AMPL Models for “Not Linear” Optimization Using Linear Solvers

Robert Fourer

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

Industrial Eng & Management Sciences, Northwestern Univ

INFORMS Annual Meeting
Charlotte, NC — November 13-16, 2011
Session TC10, *Software Demonstrations*

AMPL Models for Unconventional Optimization Using Conventional Solvers

Robert Fourer

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

Industrial Eng & Management Sciences, Northwestern Univ

INFORMS Annual Meeting

Charlotte, NC — November 13-16, 2011
Session TC10, *Software Demonstrations*

AMPL

Algebraic modeling language: symbolic data

```
set SHIFTS;                 # shifts

param Nsched;               # number of schedules;
set SCHEDS = 1..Nsched;     # set of schedules

set SHIFT_LIST {SCHEDS} within SHIFTS;

param rate {SCHEDS} >= 0;        # pay rates
param required {SHIFTS} >= 0;    # staffing requirements

param least_assign >= 0;         # min workers on any schedule used
```

AMPL

Algebraic modeling language: symbolic model

```
var Work {SCHEDS} >= 0 integer;
var Use  {SCHEDS} >= 0 binary;

minimize Total_Cost:
   sum {j in SCHEDS} rate[j] * Work[j];

subject to Shift_Needs {i in SHIFTS}:
   sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j] >= required[i];

subject to Least_Use1 {j in SCHEDS}:
   least_assign * Use[j] <= Work[j];

subject to Least_Use2 {j in SCHEDS}:
   Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

AMPL

Explicit data independent of symbolic model

```
set SHIFTS := Mon1 Tue1 Wed1 Thu1 Fri1 Sat1
              Mon2 Tue2 Wed2 Thu2 Fri2 Sat2
              Mon3 Tue3 Wed3 Thu3 Fri3 ;

param Nsched := 126 ;

set SHIFT_LIST[1] := Mon1 Tue1 Wed1 Thu1 Fri1 ;
set SHIFT_LIST[2] := Mon1 Tue1 Wed1 Thu1 Fri2 ;
set SHIFT_LIST[3] := Mon1 Tue1 Wed1 Thu1 Fri3 ;
set SHIFT_LIST[4] := Mon1 Tue1 Wed1 Thu1 Sat1 ;
set SHIFT_LIST[5] := Mon1 Tue1 Wed1 Thu1 Sat2 ;  .......

param required :=  Mon1 100  Mon2 78  Mon3 52
                   Tue1 100  Tue2 78  Tue3 52
                   Wed1 100  Wed2 78  Wed3 52
                   Thu1 100  Thu2 78  Thu3 52
                   Fri1 100  Fri2 78  Fri3 52
                   Sat1 100  Sat2 78 ;
```

AMPL

Solver independent of model & data

```
ampl: model sched1.mod;
ampl: data sched.dat;

ampl: let least_assign := 7;

ampl: option solver cplex;
ampl: solve;

CPLEX 12.3.0.1: optimal integer solution; objective 266
1131 MIP simplex iterations
142 branch-and-bound nodes

ampl: option omit_zero_rows 1, display_1col 0;
ampl: display Work;

Work [*] :=
  6 28     31  9     66 11     89  9    118 18
 18 18     36  7     78 26     91 25    119  7
 20  9     37 18     82 18    112 27    122 36
;
```

AMPL

Language independent of solver

```
ampl: option solver gurobi;
ampl: solve;

Gurobi 4.5.0: optimal solution; objective 266
504 simplex iterations
50 branch-and-cut nodes

ampl: display Work;

Work [*] :=
  1 20    37 36    89 28   101 12   119  7
  2  8    71  7    91 16   109 28   122  8
 21 36    87  7    95  8   116 17   124 28
;
```

Topics

Discontinuous domains

- Semi-continuous case
- Discrete case

Implications

- CPLEX indicator constraints

Piecewise-linear terms

Complementarity conditions

Quadratic functions

- Elliptic forms
- Conic forms

Discontinuous Domains

Formulation with zero-one variables

```
var Work {SCHEDS} >= 0 integer;
var Use  {SCHEDS} >= 0 binary;

subject to Least_Use1 {j in SCHEDS}:
   least_assign * Use[j] <= Work[j];

subject to Least_Use2 {j in SCHEDS}:
   Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

Formulation with discrete domains

```
var Work {j in SCHEDS} integer, in {0} union
   interval [least_assign, (max {i in SHIFT_LIST[j]} required[i])];
```

Two Common Cases

Instead of a continuous variable . . .

```
var Buy {j in FOOD} >= 0;
```

Semi-continuous case

```
var Buy {j in FOOD} in {0} union interval[30,40];
```

Discrete case

```
var Buy {j in FOOD} in {1,2,5,10,20,50};
```

. . . any union of points & intervals possible

Semi-Continuous Case

Continuous

```
CPLEX 12.3.0.1: optimal solution; objective 88.2
1 dual simplex iterations (0 in phase I)

ampl: display Buy;

BEEF  0       FISH  0        MCH 46.6667    SPG  0
 CHK  0        HAM  0        MTL  0         TUR  0
```

Semi-Continuous

```
CPLEX 12.3.0.1: optimal integer solution; objective 116.4

27 MIP simplex iterations
5 branch-and-bound nodes

ampl: display Buy;

BEEF  0       FISH  0        MCH 30         SPG  0
 CHK  0        HAM  0        MTL 30         TUR  0
```

Semi-Continuous Case *(cont'd)*

Continuous

```
8 variables, all linear

4 constraints, all linear; 31 nonzeros

1 linear objective; 8 nonzeros.
```

