Computational Optimization for Economists

Sven Leyffer, Jorge Moré, and Todd Munson
Mathematics and Computer Division
Argonne National Laboratory
{leyffer,more,tmunson}@mcs.anl.gov

Institute for Computational Economics
Argonne National Laboratory
July 17-21, 2006

Computational Optimization Overview

1. Introducion to Optimization [Moré]
2. Continuous Optimization in AMPL [Munson]
3. Optimization Software [Leyffer]
4. Complementarity & Games [Munson]

Part I

Introduction, Applications, and Formulations

Outline

- Software
  - Views of optimization
  - Characteristic of optimization software
  - Case studies in optimization software
- Environments
  - Modeling Languages: AMPL, GAMS
  - Solving optimization problems
  - Automatic differentiation
- Tools
  - Benchmarking
  - Performance profiles
  - Scale invariance

Nonlinearly Constrained Optimization

$$\min \{ f(x) : x_l \le x \le x_u,\ c_l \le c(x) \le c_u \}$$

- Objective function is defined by $f : \mathbb{R}^n \mapsto \mathbb{R}$
- Constraints are defined by $c : \mathbb{R}^n \mapsto \mathbb{R}^m$.
- Bounds $x_l \le x \le x_u$ on the variables $x \in \mathbb{R}^n$.

- First-order algorithms require the gradient

$$\nabla f(x) = (\partial_i f(x)), \qquad \nabla c_1(x), \ldots, \nabla c_m(x)$$

- Second order algorithms require the Hessians

$$\nabla^2 f(x) = (\partial_{i,j} f(x)), \qquad \nabla^2 c_1(x), \ldots, \nabla^2 c_m(x)$$

Part II

Continuous Optimization in AMPL

Modeling Languages

- Portable language for optimization problems
  - Algebraic description
  - Models easily modified and solved
  - Large problems can be processed
  - Programming language features
- Many available optimization algorithms
  - No need to compile C/FORTRAN code
  - Derivatives automatically calculated
  - Algorithms specific options can be set
- Communication with other tools
  - Relational databases and spreadsheets
  - MATLAB interface for function evaluations
- Excellent documentation
- Large user communities

Model Declaration

- Sets
  - Unordered, ordered, and circular sets
  - Cross products and point to set mappings
  - Set manipulation
- Parameters and variables
  - Initial and default values
  - Lower and upper bounds
  - Check statements
  - Defined variables
- Objective function and constraints
  - Equality, inequality, and range constraints
  - Complementarity constraints
  - Multiple objectives
- Problem statement

Data and Commands

- Data declaration
  - Set definitions
    - Explicit list of elements
    - Implicit list in parameter statements
  - Parameter definitions
    - Tables and transposed tables
    - Higher dimensional parameters
- Execution commands
  - Load model and data
  - Select problem, algorithm, and options
  - Solve the instance
  - Output results
- Other operations
  - Let and fix statements
  - Conditionals and loop constructs
  - Execution of external programs

Model Formulation

- Economy with n agents and m commodities
  - $e \in \Re^{n \times m}$ are the endowments
  - $\alpha \in \Re^{n \times m}$ and $\beta \in \Re^{n \times m}$ are the utility parameters
  - $\lambda \in \Re^{n}$ are the social weights
- Social planning problem

$$
\begin{array}{lll}
\max\limits_{x \geq 0} & \displaystyle\sum_{i=1}^{n} \lambda_i \left(\sum_{k=1}^{m} \frac{\alpha_{i,k}(1 + x_{i,k})^{1-\beta_{i,k}}}{1 - \beta_{i,k}} \right) & \\
\text{subject to} & \displaystyle\sum_{i=1}^{n} x_{i,k} \leq \sum_{i=1}^{n} e_{i,k} & \forall k = 1, \ldots, m
\end{array}
$$

Model: social1.mod

```
param n > 0, integer;                      # Agents
param m > 0, integer;                      # Commodities

param e {1..n, 1..m} >= 0, default 1; # Endowment

param lambda {1..n} > 0;                   # Social weights
param alpha {1..n, 1..m} > 0;              # Utility parameters
param beta {1..n, 1..m} > 0;


var x{1..n, 1..m} >= 0;                    # Consumption
var u{i in 1..n} =                         # Utility
  sum {k in 1..m} alpha[i,k] * (1 + x[i,k])^(1 - beta[i,k]) / (1 - beta[i,k]);


maximize welfare:
    sum {i in 1..n} lambda[i] * u[i];

subject to
  consumption {k in 1..m}:
    sum {i in 1..n} x[i,k] <= sum {i in 1..n} e[i,k];
```

Data: social1.dat

```
param n := 3;           # Agents
param m := 4;           # Commodities


param alpha : 1    2    3    4 :=
       1      1    1    1    1
       2      1    2    3    4
       3      2    1    1    5;

param beta (tr) : 1    2    3 :=
       1         1.5   2   0.6
       2         1.6   3   0.7
       3         1.7   2   2.0
       4         1.8   2   2.5;

param : lambda :=
    1     1
    2     1
    3     1;
```

Commands: social1.cmd

```
# Load model and data
model social1.mod;
data social1.dat;

# Specify solver and options
option solver "minos";
option minos_options "outlev=1";

# Solve the instance
solve;

# Output results
display x;
printf {i in 1..n} "%2d: % 5.4e\n", i, u[i];
```

Output

```
ampl: include social1.cmd;
MINOS 5.5: outlev=1
MINOS 5.5: optimal solution found.
25 iterations, objective 2.252422003
Nonlin evals: obj = 44, grad = 43.
x :=
1 1   0.0811471
1 2   0.574164
1 3   0.703454
1 4   0.267241
2 1   0.060263
2 2   0.604858
2 3   1.7239
2 4   1.47516
3 1   2.85859
3 2   1.82098
3 3   0.572645
3 4   1.2576
;

 1: -5.2111e+00
 2: -4.0488e+00
 3:  1.1512e+01
ampl: quit;
```

Model: social2.mod

```
set AGENTS;                                     # Agents
set COMMODITIES;                                # Commodities

param e {AGENTS, COMMODITIES} >= 0, default 1; # Endowment

param lambda {AGENTS} > 0;                      # Social weights
param alpha {AGENTS, COMMODITIES} > 0;          # Utility parameters
param beta {AGENTS, COMMODITIES} > 0;

param gamma {i in AGENTS, k in COMMODITIES} := 1 - beta[i,k];

var x{AGENTS, COMMODITIES} >= 0;               # Consumption
var u{i in AGENTS} =                            # Utility
  sum {k in COMMODITIES} alpha[i,k] * (1 + x[i,k])^gamma[i,k] / gamma[i,k];

maximize welfare:
    sum {i in AGENTS} lambda[i] * u[i];

subject to
  consumption {k in COMMODITIES}:
    sum {i in AGENTS} x[i,k] <= sum {i in AGENTS} e[i,k];
```

Data: social2.dat

```
set COMMODITIES := Books, Cars, Food, Pens;

param: AGENTS : lambda :=
       Jorge      1
       Sven       1
       Todd       1;


param alpha : Books  Cars  Food  Pens :=
     Jorge      1      1     1     1
     Sven       1      2     3     4
     Todd       2      1     1     5;

param beta (tr): Jorge  Sven  Todd :=
     Books        1.5    2     0.6
     Cars         1.6    3     0.7
     Food         1.7    2     2.0
     Pens         1.8    2     2.5;
```

Commands: social2.cmd

```
# Load model and data
model social2.mod;
data social2.dat;

# Specify solver and options
option solver "minos";
option minos_options "outlev=1";

# Solve the instance
solve;

# Output results
display x;
printf {i in AGENTS} "%5s: % 5.4e\n", i, u[i];
```

Output

```
ampl: include social2.cmd
MINOS 5.5: outlev=1
MINOS 5.5: optimal solution found.
25 iterations, objective 2.252422003
Nonlin evals: obj = 44, grad = 43.
x :=
Jorge Books   0.0811471
Jorge Cars    0.574164
Jorge Food    0.703454
Jorge Pens    0.267241
Sven  Books   0.060263
Sven  Cars    0.604858
Sven  Food    1.7239
Sven  Pens    1.47516
Todd  Books   2.85859
Todd  Cars    1.82098
Todd  Food    0.572645
Todd  Pens    1.2576
;

Jorge: -5.2111e+00
 Sven: -4.0488e+00
 Todd:  1.1512e+01
ampl: quit;
```

