The AMPL Modeling Language — an Aid to Formulating and Solving Optimization Problems

David M. Gay
AMPL Optimization, Inc.
dmg@ampl.com
http://www.ampl.com
AMPL summary

AMPL: a language for mathematical programming problems, e.g.,

\[ \text{minimize } f(x) \]
\[ \text{s.t. } \ell \leq c(x) \leq u, \]

with \( x \in \mathbb{R}^n \) and \( c : \mathbb{R}^n \to \mathbb{R}^m \) given algebraically and some \( x_i \) discrete.
AMPL goals

- Easy transcription from math (*avoid mistakes*)
- Explicit indexing (*no hidden magic*)
- Declare before use (*one-pass reading*)
- Separate model, data, commands (*orthogonality*)
- Separate solvers (*open solver interface*)
- Update entities as needed (*lazy eval.*)
- Builtin math. prog. stuff (*presolve, red. costs, ...*)
- Aim for large scale nonlinear (*sparsity, generality*)
AMPL history

• Early 1980’s: little languages at Bell Labs
• 1983: Fourer’s TOMS paper, ML’s vs MG’s
• 1984: Karmarkar hoopla begins
• 1985–6: Fourer sabbatical at Bell Labs
• 1987: AMPL CSTR (Fourer, Gay, Kernighan)
• Late 1980’s: reimplementation; extensions begin
• 1993: 1st edition of AMPL book
• 2001: collapse of Bell Labs
• 2003: 2nd edition of AMPL book; dmg → Sandia
• 2010: dmg → AMPL Optimization
Simple declarations, commands

ampl:  param  p;
ampl:  param  q = p + 10;
ampl:  data; param  p := 2.5;
ampl  data:  display  p,  q;
    p = 2.5
    q = 12.5

ampl:  let  p := 17;  display  p,  q;
    p = 17
    q = 27
Simple sets

ampl: set A; set B;

ampl: set C = p .. q;

ampl: display A;

Error executing "display" command:
no data for set A

ampl: data; set A := a b c; set B := c d;

ampl data: display A, B, C;

set A := a b c;

set B := c d;

set C := 17 18 19 20 21 22 23 24 25 26 27;
Simple set operations

ampl: display A intersect B, A union B;
set A inter B := c;

set A union B := a b c d;

ampl: display A diff B, A symdiff B;
set A diff B := a b;

set A symdiff B := a b d;
Iterated expressions, declarations

ampl: print sum {i in 1..4} i;
10

ampl: print prod {i in 1..4} i;
24

ampl: param fac{ i in 1..9 }
ampl? = if i == 1 then 1 else i*fac[i-1];

ampl: print max{i in 1..9}
ampl? abs(fac[i] - prod{j in 2..i} j);
0
More iterated commands

```ampl
display fac, \{i in 1..9\} prod\{j in 2..i\} j;
:   fac   prod\{j in 2 .. i\} j :=
  1   1   1
  2   2   2
  3   6   6
  4  24  24
  5 120 120
  6 720 720
  7 5040 5040
  8 40320 40320
  9 362880 362880;
```

Example model: diet.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} >= 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];
Example data file: diet.dat (beginning)

```AMPL
data;
set NUTR := A B1 B2 C ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;

param:  cost  f_min  f_max :=
       BEEF  3.19   0    100
       CHK   2.59   0    100
       FISH  2.29   0    100
       HAM   2.89   0    100
       MCH   1.89   0    100
       MTL   1.99   0    100
       SPG   1.99   0    100
       TUR   2.49   0    100 ;
```

Example data file continued: diet.dat

param: n_min n_max :=
  A   700 10000
  C   700 10000
  B1  700 10000
  B2  700 10000 ;

param amt (tr):
  A   C   B1  B2 :=
  BEEF 60  20  10  15
  CHK   8   0  20  20
  FISH  8  10  15  10
  HAM   40  40  35  10
  MCH   15  35  15  15
  MTL   70  30  15  15
  SPG   25  50  25  15
  TUR   60  20  15  10 ;
Example session

ampl: model diet.mod; data diet.dat;
ampl: solve;
MINOS 5.51: optimal solution found.
6 iterations, objective 88.2
ampl: display Buy;
Buy [*] :=
BEEF 0
CHK 0
FISH 0
HAM 0
MCH 46.6667
MTL 1.57618e-15
SPG 8.42982e-15
TUR 0
;