Semi-Continuous

```
16 variables:
        8 binary variables
        8 linear variables

20 constraints, all linear; 63 nonzeros

1 linear objective; 8 nonzeros.
```

Discontinuous Domains

Semi-Continuous Case *(cont'd)*

Converted to MIP with extra binary variables . . .

```
subject to (Buy[BEEF]+IUlb):
    Buy['BEEF'] - 30*(Buy[BEEF]+b) >= 0;
subject to (Buy[BEEF]+IUub):
    -Buy['BEEF'] + 40*(Buy[BEEF]+b) >= 0;

subject to (Buy[CHK]+IUlb):
    Buy['CHK'] - 30*(Buy[CHK]+b) >= 0;
subject to (Buy[CHK]+IUub):
    -Buy['CHK'] + 40*(Buy[CHK]+b) >= 0;

subject to (Buy[FISH]+IUlb):
    Buy['FISH'] - 30*(Buy[FISH]+b) >= 0;
subject to (Buy[FISH]+IUub):
    -Buy['FISH'] + 40*(Buy[FISH]+b) >= 0;

.......
```

Discontinuous Domains

Discrete Case

Continuous

```
CPLEX 12.3.0.1: optimal solution; objective 88.2
1 dual simplex iterations (0 in phase I)

ampl: display Buy;

BEEF  0       FISH  0        MCH 46.6667    SPG  0
 CHK  0        HAM  0        MTL   0        TUR  0
```

Discrete

```
CPLEX 12.3.0.1: optimal integer solution; objective 95.49

51 MIP simplex iterations
23 branch-and-bound nodes

ampl: display Buy;

BEEF  1   FISH  1    MCH 10    SPG  5
 CHK 20    HAM  1    MTL  2    TUR  1
```

Discrete Case *(cont'd)*

Continuous

```
8 variables, all linear
4 constraints, all linear; 31 nonzeros
1 linear objective; 8 nonzeros.
```

Discrete

```
Substitution eliminates 8 variables.
48 variables, all binary
12 constraints, all linear; 234 nonzeros
1 linear objective; 48 nonzeros.
```

Discrete Case *(cont'd)*

Converted to MIP in binary variables . . .

```
minimize Total_Cost:
3.19*(Buy[BEEF]+b)[0] + 6.38*(Buy[BEEF]+b)[1] +
15.95*(Buy[BEEF]+b)[2] + 31.9*(Buy[BEEF]+b)[3] +
63.8*(Buy[BEEF]+b)[4] + 159.5*(Buy[BEEF]+b)[5] +
2.59*(Buy[CHK]+b)[0] + 5.18*(Buy[CHK]+b)[1] +
12.95*(Buy[CHK]+b)[2] + 25.9*(Buy[CHK]+b)[3] +
51.8*(Buy[CHK]+b)[4] + 129.5*(Buy[CHK]+b)[5] + ...

subject to Diet['A']:
700 <= 60*(Buy[BEEF]+b)[0] + 120*(Buy[BEEF]+b)[1] +
300*(Buy[BEEF]+b)[2] + 600*(Buy[BEEF]+b)[3] +
1200*(Buy[BEEF]+b)[4] + 3000*(Buy[BEEF]+b)[5] +
8*(Buy[CHK]+b)[0] + 16*(Buy[CHK]+b)[1] + 40*(Buy[CHK]+b)[2] +
80*(Buy[CHK]+b)[3] + 160*(Buy[CHK]+b)[4] + 400*(Buy[CHK]+b)[5] + ...
```

Discontinuous Domains

Discrete Case *(cont'd)*

and SOS type 1 constraints . . .

```
subject to (Buy[BEEF]+sos1):
(Buy[BEEF]+b)[0] + (Buy[BEEF]+b)[1] + (Buy[BEEF]+b)[2] +
(Buy[BEEF]+b)[3] + (Buy[BEEF]+b)[4] + (Buy[BEEF]+b)[5] = 1;

subject to (Buy[CHK]+sos1):
(Buy[CHK]+b)[0] + (Buy[CHK]+b)[1] + (Buy[CHK]+b)[2] +
(Buy[CHK]+b)[3] + (Buy[CHK]+b)[4] + (Buy[CHK]+b)[5] = 1;  ...
```

Discrete Case *(cont'd)*

with SOS type 1 markers in output file

```
S0 48 sos
0 20
1 20
2 20
3 20
4 20
5 20
6 36
7 36  ...
S4 48 sosref
0 1
1 2
2 5
3 10
4 20
5 50
6 1
7 2  ...
```

Conversion for Solver

General case

- Arbitrary union of points and intervals
- Auxiliary binary variable for each point or interval
- 3 auxiliary constraints for each variable

Union of points

- Auxiliary binary variable for each point
- Auxiliary constraint for each variable
- Enhanced branching in solver
  - "special ordered sets of type 1"

Zero union interval (semi-continuous)

- Auxiliary binary variable for each variable
- 2 auxiliary constraints for each variable
- Enhanced branching in solver

Implications

Formulation with zero-one variables

```
subject to Least_Use1 {j in SCHEDS}:
   least_assign * Use[j] <= Work[j];

subject to Least_Use2 {j in SCHEDS}:
   Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

Formulation with implications

