Modeling and Solving Nontraditional Optimization Problems

Session 2b: Complementarity Conditions

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 2b: Complementarity

Focus

- A condition closely associated with optimization that says two quantities cannot both be positive

Topics

- Motivating examples
- AMPL syntax
- More examples
- Solver strategies

Classical Linear Program

Minimum-cost production

```
set PROD;   # products
set ACT;    # activities

param cost {ACT} > 0;       # cost per unit of each activity
param demand {PROD} >= 0;  # units of demand for each product

param io {PROD,ACT} >= 0;  # units of each product from
                           # 1 unit of each activity

var Level {ACT} >= 0;

minimize TotalCost:
   sum {j in ACT} cost[j] * Level[j];

subject to Demands {i in PROD}:
   sum {j in ACT} io[i,j] * Level[j] >= demand[i];
```

Classical Linear Complementarity

Economic equilibrium

```
set PROD;   # products
set ACT;    # activities

param cost {ACT} > 0;       # cost per unit of each activity
param demand {PROD} >= 0;   # units of demand for each product

param io {PROD,ACT} >= 0;   # units of each product from
                            # 1 unit of each activity

var Price {PROD};
var Level {ACT};

subject to Pri_Compl {i in PROD}:
   Price[i] >= 0 complements
      sum {j in ACT} io[i,j] * Level[j] >= demand[i];

subject to Lev_Compl {j in ACT}:
   Level[j] >= 0 complements
      sum {i in PROD} Price[i] * io[i,j] <= cost[j];
```

. . . complementary slackness conditions for the classical linear program

Complementarity

Bounded-Variable Linear Program

Minimum-cost production

```
set PROD;   # products
set ACT;    # activities

param cost {ACT} > 0;        # cost per unit of each activity
param demand {PROD} >= 0;    # units of demand for each product

param io {PROD,ACT} >= 0;    # units of each product from
                             # 1 unit of each activity

param level_min {ACT} > 0;  # min allowed level for each activity
param level_max {ACT} > 0;  # max allowed level for each activity

var Level {ACT};

minimize TotalCost:
   sum {j in ACT} cost[j] * Level[j];

subject to LevelLimits {j in ACT}:
   level_min[j] <= Level[j] <= level_max[j]

subject to Demands {i in PROD}:
   sum {j in ACT} io[i,j] * Level[j] >= demand[i];
```

Mixed Linear Complementarity

Economic equilibrium with bounded variables

```
set PROD;   # products
set ACT;    # activities

param cost {ACT} > 0;          # cost per unit
param demand {PROD} >= 0;      # units of demand

param io {PROD,ACT} >= 0;      # units of product per unit of activity

param level_min {ACT} > 0;     # min allowed level for each activity
param level_max {ACT} > 0;     # max allowed level for each activity

var Price {PROD};
var Level {ACT};

subject to Pri_Compl {i in PROD}:
   Price[i] >= 0 complements
      sum {j in ACT} io[i,j] * Level[j] >= demand[i];

subject to Lev_Compl {j in ACT}:
   level_min[j] <= Level[j] <= level_max[j] complements
      cost[j] - sum {i in PROD} Price[i] * io[i,j];
```

. . . generalized complementary slackness conditions

Nonlinear Complementarity

Equilibrium with price-dependent demands

```
set PROD;    # products
set ACT;     # activities

param cost {ACT} > 0;         # cost per unit
param demand {PROD} >= 0;    # units of demand

param io {PROD,ACT} >= 0;    # units of product per unit of activity

param demzero {PROD} > 0;    # intercept and slope of the demand
param demrate {PROD} >= 0;   # as a function of price

var Price {PROD};
var Level {ACT};

subject to Pri_Compl {i in PROD}:
   Price[i] >= 0 complements
      sum {j in ACT} io[i,j] * Level[j]
         >= demzero[i] + demrate[i] * Price[i];

subject to Lev_Compl {j in ACT}:
   Level[j] >= 0 complements
      sum {i in PROD} Price[i] * io[i,j] <= cost[j];
```

. . . not equivalent to a linear program

AMPL's `complements` operator

Two single inequalities

single-ineq1 **`complements`** *single-ineq2*

Both inequalities must hold, at least one at equality

One double inequality

double-ineq **`complements`** *expr*
expr **`complements`** *double-ineq*

The double-inequality must hold, and

if at lower limit then *expr* ≥ 0,
if at upper limit then *expr* ≤ 0,
if between limits then *expr* $= 0$

