New AMPL Interfaces for Enhanced Optimization Model Development and Deployment

Robert Fourer

AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

INFORMS Conference on
Business Analytics & Operations Research

Boston, 30 March – 1 April 2014
Track 11, Monday 1:50-2:40, *Software Tutorials*

Outline

Deploying models

- Scripting
  - Internal modeling/programming language
- **AMPL API** (Application Programming Interfaces)
  - External programming language support
  - General-purplse languages: C++, Java, .NET, Python
  - Analytics languages: MATLAB, R

Developing models

- More natural formulations
  - Logical conditions
  - Quadratic constraints
- **AMPL IDE** (Integrated Development Environment)
  - Unified editor & command processor
  - Built on the Eclipse platform

Introductory Example

Multicommodity transportation . . .

- Products available at factories
- Products needed at stores
- Plan shipments at lowest cost

. . . with practical restrictions

- Cost has fixed and variable parts
- Shipments cannot be too small
- Factories cannot serve too many stores

Multicommodity Transportation

Given

- O Set of origins (factories)
- D Set of destinations (stores)
- P Set of products

and

- a_{ip} Amount available, for each $i \in O$ and $p \in P$
- b_{jp} Amount required, for each $j \in D$ and $p \in P$
- l_{ij} Limit on total shipments, for each $i \in O$ and $j \in D$
- c_{ijp} Shipping cost per unit, for each $i \in O$, $j \in D$, $p \in P$
- d_{ij} Fixed cost for shipping any amount from $i \in O$ to $j \in D$
- s Minimum total size of any shipment
- n Maximum number of destinations served by any origin

Multicommodity Transportation

Mathematical Formulation

Determine

X_{ijp} Amount of each $p \in P$ to be shipped from $i \in O$ to $j \in D$

Y_{ij} 1 if any product is shipped from $i \in O$ to $j \in D$
0 otherwise

to minimize

$\sum_{i \in O} \sum_{j \in D} \sum_{p \in P} c_{ijp} X_{ijp} + \sum_{i \in O} \sum_{j \in D} d_{ij} Y_{ij}$

Total variable cost plus total fixed cost

Mathematical Formulation

Subject to

$\sum_{j \in D} X_{ijp} \leq a_{ip}$ for all $i \in O$, $p \in P$

Total shipments of product p out of origin i must not exceed availability

$\sum_{i \in O} X_{ijp} = b_{jp}$ for all $j \in D$, $p \in P$

Total shipments of product p into destination j must satisfy requirements

Mathematical Formulation

Subject to

$\sum_{p \in P} X_{ijp} \leq l_{ij} Y_{ij}$ for all $i \in O$, $j \in D$

When there are shipments from origin i to destination j, the total may not exceed the limit, and Y_{ij} must be 1

$\sum_{p \in P} X_{ijp} \geq s Y_{ij}$ for all $i \in O$, $j \in D$

When there are shipments from origin i to destination j, the total amount of shipments must be at least s

$\sum_{j \in D} Y_{ij} \leq n$ for all $i \in O$

Number of destinations served by origin i must be as most n

Multicommodity Transportation

AMPL Formulation

Symbolic data

```
set ORIG;   # origins
set DEST;   # destinations
set PROD;   # products

param supply {ORIG,PROD} >= 0;  # availabilities at origins
param demand {DEST,PROD} >= 0;  # requirements at destinations
param limit {ORIG,DEST} >= 0;   # capacities of links

param vcost {ORIG,DEST,PROD} >= 0; # variable shipment cost
param fcost {ORIG,DEST} > 0;       # fixed usage cost

param minload >= 0;             # minimum shipment size
param maxserve integer > 0;     # maximum destinations served
```

AMPL Formulation

Symbolic model: variables and objective

```
var Trans {ORIG,DEST,PROD} >= 0;   # actual units to be shipped
var Use {ORIG, DEST} binary;       # 1 if link used, 0 otherwise

minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD} vcost[i,j,p] * Trans[i,j,p]
 + sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];
```

$$\sum_{i\in O}\sum_{j\in D}\sum_{p\in P} c_{ijp}\, X_{ijp} + \sum_{i\in O}\sum_{j\in D} d_{ij}\, Y_{ij}$$

