Strategies for Using Algebraic Modeling Languages to Formulate Second-Order Cone Programs

Robert Fourer, Jared Erickson

Industrial Engineering & Management Sciences
Northwestern University

4er@northwestern.edu, jarederickson2012@u.northwestern.edu

21st International Symposium on Mathematical Programming
Berlin — 20-24 August 2012

Thu.1.H1058 Modeling Languages and Software I

Strategies for Using Algebraic Modeling Languages to Formulate Second-Order Cone Programs

A surprising variety of optimization applications can be written as convex quadratic problems that linear solvers can be extended to handle effectively. Particular interest has focused on conic constraint regions and the "second-order cone programs" (or SOCPs) that they define. Whether given quadratic constraints define a convex cone can in principle be determined numerically, but of greater interest are the varied combinations of sums and maxima of Euclidean norms, quadratic-linear ratios, products of powers, p-norms, and log-Chebychev terms that can be identified and transformed symbolically. The power and convenience of algebraic modeling language may be extended to support such forms, with the help of a recursive tree-walk approach that detects and converts arbitrarily complex instances – freeing modelers from the time-consuming and error-prone work of maintaining the equivalent SOCPs explicitly. These facilities moreover integrate well with other common linear and quadratic transformations. We describe the challenges of creating the requisite detection and transformation routines, and report computational tests using the AMPL language.

Example: Traffic Network

Given

N Set of nodes representing intersections

e Entrance to network

f Exit from network

$A \subseteq N \cup \{e\} \times N \cup \{f\}$

Set of arcs representing road links

and

b_{ij} Base travel time for each road link $(i, j) \in A$

s_{ij} Traffic sensitivity for each road link $(i, j) \in A$

c_{ij} Capacity for each road link $(i, j) \in A$

T Desired throughput from e to f

Formulation

Determine

x_{ij} Traffic flow through road link $(i,j) \in A$

t_{ij} Actual travel time on road link $(i,j) \in A$

to minimize

$\Sigma_{(i,j)\in A}\, t_{ij} x_{ij} / T$

Average travel time from e to f

Traffic Network

Formulation *(cont'd)*

Subject to

$$t_{ij} = b_{ij} + \frac{s_{ij}x_{ij}}{1 - x_{ij}/c_{ij}} \quad \text{for all } (i,j) \in A$$

Travel times increase as flow approaches capacity

$$\Sigma_{(i,j)\in A}\, x_{ij} = \Sigma_{(j,i)\in A}\, x_{ji} \quad \text{for all } i \in N$$

Flow out equals flow in at any intersection

$$\Sigma_{(e,j)\in A}\, x_{ej} = T$$

Flow into the entrance equals the specified throughput

AMPL Formulation

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
```

Traffic Network

AMPL Formulation *(cont'd)*

Symbolic model

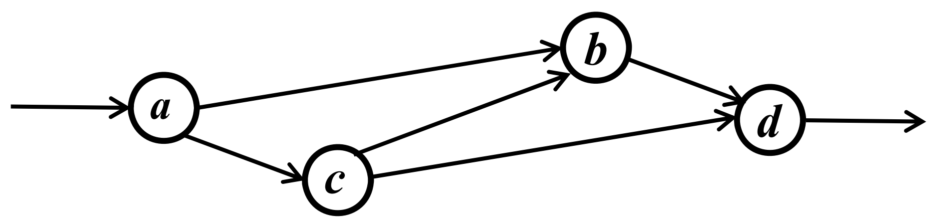
```
var Flow {(i,j) in ROADS} >= 0, <= .9999 * cap[i,j];
var Time {ROADS} >= 0;

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;
```

Traffic Network

AMPL Data

Explicit data independent of symbolic model

```
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 ;
```



Traffic Network

AMPL Solution

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
;
```

AMPL Solution *(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
;
```

AMPL Solution *(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.
```

Traffic Network

AMPL Solution *(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;
```

Traffic Network

AMPL Solution *(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;
```

AMPL Solution *(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.
```

AMPL Solution *(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;
```

