Numerical Methods - Root Finding & Optimization

David Wiczer

Wash U Econ 5121

January 26, 2016

Outline

Mortensen Pissarides Model

The setup

- Linear production, productivity z
- Employed workers are paid $w(z)$
- Unemployed workers on the dole get b
- Preferences are linear in time and goods
- Firms flow profits are $\pi = z - w(z)$

Search & Matching

- There's a congestion friction in the market: more workers around mean it's more difficult for each to find the jobs
- Just to keep them off the dock, firms make offers, v, each costing κ
- The mass of supplicants is u
- Total matches are $m(u, v)$
  - Offers arrive at rate $p(\theta) = \frac{m}{u}$
  - Applications arrive at rate $q(\theta) = \frac{m}{v}$
- $\theta = \frac{v}{u} \Rightarrow q(\theta)\theta = p(\theta)$
- Separations occur exogenously at rate δ

Dynamic problems

We're going to do this in discrete time, as Pissarides (AER 1985). The value of employed and unemployed workers is:

$$W(z) = w(z) + \beta E[(1-\delta)W(z') + \delta U(z')] \quad (1)$$
$$U(z) = b + \beta E[p(\theta)W(z') + (1-p(\theta))U(z')] \quad (2)$$

Firms who are matched or posting vacancy are valued:

$$J(z) = z - w(z) + \beta E[(1-\delta)J(z') + \delta V(z')] \quad (3)$$
$$V(z) = -\kappa + \beta E[q(\theta)J(z') + (1-q(\theta))V(z')] \quad (4)$$

Free entry gives us that

$$V(z) = 0 \; \forall z$$

Solving the model!

- Divide profits of the match by Gen N-B
- Define the match surplus

$$S(z) = W(z) - U(z) + J(z)$$

- Workers' bargaining power is μ and wages solve $\max(W(z) - U(z))^{\mu} J(z)^{1-\mu}$, which yields

$$w(z) = \mu z + (1 - \mu)b + \mu z \kappa \theta \tag{5}$$

Solving the model!

- Divide profits of the match by Gen N-B
- Define the match surplus

$$S(z) = W(z) - U(z) + J(z)$$

- Workers' bargaining power is μ and wages solve $\max(W(z) - U(z))^{\mu} J(z)^{1-\mu}$, which yields

$$w(z) = \mu z + (1-\mu)b + \mu z \kappa \theta \tag{5}$$

- From the zero profit and value of firm:

$$\kappa = \beta q(\theta) E[J(z')]$$

- Using the surplus equation to sub out J (and a bunch of other stuff) we get a stochastic difference equation

$$\frac{\kappa}{\beta q(\theta(z))} = E[(1-\mu)(z'-b) - \kappa\, \mu\theta(z') + \frac{(1-\delta)\kappa}{q(\theta(z'))}] \tag{6}$$

With discrete values for z we have a system of equations!

Limits of the model

- Mortensen and Nagypal (2007) document how it fares with cyclical fluctuations (poor)
- Also! It cannot account for unemployment duration

