Numerical Methods in Economics
MIT Press, 1998

Notes for Lecture 4: Unconstrained Optimization

February 26, 2020

Optimization Problems

- Canonical problem:

$$\begin{aligned} &\min_x f(x) \\ &s.t.\ \ g(x) = 0, \\ &\qquad h(x) \leq 0, \end{aligned}$$

  - $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*
  - $g : \mathbb{R}^n \to \mathbb{R}^m$ is the vector of m *equality constraints*
  - $h : \mathbb{R}^n \to \mathbb{R}^\ell$ is the vector of ℓ *inequality constraints*.

- Examples:

  - Maximization of consumer utility subject to a budget constraint
  - Optimal incentive contracts
  - Portfolio optimization
  - Life-cycle consumption

- Assumptions

  - Always assume f, g, and h are continuous
  - Usually assume f, g, and h are C^1
  - Often assume f, g, and h are C^3

.

- Topics
  - Unconstrained optimization
    * Unconstrained optimization problems occur naturally – maximum likelihood, minimize moment criteria
    * They are also the foundation of constrained optimization methods
  - Nonlinear equations
    * Similar to unconstrained optimization
    * Not as easy as unconstrained optimization
  - Constrained optimization
    * Optimal life-cycle problems with budget constraint
    * Maximize profit given production constraints
    * Optimal taxation given incentive compatibility constraints
    * Econometric estimation of structural models

One-D Unconstrained Minimization: Newton's Method

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

- Assume $f(x)$ is C^2 functions $f(x)$
  - At a point a, the quadratic polynomial, $p(x)$

    $$p(x) \equiv f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2.$$

    is the second-order approximation of $f(x)$ at a
  - Approximately minimize f by minimizing $p(x)$
  - If $f''(a) > 0$, then p is convex, and $x_m = a - f'(a)/f''(a)$.
  - Hope: x_m is closer than a to the minimum.
- Newton's method:

Algorithm 4.2 Newton's Method in $\mathbb{R}^1$

Initialize. Choose initial guess x_0 and stopping parameters $\delta, \epsilon > 0$.

Step 1. $x_{k+1} = x_k - f'(x_k)/f''(x_k)$.

Step 2. If $|x_k - x_{k+1}| < \epsilon(1 + |x_k|)$ and $|f'(x_k)| < \delta$, STOP and report success; else go to step 1.

- Properties:
  - Newton's method finds critical points, that is, solutions to $f'(x) = 0$, not min or max.
  - If x_n converges to x^*, must check $f''(x^*)$ to check if min or max
  - Only find local extrema.
- Good news: convergence is locally quadratic.

Theorem 1 *Suppose that $f(x)$ is minimized at x^*, C^3 in a neighborhood of x^*, and that $f''(x^*) \neq 0$. Then there is some $\epsilon > 0$ such that if $|x_0 - x^*| < \epsilon$, then the x_n sequence defined in (4.1.2) converges quadratically to x^*; in particular,*

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^2} = \frac{1}{2}\left|\frac{f'''(x^*)}{f''(x^*)}\right| \tag{4.1.3}$$

is the quadratic rate of convergence.

- Consumer problem example:
  - Consumer has \$1; price of x is \$2, price of y is \$3, utility function is $x^{1/2} + 2y^{1/2}$.
  - If θ is amount spent on x then we have

$$\max_{\theta} \quad \left(\frac{\theta}{2}\right)^{1/2} + 2\left(\frac{1-\theta}{3}\right)^{1/2} \tag{4.1.6}$$

  - Solution $\theta^* = 3/11 = .272727$
  - If $\theta_0 = 1/2$, Newton iteration is

$$0.5, 0.2595917942,\ 0.2724249335,\ 0.2727271048,\ 0.2727272727$$

    and magnitude of the errors are

$$2.3\,(-1)\,,\ 1.3\,(-2)\,,\ 3.1\,(-4)\,,\ 1.7\,(-7)\,,\ 4.8\,(-14)$$

- Problems with Newton's method
  - May not converge if initial guess is too far away from solution.
  - $f''(x)$ may be difficult to calculate.