```
subject to Least_Use1_logical {j in SCHEDS}:
   Use[j] = 1 ==> Work[j] >= least_assign;

subject to Least_Use2_logical {j in SCHEDS}:
   Use[j] = 0 ==> Work[j] = 0;
```

```
subject to Least_Use_logical {j in SCHEDS}:
   Use[j] = 1 ==> least_assign <= Work[j] else Work[j] = 0;
```

Implications

Design of Conditional Operators

General possibilities

- Conditional expression
- Conditional constraint
- Conditional command

AMPL syntax choices

- `if` *condition* `then` *expr1* `else` *expr2*
- *condition* ==> *constraint1* `else` *constraint2*
  - also <== and <==>
- `if` *condition* `then` {*commands*} `else` {*commands*}

Supported by solvers

- Nonlinear if-then-else
- CPLEX indicator constraints

Implications

Nonlinear *if-then-else*

More stable expression near zero

```
subject to logRel {j in 1..N}:
   (if X[j] < -delta || X[j] > delta
      then log(1+X[j]) / X[j] else (1 - X[j] / 2) <= logLim;
```

CPLEX Indicator Constraints

Indicator constraints

- (*binary variable* = 0) implies *constraint*
- (*binary variable* = 1) implies *constraint*

. . . handled directly by solver

AMPL "implies" operator

- Use `==>` for "implies"
- Also recognize an `else` clause
- Similarly define `<==` and `<==>`
  - `if-then-else` expressions & statements as before

Implications

Example 1

Multicommodity flow with fixed costs

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

param supply {ORIG,PROD} >= 0;  # amounts available at origins
param demand {DEST,PROD} >= 0;  # amounts required at destinations
param limit {ORIG,DEST} >= 0;

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

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

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];
```

Implications

Example 1 *(cont'd)*

Conventional 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 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 UseDefinition {i in ORIG, j in DEST, p in PROD}:
   Trans[i,j,p] <= min(supply[i,p], demand[j,p]) * Use[i,j];
```

Example 1 *(cont'd)*

Indicator constraint formulations

```
subject to DefineUsedA {i in ORIG, j in DEST}:
   Use[i,j] = 0 ==> sum {p in PROD} Trans[i,j,p] = 0;
```

```
subject to DefineUsedB {i in ORIG, j in DEST, p in PROD}:
   Use[i,j] = 0 ==> Trans[i,j,p] = 0;
```

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

Implications

Example 2

Assignment to groups with "no one isolated"

```
var Lone {(i1,i2) in ISO, j in REST} binary;

param give {ISO} default 2;
param giveTitle {TITLE} default 2;
param giveLoc {LOC} default 2;

param upperbnd {(i1,i2) in ISO, j in REST} :=
   min (ceil((number2[i1,i2]/card {PEOPLE}) * hiDine[j]) + give[i1,i2],
        hiTargetTitle[i1,j] + giveTitle[i1],
        hiTargetLoc[i2,j] + giveLoc[i2], number2[i1,i2]);

subj to Isolation1 {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] <= upperbnd[i1,i2,j] * Lone[i1,i2,j];

subj to Isolation2a {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] >= Lone[i1,i2,j];

subj to Isolation2b {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] +
      sum {ii1 in ADJACENT[i1]: (ii1,i2) in TYPE2} Assign2[ii1,i2,j]
         >= 2 * Lone[i1,i2,j];
```

Implications

Example 2

Same using indicator constraints

```
var Lone {(i1,i2) in ISO, j in REST} binary;

subj to Isolation1 {(i1,i2) in ISO, j in REST}:
   Lone[i1,i2,j] = 0 ==> Assign2[i1,i2,j] = 0;

subj to Isolation2b {(i1,i2) in ISO, j in REST}:
   Lone[i1,i2,j] = 1 ==> Assign2[i1,i2,j] +
     sum {ii1 in ADJACENT[i1]: (ii1,i2) in TYPE2} Assign2[ii1,i2,j] >= 2;
```

Implications

Example 3

Workforce planning

```
var LayoffCost {m in MONTHS} >=0;

subj to LayoffCostDefn1 {m in MONTHS}:
   LayoffCost[m]
      <= snrLayOffWages * 31 * maxNbrSnrEmpl * (1 - NoShut[m]);

subj to LayoffCostDefn2a {m in MONTHS}:
   LayoffCost[m] - snrLayOffWages * ShutdownDays[m] * maxNbrSnrEmpl
      <= maxNbrSnrEmpl * 2 * dayAvail[m] * snrLayOffWages * NoShut[m];

subj to LayoffCostDefn2b {m in MONTHS}:
   LayoffCost[m] - snrLayOffWages * ShutdownDays[m] * maxNbrSnrEmpl
      >= -maxNbrSnrEmpl * 2 * dayAvail[m] * snrLayOffWages * NoShut[m];
```

Implications

Example 3

Same using indicator constraints

```
var LayoffCost {m in MONTHS} >=0;

subj to LayoffCostDefn1 {m in MONTHS}:
   NoShut[m] = 1 ==> LayoffCost[m] = 0;

subj to LayoffCostDefn2 {m in MONTHS}:
   NoShut[m] = 0 ==> LayoffCost[m] =
      snrLayoffWages * ShutdownDays[m] * maxNumberSnrEmpl;
```

Implications

Conversion for Solver

Pass logic to CPLEX

- AMPL writes “logical” constraints as expression trees
- AMPL-CPLEX driver “walks” the trees
  - detects indicator forms
  - converts to CPLEX library calls
- CPLEX solves using standard MIP software

```
ampl: solve;

252 variables, all nonlinear
17 algebraic constraints, all linear; 630 nonzeros
         17 inequality constraints
126 logical constraints
1 linear objective; 2 nonzeros.

CPLEX 12.3.0.1: optimal integer solution; objective 266
1265016 MIP simplex iterations
231882 branch-and-bound nodes
```

Implications

Which is Fastest?

