TA Session 5: Numerical Optimization

Shuowen Chen[1]

EC708: PhD Econometrics I (Spring 2020)

[1]Parts of the materials are borrowed from teaching slides by Pierre Perron, Gianluca Violante, Jean–Jacques Forneron and textbook by Kenneth Train (2009). I thank Iván Fernández-Val, Hiroaki Kaido and Jean–Jacques Forneron for discussions of material coverage. I'm grateful to Yinuo Zhang for kindly sharing Violante's slides.

Outline

- Optimization problem
- Full–Newton Method:
    - Newton–Ralphson
    - Gauss–Newton
- Quasi–Newton Method
    - Berndt–Hall–Hall–Hausman (BHHH) algorithm
    - Steepest Ascent
    - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm
    - Davidon–Fletcher–Powell (DFP) algorithm
- Bonus Algorithms:
    - (Stochastic) Gradient Descent
    - Some comparison methods
        - Simplex method (Nelder–Meade/Polytope)
        - Simulated Annealing

Optimization Problem: General Setup

We first consider the problem of minimizing a criterion function $f(\beta)$ with β being an $K \times 1$ vector of parameter.

Remark: for maximization, simply work with $-f(\beta)$.

Notations:

- Vector of first derivatives

$$g(\beta) = \frac{\partial f(\beta)}{\partial \beta}$$

- Matrix of second derivatives (Hessian)

$$G(\beta) = \frac{\partial f^2(\beta)}{\partial \beta \partial \beta'}$$

Global or Local Optimum?

Remark: A sufficient condition for a local minimum at $\widehat{\beta}$ is

$$g(\widehat{\beta}) = 0 \text{ and } G(\widehat{\beta}) \text{ is positive definite}$$

Why not necessary? A counterexample: $y = \beta^4$ and $\widehat{\beta} = 0$.

In practice no algorithm guarantees finding the global minimum.

One remedy: repeat the optimization with different starting values.

Numerical Evaluation of Derivatives

Recall the definition of first-order partial derivative of function f at β:

$$\frac{\partial f(\beta)}{\partial \beta_j} = \lim_{h \to 0} \frac{f(\beta_1, \ldots \beta_j + h, \ldots, \beta_K) - f(\beta_1, \ldots \beta_j, \ldots, \beta_K)}{h}$$

Numerically, we can approximate it by calculating

$$g_j(\widehat{\beta}) = \frac{f(\widehat{\beta} + h_j d_j) - f(\widehat{\beta})}{h_j}, \quad j = 1, \ldots, K$$

where

- $d_j = (0, , \ldots 0, 1, 0, \ldots, 0)$: unit vector with 1 in position j
- h_j: step length. Small for estimate to be close, larger enough than rounding error

Approximate second-order derivatives using similar notations:

$$G_{ji}(\widehat{\beta}) = \frac{g_j(\widehat{\beta} + h_i d_i) - g_j(\widehat{\beta})}{h_i}, \quad i, j = 1, \ldots, K$$

▸ A more accurate evaluation

Optimization Problem: MLE

Consider maximizing the log likelihood function[2]

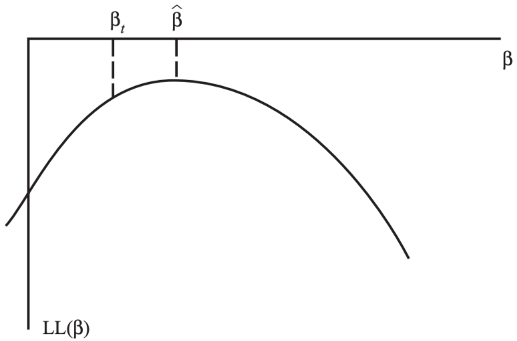
$$L(\beta) = \sum_{n=1}^{N} \log(P_n(\beta))/N,$$

- $P_n(\beta)$: prob of the observed outcome for decision maker n
- N: sample size
- β: $K \times 1$ vector of parameters

Goal: find $\widehat{\beta} = \arg\max_{\beta} L(\beta)$

[2] To utilize the minimization packages in practice, usually work with $-L(\beta)$.

