Computational Optimization Overview

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

Outline: Six Topics

- Introduction
- Unconstrained optimization
  - Limited-memory variable metric methods
- Systems of Nonlinear Equations
  - Sparsity and Newton’s method
- Automatic Differentiation
  - Computing sparse Jacobians via graph coloring
- Constrained Optimization
  - All that you need to know about KKT conditions
- Solving optimization problems
  - Modeling languages: AMPL and GAMS
  - NEOS
Classification of Constrained Optimization Problems

\[ \min \{ f(x) : x_l \leq x \leq x_u, \ c_l \leq c(x) \leq c_u \} \]

- Number of variables \( n \)
- Number of constraints \( m \)
- Number of linear constraints
- Number of equality constraints \( n_e \)
- Number of degrees of freedom \( n - n_e \)
- Sparsity of \( c'(x) = (\partial_i c_j(x)) \)
- Sparsity of \( \nabla^2 \mathcal{L}(x, \lambda) = \nabla^2 f(x) + \sum_{k=1}^{m} \nabla^2 c_k(x) \lambda_k \)

Classification of Constrained Optimization Software

- Formulation
- Interfaces: MATLAB, AMPL, GAMS
- Second-order information options:
  - Differences
  - Limited memory
  - Hessian-vector products
- Linear solvers
  - Direct solvers
  - Iterative solvers
  - Preconditioners
- Partially separable problem formulation
- Documentation
- License
Life-Cycles Saving Problem

Maximize the utility

\[ \sum_{t=1}^{T} \beta^t u(c_t) \]

where \( S_t \) are the saving, \( c_t \) is consumption, \( w_t \) are wages, and

\[ S_{t+1} = (1 + r)S_t + w_{t+1} - c_{t+1}, \quad 0 \leq t < T \]

with \( r = 0.2 \) interest rate, \( \beta = 0.9 \), \( S_0 = S_T = 0 \), and

\[ u(c) = -\exp(-c) \]

Assume that \( w_t = 1 \) for \( t < R \) and \( w_t = 0 \) for \( t \geq R \).

**Question.** What are the characteristics of the life-cycle problem?

---

Topic 2: Unconstrained Optimization

Augustin Louis Cauchy (August 21, 1789 – May 23, 1857)

Additional information at [Mac Tutor](http://www-history.mcs.st-andrews.ac.uk)

---

Constrained Optimization Software: IPOPT

- **Formulation**
  \[ \min \{ f(x) : x_l \leq x \leq x_u, \ c(x) = 0 \} \]

- **Interfaces:** AMPL
- **Second-order information options:**
  - Differences
  - Limited memory
  - Hessian-vector products
- **Direct solvers:** MA27, MA57
- **Partially separable problem formulation:** None
- **Documentation**
- **License**
Unconstrained Optimization: Background

Given a continuously differentiable \( f : \mathbb{R}^n \mapsto \mathbb{R} \) and

\[
\min \{ f(x) : x \in \mathbb{R}^n \}
\]

generate a sequence of iterates \( \{x_k\} \) such that the gradient test

\[
\| \nabla f(x_k) \| \leq \tau
\]

is eventually satisfied

**Theorem.** If \( f : \mathbb{R}^n \mapsto \mathbb{R} \) is continuously differentiable and bounded below, then there is a sequence \( \{x_k\} \) such that

\[
\lim_{k \to \infty} \| \nabla f(x_k) \| = 0.
\]

**Exercise.** Prove this result.

---

Ginzburg-Landau Model

Minimize the Gibbs free energy for a homogeneous superconductor

\[
\int_D \left\{ -|v(x)|^2 + \frac{1}{2}|v(x)|^4 + \|\nabla - iA(x)\|v(x)\|^2 + \kappa^2 \|\nabla \times A(x)\|^2 \right\} \, dx
\]

\[v : \mathbb{R}^2 \mapsto \mathbb{C} \text{ (order parameter)}\]

\[A : \mathbb{R}^2 \mapsto \mathbb{R}^2 \text{ (vector potential)}\]

Unconstrained problem. Non-convex function. Hessian is singular. Unique minimizer, but there is a saddle point.

---

Unconstrained Optimization

What can I use if the gradient \( \nabla f(x) \) is not available?

- Geometry-based methods: Pattern search, Nelder-Mead, ...
- Model-based methods: Quadratic, radial-basis models, ...

What can I use if the gradient \( \nabla f(x) \) is available?

- Conjugate gradient methods
- Limited-memory variable metric methods
- Variable metric methods

---

Computing the Gradient

**Hand-coded gradients**

- Generally efficient
- Error prone
- The cost is usually less than 5 function evaluations

**Difference approximations**

\[
\partial_i f(x) \approx \frac{f((x + he_i) - f(x))}{h_i}
\]

- Choice of \( h_i \) may be problematic in the presence of noise.
- Costs \( n \) function evaluations
- Accuracy is about the \( \varepsilon_f^{1/2} \) where \( \varepsilon_f \) is the noise level of \( f \)
Cheap Gradient via Automatic Differentiation

Code generated by automatic differentiation tools

- Accurate to full precision
- For the reverse mode the cost is $\Omega_T T \{ f(x) \}$.
- In theory, $\Omega_T \leq 5$.
- For the reverse mode the memory is proportional to the number of intermediate variables.

Exercise

Develop an order $n$ code for computing the gradient of

$$f(x) = \prod_{k=1}^{n} x_k$$

Line Search Methods

A sequence of iterates \{x_k\} is generated via

$$x_{k+1} = x_k + \alpha_k p_k,$$

where $p_k$ is a descent direction at $x_k$, that is,

$$\nabla f(x_k)^T p_k < 0,$$

and $\alpha_k$ is determined by a line search along $p_k$.

Line searches

- Geometry-based: Armijo, . . .
- Model-based: Quadratics, cubic models, . . .