```
Use[j] = 1 ==> least_assign <= Work[j] else Work[j] = 0;
```

```
CPLEX 12.3.0.1: optimal integer solution; objective 266
1265016 MIP simplex iterations
231882 branch-and-bound nodes
```

```
least_assign * Use[j] <= Work[j];
Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

```
CPLEX 12.3.0.1: optimal integer solution; objective 266
776836 MIP simplex iterations
109169 branch-and-bound nodes
```

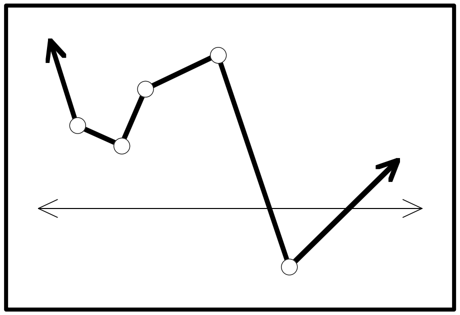
```
Use[j] = 1 ==> least_assign <= Work[j] <=
   (max {i in SHIFT_LIST[j]} required[i]) else Work[j] = 0;
```

```
CPLEX 12.3.0.1: optimal integer solution; objective 266
13470 MIP simplex iterations
2161 branch-and-bound nodes
```

Piecewise-Linear Terms

Definition

- Function of one variable
- Linear on intervals
- Continuous



Issues

- Describing the function
  - choice of specification
  - syntax in the modeling language
- Communicating the function to a solver
  - direction description
  - transformation to linear or linear-integer

Piecewise-Linear

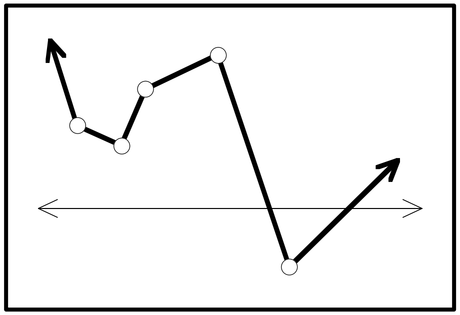
Specification

Possibilities

- List of breakpoints and either:
  - change in slope at each breakpoint
  - value of the function at each breakpoint
- List of slopes and either:
  - distance between breakpoints bounding each slope
  - value of intercept associated with each slope
- **Lists of breakpoints and slopes**

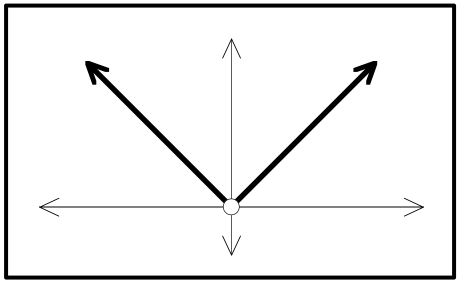
Also needed in some cases

- One particular breakpoint
- One particular slope
- **Value at one particular point**

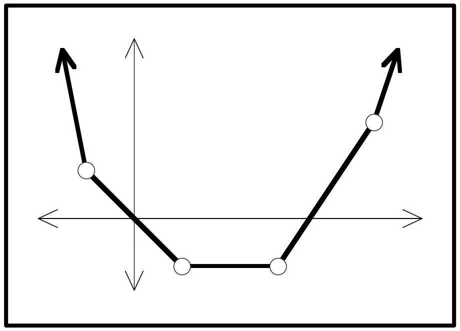


Piecewise-Linear

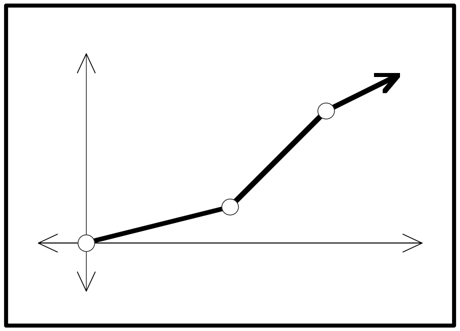
AMPL Specification: Examples



```
<<0; -1,1>> x[j]
```



```
<<-1,1,3,5; -5,-1,0,1.5,3>> x[j]
```



```
<<3,5; 0.25,1.00,0.50>> x[j]
```

AMPL Specification: Syntax

General forms

- *<breakpoint-list*; *slope-list> variable*
  - Zero at zero
  - Bounds on variable specified independently
- *<breakpoint-list*; *slope-list>* (*variable*, *zero-point*)
  - Zero at *zero-point*
- *<breakpoint-list*; *slope-list> variable* + *constant*
  - Has value *constant* at zero

Breakpoint & slope list forms

- Simple list
  - `<<lim1[i,j],lim2[i,j]; r1[i,j],r2[i,j],r3[i,j]>>`
- Indexed list
  - ```
    << {k in 1..nlim[i,j]} lim[i,j,k];
       {k in 1..nlim[i,j]+1} r[i,j,k]>>
    ```
    ```

Piecewise-Linear

AMPL Applications (1)

Design of a planar structure

```
var Force {bars};    # Forces on bars:
                     # positive in tension, negative in compression

minimize TotalWeight:  (density / yield_stress) *

   sum {(i,j) in bars} length[i,j] * <<0; -1,+1>> Force[i,j];

                     # Weight is proportional to length
                     # times absolute value of force

subject to Xbal {k in joints: k <> fixed}:

     sum {(i,k) in bars} xcos[i,k] * Force[i,k]
   - sum {(k,j) in bars} xcos[k,j] * Force[k,j] = xload[k];

subject to Ybal {k in joints: k <> fixed and k <> rolling}:

     sum {(i,k) in bars} ycos[i,k] * Force[i,k]
   - sum {(k,j) in bars} ycos[k,j] * Force[k,j] = yload[k];