One equality

equality **`complements`** *expr*
expr **`complements`** *equality*

The equality must hold *(included for completeness)*

Logic vs. New Operator

Using complements operator

```
subject to delct {cr in creg, u in users}:
  0 <= ct[cr,u] complements
    ctcost[cr,u] + cv[cr] >= p["C",u];
```

Using logic operators

```
subject to delct {cr in creg, u in users}:
  0 <= ct[cr,u] and
  ctcost[cr,u] + cv[cr] >= p["C",u] and
  (0 = ct[cr,u] or ctcost[cr,u] + cv[cr] = p["C",u]);
```

. . . it's a tradeoff, of course

Examples *(cont'd)*

Prices of coal shipments

```
subject to delct {cr in creg, u in users}:

  0 <= ct[cr,u] complements

    ctcost[cr,u] + cv[cr] >= p["C",u];
```

Height of membrane

```
subject to dv {i in 1..M, j in 1..N}:

  lb[i,j] <= v[i,j] <= ub[i,j] complements

     (dy/dx) * (2*v[i,j] - v[i+1,j] - v[i-1,j])
   + (dx/dy) * (2*v[i,j] - v[i,j+1] - v[i,j-1])
   - c * dx * dy ;
```

. . . ***more at Complementarity Problem Net***
`http://www.cs.wisc.edu/cpnet/`

Optimization Examples (MPECs)

Cournot Nash equilibrium (gnash1m)

```
g1: 0 <= y[1] <= L  complements   l[1];
g3: 0 <= y[2] <= L  complements   l[2];
g5: 0 <= y[3] <= L  complements   l[3];
g7: 0 <= y[4] <= L  complements   l[4];
```

Min area packaging membrane (pac-comp1)

```
obst {i in int_nodes}:

   0 <= s1[i]
       complements
            u[i] - xi[i] - c*(l[i] - Au[i]) >= 0;
```

. . . more at Mac MPEC
www-unix.mcs.anl.gov/~leyffer/MacMPEC/

Solving

"Square" systems

- # of variables =
  # of complementarity constraints +
  # of equality constraints
- Transformation to a simpler canonical form required

MPECs (or MPCCs)

- Mathematical programs with equilibrium constraints
  (or . . . with complementarity constraints)
- No restriction on numbers of variables & constraints
- Objective functions permitted

. . . ***solvers continuing to emerge***

Solving
MPECs

Square systems: well understood

- PATH is a well-known solver

Theory of MPECs: terrible

- Easily convert to smooth functions, but . . .
- Constraint qualifications are violated at every feasible point

Practice with MPECs: promising

- ***Convert*** to an equivalent smooth problem
- Apply a standard method for nonlinearly constrained nonlinear optimization

. . . ***choice of conversion depends on type of solver***

Solving

Representative MPEC Conversions

$$\boxed{0 \le z_1 \perp z_2 \ge 0}$$

$$\boxed{\begin{array}{l} z_1 \ge 0 \\ z_2 \ge 0 \\ z_1^T z_2 \le 0 \end{array}}$$

$$\boxed{\sqrt{(z_{1i} - z_{2i})^2 + 4\mu} - z_{1i} - z_{2i} = 0}$$

$$\boxed{\sqrt{z_{1i}^2 + z_{2i}^2 + \mu} - z_{1i} - z_{2i} = 0}$$

Solving

Representative MPEC Conversions

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & c_i(x) = 0, \quad i \in \mathcal{E} \\ & c_i(x) \geq 0, \quad i \in \mathcal{I} \\ & 0 \leq x_1 \perp x_2 \geq 0. \end{array}$$

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & c_i(x) = 0, \quad i \in \mathcal{E} \\ & c_i(x) \geq 0, \quad i \in \mathcal{I} \\ & x_1 \geq 0, \quad x_2 \geq 0 \\ & x_{1i}x_{2i} \leq 0 \;\; i = 1, \ldots, p. \end{array}$$

Alternative penalty formulation

$$\begin{array}{ll} \text{minimize} & f(x) + \pi x_1^T x_2 \\ \text{subject to} & c_i(x) = 0, \quad i \in \mathcal{E} \\ & c_i(x) \geq 0, \quad i \in \mathcal{I} \\ & x_1 \geq 0, \; x_2 \geq 0, \end{array}$$

Carrying Out the MPEC Conversions

Same theme as before

- **Detection:** Look for `complements` operators in constraints
- **Transformation:** Convert each such constraint
  - different for each solver
- **Implementation:** Recursive tree walks . . .