Model Formulation

- Route commodities through a network
  - $\mathcal{N}$ is the set of nodes
  - $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of arcs
  - $\mathcal{K}$ is the set of commodities
  - α and β are the congestion parameters
  - b denotes the supply and demand
- Multicommodity network flow problem

$$
\begin{array}{lll}
\displaystyle\max_{x \geq 0, f \geq 0} & \displaystyle\sum_{(i,j)\in\mathcal{A}} \left(\alpha_{i,j} f_{i,j} + \beta_{i,j} f_{i,j}^4\right) & \\
\text{subject to} & \displaystyle\sum_{(i,j)\in\mathcal{A}} x_{i,j,k} \leq \sum_{(j,i)\in\mathcal{A}} x_{j,i,k} + b_{i,k} & \forall i \in \mathcal{N}, k \in \mathcal{K} \\
 & \displaystyle f_{i,j} = \sum_{k\in\mathcal{K}} x_{i,j,k} & \forall (i,j) \in \mathcal{A}
\end{array}
$$

Model: network.mod

```
set NODES;                                   # Nodes in network
set ARCS within NODES cross NODES;           # Arcs in network
set COMMODITIES := 1..3;                     # Commodities


param b {NODES, COMMODITIES} default 0;      # Supply/demand
check {k in COMMODITIES}:                    # Supply exceeds demand
  sum{i in NODES} b[i,k] >= 0;

param alpha{ARCS} >= 0;                      # Linear part
param beta{ARCS} >= 0;                       # Nonlinear part


var x{ARCS, COMMODITIES} >= 0;               # Flow on arcs
var f{(i,j) in ARCS} =                       # Total flow
  sum {k in COMMODITIES} x[i,j,k];


minimize time:
  sum {(i,j) in ARCS} (alpha[i,j]*f[i,j] + beta[i,j]*f[i,j]^4);

subject to
  conserve {i in NODES, k in COMMODITIES}:
    sum {(i,j) in ARCS} x[i,j,k] <= sum{(j,i) in ARCS} x[j,i,k] + b[i,k];
```

Data: network.dat

```
set NODES := 1 2 3 4 5;

param: ARCS : alpha beta =
        1 2     1    0.5
        1 3     1    0.4
        2 3     2    0.7
        2 4     3    0.1
        3 2     1    0.0
        3 4     4    0.5
        4 1     5    0.0
        4 5     2    0.1
        5 2     0    1.0;


let b[1,1] :=  7;     # Node 1, Commodity 1 supply
let b[4,1] := -7;     # Node 4, Commodity 1 demand
let b[2,2] :=  3;     # Node 2, Commodity 2 supply
let b[5,2] := -3;     # Node 5, Commodity 2 demand
let b[3,3] :=  5;     # Node 1, Commodity 3 supply
let b[1,3] := -5;     # Node 4, Commodity 3 demand

fix {i in NODES, k in COMMODITIES: (i,i) in ARCS} x[i,i,k] := 0;
```

Commands: network.cmd

```
# Load model and data
model network.mod;
data network.dat;

# Specify solver and options
option solver "minos";
option minos_options "outlev=1";

# Solve the instance
solve;

# Output results
for {k in COMMODITIES} {
  printf "Commodity: %d\n", k > network.out;
  printf {(i,j) in ARCS: x[i,j,k] > 0} "%d.%d = % 5.4e\n", i, j, x[i,j,k] > network.out;
  printf "\n" > network.out;
}
```

Output

```
ampl: include network.cmd;
MINOS 5.5: outlev=1
MINOS 5.5: optimal solution found.
12 iterations, objective 1505.526478
Nonlin evals: obj = 14, grad = 13.
ampl: quit;
```

Results: network.out

```
Commodity: 1
1.2 =  3.3775e+00
1.3 =  3.6225e+00
2.4 =  6.4649e+00
3.2 =  3.0874e+00
3.4 =  5.3510e-01

Commodity: 2
2.4 =  3.0000e+00
4.5 =  3.0000e+00

Commodity: 3
3.4 =  5.0000e+00
4.1 =  5.0000e+00
```

Initial Coordinate Descent: wardrop0.cmd

```
# Load model and data
model network.mod;
data network.dat;

option solver "minos";
option minos_options "outlev=1";

# Coordinate descent method
fix {(i,j) in ARCS, k in COMMODITIES} x[i,j,k];
drop {i in NODES, k in COMMODITIES} conserve[i,k];

for {iter in 1..100} {
  for {k in COMMODITIES} {
    unfix {(i,j) in ARCS} x[i,j,k];
    restore {i in NODES} conserve[i,k];

    solve;

    fix {(i,j) in ARCS} x[i,j,k];
    drop {i in NODES} conserve[i,k];
  }
}

# Output results
for {k in COMMODITIES} {
  printf "\nCommodity: %d\n", k > network.out;
  printf {(i,j) in ARCS: x[i,j,k] > 0} "%d.%d = % 5.4e\n", i, j, x[i,j,k] > network.out;
}
```

Improved Coordinate Descent: wardrop.mod

```
set NODES;                                      # Nodes in network
set ARCS within NODES cross NODES;              # Arcs in network
set COMMODITIES := 1..3;                        # Commodities

param b {NODES, COMMODITIES} default 0;         # Supply/demand
param alpha {ARCS} >= 0;                        # Linear part
param beta {ARCS} >= 0;                         # Nonlinear part

var x {ARCS, COMMODITIES} >= 0;                 # Flow on arcs
var f {(i,j) in ARCS} =                         # Total flow
  sum {k in COMMODITIES} x[i,j,k];

minimize time {k in COMMODITIES}:
  sum {(i,j) in ARCS} (alpha[i,j]*f[i,j] + beta[i,j]*f[i,j]^4);

subject to
  conserve {i in NODES, k in COMMODITIES}:
    sum {(i,j) in ARCS} x[i,j,k] <= sum{(j,i) in ARCS} x[j,i,k] + b[i,k];

problem subprob {k in COMMODITIES}: time[k], {i in NODES} conserve[i,k],
                                    {(i,j) in ARCS} x[i,j,k], f;
```

Improved Coordinate Descent: wardrop1.cmd

```
# Load model and data
model wardrop.mod;
data wardrop.dat;

# Specify solver and options
option solver "minos";
option minos_options "outlev=1";

# Coordinate descent method
for {iter in 1..100} {
  for {k in COMMODITIES} {
    solve subprob[k];
  }
}

for {k in COMMODITIES} {
  printf "Commodity: %d\n", k > wardrop.out;
  printf {(i,j) in ARCS: x[i,j,k] > 0} "%d.%d = % 5.4e\n", i, j, x[i,j,k] > wardrop.out;
  printf "\n" > wardrop.out;
}
```

Final Coordinate Descent: wardrop2.cmd

```
# Load model and data
model wardrop.mod;
data wardrop.dat;

# Specify solver and options
option solver "minos";
option minos_options "outlev=1";

# Coordinate descent method
param xold{ARCS, COMMODITIES};
param xnew{ARCS, COMMODITIES};

repeat {
  for {k in COMMODITIES} {
    problem subprob[k];
    let {(i,j) in ARCS} xold[i,j,k] := x[i,j,k];
    solve;
    let {(i,j) in ARCS} xnew[i,j,k] := x[i,j,k];
  }
} until (sum {(i,j) in ARCS, k in COMMODITIES} abs(xold[i,j,k] - xnew[i,j,k]) <= 1e-6);

for {k in COMMODITIES} {
  printf "Commodity: %d\n", k > wardrop.out;
  printf {(i,j) in ARCS: x[i,j,k] > 0} "%d.%d = % 5.4e\n", i, j, x[i,j,k] > wardrop.out;
  printf "\n" > wardrop.out;
}
```

Ordered Sets

```
param V, integer;                              # Number of vertices
param E, integer;                              # Number of elements

set VERTICES := {1..V};                        # Vertex indices
set ELEMENTS := {1..E};                        # Element indices
set COORDS   := {1..3} ordered;                # Spatial coordinates

param T{ELEMENTS, 1..4} in VERTICES;           # Tetrahedral elements

var x{VERTICES, COORDS};                       # Position of vertices

var norm{e in ELEMENTS} = sum{i in COORDS, j in 1..4}
  (x[T[e,j], i] - x[T[e,1], i])^2;

var area{e in ELEMENTS} = sum{i in COORDS}
  (x[T[e,2], i] - x[T[e,1], i]) *
    ((x[T[e,3], nextw(i)] - x[T[e,1], nextw(i)]) *
     (x[T[e,4], prevw(i)] - x[T[e,1], prevw(i)]) -
     (x[T[e,3], prevw(i)] - x[T[e,1], prevw(i)]) *
     (x[T[e,4], nextw(i)] - x[T[e,1], nextw(i)]));

minimize f: sum {e in ELEMENTS} norm[e] / max(area[e], 0) ^ (2 / 3);
```