Multicommodity Transportation

AMPL Formulation

Symbolic model: constraint

```
subject to Supply {i in ORIG, p in PROD}:
   sum {j in DEST} Trans[i,j,p] <= supply[i,p];
```

$$\sum_{j \in D} X_{ijp} \leq a_{ip}, \text{ for all } i \in O, p \in P$$

AMPL Formulation

Symbolic model: constraints

```
subject to Supply {i in ORIG, p in PROD}:
   sum {j in DEST} Trans[i,j,p] <= supply[i,p];

subject to Demand {j in DEST, p in PROD}:
   sum {i in ORIG} Trans[i,j,p] = demand[j,p];

subject to Multi {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];

subject to Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] >= minload * Use[i,j];

subject to Max_Serve {i in ORIG}:
   sum {j in DEST} Use[i,j] <= maxserve;
```

AMPL Formulation

Explicit data independent of symbolic model

```
set ORIG := GARY CLEV PITT ;
set DEST := FRA DET LAN WIN STL FRE LAF ;
set PROD := bands coils plate ;

param supply (tr):  GARY   CLEV   PITT :=
            bands    400    700    800
            coils    800   1600   1800
            plate    200    300    300 ;

param demand (tr):
           FRA   DET   LAN   WIN   STL   FRE   LAF :=
   bands   300   300   100    75   650   225   250
   coils   500   750   400   250   950   850   500
   plate   100   100     0    50   200   100   250 ;

param limit default 625 ;

param minload := 375 ;
param maxserve := 5 ;
```

Multicommodity Transportation

AMPL Formulation

Explicit data (continued)

```
param vcost :=

 [*,*,bands]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
        GARY    30   10    8   10   11   71    6
        CLEV    22    7   10    7   21   82   13
        PITT    19   11   12   10   25   83   15

 [*,*,coils]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
        GARY    39   14   11   14   16   82    8
        CLEV    27    9   12    9   26   95   17
        PITT    24   14   17   13   28   99   20

 [*,*,plate]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
        GARY    41   15   12   16   17   86    8
        CLEV    29    9   13    9   28   99   18
        PITT    26   14   17   13   31  104   20 ;

param fcost:   FRA  DET  LAN  WIN  STL  FRE  LAF :=
        GARY  3000 1200 1200 1200 2500 3500 2500
        CLEV  2000 1000 1500 1200 2500 3000 2200
        PITT  2000 1200 1500 1500 2500 3500 2200 ;
```

AMPL Solution

Model + data = problem instance to be solved

```
ampl: model multmipG.mod;
ampl: data multmipG.dat;

ampl: option solver gurobi;

ampl: solve;

Gurobi 5.6.0: optimal solution; objective 235625
293 simplex iterations
28 branch-and-cut nodes

ampl: display Use;

Use [*,*]

:    DET FRA FRE LAF LAN STL WIN  :=
CLEV   1   1   1   0   1   1   0
GARY   0   0   0   1   0   1   1
PITT   1   1   1   1   0   1   0
;
```

Multicommodity Transportation

AMPL Solution

Solver choice independent of model and data

```
ampl: model multmipG.mod;
ampl: data multmipG.dat;

ampl: option solver cplex;

ampl: solve;

CPLEX 12.6.0.0: optimal integer solution; objective 235625
136 MIP simplex iterations
0 branch-and-bound nodes

ampl: display Use;

Use [*,*]

:     DET FRA FRE LAF LAN STL WIN  :=
CLEV   1   1   1   0   1   1   0
GARY   0   0   0   1   0   1   1
PITT   1   1   1   1   0   1   0
;
```

Multicommodity Transportation

AMPL Solution

Examine results

```
ampl: display {i in ORIG, j in DEST}
ampl?    sum {p in PROD} Trans[i,j,p] / limit[i,j];

:       DET     FRA     FRE     LAF     LAN     STL     WIN     :=
CLEV    1       0.6     0.88    0       0.8     0.88    0
GARY    0       0       0       0.64    0       1       0.6
PITT    0.84    0.84    1       0.96    0       1       0
;

ampl: display Max_Serve.body;

CLEV   5
GARY   3
PITT   5
;

ampl: display TotalCost,
ampl?    sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];

TotalCost = 235625
sum {i in ORIG, j in DEST} fcost[i,j]*Use[i,j] = 27600
```

Multicommodity Transportation

AMPL "Sparse" Network

Indexed over sets of pairs and triples

```
set ORIG;   # origins
set DEST;   # destinations
set PROD;   # products

set SHIP within {ORIG,DEST,PROD};

            # (i,j,p) in SHIP ==> can ship p from i to j

set LINK = setof {(i,j,p) in SHIP} (i,j);

            # (i,j) in LINK ==> can ship some products from i to j

...........

var Trans {SHIP} >= 0;   # actual units to be shipped

var Use {LINK} binary;   # 1 if link used, 0 otherwise

minimize Total_Cost:

   sum {(i,j,p) in SHIP} vcost[i,j,p] * Trans[i,j,p]

 + sum {(i,j) in LINK} fcost[i,j] * Use[i,j];
```

Scripting

Incorporate programming concepts . . .

