Lecture 5: Global Solution Methods (continued)

Victoria Gregory

July 13, 2017

New York University

Road Map

Continue illustrating methods for solving a dynamic programming problem **globally**. Today's topics:

Howard Improvement

Policy Function Iteration

Endogenous Grid Method

Collocation

Howard Improvement

Howard Improvement

For the moment, go back to value function iteration method Chase talked about last week.

- Choosing the maximizing policy is typically the most time-consuming part of these algorithms
- **Idea:** reduce the number of times we update the policy function rather than the value function
- On some iterations, use the current guess for the policy function to update the value function
- This works because the policy function typically converges faster than the value function

Howard Improvement

First need to choose a parameter H, the number of times you update the value function using the existing policy function

- Requires some experimentation
- Can increase H after each iteration
- Start using Howard Improvement only after doing VFI a few times
- Too high of an H can cause the value function to move too far away from the true one

Howard Improvement

1. Choose grids on state variables: $\mathcal{A} \times \mathcal{S}$
2. Initial guess of value function: $V_0(a, s)\ \forall (a, s) \in \mathcal{A} \times \mathcal{S}$
3. Given V_j, for each $(a_t, s_t) \in \mathcal{A} \times \mathcal{S}$

   3.1 Find $a_{t+1} \in \mathcal{A}$ such that

   $$a_{t+1} \in \text{argmax}\ u(y(s_t) + (1 + r)a_t - a_{t+1}) + \beta E\left[V_j(a_{t+1}, s_{t+1})\right]$$

   3.2 Update policy function

   $$a^*(a_t, s_t) = a_{t+1}$$

   3.3 Update value function

   $$V_{j+1}(a_t, s_t) = u(c_t) + \beta E\left[V_j(a_{t+1}, s_{t+1})\right]$$

4. If $d(V_{j+1}, V_j) < \varepsilon$ then done, otherwise return to 3

Policy Function Iteration

Income Fluctuation Problem

- Idiosyncratic state variables: (a, y)
- Discretize the income process: y_j for $j = 1, \ldots, N$
- Transition function $\pi(y'|y)$
- Household problem in recursive form:

$$V(a, y) = \max_{c, a'} u(c) + \beta \sum_{y' \in Y} \pi(y'|y) V(a', y')$$

subject to:

$$c + a' \leq Ra + y$$

$$a' \geq -\phi$$

Policy Function Iteration

Euler equation, after substituting budget constraint:

$$u_c(Ra + y_j - a') - \beta R \sum_{y_i \in Y} \pi(y_i|y_j) u_c(Ra' + y_i - a'') \geq 0$$

Idea:

- Guess a policy function instead of a value function
- Iterate on the Euler equation instead of the Bellman equation
- Policy function typically converges faster than the value function

Algorithm

1. Construct a grid on the asset space $\{a_0, a_1, \ldots, a_m\}$, where $a_0 = -\phi$.
2. Guess a policy function for a'' on the grid points: $\hat{a}_0(a_i, y_j)$.
3. For every point on the grid (a_i, y_j), check whether the borrowing constraint binds:

$$u_c(Ra_i + y_j - a_0) - \beta R \sum_{y' \in Y} \pi(y'|y_j) u_c(Ra_0 + y' - \hat{a}_0(a_0, y_j)) > 0$$

4. **If the inequality holds**, the borrowing constraint binds. Set $a'_0(a_i, y_j) = a_0$ and repeat step 3 for the next grid point.

Algorithm (cont'd)

5. **If the inequality does not hold**, there is an interior solution. Use an non-linear solver to find the solution a^* of:

$$u_c(Ra_i + y_j - a^*) - \beta R \sum_{y' \in Y} \pi(y'|y_j) u_c(Ra^* + y' - \hat{a}_0(a^*, y_j)) > 0$$

   - To evaluate $\hat{a}_0(a, y')$ outside the grid points, use linear interpolation.
   - For a given a^*, find the adjacent grid points $\{a_i, a_{i+1}\}$ and compute

$$\hat{a}_0(a^*, y') = \hat{a}_0(a_i, y') + (a^* - a_i) \left(\frac{\hat{a}_0(a_{i+1}, y') - \hat{a}_0(a_i, y')}{a_{i+1} - a_i} \right)$$

   Set $a'_0(a_i, y_j) = a^*$ and repeat step 3 for the next grid point.

Algorithm (cont'd)

6. Check convergence by comparing $a'_0(a_i, y_j)$ to $\hat{a}_0(a_i, y_j)$. Declare convergence on iteration n if:

$$\max_{i,j}\{|a'_n(a_i, y_j) - \hat{a}_n(a_i, y_j)|\} < \varepsilon$$

7. Stop if convergence is achieved, otherwise go back to step 3 with new guess $\hat{a}_1(a_i, y_j) - a'_0(a_i, y_j)$.

Endogenous Grid Method

Endogenous Grid Method

Euler equation:

$$u_c(c(a,y)) \geq \beta R \sum_{y' \in Y} \pi(y'|y) u_c(c(a',y'))$$

(with equality if $a' > \phi$)

Idea:

- Instead of building a grid over a, build a grid over a', next period's asset holdings
- Iterate over the consumption policy $c(a,y)$