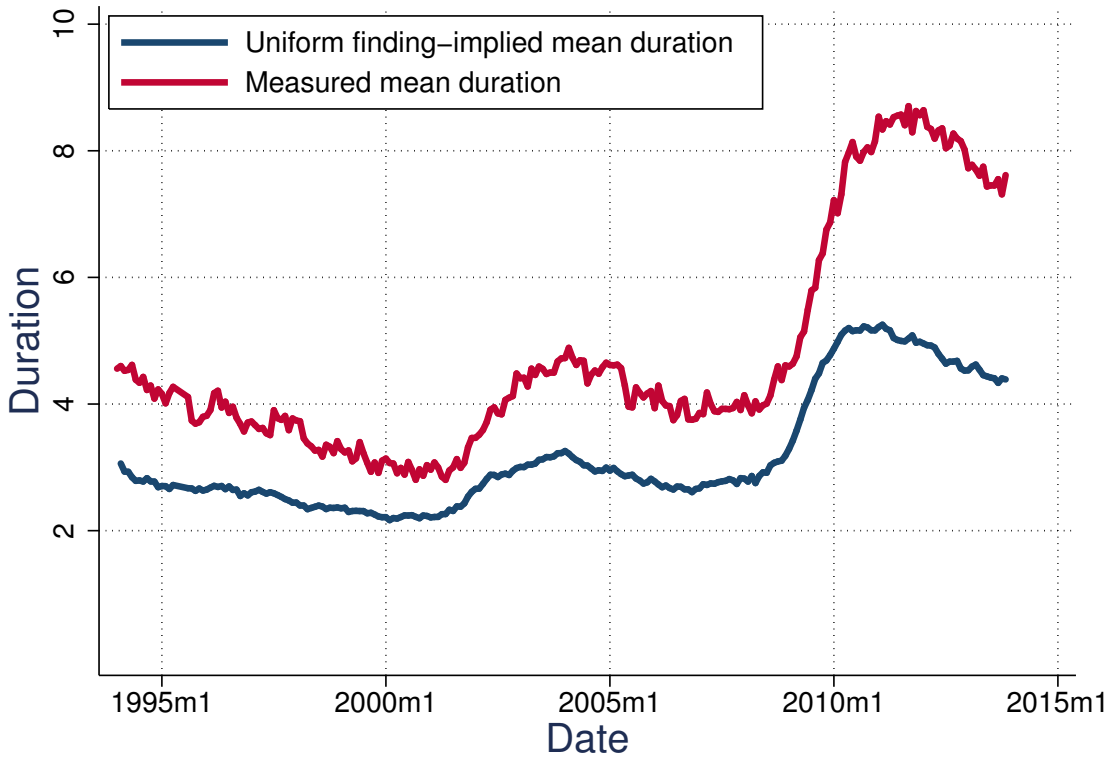

Figure: The mean duration before finding a job

Translating it for the computer

- Suppose
$$z' = (1-\rho)\bar{z} + \rho z + \sigma\epsilon'$$
- We can discretize this AR(1) as a Markov chain with $z \in \{z_0, \ldots, z_N\}$ (more on this later)
- Then expectations are just a dot product, let P be the Markov transition matrix

$$\begin{pmatrix} \frac{\kappa}{\beta q(\theta_0)} \\ \vdots \\ \frac{\kappa}{\beta q(\theta_N)} \end{pmatrix} - P \begin{pmatrix} (1-\mu)(z_0-b) - \kappa\mu\theta_0 + \frac{(1-\delta)\kappa}{q(\theta_0)} \\ \vdots \\ (1-\mu)(z_N-b) - \kappa\mu\theta_N + \frac{(1-\delta)\kappa}{q(\theta_N)} \end{pmatrix} = 0 \quad (7)$$

- This is $N+1$ equations to solve for $N+1$ values of θ.

Some discussion on parameters

- Note the crucial two values, the size of the surplus and firms' share of it determine $\frac{\partial \theta}{\partial z}$
- Shimer (2005) suggests that MP cannot match fluctuations with ordinary parameter values (Hosios condition and 40% replacement)
- Hagedorn and Manovskii (2008) counter that we can juice the replacement and the firms' share
- There is also the issue of the elasticity of the matching function

$$p(\theta) = \frac{\theta}{(1+\theta^{\phi})^{1/\phi}} \Rightarrow \frac{\partial p}{\partial \theta}\frac{\theta}{p} = (1+\theta^{\phi})^{-1}$$

Given HM parameters, this is about 0.52, but it's .38 in data.

Root Finding and Optimization

Big themes

- These will all be iterative:
  1. Create an approximation of the function (often called a “model”)
  2. Find it’s zero
  3. From that new point, re-approximate and solve
- We need to keep in mind what’s global and what’s local
- Optimization is just root-finding because we’re looking for a zero of the derivative.

Root Finding and Optimization:

One Dimension

Bisection

Bisection and golden section assume we can pick points on the domain straddling a zero

1. Start with (x_L^0, x_H^0) such that $f(x_L^0) < 0$ and $f(x_H^0) > 0$
2. Choose x_M^0 in between (x_L^0, x_H^0)
3. Evaluate $f(x_M^0)$
4. If $f(x_M^0) < 0$ then set $x_L^1 = x_M^0, x_H^1 = x_H^0$, and if $f(x_M^0) > 0$ set $x_L^1 = x_L^0,\ x_H^1 = x_M^0$
5. Check two objectives: $|f(x_M)| < \epsilon_f$ and $|x_H - x_L| < \epsilon_x$
6. If neither is satisfied return to step 2