Example session continued:  
Imposing integrality

```ampl
ampl: redeclare var Buy{j in FOOD}
ampl? integer >= f_min[j] <= f_max[j];
ampl: solve;
MINOS 5.51: ignoring integrality of 8 variables
MINOS 5.51: optimal solution found.
4 iterations, objective 88.2
ampl: option solver cplex;
ampl: solve;
CPLEX 12.6.0.0: optimal integer solution; objective 88.44
4 MIP simplex iterations
0 branch-and-bound nodes
```
Example session continued: result of imposing integrality

```AMPL
ampl: display Buy;
Buy [*] :=
  BEEF  0
  CHK   2
  FISH  0
  HAM  0
  MCH  43
  MTL  1
  SPG  0
  TUR  0
;
```
Example: modified data (diet2.dat):

data;
set NUTR := A B1 B2 C NA CAL ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;

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 ;
Example: more diet2.dat:

```plaintext
param: n_min n_max :=
    A  700  20000
    C  700  20000
    B1 700  20000
    B2 700  20000
    NA  0   40000
    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;
```
Using the new data file

ampl: reset data;
ampl: data diet2.dat;
ampl: solve;
Cplex 12.6.0.0: integer infeasible.
1 MIP simplex iterations
0 branch-and-bound nodes
No basis.
ampl: display Buy;
Buy [*] :=
BEEF 0
  CHK 0
FISH 0
  HAM 0
  MCH 0
  MTL 0
  SPG 0
  TUR 0
;
Analyzing the infeasibility

```ampl
ampl: option solver minos; solve;
MINOS 5.51: ignoring integrality of 8 variables
MINOS 5.51: infeasible problem.
9 iterations
ampl: display diet.lb, diet.body, diet.ub, diet.slack;
: diet.lb diet.body diet.ub diet.slack :=
A  700  1993.09  20000  1293.09
B1 700  841.091  20000  141.091
B2 700  601.091  20000 -98.9086
C  700  1272.55  20000  572.547
CAL 16000  17222.9  24000  1222.92
NA  0   40000    40000   7.27596e-12
;```

19
Fixing the infeasibility

```ampl
ampl: print n_max['NA'];
40000
ampl: let n_max['NA'] := 50000;
ampl: solve;
MINOS 5.51: ignoring integrality of 8 variables
MINOS 5.51: optimal solution found.
5 iterations, objective 118.0594032
ampl: param MinosSoln{FOOD};
ampl: let{i in FOOD} MinosSoln[i] := Buy[i];
ampl: option solver cplex; solve;
CPLEX 12.6.0.0: optimal integer solution; objective 119.3
9 MIP simplex iterations
0 branch-and-bound nodes
absmipgap = 2.84217e-14, relmipgap = 2.38237e-16
```
Looking at the solutions

```AMPL
ampl: display Buy, MinosSoln;

:      Buy   MinosSoln :=
BEEF   9     5.36061
CHK    2     2
FISH   2     2
HAM    8     10
MCH    10    10
MTL    10    10
SPG    7     9.30605
TUR    2     2

;```

A nonlinear example, \texttt{pgon.mod}

\begin{verbatim}
# Maximum area for unit-diameter polygon of N sides.
# The following model started as a GAMS model by Francisco J. Prieto.

param N integer > 0;
set I = 1..N;

param pi = 4*atan(1.);

var rho{i in I} <= 1, >= 0  # polar radius (distance to fixed vertex)
  := 4*i*(N + 1 - i)/(N+1)**2;

var theta{i in I} >= 0  # polar angle (measured from fixed dir.)
  := pi*i/N;

s.t. cd{i in I, j in i+1 .. N}:
    rho[i]**2 + rho[j]**2
    - 2*rho[i]*rho[j]*cos(theta[j] - theta[i]) <= 1;
\end{verbatim}
A nonlinear example (con’d), \texttt{pgon.mod}

s.t. \texttt{ac\{i in 2..N\}}:

\begin{align*}
\theta[i] \geq \theta[i-1];
\end{align*}

s.t. \texttt{fix\_theta}: \theta[N] = \pi;

s.t. \texttt{fix\_rho}: \rho[N] = 0;

\texttt{maximize area:}

\begin{align*}
.5\sum\{i in 2..N\} \rho[i]\rho[i-1]\sin(\theta[i] - \theta[i-1]);
\end{align*}
Solution of pgon.mod, $N = 6$
Solution does *not* lie on a circle
 Commands to generate *pic* input

Commands in file “pgwrite” used via

```
include pgwrite
```

or

```
commands pgwrite;
```

```
printf ".PS 6i\nline from 0,0" >pgon60.ms;
printf \{i in I\} " \n\ntto %g,%g",
    rho[i]*cos(theta[i]),
    rho[i]*sin(theta[i]) >>pgon60.ms;
printf ";\n.PE\n" >>pgon60.ms;
close pgon.ms;
```
Slices can turn $O(n^2)$ into $O(n)$

