Part I

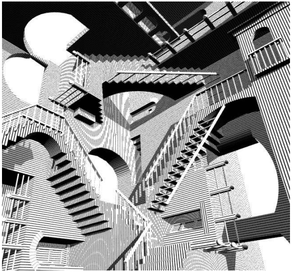
Introduction, Applications, and Formulations

Outline: Six Topics

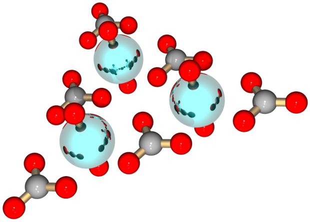
- ◇ Introduction
- ◇ Unconstrained optimization
  - Limited-memory variable metric methods
- ◇ Systems of Nonlinear Equations
  - Sparsity and Newton's method
- ◇ Automatic Differentiation
  - Computing sparse Jacobians via graph coloring
- ◇ Constrained Optimization
  - All that you need to know about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

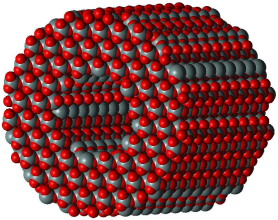
Topic 1: The Optimization Viewpoint

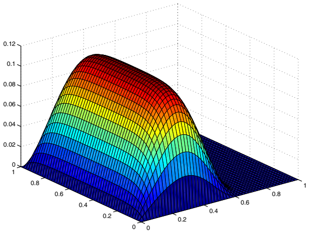
- Modeling
- Algorithms
- Software
- Automatic differentiation tools
- Application-specific languages
- High-performance architectures

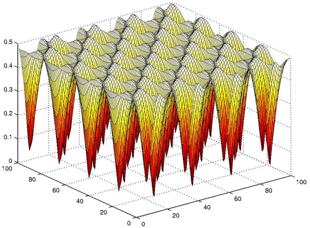


View of Optimization from Applications









Classification of Constrained Optimization Problems

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

- 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 $\nabla_x^2 \mathcal{L}(x, \lambda) = \nabla^2 f(x) + \sum_{k=1}^m \nabla^2 c_k(x) \lambda_k$

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

$$S_{t+1} = (1+r)S_t + w_{t+1} - c_{t+1}, \qquad 0 \le t < T$$

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?

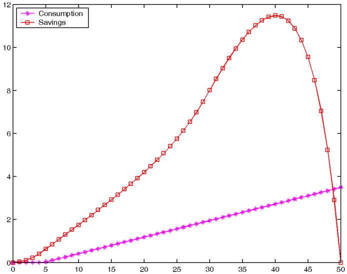
Constrained Optimization Software: IPOPT

- Formulation

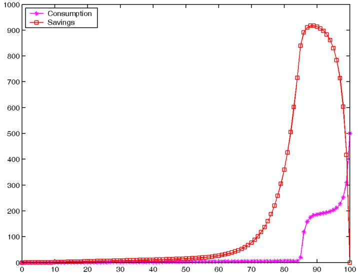
$$\min \{f(x) : x_l \le x \le x_u,\ c(x) = 0\}$$

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

Life-Cycles Saving Problem: Results



$(R, T) = (30, 50)$



$(R, T) = (60, 100)$

Question. Problem formulation to results: How long?

Topic 2: Unconstrained Optimization

Augustin Louis Cauchy (August 21, 1789 – May 23, 1857)
Additional information at **Mac Tutor**
`www-history.mcs.st-andrews.ac.uk`

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

$$\|\nabla f(x_k)\| \le \tau$$

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

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

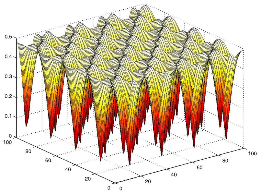
Exercise. Prove this result.

Ginzburg-Landau Model

Minimize the Gibbs free energy for a homogeneous superconductor

$$\int_{\mathcal{D}} \left\{ -|v(x)|^2 + \tfrac{1}{2}|v(x)|^4 + \|[\nabla - iA(x)]\, v(x)\|^2 + \kappa^2 \, \|(\nabla \times A)(x)\|^2 \right\} dx$$

$v : \mathbb{R}^2 \to \mathbb{C}$ (order parameter)
$A : \mathbb{R}^2 \to \mathbb{R}^2$ (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 ε_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 α_k is determined by a line search along p_k.

