



INFORMS Annual Meeting 2010

Overview of and Update on ASL, the AMPL/Solver Interface Library

David M. Gay

AMPL Optimization LLC

`dmg@ampl.com`



AMPL: a language for
mathematical programming problems:

$$\begin{aligned} & \text{minimize } f(x) \\ & \text{s.t. } \ell \leq c(x) \leq u, \end{aligned}$$

with $x \in \mathbb{R}^n$ and $c : \mathbb{R}^n \rightarrow \mathbb{R}^m$ given algebraically and some x_i discrete.



Sample session

```
ampl: model dieti.mod;
ampl: data diet2a.dat;
ampl: option solver gurobi;
ampl: solve;
Gurobi 4.0.0: optimal solution; objective 119.3
28 simplex iterations
21 branch-and-cut nodes
absmipgap = 2.84e-14, relmipgap = 2.38e-16
ampl: display Buy;
Buy [*] :=
BEEF    9
    CHK    2
```



FISH 2

HAM 8

MCH 10

MTL 10

SPG 7

TUR 2

;



What did “solve;” do?

- Write temp. “.nl” file representing the problem;
- invoke `$solver`, passing current *environment* — option names and values;
- read solution from “.sol” file.

Solver uses AMPL/Solver interface Library (ASL) to

- obtain problem details;
- do nonlinear evaluations (if needed);
- return solution — write “.sol” file.



Problem details in .nl file

A .nl file conveys

- Problem dimensions: e.g., numbers of
 - continuous and discrete variables
 - linear and nonlinear constraints and objectives
- initial primal and dual variable values
- initial basis
- *expression graphs* for nonlinearities.



Readers of .nl files

ASL offers several `.nl` readers that prepare for solving various kinds of problems, including

- linear
- quadratic
- nonlinear functions and gradients
- nonlinear functions, gradients, and Hessians:
 - Hessian-vector products
 - explicit (sparse) Hessians
 - exploiting (or ignoring) partial separability.

Suffixes



Suffixes convey auxiliary information on variables, constraints, objectives, problems.

Most commonly, `.sstatus` (for “solver status”) supplies initial basis values:

```
AMPL: option sstatus_table;
option sstatus_table '\
0      none      no status assigned\
1      bas       basic\
2      sup       superbasic\
3      low       nonbasic <= (normally =) lower bound\
4      upp       nonbasic >= (normally =) upper bound\
5      equ       nonbasic at equal lower and upper bounds\
6      btw       nonbasic between bounds\
';
```

.astatus



Suffix `.astatus` (for “AMPL status”) gives AMPL’s view of constraints and variables:

```
AMPL: option astatus_table;
option astatus_table '\
0      in      normal state (in problem)\
1      drop    removed by drop command\
2      pre     eliminated by presolve\
3      fix     fixed by fix command\
4      sub     defined variable, substituted out\
5      unused  not used in current problem\
6      log     logical constraint in current problem\
7      rand    random variable in current problem';
```

.status



Suffix `.status` is `.sstatus` if `.astatus` is “in” and is `.astatus` otherwise:

```
ampl: option solver minos; solve;
MINOS 5.51: ignoring integrality of 8 variables
MINOS 5.51: optimal solution found.
3 iterations, objective 118.0594032
ampl: display Buy, Buy.astatus, Buy.sstatus, Buy.status;
:      Buy      Buy.astatus Buy.sstatus Buy.status      :=
BEEF   5.36061   in          bas          bas
CHK     2        in          low         low
FISH    2        in          low         low
HAM    10        in          upp         upp
MCH    10        in          upp         upp
MTL    10        in          upp         upp
SPG     9.30605  in          bas          bas
```



Suffixes for SOS sets

Special Ordered Set (SOS) structure can be supplied explicitly with `.sosno` and `.ref`.

Most commonly, SOS2 sets are used to express piecewise-linear terms.

AMPL renders nonconvex

<<bkpoint-list ; slope-list>> var

terms with extra constraints and variables adorned with `.sos` and `.sosref` suffixes.

Interfaces to solvers that handle SOS sets can call the ASL's `suf_sos(...)` to remove the added constraints and variables that imply SOS conditions.



Returned suffixes

Solver interfaces can return new or updated suffix values via the `.sol` file.

Many solvers return updated `.sstatus` values.

For unbounded problems, some return `.unbdd` (for a feasible ray). For unbounded dual problems, some return `.dunbdd`.

For MIPs, some return `.absmipgap` or `.relmipgap` and `.bestbound` or `.bestnode` on objectives or problems.