Example where changing

\[
\text{s.t. } \forall a \in A: \sum_{(i,j) \in S: i = a} x[i,j] = 1;
\]

into

\[
\text{s.t. } \forall a \in A: \sum_{(a,j) \in S} x[a,j] = 1;
\]

reduced problem instantiation from 4 hours to a minute.
Iterated union via setof

## Portion of a data-fusion model

```ampl
## Portion of a data-fusion model

set A dimen 2;  # (observation, classifier) pairs
param E{A};  # weighted predictions
set I = setof {(i,j) in A} i;  # observations
set J = setof {(i,j) in A} j;  # classifiers
param y{I} in {1,-1};  # y[i] = 1 ==> "yes", -1 ==> "no"
var x{J} >= 0;  # weights on classifiers
set B = {(i,j) in A: y[i]*E[i,j] < 0};  # mis-classified pairs

minimize errsq: sum{i in I} (sum{(i,j) in B} y[i]*E[i,j]*x[j])^2;
  s.t. convex: sum{i in J} x[i] = 1;

param Majority = floor(card(J)/2) + 1;
  # bad training cases with simple voting
set Badvote = {i in I: card{(i,j) in B} >= Majority};
```

28
Iterated union, defined var

# Mesh untangling constraints, right-hand rule at each vertex.
set D3 = 1 .. 3;               # three spatial coordinates
param Npoints;
set P default 0 .. Npoints-1;  # set of points
var v{P,D3};                   # vertices

param Nfixed default 0;
set Fixed within P default card(P) - Nfixed .. card(P) - 1;
set Edges = union 
{ (a,b,c,d,e,f,g,h) in Hexes } {
  (a,b), (a,d), (a,e),
  (b,c), (b,a), (b,f),
  (c,d), (c,b), (c,g),
  (d,a), (d,c), (d,h),
  (e,h), (e,f), (e,a),
  (f,e), (f,g), (f,b),
  (g,f), (g,h), (g,c),
  (h,g), (h,e), (h,d) };

var dx{(a,b) in Edges, j in D3} = v[b,j] - v[a,j];
Iterated union, defined var (con’d)

set HexCorners = \{(a,b,c,d,e,f,g,h) \in \text{Hexes},
  (A,B,D,E) \in \{(a,b,d,e),
   (b,c,a,f),
   (c,d,b,g),
   (d,a,c,h),
   (e,h,f,a),
   (f,e,g,b),
   (g,f,h,c),
   (h,g,e,d)\}\};

var \text{volsign}\{(a,b,c,d,e,f,g,h,A,B,D,E) \in \text{HexCorners}\} = 
  dx[A,B,1] \times (dx[A,D,2] \times dx[A,E,3] - dx[A,D,3] \times dx[A,E,2]) + 
  dx[A,B,2] \times (dx[A,D,3] \times dx[A,E,1] - dx[A,D,1] \times dx[A,E,3]) + 
  dx[A,B,3] \times (dx[A,D,1] \times dx[A,E,2] - dx[A,D,2] \times dx[A,E,1]);

var mn; maximize maximin: mn;

s.t. mn_bound\{(a,b,c,d,e,f,g,h,A,B,D,E) \in \text{HexCorners}\}:
  mn \leq \text{volsign}[a,b,c,d,e,f,g,h,A,B,D,E];

s.t. Volsign\{(a,b,c,d,e,f,g,h,A,B,D,E) \in \text{HexCorners}\}:
  \text{volsign}[a,b,c,d,e,f,g,h,A,B,D,E] \geq 0;
AMPL flexibility goals

- Allow interactive, “batch”, and “GUI” use
- Allow extensions via shared libs
  - imported functions
  - table handlers
- Promote interaction with host OS
  - shell command
  - pipe functions
  - options \(\rightarrow\) environment
  - file redirections, \texttt{remove} command
Options

Environment: (name, $value) pairs.
Initial environment from invocation

\textit{with defaults for names not there.}

Changed by \texttt{option} command.

AMPL interprets some option settings (e.g., $\texttt{solver}$).
Invoked processes (solvers, shell) see modified env.