Traffic Network

AMPL Solution *(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
;
```

Traffic Network

AMPL Solution *(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

- Convex quadratic formulation
  - known global optimum
- No derivative evaluations
  - no problems with nondifferentiable points
- Specialized large-scale solver technologies

What we're studying

- Write the model in its natural formulation
- Convert automatically to the convex quadratic formulation

Outline

Convex quadratic programs

- “Elliptic” forms
- “Conic” forms
  - ***Second Order Cone Programs: SOCPs***

SOCP-solvable forms

- Quadratic
- SOC-representable
- Other objective functions

Detection & transformation of SOCPs

- General principles
- Example: Sum of norms
- Survey of nonlinear test problems
- Two small computational examples

Convex Quadratic Programs

"Elliptic" quadratic programming

- Detection
- Solving

"Conic" quadratic programming

- Detection
- Solving

"Elliptic" Quadratic Programming

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

. . . ***your speedup may vary***

Elliptic QP

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 QP

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 QP

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:
```

Elliptic QP

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:
```

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:
```

Example *(cont'd)*

Optimal portfolio (discrete)

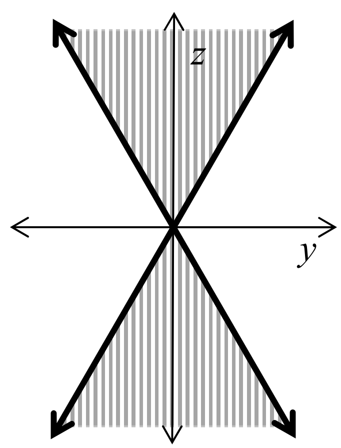
```
var Share {A} integer >= 0, <= 100;

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 ;
```

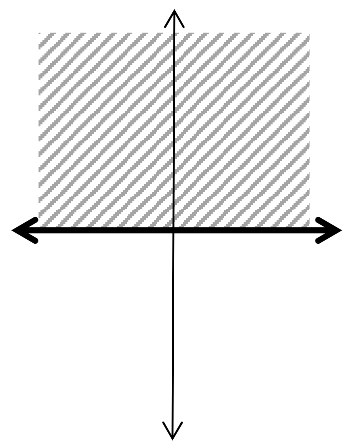
“Conic” Quadratic Programming

Standard cone

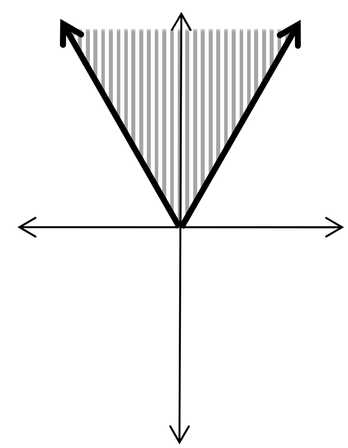
$x^2 + y^2 \leq z^2$



$z \geq 0$



$x^2 + y^2 \leq z^2, \ z \geq 0$



. . . boundary not smooth

Rotated cone

- $x^2 \leq yz, \ y \geq 0, z \geq 0, \ . \ . \ .$

"Conic" Quadratic Programming

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 $\mathbf{Q}$ has one negative eigenvalue
  - see Ashutosh Mahajan and Todd Munson, "Exploiting Second-Order Cone Structure for Global Optimization"

Conic QP

Solving

Similarities

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

Differences

- Quadratic part not positive semi-definite
- Nonnegativity is essential
- Boundary of feasible region is not differentiable
- ***Many convex problems can be reduced to these . . .***

SOCP-Solvable Forms

Quadratic

- Constraints (already seen)
- Objectives

SOC-representable

- Quadratic-linear ratios
- Generalized geometric means
- Generalized p-norms

Other objective functions

- Generalized product-of-powers
- Logarithmic Chebychev

SOCP-solvable

Quadratic

Standard cone constraints

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

Rotated cone constraints

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

Sum-of-squares objectives

- Minimize $\sum_{i=1}^{n} a_i(\mathbf{f}_i\mathbf{x} + g_i)^2$
  - Minimize v
    Subject to $\sum_{i=1}^{n} a_i(\mathbf{f}_i\mathbf{x} + g_i)^2 \leq v^2,\ v \geq 0$

SOCP-solvable

SOC-Representable

Definition

- Function $s(x)$ is SOC-representable *iff* . . .
- $s(x) \leq a_n(\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})$ is equivalent to some combination of linear and quadratic cone constraints

Minimization property

- Minimize $s(x)$ is SOC-solvable
  - ∗ Minimize $\quad v_{n+1}$
    Subject to $\quad s(x) \leq v_{n+1}$

Combination properties

- $a \cdot s(x)$ is SOC-representable for any $a \geq 0$
- $\sum_{i=1}^{n} s_i(x)$ is SOC-representable
- $max_{i=1}^{n}\, s_i(x)$ is SOC-representable