Powell-Wolfe Conditions on the Line Search

Given $0 \leq \mu < \eta \leq 1$, require that

$$f(x + \alpha p) \leq f(x) + \mu \alpha \nabla f(x_k)^T p_k \quad \text{sufficient decrease}$$

$$|\nabla f(x + \alpha p)^T p| \leq \eta |\nabla f(x)^T p| \quad \text{curvature condition}$$

Conjugate Gradient Algorithms

Given a starting vector $x_0$ generate iterates via

$$x_{k+1} = x_k + \alpha_k p_k$$

$$p_{k+1} = \nabla f(x_k) + \beta_k p_k$$

where $\alpha_k$ is determined by a line search.

Three reasonable choices of $\beta_k$ are $(g_k = \nabla f(x_k))$:

$$\beta_{FR}^k = \left( \frac{\|g_{k+1}\|}{\|g_k\|} \right)^2,$$  
Fletcher-Reeves

$$\beta_{PR}^k = \frac{\langle g_{k+1}, g_{k+1} - g_k \rangle}{\|g_k\|^2},$$  
Polak-Rivièere

$$\beta_{PR+}^k = \max \{ \beta_{PR}^k, 0 \},$$  
PR-plus
Limited-Memory Variable-Metric Algorithms

Given a starting vector $x_0$ generate iterates via

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)$$

where $\alpha_k$ is determined by a line search.

The matrix $H_k$ is defined in terms of information gathered during the previous $m$ iterations.

- $H_k$ is positive definite.
- Storage of $H_k$ requires $2mn$ locations.
- Computation of $H_k \nabla f(x_k)$ costs $(8m + 1)n$ flops.

Recommendations

But what algorithm should I use?

- If the gradient $\nabla f(x)$ is not available, then a model-based method is a reasonable choice. Methods based on quadratic interpolation are currently the best choice.
- If the gradient $\nabla f(x)$ is available, then a limited-memory variable metric method is likely to produce an approximate minimizer in the least number of gradient evaluations.
- If the Hessian is also available, then a state-of-the-art implementation of Newton’s method is likely to produce the best results if the problem is large and sparse.

Topic 3: Newton’s Method

![Sir Isaac Newton](https://via.placeholder.com/150)

Sir Isaac Newton (January 4, 1643 – March 331, 1727)
Additional information at [Mac Tutor](http://www-history.mcs.st-andrews.ac.uk)

Motivation

Given a continuously differentiable $f : \mathbb{R}^n \mapsto \mathbb{R}^n$, solve

$$f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{pmatrix} = 0$$

**Linear models.** The mapping defined by

$$L_k(s) = f(x_k) + f'(x_k)s$$

is a linear model of $f$ near $x_k$, and thus it is sensible to choose $s_k$ such that $L_k(s_k) = 0$ provided $x_k + s_k$ is near $x_k$. 
Newton’s Method

Given a starting point $x_0$, Newton’s method generates iterates via

$$f'(x_k)s_k = -f(x_k), \quad x_{k+1} = x_k + s_k.$$ 

**Computational Issues**

- How do we solve for $s_k$?
- How do we handle a (nearly) singular $f'(x_k)$?
- How do we enforce convergence if $x_0$ is not near a solution?
- How do we compute/approximate $f'(x_k)$?
- How accurately do we solve for $s_k$?
- Is the algorithm scale invariant?
- Is the algorithm mesh-invariant?

**Flow in a Channel Problem**

Analyze the flow of a fluid during injection into a long vertical channel, assuming that the flow is modeled by the boundary value problem below, where $u$ is the potential function and $R$ is the Reynolds number.

$$u''' = R(u'u'' - uu''')$$

$u(0) = 0, \quad u(1) = 1$

$u'(0) = u'(1) = 0$

**Sparsity**

Assume that the Jacobian matrix is sparse, and let $\rho_i$ be the number of non-zeroes in the $i$-th row of $f'(x)$.

- Sparse linear solvers can solve $f'(x)s = -f(x)$ in order $\rho_A$ operations, where $\rho_A = \text{avg}\{\rho_i^2\}$.
- Graph coloring techniques (see Topic 4) can compute or approximate the Jacobian matrix with $\rho_M$ function evaluations where $\rho_M = \text{max}\{\rho_i\}$.

**Topic 4: Automatic Differentiation**

Gottfried Wilhelm Leibniz (July 1, 1646 – November 14, 1716)

Additional information at Mac Tutor

www-history.mcs.st-andrews.ac.uk
Computing Gradients and Sparse Jacobians

Theorem. Given $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, automatic differentiation tools compute $f'(x)v$ at a cost comparable to $f(x)$.

Tasks

- Given $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ with a sparse Jacobian, compute $f'(x)$ with $p \ll n$ evaluations of $f'(x)v$.
- Given a partially separable $f : \mathbb{R}^n \mapsto \mathbb{R}$, compute $\nabla f(x)$ with $p \ll n$ evaluations of $\langle \nabla f(x), v \rangle$.

Requirements:

$$T\{f'(x)\} \leq \Omega T\{f(x)\}, \quad M\{\nabla f(x)\} \leq \Omega M\{f(x)\}$$

where $T\{\cdot\}$ is computing time and $M\{\cdot\}$ is memory.

Structurally Orthogonal Columns

Structurally orthogonal columns do not have a nonzero in the same row position.

Observation.

We can compute the columns in a group of structurally orthogonal columns with an evaluation of $f'(x)v$.

$$f'(x) = \begin{pmatrix} \times & \times & \times \times & \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \times \time

Coloring the Jacobian matrix $f'(x)$

Partitioning the columns of $f'(x)$ into $p$ groups of structurally orthogonal columns is equivalent to a graph coloring problem.

For each group of structurally orthogonal columns, define $v \in \mathbb{R}^n$ with $v_i = 1$ if column $i$ is in the group, and $v_i = 0$ otherwise. Set $V = (v_1, v_2, \ldots, v_p)$.

Compute $f'(x)$ from the compressed Jacobian matrix $f'(x)V$.