                     # Forces balance in
                     # horizontal and vertical directions
```

AMPL Applications (2)

Data fitting for credit scoring

```
var Wt_const;                # Constant term in computing all scores

var Wt {j in factors} >= if wttyp[j] = 'pos' then 0 else -Infinity
                      <= if wttyp[j] = 'neg' then 0 else +Infinity;
                             # Weights on the factors

var Sc {i in people};        # Scores for the individuals

minimize Penalty:            # Sum of penalties for all individuals

   Gratio * sum {i in Good} << {k in 1..Gpce-1} if Gbktyp[k] = 'A'
                                   then Gbkfac[k]*app_amt
                                   else Gbkfac[k]*bal_amt[i];
                               {k in 1..Gpce} Gslope[k] >> Sc[i] +

   Bratio * sum {i in Bad}  << {k in 1..Bpce-1} if Bbktyp[k] = 'A'
                                   then Bbkfac[k]*app_amt
                                   else Bbkfac[k]*bal_amt[i];
                               {k in 1..Bpce} Bslope[k] >> Sc[i];
```

Piecewise-Linear

Conversion for Solver

Transportation costs

```
param rate1 {i in ORIG, j in DEST} >= 0;
param rate2 {i in ORIG, j in DEST} >= rate1[i,j];
param rate3 {i in ORIG, j in DEST} >= rate2[i,j];

param limit1 {i in ORIG, j in DEST} >= 0;
param limit2 {i in ORIG, j in DEST} >= limit1[i,j];

var Trans {ORIG,DEST} >= 0;

minimize Total_Cost:

   sum {i in ORIG, j in DEST}
      <<limit1[i,j], limit2[i,j];
        rate1[i,j], rate2[i,j], rate3[i,j]>> Trans[i,j];
```

Minimizing Convex Costs

Equivalent linear program

```
ampl: model trpl2.mod; data trpl.dat; solve;

Substitution eliminates 15 variables.
21 piecewise-linear terms replaced by 35 variables and 15 constraints.

Adjusted problem:
41 variables, all linear
10 constraints, all linear; 82 nonzeros
1 linear objective; 41 nonzeros.

CPLEX 10.1.0: optimal solution; objective 199100
12 dual simplex iterations (0 in phase I)

ampl: display Trans;

:       DET   FRA   FRE   LAF   LAN   STL   WIN  :=

CLEV    500     0   200   500   500    500  400
GARY      0     0   900   300     0    200    0
PITT    700   900     0   200   100   1000    0 ;
```

Minimizing Non-Convex Costs

Equivalent mixed-integer program

```
model trpl3.mod; data trpl.dat; solve;

Substitution eliminates 18 variables.
21 piecewise-linear terms replaced by 87 variables and 87 constraints.

Adjusted problem:
90 variables:
        41 binary variables
        49 linear variables
79 constraints, all linear; 251 nonzeros
1 linear objective; 49 nonzeros.

CPLEX 10.1.0: optimal integer solution; objective 256100
189 MIP simplex iterations
144 branch-and-bound nodes

ampl: display Trans;

:      DET    FRA    FRE    LAF    LAN    STL    WIN   :=

CLEV   1200     0      0   1000      0      0    400
GARY      0     0   1100      0    300      0      0
PITT      0   900      0      0    300   1700      0
```

Piecewise-Linear

Minimizing Non-Convex Costs *(cont'd)*

. . . with SOS type 2 markers in output file

```
S0 87 sos
 3 16
49 18
 4 16
50 18  ...

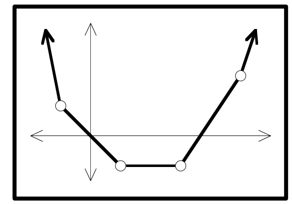
S1 64 sos
10 19
11 18
12 18
14 35  ...

S4 46 sosref
3 -501
4  751
5 -501
6  500  ...
```

Piecewise-Linear

Conversion for Solver

Equivalent linear program if . . .



- Objective
  - minimizes convex (increasing slopes) *or*
  - maximizes concave (decreasing slopes)
- Constraints expressions
  - convex and on the left-hand side of a $\le$ constraint
  - convex and on the right-hand side of a $\ge$ constraint
  - concave and on the left-hand side of a $\ge$ constraint
  - concave and on the right-hand side of a $\le$ constraint

Equivalent mixed-integer program otherwise

- At least one binary variable per piece
- Enhanced branching in solver
  - "special ordered sets of type 2"

Complementarity Conditions

Closely associated with optimization

- Two inequalities must both hold
- At least one must hold with equality

Now can be readily solved

- Send to standard solver like KNITRO
- Let solver reformulate for tractability

Quadratic Functions

Elliptic functions

Conic functions

Elliptic Quadratic: Example

Portfolio optimization

```
set A;                        # asset categories
set T := {1973..1994};        # years

param R {T,A};                # returns on asset categories
param mu default 2;           # weight on variance

param mean {j in A} = (sum {i in T} R[i,j]) / card(T);

param Rtilde {i in T, j in A} = R[i,j] - mean[j];

var Frac {A} >=0;

var Mean = sum {j in A} mean[j] * Frac[j];

var Variance =
   sum {i in T} (sum {j in A} Rtilde[i,j]*Frac[j])^2 / card{T};

minimize RiskReward:  mu * Variance - Mean;

subject to TotalOne:  sum {j in A} Frac[j] = 1;
```

Elliptic Quadratic

Example *(cont'd)*

Portfolio data

