Introduction to Optimization

Ken Judd\textsuperscript{1}
July 28, 2008

\textsuperscript{1}The nice slides were created by Sven Leyffer, Jorge More, and Todd 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
Topic 1: The Optimization Viewpoint

- Modeling
- Algorithms
- Software
- Automatic differentiation tools
- Application-specific languages
- High-performance architectures
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 \( \nabla^2_x L(x, \lambda) = \nabla^2 f(x) + \sum_{k=1}^{m} \nabla^2 c_k(x) \lambda_k \)
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
www-history.mcs.st-andrews.ac.uk
Unconstrained Optimization: Background

Given a continuously differentiable \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) 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 \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is continuously differentiable and bounded below, then there is a sequence \( \{x_k\} \) such that

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

**Exercise.** Prove this result.
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

\[
\begin{align*}
f(x + \alpha p) & \leq f(x) + \mu \alpha \nabla f(x_k)^T p_k \quad \text{sufficient decrease} \\
|\nabla f(x + \alpha p)^T p| & \leq \eta |\nabla f(x)^T p| \quad \text{curvature condition}
\end{align*}
\]
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 ($g_k = \nabla f(x_k)$):

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

$$\beta^{PR}_k = \frac{\langle g_{k+1}, g_{k+1} - g_k \rangle}{\|g_k\|^2}$$, \hspace{1cm} \text{Polak-Rivi\`ere}$$

$$\beta^{PR+}_k = \max \{ \beta^{PR}_k, 0 \}$$, \hspace{1cm} \text{PR-plus}$$
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 (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

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

**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?
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 $\rho_M = \max\{\rho_i\}$.
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:**

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

where $T\{\cdot\}$ is computing time and $M\{\cdot\}$ is memory.
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 \( \Omega \), 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 $\lambda_i \geq 0$,

$$\nabla f(x) = \sum_{i=1}^{m} \lambda_i \nabla c_i(x), \quad \lambda_i \geq 0,$$
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 $c_i(x) \not\in \{l_i, u_i\}$
- $\lambda_i \geq 0$ 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

\[
\begin{pmatrix}
\nabla^2_x \mathcal{L}(x, \lambda) & -\nabla c(x) \\
\nabla c(x)^T & 0
\end{pmatrix}
\begin{pmatrix}
s_x \\
s_\lambda
\end{pmatrix}
= -
\begin{pmatrix}
\nabla_x \mathcal{L}(x, \lambda) \\
c(x)
\end{pmatrix}
\]
Topic 6: Solving Optimization Problems

**Environments**

- Modeling Languages: AMPL, GAMS
- NEOS
The Classical Model

Fortran
C
Matlab
NWChem
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
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

\begin{verbatim}
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;
\end{verbatim}

Leyffer, Moré, and Munson
Computational Optimization
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

```plaintext
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;
```