Observation. In practice $p \approx \rho_M$ where $\rho_M \equiv \max\{\rho_i\}$, and $\rho_i$ is the number of non-zeros in the $i$-th row of $f'(x)$.
Sparsity Pattern of the Jacobian Matrix

Optimization software tends to require the closure of the sparsity pattern

\[ \bigcup \{ S(f'(x)) : x \in \mathcal{D} \} . \]

in a region \( \mathcal{D} \) of interest. In our case,

\[ \mathcal{D} = \{ x \in \mathbb{R}^n : x_l \leq x \leq x_u \} \]

Given \( x_0 \in \mathcal{D} \), we evaluate the sparsity pattern of \( f_E'(\bar{x}_0) \), where \( \bar{x}_0 \) is a random, small perturbation of \( x_0 \), for example,

\[ \bar{x}_0 = (1 + \varepsilon)x_0 + \varepsilon, \quad \varepsilon \in [10^{-6}, 10^{-4}] \]

Partially Separable Functions

The mapping \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is partially separable if

\[ f(x) = \sum_{i=1}^m f_i(x), \]

and \( f_i \) only depends on \( p_i \ll n \) variables.

**Theorem** (Griewank and Toint [1981]). If \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) has a sparse Hessian matrix then \( f \) is partially separable.

Partially Separable Functions: The Trick

If \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is partially separable, the extended function

\[ f_E(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix} \]

has a sparse Jacobian matrix \( f_E'(x) \). Moreover,

\[ f(x) = f_E(x)^T e \quad \implies \quad \nabla f(x) = f_E'(x)^T e \]

**Observation.** We can compute the dense gradient by computing the sparse Jacobian matrix \( f_E'(x) \).

Computational Experiments

Experiments based on the MINPACK-2 collection of large-scale problems show that gradients of partially separable functions can be computed efficiently.

\[ T\{\nabla f(x)\} = \kappa \rho_M \max T\{f(x)\} \]

Quartiles of \( \kappa \)

\[
\begin{array}{cccccc}
2,500 & \leq n & \leq 40,000 \\
1.3 & 2.9 & 5.0 & 8.2 & 22.2 \\
\end{array}
\]
Topic 5: Constrained Optimization

Joseph-Louis Lagrange (January 25, 1736 – April 10, 1813)
Additional information at Mac Tutor
www-history.mcs.st-andrews.ac.uk

Geometric Viewpoint of the KKT Conditions

For any closed set Ω, consider the abstract problem
\[
\min \{ f(x) : x \in \Omega \}
\]

The tangent cone
\[
T(x^*) = \left\{ v : v = \lim_{k \to \infty} \frac{x_k - x^*}{\alpha_k}, \ x_k \in \Omega, \ \alpha_k \geq 0 \right\}
\]

The normal cone
\[
N(x^*) = \{ w : \langle w, v \rangle \leq 0, \ v \in T(x^*) \}
\]

First order conditions
\[
-\nabla f(x^*) \in N(x^*)
\]

Computational Viewpoint of the KKT Conditions

In the case Ω = \{x \in \mathbb{R}^n : c(x) \geq 0\}, define
\[
C(x^*) = \left\{ w : w = \sum_{i=1}^{m} \lambda_i (-\nabla c_i(x^*)), \ \lambda_i \geq 0 \right\}
\]

In general \(C(x^*) \subset N(x^*)\), and under a constraint qualification
\[
\text{Linear independence}\quad C(x^*) = N(x^*)
\]

Hence, for some multipliers \(\lambda_i \geq 0\),
\[
\nabla f(x) = \sum_{i=1}^{m} \lambda_i \nabla c_i(x), \quad \lambda_i \geq 0,
\]

Constraint Qualifications

In the case where
\[
\Omega = \{x \in \mathbb{R}^n : l \leq c(x) \leq u\}
\]

the main two constraint qualifications are

Linear independence
The active constraint normals are positively linearly independent, that is, if
\[
C_A = (\nabla c_i(x) : c_i(x) \in \{l_i, u_i\})
\]
then \(C_A\) has full rank.

Mangasarian-Fromovitz
The active constraint normals are positively linearly independent.
**Lagrange Multipliers**

For the general problem with 2-sided constraints

$$\min \{ f(x) : l \leq c(x) \leq u \}$$

the KKT conditions for a local minimizer are

$$\nabla f(x) = \sum_{i=1}^{m} \lambda_i \nabla c_i(x), \quad l \leq c(x) \leq u,$$

where the multipliers satisfy complementarity conditions

- $\lambda_i$ is unrestricted if $l_i = u_i$.
- $\lambda_i = 0$ if $c_i(x) \notin \{l_i, u_i\}$
- $\lambda_i \geq 0$ if $c_i(x) = l_i$
- $\lambda_i \leq 0$ if $c_i(x) = u_i$

**Lagrangians**

The KKT conditions for the problem with constraints $l \leq c(x) \leq u$ can be written in terms of the Lagrangian

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i=1}^{m} \lambda_i c_i(x).$$

**Examples.**

The KKT conditions for the equality-constrained $c(x) = 0$ are

$$\nabla_x \mathcal{L}(x, \lambda) = 0, \quad c(x) = 0.$$  

The KKT conditions for the inequality-constrained $c(x) \geq 0$ are

$$\nabla_x \mathcal{L}(x, \lambda) = 0, \quad c(x) \geq 0, \quad \lambda \geq 0, \quad \lambda \perp c(x)$$

where $\lambda \perp c(x)$ means that $\lambda_i c_i(x) = 0$.

**Newton’s Method: Equality-Constrained Problems**

The KKT conditions for the equality-constrained problem $c(x) = 0,$

$$\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) - \sum_{i=1}^{m} \lambda_i \nabla c_i(x) = 0, \quad c(x) = 0.$$  

are a system of $n + m$ nonlinear equations.

Newton’s method for this system can be written as

$$x_+ = x + s_x, \quad \lambda_+ = \lambda + s_\lambda$$

where

$$\begin{pmatrix} \nabla^2_x \mathcal{L}(x, \lambda) & -\nabla c(x) \\ \nabla c(x)^T & 0 \end{pmatrix} \begin{pmatrix} s_x \\ s_\lambda \end{pmatrix} = -\begin{pmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ c(x) \end{pmatrix}.$$

**Saddle Point Problems**

Given a symmetric $n \times n$ matrix $H$ and a $n \times m$ matrix $C$, under what conditions is

$$A = \begin{pmatrix} H & C \\ C^T & 0 \end{pmatrix}$$

nonsingular?