```
set A :=
  US_3-MONTH_T-BILLS US_GOVN_LONG_BONDS SP_500 WILSHIRE_5000
  NASDAQ_COMPOSITE LEHMAN_BROTHERS_CORPORATE_BONDS_INDEX EAFE GOLD;

param R:
  US_3-MONTH_T-BILLS US_GOVN_LONG_BONDS SP_500 WILSHIRE_5000
  NASDAQ_COMPOSITE LEHMAN_BROTHERS_CORPORATE_BONDS_INDEX EAFE GOLD :=

1973  1.075  0.942  0.852  0.815  0.698  1.023  0.851  1.677
1974  1.084  1.020  0.735  0.716  0.662  1.002  0.768  1.722
1975  1.061  1.056  1.371  1.385  1.318  1.123  1.354  0.760
1976  1.052  1.175  1.236  1.266  1.280  1.156  1.025  0.960
1977  1.055  1.002  0.926  0.974  1.093  1.030  1.181  1.200
1978  1.077  0.982  1.064  1.093  1.146  1.012  1.326  1.295
1979  1.109  0.978  1.184  1.256  1.307  1.023  1.048  2.212
1980  1.127  0.947  1.323  1.337  1.367  1.031  1.226  1.296
1981  1.156  1.003  0.949  0.963  0.990  1.073  0.977  0.688
1982  1.117  1.465  1.215  1.187  1.213  1.311  0.981  1.084
1983  1.092  0.985  1.224  1.235  1.217  1.080  1.237  0.872
1984  1.103  1.159  1.061  1.030  0.903  1.150  1.074  0.825 ...
```

Elliptic Quadratic

Example *(cont'd)*

Solving with CPLEX

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

ampl: option solver cplexamp;

ampl: solve;

8 variables, all nonlinear
1 constraint, all linear; 8 nonzeros
1 nonlinear objective; 8 nonzeros.

CPLEX 12.2.0.0: optimal solution; objective -1.098362471
12 QP barrier iterations

ampl:
```

Example *(cont'd)*

Solving with CPLEX (simplex)

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

ampl: option solver cplexamp;
ampl: option cplex_options 'primalopt';

ampl: solve;

8 variables, all nonlinear
1 constraint, all linear; 8 nonzeros
1 nonlinear objective; 8 nonzeros.

CPLEX 12.2.0.0: primalopt
No QP presolve or aggregator reductions.

CPLEX 12.2.0.0: optimal solution; objective -1.098362476
5 QP simplex iterations (0 in phase I)

ampl:
```

Elliptic Quadratic

Example *(cont'd)*

Optimal portfolio

```
ampl: option omit_zero_rows 1;

ampl: display Frac;

                                 EAFE  0.216083
                                 GOLD  0.185066
LEHMAN_BROTHERS_CORPORATE_BONDS_INDEX  0.397056
                         WILSHIRE_5000  0.201795 ;

ampl: display Mean, Variance;

Mean = 1.11577
Variance = 0.00870377

ampl:
```

Elliptic Quadratic

Example *(cont'd)*

Optimal portfolio (discrete)

```
var Share {A} integer >= 0, <= 100;

var Frac {j in A} = Share[j] / 100;
```

```
ampl: solve;

CPLEX 12.2.0.0: optimal integer solution within mipgap or absmipgap;
   objective -1.098353751

10 MIP simplex iterations
0 branch-and-bound nodes

absmipgap = 8.72492e-06, relmipgap = 7.94364e-06

ampl: display Frac;

                               EAFE  0.22
                               GOLD  0.18
LEHMAN_BROTHERS_CORPORATE_BONDS_INDEX  0.4
                       WILSHIRE_5000  0.2 ;
```

Elliptic Quadratic

Detection for Solver

Symbolic detection

- Objectives
  - Minimize $x_1^2 + \ldots + x_n^2$
  - Minimize $\sum_{i=1}^{n} a_i(\mathbf{f}_i\mathbf{x} + g_i)^2,\ \ a_i \geq 0$
- Constraints
  - $x_1^2 + \ldots + x_n^2 \leq r$
  - $\sum_{i=1}^{n} a_i(\mathbf{f}_i\mathbf{x} + g_i)^2 \leq r,\ \ a_i \geq 0$

Numerical detection

- Objectives
  - Minimize $\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{q}\mathbf{x}$
- Constraints
  - $\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{q}\mathbf{x} \leq r$
- . . . where **Q** is positive semidefinite

Solving

Representation

- Much like LP
  - Coefficient lists for linear terms
  - Coefficient lists for quadratic terms
- A lot simpler than general NLP

Optimization

- Much like LP
  - Generalizations of barrier methods
  - Generalizations of simplex methods
  - Extensions of mixed-integer branch-and-bound schemes
- Simple derivative computations
- Less overhead than general-purpose nonlinear solvers

. . . ***actual speedup will vary***

Conic Quadratic: Example

Traffic network: symbolic data

```
set INTERS;          # intersections (network nodes)

param EN symbolic;   # entrance
param EX symbolic;   # exit

   check {EN,EX} not within INTERS;

set ROADS within {INTERS union {EN}} cross {INTERS union {EX}};

                         # road links (network arcs)

param base {ROADS} > 0;  # base travel times
param sens {ROADS} > 0;  # traffic sensitivities
param cap {ROADS} > 0;   # capacities

param through > 0;       # throughput
```

Conic Quadratic

Example *(cont'd)*

Traffic network: symbolic model