Bisection splits $x_M = \frac{1}{2}(x_L + x_H)$. Golden section uses two middle values at

$$x_{ML} = x_L + \frac{3 - \sqrt{5}}{2}(x_H - x_L) \quad x_{MH} = x_L + \frac{\sqrt{5} - 1}{2}(x_H - x_L)$$

Quadratic approximation/Brent's Method

We can easily find the minimum of a quadratic function

1. Evaluate at three points $x_0^0, x_1^0, x_2^0 \in [x_L, x_H]$
2. Solve for a, b, c in $ax^2 + bx + c$, i.e. interpolate the three using a quadratic function.
3. Analytically compute the minimum, x_*
4. Replace $\text{argmax}_{x_0^0, x_1^0, x_2^0} f(x)$ with x_* to make $\mathbf{x}^1$
5. Test whether $|x_0^1 - x_1^1|, |x_0^1 - x_2^1|, |x_1^1 - x_2^1| < \epsilon_x$ or whether $|f(x_*) - \max_{x_0^0, x_1^0, x_2^0} f(x)| < \epsilon_f$

Brent's method adds some additional checks to make it robust.

- If the solution x_* is "reasonable," continue with quadratic
- Otherwise, use golden secion search with the points we've already computed to find the next point.

Newton's Method - Root Finding

Uses a 1st order Taylor approx, iteratively

1. Begin at x^0
2. Compute a first-order Taylor expansion around x^0, which requires we compute analytical or numerical derivatives.

$$\tilde{f}(x) = f(x^0) + f'(x^0)(x - x^0)$$

3. We know that at the solution $f(x) = 0$, plugging that in the incremental optimal is:

$$x_* = -\frac{f(x^0)}{f'(x^0)} + x^0$$

4. Check for optimality: $|f(x_*)| < \epsilon_f$ or $|x^0 - x_*| < \epsilon_x$

Newton's Method - Optimization

Uses a 2nd order Taylor approx at x^0

1. Begin at x^0
2. Compute a second-order Taylor expansion around x^0, this involves numerical or analytical derivatives

$$\tilde{f}(x) = f(x^0) + f'(x^0) \cdot (x - x_0) + \frac{1}{2}(x - x_0)^2 f''(x^0)$$

3. Solve for $\tilde{f}'(x_*) = 0$, which in one dimension is

$$x_* = x^0 - \frac{f'(x^0)}{f''(x^0)}$$

4. Check whether $|f(x^0) - f(x_*)| < \epsilon_f$ or $|x^0 - x_*| < \epsilon_x$
5. Set $x^0 = x_*$ and return

Root Finding and Optimization: Multidimensions

Simple iteration for root finding

We wish to solve:

$$0 = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_J(\mathbf{x}) \end{pmatrix}$$

- We will take one dimension at a time, solving for $f_j(\mathbf{x}^0)$
- After solving, we have $\mathbf{x}^1$
- Solve $f_j(\mathbf{x}^1) \forall j$
- Repeat!

Jacobi and Seidel approaches

But, when to plug it back in?

1. Jacobi : solve the j^{th} function as

$$0 = f_j(\mathbf{x}^0) \forall j$$

2. Seidel : solve the j^{th} and then $j+1^{th}$ function as

$$f_j \begin{pmatrix} x_0^1 \\ \vdots \\ x_j^0 \\ \vdots \\ x_J^0 \end{pmatrix} \rightarrow f_{j+1} \begin{pmatrix} x_0^1 \\ \vdots \\ x_j^1 \\ x_{j+1}^0 \\ \vdots \\ x_J^0 \end{pmatrix}$$

Jacobi and Seidel approaches

But, when to plug it back in?

1. Jacobi : solve the j^{th} function as

$$0 = f_j(\mathbf{x}^0) \forall j$$

2. Seidel : solve the j^{th} and then $j+1^{th}$ function as

$$f_j \begin{pmatrix} x_0^1 \\ \vdots \\ x_j^0 \\ \vdots \\ x_J^0 \end{pmatrix} \rightarrow f_{j+1} \begin{pmatrix} x_0^1 \\ \vdots \\ x_j^1 \\ x_{j+1}^0 \\ \vdots \\ x_J^0 \end{pmatrix}$$

Newtons's method in multiple dimensions:

1. Begin at $\mathbf{x}^0$ with a Taylor expansion
2. Compute a Taylor expansion around $\mathbf{x}^0$, this involves numerical or analytical gradient and hessian. For root-finding or optimization we have:

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}^0) + Jf(\mathbf{x}^0) \cdot (\mathbf{x} - \mathbf{x}_0)$$

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}^0) + \nabla f(\mathbf{x}^0) \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)' \nabla^2 f(\mathbf{x}^0)(\mathbf{x} - \mathbf{x}_0)$$

3. Solve for $\tilde{f}(\mathbf{x}_*) = 0$ or $\nabla \tilde{f}(\mathbf{x}_*) = 0$, which is

$$\mathbf{x}_* = \mathbf{x}^0 - |Jf(\mathbf{x}^0)|^{-1} f(\mathbf{x}^0) \quad \text{or} \quad \mathbf{x}_* = \mathbf{x}^0 - |\nabla^2 f(\mathbf{x}^0)|^{-1} \nabla f(\mathbf{x}^0)$$

4. Check whether $|f(\mathbf{x}^0) - f(\mathbf{x}_*)| < \epsilon_f$ or $|\mathbf{x}^0 - \mathbf{x}_*| < \epsilon_x$
5. Set $\mathbf{x}^0 = \mathbf{x}_*$ and return

Quasi-Newton Methods

- The hessian sucks. Especially in multiple dimensions, QN methods help:
  - Do not update the entire Hessian every iteration
  - Use approximation methods that ensure negative semi-definiteness.
- Why is this ok? Look at:

$$\mathbf{x}_* = \mathbf{x}^0 - |\nabla^2 f(\mathbf{x}^0)|^{-1} \nabla f(\mathbf{x}^0)$$

- The Hessian really only gives us the "rate" at which we approach. The gradient gives us the direction.
- A very simple Quasi-Newton would be a constant for $|\nabla^2 f(\mathbf{x}^0)|^{-1}$

Quasi-Newton Methods

- The hessian sucks. Especially in multiple dimensions, QN methods help:
  - Do not update the entire Hessian every iteration
  - Use approximation methods that ensure negative semi-definiteness.
- Why is this ok? Look at:

$$\mathbf{x}_* = \mathbf{x}^0 - |\nabla^2 f(\mathbf{x}^0)|^{-1} \nabla f(\mathbf{x}^0)$$

- The Hessian really only gives us the "rate" at which we approach. The gradient gives us the direction.
- A very simple Quasi-Newton would be a constant for $|\nabla^2 f(\mathbf{x}^0)|^{-1}$
- These are LOCAL methods.
- The true, global Hessian may actually be a hindrance, especially if it's costly to compute and not negative semi-definite.

Discretizing Autoregressive Processes

Discretizing a process

- The general idea is to take a distribution f and find a set of points $\{x_i\}$ and assign to them probability $Pr[x_i] = \int_{c_i}^{c_{i+1}} f(x)dx$ for some set of cut off points $\{c_i\}$: $c_i \in (x_i, x_{i+1})$
- Generally we are concerned with making a markov chain and generally dynamic process will not be row-identical.
- With Tauchen (1986), you can conveniently pick the points of the process at which to evaluate and choose probabilities to suit.
- To accurately approximate a process with the fewest points, we can often do better. However, that may not be the point

Tauchen's Method

- Fit $z' = \rho z + \epsilon \ : \ \epsilon \sim \mathcal{N}(0, \sigma^2)$ with $n \times n$ transition matrix Π and vector of points $\{z_i\}$
- The stationary distribution of $z \sim \mathcal{N}(0, \sigma_z^2) \ : \ \sigma_z^2 = \frac{\sigma^2}{1-\rho^2}$
- Choose a grid for $\{z_i\}$, *tradition:* equidistant cutoffs between $z_0 = -\lambda\sigma_z, \ z_I = \lambda\sigma_z$ and let $c_i = \frac{z_{i+1}+z_i}{2}$
- Because $Pr[z' \in [c_{j-1}, c_j]|z_i] = Pr[\epsilon' \in (c_j - \rho z_i, c_{j-1} - \rho z_i)|z_i]$

$$p_{ij} = \Phi\left(\frac{c_j - \rho z_i}{\sigma}\right) - \Phi\left(\frac{c_{j-1} - \rho z_i}{\sigma}\right)$$

- on the ends: $p_{i0} = \Phi\left(\frac{c_0 - \rho z_i}{\sigma}\right) \quad p_{iI} = 1 - \Phi\left(\frac{c_{I-1} - \rho z_i}{\sigma}\right)$

Rouwenhorst method

- With extremely persistent processes, we often need many points in Tauchen and the preformance is poor.
- Kopecky and Suen (2009) introduce a simple way to construct the matrix. The size-N grid is evenly spaced over the domain.
- The basic idea is to approximate a normal distribution with binomials. CLT says this works
- We can build it recursively, multiplying probabilities of landing in a bin each time.