**Lemma.** If $C$ has full rank and

$$C^T u = 0, \quad u \neq 0, \quad \implies \quad u^T H u > 0$$

then $A$ is nonsingular.
**Topic 6: Solving Optimization Problems**

**Environments**
- Modeling Languages: AMPL, GAMS
- NEOS

**The Classical Model**

**The NEOS Model**

A collaborative research project that represents the efforts of the optimization community by providing access to 50+ solvers from both academic and commercial researchers.

**NEOS: Under the Hood**
- Modeling languages for optimization: AMPL, GAMS
- Automatic differentiation tools: ADIFOR, ADOL-C, ADIC
- Python
- Optimization solvers (50+)
  - Benchmark, GAMS/AMPL (Multi-Solvers)
  - MINLP, FortMP, GLPK, Xpress-MP, ...
  - CONOPT, FILTER, IPOPT, KNITRO, LANCELOT, LOQO, MINOS, MOSEK, PATHNLp, PENNON, SNOPT
  - BPMPD, FortMP, MOSEK, OOQP, Xpress-MP, ...
  - CSDP, DSDP, PENSDPP, SDPA, SeDuMi, ...
  - BLMVM, L-BFGS-B, TRON, ...
  - MILES, PATH
  - Concorde
Research Issues for NEOS

- How do we add solvers?
- How are problems specified?
- How are problems submitted?
- How are problems scheduled for solution?
- How are the problems solved?
- Where are the problems solved?

Arizona State University
Lehigh University
Universidade do Minho, Portugal
Technical University Aachen, Germany
National Taiwan University, Taiwan
Northwestern University
Università di Roma *La Sapienza*, Italy
Wisconsin University

Solving Optimization Problems: NEOS Interfaces

Interfaces
- Kestrel
- NEOS Submit
- Web browser
- Email

Pressure in a Journal Bearing

$$\min \left\{ \int_D \left\{ \frac{1}{2} w_q(x) \|\nabla v(x)\|^2 - w_l(x)v(x) \right\} \, dx : v \geq 0 \right\}$$

$$w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3$$
$$w_l(\xi_1, \xi_2) = \epsilon \sin \xi_1$$
$$D = (0, 2\pi) \times (0, 2b)$$

Number of active constraints depends on the choice of $\epsilon$ in $(0, 1)$. Nearly degenerate problem. Solution $v \notin C^2$.

AMPL Model for the Journal Bearing: Parameters

Finite element triangulation

```
param nx > 0, integer; # grid points in 1st direction
param ny > 0, integer; # grid points in 2nd direction
param b; # grid is (0,2*pi)x(0,2*b)
param e; # eccentricity

param pi := 4*atan(1);
param hx := 2*pi/(nx+1); # grid spacing
param hy := 2*b/(ny+1); # grid spacing
param area := 0.5*hx*hy; # area of triangle
param wq {i in 0..nx+1} := (1+e*cos(i*hx))^3;
```
AMPL Model for the Journal Bearing

\[
\text{var } v \{i \in 0..nx+1, j \in 0..ny+1\} \geq 0;
\]

\[
\text{minimize } q:\]
\[
0.5*(hx*hy/6) * \sum \{i \in 0..nx, j \in 0..ny\} (wq[i] + 2*wq[i+1]) * \left(\frac{(v[i+1,j] - v[i,j])}{hx}\right)^2 + \left(\frac{(v[i,j+1] - v[i,j])}{hy}\right)^2 + \]
\[
0.5*(hx*hy/6) * \sum \{i \in 1..nx+1, j \in 1..ny+1\} (wq[i] + 2*wq[i-1]) * \left(\frac{(v[i-1,j] - v[i,j])}{hx}\right)^2 + \left(\frac{(v[i,j-1] - v[i,j])}{hy}\right)^2 - hx*hy * \sum \{i \in 0..nx+1, j \in 0..ny+1\} (e*sin(i*hx)*v[i,j]);
\]

subject to c1 \{i \in 0..nx+1\}: v[i,0] = 0;
subject to c2 \{i \in 0..nx+1\}: v[i,ny+1] = 0;
subject to c3 \{j \in 0..ny+1\}: v[0,j] = 0;
subject to c4 \{j \in 0..ny+1\}: v[nx+1,j] = 0;

AMPL Model for the Journal Bearing: Data

\[
\text{# Set the design parameters}
\]
\[
\text{param } b := 10;
\text{param } e := 0.1;
\]

\[
\text{# Set parameter choices}
\]
\[
\text{let } nx := 50;
\text{let } ny := 50;
\]

\[
\text{# Set the starting point.}
\]
\[
\text{let } \{i \in 0..nx+1, j \in 0..ny+1\} v[i,j] := \max(sin(i*hx),0);
\]

AMPL Model for the Journal Bearing: Commands

\[
\text{option show_stats 1;}
\]
\[
\text{option solver "knitro";}
\text{option solver "snopt";}
\text{option solver "loqo";}
\text{option loqo_options "outlev=2 timing=1 iterlim=500";}
\]
\[
\text{model;}
\text{include bearing.mod;}
\]
\[
\text{data;}
\text{include bearing.dat;}
\]
\[
\text{solve;}
\]
\[
\text{printf } \{i \in 0..nx+1, j \in 0..ny+1\}: "\%21.15e\n", v[i,j] > cops.dat;
\text{printf } "\%10d\n \%10d\n", nx, ny > cops.dat;
\]