- Loops of various kinds
- If-then and If-then-else conditionals
- Assignments

. . . using modeling language syntax

- Same algebraic expressions
- Same set indexing expressions
- All commands from interactive mode

1: Parametric Analysis

Try different limits on destinations served

- Reduce parameter `maxserve` and re-solve
  - until there is no feasible solution
- Display results
  - parameter value
  - numbers of destinations actually served

Try different supplies of plate at Gary

- Increase parameter `supply['GARY','plate']` and re-solve
  - until dual is zero (constraint is slack)
- Record results
  - distinct dual values
  - corresponding objective values

. . . display results at the end

Parametric Analysis *on limits*

Script to test sensitivity to serve limit

```
model multmipG.mod;
data multmipG.dat;

option solver gurobi;

for {m in 7..1 by -1} {
   let maxserve := m;
   solve;
   if solve_result = 'infeasible' then break;
   display maxserve, Max_Serve.body;
}
```

```
subject to Max_Serve {i in ORIG}:
   sum {j in DEST} Use[i,j] <= maxserve;
```

Parametric Analysis *on limits*

Run showing sensitivity to serve limit

```
ampl: include multmipServ.run;

Gurobi 5.6.0: optimal solution; objective 233150
maxserve = 7
CLEV 5   GARY 3   PITT 6

Gurobi 5.6.0: optimal solution; objective 233150
maxserve = 6
CLEV 5   GARY 3   PITT 6

Gurobi 5.6.0: optimal solution; objective 235625
maxserve = 5
CLEV 5   GARY 3   PITT 5

Gurobi 5.6.0: infeasible
```

Parametric Analysis *on supplies*

Script to test sensitivity to plate supply at GARY

```
set SUPPLY default {};
param sup_obj {SUPPLY};
param sup_dual {SUPPLY};

let supply['GARY','plate'] := 200;
param sup_step = 10;
param previous_dual default -Infinity;

repeat while previous_dual < 0 {

  solve;

  if Supply['GARY','plate'].dual > previous_dual then {
    let SUPPLY := SUPPLY union {supply['GARY','plate']};
    let sup_obj[supply['GARY','plate']] := Total_Cost;
    let sup_dual[supply['GARY','plate']] := Supply['GARY','plate'].dual;
    let previous_dual := Supply['GARY','plate'].dual;
    }

  let supply['GARY','plate'] := supply['GARY','plate'] + supply_step;
  }
```

Parametric Analysis *on supplies*

Run showing sensitivity to plate supply at GARY

```
ampl: include multmipSupply.run;

ampl: display sup_obj, sup_dual;
:      sup_obj   sup_dual    :=
200     223504    -13
380     221171    -11.52
460     220260    -10.52
510     219754     -8.52
560     219413      0
;
```

Parametric: Observations

Results of solve can be tested

- Check whether problem is no longer feasible
  - `if solve_result = 'infeasible' then break;`

Parameters are true objects

- Assign new value to param **supply**
  - ```
    let supply['GARY','plate'] :=
          supply['GARY','plate'] + supply_step;
    ```
- Problem instance changes accordingly

Sets are true data

- Assign new value to set SUPPLY
  - `let SUPPLY := SUPPLY union {supply['GARY','plate']};`
- All indexed entities change accordingly
    ```

2a: Solution Generation *via Cuts*

Same multicommodity transportation model

Generate n best solutions using different routes

- Display routes used by each solution

Solutions *via Cuts*

Script

```
param nSols default 0;
param maxSols = 3;

model multmipG.mod;
data multmipG.dat;

set USED {1..nSols} within {ORIG,DEST};

subject to exclude {k in 1..nSols}:

   sum {(i,j) in USED[k]} (1-Use[i,j]) +
   sum {(i,j) in {ORIG,DEST} diff USED[k]} Use[i,j] >= 1;

repeat {
   solve;
   display Use;
   let nSols := nSols + 1;
   let USED[nSols] := {i in ORIG, j in DEST: Use[i,j] > .5};
} until nSols = maxSols;
```

Solutions *via Cuts*

Run showing 3 best solutions

```
ampl: include multmipBestA.run;

Gurobi 5.6.0: optimal solution; objective 235625

:    DET FRA FRE LAF LAN STL WIN    :=
CLEV   1   1   1   0   1   1   0
GARY   0   0   0   1   0   1   1
PITT   1   1   1   1   0   1   0 ;