Example of .dunbdd

```
ampl: reset; model diet.mod; data diet2.dat;
ampl: option solver cplex; solve;
CPLEX 12.2.0.0: No LP presolve or aggregator reductions.
infeasible problem.
2 dual simplex iterations (0 in phase I)
constraint.dunbdd returned
suffix dunbdd OUT;
ampl: display _conname, _con.dunbdd;
:      _conname      _con.dunbdd      :=
1  "Diet['A']"      0
2  "Diet['B1']"      0
3  "Diet['B2']"      0.226598
4  "Diet['C']"      0
5  "Diet['NA']"     -0.00255754
6  "Diet['CAL']"    0
```



Forthcoming ASL enhancements

- Support for stochastic programming, including new `.stage` suffix.
- More constraint programming facilities: e.g., variables in subscripts.
- New routines for bounds from slopes.
- New suffix for multi-level program structure.
- Updated documentation.



AMPL extension: random variables

Debated whether to add “*random parameters*” or “*random variables*”.

Internally, they act like nonlinear variables, and “random variable” is a conventional term, so **random** in a **var** declaration introduces a random variable:

```
var x random;
```

Declarations may specify a value (with **=** or **default**):

```
var y random = Uniform01();
```

or subsequently be assigned:

```
let x := Normal(0,2);
```



Dependent random variables

Dependent random variables may only be declared in

`var ... =` and `var ... default` declarations:

```
var x random;
```

```
var y = x + 1;
```

Random variables may appear as variables in constraint and objective declarations:

```
s.t. Demand: sum {i in A} build[i] >= y;
```



Seeing random variable values

Printing commands see random variables as strings expressing distributions...

```
var x random = Normal01();  
var y = x + Uniform(3,5);  
display x, y;
```

gives

```
x = 'Normal01()'  
y = 'Uniform(3, 5) + x'
```



Sampling random variables

```
display {1..5} (Sample(x), Sample(y));
```

gives

```
:      Sample(x)  Sample(y)      :=  
1      1.51898    3.62453  
2      -3.65725   2.50557  
3      -0.412257  5.4215  
4      0.726723   2.89672  
5      -0.606458  3.776  
;
```



Conventional uses of random functions

Without **random**, we get ordinary sampling:

```
var x := Uniform(0,10);  
minimize zot: (x - Normal01())^2;  
display x;  
expand zot;
```

gives

```
x = 6.09209
```

```
minimize zot:  
      (x - 1.51898)^2;
```



New builtin functions

New “builtin” functions for solvers to interpret:

- $\text{Expected}(\xi)$
- $\text{Moment}(\xi, n), n = 1, 2, 3, \dots$
- $\text{Percentile}(\xi, p), 0 \leq p \leq 100$
- $\text{Sample}(\xi)$
- $\text{StdDev}(\xi)$
- $\text{Variance}(\xi)$
- $\text{Probability}(\textit{logical condition})$



What happens when?

Stages indicate what happens when.

SMPS convention: **Stage = event** followed by **decision**, perhaps with first stage “event” known.

A variable is split into separate copies, one for each realization of its stage (but not of subsequent stages).

For more on SMPS, see

<http://myweb.dal.ca/gassmann/smps2.htm>



New “system suffix” `.stage`

New reserved suffix `.stage`, e.g.,

```
set A; set Stages;
```

```
var x {A, s in Stages} suffix stage s;
```

or

```
var x {A, s in Stages};
```

```
...
```

```
let {a in A, s in Stages}  
    x[a,s].stage := s;
```



Example: stochastic diet problem

Buy in two stages; constrain budget in first stage, suffer random price changes in second stage.

What to buy in first stage?

Old: `var Buy {j in FOOD} integer >= f_min[j],
 <= f_max[j];`

New: `set T = 1 .. 2; # times (stages)
var Buy {FOOD, t in T} integer >= 0
 suffix stage t;
s.t. FoodBounds {j in FOOD}: f_min[j]
 <= sum{t in T} Buy[j,t] <= f_max[j];`



Stochastic diet problem (cont'd)

Old:

```
minimize Total_Cost:  
    sum {j in FOOD} cost[j] * Buy[j];
```

New:

```
var CostAdj {FOOD} random;  
minimize Total_Cost:  
    sum {j in FOOD} cost[j] * Buy[j,1]  
+ Expected(sum {j in FOOD}  
    cost[j]*CostAdj[j]*Buy[j,2]);
```



Stochastic diet problem (cont'd)

Old: $\text{sum } \{j \text{ in FOOD}\} \text{ amt}[i,j] * \text{Buy}[j]$

New: $\text{sum } \{j \text{ in FOOD}, t \text{ in T}\}$
 $\text{amt}[i,j] * \text{Buy}[j,t]$
param init_budget;
s.t. Init_Bud: $\text{sum } \{j \text{ in FOOD}\} \text{Buy}[j,1]$
 $\leq \text{init_budget};$
...
let{j in FOOD} CostAdj[j]
 $:= \text{Uniform}(.7, 1.3);$



“Constant” distributions

Assign numerical value to random variable \implies
simplified problem (for debugging and model
development).