Life-Cycles Saving Problem

Maximize the utility
\[
\sum_{t=1}^{T} \beta^t u(c_t)
\]
where \(S_t\) are the saving, \(c_t\) is consumption, \(w_t\) are wages, and
\[
S_{t+1} = (1 + r)S_t + w_{t+1} - c_{t+1}, \quad 0 \leq t < T
\]
with \(r = 0.2\) interest rate, \(\beta = 0.9\), \(S_0 = S_T = 0\), and
\[
u(c) = -\exp(-c)
\]
Assume that \(w_t = 1\) for \(t < R\) and \(w_t = 0\) for \(t \geq R\).
Life-Cycles Saving Problem: Model

param T integer;  # Number of periods
param R integer;  # Retirement
param beta;      # Discount rate
param r;         # Interest rate
param S0;        # Initial savings
param ST;        # Final savings
param w{1..T};   # Wages

var S{0..T};     # Savings
var c{0..T};     # Consumption

maximize utility: sum{t in 1..T} beta^t*(-exp(-c[t]));
subject to budget {t in 0..T-1}: S[t+1] = (1+r)*S[t] + w[t+1] - c[t+1];
subject to savings {t in 0..T}: S[t] >= 0.0;
subject to consumption {t in 1..T}: c[t] >= 0.0;

subject to bc1: S[0] = S0;
subject to bc2: S[T] = ST;
subject to bc3: c[0] = 0.0;

Life-Cycles Saving Problem: Data

param T := 100;
param R := 60;
param beta := 0.9;
param r := 0.2;
param S0 := 0.0;
param ST := 0.0;
# Wages
let {i in 1..R} w[i] := 1.0;
let {i in R..T} w[i] := 0.0;
let {i in 1..R} w[i] := (i/R);
let {i in R..T} w[i] := (i - T)/(R - T);

Life-Cycles Saving Problem: Commands

option show_stats 1;
option solver "filter";
option solver "ipopt";
option solver "knitro";
option solver "loqo";

model;
include life.mod;
data;
include life.dat;
solve;

printf {t in 0..T}: "%21.15e %21.15e\n", c[t], S[t] > cops.dat;

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
  - Conditional and loop constructs
  - Execution of external programs

Model Formulation

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

\[
\begin{align*}
\max_{x \geq 0} & \quad \sum_{i=1}^{n} \lambda_i \left( \sum_{k=1}^{m} \frac{\alpha_{i,k}(1 + x_{i,k})^{1-\beta_{i,k}}}{1 - \beta_{i,k}} \right) \\
\text{subject to} & \quad \sum_{i=1}^{n} x_{i,k} \leq \sum_{i=1}^{n} e_{i,k} \quad \forall k = 1, \ldots, m
\end{align*}
\]
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} =
    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
    2 1 2 3
    3 2 1 1

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;
dprintf {i in 1..n} "%2d: % 5.4e\n", i, u[i];

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  0.8881471
    2  0.574164
    3  0.703454
    4  0.267241
    5  0.060263
    6  0.604858
    7  1.7239
    8  1.47516
    9  2.85859
   10  1.82098
   11  0.572645
   12  1.2576
   13  1.2876

ampl: quit;
**Model: social2.mod**

```plaintext
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**

```plaintext
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**

```plaintext
# 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.
26 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 1.82098
    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
  - \( \alpha \) and \( \beta \) are the congestion parameters
  - \( b \) denotes the supply and demand

- Multicommodity network flow problem

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

Data: network.dat

```plaintext
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;
```

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

Commands: network.cmd

```plaintext
# 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;
}
```

Model: network.mod

```plaintext
set NODES;
set ARCS within NODES cross NODES;
set COMMODITIES := 1..3;

param b {NODES, COMMODITIES} default 0;
check {k in COMMODITIES}:
    sum{i in NODES} b[i,k] >= 0;
param alpha{ARCS} >= 0;
param beta{ARCS} >= 0;
var x{ARCS, COMMODITIES} >= 0;
var f{(i,j) in ARCS} =
    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];
```

Leyffer, Moré, and Munson
### Output

```ampl
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`

```plaintext
# 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 "Commodity: %d
", k;
  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`

```plaintext
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;
```

---

**Overview**

**Examples**

- Social Planning for Endowment Economy
- Traffic Routing with Congestion
- Finite Element Method

**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
- Define solver options
- Coordinate descent method
- Solve for iterations
- Output results

**Improved Coordinate Descent: `wardrop.mod`**

- Define sets and parameters
- Define variables
- Minimization objective
- Conservation constraints
- Problem setup
**Improved Coordinate Descent: wardrop1.cmd**

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

**Final Coordinate Descent: wardrop2.cmd**

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

**Ordered Sets**