Graphical Illustration of MLE



Finding $\widehat{\beta}$ is a hill–climbing process:

1. Given a starting point, decide what direction and how far to climb
2. Update point of starting and keep climbing
3. Stop when some criteria are met

We formalize this process via math, starting from full–Newton method

Newton–Raphson Method

Take a second–order Taylor expansion of $f(\beta)$ around the optimum $\widehat{\beta}$:

$$f(\beta) \approx f(\widehat{\beta}) + (\beta - \widehat{\beta})' g(\widehat{\beta}) + \frac{1}{2}(\beta - \widehat{\beta})' G(\widehat{\beta})(\beta - \widehat{\beta})$$

Differentiate w.r.t β yields

$$g(\beta) \approx g(\widehat{\beta}) + G(\widehat{\beta})(\beta - \widehat{\beta})$$

Since $g(\widehat{\beta}) = 0$ by definition, we have

$$\widehat{\beta} \approx \beta - G^{-1}(\widehat{\beta}) g(\beta)$$

The derivation suggests the following iteration procedure:

1. Pick an initial guess β_0 and compute $\beta_1 = \beta_0 - G^{-1}(\beta_0) g(\beta_0)$
2. Compute $\beta_2 = \beta_1 - G^{-1}(\beta_1) g(\beta_1)$
3. Continue step 2 until convergence

Remark: $G^{-1}(\widehat{\beta})$ is infeasible, so use iterated estimate instead

Newton–Raphson to Log Likelihood Function

Define the gradient in iteration k, β_k:

$$g_k = \frac{\partial L(\beta)}{\partial \beta}\Big|_{\beta=\beta_k}$$

and Hessian:

$$H_k = \Big(\frac{\partial^2 L(\beta)}{\partial \beta \partial \beta'}\Big)\Big|_{\beta=\beta_k}$$

The Newton–Raphson update in iteration $k+1$ is:

$$\beta_{k+1} = \beta_k - H_k^{-1} g_k$$

Convergence Criteria

If f is exactly quadratic, then Newton–Raphson finds the optimum in one–step. Consider $f(\beta) = a + b\beta + c\beta^2$, whose optimum is $\widehat{\beta} = -\frac{b}{2c}$.

- Its first and second order derivatives, evaluated at iteration k, are $b + 2c\beta_t$ and $2c$ respectively
- Hence $\beta_{k+1} = \beta_k - \frac{1}{2c}(b + 2c\beta_k) = -\frac{b}{2c} = \widehat{\beta}$

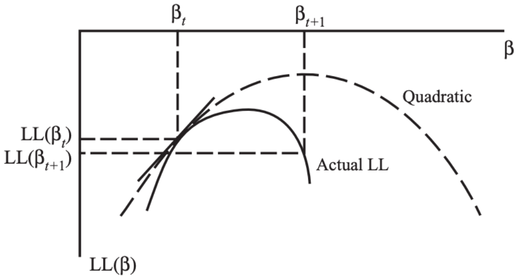
In practice, convergece can be defined in many ways:

- $f(\beta_{k+1})$ close to $f(\beta_k)$
- β_{k+1} close to β_k
- $g(\beta_{k+1})$ close to $g(\beta_k)$

Convergence can be hard because

- Local versus global optimums
- Objective functions being flat (weakly identified IV and GMM)

Overshooting and Stepsize



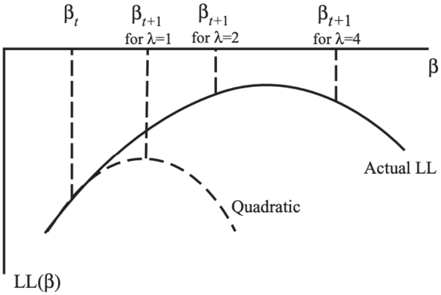
- The Newton–Raphson method updates from β_k to β_{k+1}, but $L(\beta_{k+1}) < L(\beta_k)$. The difference between β_{k+1} and β_k too large
- To ensure each iteration provides an increase in $L(\beta)$, we introduce stepsize λ_k :

$$\beta_{k+1} = \beta_k - \lambda_k G^{-1}(\beta_k) g(\beta_k)$$

Determining the Stepsize: Backtracking Line Search

For each iteration, start with a large λ and gradually shrink it to guarantee an increase in $L(\beta)$. One such procedure for iteration k:

1. Start with $\lambda_k = 1$, if $L(\beta_{k+1}) > L(\beta_k)$, move to β_{k+1} and start iteration $k + 1$
2. If $L(\beta_{k+1}) < L(\beta_k)$, set $\lambda_k = \frac{1}{2}$ and try again
3. Continue step 2 until we find the first λ_k that yields $L(\beta_{k+1}) > L(\beta_k)$. Now move on to iteration $k + 1$.