Example:

```
let{j in FOOD} CostAdj[j]  
    := Sample(Uniform(.7, 1.3));
```

With imported function $\text{Expected}(x) = x$, this works
with conventional solvers.



Some things work now

Things that work include

- Most details of random-variable handling
 - Declarations
 - Assignments of distributions
 - Assignments of constants
 - Printing and sampling (in AMPL sessions)
 - Determining what the solver will see as linear
- Writing `.nl` files with random distributions
- Suffix `“.stage”` and functions of distributions.



Nonanticipativity

Nonanticipativity is implicit in stating problems (compact form). The `.nl` file has sparsity structure for all constraints and objectives, indicating which variables appear (and giving linear coefficients). This includes random variables. Stage structure is in `.stage` suffixes. Solvers can split variables if desired.



Work in progress

Updates to solver-interface library (for sampling), sample drivers not yet finished. Plans include

- Routines to pose *deterministic equivalents*, e.g., with stratified sampling such as Latin hypercube. Options `randoptions` and `($solver)_randoptions` would control sampling and discretization.
- Program to write `.nl` file for deterministic equivalent.
- Program to write SMPS format.
- Solver drivers, e.g., for Gassmann's MSLiP.



Bound computations

Forthcoming additions to ASL (AMPL/Solver interface Library) include routines for bound computations. See paper “Bounds from Slopes” in <http://www.sandia.gov/~dmgay/bounds10.pdf>

Possible application: importance sampling. Sample next where support measure times variation bound is largest.



For more details (dmg@ampl.com)

<http://www.ampl.com> points to

- The AMPL book
- examples (models, data)
- descriptions of new stuff (in book 2nd ed., not 1st)
- downloads
 - student binaries; trial-license form
 - solver interface library source
 - “standard” table handler & source
 - papers and reports



Aux. slide: dieti.mod

```
set NUTR;   set FOOD;
```

```
param cost {FOOD} > 0;
```

```
param f_min {FOOD} >= 0;
```

```
param f_max {j in FOOD} >= f_min[j];
```

```
param n_min {NUTR} >= 0;
```

```
param n_max {i in NUTR} >= n_min[i];
```

```
param amt {NUTR,FOOD} >= 0;
```

```
var Buy {j in FOOD} integer >= f_min[j], <= f_max[j];
```

```
minimize total_cost:  sum {j in FOOD} cost[j] * Buy[j];
```

```
subject to diet {i in NUTR}:
```

```
    n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```



Aux. slide: diet2a.dat

```
param:    cost    f_min    f_max    :=  
  BEEF    3.19     2         10  
  CHK     2.59     2         10  
  FISH    2.29     2         10  
  HAM     2.89     2         10  
  MCH     1.89     2         10  
  MTL     1.99     2         10  
  SPG     1.99     2         10  
  TUR     2.49     2         10    ;
```

```
param:    n_min    n_max    :=  
  A        700     20000  
  C        700     20000  
  B1       700     20000
```



Aux. slide: diet2a.dat *cont'd*

```
B2      700    20000
NA       0     50000
CAL  16000    24000 ;
```

param amt (tr):

| | A | C | B1 | B2 | NA | CAL | := |
|------|----|----|----|----|------|-----|----|
| BEEF | 60 | 20 | 10 | 15 | 938 | 295 | |
| CHK | 8 | 0 | 20 | 20 | 2180 | 770 | |
| FISH | 8 | 10 | 15 | 10 | 945 | 440 | |
| HAM | 40 | 40 | 35 | 10 | 278 | 430 | |
| MCH | 15 | 35 | 15 | 15 | 1182 | 315 | |
| MTL | 70 | 30 | 15 | 15 | 896 | 400 | |
| SPG | 25 | 50 | 25 | 15 | 1329 | 370 | |
| TUR | 60 | 20 | 15 | 10 | 1397 | 450 | ; |