Gurobi 5.6.0: optimal solution; objective 237125

:    DET FRA FRE LAF LAN STL WIN    :=
CLEV   1   1   1   1   0   1   0
GARY   0   0   0   1   0   1   1
PITT   1   1   1   0   1   1   0 ;

Gurobi 5.6.0: optimal solution; objective 238225

:    DET FRA FRE LAF LAN STL WIN    :=

CLEV   1   0   1   0   1   1   1
GARY   0   1   0   1   0   1   0
PITT   1   1   1   1   0   1   0 ;
```

Solutions *via Cuts:* Observations

Same expressions describe sets and indexing

- Index a summation
  - `... sum {(i,j) in {ORIG,DEST} diff USED[k]} Use[i,j] >= 1;`
- Assign a value to a set
  - `let USED[nSols] := {i in ORIG, j in DEST: Use[i,j] > .5};`

New cuts defined automatically

- Index cuts over a set
  - `subject to exclude {k in 1..nSols}: ...`
- Add a cut by expanding the set
  - `let nSols := nSols + 1;`

2b: Solution Generation *via Solver*

Same model

Ask solver to return multiple solutions

- Set options
- Get all results from one "solve"
- Retrieve and display each solution

Solutions *via Solver*

Script

```
option solver cplex;

option cplex_options "poolstub=multmip poolcapacity=3 \
   populate=1 poolintensity=4 poolreplace=1";

solve;

for {i in 1..Current.npool} {

   solution ("multmip" & i & ".sol");
   display Use;
}
```

Solutions *via Solver*

Results

```
ampl: include multmipBestB.run;

CPLEX 12.6.0.0: poolstub=multmip
poolcapacity=3
populate=1
poolintensity=4
poolreplace=1

CPLEX 12.6.0.0: optimal integer solution; objective 235625
742 MIP simplex iterations
56 branch-and-bound nodes

Wrote 3 solutions in solution pool
to files multmip1.sol ... multmip3.sol.

Suffix npool OUT;
```

Solutions *via* Solver

Results (continued)

```
Solution pool member 1 (of 3); objective 235625

:    DET FRA FRE LAF LAN STL WIN  :=
CLEV   1   1   1   0   1   1   0
GARY   0   0   0   1   0   1   1
PITT   1   1   1   1   0   1   0 ;

Solution pool member 2 (of 3); objective 238225

:    DET FRA FRE LAF LAN STL WIN :=
CLEV   1   0   1   0   1   1   1
GARY   0   1   0   1   0   1   0
PITT   1   1   1   1   0   1   0 ;

Solution pool member 3 (of 3); objective 237125

:    DET FRA FRE LAF LAN STL WIN :=
CLEV   1   1   1   1   0   1   0
GARY   0   0   0   1   0   1   1
PITT   1   1   1   0   1   1   0 ;
```

Solutions *via Solver:* Observations

Filenames can be formed dynamically

- Write a (string expression)
- Numbers are automatically converted
  - `solution ("multmip" & i & ".sol");`

General Observations

Scripts in practice

- Large and complicated
  - Multiple files
  - Hundreds of statements
  - Millions of statements executed
- Run within broader applications

Prospective improvements

- Faster loops
- True script functions
  - Arguments and return values
  - Local sets & parameters
  - Callback functions

But . . .

Scripting

Limitations

Performance

- Interpreted language
- General set & data structures

Power

- Not a complete programming language
  - Specific to optimization models
- Not object-oriented

So . . .

AMPL API

Application Programming Interface

- General-purpose languages: C++, Java, .NET, Python
- Analytics languages: MATLAB, R

Facilitates use of AMPL for

- Complex algorithmic schemes
- Embedding in other applications
- Deployment of models

Example: Efficient Frontier

Compute a series of "nondominated" portfolios

- Model in AMPL
- Data & logic from a programming language
  - Java
  - MATLAB

Key to examples

- AMPL entities
- Java/MATLAB objects
- Java/MATLAB methods for working with AMPL

AMPL API Example

AMPL Model Files

Sets, parameters, variables

```
set stockall;                     # All stocks in the universe of assets
set stockrun within stockall;     # Stocks used for a given run
set stockopall within stockrun;   # Stocks that had weight > 0.5

param ncard default 10;               # Maximum cardinality of portfolio
param averret {stockall};             # Average return of each stock
param covar {stockall,stockall};      # Covariance matrix
param cutoffl default 0.0001;         # Weight > low cutoff ==> stock out
param cutoffh {stockall} default 1;   # Weight > high cutoff ==> stock in
param targetret default 0;            # Target return for efficient frontier

var weights {stockrun} >= 0;          # Weight of each stock in current run
var ifstock {stockrun} binary;        # = 1 iff a stock is in
var portret >= targetret;             # Portfolio return
```

AMPL API Example

AMPL Model Files

Objective, constraints