Circular Sets

```
param V, integer;                              # Number of vertices
param E, integer;                              # Number of elements

set VERTICES := {1..V};                        # Vertex indices
set ELEMENTS := {1..E};                        # Element indices
set COORDS   := {1..3} circular;               # Spatial coordinates

param T{ELEMENTS, 1..4} in VERTICES;           # Tetrahedral elements

var x{VERTICES, COORDS};                       # Position of vertices

var norm{e in ELEMENTS} = sum{i in COORDS, j in 1..4}
  (x[T[e,j], i] - x[T[e,1], i])^2;

var area{e in ELEMENTS} = sum{i in COORDS}
  (x[T[e,2], i] - x[T[e,1], i]) *
    ((x[T[e,3], next(i)] - x[T[e,1], next(i)]) *
     (x[T[e,4], prev(i)] - x[T[e,1], prev(i)]) -
     (x[T[e,3], prev(i)] - x[T[e,1], prev(i)]) *
     (x[T[e,4], next(i)] - x[T[e,1], next(i)]));

minimize f: sum {e in ELEMENTS} norm[e] / max(area[e], 0) ^ (2 / 3);
```

Part III

Optimization Software

Generic Nonlinear Optimization Problem

Nonlinear Programming (NLP) problem

$$\left\{ \begin{array}{lll} \underset{x}{\text{minimize}} & f(x) & \text{objective} \\ \text{subject to} & c(x) = 0 & \text{constraints} \\ & x \geq 0 & \text{variables} \end{array} \right.$$

- $f : R^n \to R$, $c : R^n \to R^m$ smooth (typically $\mathcal{C}^2$)
- $x \in R^n$ finite dimensional (may be large)
- more general $l \leq c(x) \leq u$ possible

Optimality Conditions for NLP

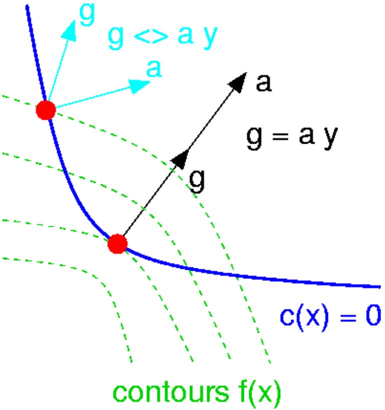
Constraint qualification (CQ)
Linearizations of $c(x) = 0$ characterize all feasible perturbations
$\Rightarrow$ rules out cusps etc.

x^* local minimizer & CQ holds $\Rightarrow \exists$ multipliers y^*, z^*:

$$\begin{aligned} \nabla f(x^*) - \nabla c(x^*)^T y^* - z^* &= 0 \\ c(x^*) &= 0 \\ X^* z^* &= 0 \\ x^* \geq 0,\ z^* &\geq 0 \end{aligned}$$

where $X^* = \text{diag}(x^*)$, thus $X^* z^* = 0 \ \Leftrightarrow\ x_i^* z_i^* = 0$
Lagrangian: $\mathcal{L}(x, y, z) := f(x) - y^T c(x) - z^T x$

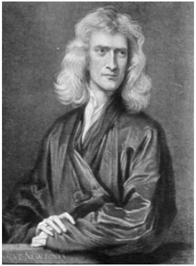
Optimality Conditions for NLP



Objective gradient is linear combination of constraint gradients

$$g(x) = A(x)y, \qquad \text{where } g(x) := \nabla f(x),\ A(x) := \nabla c(x)^T$$

Newton's Method for Nonlinear Equations



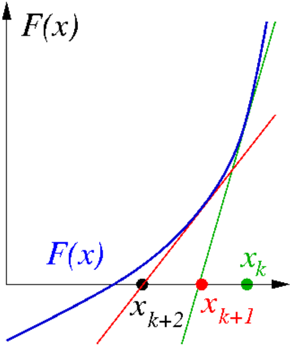
Solve $F(x) = 0$:
Get approx. x_{k+1} of solution of $F(x) = 0$
by solving linear model about x_k:

$$F(x_k) + \nabla F(x_k)^T (x - x_k) = 0$$

for $k = 0, 1, \ldots$

Theorem: If $F \in \mathcal{C}^2$, and $\nabla F(x^*)$ nonsingular,
then Newton converges quadratically near x^*.

Newton's Method for Nonlinear Equations



Next: two classes of methods based on Newton ...

Sequential Quadratic Programming (SQP)

Consider equality constrained NLP

$$\underset{x}{\text{minimize}} \; f(x) \quad \text{subject to } c(x) = 0$$

Optimality conditions:

$$\begin{array}{rcl} \nabla f(x) - \nabla c(x)^T y & = & 0 \quad \text{and} \\ c(x) & = & 0 \end{array}$$

... system of nonlinear equations: $F(w) = 0$ for $w = (x, y)$.

... solve using Newton's method

Sequential Quadratic Programming (SQP)

Nonlinear system of equations (KKT conditions)

$$\nabla f(x) - \nabla c(x)^T y = 0 \quad \text{and} \quad c(x) = 0$$

Apply Newton's method from $w_k = (x_k, y_k)$ ... $H_k = \nabla^2 \mathcal{L}(x_k, y_k)$

$$\begin{bmatrix} H_k & -A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} s_x \\ s_y \end{pmatrix} = - \begin{pmatrix} \nabla_x \mathcal{L}(x_k, y_k) \\ c_k \end{pmatrix}$$

... set $(x_{k+1}, y_{k+1}) = (x_k + s_x, y_k + s_y)$ ... $A^k = \nabla c(x_k)^T$
... solve for $y_{k+1} = y_k + s_y$ directly instead:

$$\begin{bmatrix} H_k & -A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} s \\ y_{k+1} \end{pmatrix} = - \begin{pmatrix} \nabla f_k \\ c_k \end{pmatrix}$$

... set $(x_{k+1}, y_{k+1}) = (x_k + s, y_{k+1})$

Sequential Quadratic Programming (SQP)

Newton's Method for KKT conditions leads to:

$$\begin{bmatrix} H_k & -A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} s \\ y_{k+1} \end{pmatrix} = - \begin{pmatrix} \nabla f_k \\ c_k \end{pmatrix}$$

... are optimality conditions of QP

$$\begin{cases} \underset{s}{\text{minimize}} & \nabla f_k^T s + \frac{1}{2} s^T H_k s \\ \text{subject to} & c_k + A_k^T s = 0 \end{cases}$$

... hence Sequential Quadratic Programming

Sequential Quadratic Programming (SQP)

SQP for inequality constrained NLP:

$$\underset{x}{\text{minimize}} \; f(x) \quad \text{subject to } c(x) = 0 \quad \& \quad x \geq 0$$

REPEAT

1. Solve QP for (s, y_{k+1}, z_{k+1})

$$\left\{ \begin{array}{ll} \underset{s}{\text{minimize}} & \nabla f_k^T s + \frac{1}{2} s^T H_k s \\ \text{subject to} & c_k + A_k^T s = 0 \\ & x_k + s \geq 0 \end{array} \right.$$

2. Set $x_{k+1} = x_k + s$

Modern Interior Point Methods (IPM)

General NLP

$$\underset{x}{\text{minimize}} \; f(x) \quad \text{subject to } c(x) = 0 \quad \& \quad x \geq 0$$

Perturbed $\mu > 0$ optimality conditions ($x, z > 0$)

$$F_\mu(x, y, z) = \left\{ \begin{array}{r} \nabla f(x) - \nabla c(x)^T y - z \\ c(x) \\ Xz - \mu e \end{array} \right\} = 0$$

- Primal-dual formulation, where $X = \text{diag}(x)$
- Central path $\{x(\mu), y(\mu), z(\mu) \; : \; \mu > 0\}$
- Apply Newton's method for sequence $\mu \searrow 0$

Modern Interior Point Methods (IPM)

Newton's method applied to primal-dual system ...

$$\begin{bmatrix} \nabla^2 \mathcal{L}_k & -A_k & -I \\ A_k^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = -F_\mu(x_k, y_k, z_k)$$

where $A_k = \nabla c(x_k)^T$, X_k diagonal matrix of x_k.