```
var Flow {(i,j) in ROADS} >= 0, <= .9999 * cap[i,j];
var Time {ROADS} >= 0;

minimize Avg_Time:
  (sum {(i,j) in ROADS} Time[i,j] * Flow[i,j]) / through;

subject to Travel_Time {(i,j) in ROADS}:
  Time[i,j] = base[i,j] + (sens[i,j]*Flow[i,j]) / (1-Flow[i,j]/cap[i,j]);

subject to Balance_Node {i in INTERS}:
  sum{(i,j) in ROADS} Flow[i,j] = sum{(j,i) in ROADS} Flow[j,i];

subject to Balance_Enter:
  sum{(EN,j) in ROADS} Flow[EN,j] = through;
```

Conic Quadratic

Example *(cont'd)*

Traffic network: sample data

```
set INTERS := b c ;

param EN := a ;
param EX := d ;

param: ROADS: base cap sens :=
         a b    4   10   .1
         a c    1   12   .7
         c b    2   20   .9
         b d    1   15   .5
         c d    6   10   .1 ;

param through := 20 ;
```

Conic Quadratic

Example *(cont'd)*

Model + data = problem to solve, using KNITRO

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

ampl: option solver knitro;
ampl: solve;

KNITRO 7.0.0: Locally optimal solution.
objective 61.04695019; feasibility error 3.55e-14
12 iterations; 25 function evaluations

ampl: display Flow, Time;

:        Flow       Time    :=
a b     9.55146   25.2948
a c    10.4485    57.5709
b d    11.0044    21.6558
c b     1.45291    3.41006
c d     8.99562   14.9564
;
```

Example *(cont'd)*

Same with integer-valued variables

```
var Flow {(i,j) in ROADS} integer >= 0, <= .9999 * cap[i,j];
```

```
ampl: solve;

KNITRO 7.0.0: Locally optimal solution.
objective 76.26375; integrality gap 0
3 nodes; 5 subproblem solves

ampl: display Flow, Time;

:     Flow   Time  :=
a b     9     13
a c    11     93.4
b d    11     21.625
c b     2      4
c d     9     15
;
```

Conic Quadratic

Example *(cont'd)*

Model + data = problem to solve, using CPLEX?

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

ampl: option solver cplex;
ampl: solve;

CPLEX 12.3.0.0:
Constraint _scon[1] is not convex quadratic
since it is an equality constraint.
```

Conic Quadratic

Example *(cont'd)*

Look at the model again . . .

```
var Flow {(i,j) in ROADS} >= 0, <= .9999 * cap[i,j];
var Time {ROADS} >= 0;

minimize Avg_Time:
  (sum {(i,j) in ROADS} Time[i,j] * Flow[i,j]) / through;

subject to Travel_Time {(i,j) in ROADS}:
  Time[i,j] = base[i,j] + (sens[i,j]*Flow[i,j]) / (1-Flow[i,j]/cap[i,j]);

subject to Balance_Node {i in INTERS}:
  sum{(i,j) in ROADS} Flow[i,j] = sum{(j,i) in ROADS} Flow[j,i];

subject to Balance_Enter:
  sum{(EN,j) in ROADS} Flow[EN,j] = through;
```

Conic Quadratic

Example *(cont'd)*

Quadratically constrained reformulation

```
var Flow {(i,j) in ROADS} >= 0, <= .9999 * cap[i,j];
var Delay {ROADS} >= 0;

minimize Avg_Time:
  sum {(i,j) in ROADS} (base[i,j]*Flow[i,j] + Delay[i,j]) / through;

subject to Delay_Def {(i,j) in ROADS}:
  sens[i,j] * Flow[i,j]^2 <= (1 - Flow[i,j]/cap[i,j]) * Delay[i,j];

subject to Balance_Node {i in INTERS}:
  sum{(i,j) in ROADS} Flow[i,j] = sum{(j,i) in ROADS} Flow[j,i];

subject to Balance_Enter:
  sum{(EN,j) in ROADS} Flow[EN,j] = through;
```

Conic Quadratic

Example *(cont'd)*

Model + data = problem to solve, using CPLEX?

```
ampl: model trafficQUAD.mod;
ampl: data traffic.dat;

ampl: option solver cplex;
ampl: solve;

CPLEX 12.3.0.0:
QP Hessian is not positive semi-definite.
```

Example *(cont'd)*

Quadratic reformulation #2

```
var Flow {(i,j) in ROADS} >= 0, <= .9999 * cap[i,j];
var Delay {ROADS} >= 0;
var Slack {ROADS} >= 0;

minimize Avg_Time:
  sum {(i,j) in ROADS} (base[i,j]*Flow[i,j] + Delay[i,j]) / through;

subject to Delay_Def {(i,j) in ROADS}:
  sens[i,j] * Flow[i,j]^2 <= Slack[i,j] * Delay[i,j];

subject to Slack_Def {(i,j) in ROADS}:
  Slack[i,j] = 1 - Flow[i,j]/cap[i,j];

subject to Balance_Node {i in INTERS}:
  sum{(i,j) in ROADS} Flow[i,j] = sum{(j,i) in ROADS} Flow[j,i];

subject to Balance_Enter:
  sum{(EN,j) in ROADS} Flow[EN,j] = through;
```

Conic Quadratic

Example *(cont'd)*

Model + data = problem to solve, using CPLEX!