. . . ***requires a recursive detection algorithm!***

SOCP-solvable

SOC-Representable (1)

Vector norm

- $\|\mathbf{a} \cdot (\mathbf{F}\mathbf{x} + \mathbf{g})\| = \sqrt{\sum_{i=1}^{n} a_i^2 (\mathbf{f}_i\mathbf{x} + g_i)^2} \leq a_{n+1}(\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})$
  - square both sides to get standard SOC
    $\sum_{i=1}^{n} a_i^2 (\mathbf{f}_i\mathbf{x} + g_i)^2 \leq a_{n+1}^2 (\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})^2$

Quadratic-linear ratio

- $\dfrac{\sum_{i=1}^{n} a_i (\mathbf{f}_i\mathbf{x} + g_i)^2}{\mathbf{f}_{n+2}\mathbf{x} + g_{n+2}} \leq a_{n+1}(\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})$
  - where $\mathbf{f}_{n+2}\mathbf{x} + g_{n+2} \geq 0$
  - multiply by denominator to get rotated SOC
    $\sum_{i=1}^{n} a_i \, (\mathbf{f}_i\mathbf{x} + g_i)^2 \leq a_{n+1} \, (\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})(\mathbf{f}_{n+2}\mathbf{x} + g_{n+2})$

SOCP-solvable

SOC-Representable (2)

Negative geometric mean

- $-\prod_{i=1}^{p}(\mathbf{f}_i\mathbf{x}+g_i)^{1/p} \le \mathbf{f}_{n+1}\mathbf{x}+g_{n+1},\ \ p \in \mathbb{Z}^+$
  - $-x_1^{1/4}x_2^{1/4}x_3^{1/4}x_4^{1/4} \le -x_5$ becomes rotated SOCs: $x_5^2 \le v_1v_2,\ v_1^2 \le x_1x_2,\ v_2^2 \le x_3x_4$
  - apply recursively $\lceil \log_2 p \rceil$ times

Generalizations

- $-\prod_{i=1}^{n}(\mathbf{f}_i\mathbf{x}+g_i)^{\alpha_i} \le a_{n+1}(\mathbf{f}_{n+1}\mathbf{x}+g_{n+1})$: $\sum_{i=1}^{n}\alpha_i \le 1,\ \alpha_i \in \mathbb{Q}^+$
- $\prod_{i=1}^{n}(\mathbf{f}_i\mathbf{x}+g_i)^{-\alpha_i} \le a_{n+1}(\mathbf{f}_{n+1}\mathbf{x}+g_{n+1}),\ \ \alpha_i \in \mathbb{Q}^+$
  - all require $\mathbf{f}_i\mathbf{x}+g_i$ to have proper sign

SOCP-solvable

SOC-Representable (3)

p-norm

- $(\sum_{i=1}^{n} |\mathbf{f}_i\mathbf{x} + g_i|^p)^{1/p} \le \mathbf{f}_{n+1}\mathbf{x} + g_{n+1},\ p \in \mathbb{Q}^+,\ p \ge 1$
  - $(|x_1|^5 + |x_2|^5)^{1/5} \le x_3$ can be written
    $|x_1|^5/x_3^4 + |x_2|^5/x_3^4 \le x_3$ which becomes
    $v_1 + v_2 \le x_3$ with $-v_1^{1/5}\, x_3^{4/5} \le \pm x_1, -v_1^{1/5}\, x_3^{4/5} \le \pm x_2$
  - reduces to product of powers

Generalizations

- $(\sum_{i=1}^{n} |\mathbf{f}_i\mathbf{x} + g_i|^{\alpha_i})^{1/\alpha_0} \le \mathbf{f}_{n+1}\mathbf{x} + g_{n+1},\ \alpha_i \in \mathbb{Q}^+,\ \alpha_i \ge \alpha_0 \ge 1$
- $\sum_{i=1}^{n} |\mathbf{f}_i\mathbf{x} + g_i|^{\alpha_i} \le (\mathbf{f}_{n+1}\mathbf{x} + g_{n+1})^{\alpha_0}$
- Minimize $\sum_{i=1}^{n} |\mathbf{f}_i\mathbf{x} + g_i|^{\alpha_i}$

. . . ***standard SOCP has $\boldsymbol{\alpha_i \equiv 2}$***

Other Objective Functions

Unrestricted product of powers

- Minimize $-\prod_{i=1}^{n}(\mathbf{f}_i\mathbf{x} + g_i)^{\alpha_i}$ for any $\alpha_i \in \mathbb{Q}^+$

Logarithmic Chebychev approximation

- Minimize $\max_{i=1}^{n}|\log(\mathbf{f}_i\mathbf{x}) - \log(g_i)|$

Why no constraint versions?