```plaintext
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**

```plaintext
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

\[
\begin{align*}
\text{minimize} & \quad f(x) & \quad \text{objective} \\
\text{subject to} & \quad c(x) = 0 & \quad \text{constraints} \\
& \quad x \geq 0 & \quad \text{variables}
\end{align*}
\]

- \( f : \mathbb{R}^n \to \mathbb{R} \), \( c : \mathbb{R}^n \to \mathbb{R}^m \) smooth (typically \( C^2 \))
- \( x \in \mathbb{R}^n \) 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 \( y^*, z^* \):

\[
\begin{align*}
\nabla f(x^*) - \nabla c(x^*)^T y^* - z^* &= 0 \\
c(x^*) &= 0 \\
X^* z^* &= 0 \\
x^* &\geq 0, \quad z^* \geq 0
\end{align*}
\]

where \( X^* = \text{diag}(x^*) \), thus \( X^* z^* = 0 \) \( \iff \) \( x^*_i z^*_i = 0 \)

Lagrangian: \( \mathcal{L}(x, y, z) := f(x) - y^T c(x) - z^T x \)

Objective gradient is linear combination of constraint gradients

\( g(x) = A(x)y, \quad \text{where} \quad g(x) := \nabla f(x), \quad A(x) := \nabla c(x)^T \)
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 $F \in C^2$, and $\nabla F(x^*)$ nonsingular, then Newton converges quadratically near $x^*$.

Sequential Quadratic Programming (SQP)

Consider equality constrained NLP

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

Optimality conditions:

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

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

... solve using Newton’s method

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

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 $w_k = (x_k, y_k)$ ...

$$H_k - A_k y = -\nabla \mathcal{L}(x_k, y_k)$$

... set $(x_{k+1}, y_{k+1}) = (x_k + s, y_k + y_{k+1})$ ... $A_k = \nabla c(x_k)^T$

... solve for $y_{k+1} = y_k + s_y$ directly instead:

$$H_k - A_k y = -\nabla f_k$$

... set $(x_{k+1}, y_{k+1}) = (x_k + s, y_k + y_{k+1})$
Sequential Quadratic Programming (SQP)

Newton’s Method for KKT conditions leads to:

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

... are optimality conditions of QP

\[
\begin{aligned}
\text{minimize } & \quad \nabla f^T_k s + \frac{1}{2} s^T H_k s \\
\text{subject to } & \quad c_k + A^T_k s = 0
\end{aligned}
\]

... hence Sequential Quadratic Programming

Modern Interior Point Methods (IPM)

General NLP

minimize \( f(x) \) subject to \( c(x) = 0 \) \& \( x \geq 0 \)

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

\[
F_{\mu}(x, y, z) = \begin{bmatrix}
\nabla f(x) - \nabla c(x)^T y - z \\
c(x) \\
X z - \mu e
\end{bmatrix} = 0
\]

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

Sequential Quadratic Programming (SQP)

SQP for inequality constrained NLP:

minimize \( f(x) \) subject to \( c(x) = 0 \) \& \( x \geq 0 \)

REPEAT

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

\[
\begin{aligned}
\text{minimize } & \quad \nabla f^T_k s + \frac{1}{2} s^T H_k s \\
\text{subject to } & \quad c_k + A^T_k s = 0 \\
x_k + s & \geq 0
\end{aligned}
\]

2. Set \( x_{k+1} = x_k + s \)

Modern Interior Point Methods (IPM)

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

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

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)

\[
\begin{align*}
\text{minimize} & \quad f(x) \quad \text{subject to} \quad c(x) = 0 & \text{and} & \quad x \geq 0 \\
\text{Related to classical barrier methods} & \quad [\text{Fiacco & McCormick}] \\
\left\{ \begin{array}{l}
\text{minimize} \\
\text{subject to}
\end{array} \right. & \quad f(x) - \mu \sum \log(x_i) \\
\mu = 0.1 & \quad \mu = 0.001
\end{align*}
\]

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

Classical Interior Point Methods (IPM)

Newton’s method for barrier problem from \( x_k \) ...

\[
\begin{bmatrix}
\nabla^2 L_k + \mu X_k^{-2} & -A_k \\
A_k^T & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} = ...
\]

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

\[
\begin{bmatrix}
\nabla^2 L_k + Z(x_k)X_k^{-1} & -A_k \\
A_k & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} = ...
\]

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

Classical Interior Point Methods (IPM)

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

\[
\begin{bmatrix}
\nabla^2 L_k & -A_k & -I \\
A_k^T & 0 & 0 \\
Z_k & 0 & X_k
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y \\
\Delta z
\end{bmatrix} = -F_{\mu}(x_k, y_k, z_k)
\]

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

\[
\begin{bmatrix}
\nabla^2 L_k + Z_kX_k^{-1} & -A_k \\
A_k & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} = ...
\]
Interior Point Methods (IPM)

Primal-dual system ...

\[
\begin{bmatrix}
\nabla^2 L_k + Z_k X_k^{-1} & -A_k \\
A_k & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} = ...
\]

... compare to barrier system ...

\[
\begin{bmatrix}
\nabla^2 L_k + Z(x_k) X_k^{-1} & -A_k \\
A_k & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} = ...
\]

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

Penalty Functions (i)

Augmented Lagrangian Methods

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

As \(y_k \to y^*\):
- \(x_k \to x^*\) for \(\rho_k > \bar{\rho}\)
- No ill-conditioning, improves convergence rate

- update \(\rho_k\) 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

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

- combine constraints ad objective
- equivalence of optimality \(\Rightarrow\) exact for \(\pi > \|y^*\|_D\)
  \(\Rightarrow\) now apply unconstrained techniques
- \(\Phi\) 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

Trust Region Methods

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

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

**REPEAT**

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

**UNTIL** convergence
Filter Methods for NLP

Penalty function can be inefficient

- Penalty parameter not known a priori
- Large penalty parameter ⇒ slow convergence

Two competing aims in optimization:
1. Minimize $f(x)$
2. Minimize $h(x) := \|c(x)\|$ ... more important

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

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 $\Delta = \Delta/2$

⇒ 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
  - $\ell_1$ exact penalty function
- SLIQUE (part of KNITRO)
  - SLP-EQP (“SQP” for larger problems)
  - trust-region with $\ell_1$ 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
  - $\ell_1$ 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 $\frac{\partial c_i(x)}{\partial x_j} \approx \frac{c_i(x+\delta e_j) - c_i(x)}{\delta}$ 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*

```plaintext
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
```

... results in error message like
*Cannot evaluate objective at start*

... change initialization of s:
```plaintext
var s := 1;
```

... difficult, if IEEE during solve ...

**Unbounded Objective**

*Penalized MPEC $\pi = 1$:

```plaintext
minimize \[ x_1^2 + x_2^2 - 4x_1x_2 + \pi x_1x_2 \]
subject to \[ x_1, x_2 \geq 0 \]
```

... 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*

```plaintext
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**

<table>
<thead>
<tr>
<th></th>
<th>Primal</th>
<th></th>
<th>Dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter</td>
<td>Obj Value</td>
<td>Infeas</td>
<td>Obj Value</td>
</tr>
<tr>
<td>1</td>
<td>-1.0000000e+03</td>
<td>4.2e+00</td>
<td>-6.0000000e+00</td>
</tr>
<tr>
<td>[...]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2.312535e-04</td>
<td>7.9e-01</td>
<td>1.715213e+12</td>
</tr>
</tbody>
</table>

LOQO 6.06: iteration limit

... fails to converge ... not useful for user
dual unbounded → ∞ ⇒ primal infeasible

