Modeling and Solving Nontraditional Optimization Problems

Session 2a: Conic Constraints

Robert Fourer

Industrial Engineering & Management Sciences
Northwestern University

AMPL Optimization LLC

4er@northwestern.edu — 4er@ampl.com

Chiang Mai University International Conference
Workshop
Chiang Mai, Thailand — 4-5 January 2011

Session 2a: Conic Constraints

Focus

- Variety of quadratic optimization problems

Topics

- Traditional quadratic programming
- Conic quadratic programming
  (second-order cone programming)
  - Definition & example
  - Numerous equivalent problems
- Detection and transformation
  - Solver interaction with AMPL
  - Detection by recursive tree walk
  - Requirements for transformation

Traditional Quadratic Programming

Convex quadratic functions

- $x^T Q x + qx$
- Q is positive semi-definite
  - $x^T Q x \geq 0$ for all x

Formulation

- Minimize convex quadratic function
  - (or maximize concave quadratic)
- Subject to constraints on convex functions
  - convex quadratic function $\leq$ constant
  - linear equations and inequalities

Solvers

- Generalization of simplex method
- Generalization of barrier (interior) method
  - usable inside branch-and-bound framework for MIQP

Traditional 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;
```

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 ...
```

Traditional 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:
```

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 Programming

Convex quadratic constraint regions

- Ball: $x_1^2 + \ldots + x_n^2 \leq b$
- Cone: $x_1^2 + \ldots + x_n^2 \leq y^2, \;\; y \geq 0$
- Cone: $x_1^2 + \ldots + x_n^2 \leq yz, \;\; y \geq 0, \;\; z \geq 0$

. . . may substitute a linear term for any variable

Similarities

- Describe by lists of coefficients
- Solve by extensions of LP barrier methods; extend to MIP

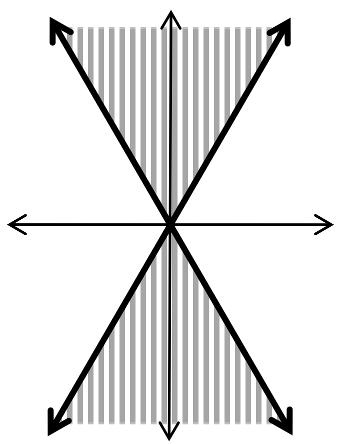
Differences

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

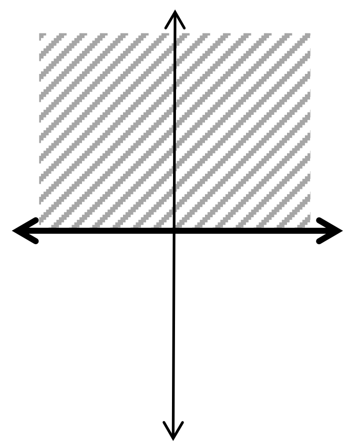
Geometry

Standard cone

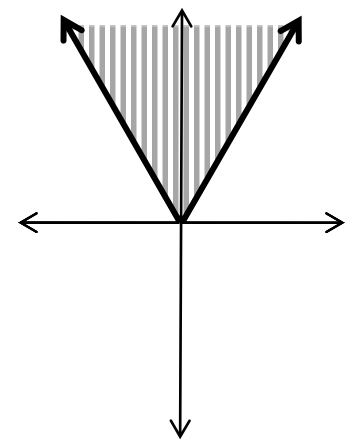
$x^2 \le y^2$



$y \ge 0$



$x^2 \le y^2,\ \ y \ge 0$



. . . boundary not smooth

Rotated cone

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

Applications

Antenna array weight design

Grasping force optimization

Finite impulse response filter design

Portfolio optimization with loss risk constraints

Truss design

Equilibrium of system with piecewise-linear springs

- Lobo, Vandenberghe, Boyd, Lebret, Applications of Second-Order Cone Programming. *Linear Algebra and Its Applications* 284 (1998) 193-228.

Example: Sum of Norms

```
param p integer > 0;
param m {1..p} integer > 0;
param n integer > 0;

param F {i in 1..p, 1..m[i], 1..n};
param g {i in 1..p, 1..m[i]};
```

```
param p := 2 ;
param m := 1 5  2 4 ;
param n := 3 ;

param g (tr): 1   2 :=
          1  12   2
          2   7  11
          3   7   1
          4   8   0
          5   4   . ;

param F := ...
```

Example: Original Formulation

```
var x {1..n};

minimize SumOfNorms:
   sum {i in 1..p} sqrt(
      sum {k in 1..m[i]} (sum {j in 1..n} F[i,k,j] * x[j] + g[i,k])^2 );
```

```
3 variables, all nonlinear
0 constraints
1 nonlinear objective; 3 nonzeros.