β_t
β_{t+1} for λ=1
β_{t+1} for λ=2
β_{t+1} for λ=4
β
Actual LL
Quadratic
LL(β)

Futher Issues with Newton–Raphson Method

Objective function not necessarily globally convex[3]

- Hessian may not be positive definite[4]
- Newton–Raphson update actually moves to opposite direction
- One Remedy: regularization, instead of using $G(\beta_k)^{-1}$ directly, use
$$\left[G(\beta_k) + \mu_k I_K\right]^{-1},$$
where $\mu_k > 0$: guarantees positive definiteness

Computing Hessian is costly

- Too many function evaluations are required
- Numerically calculated Hessian might be ill–behaved (singular)
- Consider algorithms that approximate the Hessian.

Let's start with Gauss–Newton method, which is designed for calculation of nonlinear least–squared estimates

[3] For maximization problem objective function is not globally concave.

[4] For maximization problem replace with negative defnite

Gauss–Newton Method

Consider the following general nonlinear model:

$$y_t = f(x_t; \beta) + u_t, \quad t = 1, \ldots, T$$

- $u_t \sim i.i.d.\ \mathcal{N}(0, \sigma^2)$. Assume σ^2 is known here.
- x_t: $M \times 1$ exogenous regressors
- β: $K \times 1$ vector of parameters
- $f(\cdot)$: some function satisfying some regularity conditions

Gauss–Newton Method Cont.d

Due to normality assumption, we have the log likelihood function

$$L(\beta) = -\frac{T}{2}\log 2\pi - \frac{T}{2}\log\sigma^2 - \frac{1}{2\sigma^2}S(\beta),$$

where $S(\beta) = \sum_{t=1}^{T}[y_t - f(x_t;\beta)]^2 = \sum_{t=1}^{T} u_t^2$. It suffices to work with $S(\beta)$. Its first and second order derivatives w.r.t β are

$$\mathrm{g}(\beta) = 2\sum_{t=1}^{T}\frac{\partial u_t}{\partial\beta}u_t, \quad G(\beta) = 2\sum_{t=1}^{T}\Big[\frac{\partial u_t}{\partial\beta}\frac{\partial u_t}{\partial\beta'} + \frac{\partial^2 u_t}{\partial\beta\partial\beta'}u_t\Big]$$

In $G(\beta)$, the blue term is usually small relative to the red term, so we **neglect** it and use the following:

$$\beta_{k+1} = \beta_k - \Big[\sum_{t=1}^{T}\frac{\partial u_t}{\partial\beta}\frac{\partial u_t}{\partial\beta'}\Big]^{-1}\Big|_{\beta=\beta_k}\sum_{t=1}^{T}\frac{\partial u_t}{\partial\beta}u_t\Big|_{\beta=\beta_k}$$

Remarks on the Gauss–Newton Method

- The method doesn't compute Hessian
- Has an OLS interpretaion. Let $z_t = -\partial u_t / \partial \beta$, then

$$\beta_{k+1} = \beta_k + \Big(\sum_{t=1}^{T} z_t z_t' \Big)^{-1} \Big|_{\beta=\beta_k} \sum_{t=1}^{T} z_t u_t \Big|_{\beta=\beta_k}$$

- Similar modification can be incorporated as in the Newton–Raphson: Marquart quadratic hill climbing

$$\beta_{k+1} = \beta_k + \Big(\sum_{t=1}^{T} z_t z_t' + \mu I \Big)^{-1} \Big|_{\beta=\beta_k} \sum_{t=1}^{T} z_t u_t \Big|_{\beta=\beta_k}$$

Quasi–Newton Method

Full–Newton method numerically evaluates the Hessian.
Quasi–Newton methods avoid it by approximating the Hessian, with differences in terms of how to approximate.

General procedures:

1. Specify initial β_0 and G_0. In each iteration k
2. Compute Quasi–Newton direction $\Delta\beta_k = -G(\beta_k)^{-1}g(\beta_k)$
3. Determine the stepsize λ_k (by backtracking line search)
4. Compute $\beta_{k+1} = \beta_k + \lambda_k\Delta\beta_k$
5. Compute $G(\beta_{k+1})$ from $G(\beta_k)$

I will use G_k and g_k to denote $G(\beta_k)$ and $g(\beta_k)$ now.