- Not SOC-representable
- Transformation changes objective value (but not solution)

Detection & Transformation of SOCPs

Principles

- Representation of expressions by trees
- Recursive tree-walk functions
  - `isLinear()`, `isQuadratic()`, `buildLinear()`

Example: Sum of norms

Survey of nonlinear test problems

Two small computational examples

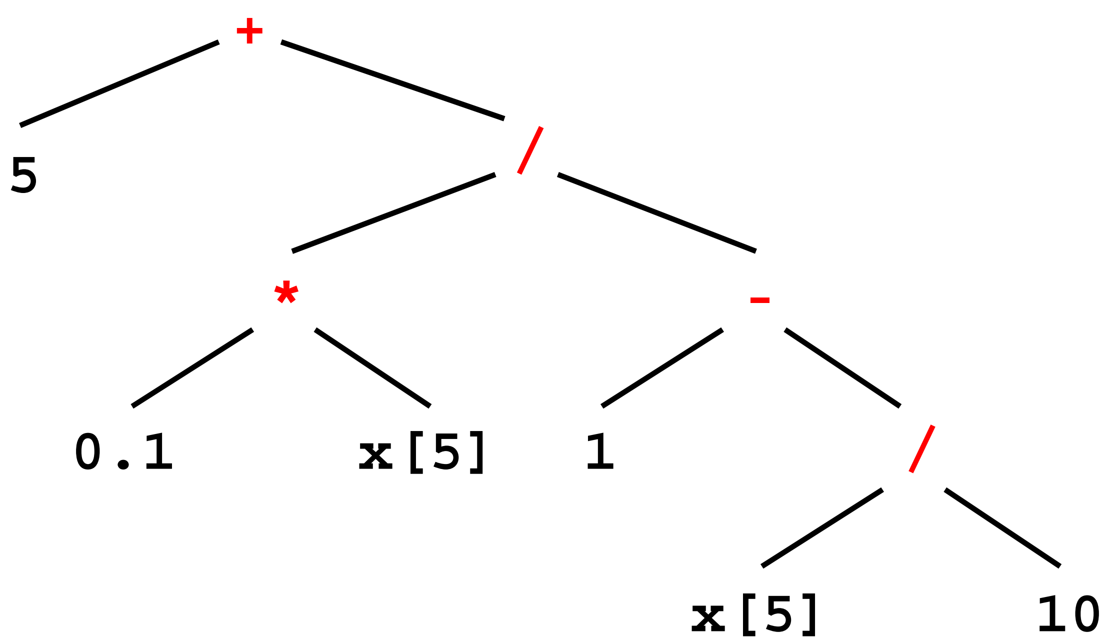
Principles

Representation

Expression

```
base[i,j] + (sens[i,j]*Flow[i,j]) / (1-Flow[i,j]/cap[i,j])
```

Expression tree



. . . actually a DAG

Principles

Detection: `isLinear()`

```
boolean isLinear (Node);

case of Node {

 PLUS:
 MINUS: return( isLinear(Node.left) and isLinear(Node.right) );

 TIMES: return( isConst(Node.left) and isLinear(Node.right) or
                isLinear(Node.left) and isConst(Node.right) );

 DIV:   return( isLinear(Node.left) and isConst(Node.right) );

 VAR:   return( TRUE );

 CONST: return( TRUE );

}
```

. . . to detect, test `isLinear(root)`

Principles

Detection: `isQuadr()`

```
boolean isQuadr (Node);

case of Node {

 PLUS:
 MINUS: return( isQuadr(Node.left) and isQuadr(Node.right) );

 TIMES: return( isLinear(Node.left) and isLinear(Node.right) or
                isQuadr(Node.left) and isConst(Node.right)   or
                isConst(Node.left) and isQuadr(Node.right) );

 POWER: return( isLinear(Node.left) and
                isConst(Node.right) and value(Node.right) == 2 );

 VAR:   return( TRUE );

 CONST: return( TRUE );

}
```

Transformation: buildLinear()

```
(coeff,const) = buildLinear (Node);

if Node.L then (coefL,consL) = buildLinear(Node.L);
if Node.R then (coefR,consR) = buildLinear(Node.R);

case of Node {

 PLUS:  coeff = mergeLists( coefL, coefR );
        const = consL + consR;

 TIMES: ...

 DIV:   coeff = coefL / consR;
        const = consL / consR;

 VAR:   coeff = makeList( 1, Node.index );
        const = 0;

 CONST: coeff = makeList( );
        const = Node.value;

}
```