CPLEX 12.2.0.0: at12228.nl contains a nonlinear objective.
```

Example: Converted to Quadratic

```
var x {1..n};
var Max {1..p} >= 0;

minimize SumOfNorms: sum {i in 1..p} Max[i];

subj to MaxDefinition {i in 1..p}:
   sum {k in 1..m[i]} (sum {j in 1..n} F[i,k,j] * x[j] + g[i,k])^2
      <= Max[i]^2;
```

```
5 variables, all nonlinear
2 constraints, all nonlinear; 8 nonzeros
1 linear objective; 2 nonzeros.

CPLEX 12.2.0.0: QP Hessian is not positive semi-definite.
```

Example: Simpler Quadratic

```
var x {1..n};
var Max {1..p} >= 0;
var Fxplusg {i in 1..p, 1..m[i]};

minimize SumOfNorms: sum {i in 1..p} Max[i];

subj to MaxDefinition {i in 1..p}:
   sum {k in 1..m[i]} Fxplusg[i,k]^2 <= Max[i]^2;

subj to FxplusgDefinition {i in 1..p, k in 1..m[i]}:
   Fxplusg[i,k] = sum {j in 1..n} F[i,k,j] * x[j] + g[i,k];
```

```
14 variables:
      11 nonlinear variables
      3 linear variables
11 constraints; 41 nonzeros
      2 nonlinear constraints
      9 linear constraints
1 linear objective; 2 nonzeros.

CPLEX 12.2.0.0: primal optimal; objective 11.03323293
11 barrier iterations
```

Example: Integer Quadratic

```
var xint {1..n} integer;
var x {j in 1..n} = xint[j] / 10;
.......
```

```
Substitution eliminates 3 variables.
14 variables:
      11 nonlinear variables
      3 integer variables
11 constraints; 41 nonzeros
      2 nonlinear constraints
      9 linear constraints
1 linear objective; 2 nonzeros.
CPLEX 12.2.0.0: optimal integer solution; objective 11.12932573
88 MIP simplex iterations
19 branch-and-bound nodes
```

Equivalent Problems: Minimize

Sums of . . .

- norms or squared norms
  - $\sum_i \|F_i x + g_i\|$
  - $\sum_i (F_i x + g_i)^2$
- quadratic-linear fractions
  - $\sum_i \frac{(F_i x + g_i)^2}{a_i x + b_i}$

Max of . . .

- norms
  - $\max_i \|F_i x + g_i\|$
- logarithmic Chebychev terms
  - $\max_i |\log(F_i x) - \log(g_i)|$

Equivalent Problems: Objective

Products of . . .

- negative powers
  - $\min \prod_i (F_i x + g_i)^{-\alpha_i}$ for rational $\alpha_i > 0$
- positive powers
  - $\max \prod_i (F_i x + g_i)^{\alpha_i}$ for rational $\alpha_i > 0$

Combinations by . . .

- sum, max, positive multiple
  - except log Chebychev and some positive powers

$$\text{minimize}\ \max\{\sum_{i=1}^{p}(a_i x + b_i)^2, \sum_{j=1}^{q} \frac{\|F_j x + g_j\|^2}{y_j}\} + \prod_{k=1}^{r}(c_k x)^{-\pi_k}$$

Equivalent Problems: Constraints

Sums of . . .

- norms or squared norms
  - $\sum_i \|F_i x + g_i\| \le F_0 x + g_0$
  - $\sum_i (F_i x + g_i)^2 \le (F_0 x + g_0)^2$
- quadratic-linear fractions
  - $\sum_i \dfrac{(F_i x + g_i)^2}{a_i x + b_i} \le F_0 x + g_0$

Max of . . .

- norms
  - $\max_i \|F_i x + g_i\| \le F_0 x + g_0$

Equivalent Problems: Constraints

Products of . . .

- negative powers
  - $\sum_j \prod_i (F_{ji}x + g_{ji})^{-\alpha_{ji}} \le F_0 x + g_0$ for rational $\alpha_{ji} > 0$
- positive powers
  - $\sum_j - \prod_i (F_{ji}x + g_{ji})^{\alpha_{ji}} \le F_0 x + g_0$ for rational $\alpha_{ji} > 0, \sum_i \alpha_{ji} \le 1$

Combinations by . . .

- sum, max, positive multiple

Modeling

Current situation

- Each solver recognizes some elementary forms
- Modeler must convert to these forms

Goal

- Recognize many equivalent forms
- Automatically convert to a canonical form
- Further convert as necessary for each solver

Detection & Transformation

Canonical form

- Linear objective
- Standard cone, rotated cone, linear constraints

Algorithms

- **Detection:** Determine if problem has equivalent form
- **Transformation:** Convert to canonical form
- **Implementation:** Recursive tree walks . . .

How a Solver Interacts with AMPL

User types . . .

