Optimization: Methods

Judd Chapter 4

David Childers (thanks to Y. Kryukov, K. Judd, and U. Doraszelski)

CMU, Tepper School of Business

March 22, 2022

The Plan

- Unconstrained optimization
  - Newton: Univariate & Multivariate
  - Direction methods
  - Derivative-free methods: golden section & polytope
- Constrained optimization
  - Penalty function
  - Kuhn-Tucker conditions:
    GZ and Active set methods
- Structured Problems
- Code

General points

- Optimization is central to Economics
  - Almost every economic decision is represented as solution to an optimization problem
  - Selfless behavior: add welfare of others to the objective
- As with nonlinear eq-ns, issue with uniqueness and local solutions
  - Uniqueness $\Leftarrow$ global concavity, or quasi-concavity
  - Local solutions: find all and pick the best
- Solving FOC is viable (Newton method), but:
  - Check SOC to avoid wrong kind of extremum, or inflection points
  - Newton requires second derivative of objective
  - Value of objective measures quality of current guess

Newton's method – Univariate

$$\min_{x \in \mathbb{R}} f(x)$$

assume $f(x)$ is C^3 (3x continuously differentiable).

- Use Newton's method to solve F.O.C. of the problem.
- Iterate on:
$$x^{k+1} = x^k - f'(x^k)/f''(x^k)$$
- Must check S.O.C. after convergence $(f''(x^k) > 0)$
- Will only find local extrema
    - Need global convexity to find global min
    - Or quasi-convexity: $f'(x^k) = 0 \Rightarrow f''(x^k) > 0$
    - Otherwise, try different (random) starting values
- Converges quadratically to critical point (under same conditions as last time)

Multivariate Newton

$f : \mathbb{R}^n \to \mathbb{R}, \in C^3$

- Iterate on:

$$x^{k+1} = x^k - H(x^k)^{-1}\nabla f(x^k)'$$

  $\nabla f(x) \in \mathbb{R}^n$ – gradient of f (first derivative)
  $H(x^k)$ – Hessian of f (second derivative)

- Stopping rules:
  - If $||x^{k+1} - x^k|| < \epsilon(1 + ||x^k||)$, go to next point.
  - If $||\nabla f(x^k)|| < \delta(1 + |f(x^k)|)$, stop and report success; otherwise stop and report failure.
  - ε and δ should exceed square root of machine epsilon.
- Computing Hessian by finite diff. $= n^2$ evaluations of f

Direction & line search method

A sequence of univariate optimization problems

1. Compute the search (step) direction s^k:
    - Coordinate directions: Cycle through unit basis vectors $\{e^j\}_{j=1}^n$.
    - Steepest descent: $s^k = -\nabla f(x^k)'$.
    - Newton with line search: $s^k = -H(x^k)^{-1}\nabla f(x^k)'$.
2. Compute the step length λ^k from a univariate problem:

$$\lambda^k = \arg\min_\lambda f(x^k + \lambda s^k).$$

3. Set $x^{k+1} = x^k + \lambda^k s^k$.
4. If $||x^{k+1} - x^k|| < \epsilon(1 + ||x^k||)$, go to step 5; o/w go to 1.
5. If $||\nabla f(x^k)|| < \delta(1 + |f(x^k)|)$, report success; o/w, failure.

- Under conditions, can ensure guaranteed convergence to stationary point + comparable rate to Newton

Advanced direction choice methods

Trust Region methods: alternative to line search

- Choose direction & distance subject to sequence of constraints

Quasi-Newton methods:

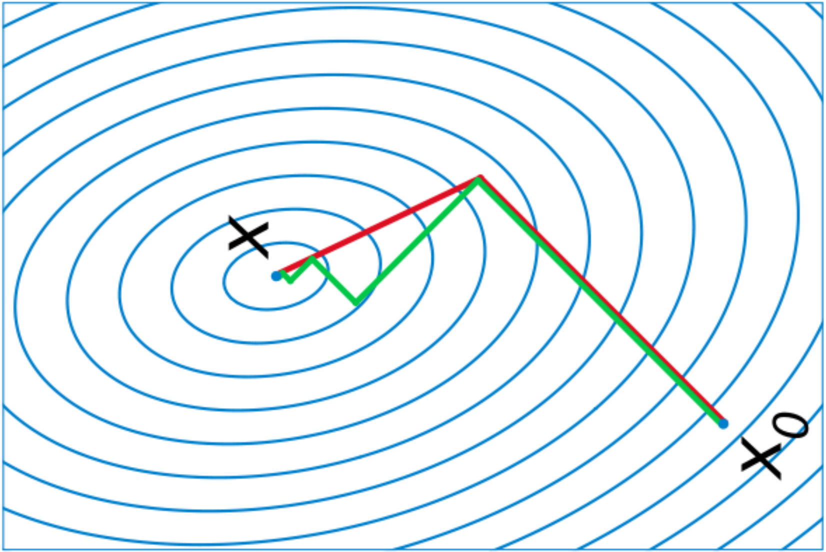
- Iteratively update Hessian H^{k+1} instead of computing it
- Use H^k, $\nabla f(x^{k+1})$, $\nabla f(x^k)$, x^{k+1}, x^k
- Methods: DFP, BFGS, Limited-memory (L-)BFGS
    - BFGS like Broyden, $O(n^2)$ per update but superlinear rate
    - L-BFGS stores only $m < n$ Broyden updates: $O(nm)$ per iterate
- Hessian does not always to converge to the true one

Conjugate gradients

- Do not compute or store the Hessian
- Dampen the steepest-descent direction:

$$s^{k+1} = -\nabla f(x^{k+1})' + \frac{||\nabla f(x^{k+1})||^2}{||\nabla f(x^k)||^2} s^k$$

Steepest descent vs. Conjugate gradient



Golden section method (univariate)

Solving $\max_x f(x)$, where $f : [\underline{x}, \bar{x}] \to \mathbb{R}^1$ is quasi-concave

1. Set $A^0 = \underline{x}$, $D^0 = \bar{x}$ (or an interval containing the max)
   $B^0 = \varphi A^0 + (1-\varphi) D^0$, $C^0 = (1-\varphi) A^0 + \varphi D^0$;
   evaluate $f_B = f\left(B^0\right)$, $f_C = f(C^0)$; set $k = 0$
   - $\varphi = \left(\sqrt{5}-1\right)/2$ solves $(1-\varphi)/\varphi = \varphi$
2. - If $f_B > f_C$: $A^{k+1} = A^k$, $D^{k+1} = C^k$, $C^{k+1} = B^k$, $f_C = f_B$;
     $B^{k+1} = (1-\varphi) A^{k+1} + \varphi C^{k+1}$, $f_B = f\left(B^{k+1}\right)$.
   - If $f_B < f_C$: $A^{k+1} = B^k$, $D^{k+1} = D^k$, $B^{k+1} = C^k$, $f_C = f_B$
     $C^{k+1} = \varphi B^{k+1} + (1-\varphi) D^{k+1}$, $f_C = f\left(C^{k+1}\right)$
3. If $\dfrac{\left|D^{k+1} - A^{k+1}\right|}{1 + \left|B^{k+1}\right|} < \delta$, report B^{k+1} as the max

- Linear convergence at rate $\frac{1}{\varphi}$.

Polytope (Nelder-Mead, "Amoeba" method) (multivariate)

Initialization: Choose initial simplex $\{x^1, \dots, x^{n+1}\}$

1. Reorder vertices such that $f(x^i) \geq f(x^{i+1})$, $i = 1, \dots, n$.
2. Find the lowest i such that $f(x^i) > f(y^i)$, where

   $$y^i = x^i + 2\left(\frac{1}{n}\sum_{j \neq i} x^j - x^i\right)$$

   is the reflection of x^i through the opposing face.
   If found, set $x^i = y^i$ and go to step 1; o/w go to step 3.
3. If width of simplex is less than ε, stop; o/w go to step 4.
4. Replace x^i with $\frac{1}{2}\left(x^i + x^{n+1}\right)$, $i = 1, \dots, n$, i.e., shrink the simplex toward x^{n+1}, and go to step 1.

- Guaranteed to converge to a local minimum if f is continuous.