```
minimize cst:  # Overall risk
  sum {s in stockrun,s1 in stockrun} weights[s] * covar[s,s1] * weights[s1];

subject to invest:  # Weights must total 1
  sum{s in stockrun} weights[s]=1;

subject to defret:  # Returns must equal specified level
  sum{s in stockrun} averret[s] * weights[s] = portret;

subject to lowlnk {s in stockrun}:    # Weight >= low cutoff if stock is in
   weights[s] >= cutoffl * ifstock[s];

subject to uplink {s in stockrun}:    # Weight <= high cutoff if stock is in
   weights[s] <= cutoffh[s] * ifstock[s];

subject to fixing {s in stockopall}:  # Specified stocks must be in
   ifstock[s] = 1;

subject to carda:  # Limit on number of stocks in
   sum {s in stockrun} ifstock[s] <= ncard;
```

AMPL API Example

AMPL Model Files

Data table definitions

```
param data_dir symbolic;

table assetstable IN (data_dir & "/assetsReturns.bit"):
   stockall <- [stockall], averret;

table astrets IN (data_dir & "/covar.bit"):
   [Asset, stockall], covar;
```

AMPL API Example

Java Program

AMPL packages

```
package com.ampl.examples;

import com.ampl.AMPL;
import com.ampl.Objective;
import com.ampl.Parameter;
import com.ampl.Variable;

import java.io.IOException;
```

AMPL API Example

Java Program

Main class & method

```
public class EfficientFrontier {

   public static void main(String[] args) {

      int steps = 10;
      String modelDirectory = args[0];

      // initialize AMPL object

      AMPL ampl = new AMPL();

      try {

         // DETAILS OF THE PROGRAM

      } catch (IOException e) {
           System.out.println ("Model file not found.");

      } finally {
           ampl.close();
      }
   }

   // definitions of auxiliary methods
}
```

AMPL API Example

Java Program

Setup

```
// set AMPL options

ampl.setBoolOption("reset_initial_guesses", true);
ampl.setBoolOption("send_statuses", false);

// load AMPL model from file

ampl.read(modelDirectory + "/qpmv.mod");
ampl.read(modelDirectory + "/qpmvbit.run");

// pass tables directory to AMPL, then read tables

ampl.getParameter("data_dir").set(modelDirectory);

ampl.readTable("assetstable");
ampl.readTable("astrets");
```

AMPL API Example

Java Program

Initial solve

```
// associate Java variables with AMPL entities

Variable portfolioReturn = ampl.getVariable("portret");
Parameter averageReturn = ampl.getParameter("averret");
Parameter targetReturn = ampl.getParameter("targetret");
Objective deviation = ampl.getObjective("cst");

// run AMPL commands directly

ampl.eval("option solver gurobi;");
ampl.eval("let stockopall := {};");
ampl.eval("let stockrun := stockall;");

// relax integrality

ampl.setBoolOption("relax_integrality", true);

// solve

ampl.solve();
```

AMPL API Example

Java Program

Solve loop

```
// calibrate efficient frontier range

double minret = portfolioReturn.value();
double maxret = fMax(averageReturn.getValues().getColumnAsDoubles("val"));
double stepsize = (maxret - minret) / steps;

// initialize arrays to hold results

double[] returns = new double[steps];
double[] deviations = new double[steps];

for (int i = 0; i < steps; i++) {

   // SOLVE LOOP

}

// Display efficient frontier points

System.out.format("%-8s  %-8s%n", "RETURN", "DEVIATION");

for (int i = 0; i < returns.length; i++)
   System.out.format("%-6f  %-6f%n", returns[i], deviations[i]);
```

AMPL API Example

Java Program

Inside the solve loop (1)

```
for (int i = 0; i < steps; i++) {

   System.out.format
      ("Solving for return = %f%n", maxret - (i-1)*stepsize);

   // set target return to desired point

   targetReturn.setValues(maxret - (i-1) * stepsize);

   ampl.eval("let stockopall := {};");
   ampl.eval("let stockrun := stockall;");

   // relax integrality and solve

   ampl.setBoolOption("relax_integrality", true);
   ampl.solve();

   System.out.format("QP result = %f %n", deviation.value());
```

AMPL API Example

Java Program

Inside the solve loop (2)