Line searches

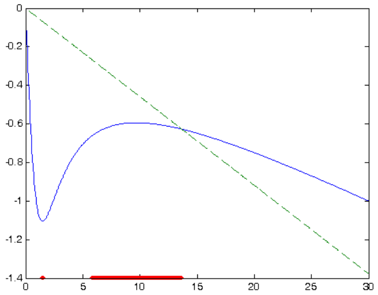
- Geometry-based: Armijo, ...
- Model-based: Quadratics, cubic models, ...

Powell-Wolfe Conditions on the Line Search

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

$$f(x + \alpha p) \le f(x) + \mu\, \alpha \nabla f(x_k)^T p_k \qquad \text{sufficent decrease}$$

$$|\nabla f(x + \alpha p)^T p| \le \eta\, |\nabla f(x)^T p| \qquad \text{curvature condition}$$



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 α_k is determined by a line search.

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

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

$$\beta_k^{PR} = \frac{\langle g_{k+1}, g_{k+1} - g_k \rangle}{\|g_k\|^2}, \qquad \text{Polak-Rivière}$$

$$\beta_k^{PR+} = \max \left\{ \beta_k^{PR}, 0 \right\}, \qquad \text{PR-plus}$$

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 α_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

Library of Congress

Sir Isaac Newton (January 4, 1643 – March 331, 1727)
Additional information at **Mac Tutor**
`www-history.mcs.st-andrews.ac.uk`

Motivation

Give 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

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

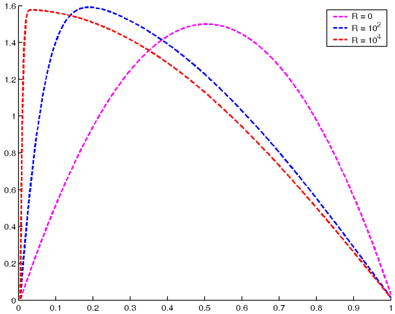
Computational Issues

- 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?

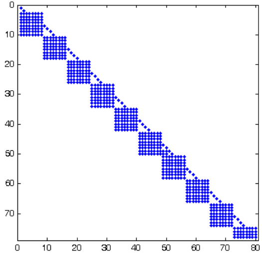
Flow in a Channel Problem

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 ρ_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 ρ_A operations, where $\rho_A = \text{avg}\{\rho_i^2\}$.
- ◇ Graph coloring techniques (see Topic 4) can compute or approximate the Jacobian matrix with ρ_M function evaluations where $\rho_M = \max\{\rho_i\}$

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:

$$T\{f'(x)\} \leq \Omega_T \, T\{f(x)\}, \qquad M\{\nabla f(x)\} \leq \Omega_M \, M\{f(x)\}$$

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 $f'(x)v$.

$$f'(x) = \begin{pmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}, \qquad v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

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

$$V = (v_1, v_2, \ldots, v_p)$$

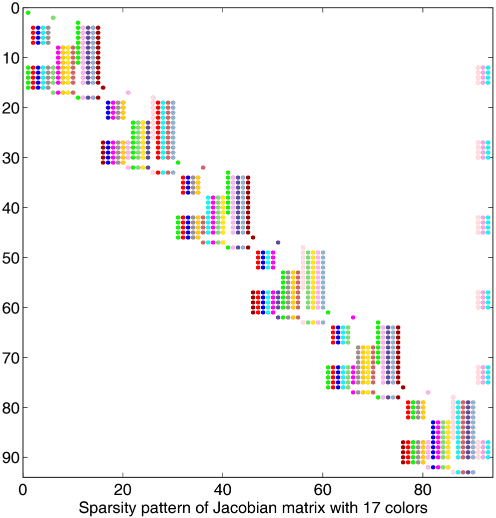
Compute $f'(x)$ from the compressed Jacobian matrix $f'(x)V$.