. . . to transform, call `buildLinear(root)`

Example: Sum-of-Norms Objective

Given

- Minimize $\sum_{i=1}^{m} a_i \sqrt{\sum_{j=1}^{n_i} \left(\mathbf{f}_{ij}\mathbf{x} + g_{ij}\right)^2}$

Transform to

- Minimize $\sum_{i=1}^{m} a_i y_i$
- $\sum_{j=1}^{n_i} z_{ij}^2 \le y_i^2,\ \ y_i \ge 0, \qquad i = 1,\ldots,m$
- $z_{ij} = \mathbf{f}_{ij}\mathbf{x} + g_{ij}, \qquad i = 1,\ldots,m,\ \ j = 1,\ldots,n_i$

Two steps

- *Detection*
- *Transformation*

Sum of Norms

Detection

SUMOFNORMS

Sum: $e_1 + e_2$ is SUMOFNORMS if e_1, e_2 are SUMOFNORMS

Product: $e_1 e_2$ is SUMOFNORMS if
e_1 is SUMOFNORMS and e_2 is POSCONSTANT or
e_2 is SUMOFNORMS and e_1 is POSCONSTANT

Square root: $\sqrt{e}$ is SUMOFNORMS if e is SUMOFSQUARES

SUMOFSQUARES

Sum: $e_1 + e_2$ is SUMOFSQUARES if e_1, e_2 are SUMOFSQUARES

Product: $e_1 e_2$ is SUMOFSQUARES if
e_1 is SUMOFSQUARES and e_2 is POSCONSTANT or
e_2 is SUMOFSQUARES and e_1 is POSCONSTANT

Square: e^2 is SUMOFSQUARES if e is LINEAR

Constant: c is SUMOFSQUARES if c is POSCONSTANT

Detection Issues

Mathematical

- Minimize $\sum_{i=1}^{m} a_i \sqrt{\sum_{j=1}^{n_i} \left(\mathbf{f}_{ij}\mathbf{x} + g_{ij}\right)^2}$

Practical

- Constant multiples inside any sum
- Recursive nesting of constant multiples & sums
- Constant as a special case of a square
  - $\sqrt{3(4x_1 + 7(x_2 + 2x_3) + 6)^2 + (x_4 + x_5)^2 + 17}$

Transformation

TransformSumOfNorms (`Expr e, Obj o, real k`)

Sum: `e1 + e2` where `e1`, `e2` are SumOfNorms
  TransformSumOfNorms (`e1,o,k`)
  TransformSumOfNorms (`e2,o,k`)

Product: `e1 * c2` where `e1` is SumOfNorms and `c2` is PosConstant
  TransformSumOfNorms (`e1,o,c2*k`)

Product: `c1 * e2` where `e2` is SumOfNorms and `c1` is PosConstant
  TransformSumOfNorms (`e2,o,c1*k`)

Square root: `sqrt(e)` where `e` is SumOfSquares
  `yi` := NewNonnegVar(); `o += k * yi`
  `qi` := NewLeCon(); `qi += -yi^2`
  TransformSumOfSquares (`e,qi,1`)

Sum of Norms

Transformation *(cont'd)*

TRANSFORMSUMOFSQUARES (`Expr e, LeCon qi, real k`)

Sum: `e1 + e2` where `e1`, `e2` are SUMOFSQUARES
  TRANSFORMSUMOFSQUARES (`e1,o,k`)
  TRANSFORMSUMOFSQUARES (`e2,o,k`)

Product: `e1 * c2` where `e1` is SUMOFSQUARES and `c2` is POSCONSTANT
  TRANSFORMSUMOFSQUARES (`e1,o,c2*k`)

Product: `c1 * e2` where `e2` is SUMOFSQUARES and `c1` is POSCONSTANT
  TRANSFORMSUMOFSQUARES (`e2,o,c1*k`)

Square: `sqr(zij)` where `zij` is VARIABLE
  `qi += k * zij^2`

Square: `sqr(e)` where `e` is LINEAR
  `zij` := NEWVAR(); `qi += k * zij^2`
  `lij` := NEWEQCON(); `lij += zij - e`

Constant: `c` is POSCONSTANT
  `zij` := NEWVAR(); `qi += k * zij^2`
  `lij` := NEWEQCON(); `lij += zij – sqrt(c)`