Multidimensional Unconstrained Optimization: Comparison Methods

- Grid Search
  - Pick a finite set of points, X; for example, a Cartesian grid:

$$V = \{v_i | i = 1, ..., n\}$$
$$X = \{x \in \mathbb{R}^n | \forall i, x_i \in V\}$$

  - Compute $f(x)$, $x \in X$, and locate max
  - Grid search is often the first method to use.
    * Only involves function evaluations
    * It is embarassingly parallelizable
    * It should get you a good initial guess
  - A good initial guess is not critical for grid search, but is for all good algorithms
  - Grid search is sloooooooow, so you should always switch to something better
- General lesson: start with a reliable but slow method to find good initial guess for a faster method

- Polytope Methods (a.k.a. Nelder-Mead, simplex, "amoeba")

Algorithm 4.3 Polytope Algorithm

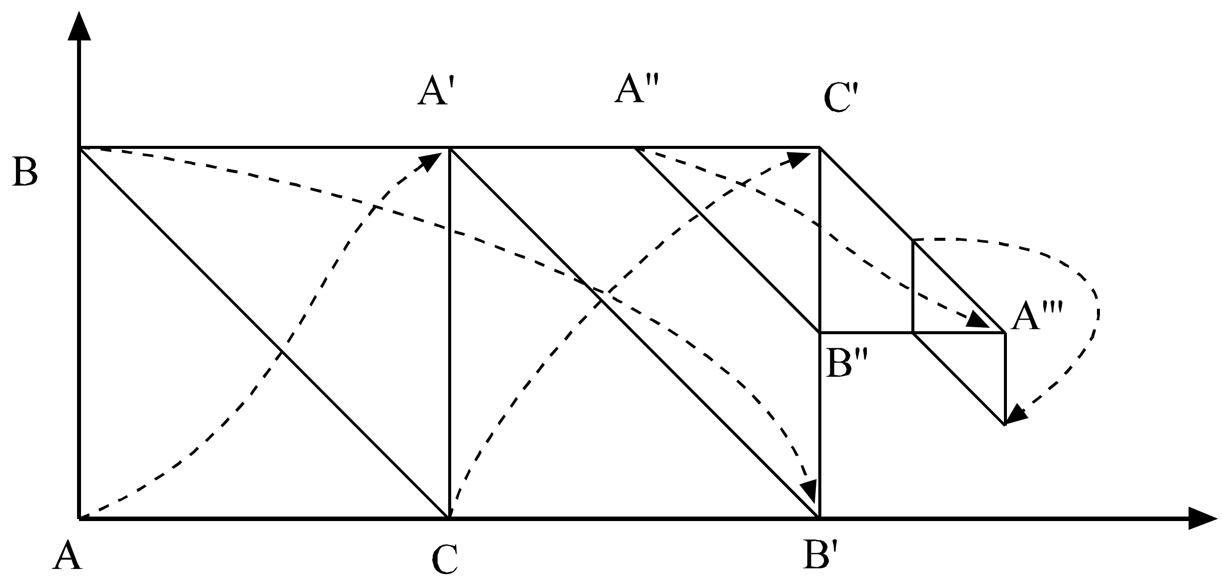
Initialize. Choose the stopping rule parameter ϵ. Choose an initial simplex $\{x^1, x^2, \cdots, x^{n+1}\}$.

Step 1. Reorder vertices so $f(x^i) \geq f(x^{i+1}),\ i = 1, \cdots, n$.

Step 2. Look for least i s.t. $f(x^i) > f(y^i)$ where y^i is reflection of x^i. If such an i exists, set $x^i = y^i$, and go to step 1. Otherwise, go to step 3.

Step 3. Stopping rule: If the width of the current simplex is less than ϵ, STOP. Otherwise, go to step 4.

Step 4. Shrink simplex: For $i = 1, 2, \cdots, n$ set $x^i = \frac{1}{2}(x^i + x^{n+1})$, and go to step 1.



Multidimensional Optimization: Newton's Method

- Idea: Given x^k, compute local quadratic approximation, $p(x)$, of $f(x)$ around x^k, and let x^{k+1} be max of $p(x)$

Algorithm 4.4 Newton's Method in $\mathbb{R}^n$

Initialize.	Choose x^0 and stopping parameters δ and $\epsilon > 0$.
Step 1.	Compute Hessian, $H(x^k)$, and gradient, $\bigtriangledown f(x^k)$, and solve $H(x^k)s^k = -(\bigtriangledown f(x^k))^\top$ for the step s^k.
Step 2.	$x^{k+1} = x^k + s^k$.
Step 3.	If $\parallel x^k - x^{k+1} \parallel < \epsilon(1+ \parallel x^k \parallel)$, go to step 4; else go to step 1.
Step 4.	If $\parallel \bigtriangledown f(x^{k+1}) \parallel < \delta(1 + \lvert f(x^{k+1})\rvert)$, STOP and report success; else STOP and report convergence to nonoptimal point.

