

Principles of Programming in Econometrics

Introduction, structure, and advanced programming techniques

Charles S. Bos

Vrije Universiteit Amsterdam
Tinbergen Institute

`c.s.bos@vu.nl`

August 2020 – Version Python

Separate lecture slides

Compilation: July 27, 2020

Overview

Principles of Programming in Econometrics

D0: Syntax, example 2⁸

D1: Structure, scope

D2: Numerics, packages

D3: Optimisation, speed

Day 3: Optimisation

9.30 Optimization (minimize)

- ▶ Idea behind optimization
- ▶ Gauss-Newton/Newton-Raphson
- ▶ Stream/order of function calls
- ▶ Standard deviations
- ▶ Restrictions
- ▶ Speed

13.30 Practical

- ▶ Regression: Maximize likelihood
- ▶ GARCH-M: Intro and likelihood

Optimisation

- ▶ Theory: What is (to be) done
- ▶ Inputs
- ▶ Practice/implementation
- ▶ Standard errors
- ▶ Transformations

Optimisation

Doing Econometrics \equiv estimating models, e.g.:

1. Optimise likelihood
2. Minimise sum of squared residuals
3. Minimise difference in moments
4. Solving utility problems (macro/micro)
5. Do Bayesian simulation, MCMC

Options 1-3 evolve around

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} f(y; \theta), \quad f(y; \theta) : \mathbb{R}^p \rightarrow \mathbb{R}$$

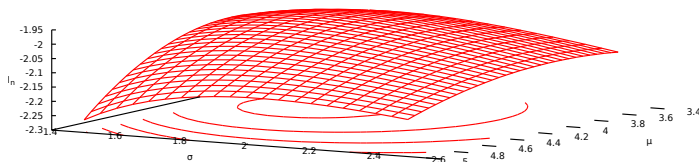
Option 4 evolves around

$$r(y; \hat{\theta}) \equiv \mathbf{0}, \quad r(y; \theta) : \mathbb{R}^p \rightarrow \mathbb{R}^p$$

Example

For simplicity: Econometrics example, ...

$$\bar{l}(y; \theta) = -\frac{1}{2n} \sum_{i=1}^n \left(\log 2\pi + \log \sigma^2 + \frac{(y_i - \mu)^2}{\sigma^2} \right)$$

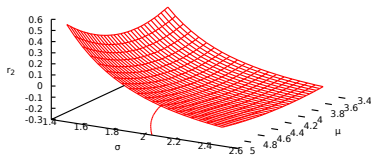
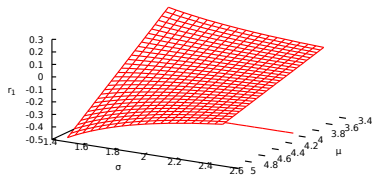


Relatively simple function to optimize, but how?

Example II

... translated to Macro/Micro solving equations

$$r(y; \theta) \equiv \frac{\partial \bar{l}(y; \theta)}{\partial \theta} = \begin{pmatrix} \frac{1}{n\sigma^2} \sum (y_i - \mu) \\ -\frac{1}{\sigma} + \frac{\sum (y_i - \mu)^2}{n\sigma^3} \end{pmatrix}$$



Score = derivative of (avg) loglikelihood $\bar{l}(y; \theta)$, $\mathbb{R}^2 \rightarrow \mathbb{R}^2$

Crawling up a hill

Step back and concentrate:

- ▶ Searching for

$$\hat{\theta} = \operatorname{argmin}_{\theta} f(y; \theta) = \operatorname{argmax}_{\theta} -f(y; \theta)$$

- ▶ How would you do that?

Crawling up a hill

Step back and concentrate:

- ▶ Searching for

$$\hat{\theta} = \operatorname{argmin}_{\theta} f(y; \theta) = \operatorname{argmax}_{\theta} -f(y; \theta)$$

- ▶ How would you do that?
- ▶ Imagine Alps:
 - a. Step outside hotel
 - b. What way goes up?
 - c. Start **Crawling up a hill**
 - d. Continue for a while
 - e. If not at top, go to b.

Use function characteristics

Translate to mathematics:

- a. Set $j = 0$, start in some point $\theta^{(j)}$
- b. Choose a direction s
- c. Move distance α in that direction, $\theta^{(j+1)} = \theta^{(j)} + \alpha s$
- d. Increase j , and if not at top continue from b

Direction s : Linked to gradient?

Minimum: Gradient 0, second derivative *positive* definite?

(Maximum: Gradient 0, second derivative *negative* definite?)

Ingredients

Inputs are

- ▶ f , use (*negative*) *average log* likelihood, or *average* sum-of-squares;
- ▶ Starting value $\theta^{(0)}$;
- ▶ Possibly $g = f'$, analytical first derivatives of f ;
- ▶ (and possibly $H = f''$, analytical second derivatives of f).