```
option solver yrslv;
option yrslv_options "maxiter=10000";
solve;
```

AMPL . . .

Writes `at13151.nl`

Executes `yrslv at13151 -AMPL`

YRSLV "driver" . . .

Reads `at13151.nl`

Gets environment variable `yrslv_options`

Calls YRSLV routines to solve the problem

Writes `at13151.sol`

AMPL . . .

Reads `at13151.sol`

How a Solver Interacts with AMPL

What the Driver Does

Reads `.nl` problem file

- Loads everything into `ASL` data structure
- Copies linear coefficients, bounds, etc. to solver's arrays
- Sets directives indicated by `_options` string

Runs algorithm

- Uses `ASL` data structure to compute
  nonlinear expression values, 1st & 2nd derivatives

Writes `.sol` solution file

- Generates result message
- Writes values of variables
- Writes other solution values, as appropriate

. . . can define new "suffixes" for solver-specific information

AMPL's `.nl` File Format

File contents

Numbers of variables, constraints,
integer variables, nonlinear constraints, *etc*.

Coefficient lists for linear part

Expression tree for nonlinear part
plus sparsity pattern of derivatives

Expression tree nodes

Variables, constants

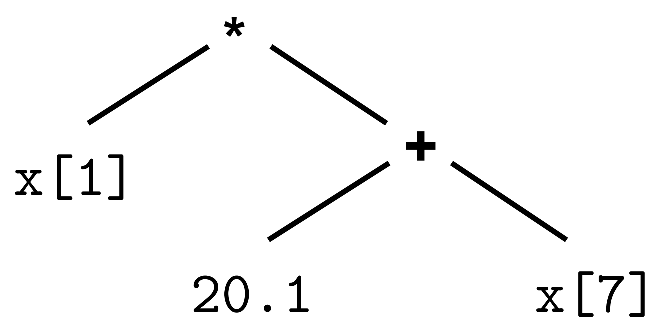
Binary, unary operators

Summations

Function calls

Piecewise-linear terms

If-then-else terms



. . . single array of variables

Example of `.nl` File

Header

```
g3 0 1 0       # problem sum-of-norms3
 14 11 1 0 9   # vars, constraints, objectives, ranges, eqns
 2 0           # nonlinear constraints, objectives
 0 0           # network constraints: nonlinear, linear
 11 0 0        # nonlinear vars in constraints, objectives, both
 0 0 0 1       # linear network variables; functions; arith, flags
 0 0 0 0 0     # discrete variables: binary, integer, nonlinear (b,c,o)
 41 2          # nonzeros in Jacobian, gradients
 0 0           # max name lengths: constraints, variables
 0 0 0 0 0     # common exprs: b,c,o,c1,o1
```

Example of `.nl` File

Expression trees for nonlinear constraints

```
C0    #MaxDefinition[1]
o54   #sumlist
6
o5    #^
v2    #Fxplusg[1,1]
n2
o5    #^
v3    #Fxplusg[1,2]
n2
o5    #^
v4    #Fxplusg[1,3]
n2
o5    #^
v5    #Fxplusg[1,4]
n2
o5    #^
v6    #Fxplusg[1,5]
n2
o16   #-
o5    #^
v0    #Max[1]
n2

C1    #MaxDefinition[2] ...
```

```
subj to MaxDefinition {i in 1..p}:
   sum {k in 1..m[i]} Fxplusg[i,k]^2
      <= Max[i]^2;
```

Detecting Quadratic Functions

Recursive functions needed

```
isQuadratic(e)
isLinear(e)
isConstant(e)
```

Information to be accumulated

- Quadratic terms:
  two variable indices, one coefficient
- Linear terms:
  one variable index, one coefficient
- Constant term

Tree-Walk Procedure

Call target function at root

```
isQuadratic(root)
```

Definition triggers recursive calls on child nodes . . .

Definition of `isQuadratic`

"True" cases for + node

```
isQuadratic (e->L.e) && isQuadratic (e->R.e)
```

"True" cases for × node

```
isQuadratic (e->L.e) && isConstant (e->R.e)
isConstant (e->L.e) && isQuadratic (e->R.e)
isLinear (e->L.e) && isLinear (e->R.e)
```

"True" cases for ^ node

```
isLinear (e->L.e) && isConstant (e->R.e) &&
   e->R.e->val == 2
```

. . . also return appropriate index, coefficient lists

Conic Objectives & Constraints

Apply recursive detection functions

- Sum & max of norms
  - `isSMN(e)`
- Norm squared
  - `isNS(e)`
- . . .

Apply recursive transformation functions . . .

Definition of `isSMN`

`constant` *node*

`variable` *node*

```
True
```

+ *node*

`max` *node*

```
isSMN (e->L.e) && isSMN (e->R.e)
```

× *node*

```
isSMN (e->L.e) && isConstant (e->R.e)
isConstant (e->L.e) && isSMN (e->R.e)
```

`sqrt` *node*

```
isNS (e->L.e)
```

Definition of `isNS`

`constant` *node*

```
True if >= 0
```

+ *node*

`max` *node*

```
isNS (e->L.e) && isNS (e->R.e)
```

× *node*

```
isNS (e->L.e) && isPosConstant (e->R.e)
isPosConstant (e->L.e) && isNS (e->R.e)
```

`^2` *node*

```
isLinear (e->L.e)
```

Detecting Quadratic Functions

Transformation Functions . . .