Polynomial run-time guarantee for convex problems

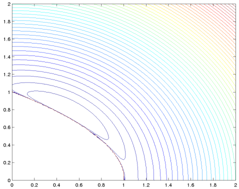
Classical Interior Point Methods (IPM)

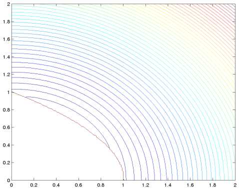
$$\underset{x}{\text{minimize}} \; f(x) \quad \text{subject to } c(x) = 0 \quad \& \quad x \geq 0$$

Related to classical barrier methods [Fiacco & McCormick]

$$\left\{ \begin{array}{ll} \underset{x}{\text{minimize}} & f(x) - \mu \sum \log(x_i) \\ \text{subject to} & c(x) = 0 \end{array} \right.$$

$\mu = 0.1$ $\mu = 0.001$





$$\text{minimize } x_1^2 + x_2^2 \quad \text{subject to} \quad x_1 + x_2^2 \geq 1$$

Classical Interior Point Methods (IPM)

$$\underset{x}{\text{minimize}} \;\; f(x) \quad \text{subject to } c(x) = 0 \quad \& \quad x \geq 0$$

Relationship to barrier methods

$$\begin{cases} \underset{x}{\text{minimize}} & f(x) - \mu \sum \log(x_i) \\ \text{subject to} & c(x) = 0 \end{cases}$$

First order conditions

$$\begin{aligned} \nabla f(x) - \mu X^{-1} e - A(x) y &= 0 \\ c(x) &= 0 \end{aligned}$$

... apply Newton's method ...

Classical Interior Point Methods (IPM)

Newton's method for barrier problem from x_k ...

$$\begin{bmatrix} \nabla^2 \mathcal{L}_k + \mu X_k^{-2} & -A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \ldots$$

Introduce $Z(x_k) := \mu X_k^{-1}$ ... or ... $Z(x_k) X_k = \mu e$

$$\begin{bmatrix} \nabla^2 \mathcal{L}_k + Z(x_k) X_k^{-1} & -A_k \\ A_k & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \ldots$$

... compare to primal-dual system ...

Classical Interior Point Methods (IPM)

Recall: Newton's method applied to primal-dual system ...

$$\begin{bmatrix} \nabla^2 \mathcal{L}_k & -A_k & -I \\ A_k^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = -F_\mu(x_k, y_k, z_k)$$

Eliminate $\Delta z = -X^{-1} Z \Delta x - Ze - \mu X^{-1} e$

$$\begin{bmatrix} \nabla^2 \mathcal{L}_k + Z_k X_k^{-1} & -A_k \\ A_k & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \ldots$$

Interior Point Methods (IPM)

Primal-dual system ...

$$\begin{bmatrix} \nabla^2 \mathcal{L}_k + Z_k X_k^{-1} & -A_k \\ A_k & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \ldots$$

... compare to barrier system ...

$$\begin{bmatrix} \nabla^2 \mathcal{L}_k + Z(x_k) X_k^{-1} & -A_k \\ A_k & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \ldots$$

- Z_k is free, not $Z(x_k) = \mu X_k^{-1}$ (primal multiplier)
- avoid difficulties with barrier ill-conditioning

Convergence from Remote Starting Points

$$\underset{x}{\text{minimize}} \; f(x) \quad \text{subject to } c(x) = 0 \quad \& \quad x \geq 0$$

- Newton's method converges quadratically near a solution
- Newton's method may diverge if started far from solution
- How can we safeguard against this failure?

... motivates penalty or merit functions that

1. monitor progress towards a solution
2. combine objective $f(x)$ and constraint violation $\|c(x)\|$

Penalty Functions (i)

Augmented Lagrangian Methods

$$\underset{x}{\text{minimize}}\ L(x, y_k, \rho_k) \ = \ f(x) \ - \ y_k^T c(x) + \frac{1}{2}\rho_k \|c(x)\|^2$$

As $y_k \rightarrow y_*$:
- $x_k \rightarrow x_*$ for $\rho_k > \bar{\rho}$
- No ill-conditioning, improves convergence rate

- update ρ_k based on reduction in $\|c(x)\|^2$
- approx. minimize $L(x, y_k, \rho_k)$
- first-order multiplier update: $y_{k+1} = y_k - \rho_k c(x_k)$
  $\Rightarrow$ dual iteration

Penalty Functions (ii)

Exact Penalty Function

$$\underset{x}{\text{minimize}} \ \Phi(x, \pi) \ = \ f(x) + \pi \|c(x)\|$$

- combine constraints ad objective
- equivalence of optimality $\Rightarrow$ exact for $\pi > \|y^*\|_D$
  ... now apply unconstrained techniques
- Φ nonsmooth, but equivalent to smooth problem (exercise)

... how do we enforce descent in merit functions???

Line Search Methods

SQP/IPM compute s descend direction: $s^T \nabla \Phi < 0$

Backtracking-Armijo line search

Given $\alpha^0 = 1$, $\beta = 0.1$, set $l = 0$

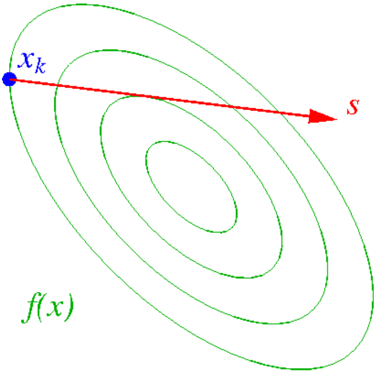
REPEAT

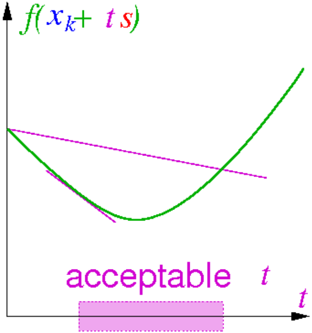
1. $\alpha^{l+1} = \alpha^l / 2$ & evaluate $\Phi(x + \alpha^{l+1} s)$
2. $l = l + 1$

UNTIL $\Phi(x + \alpha^l s) \leq f(x) + \alpha^l \beta s^T \nabla \Phi$

Converges to stationary point, or unbounded, or zero descend

Line Search Methods

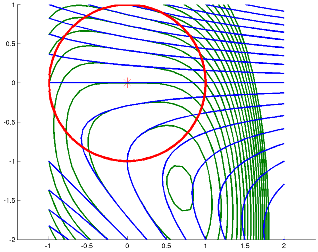


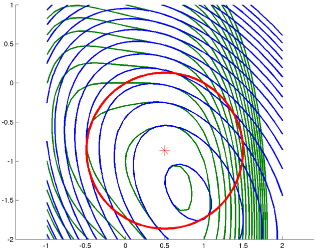


Trust Region Methods

Globalize SQP (IPM) by adding trust region, $\Delta^k > 0$

$$\begin{cases} \underset{s}{\text{minimize}} & \nabla f_k^T s + \frac{1}{2} s^T H_k s \\ \text{subject to} & c_k + A_k^T s = 0, \quad x_k + s \geq 0, \quad \|s\| \leq \Delta^k \end{cases}$$





Trust Region Methods

Globalize SQP (IPM) by adding trust region, $\Delta^k > 0$

$$\begin{cases} \underset{s}{\text{minimize}} & \nabla f_k^T s + \frac{1}{2} s^T H_k s \\ \text{subject to} & c_k + A_k^T s = 0, \quad x_k + s \geq 0, \quad \|s\| \leq \Delta^k \end{cases}$$

REPEAT

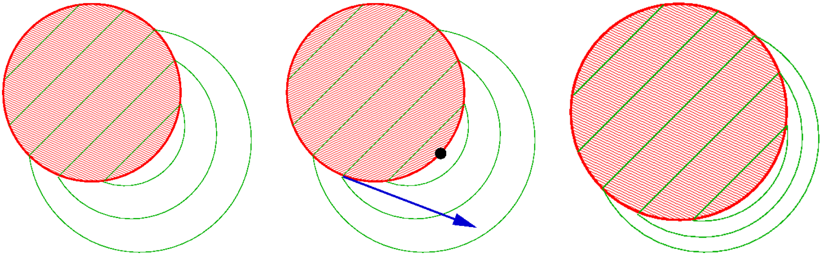
1. Solve QP approximation about x_k
2. Compute actual/predicted reduction, r_k
3. **IF** $r_k \geq 0.75$ **THEN** $x_{k+1} = x_k + s$ increase Δ
   ELSEIF $r_k \geq 0.25$ **THEN** $x_{k+1} = x_k + s$
   ELSE $x_{k+1} = x_k$ & decrease Δ reject step

UNTIL convergence

Filter Methods for NLP

Penalty function can be inefficient

- Penalty parameter not known a priori
- Large penalty parameter $\Rightarrow$ slow convergence



Filter Methods for NLP

Penalty function can be inefficient

- Penalty parameter not known a priori
- Large penalty parameter $\Rightarrow$ slow convergence

Two competing aims in optimization:

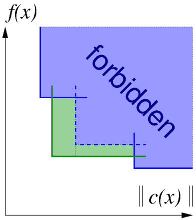
1. Minimize $f(x)$
2. Minimize $h(x) := \|c(x)\|$ ... more important

$\Rightarrow$ concept from multi-objective optimization:
(h_{k+1}, f_{k+1}) dominates (h_l, f_l) iff $h_{k+1} \le h_l$ & $f_{k+1} \le f_l$