Ingredients

Inputs are

- ▶ f , use (*negative*) *average log* likelihood, or *average* sum-of-squares;
- ▶ Starting value $\theta^{(0)}$;
- ▶ Possibly $g = f'$, analytical first derivatives of f ;
- ▶ (and possibly $H = f''$, analytical second derivatives of f).

or

- ▶ r , use set of equations, if necessary *scaled*;
- ▶ Starting value $\theta^{(0)}$;
- ▶ If available $J = r'$, analytical Jacobian of r

Ingredients II (optimize)

$$f(\theta) : \mathbb{R}^p \rightarrow \mathbb{R}$$

Function, scalar

$$f'(\theta) = \left[\frac{\partial f(\theta)}{\partial \theta_1}, \dots, \frac{\partial f(\theta)}{\partial \theta_p} \right]^T \equiv g$$

Derivative, gradient, $p \times 1$

$$f''(\theta) = \left[\frac{\partial^2 f(\theta)}{\partial \theta_i \partial \theta_j} \right]_{i,j=1}^p \equiv H$$

Second derivative, Hessian, $p \times p$

If derivatives are continuous (as we assume), then

$$\frac{\partial^2 f(\theta)}{\partial \theta_i \partial \theta_j} = \frac{\partial^2 f(\theta)}{\partial \theta_j \partial \theta_i} \quad H = H^T$$

Hessian symmetric

Ingredients III (solve)

$$r(\theta) : \mathbb{R}^p \rightarrow \mathbb{R}^p$$

Function, $p \times 1$

$$r'(\theta) = \left[\frac{\partial r(\theta)}{\partial \theta_1}, \dots, \frac{\partial r(\theta)}{\partial \theta_p} \right] \equiv J$$

Derivative, Jacobian, $p \times p$

No reason for Jacobian to be symmetric

Newton-Raphson for minimisation

- ▶ Approximate $f(\theta)$ locally with quadratic function

$$f(\theta + h) \approx q(h) = f(\theta) + h^T f'(\theta) + \frac{1}{2} h^T f''(\theta) h$$

- ▶ Minimise $q(h)$ (instead of $f(\theta + h)$)

$$q'(h) = f'(\theta) + f''(\theta)h = 0 \Leftrightarrow f''(\theta)h = -f'(\theta) \text{ or } Hh = -g$$

by solving last expression, $h = -H^{-1}g$

- ▶ Set $\theta = \theta + h$, and repeat as necessary

Problems:

- ▶ Is H positive definite/invertible, at each step?
- ▶ Is step h , of length $\|h\|$, too big or small?
- ▶ Do we converge to true solution?

Newton-Raphson for solving equations

- ▶ Approximate $r(y; \theta)$ locally with linear function

$$r(\theta + h) \approx q'(h) = r(\theta) + r'(\theta)h$$

- ▶ Solve $q'(h) = \mathbf{0}$ (instead of $r(\theta + h) = \mathbf{0}$)

$$q'(h) = r(\theta) + r'(\theta)h = \mathbf{0} \Leftrightarrow r'(\theta)h = -r(\theta) \text{ or } Jh = -r$$

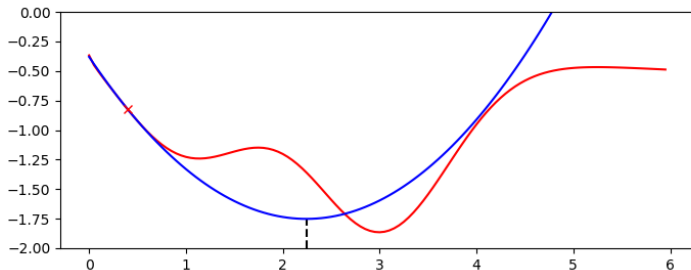
by solving last expression, $h = -J^{-1}r$

- ▶ Set $\theta = \theta + h$, and repeat as necessary

Problems:

- ▶ Is J ~~positive definite~~/invertible, at each step?
- ▶ Is step h , of length $\|h\|$, too big or small?
- ▶ Do we converge to true solution?

Newton-Raphson II



$$f(\theta) = -e^{-(\theta-1)^2} - 1.5e^{-(\theta-3)^2} - .2\sqrt{\theta}$$

- ▶ How does the algorithm converge?
- ▶ Where does it converge to?

```
ipython np_newton_show2, theta= 5.9/1/0.1/0.4
```

Problematic Hessian?

Algorithms based on NR need $H_j = f''(\theta^{(j)})$. Problematic:

- ▶ Taking derivatives is not stable (...)
- ▶ Needs many function-evaluations
- ▶ H not guaranteed to be positive definite

Problem is in step

$$s_j = -H_j^{-1}g_j \approx -M_jg_j$$

Replace H_j^{-1} by some M_j , positive definite by definition?

BFGS

Broyden, Fletcher, Goldfarb and Shanno (BFGS) thought of following trick:

1. Start with $j = 0$ and positive definite M_j , e.g. $M_0 = I$
2. Calculate $s_j = -M_j g_j$, with $g_j = f'(\theta^{(j)})$
3. Find new $\theta^{(j+1)} = \theta^{(j)} + h_j$, $h_j = \alpha s_j$
4. Calculate, with $q_j = g_j - g_{j+1}$

$$M_{j+1} = M_j + \left(1 + \frac{q_j' M_j q_j}{h_j' q_j} \right) \frac{h_j h_j'}{h_j' q_j}$$

Result:

▶ No Hessian needed

▶ Still good convergence

▶ No problems with negative definite H_j

⇒ `scipy.optimize.minimize(method="BFGS", ...)` in Python, similar routines in Ox/Matlab/Gauss/other.

$$- \frac{1}{h_j' q_j} (h_j q_j' M_j + M_j q_j h_j')$$