**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

**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

```latex
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

```plaintext
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
display optval;
```

... there exist better multi-start procedures

Optimization with Integer Variables

- modeling discrete choices ⇒ 0 – 1 variables
- modeling integer decisions ⇒ integer variables
e.g. number of different stocks in portfolio (8-10)
not number of beers sold at Goose Island (millions)
⇒ 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: $u(x) \overset{\text{def}}{=} -\alpha^T x + \lambda x^T Q x$
  - $\alpha$: Expected returns
  - $Q$: Variance-covariance matrix of expected returns
  - $\lambda$: Risk aversion parameter
More Realistic Models

- $b \in \mathbb{R}^{|N|}$ of “benchmark” holdings
- Benchmark Tracking: $u(x) \overset{\text{def}}{=} (x - b)^T Q(x - b)$
  - Constraint on $\mathbb{R}[\text{Return}]$: $\alpha^T x \geq r$
- 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:
    $$x_i \leq By_i \quad \forall i \in N$$
  - $\sum_{i \in N} y_i \leq K$

Even More Models

- Min Holdings: $(x_i = 0) \lor (x_i \geq m)$
  - 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: NOMAD.m; parallel APPSPACK
- solvers based on building quadratic models
- “voodoo” solvers: genetic algorithm & simulated annealing
  no convergence theory ... usually worse than deterministic
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 \((x^*, y^*)\)

\[
\begin{align*}
x^* &\in \arg\min_{x\geq 0} \ f_1(x, y^*) \\
&\text{subject to} \ c_1(x) \leq 0 \\
y^* &\in \arg\min_{y\geq 0} \ f_2(x^*, y) \\
&\text{subject to} \ c_2(y) \leq 0
\end{align*}
\]

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

Part IV

Complementarity Problems in AMPL

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

\[
\begin{align*}
\min_{x\geq 0} \ f_1(x, y^*) &\iff 0 \leq x \perp \nabla_x f_1(x, y^*) + \lambda_1^T \nabla_x c_1(x) \geq 0 \\
\text{subject to} \ c_1(x) &\leq 0 \\
\lambda_1 &\geq 0
\end{align*}
\]

\[
\begin{align*}
\min_{y\geq 0} \ f_2(x^*, y) &\iff 0 \leq y \perp \nabla_y f_2(x^*, y) + \lambda_2^T \nabla_y c_2(y) \geq 0 \\
\text{subject to} \ c_2(y) &\leq 0 \\
\lambda_2 &\geq 0
\end{align*}
\]

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
Formulation

• Firm $f \in \mathcal{F}$ chooses output $x_f$ to maximize profit
  • $u$ is the utility function
    \[ u = \left(1 + \sum_{f \in \mathcal{F}} x_f^\alpha\right)^\eta \]
  • $\alpha$ and $\eta$ are parameters
  • $c_f$ is the unit cost for each firm
• In particular, for each firm $f \in \mathcal{F}$
  \[ x_f^* \in \arg \max_{x_f \geq 0} \left( \frac{\partial u}{\partial x_f} - c_f \right) x_f \]
• First-order optimality conditions
  \[ 0 \leq x_f \perp c_f - \frac{\partial u}{\partial x_f} - x_f \frac{\partial^2 u}{\partial x_f^2} \geq 0 \]

Model: oligopoly.mod

```plaintext
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

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

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

Commands: oligopoly.cmd

```plaintext
# 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
  - $e \in \mathbb{R}^{n \times m}$ are the endowments
  - $\alpha \in \mathbb{R}^{n \times m}$ and $\beta \in \mathbb{R}^{n \times m}$ are the utility parameters
  - $p \in \mathbb{R}^m$ are the commodity prices
- Agent $i$ maximizes utility with budget constraint
  \[
  \max_{x_i, \mathbf{s} \geq 0} \sum_{k=1}^{m} \alpha_{i,k} (1 + x_{i,k})^{1-\beta_{i,k}} / (1 - \beta_{i,k})
  \]
  subject to
  \[
  \sum_{k=1}^{m} p_k (x_{i,k} - e_{i,k}) \leq 0
  \]
- Market $k$ sets price for the commodity
  \[
  0 \leq p_k \perp \sum_{i=1}^{n} (e_{i,k} - x_{i,k}) \geq 0
  \]

Model: cge.mod

```plaintext
set AGENTS; # Agents
set COMMODITIES; # Commodities
param e (AGENTS, COMMODITIES) >= 0, default 1; # Endowment
param alpha (AGENTS, COMMODITIES) > 0; # Utility parameters
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} = alpha[i,k] / (1 + x[i,k])^beta[i,k]; # Marginal prices
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

```plaintext
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;
```
Nash Games
Stackelberg Games
Introduction
Oligopoly Model
Equilibrium for Endowment Economy
Bimatrix Games

Results: cge.out

<table>
<thead>
<tr>
<th></th>
<th>Jorge</th>
<th>Sven</th>
<th>Todd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>8.9825e-01</td>
<td>2.5392e-01</td>
<td>1.8478e+00</td>
</tr>
<tr>
<td>Cars</td>
<td>1.4651e+00</td>
<td>7.2054e-01</td>
<td>8.1431e-01</td>
</tr>
<tr>
<td>Food</td>
<td>1.2021e+00</td>
<td>1.6271e+00</td>
<td>1.7081e-01</td>
</tr>
<tr>
<td>Pens</td>
<td>6.8392e-01</td>
<td>1.4787e+00</td>
<td>8.3738e-01</td>
</tr>
<tr>
<td>Books</td>
<td>2.5392e-01</td>
<td>7.2054e-01</td>
<td>8.1431e-01</td>
</tr>
<tr>
<td>Cars</td>
<td>6.6835e+00</td>
<td>6.1742e-01</td>
<td>6.1742e-01</td>
</tr>
<tr>
<td>Food</td>
<td>7.3983e+00</td>
<td>6.8345e-01</td>
<td>6.8345e-01</td>
</tr>
<tr>
<td>Pens</td>
<td>1.1081e+01</td>
<td>1.0237e+00</td>
<td>1.0237e+00</td>
</tr>
</tbody>
</table>

Results: cgenum.out

<table>
<thead>
<tr>
<th></th>
<th>Jorge</th>
<th>Sven</th>
<th>Todd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td>1.0000e+00</td>
<td>1.0825e+01</td>
<td>1.0825e+01</td>
</tr>
<tr>
<td>Cars</td>
<td>6.6835e+00</td>
<td>6.1742e-01</td>
<td>6.1742e-01</td>
</tr>
<tr>
<td>Food</td>
<td>7.3983e+00</td>
<td>6.8345e-01</td>
<td>6.8345e-01</td>
</tr>
<tr>
<td>Pens</td>
<td>1.0237e+00</td>
<td>1.0237e+00</td>
<td>1.0237e+00</td>
</tr>
</tbody>
</table>