Filter Methods for NLP

Filter $\mathcal{F}$: list of non-dominated pairs (h_l, f_l)

- new x_{k+1} acceptable to filter $\mathcal{F}$, iff
    1. $h_{k+1} \leq h_l \; \forall l \in \mathcal{F}$, or
    2. $f_{k+1} \leq f_l \; \forall l \in \mathcal{F}$
- remove redundant entries
- reject new x_{k+1},
  if $h_{k+1} > h_l$ & $f_{k+1} > f_l$
  ... reduce trust region radius $\Delta = \Delta/2$



$\Rightarrow$ often accept new x_{k+1}, even if penalty function increases

Sequential Quadratic Programming

- `filterSQP`
  - trust-region SQP; robust QP solver
  - filter to promote global convergence
- `SNOPT`
  - line-search SQP; null-space CG option
  - ℓ_1 exact penalty function
- `SLIQUE` (part of `KNITRO`)
  - SLP-EQP ("SQP" for larger problems)
  - trust-region with ℓ_1 penalty

Other Methods: `CONOPT` generalized reduced gradient method

Interior Point Methods

- IPOPT (free: part of COIN-OR)
  - line-search filter algorithm
  - 2nd order convergence analysis for filter
- KNITRO
  - trust-region Newton to solve barrier problem
  - ℓ_1 penalty barrier function
  - Newton system: direct solves or null-space CG
- LOQO
  - line-search method
  - Cholesky factorization; no convergence analysis

Other solvers: MOSEK (unsuitable or nonconvex problem)

Augmented Lagrangian Methods

- `LANCELOT`
  - minimize augmented Lagrangian subject to bounds
  - trust-region to force convergence
  - iterative (CG) solves
- `MINOS`
  - minimize augmented Lagrangian subject to linear constraints
  - line-search; recent convergence analysis
  - direct factorization of linear constraints
- `PENNON`
  - suitable for semi-definite optimization
  - alternative penalty terms

Automatic Differentiation

How do I get the derivatives $\nabla c(x)$, $\nabla^2 c(x)$ etc?

- hand-coded derivatives are error prone
- finite differences $\frac{\partial c_i(x)}{\partial x_j} \simeq \frac{c_i(x+\delta e_j)-c_i(x)}{\delta}$ can be dangerous
  where $e_j = (0,\ldots,0,1,0,\ldots,0)$ is j^{th} unit vector

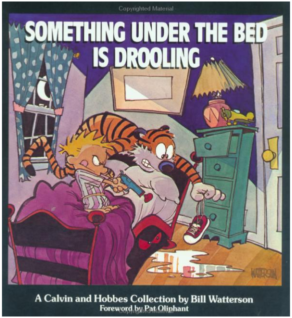
Automatic Differentiation

- chain rule techniques to differentiate program
- recursive application $\Rightarrow$ "exact" derivatives
- suitable for huge problems, see `www.autodiff.org`

... already done for you in AMPL/GAMS etc.

Something Under the Bed is Drooling

1. floating point (IEEE) exceptions
2. unbounded problems
    2.1 unbounded objective
    2.2 unbounded multipliers
3. (locally) inconsistent problems
4. suboptimal solutions



... identify problem & suggest remedies

Floating Point (IEEE) Exceptions

Bad example: minimize barrier function

```
param mu default 1;
var x{1..2} >= -10, <= 10;
var s;
minimize barrier: x[1]^2 + x[2]^2 - mu*log(s);
subject to
   cons: s = x[1] + x[2]^2 - 1;
```

... results in error message like

`Cannot evaluate objective at start`

... change initialization of `s`:

`var s := 1;` ... difficult, if IEEE during solve ...

Unbounded Objective

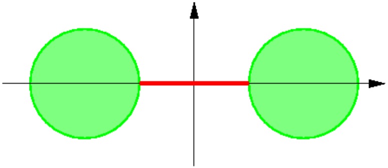
Penalized MPEC $\pi = 1$:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & x_1^2 + x_2^2 - 4x_1x_2 \quad + \pi x_1 x_2 \\ \text{subject to} & x_1, x_2 \geq 0 \end{array}$$

... unbounded below for all $\pi < 2$

Locally Inconsistent Problems

NLP may have no feasible point



feasible set: intersection of circles

Locally Inconsistent Problems

NLP may have no feasible point

```
var x{1..2} >= -1;
minimize objf: -1000*x[2];
subject to
   con1: (x[1]+2)^2 + x[2]^2 <= 1;
   con2: (x[1]-2)^2 + x[2]^2 <= 1;
```

- not all solvers recognize this ...
- finding feasible point $\Leftrightarrow$ global optimization

Locally Inconsistent Problems

LOQO

```
     |          Primal         |           Dual
Iter |   Obj Value      Infeas   |   Obj Value      Infeas
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  1   -1.000000e+03   4.2e+00    -6.000000e+00   1.0e-00
[...]
500    2.312535e-04   7.9e-01     1.715213e+12   1.5e-01
LOQO 6.06: iteration limit
```

... fails to converge ... not useful for user

dual unbounded $\rightarrow \infty \Rightarrow$ primal infeasible

Locally Inconsistent Problems

FILTER

```
iter  |    rho    |   ||d||     |    f / hJ   |   ||c||/hJt
------+-----------+-------------+-------------+------------
   0:0  10.0000    0.00000     -1000.0000    16.000000
   1:1  10.0000    2.00000     -1000.0000    8.0000000
[...]
   8:2  2.00000   0.320001E-02  7.9999693   0.10240052E-04
   9:2  2.00000   0.512000E-05  8.0000000   0.26214586E-10
filterSQP: Nonlinear constraints locally infeasible
```

... fast convergence to minimum infeasibility

... identify “blocking” constraints ... modify model/data

Locally Inconsistent Problems

Remedies for locally infeasible problems:

1. check your model: print constraints & residuals, e.g.
```
solve;
display _conname, _con.lb, _con.body, _con.ub;
display _varname, _var.lb, _var, _var.ub;
```
   ... look at violated and active constraints
2. try different nonlinear solvers (easy with AMPL)
3. build-up model from few constraints at a time
4. try different starting points ... global optimization

Suboptimal Solution & Multi-start

Problems can have many local mimimizers

```
param pi := 3.1416;
param n integer, >= 0, default 2;
set N := 1..n;
var x{N} >= 0, <= 32*pi, := 1;
minimize objf:
- sum{i in N} x[i]*sin(sqrt(x[i]));
```

default start point converges to local minimizer

Suboptimal Solution & Multi-start

```
param nD := 5;       # discretization
set    D := 1..nD;
param  hD := 32*pi/(nD-1);
param optval{D,D};
model schwefel.mod;  # load model

for {i in D}{
   let x[1] := (i-1)*hD;
   for {j in D}{
      let x[2] := (j-1)*hD;
      solve;
      let optval[i,j] := objf;
   }; # end for
}; # end for
```

Suboptimal Solution & Multi-start

```
display optval;
optval [*,*]
:     1         2          3           4           5    :=
1     0        24.003    -36.29     -50.927      56.909
2    24.003    -7.8906   -67.580    -67.580     -67.580
3   -36.29    -67.5803  -127.27    -127.27     -127.27
4   -50.927   -67.5803  -127.27    -127.27     -127.27
5    56.909   -67.5803  -127.27    -127.27     -127.27
;
```

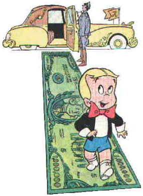
... there exist better multi-start procedures

Optimization with Integer Variables

- modeling discrete choices $\Rightarrow$ $0-1$ variables
- modeling integer decisions $\Rightarrow$ integer variables
  e.g. number of different stocks in portfolio (8-10)
  not number of beers sold at Goose Island (millions)

$\Rightarrow$ Mixed Integer Nonlinear Program (MINLP)
MINLP solvers:

- branch (separate $z_i = 0$ and $z_i = 1$) and cut
- solve millions of NLP relaxations: `MINLPBB`, `SBB`
- outer approximation: iterate MILP and NLP solvers
  `BONMIN` soon on `COIN-OR`



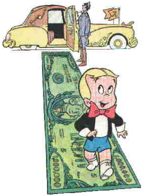
Portfolio Management

- N: Universe of asset to purchase
- x_i: Amount of asset i to hold
- B: Budget

$$\min_{x \in \mathbb{R}_+^{|N|}} \left\{ u(x) \mid \sum_{i \in N} x_i = B \right\}$$

- Markowitz: $u(x) \stackrel{\text{def}}{=} -\alpha^T x + \lambda x^T Q x$
  - α: Expected returns
  - Q: Variance-covariance matrix of expected returns
  - λ: Risk aversion parameter