Transformation Issues

Mathematical

- Minimize $\sum_{i=1}^{m} a_i y_i$
- $\sum_{j=1}^{n_i} z_{ij}^2 \leq y_i^2,\ y_i \geq 0$
- $z_{ij} = \mathbf{f}_{ij}\mathbf{x} + g_{ij}$

Practical

- Generalization: handle all previously mentioned
- Efficiency: don't define z_{ij} when $\mathbf{f}_{ij}\mathbf{x} + g_{ij}$ is a single variable
- Trigger by calling TRANSFORMSUMOFNORMS(`e`,`o`,`k`) with
  - `e` the root node
  - `o` an empty objective
  - `k` = 1

Challenges

Extending to all cases previously cited

- All prove amenable to recursive tree-walk
- Details much harder to work out

Checking nonnegativity of linear expressions

- Heuristic catches many non-obvious instances

Assessing usefulness . . .

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 $4/x_1 + 32/x_2 + 120/x_3 \le 1$
- `hs036` minimizes $-x_1 x_2 x_3$
- `hs073` has $1.645 \sqrt{0.28x_1^2 + 0.19x_2^2 + 20.5x_3^2 + 0.62x_4^2} \le \ldots$
- `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 $\sum_{k=1}^{2} \left(x_{jk} - a_{ik}\right)^2 \le s_{ij}^2$

. . . similar for nonlinear integer programs

Computational Experience

Two solver possibilities

- NLP: General-purpose mixed-integer nonlinear
  - KNITRO, Bonmin, BARON
- SOCP: Linear mixed-integer extended to convex quadratic
  - CPLEX, Gurobi, Xpress

Reliability: Advantage to SOCP

- Far fewer failures
- Global optimum is assured

Efficiency: Undecided

- Times can be comparable
- Limited experience with difficult integer models

Small Example 1

SOCP-solvable with nonsmooth functions

```
var x {1..5} integer;
var y {1..5} >= 0;

minimize obj: sum {i in 1..5} (
   sqrt( (x[i]+2)^2 + (y[i]+1)^2 ) + sqrt( (x[i]+y[i])^2 ) + y[3]^2 );

subj to xsum: sum {i in 1..5} x[i] <= -12;
subj to ysum: sum {i in 1..5} y[i] >= 10;

subj to socprep:
   max {i in 1..5} ( (x[i]^2 + 1)/(i+y[i]) + y[i]^3 ) <= 30;
```

Small Example 1 *(cont'd)*

General nonlinear solver (integer)

```
KNITRO 8.0.0: Convergence to an infeasible point.

Problem may be locally infeasible.
```

General nonlinear solver (continuous relaxation)

```
KNITRO 8.0.0:

--- ERROR evaluating objective gradient.
--- ERROR evaluating constraint gradients.

Evaluation error.

objective 17.14615551; feasibility error 0
233 iterations; 1325 function evaluations
```

Small Example 1 *(cont'd)*

Convex quadratic solver (integer)

```
CPLEX 12.4.0

Total time (root+branch&cut) = 0.21 sec.

Solution value  = 17.246212

:    x         y
1   -3   3
2   -2   1.99993
3   -2   0.000300084
4   -3   3
5   -2   1.99993
;
```

Computational Example *(cont'd)*

Convex quadratic solver (continuous relaxation)

```
CPLEX 12.4.0

Total time = 0.04 sec.

Solution value  = 17.141355

:     x          y
1   -2.49707   2.49707
2   -2.49707   2.49707
3   -2.01171   0.011716
4   -2.49707   2.49707
5   -2.49707   2.49707
;
```

Small Example 2

SOCP-solvable with p-norms

```
var x {1..100} >= 0 integer;

minimize obj:
   (sum {i in 1..60} x[i]^3) ^ (1/3) +
   (sum {i in 40..99} (1+x[i]-x[i+1])^4) ^ (1/4);

subject to c1 {i in 1..50}:
   x[i] + x[i+50] >= i/10;
```

Small Example 2 *(cont'd)*

General nonlinear solver (integer)

```
KNITRO 8.0.0: Locally optimal solution.

objective 4.24223232; integrality gap -1.5e-09
5517 nodes; 5517 subproblem solves

Total time = 70.1364 sec.
```

Convex quadratic solver (integer)

```
CPLEX 12.4.0

objective 4.242235
352 branch-and-cut nodes, 18360 iterations

Total time (root+branch&cut) = 6.45 sec.
```