```
    // adjust included stocks

    ampl.eval("let stockrun := {i in stockrun: weights[i] > 0};");
    ampl.eval("let stockopall := {i in stockrun: weights[i] > 0.5};");

    // restore integrality

    ampl.setBoolOption("relax_integrality", false);
    ampl.solve();

    System.out.format("QMIP result = %f%n", deviation.value());

    // save current frontier point

    returns[i] = maxret - (i-1)*stepsize;
    deviations[i] = deviation.value();
}
```

AMPL API Example

MATLAB Program

Setup

```
steps = 10;
modelDirectory = '/amplapi/examples/';

% initialize AMPL object

ampl = initAMPL;

% set AMPL options

ampl.setBoolOption("reset_initial_guesses", true);
ampl.setBoolOption("send_statuses", false);

% load AMPL model from file

ampl.read([modelDirectory 'qpmv.mod'])
ampl.read([modelDirectory 'qpmvbit.run'])

// pass tables directory to AMPL, then read tables

ampl.getParameter('data_dir').set(modelDirectory);

ampl.readTable('assetstable');
ampl.readTable('astrets');
```

AMPL API Example

MATLAB Program

Initial solve

```
% associate Java variables with AMPL entities

portfolioReturn = ampl.getVariable('portret');
averageReturn = ampl.getParameter('averret');
targetReturn = ampl.getParameter('targetret');
deviation = ampl.getObjective('cst');

% run AMPL commands directly

ampl.eval('option solver gurobi;');
ampl.eval('let stockopall := {};');
ampl.eval('let stockrun := stockall;');

% relax integrality

ampl.setBoolOption('relax_integrality', true);

% solve

ampl.solve();
```

AMPL API Example

MATLAB Program

Solve loop

```
% calibrate efficient frontier range

minret = portfolioReturn.value();
maxret = max(averageReturn.getValues());
stepsize = (maxret-minret)/steps;

% initialize arrays to hold results

returns = zeros(steps, 1);
deviations = zeros(steps, 1);

for i=1:steps

   % SOLVE LOOP

End

% Plot efficient frontier points

plot(returns, deviations)
```

AMPL API Example

MATLAB Program

Inside the solve loop (1)

```
for i=1:steps
   fprintf('Solving for return = %f\n', maxret - (i-1)*stepsize)

   % set target return to desired point

   targetReturn.setValues(maxret - (i-1)*stepsize);

   ampl.eval('let stockopall:={ };');
   ampl.eval('let stockrun:=stockall;');

   % relax integrality and solve

   ampl.setBoolOption('relax_integrality', 1);
   ampl.solve();

   fprintf('QP result = %f ', deviation.value())
```

AMPL API Example

MATLAB Program

Inside the solve loop (2)

```
    % adjust included stocks

    ampl.eval('let stockrun := {i in stockrun: weights[i]>0};');
    ampl.eval('let stockopall := {i in stockrun: weights[i]>0.5};');

    % restore integrality

    ampl.setBoolOption('relax_integrality', 0);
    ampl.solve();

    fprintf(' QMIP result = %f\n ', deviation.value());

    % save current frontier point

    returns(i) = maxret - (i-1)*stepsize;
    deviations(i) = deviation.value();
end
```

AMPL API

Data Transfer

Process

- Define symbolic sets & parameters in AMPL model
- Create corresponding objects in program
- Transfer data using API methods
  - Program to AMPL
  - AMPL to program

Methods for transfer between . . .

- Scalar values
- Collections of values
  - AMPL indexed expressions
  - Java arrays, MATLAB matrices
- Relational tables
  - AMPL “table” structures
  - API DataFrame objects in Java, MATLAB

Deployment Alternatives

Stand-alone: Give (temporary) control to AMPL

- Write needed files
- Invoke AMPL to run some scripts
- Read the files that AMPL leaves on exit

API: Interact with AMPL

- Execute AMPL statements individually
- Read model, data, script files when convenient
- Exchange data tables directly with AMPL
  - populate sets & parameters
  - invoke any available solver
  - extract values of variables & result expressions

. . . all embedded within your program's logic

Planned Availability

Initial languages: Java, MATLAB

- Beta test
  - April 2014
  - *Seeking beta testers now*
- Release
  - Summer 2014
  - Available with all AMPL distributions

More languages to follow

- C++, C# (.NET), Python, R

Development details

- Partnership with OptiRisk Systems
- Long-term development & maintenance by AMPL

More Natural Formulations

Logical conditions

- Implications (`==>`)
- Disjunctions (`or`)

Convex quadratics

- Objectives and constraints
- Elliptic and conic

Multicommodity Revisited

Minimum-shipment constraints

- From each origin to each destination, *either* ship nothing *or* ship at least `minload` units

Conventional linear mixed-integer formulation