Convention: \texttt{option solver_options} affects \texttt{solver}.
Option examples

option cplex_options 'advance=2 lpdisplay=1 \ 
prestats = 1 \ 
primalopt’
" aggregate=1 aggfill=20";

option solver knitro,

knitro_options "maxit=30";
Interactive option examples

```ampl
option cplex_options 'advance=2 lpdisplay=1 \\
    prestats = 1 \\
    primalopt'
" aggregate=1 aggfill=20";
```

```ampl
option cplex_options;
option cplex_options 'advance=2 lpdisplay=1 \\
    primalopt aggregate=1 aggfill=20';
```

```ampl
option;
option AMPLFUNC ampltabl.dll;
option Cautions 1;
option MD_precision 0;
option OPTIONS_IN ' ';
option OPTIONS_INOUT ' ';
option OPTIONS_OUT ' ';
option PATH ':/home/dmg/h/bin:/usr/local/bin:/usr/bin:/bin';
option SHELL ' /bin/bash';
...  ```
Interactive option examples (con’d)

```ampl
ampl: option solver cplex, re*es 1, send_** 0;
option reset_initial_guesses 1;
option send_statuses 0;
ampl: solve;
CPLEX 12.6.0.0: advance=2
lpdisplay=1
primalopt
aggregate=1
aggfill=20
LP Presolve eliminated 6 rows and 9 columns.
All rows and columns eliminated.
CPLEX 12.6.0.0: optimal integer solution; objective 119.3
9 MIP simplex iterations
0 branch-and-bound nodes
absmipgap = 2.84217e-14, relmipgap = 2.38237e-16
```
Redirections

Can redirect most output to files:

```ampl
ampl: option >foo1;
ampl: solve >solve.out;
```

CPLEX 12.6.0.0: optimal integer solution; objective 119.3
9 MIP simplex iterations
0 branch-and-bound nodes
absmipgap = 2.84217e-14, relmipgap = 2.38237e-16

```ampl
close solve.out; shell 'cat solve.out';
```
String expressions

( string expression ) can replace 'literal string'
almost everywhere.

AMPL:

```
param mypath symbolic;
let mypath := 'c:/full/path/to/somewhere/';
option solver (mypath & 'minos' & 5 + .4);
option solver;
option solver 'c:/full/path/to/somewhere/minos5.4';
print $solver;
c:/full/path/to/somewhere/minos5.4
```
Interaction with Solvers

AMPL writes .nl file with

• problem statistics
• coefficients for linear expressions
• expression graphs for nonlinear expressions
• initial guesses (primal and dual)
• suffixes (user declared; basis)

AMPL options modify the solver’s environment. Solver writes .sol file with solution and returned suffixes (e.g., basis).
Problem Transformations

AMPL’s presolve derives and propagates bounds with directed roundings and may fix variables, remove constraints (e.g., inequalities that are never tight), resolve complementarities, turn nonlinear expressions into linear (after fixing relevant variables), simplify convex piecewise-linear expressions, and convert nonconvex piecewise-linear expressions to equivalent systems of integer variables and SOS-2 constraints.
Implementation Techniques

- `lex` and `yacc`; `cfront`
- expression graph manipulations
- hashing for symbols, sets, common expressions
- lifting invariants out of loops
- on-the-fly expression rewrites
- error handling with registered clean-up routines and `longjump`
- reference counting where appropriate
- sparse matrix methods
Hoped-for Enhancements

- programming interfaces
- AMPL functions for modeling and solver call-backs
- ordered sets of tuples
- tuples as atoms
- more efficient instantiation of related instances (e.g., when adding cutting planes)
- variables in subscripts for constraint programming
- with one objective, exploiting duality in presolve
- extend AMPL/solver-interface library (ASL) for stochastic programming
More Wish-List Items

• facilities for SDP and multi-level optimization
• conversations with solvers: just supply instance differences when the next problem instance is not too different from the current one
• units
• other data types (rational, complex)
• parallel ASL evaluations
• constructs for parallelism in AMPL
AMPL facilities not treated in these slides

- drop, restore, fix, unfix; named problems
- looping and flow-of-control commands
- suffixes
- tables
- column-generation syntax (e.g., node and arc)
- complementarity constraints
- subscripted sets versus tuples
- imported functions (with OUT-args)
- constraint-programming extensions (partly done)
For more details (info@ampl.org or dmg@ampl.com)

http://www.ampl.com points to

• The AMPL book (PDF files now freely available)
• examples (models, data)
• descriptions of new stuff, e.g., new IDE
• Try AMPL! and NEOS; trial licenses
• downloads
  o student binaries and requests for course licenses
  o solver interface library source
  o “standard” table handler & source
  o papers and reports
Selected References


Selected References (continued)


Selected References (continued)


These Slides