Quasi–Newton Algorithm: BHHH

Very suitable for log likelihood function maximization as it uses score to approximate Hessian:

$$s_n(\beta_k) = \left.\frac{\partial \log P_n(\beta)}{\partial \beta}\right|_{\beta=\beta_k},$$

For a log likelihood function, **gradient is average scores**:

$$g_k = \sum_{n=1}^{N} \frac{s_n(\beta_k)}{N}$$

Outer product of observation n's score is the $K \times K$ matrix:

$$s_n(\beta_k) s_n(\beta_k)' = \begin{pmatrix} s_n^1 s_n^1 & s_n^1 s_n^2 & \cdots & s_n^1 s_n^K \\ s_n^2 s_n^1 & s_n^2 s_n^2 & \cdots & s_n^2 s_n^K \\ \vdots & \vdots & & \vdots \\ s_n^K s_n^1 & s_n^K s_n^2 & \cdots & s_n^K s_n^K \end{pmatrix}$$

where s_n^j is the j–th element of $s_n(\beta_k)$

BHHH Update

Average outer product:

$$G_k = \frac{1}{N}\sum_{n=1}^{N} s_n(\beta_k)s_n(\beta_k)'$$

BHHH update:

$$\beta_{k+1} = \beta_k + \lambda_k G_k^{-1} g_k$$

Why does this work?

- At maximum, the average score is zero and thus average outer product becomes the variance of scores in the sample
- This variance, like Hessian, provides a measure of the log–likelihood function's curvature, A higher variance implies a greater curvature
- Formally speaking, this is the **information matrix eqality**:

$$\mathbb{E}\Big[\frac{\partial L(\beta)}{\partial \beta}\frac{\partial L(\beta)}{\partial \beta'}\Big] = -\mathbb{E}\Big[\frac{\partial^2 L(\beta)}{\partial \beta \beta'}\Big]$$

Quasi–Newton Algorithm: Steepest Ascent

BHHH can suffer from singularity or bad scaling in practice. One alterantive formula is the following:

$$\beta_{k+1} = \beta_k + \lambda_k g_k$$

This is using the identity matrix to approximate the Hessian[5].

- Identity matrix is pd, guarantees an increase in each iteration
- Steepest as it provides the greatest possible increase in $L(\beta)$ for the distance between β_k and β_{k+1}, but choice of λ_k is critical. Need to be small, which slows convergence.
- For minimization problem, this is called the gradient descent. We will briefly talk about stochastic gradient descent, which is popular in deep learning.

[5] Again we are considering maximization here, for minimization we consider descent and negative identity matrix

BFGS and DFP

BFGS is the algorithm behind matlab's fminunc. The update of G_{k+1} from G_k is the following:

$$G_{k+1} = G_k + \frac{y_k y_k'}{y_k' \gamma_k} - \frac{G_k \gamma_k \gamma_k' G_k}{\gamma_k' G_k \gamma_k},$$

where $\gamma_k = \beta_{k+1} - \beta_k$ and $y_k = g_{k+1} - g_k$. Using Sherman–Morrison formula, we have

$$G_{k+1}^{-1} = \left(I - \frac{\gamma_k y_k'}{y_k' \gamma_k}\right) G_k^{-1} \left(I - \frac{y_k \gamma_k'}{y_k' \gamma_k}\right) + \frac{\gamma_k \gamma_k'}{y_k' \gamma_k}$$

DFP precedes BFGS, but gets superseded by the latter:

$$G_{k+1}^{DFP} = G_k^{DFP} + \frac{\gamma_k \gamma_k'}{\gamma_k' y_k} - \frac{G_k^{DFP} y_k y_k' G_k^{DFP}}{y_k' G_k^{DFP} y_k}$$

The two update swap the role of γ_k and y_k.

Gradient Descent

We've seen steepest ascent in maximization problem. Now minimize an unconstrainted, smooth convex and twice differentiable function

$$\min_x f(x)$$

The Newton–Raphson method updates x as follows:

$$x_{k+1} = x_k - G_k^{-1} g_k$$

while gradient descent updates as follows:

$$x_{k+1} = x_k - \lambda_k g_k$$

Reminder: G_k and g_k respectively denote Hessian and gradient evaluated at x_k. λ_k denotes the step size.