- Stopping rule: Don't be too fussy!
  - Good values for ε and δ are close to the square root of machine epsilon.
  - First use sloppy ε and δ, such as 10^(-3).
  - Then reduce ε and δ until failure.
  - You can try to push them below square root of machine epsilon but you will probably not get too far.

Theorem 2 *Suppose that $f(x)$ is C^3, minimized at x^*, and that $H(x^*)$ is nonsingular. Then there is some $\epsilon > 0$ such that if $\| x^0 - x^* \| < \epsilon$, then the sequence defined in (4.3.1) converges quadratically to x^*.*

- Problems with Newton's method:
  - May not converge
  - Computational demands may be excessive
    * need at least $\mathcal{O}(n^2)$ time to compute $H(x^k)$, perhaps more if one does not have efficient code for $H(x)$
    * need $\mathcal{O}(n^2)$ space for $H(x^k)$
    * need $\mathcal{O}(n^3)$ time to solve $H(x^k)s^k = -(\bigtriangledown f(x^k))^\top$ for s^k
  - May converge to local solution, not global solution
  - We now consider methods which address these problems.

Direction Set Methods

- Problem: may not converge, or go to wrong kind of extremum
- Solution: if we always move uphill, we will eventually get to a local maximum

Algorithm 4.5 Generic Direction Method

Initialize.	Choose initial x^0 and stopping parameters δ and $\epsilon > 0$.
Step 1.	Compute a search direction s^k.
Step 2.	Solve $\lambda_k = \arg\min_\lambda f(x^k + \lambda s^k)$.
Step 3.	$x^{k+1} = x^k + \lambda_k s^k$.
Step 4.	If $\| x^k - x^{k+1} \| < \epsilon(1+ \| x^k \|)$, go to step 5; else go to step 1.
Step 5.	If $\| \bigtriangledown f(x^{k+1}) \| < \delta(1 + f(x^{k+1}))$, STOP and report success; else STOP and report convergence to nonoptimal point.

- Possible direction set methods
  - Coordinate Directions
    - Let search directions be coordinate, x_1, x_2, etc.
    - Search direction $s_{2n+k} = x_k$
  - Steepest Descent: $s_k = \bigtriangledown f(x^k)$
  - Newton's Method with Line Search: $H_k s^k = -(\bigtriangledown f(x^k))^\top$
- Will converge to a local optimum IF we apply something like the Armijo rule (see website).

Quasi-Newton Methods

- Problem: Hessians are expensive to compute
- Solution: Don't need true Hessians (see Carter, 1993), so approximate them

Generic Quasi-Newton Method

Initialize.	Choose initial x^0, Hessian H^0 $(I)$and stopping parameters δ and $\epsilon > 0$.
Step 1.	Solve $H_k s^k = -(\bigtriangledown f(x^k))^\top$ for the search direction s^k.
Step 2.	Solve $\lambda_k = \arg\min_\lambda f(x^k + \lambda s^k)$
Step 3.	$x^{k+1} = x^k + \lambda_k s^k$.
Step 4.	Compute H_{k+1} using H_k, $\bigtriangledown f(x^{k+1})$, x^{k+1}, $\bigtriangledown f(x^k)$, etc.
Step 5.	If $\parallel x^k - x^{k+1} \parallel < \epsilon(1+ \parallel x^k \parallel)$, go to step 6;. else go to step 1
Step 6.	If $\parallel \bigtriangledown f(x^{k+1}) \parallel < \delta\|1 + f(x^{k+1})\|$, STOP and report success; else STOP and report convergence to nonoptimal point.

- Example: BFGS:

$$
\begin{aligned}
z_k &= x^{k+1} - x^k \\
y_k &= (\bigtriangledown f(x^{k+1}))^\top - (\bigtriangledown f(x^k))^\top \\
H_{k+1} &= H_k - \frac{H_k z_k z_k^\top H_k}{z_k^\top H_k z_k} + \frac{y_k y_k^\top}{y_k^\top z_k}
\end{aligned}
$$

  - Preserves positive definiteness
  - Uses only gradients that are already needed
  - Warning: denominators may get too small; should keep them away from zero since small z_k does not necessarily stop iteration.