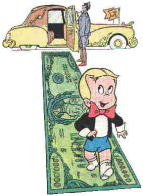
More Realistic Models



- $b \in \mathbb{R}^{|N|}$ of "benchmark" holdings
- Benchmark Tracking: $u(x) \stackrel{\text{def}}{=} (x-b)^T Q(x-b)$
  - Constraint on $\mathbb{E}$[Return]: $\alpha^T x \geq r$
- Limit Names: $|i \in N \ : \ x_i > 0| \leq K$
  - Use binary indicator variables to model the implication $x_i > 0 \Rightarrow y_i = 1$
  - Implication modeled with variable upper bounds:

$$x_i \leq B y_i \qquad \forall i \in N$$

  - $\sum_{i \in N} y_i \leq K$



Even More Models

- Min Holdings: $(x_i = 0) \vee (x_i \geq m)$
  - Model implication: $x_i > 0 \Rightarrow x_i \geq m$
  - $x_i > 0 \Rightarrow y_i = 1 \Rightarrow x_i \geq m$
  - $x_i \leq By_i, x_i \geq my_i \;\forall i \in N$
- Round Lots: $x_i \in \{kL_i, k = 1, 2, \ldots\}$
  - $x_i - z_i L_i = 0, z_i \in \mathbb{Z}_+ \;\forall i \in N$
- Vector h of initial holdings
- Transactions: $t_i = |x_i - h_i|$
- Turnover: $\sum_{i \in N} t_i \leq \Delta$
- Transaction Costs: $\sum_{i \in N} c_i t_i$ in objective
- Market Impact: $\sum_{i \in N} \gamma_i t_i^2$ in objective

Global Optimization

I need to find the GLOBAL minimum!

- use any NLP solver (often work well!)
- use the multi-start trick from previous slides
- global optimization based on branch-and-reduce: `BARON`
  - constructs global underestimators
  - refines region by branching
  - tightens bounds by solving LPs
  - solve problems with 100s of variables
- "voodoo" solvers: genetic algorithm & simulated annealing
  no convergence theory ... usually worse than deterministic

Derivative-Free Optimization

My model does not have derivatives!

- Change your model ... good models have derivatives!
- pattern-search methods for min $f(x)$
  - evaluate $f(x)$ at stencil $x_k + \Delta M$
  - move to new best point
  - extend to NLP; some convergence theory
  - matlab: `NOMADm.m`; parallel `APPSPACK`
- solvers based on building quadratic models
- "voodoo" solvers: genetic algorithm & simulated annealing
  no convergence theory ... usually worse than deterministic

COIN-OR

http://www.coin-or.org

- COmputational INfrastructure for Operations Research
- A library of (interoperable) software tools for optimization
- A development platform for open source projects in the OR community
- Possibly Relevant Modules:
  - OSI: Open Solver Interface
  - CGL: Cut Generation Library
  - CLP: Coin Linear Programming Toolkit
  - CBC: Coin Branch and Cut
  - IPOPT: Interior Point OPTimizer for NLP
  - NLPAPI: NonLinear Programming API

Part IV

Complementarity Problems in AMPL

Definition

- Non-cooperative game played by n individuals
  - Each player selects a strategy to optimize their objective
  - Strategies for the other players are fixed
- Equilibrium reached when no improvement is possible
- Characterization of two player equilibrium (x^*, y^*)

$$x^* \in \left\{ \begin{array}{ll} \displaystyle\operatorname*{arg\,min}_{x \geq 0} & f_1(x, y^*) \\ \text{subject to} & c_1(x) \leq 0 \end{array} \right.$$
$$y^* \in \left\{ \begin{array}{ll} \displaystyle\operatorname*{arg\,min}_{y \geq 0} & f_2(x^*, y) \\ \text{subject to} & c_2(y) \leq 0 \end{array} \right.$$

- Many applications in economics
  - Bi-matrix games
  - Cournot duopoly models
  - General equilibrium models
  - Arrow-Debreau models

Complementarity Formulation

- Assume each optimization problem is convex
  - $f_1(\cdot, y)$ is convex for each y
  - $f_2(x, \cdot)$ is convex for each x
  - $c_1(\cdot)$ and $c_2(\cdot)$ satisfy constraint qualification
- Then the first-order conditions are necessary and sufficient

$$\begin{array}{ll} \min\limits_{x \geq 0} & f_1(x, y^*) \\ \text{subject to} & c_1(x) \leq 0 \end{array} \quad \Leftrightarrow \quad \begin{array}{lcl} 0 \leq x & \perp & \nabla_x f_1(x, y^*) + \lambda_1^T \nabla_x c_1(x) \geq 0 \\ 0 \leq \lambda_1 & \perp & -c_1(x) \geq 0 \end{array}$$

$$\begin{array}{ll} \min\limits_{y \geq 0} & f_2(x^*, y) \\ \text{subject to} & c_2(y) \leq 0 \end{array} \quad \Leftrightarrow \quad \begin{array}{lcl} 0 \leq y & \perp & \nabla_y f_2(x^*, y) + \lambda_2^T \nabla_y c_2(y) \geq 0 \\ 0 \leq \lambda_2 & \perp & -c_2(y) \geq 0 \end{array}$$

Formulation

- Firm $f \in \mathcal{F}$ chooses output x_f to maximize profit
  - u is the utility function

$$u = \left(1 + \sum_{f \in \mathcal{F}} x_f^\alpha \right)^{\frac{\eta}{\alpha}}$$

  - α and η are parameters
  - c_f is the unit cost for each firm
- In particular, for each firm $f \in \mathcal{F}$

$$x_f^* \quad \in \quad \arg\max_{x_f \geq 0} \left(\frac{\partial u}{\partial x_f} - c_f \right) x_f$$

- First-order optimality conditions

$$0 \leq x_f \quad \perp c_f - \frac{\partial u}{\partial x_f} - x_f \frac{\partial^2 u}{\partial x_f^2} \geq 0$$

Model: oligopoly.mod

```
set FIRMS;                                      # Firms in problem

param c {FIRMS};                                # Unit cost
param alpha > 0;                                # Constants
param eta > 0;

var x {FIRMS} default 0.1;                      # Output (no bounds!)

var s = 1 + sum {f in FIRMS} x[f]^alpha;        # Summation term
var u = s^(eta/alpha);                          # Utility
var du {f in FIRMS} =                           # Marginal price
  eta * s^(eta/alpha - 1) * x[f]^(alpha - 1);
var dudu {f in FIRMS} =                         # Derivative
  eta * (eta - alpha) * s^(eta/alpha - 2) * x[f]^(2 * alpha - 2) +
  eta * (alpha - 1  ) * s^(eta/alpha - 1) * x[f]^(    alpha - 2);

compl {f in FIRMS}:
  0 <= x[f] complements c[f] - du[f] - x[f] * dudu[f] >= 0;
```

Data: oligopoly.dat

```
param: FIRMS :  c :=
        1     0.07
        2     0.08
        3     0.09;

param alpha := 0.999;
param eta := 0.2;
```

Commands: oligopoly.cmd

```
# Load model and data
model oligopoly.mod;
data oligopoly.dat;

# Specify solver and options
option presolve 0;
option solver "pathampl";

# Solve complementarity problem
solve;

# Output the results
printf {f in FIRMS} "Output for firm %2d: % 5.4e\n", f, x[f] > oligcomp.out;
```

Results: oligopoly.out

```
Output for firm  1:  8.3735e-01
Output for firm  2:  5.0720e-01
Output for firm  3:  1.7921e-01
```

Model Formulation

- Economy with n agents and m commodities
  - $e \in \Re^{n \times m}$ are the endowments
  - $\alpha \in \Re^{n \times m}$ and $\beta \in \Re^{n \times m}$ are the utility parameters
  - $p \in \Re^{m}$ are the commodity prices
- Agent i maximizes utility with budget constraint

$$\begin{array}{ll} \max\limits_{x_{i,*} \geq 0} & \displaystyle\sum_{k=1}^{m} \frac{\alpha_{i,k}(1 + x_{i,k})^{1-\beta_{i,k}}}{1 - \beta_{i,k}} \\ \text{subject to} & \displaystyle\sum_{k=1}^{m} p_k \left(x_{i,k} - e_{i,k}\right) \leq 0 \end{array}$$