Observation. In practice $p \approx \rho_M$ where

$$\rho_M \equiv \max\{\rho_i\},$$

and ρ_i is the number of non-zeros in the i-th row of $f'(x)$.

Coloring the Jacobian matrix with $p = 17$ colors



Sparsity Pattern of the Jacobian Matrix

Optimization software tends to require the **closure** of the sparsity pattern

$$\bigcup \left\{ \mathcal{S}(f'(x)) : x \in \mathcal{D} \right\}.$$

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,

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

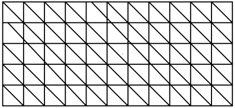
Partially Separable Functions

The mapping $f : \mathbb{R}^n \to \mathbb{R}$ 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 $f : \mathbb{R}^n \to \mathbb{R}$ has a sparse Hessian matrix then f is partially separable.



Optimization problems with a finite element formulation usually associate f_i with each element.

Partially Separable Functions: The Trick

If $f : \mathbb{R}^n \to \mathbb{R}$ 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,

$$f(x) = f_E(x)^T e \qquad \Longrightarrow \qquad \nabla f(x) = {f_E}'(x)^T e$$

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.

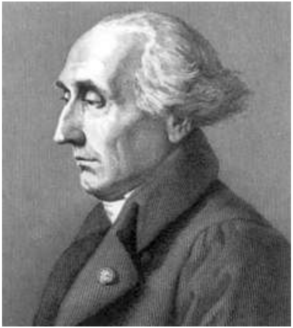
$$T\{\nabla f(x)\} = \kappa \ \rho_M \max T\{f(x)\}$$

Quartiles of κ

$2,500 \leq n \leq 40,000$

1.3	2.9	5.0	8.2	22.2

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 $\Omega = \{x \in \mathbb{R}^n : c(x) \geq 0\}$, define

$$C(x^*) = \left\{ w : w = \sum_{i=1}^{m} \lambda_i \left(-\nabla c_i(x^*) \right), \; \lambda_i \geq 0 \right\}$$

In general $C(x^*) \subset N(x^*)$, and under a **constraint qualification**

$$C(x^*) = N(x^*)$$

Hence, for some multipliers $\lambda_i \geq 0$,

$$\nabla f(x) = \sum_{i=1}^{m} \lambda_i \nabla c_i(x), \qquad \lambda_i \geq 0,$$

Constraint Qualifications

In the case where

$$\Omega = \{x \in \mathbb{R}^n : l \le c(x) \le u\}$$

the main two constraint qualifications are

Linear independence

The active constraint normals are positively linearly independent, that is, if

$$C_{\mathcal{A}} = (\nabla c_i(x) : c_i(x) \in \{l_i, u_i\})$$

then $C_{\mathcal{A}}$ has full rank.

Mangasarian-Fromovitz

The active constraint normals are positively linearly independent.

Lagrange Multipliers

For the general problem with 2-sided constraints

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

the KKT conditions for a local minimizer are

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

where the multipliers satisfy complementarity conditions

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

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

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

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

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 $c(x) = 0$,

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

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

Newton's method for this system can be written as

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

where

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

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

$$C^T u = 0, \ u \neq 0, \qquad \Longrightarrow \qquad u^T H u > 0$$

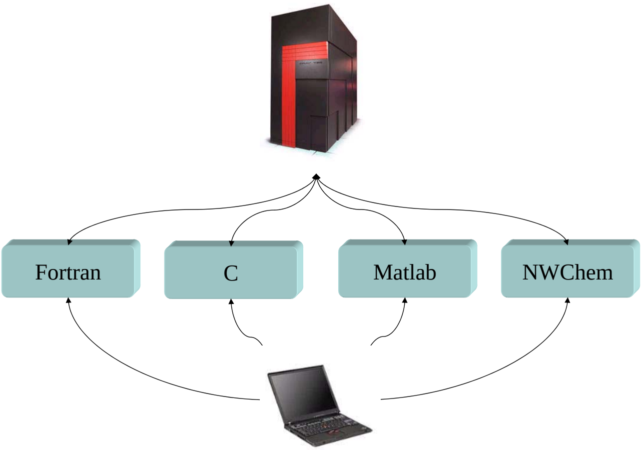
then A is nonsingular.