- Note: The Hessian iterates H_k may not converge to true Hessian at solution, even if x_k converges to solution. NEVER USE APPROXIMATE HESSIANS TO COMPUTE STANDARD ERRORS!!!!

Monopoly Example

- We look at a simple monopoly pricing example:
  - Utility function: if M is spending on other goods,

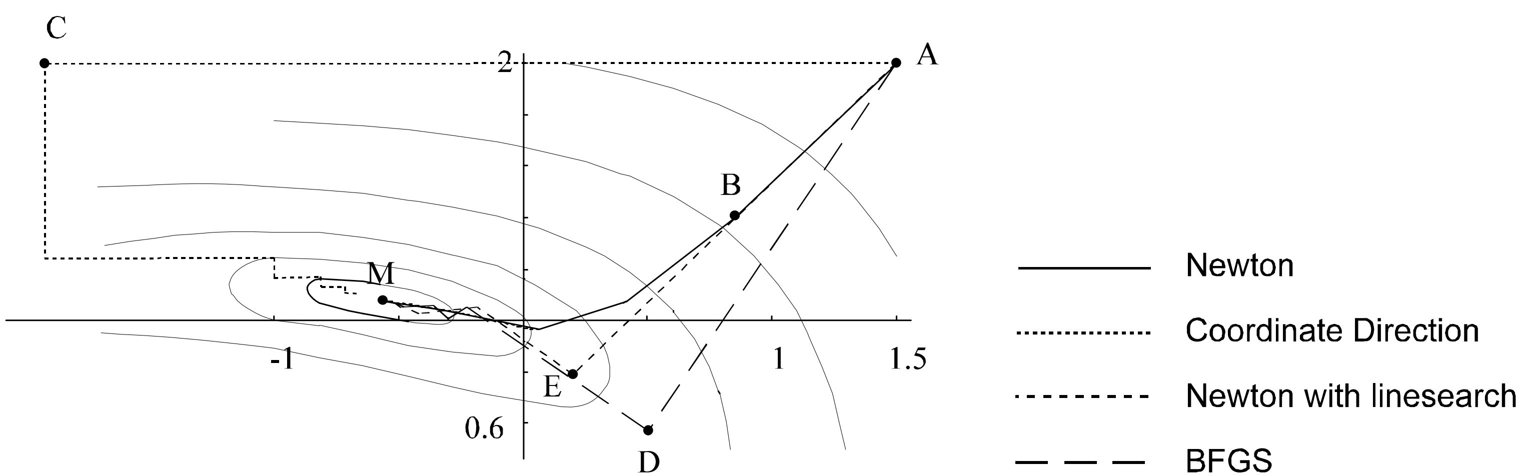
$$U(Y,Z) = (Y^\alpha + Z^\alpha)^{\eta/\alpha} + M = u(Y,Z) + M,$$

  - Output Y and Z implies prices of u_Y and u_Z.
  - Monopoly problem is

$$\max_{Y,Z} \Pi(Y,Z) \equiv Y u_Y(Y,Z) + Z u_Z(Y,Z) - C_Y(Y) - C_Z(Z), \tag{1}$$

  - Restate in terms of $y \equiv \ln Y$ and $z \equiv \ln Z$, $\pi(y,z) \equiv \Pi\left(e^y, e^z\right)$

$$\max_{y,z} \pi(y,z), \tag{2}$$



Example: A Dynamic Optimization Problem

- Life-cycle savings problem.
  - an individual lives for T periods
  - earns wages w_t in period $t, t = 1, \cdots, T$
  - consumes c_t in period t
  - earns interest on savings per period at rate r
  - define S_t to be end-of-period savings:

$$S_{t+1} = (1 + r)S_t + w_{t+1} - c_{t+1}.$$

  - Set initial wealth: $S_0 = 0$
  - utility function $\sum_{t=1}^{T} \beta^t u(c_t) + W(S_T)$
  - Substitute $c_t = S_{t-1}(1 + r) + w_t - S_t$
- Problem now has T choices:

$$\max_{S_t} \sum_{t=1}^{T} \beta^t u(S_{t-1}(1 + r) + w_t - S_t) + W(S_T)$$

- Newton's method looks impractical if T large. BUT
  - Hessian is tridiagonal (a sparse matrix)
    * The choice of S_t interacts only with the choices for S_{t-1} and S_{t+1}
    * Newton step is easy to compute.
    * The normal Hessian has size T^2
    * The tridiagonal matrix has size $3T$
  - Sparse Hessians are common in dynamic problems because time t variables interact only with time $t-1$ and time $t+1$ variables.
  - *You* must recognize this and implement Newton or quasi-Newton method with sparse Hessians, *Or* use software that automatically recognizes this structure – AMPL, GAMS, AIMMS, CASADI (future lecture), and others.