Constrained optimization

$$\begin{aligned} & \min_x f(x) \\ \text{subject to:} \quad & g(x) = 0 \\ & h(x) \leq 0 \end{aligned}$$

- Uniqueness of solution requires:
  - Convexity of objective (concavity for $\max$)
  - Convexity of the feasible set:

$$\{x : g(x) = 0, h(x) \leq 0\}$$

e.g. via (quasi-)convexity of $h(x)$

- Equality constraints could be substituted into objective
  - But (nearly-)linear contraints could be easier to solve than unconstrained but highly nonlinear problem
- Can pick $f(x)$ to ensure interior solution

Penalty function method

Permit anything, but penalize violations:

$$\min_x f(x) + P\left(\sum_i g^i(x)^2 + \sum_j \max(0, h^j(x))^2\right),$$

where $P > 0$ is the penalty parameter.

- Solve repeatedly for an increasing sequence of P^k's.
- Use solution (and Hessian from P^k) as the starting point for P^{k+1}.
- Continue until constraint violations are small enough
- Beware limited function domains: $\sqrt{x}$, $\log(x)$, etc.

Karush-Kuhn-Tucker Theorem

- Define the Lagrangian:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$$

- Local minimum should satisfy:

$$\frac{\partial \mathcal{L}}{\partial x} \equiv f_x + \lambda^T g_x + \mu^T h_x = 0$$
$$g(x) = 0$$
$$\mu_i h^i(x) = 0$$
$$h^i(x) \leq 0, \mu_i \geq 0$$

  - Last two lines are called *complementary slackness* conditions
- If binding constraints are linearly independent, $\{\lambda, \mu\}$ are unique

Solving KKT conditions: brute force

- We have no way to solve systems with inequalities in them

Kuhn-Tucker approach:

- $\mathcal{J} = \{1, 2, ..., m\}$ – set of all inequality constraints
- $\mathcal{P} \subset \mathcal{J}$ – set of binding constraints:

$$\begin{aligned} i &\in \mathcal{P} : h^i(x) = 0, \mu_i \geq 0, \\ i &\notin \mathcal{P} : h^i(x) \leq 0, \mu_i = 0, \end{aligned}$$

1. Go though for every possible $\mathcal{P}$,
   solve equality conditions as a system of equations
2. Drop solutions that violate remaining inequality conditions
3. Report solution with the best objective

Guaranteed to find solution, but computationally intensive

Solving KKT: GZ transformation

Zangwil-Garcia approach:

1. For every ineqality constraint ($h_i(x) \leq 0$), introduce an unconstrained variable ξ_i.
2. Replace complementary slackness conditions with:

$$\begin{aligned} h^i(x) + [\max\{0, \xi_i\}]^2 &= 0 \\ \mu_i - [\max\{0, -\xi_i\}]^2 &= 0 \end{aligned}$$

3. We have system of equations only
    - And they are differentiable

- Can replace μ_i by $[\max\{0, -\xi_i\}]^2$, reducing # of variables

Active set methods

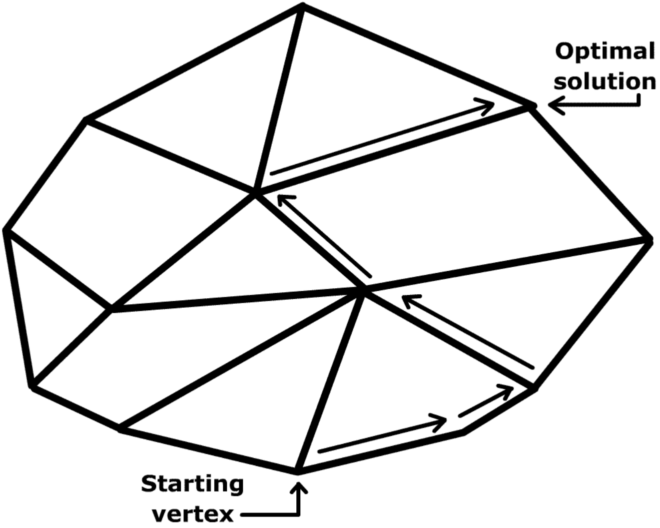
Iteratively improve $\mathcal{P}^k$ – set of binding inequality constraints