The performance is good even as the number of discretization points falls

- Distiribution moments are still close
- We still have flexibility to choose which moments to match more closely

Rouwenhorst method

- There are 2 ways of deriving the transition matrix.
- Consider the matrix-recursive form: For $N = 2$,

$$\Pi_2 = \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix}$$

Then, as $N > 2$:

$$\begin{aligned} \Pi_{N\geq 3} =& p \begin{pmatrix} \Pi_{N-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{pmatrix} + (1-p) \begin{pmatrix} \mathbf{0} & \Pi_{N-1} \\ 0 & \mathbf{0}^T \end{pmatrix} \\ &+ (1-q) \begin{pmatrix} \mathbf{0}^T & 0 \\ \Pi_{N-1} & \mathbf{0} \end{pmatrix} + q \begin{pmatrix} 0 & \mathbf{0}^T \\ \mathbf{0} & \Pi_{N-1} \end{pmatrix} \end{aligned}$$

- At each step, normalize to be a stochastic matrix: divide by 2
- This has the flavor of using binomials to approximate a normal.

Rouwenhorst's method for an AR(1)

- We can choose p, q to hit more rich process
- But, because z is normal and we restrict the grid, there is little freedom.
- Let $p = q = \frac{1+\rho}{2}$ to hit the autocorrelation
- Spacing is determined to hit variance

Table 1: Selected Moments of the Markov Chain

Conditional Mean	$\mathrm{E}(y_{t+1}\|y_t = \overline{y}_i)$	$(q-p)\,\psi + (p+q-1)\,\overline{y}_i$
Conditional Variance	$\mathrm{var}(y_{t+1}\|y_t = \overline{y}_i)$	$\frac{4\psi^2}{(N-1)^2}\left[(N-i)\,(1-p)\,p + (i-1)\,q\,(1-q)\right]$
Unconditional Mean	$\mathrm{E}(y_t)$	$\frac{(q-p)\psi}{2-(p+q)}$
Unconditional Second Moment	$\mathrm{E}(y_t^2)$	$\psi^2\left\{1 - 4s\,(1-s) + \frac{4s(1-s)}{N-1}\right\}$
First-order Autocovariance	$\mathrm{Cov}(y_t, y_{t+1})$	$(p+q-1)\mathrm{var}(y_t)$
First-order Autocorrelation	$\mathrm{Corr}(y_t, y_{t+1})$	$p+q-1$

More on Rouwenhorst's method

- p^{N-1} is the probability of staying in the high state
- q^{N-1} the probability of staying in the low state
- $p \neq q$ implies that shocks are heteroskedastic, which might be a goal, but not in the normal case
- For the grid size, choose evenly spaced over $[\mu - \nu, \mu + \nu]$, where $\nu = \sqrt{\frac{N-1}{1-\rho^2}}\sigma_\epsilon$
- This will match the unconditional variance.

How to simulate from a Markov Chain

We begin with a vector of points $\mathbf{z}$ and a transition matrix Π

There are 2 methods to initialize the starting state:

1. Initialize from the unconditional distribution
   - Use Π to get the unconditional distribution: $\bar{\pi}$
   - Draw a uniform random variable $x \in [0, 1]$
   - Set $z_0 = z_i \in \mathbf{z}$ such that $x \leq \sum_{j=1}^{i} \bar{\pi}_j$
2. Initialize from an ad hoc/arbitrary state $z_0 \in \mathbf{z}$

How to simulate from a Markov Chain

For each draw in the chain:

1. Begin with $z_{i,t-1}$, the value in state i from last period
2. Draw a uniform random variable $x \in [0, 1]$
3. Set $z_{k,t} \in \mathbf{z}$ such that $x \leq \sum_{j=1}^{i} \pi_{i,j}$

Things to remember:

- Store your sequence of x_t or set a "seed" so that every time you run your program, you get the same results.
- Often, drawing the whole vector, the history of x_t is faster than drawing one at a time
- Try to estimate the AR(1) process given your simulation to check the accuracy