Domain Problems

- Suppose $S_0 = 0$ and you want to solve

$$\max_{S_t} \sum_{t=1}^{T} \beta^t \log\left(S_{t-1}(1+r) + w_t - S_t\right) + W(S_T)$$

- Newton's method will take the guess S^k and compute a new guess S^{k+1}.
- Problem: S^{k+1} could imply consumption, $c_t = S_{t-1}(1+r) + w_t - S_t$, will be negative at some t, causing computer to crash.
- A possible solution: Alter objective function
  - E.G.; replace $u(c) = \log c$ with, for some small $\varepsilon > 0$

$$\widetilde{u}(c) = \begin{cases} u(c), & c > \varepsilon \\ u(\varepsilon) + u'(\varepsilon)(c-\varepsilon) + u''(\varepsilon)(c-\varepsilon)^2/2, & c \leq \varepsilon \end{cases}$$

  - Maintains curvature
  - Equals real $u(c)$ on most of domain, which hopefully includes solution
  - Not as easy to apply to multivariate functions
- General solution: add constraints (next week's topic) to keep this from happening.

Nonlinear Least Squares

- Objective function has form, $f^i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, ..., m.$:

$$\min_x \frac{1}{2} \sum_{i=1}^{m} f^i(x)^2 \equiv S(x),$$

- Idea: use simple approximation of Hessian
- In econometric applications
  - $f^i(x)$ are $g(\beta, y^i)$,
    - $x = \beta$ is parameter vector
    - y^i are the data.
    - $g(\beta, y^i)$ is residual for observation i
  - $S(\beta)$ is the sum of squared residuals at β.
- Let $f(x)$ denote the column vector $(f^i(x))_{i=1}^m$.
  - Let $J(x)$ be the Jacobian of $f(x) \equiv (f^1(x), \ldots, f^m(x))^\top$.
  - Let $f^i_\ell \equiv \frac{\partial f^i}{\partial x_\ell}$ and $f^i_{j\ell} \equiv \frac{\partial^2 f^i}{\partial x_j \partial x_\ell}$.
  - The gradient of $S(x)$ is $J(x)^\top f$: $S_\ell(x) = \sum_{i=1}^m f^i_\ell(x) f^i(x)$.
  - The Hessian of $S(x)$ is $J(x)^\top J(x) + G(x)$, where

$$G_{j\ell}(x) = \sum_{i=1}^{m} f^i_{j\ell}(x) f^i(x).$$

- Special structure of the gradient and Hessian.
  - $f_j^i(x)$ terms are needed to compute gradient of $S(x)$.
  - If $f(x) = 0$, then Hessian is just $J(x)^\top J(x)$: easy to compute.
  - A problem where $f(x)$ is small at the solution is called a *small residual problem*; otherwise, it is a *large residual problem*.
- Gauss-Newton algorithm
  - Do Newton except use $J(x)^\top J(x)$ for Hessian approx.

$$s^k = -(J(x^k)^\top J(x^k))^{-1}(\nabla f(x^k))^\top \tag{4.5.1}$$

    and avoid computing second derivatives of f.
  - Natural to use for small residual problems.
  - Works very well when it works.

- Problems.
  - $J(x)^\top J(x)$ is likely to be poorly conditioned, since it is the "square" of a matrix.
  - $J(x)$ may be poorly conditioned itself, particularly in statistical contexts.
  - Gauss-Newton step may not be a descent direction.
- Solution: Levenberg-Marquardt algorithm.
  - Use $J(x)^\top J(x) + \lambda I$ for some scalar λ (I is identity matrix):

$$s^k = -(J(x^k)^\top J(x^k) + \lambda I)^{-1} (\nabla f(x^k))^\top$$

  - The λI term reduces conditioning problems by "adding a little piece of the identity matrix"
  - s^k will be descent direction for large λ since s^k gets closer to steepest descent direction λ.