```
var Trans {ORIG,DEST,PROD} >= 0;
var Use {ORIG, DEST} binary;
.......
subject to Multi {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];

subject to Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] >= minload * Use[i,j];
```

Multi-Commodity

Zero-One Alternatives

Mixed-integer formulation using implications

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:
   Use[i,j] = 1  ==>
      minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j]
            else sum {p in PROD} Trans[i,j,p] = 0;
```

Solved directly by CPLEX

```
ampl: model multmipImpl.mod;
ampl: data multmipG.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.5.1.0: optimal integer solution; objective 235625
176 MIP simplex iterations
0 branch-and-bound nodes
```

Multi-Commodity

Non-Zero-One Alternatives

Disjunctive constraint

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] = 0  or
   minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Solved by CPLEX?

```
ampl: model multmipDisj.mod;
ampl: data multmipG.dat;
ampl: solve;
CPLEX 12.5.1.0: logical constraint not indicator constraint.
ampl: option solver ilogcp;
ampl: option ilogcp_options 'optimizer cplex';
ampl: solve;
ilogcp 12.5.0: optimal solution
0 nodes, 175 iterations, objective 235625
```

Logical Conditions: Current

Expressions recognized by AMPL

- Disjunctions (`or`), implications (`==>`)
- Counting expressions (`count`),
  Counting constraints (`atleast`, `atmost`)
- Aggregate constraints (`alldiff`, `numberof`)

Solvers supported

- IBM CPLEX mixed-integer programming solver
  - Applied directly
  - Applied after automatic conversion to MIP
- Constraint programming solvers
  - IBM ILOG CP
  - Gecode
  - JaCoP

Logical Conditions: Planned

What the AMPL-solver interface will do

- Recognize transformable “not linear” expressions
  - Logical operators
  - Piecewise operators: abs, min, max
- Automatically transform to LPs or MILPs

New forms to be recognized

- Object-valued variables
  - `var JobForSlot {1..nSl+1} in JOBS;`
- Variables in subscripts
  - ```
    minimize TotalCost:
       sum {k in 1..nSl} setupCost[JobForSlot[k],JobForSlot[k+1]] + ...
    ```
- Set membership constraints
  - ```
    subject to SeqRestrictions {k in 1..nSl}
       (JobForSlot[k],JobForSlot[k+1]) in ALLOWED;
    ```

Convex Quadratics: Current

Problem types

- Elliptic: quadratic programs (QPs)
- Conic: second-order cone programs (SOCPs)

What the AMPL-solver interface does

- Recognize quadratic objectives & constraints
- Multiply out products of linear terms
- Send linear & quadratic coefficient lists to solver

What the solver does

- Detect elliptic forms numerically
- Detect conic forms by structural analysis

Convex Quadratics: Planned

What the AMPL-solver interface will do

- Recognize nonquadratic SOCP-equivalent problems
- Automatically transform
  to SOCPs recogizable by solvers

Forms to be recognized

- Sum of norms
- Sum of squares divided by linear
- Generalized geometric means
- Generalized *p*-norms
- log-Chebychev objectives

. . . combinations by sum, max, positive multiple (where possible)