```
ampl: model trafficSOC.mod;
ampl: data traffic.dat;

ampl: option solver cplex;
ampl: solve;

CPLEX 12.3.0.0: primal optimal; objective 61.04693968
15 barrier iterations

ampl: display Flow;

Flow :=
a b    9.55175
a c   10.4482
b d   11.0044
c b    1.45264
c d    8.99561
;
```

Conic Quadratic

Example *(cont'd)*

Same with integer-valued variables

```
var Flow {(i,j) in ROADS} integer >= 0, <= .9999 * cap[i,j];
```

```
ampl: solve;

CPLEX 12.3.0.0: optimal integer solution within mipgap or absmipgap;
   objective 76.26375017
19 MIP barrier iterations
0 branch-and-bound nodes

ampl: display Flow;

Flow :=
a b    9
a c   11
b d   11
c b    2
c d    9
;
```

Which Solver Is Preferable?

General nonlinear solver

- Fewer variables
- More natural formulation

MIP solver with convex quadratic option

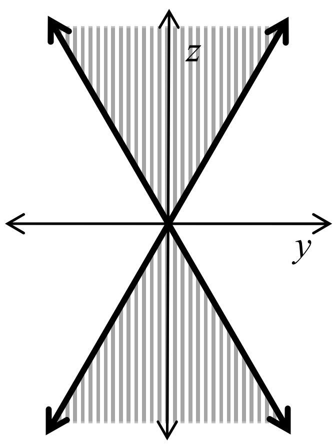
- Mathematically simpler formulation
- No derivative evaluations
  - no problems with nondifferentiable points
- More powerful large-scale solver technologies

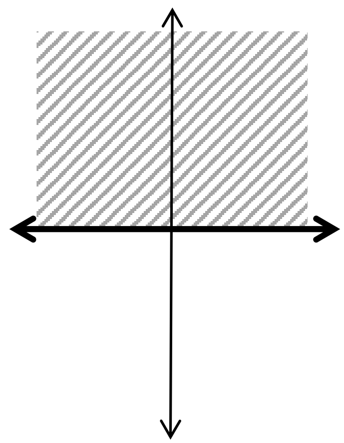
Conic Quadratic

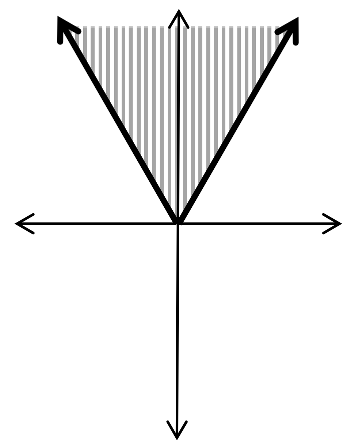
Second-Order Cone Programs (SOCPs)

Standard cone

$x^2 + y^2 \le z^2$ $\qquad$ $z \ge 0$ $\qquad$ $x^2 + y^2 \le z^2,\ z \ge 0$







. . . boundary not smooth

Rotated cone

- $x^2 \le yz,\ y \ge 0, z \ge 0, \ldots$

Conic Quadratic

Detection for Solver

Symbolic detection

- Constraints (standard)
  - $x_1^2 + \ldots + x_n^2 \leq x_{n+1}^2,\ x_{n+1} \geq 0$
  - $\sum_{i=1}^{n} a_i(\mathbf{f}_i\mathbf{x} + g_i)^2 \leq a_{n+1}(\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})^2,$
    $a_1, \ldots, a_{n+1} \geq 0,\ \mathbf{f}_{n+1}\mathbf{x} + g_{n+1} \geq 0$
- Constraints (rotated)
  - $x_1^2 + \ldots + x_n^2 \leq x_{n+1}\, x_{n+2},\ x_{n+1} \geq 0,\ x_{n+2} \geq 0$
  - $\sum_{i=1}^{n} a_i(\mathbf{f}_i\mathbf{x} + g_i)^2 \leq a_{n+1}(\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})(\mathbf{f}_{n+2}\mathbf{x} + g_{n+2}),$
    $a_1, \ldots, a_{n+1} \geq 0,\ \mathbf{f}_{n+1}\mathbf{x} + g_{n+1} \geq 0,\ \mathbf{f}_{n+2}\mathbf{x} + g_{n+2} \geq 0$

Numerical detection

- $\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{q}\mathbf{x} \leq r$
- . . . where **Q** has one negative eigenvalue
  - see Ashutosh Mahajan and Todd Munson, "Exploiting Second-Order Cone Structure for Global Optimization"

Detection & Conversion for Solver

SOC-representable functions

- Quadratic-linear ratios
- Geometric means and generalizations
- Norms, p-norms, and generalizations

Available transformations

- Objectives: Minimize $s(x)$
- Constraints: $s(x) \leq ax + b$, where $ax + b \geq 0$
- Combinations of these
  - sums
  - minimums
  - positive multiples

Other objective functions

- Generalized product-of-powers
- Logarithmic Chebychev

Solving

Similarities to elliptic quadratic

- Describe by lists of coefficients
- Solve by extensions of LP barrier methods
- Extend to mixed-integer branch-and-bound

Differences from elliptic quadratic

- Quadratic part not positive semi-definite
- Nonnegativity is essential
- Boundary of feasible region is not differentiable

Survey of Test Problems

12% of 1238 nonlinear problems were SOC-solvable!

- not counting QPs with sum-of-squares objectives
- from Vanderbei's CUTE & non-CUTE, and netlib/ampl

A variety of forms detected

- `hs064` has $4/x_1 + 32/x_2 + 120/x_3 \leq 1$
- `hs036` minimizes $-x_1 x_2 x_3$
- `hs073` has $1.645 \sqrt{0.28x_1^2 + 0.19x_2^2 + 20.5x_3^2 + 0.62x_4^2} \leq \ldots$
- `polak4` is a max of sums of squares
- `hs049` minimizes $(x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6$
- `emfl_nonconvex` has $\sum_{k=1}^{2} \left(x_{jk} - a_{ik}\right)^2 \leq s_{ij}^2$

. . . survey of integer programs to come
. . . solver tests to come