1. Pick initial guess $\mathcal{P}^0$
2. Solve using only the constraints in $\mathcal{P}^k$, e.g. using Penalty method
3. Drop constraints that do not bind from $\mathcal{P}^k$
4. Add constraints that are violated
5. If no changes in $\mathcal{P}^k$, increase the penalty

Linear problem and constraints $\Rightarrow$ **Simplex** method

- There must be n binding constraints – optimum is a vertex of the feasible set.
- Replace one constraint at a time, i.e. move from one vertex to another, along the edges of the facets.
- Always move in the direction that improves objective

Illustration of Simplex method



Other methods

- Interior point methods: like penalization, but *inside* feasible set
    - Provides very strong guarantees for convex problems
- Sequential quadratic programming method:
  Replace objective by a quadratic approximation,
  and binding constraints by a linear approximation.
- Reduced gradient method: Use binding constraints to solve for "dependent variables" in terms of "independent variables."
- Randomized methods: useful for avoiding local optima
    - genetic algorithms, simulated annealing, stochastic gradient descent

Exploiting Structure

- Known structure allows specialized methods, stronger performance
- **Linear Programs** Linear objective, linear constraints
  - 0 curvature means Newton (etc) fails, but well known solutions
  - Simplex gives fast exact solutions for well-conditioned problems
  - Interior point methods give ϵ-approx solutions but more robust
- **Quadratic Programs** Quadratic objective linear constraints
- **(Mixed) Integer Programs** Discrete objectives, or mixed discrete and linear or discrete and quadratic
  - Exponentially (NP) hard in general
  - Advanced methods feasible, fast for medium-sized cases
- Structured problems can often be fruitfully combined, and advanced solvers may be able to detect and exploit this

Optimization in Econometrics/Statistics

- Econometric problems often have special structure, unique goals
- (Nonlinear) Least squares/MLE: use information matrix equality
  - Replace Hessian by inner product of Score for 2nd order method
- Due to data randomness, exact optimization not always ideal
  - Randomized or inexact solvers may have *better* performance
- Consider M-estimator: $\hat{\theta} = \underset{\theta \in \Theta}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} m(x_i, \theta)$
- Stochastic Gradient Descent (Robbins Monro 1951)
  - Choose one data point x_i at random each step
  - Decrease gradient $\theta^{k+1} = \theta^k - \lambda^k \nabla_\theta m(x_i, \theta^k)$
  - If $\sum_{k=1}^{\infty} \lambda^k = \infty$, $\sum_{k=1}^{\infty} (\lambda^k)^2 < \infty$, sequence converges
  - n times faster per step, so ideal for big data and deep learning
  - Many helpful variants and step-size selection methods exist

Python solvers: `scipy.optimize`

Single nonlinear equation ($f(x) = 0$):

`root_scalar(lambda x: np.sin(x),x0=3,fprime=lambda x: np.cos(x), method='newton')`

- `f` is function, `lambda x: f(x)` if $f(x)$ defined inline
- `fprime`, `fprime2` take 1st, second derivatives
- `x0` is initial guess: use `bracket=[a,b]` for bracketing methods
- `method` can be `newton`, `brentq`, `secant`, etc

System of nonlinear equations:

`root(f,x0,jac=j,method:=hybr)`

- `f` is function, `j` is Jacobian (manual or automatic)
- `x0` initial vector
- `method` can be `hybr` (Powell), `broyden`, `anderson` (fixed point with acceleration)
- Options for absolute/relative tolerance, algorithm parameters like acceleration rate, etc

Python optimizers: `scipy.optimize`

Univariate optimization:

`minimize_scalar(f,bracket,bounds,method)`

- Defaults `bounded` Brent if bounds provided, regular `Brent` o/w, also `Golden`

Multivariate optimization:

`minimize(f,x0,jac,hess,bounds,constraints,method)`

- Defaults `BFGS`. Pass jacobian and hessian for first, second order methods, bounds or constraint functions.