Commands: cge.cmd

```plaintext
# 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;
```

Commands: cgenum.cmd

```plaintext
# 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;
```
Formulation

- Players select strategies to minimize loss
  - \( p \in \mathbb{R}^n \) is the probability player 1 chooses each strategy
  - \( q \in \mathbb{R}^m \) is the probability player 2 chooses each strategy
  - \( A \in \mathbb{R}^{n \times m} \) is the loss matrix for player 1
  - \( B \in \mathbb{R}^{n \times m} \) is the loss matrix for player 2
- Optimization problem for player 1
  \[
  \min_{0 \leq p \leq 1} \quad p^T A q \\
  \text{subject to} \\
  e^T p = 1
  \]
- Optimization problem for player 2
  \[
  \min_{0 \leq q \leq 1} \quad p^T B q \\
  \text{subject to} \\
  e^T q = 1
  \]
- Complementarity problem
  \[
  0 \leq p \leq 1 \perp A q - \lambda_1 \\
  0 \leq q \leq 1 \perp B^T p - \lambda_2
  \]

Model: bimatrix1.mod

```plaintext
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

```plaintext
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] >= 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;
  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

```plaintext
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] >= 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;
```

Leyffer, Moré, and Munson
Computational Optimization
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
    
    $$ x_i^* \in \arg \min_{x_i \geq 0} \ f_i(x, y) $$
    subject to $c_i(x_i) \leq 0$

- Leader solves optimization problem with equilibrium constraints
  
  $$ \min_{y \geq 0, x, \lambda} g(x, y) $$
  subject to
  
  $$ h(y) \leq 0 $$
  
  $$ 0 \leq x_i \perp \nabla x_i f_i(x, y) + \lambda_i^T \nabla x_i c_i(x_i) \geq 0 $$
  
  $$ 0 \leq \lambda_i \perp -c_i(x_i) \geq 0 $$

- Many applications in economics
  - Optimal taxation
  - Tolling problems

Penalization Approach

$$ \min_{x, y, \lambda, s, t \geq 0} \ g(x, y) + \sum_i \big( s_i^T x_i + \lambda_i t_i \big) $$
subject to

$$ h(y) \leq 0 $$

$$ s_i = \nabla x_i f_i(x, y) + \lambda_i^T \nabla x_i c_i(x_i) $$

$$ t_i = -c_i(x_i) $$

- Optimization problem satisfies constraint qualification
- Need to increase $\pi$

Nonlinear Programming Formulation

$$ \min_{x, y, \lambda, s, t \geq 0} \ g(x, y) $$
subject to

$$ h(y) \leq 0 $$

$$ s_i = \nabla x_i f_i(x, y) + \lambda_i^T \nabla x_i c_i(x_i) $$

$$ t_i = -c_i(x_i) $$

$$ \sum_i \big( s_i^T x_i + \lambda_i t_i \big) \leq 0 $$

- 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
Relaxation Approach

\[
\begin{align*}
\min_{x,y,\lambda,s,t \geq 0} & \quad g(x, y) \\
\text{subject to} & \quad h(y) \leq 0 \\
& \quad s_i = \nabla x_i f_i(x, y) + \lambda^T \nabla x_i c_i(x_i) \\
& \quad t_i = -c_i(x_i) \\
& \quad \sum_i (s_i^T x_i + \lambda_i t_i) \leq \tau
\end{align*}
\]

- Need to decrease \( \tau \)

Model Formulation

- Economy with \( n \) agents and \( m \) commodities
  - \( e \in \mathbb{R}^{n \times m} \) are the endowments
  - \( \alpha \in \mathbb{R}^{n \times m} \) and \( \beta \in \mathbb{R}^{n \times m} \) are the utility parameters
  - \( p \in \mathbb{R}^m \) are the commodity prices
- Agent \( i \) maximizes utility with budget constraint
  \[
  \max_{x_i, \ast \geq 0} \sum_{k=1}^m \alpha_{i,k} (1 + x_{i,k})^{1 - \beta_{i,k}} \left( 1 - \frac{1}{1 - \beta_{i,k}} \right)
  \]
  \[
  \text{subject to} \quad \sum_{k=1}^m p_k (x_{i,k} - e_{i,k}) \leq 0
  \]
- Market \( k \) sets price for the commodity
  \[
  0 \leq p_k \perp \sum_{i=1}^n (e_{i,k} - x_{i,k}) \geq 0
  \]
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

Stackelberg Nash Game

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 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;

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

<table>
<thead>
<tr>
<th>Iter</th>
<th>Primal</th>
<th>Dual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Obj Value</td>
<td>Infeas</td>
</tr>
<tr>
<td>1</td>
<td>1.000000e+00</td>
<td>0.0e+00</td>
</tr>
<tr>
<td>2</td>
<td>6.902180e-01</td>
<td>2.2e-01</td>
</tr>
<tr>
<td>3</td>
<td>2.773222e-01</td>
<td>1.6e-01</td>
</tr>
<tr>
<td>292</td>
<td>-8.213292e-05</td>
<td>1.7e-09</td>
</tr>
<tr>
<td>293</td>
<td>-8.202525e-05</td>
<td>1.7e-09</td>
</tr>
<tr>
<td>294</td>
<td>nan</td>
<td>nan</td>
</tr>
<tr>
<td>500</td>
<td>nan</td>
<td>nan</td>
</tr>
</tbody>
</table>

FILTER Output

| iter | rho | ||d|| | f / hJ | ||c||/hJt |
|-----|-----|--------|--------|--------|--------|
| 0:0 | 10.0000 | 0.000000 | 1.000000 | 0.000000 |
| 1:1 | 10.0000 | 1.000000 | 0.000000 | 0.000000 |
| 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 |

Leyffer, Moré, and Munson

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

Leyffer, Moré, and Munson

Computational Optimization