Stochastic Gradient Descent

- Historically gradient descent not very popular in nonlinear/nonconvex optimization problem because it gets stuck at local minima
- For deep network, local minimum are close to global minimum in terms of prediction, so not problematic
- Computing gradient can be expensive in deep network. One way to be more efficient is the stochastic gradient descent
- Instead of evaluating the gradient at each observation (with sample size N), in each iteration we consider a subsample $(x_1^\star, \ldots, x_m^\star)$, $m < N$ and compute[6]

  $$\beta_{k+1} = \beta_k - \lambda_k g_k^\star,$$

  where $g_k^\star$ denotes gradient of subsample evaluated at β_k
- In practice, prefer small m (like 1): cheaper to compute and avoids overfitting (Goodfellow et al, 2016). Good finite and large sample behavior (Toulis and Airoldi, 2017).

[6] On how to tune λ_k, refer to this.

Comparison Methods

- Most of the methods we discussed are gradient–based: use info on the slope and possibly on curvature.
- Gradient–based methods get stuck if objective function is non–smooth; started at different initial points, more likely to lead to a different local optimum
- An alternative type of method is comparison–based: compute objective function at several points and pick the one yielding the optimum value
- Comparison method better behaved with non–smooth objective functions; stochastic comparison method more likely to find global optimum (in theory)

Nonliner Simplex Method: Nelder–Meade/Polytope

This is fminsearch in Matlab

1. Choose initial simplex[7] $\{x_1, x_2, \ldots, x_{n+1}\} \in \mathbb{R}^n$
2. Sort simplex vertices in descending order

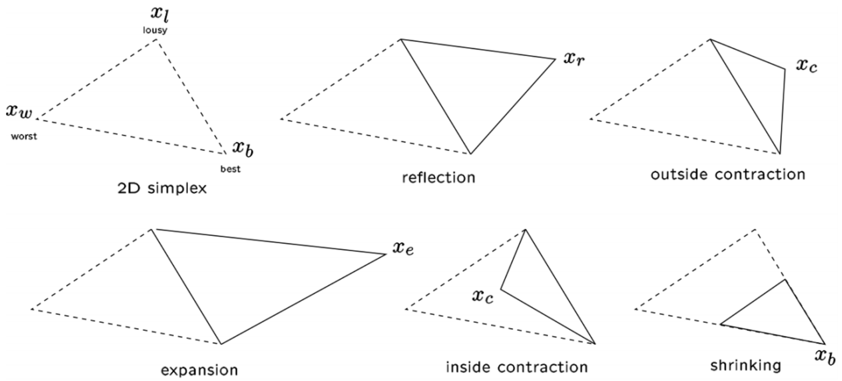
$$f(x_i) \geq f(x_{i+1}) \geq \ldots, \ \forall i$$

3. Find the smallest i such that $f(x_i^R) < f(x_i)$, where x_i^R is the reflection of x_i. If x_i^R exists, replace x_i with x_i^R and go back to step 2. Otherwise, go to step 4.
4. If width of the current simplex smaller than tolerance, stop. Otherwise go to step 5
5. For $i = 1, \ldots, n$, set $x_i^S = \frac{x_i + x_{i+1}}{2}$ to shrink the simplex. Go back to step 1.

Doesn't require information on derivatives, hence suitable for non–smooth objective functions.

[7] Think of it as an n–dimensional version of a triangle.

Some Simplex Moves



Intuition: Nelder–Mead starts with a simplex and modifies it at each iteration using one of the moves. The sequence of moves to be performed is chosen based on the relative values of the objective functions at each of the points.

Simulated Annealing

Suitable for finding a global minimum among many local ones.

1. Draw z from $\mathcal{N}(0, 1)$ and perturb initial guess x_0: $x_1 = x_0 + \lambda z$
2. If $f(x_1) < f(x_0)$, move to step 3. Otherwise accept stochastically: accept if
$$\frac{f(x_1) - f(x_0)}{|f(x_0)|} < \tau c,$$
where $c \sim U[0, 1]$ and $\tau > 0$ is temperature parameter[8]. If not, go back to step 1 and redraw.
3. Compute $|x_1 - x_0|$. Stop if less than tol, otherwise, update initial guess ($x_0 = x_1$) and back to step 1.

Fun fact: I used this once in an IO project on moment inequalities, the results suck...big time.

[8]Along the iterations decrease $\tau \to 0$ to cool down (reduce randomness).

Central Difference

For numerical differentiation, it is recommended to use

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h},$$

which is more accurate in practice.
The multivariate version should be fairly straightforward. ◂ Back