- Market k sets price for the commodity

$$0 \leq p_k \quad \perp \quad \sum_{i=1}^{n} (e_{i,k} - x_{i,k}) \geq 0$$

Model: cge.mod

```
set AGENTS;                                    # Agents
set COMMODITIES;                               # Commodities

param e {AGENTS, COMMODITIES} >= 0, default 1; # Endowment

param alpha {AGENTS, COMMODITIES} > 0;         # Utility parameters
param beta {AGENTS, COMMODITIES} > 0;

var x {AGENTS, COMMODITIES};                   # Consumption (no bounds!)
var l {AGENTS};                                # Multipliers (no bounds!)
var p {COMMODITIES};                           # Prices (no bounds!)

var du {i in AGENTS, k in COMMODITIES} =       # Marginal prices
  alpha[i,k] / (1 + x[i,k])^beta[i,k];

subject to
  optimality {i in AGENTS, k in COMMODITIES}:
    0 <= x[i,k] complements -du[i,k] + p[k] * l[i] >= 0;

  budget {i in AGENTS}:
    0 <= l[i]   complements sum {k in COMMODITIES} p[k]*(e[i,k] - x[i,k]) >= 0;

  market {k in COMMODITIES}:
    0 <= p[k]   complements sum {i in AGENTS} (e[i,k] - x[i,k]) >= 0;
```

Data: cge.dat

```
set AGENTS := Jorge, Sven, Todd;
set COMMODITIES := Books, Cars, Food, Pens;

param alpha : Books  Cars  Food  Pens :=
     Jorge      1     1     1     1
     Sven       1     2     3     4
     Todd       2     1     1     5;

param beta (tr): Jorge  Sven  Todd :=
     Books         1.5   2     0.6
     Cars          1.6   3     0.7
     Food          1.7   2     2.0
     Pens          1.8   2     2.5;
```

Commands: cge.cmd

```
# Load model and data
model cge.mod;
data cge.dat;

# Specify solver and options
option presolve 0;
option solver "pathampl";

# Solve the instance
solve;

# Output results
printf {i in AGENTS, k in COMMODITIES} "%5s %5s: % 5.4e\n", i, k, x[i,k] > cge.out;
printf "\n" > cge.out;
printf {k in COMMODITIES} "%5s: % 5.4e\n", k, p[k] > cge.out;
```

Results: cge.out

```
Jorge Books:  8.9825e-01
Jorge  Cars:  1.4651e+00
Jorge  Food:  1.2021e+00
Jorge  Pens:  6.8392e-01
 Sven Books:  2.5392e-01
 Sven  Cars:  7.2054e-01
 Sven  Food:  1.6271e+00
 Sven  Pens:  1.4787e+00
 Todd Books:  1.8478e+00
 Todd  Cars:  8.1431e-01
 Todd  Food:  1.7081e-01
 Todd  Pens:  8.3738e-01

Books:  1.0825e+01
 Cars:  6.6835e+00
 Food:  7.3983e+00
 Pens:  1.1081e+01
```

Commands: cgenum.cmd

```
# Load model and data
model cge.mod;
data cge.dat;

# Specify solver and options
option presolve 0;
option solver "pathampl";

# Solve the instance
drop market['Books'];
fix p['Books'] := 1;
solve;

# Output results
printf {i in AGENTS, k in COMMODITIES} "%5s %5s: % 5.4e\n", i, k, x[i,k] > cgenum.out;
printf "\n" > cgenum.out;
printf {k in COMMODITIES} "%5s: % 5.4e\n", k, p[k] > cgenum.out;
```

Results: cgenum.out

```
Jorge Books:  8.9825e-01
Jorge  Cars:  1.4651e+00
Jorge  Food:  1.2021e+00
Jorge  Pens:  6.8392e-01
 Sven Books:  2.5392e-01
 Sven  Cars:  7.2054e-01
 Sven  Food:  1.6271e+00
 Sven  Pens:  1.4787e+00
 Todd Books:  1.8478e+00
 Todd  Cars:  8.1431e-01
 Todd  Food:  1.7081e-01
 Todd  Pens:  8.3738e-01

Books:  1.0000e+00
 Cars:  6.1742e-01
 Food:  6.8345e-01
 Pens:  1.0237e+00
```

Formulation

- Players select strategies to minimize loss
  - $p \in \Re^n$ is the probability player 1 chooses each strategy
  - $q \in \Re^m$ is the probability player 2 chooses each strategy
  - $A \in \Re^{n \times m}$ is the loss matrix for player 1
  - $B \in \Re^{n \times m}$ is the loss matrix for player 2
- Optimization problem for player 1

$$\begin{array}{ll} \min\limits_{0 \le p \le 1} & p^T A q \\ \text{subject to} & e^T p = 1 \end{array}$$

- Optimization problem for player 2

$$\begin{array}{ll} \min\limits_{0 \le q \le 1} & p^T B q \\ \text{subject to} & e^T q = 1 \end{array}$$

- Complementarity problem

$$0 \le p \le 1 \quad \perp \quad Aq - \lambda_1$$

Model: bimatrix1.mod

```
param n > 0, integer;          # Strategies for player 1
param m > 0, integer;          # Strategies for player 2

param A{1..n, 1..m};           # Loss matrix for player 1
param B{1..n, 1..m};           # Loss matrix for player 2

var p{1..n};                   # Probability player 1 selects strategy i
var q{1..m};                   # Probability player 2 selects strategy j
var lambda1;                   # Multiplier for constraint
var lambda2;                   # Multiplier for constraint

subject to
  opt1 {i in 1..n}:            # Optimality conditions for player 1
    0 <= p[i] <= 1 complements sum{j in 1..m} A[i,j] * q[j] - lambda1;

  opt2 {j in 1..m}:            # Optimality conditions for player 2
    0 <= q[j] <= 1 complements sum{i in 1..n} B[i,j] * p[i] - lambda2;

  con1:
    lambda1 complements sum{i in 1..n} p[i] = 1;

  con2:
    lambda2 complements sum{j in 1..m} q[j] = 1;
```

Model: bimatrix2.mod

```
param n > 0, integer;           # Strategies for player 1
param m > 0, integer;           # Strategies for player 2

param A{1..n, 1..m};            # Loss matrix for player 1
param B{1..n, 1..m};            # Loss matrix for player 2

var p{1..n};                    # Probability player 1 selects strategy i
var q{1..m};                    # Probability player 2 selects strategy j
var lambda1;                    # Multiplier for constraint
var lambda2;                    # Multiplier for constraint

subject to
  opt1 {i in 1..n}:             # Optimality conditions for player 1
    0 <= p[i] complements sum{j in 1..m} A[i,j] * q[j] - lambda1 >= 0;

  opt2 {j in 1..m}:             # Optimality conditions for player 2
    0 <= q[j] complements sum{i in 1..n} B[i,j] * p[i] - lambda2 >= 0;

  con1:
    0 <= lambda1 complements sum{i in 1..n} p[i] >= 1;

  con2:
    0 <= lambda2 complements sum{j in 1..m} q[j] >= 1;
```

Model: bimatrix3.mod

```
param n > 0, integer;           # Strategies for player 1
param m > 0, integer;           # Strategies for player 2

param A{1..n, 1..m};            # Loss matrix for player 1
param B{1..n, 1..m};            # Loss matrix for player 2

var p{1..n};                    # Probability player 1 selects strategy i
var q{1..m};                    # Probability player 2 selects strategy j

subject to
  opt1 {i in 1..n}:             # Optimality conditions for player 1
    0 <= p[i] complements sum{j in 1..m} A[i,j] * q[j] >= 1;

  opt2 {j in 1..m}:             # Optimality conditions for player 2
    0 <= q[j] complements sum{i in 1..n} B[i,j] * p[i] >= 1;
```

Pitfalls

- Nonsquare systems
  - Side variables
  - Side constraints
- Orientation of equations
  - Skew symmetry preferred
  - Proximal point perturbation
- AMPL presolve
  - `option presolve 0;`

Definition

- Leader-follower game
  - Dominant player (leader) selects a strategy y^*
  - Then followers respond by playing a Nash game

$$x_i^* \in \left\{ \begin{array}{ll} \displaystyle\arg\min_{x_i \geq 0} & f_i(x, y) \\ \text{subject to} & c_i(x_i) \leq 0 \end{array} \right.$$

- Leader solves optimization problem with equilibrium constraints

$$\begin{array}{ll} \displaystyle\min_{y \geq 0, x, \lambda} & g(x, y) \\ \text{subject to} & h(y) \leq 0 \\ & 0 \leq x_i \quad \perp \quad \nabla_{x_i} f_i(x, y) + \lambda_i^T \nabla_{x_i} c_i(x_i) \geq 0 \\ & 0 \leq \lambda_i \quad \perp \quad -c_i(x_i) \geq 0 \end{array}$$

- Many applications in economics
  - Optimal taxation
  - Tolling problems

Nonlinear Programming Formulation

$$
\begin{array}{ll}
\underset{x,y,\lambda,s,t\geq 0}{\min} & g(x,y) \\
\text{subject to} & h(y) \leq 0 \\
& s_i = \nabla_{x_i} f_i(x,y) + \lambda_i^T \nabla_{x_i} c_i(x_i) \\
& t_i = -c_i(x_i) \\
& \sum_i \left(s_i^T x_i + \lambda_i t_i \right) \leq 0
\end{array}
$$