Optimizers for data

- `torch`, `tensorflow` have SGD and variants (Adam, Adagrad)
- Use data-loaders to evaluate on small "batch" of data points per iterate

Julia solvers: `using Roots, NLsolve`

Single nonlinear equation ($f(x) = 0$):

`fzero(x->sin(x),3,order=1)`

- `x->f(x)` is $f(x)$ defined inline. Call f for named function
- `order` 0 for bisection, 1 for secant
- 3 is initial guess: use `(a,b)` for bracketing methods

System of nonlinear equations:

`nlsolve(f,j,x0,method:=trust_region)`

- `f` is function, `j` is Jacobian (manual or automatic)
- `x0` initial vector
- `method` can be `trust_region`, `newton` (Newton with linesearch) or `anderson` (fixed point with acceleration)
- Options for absolute/relative tolerance, algorithm parameters like acceleration rate, etc

Julia optimizers: `using Optim`

Unconstrained optimization:

`optimize(f,g!,h!,x0,Optim.Options)`

- Defaults `Brent` if f univariate, `NelderMead` if no derivatives, `LBFGS()` if gradient, `newton` (with line search) if grad & hessian
- `g!`, `h!` in-place gradient/hessian, or set `autodiff:=forward`
- Also `ConjugateGradient()`, `GoldenSection()`, etc

Constrained optimization:

```
df=TwiceDifferentiable(f,g!,h!)
dc=TwiceDifferentiableConstraints(c,cj!,ch!,lx,lu,lc,uc)
optimize(df, dc, x0, IPNewton())
```

- minimum of `f(x)`, subject to $lx \leq x \leq lu$, $lc \leq c(x) \leq uc$
- Interior Point Newton, using function and constraint derivatives

Implementation: Matlab solvers

Single nonlinear equation ($f(x) = 0$):

`X = fzero(@MyFun,3,optimset('Display','iter'))`

- `MyFun(x)` is $f(x)$, "`@`" is function handle operator
- 3 is the initial guess (required input)
- `optimset` (optional) – sets parameters:
  - 'Display' = 'iter' means show output for each iteration
  - `'TolX'=1e-7` is termination tolerance for $\left\|x^{k+1} - x^k\right\|$ stopping criterion
  - `'TolFun'` = tolerance for $\left\|f(x^{k+1})\right\|$ criterion
- `[x,fval,exitflag] = fsolve(...)`
  - `fval` – final value of $f(x)$
  - `exitflag` = 1 if solution is found, < 0 if failed
- Read help for more options

More Matlab solvers

System of nonlinear equations:

```
[x,fval] = fsolve(@myfun,x0,options)
```

- `function F = myfun (x)` : takes x, returns $F(x)$
- `myfun` can also compute the Jacobian matrix (read help)

Unconstrained optimization:

- `fminsearch(@myfun,x0,options)`
- `fminunc(@myfun,x0,options)`

Constrained optimization:

```
x = fmincon(@myfun,x0,A,b,Aeq,beq,lb,ub,@mycon)
```

- minimum of `myfun`, subject to
- $Ax \leq b$, $A_{eq}x = b_{eq}$
- $lb \leq x \leq ub$
- $c(x) \leq 0, c_{eq}(x) = 0$ defined by:
  `function [c,ceq] = mycon(x)`

Conclusions

- Generic optimizers exist, but problem knowledge and structure can help immensely
- Optimization is an active area of computational research.
- We should take advantage of existing methods and software rather than developing our own.
- For serious problems, use structured optimization modeling languages and specialized solvers
  - AMPL or JuMP let you switch out specialized solvers
  - For hard problems like integer programs, solvers like Gurobi millions of times faster
- See code examples in Jupyter notebook

Recommended References

- General Optimization
  - Judd Chapter 4.
  - Nocedal, Jorge and Stephen J. Wright. Numerical Optimization 2nd ed. 2006
    - Standard reference on classic material, available online
- Advanced Methods
  - Ben Tal, Aharon and Arkadi Nemirovski. Lectures on Modern Convex Optimization 2001 `https://www2.isye.gatech.edu/~nemirovs/LMCOBookSIAM.pdf`
  - Bubeck, Sebastien. Convex Optimization: Algorithms and Complexity 2015 `https://arxiv.org/abs/1405.4980`
- For fun: history and politics of numerical optimization
  - Shalizi, Cosma. "In Soviet Union, Optimization Problem Solves *You*" 2012 `http://crookedtimber.org/2012/05/30/in-soviet-union-optimization-problem-solves-you/`