AMPL IDE

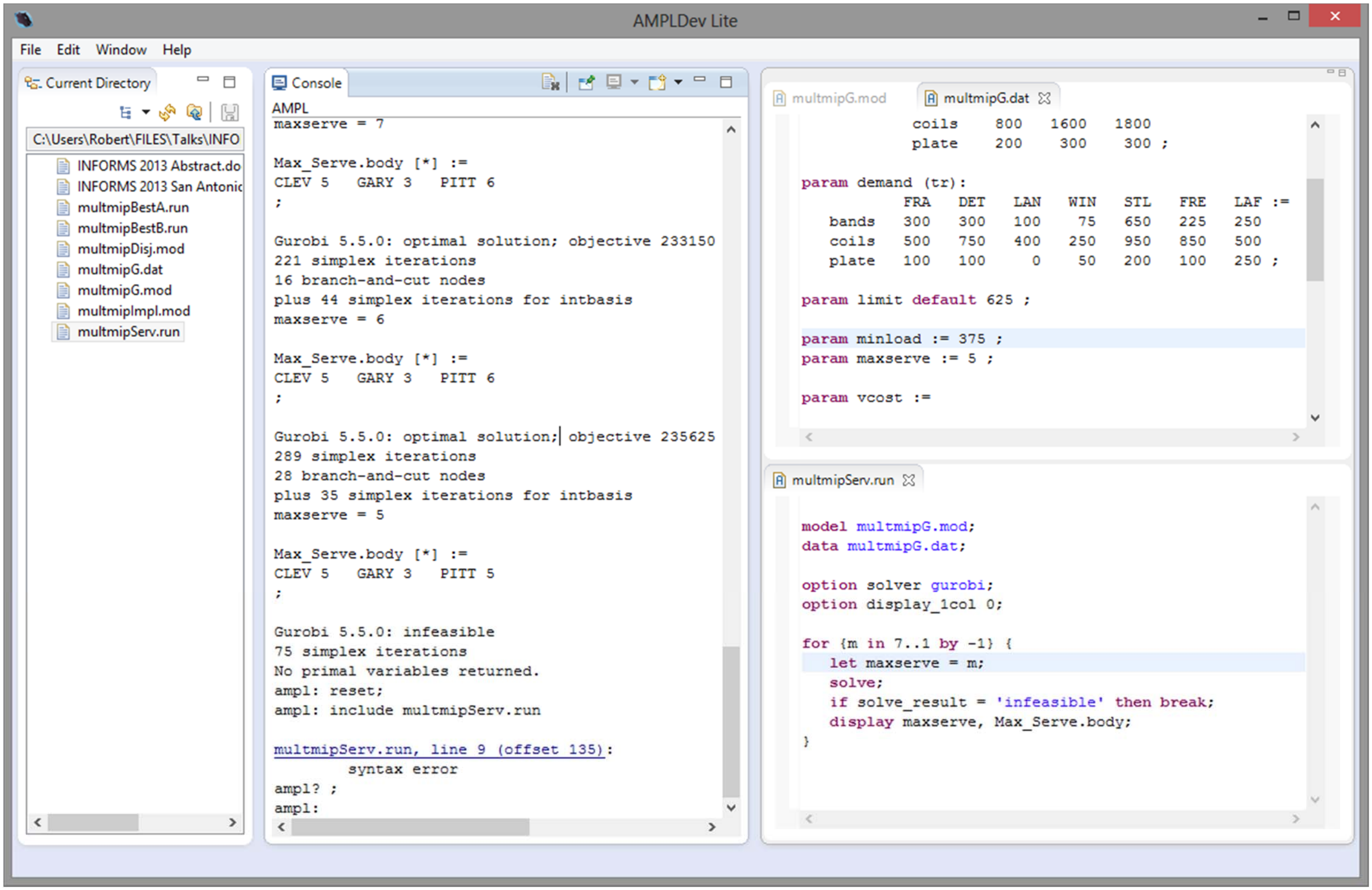
Integrated Development Environment

- ❖ Unified editor & command processor
- ❖ Included in the AMPL distribution
  - ∗ Easy upgrade path
  - ∗ Command-line, batch versions remain available
- ❖ Built on Eclipse
  - ∗ Runs under Windows, Linux, MacOS

Initial release

- ❖ Simplified for easy transition
- ❖ Works with existing installations

Sample Screenshot



Version 1.0

Now available

- Available with all AMPL distributions
- Download add-on at www.ampl.com/IDE
  - Integrated packages in preparation
- Version 1.01 updated with fixes available soon

Development details

- Partnership with OptiRisk Systems
- Long-term development & maintenance by AMPL
- "AMPLDEV" advanced IDE to be marketed by OptiRisk
  - Offers full stochastic programming support

Version *x.y*

More help

- Option selection dialogs
  - AMPL options
  - Solver options
- AMPL language quick reference

NEOS Server access

Enhanced displays

- Parameter view windows
- Graphs

Suggestions from users . . .

www.ampl.com/newsite

Readings *(AMPL)*

- R. Fourer, “Modeling Languages versus Matrix Generators for Linear Programming.” *ACM Transactions on Mathematical Software* **9** (1983) 143–183.
- R. Fourer, D.M. Gay, B.W. Kernighan, “A Modeling Language for Mathematical Programming.” *Management Science* **36** (1990) 519–554.
- Robert Fourer, “Database Structures for Mathematical Programming Models.” *Decision Support Systems* **20** (1997) 317–344.
- R. Fourer, D.M. Gay, B.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Belmont, CA (first edition 1993, second edition 2003).
- Robert Fourer, On the Evolution of Optimization Modeling Systems. M. Groetschel (ed.), *Optimization Stories*. Documenta Mathematica (2012) 377-388.