- Constraint qualification fails
    - Lagrange multiplier set unbounded
    - Constraint gradients linearly dependent
    - Central path does not exist
- Able to prove convergence results for some methods
- Reformulation very successful and versatile in practice

Penalization Approach

$$
\begin{array}{ll}
\min\limits_{x,y,\lambda,s,t\geq 0} & g(x,y)+\pi\sum\limits_i \left(s_i^T x_i+\lambda_i t_i\right)\\
\text{subject to} & h(y)\leq 0\\
 & s_i=\nabla_{x_i} f_i(x,y)+\lambda_i^T\nabla_{x_i} c_i(x_i)\\
 & t_i=-c_i(x_i)
\end{array}
$$

- Optimization problem satisfies constraint qualification
- Need to increase π

Relaxation Approach

$$
\begin{array}{ll}
\underset{x,y,\lambda,s,t\geq 0}{\min} & g(x,y) \\
\text{subject to} & h(y) \leq 0 \\
& s_i = \nabla_{x_i} f_i(x,y) + \lambda_i^T \nabla_{x_i} c_i(x_i) \\
& t_i = -c_i(x_i) \\
& \displaystyle\sum_i \left(s_i^T x_i + \lambda_i t_i \right) \leq \tau
\end{array}
$$

- Need to decrease τ

Model Formulation

- Economy with n agents and m commodities
  - $e \in \Re^{n \times m}$ are the endowments
  - $\alpha \in \Re^{n \times m}$ and $\beta \in \Re^{n \times m}$ are the utility parameters
  - $p \in \Re^{m}$ are the commodity prices
- Agent i maximizes utility with budget constraint

$$\begin{array}{ll} \max\limits_{x_{i,*} \geq 0} & \displaystyle\sum_{k=1}^{m} \frac{\alpha_{i,k}(1 + x_{i,k})^{1-\beta_{i,k}}}{1 - \beta_{i,k}} \\ \text{subject to} & \displaystyle\sum_{k=1}^{m} p_k \left(x_{i,k} - e_{i,k}\right) \leq 0 \end{array}$$

- Market k sets price for the commodity

$$0 \leq p_k \quad \perp \quad \sum_{i=1}^{n} \left(e_{i,k} - x_{i,k}\right) \geq 0$$

Model: cgempec.mod

```
set LEADER;                                  # Leader
set FOLLOWERS;                               # Followers
set AGENTS := LEADER union FOLLOWERS;        # All the agents
check: (card(LEADER) == 1 && card(LEADER inter FOLLOWERS) == 0);

set COMMODITIES;                             # Commodities

param e {AGENTS, COMMODITIES} >= 0, default 1; # Endowment

param alpha {AGENTS, COMMODITIES} > 0;       # Utility parameters
param beta {AGENTS, COMMODITIES} > 0;

var x {AGENTS, COMMODITIES};                 # Consumption (no bounds!)
var l {FOLLOWERS};                           # Multipliers (no bounds!)
var p {COMMODITIES};                         # Prices (no bounds!)

var u {i in AGENTS} =                        # Utility
  sum {k in COMMODITIES} alpha[i,k] * (1 + x[i,k])^(1 - beta[i,k]) / (1 - beta[i,k]);
var du {i in AGENTS, k in COMMODITIES} =     # Marginal prices
  alpha[i,k] / (1 + x[i,k])^beta[i,k];
```

Model: cgempec.mod

```
maximize
  objective: sum {i in LEADER} u[i];

subject to
  leader_budget {i in LEADER}:
    sum {k in COMMODITIES} p[k]*(e[i,k] - x[i,k]) >= 0;

  optimality {i in FOLLOWERS, k in COMMODITIES}:
    0 <= x[i,k] complements -du[i,k] + p[k] * l[i] >= 0;

  budget {i in FOLLOWERS}:
    0 <= l[i]   complements sum {k in COMMODITIES} p[k]*(e[i,k] - x[i,k]) >= 0;

  market {k in COMMODITIES}:
    0 <= p[k]   complements sum {i in AGENTS} (e[i,k] - x[i,k]) >= 0;
```

Data: cgempec.dat

```
set LEADER := Jorge;
set FOLLOWERS := Sven, Todd;
set COMMODITIES := Books, Cars, Food, Pens;

param alpha : Books  Cars  Food  Pens :=
     Jorge      1     1     1     1
     Sven       1     2     3     4
     Todd       2     1     1     5;

param beta (tr): Jorge  Sven  Todd :=
     Books        1.5    2     0.6
     Cars         1.6    3     0.7
     Food         1.7    2     2.0
     Pens         1.8    2     2.5;
```

Commands: cgempec.cmd

```
# Load model and data
model cgempec.mod;
data cgempec.dat;

# Specify solver and options
option presolve 0;
option solver "loqo";

# Solve the instance
drop market['Books'];
fix p['Books'] := 1;
solve;

# Output results
printf {i in AGENTS, k in COMMODITIES} "%5s %5s: % 5.4e\n", i, k, x[i,k] > cgempec.out;
printf "\n" > cgempec.out;
printf {k in COMMODITIES} "%5s: % 5.4e\n", k, p[k] > cgempec.out;
```

Output: cgempec.out

```
     Stackleberg                     Nash Game
Jorge Books:  9.2452e-01        Jorge Books:  8.9825e-01
Jorge  Cars:  1.3666e+00        Jorge  Cars:  1.4651e+00
Jorge  Food:  1.1508e+00        Jorge  Food:  1.2021e+00
Jorge  Pens:  7.7259e-01        Jorge  Pens:  6.8392e-01
 Sven Books:  2.5499e-01         Sven Books:  2.5392e-01
 Sven  Cars:  7.4173e-01         Sven  Cars:  7.2054e-01
 Sven  Food:  1.6657e+00         Sven  Food:  1.6271e+00
 Sven  Pens:  1.4265e+00         Sven  Pens:  1.4787e+00
 Todd Books:  1.8205e+00         Todd Books:  1.8478e+00
 Todd  Cars:  8.9169e-01         Todd  Cars:  8.1431e-01
 Todd  Food:  1.8355e-01         Todd  Food:  1.7081e-01
 Todd  Pens:  8.0093e-01         Todd  Pens:  8.3738e-01

Books:  1.0000e+00              Books:  1.0000e+00
 Cars:  5.9617e-01               Cars:  6.1742e-01
 Food:  6.6496e-01               Food:  6.8345e-01
 Pens:  1.0700e+00               Pens:  1.0237e+00
```

Unbounded Multipliers

```
var z{1..2} >= 0;
var z3;

minimize objf: z[1] + z[2] - z3;
subject to
  lin1:   -4 * z[1] + z3 <= 0;
  lin2:   -4 * z[2] + z3 <= 0;
  compl:  z[1]*z[2] <= 0;
```

LOQO Output

```
LOQO 6.06: outlev=2
     |           Primal           |            Dual
Iter |  Obj Value    Infeas       |   Obj Value   Infeas
- - - - - - - - - - - - - - - - - - - - - - - - - - -
  1     1.000000e+00 0.0e+00    0.000000e+00 1.1e+00
  2     6.902180e-01 2.2e-01  -2.672676e-01 2.6e-01
  3     2.773222e-01 1.6e-01  -3.051049e-01 1.1e-01

292   -8.213292e-05 1.7e-09  -4.106638e-05 9.1e-07
293   -8.202525e-05 1.7e-09  -4.101255e-05 9.1e-07

294                nan    nan              nan      nan
500                nan    nan              nan      nan
```

FILTER Output

```
iter  |     rho      |    ||d||     |     f / hJ     |    ||c||/hJt
------+------------+------------+--------------+--------------
   0:0   10.0000      0.00000      1.0000000      0.0000000
   1:1   10.0000      1.00000      0.0000000      0.0000000

  24:1 0.156250     0.196695E-05-0.98347664E-06 0.24180657E-12
  25:1 0.156250     0.983477E-06-0.49173832E-06 0.60451644E-13

Norm of KKT residual................  0.471404521
max( |lam_i| * || a_i ||)...........  2.06155281
Largest modulus multiplier..........  2711469.25
```

Limitations

- Multipliers may not exist
- Solvers can have a hard time computing solutions
  - Try different algorithms
  - Compute feasible starting point
- Stationary points may have descent directions
  - Checking for descent is an exponential problem
  - Strong stationary points found in certain cases
- Many stationary points – global optimization
- Formulation of follower problem
  - Multiple solutions to Nash game
  - Nonconvex objective or constraints
  - Existence of multipliers