  1.335e+01
EQ: min|aij| =  7.493e-02  max|aij| =  1.000e+00  ratio =  1.335e+01
Constructing initial basis...
Size of triangular part is 3
      0: obj =   0.000000000e+00 inf =   4.355e+02 (3)
      5: obj =   8.967677733e+01 inf =   0.000e+00 (0)
*     9: obj =   4.000000000e+01 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (126378 bytes)
Writing basic solution to 'diet-gmpl.sol'...
MBP-di-Alice:AMPL_Tutorial alice$
```

The output file

- Obviously, as before, the optimal solution found is 40.

diet-gmpl.sol

```
Problem:    diet
Rows:       9
Columns:    5
Non-zeros:  25
Status:     OPTIMAL
Objective:  total_cost = 40 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 daily_necessities[CALORIES]
                    NL          2000          2000                  0.0333333
     2 daily_necessities[PROTEIN ]
                    B        104.133            50
     3 daily_necessities[CALCIUM]
                    B        2122.73           700
     4 max_number[BREAD]
                    NU             4                           4            -2
     5 max_number[MILK]
                    NU             7                           7            -2
     6 max_number[EGGS]
                    NU             2                           2      -2.33333
     7 max_number[MEAT]
                    B              0                           3
     8 max_number[SWEETS]
                    B       0.533333                           2
     9 total_cost     B             40

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 x[BREAD]     B              4             0
     2 x[MILK]      B              7             0
     3 x[EGGS]      B              2             0
     4 x[MEAT]      NL             0             0                    11.3333
     5 x[SWEETS]    B       0.533333             0
```

Conclusion

- We talked about some possible ways of solving problems exploiting solvers and mathematical programming languages.
- We learned the basics of AMPL and GMPL through an example.
- And now, what's next? More problems to solve!

References

AMPL
http://ampl.com

AMPL: A Modeling Language for Mathematical Programming
R. Fourer, D.M. Gay, B. Kernighan
http://ampl.com/resources/the-ampl-book/

Short guide about AMPL for Operations Research classes (Italian)
M. Bruglieri, R. Cordone, L. Liberti,
http://profs.sci.univr.it/∼rrizzi/classes/modeling-intro/AMPL/manualeampl.pdf

Operations Research – AMPL (Italian)
D. Macaluso, M. Pennacchi, G. Stangoni
https://www.slideshare.net/macdario/ricerca-operativa-ampl

Slides about AMPL (Italian)
F. Rinaldi
http://www.math.unipd.it/∼rinaldi/teaching/IntroM.pdf and
http://www.math.unipd.it/∼rinaldi/teaching/Sintassi_Esempi.pdf

GLPK
https://en.wikibooks.org/wiki/GLPK

GLPK/GMPL
https://en.wikibooks.org/wiki/GLPK/GMPL_(MathProg)

Modeling Language GNU MathProg
A. Makhorin
https://www.cs.unb.ca/~bremner/docs/glpk/gmpl.pdf

Operations Research with GNU Linear Programming Kit
T. Sottinen
http://lipas.uwasa.fi/~tsottine/lecture_notes/or.pdf

The GNU Linear Programming Kit (GLPK) : Resources, Tutorials etc.
S. Pokutta
https://spokutta.wordpress.com/the-gnu-linear-programming-kit-glpk/

To exercise online with AMPL:
http://ampl.com/cgi-bin/ampl/amplcgi

To exercise online with several solvers and languages:
https://neos-server.org/neos/solvers/