Topic 6: Solving Optimization Problems

Environments

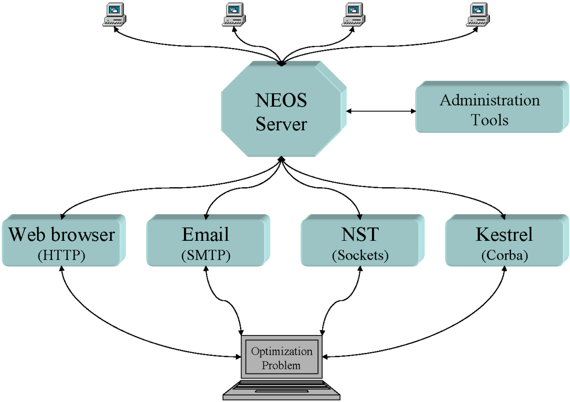
- Modeling Languages: AMPL, GAMS
- NEOS

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

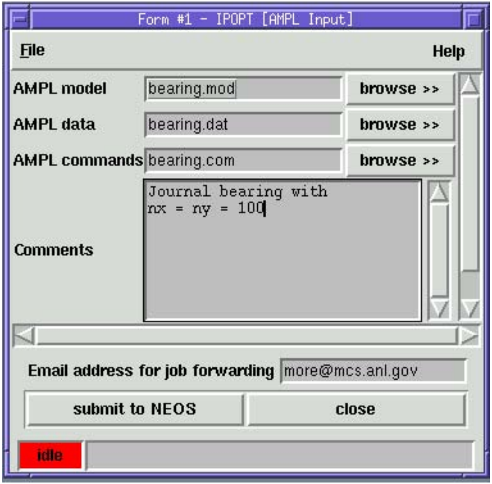
Research Issues for NEOS

- ◇ 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
  - Universitá di Roma *La Sapienza*, Italy
  - Wisconsin University

Solving Optimization Problems: NEOS Interfaces

Interfaces

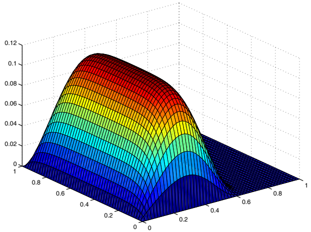
- Kestrel
- NEOS Submit
- Web browser
- Email



Pressure in a Journal Bearing

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

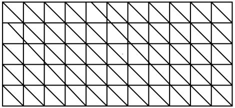
$$w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3$$
$$w_l(\xi_1, \xi_2) = \epsilon \sin \xi_1$$
$$\mathcal{D} = (0, 2\pi) \times (0, 2b)$$



Number of active constraints depends on the choice of ϵ 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

```
var v {i in 0..nx+1, 0..ny+1} >= 0;

minimize q:
  0.5*(hx*hy/6)*sum {i in 0..nx, j in 0..ny}
    (wq[i] + 2*wq[i+1])*
    (((v[i+1,j]-v[i,j])/hx)^2 + ((v[i,j+1]-v[i,j])/hy)^2) +
  0.5*(hx*hy/6)*sum {i in 1..nx+1, j in 1..ny+1}
    (wq[i] + 2*wq[i-1])*
    (((v[i-1,j]-v[i,j])/hx)^2 + ((v[i,j-1]-v[i,j])/hy)^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

```
# Set the design parameters

param b := 10;
param e := 0.1;

# Set parameter choices

let nx := 50;
let ny := 50;

# Set the starting point.

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

```
option show_stats 1;

option solver "knitro";
option solver "snopt";
option solver "loqo";
option loqo_options "outlev=2 timing=1 iterlim=500";

model;
include bearing.mod;

data;
include bearing.dat;

solve;

printf {i in 0..nx+1,j in 0..ny+1}: "%21.15e\n", v[i,j] > cops.dat;
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

$$S_{t+1} = (1 + r)S_t + w_{t+1} - c_{t+1}, \qquad 0 \le t < T$$

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 $t \ge R$.

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;
```