Algorithm

1. Construct a grid for (a', y). The bottom point on the grid for a' should be the borrowing limit, $-\phi$.
2. Guess a consumption policy, $\hat{c}_0(a_i, y_j)$. Reasonable initial guess: $\hat{c}_0(a_i, y_j) = ra_i + y_j$
3. For each pair $\{a'_i, y_j\}$ construct the right-hand side of the Euler equation using the current guess for consumption $\hat{c}_0$:

$$B(a'_i, y_j) = \beta R \sum_{y' \in Y} \pi(y'|y_j) u_c(\hat{c}_0(a'_i, y'))$$

(RHS of the Euler equation with a'_i assets tomorrow when your income today is y_j)

Algorithm (cont'd)

4. **Key step**: now we can use the Euler equation to find $\tilde{c}(a_i', y_j)$ that satisfies

$$u_c(\tilde{c}(a_i', y_j)) = B(a_i', y_j)$$

This can be done analytically: if $u_c(c) = c^{-\gamma}$, then $\tilde{c}(a_i', y_j) = [B(a_i', y_j)]^{-\frac{1}{\gamma}}$.

Advantages:

- Don't need to use a nonlinear solver!
- Consequently, only need to compute the expectation in step 3 once.

Note that this requires u_c to be invertible.

Algorithm (cont'd)

5. Now that we have consumption, can solve for assets today that would lead the consumer to have a_i' assets tomorrow is his current income shock is y_j: $a^*(a_i', y_j)$. We can back this out of the budget constraint:

$$\tilde{c}(a_i', y_j) + a_i' = Ra_i^* + y_j$$

   Now you have $c(a_i^*, y_j) = \tilde{c}(a_i', y_j)$, which is **not** defined on the original grid – this is the **endogenous grid**.

6. Let a_0^* be the value of assets that induces the borrowing constraint to bind next period (the value of a^* at the bottom grid point). You'll have one for each possible income state.

Algorithm (cont'd)

7. To **update the guess for consumption on the original grid**, $\hat{c}_1(a_i, y_j)$:
   - On grid points $a_i > a_0^*$, find a_n^*, a_{n+1}^* such that $a_n^* < a_i < a_{n+1}^*$ and use linear interpolation between $c(a_n^*, y_j)$ and $c(a_{n+1}^*, y_j)$ to get $\hat{c}_1(a_i, y_j)$.
   - On grid points $a_i < a_0^*$, use the budget constraint:

     $$\hat{c}_1(a_i, y_j) = Ra_i + y_j - a_0'$$

     since we know that borrowing constraint is going to be binding next period, so the Euler equation won't hold with equality.
8. Check convergence:

$$\max_{i,j}\{|c_{n+1}(a_i, y_j) - c_n(a_i, y_j)|\} < \varepsilon$$

Collocation

Collocation

- Main idea
  - Approximate the value function with a linear combination of basis functions
  - Need to solve for the coefficients on these basis functions
  - Family of basis functions is chosen by the economist
- Algorithm based on the one on Simon's website
- Uses tools from Laszlo's lecture on function approximation (now, the function we need to approximate is unknown)

Advantages

- Very fast, stable
- Many objects only need to be computed once, ahead of time
- End up with a linear system on each iteration
- Once you've solved for the coefficients, you can easily evaluate the value and policy function on any grid
- Extends to other settings
- Julia package `BasisMatrices.jl` very convenient for this setup
- If you're using MATLAB, everything is vectorized

Agent's Problem

$$V(a,y) = \max_{a' \in B(a,y)} u(Ra + y - a') + \beta \sum_{y' \in Y} \pi(y,y') V(a',y')$$

where $B(a,y) = [-\phi, Ra + y]$.

We can separate the expected value function to get the following system:

$$V(a,y) = \max_{a' \in B(a,y)} u(Ra + y - a') + \beta V_e(a',y)$$

$$V_e(a,y) = \sum_{y' \in Y} \pi(y,y') V(a,y')$$

Idea: if we directly approximate the expected value function, we can avoid computing the expectation each time we solve for a'.

Rewriting the System

- Let $\mathbf{a}$ be a length N_a grid for assets
- Let $\mathbf{y}$ be a length N_y grid for income, with transition matrix Π
- Stack each combination of these to get a set of collocation nodes $\mathbf{s}$, which is $N \times 2$ where $N = N_a N_y$ – `gridmake` will do this for any number of grids
- Let s_i be an asset, income pair, or a row in $\mathbf{s}$

In this notation, the system becomes:

$$V(s_i) = \max_{a' \in B(s_i)} u(Rs_{i1} + s_{i2} - a') + \beta V_e([a', s_{i2}])$$

$$V_e(s_i) = \sum_{y' \in Y} \pi(s_{i2}, y') V([s_{i1}, y'])$$

Interpolants

Let's replace the value functions with interpolants, consisting of a basis function ϕ and sets of coefficients $\{c_j\}_{j=1}^{N}$.

$$V(s_i) = \sum_{j=1}^{N} \phi(s_i) c_j$$

$$V^e(s_i) = \sum_{j=1}^{N} \phi(s_i) c_j^e$$

- Approximating functions can be splines, Chebyshev, linear, etc.
- N coefficients and N points on the state space

Substitute these back into our system:

$$\sum_{j=1}^{N} \phi(s_i) c_j = \max_{a' \in B(s_i)} u(R s_{i1} + s_{i2} - a') + \beta \sum_{j=1}^{N} \phi([a', s_{i2}]) c_j^e$$

$$\sum_{j=1}^{N} \phi(s_i) c_j^e = \sum_{y' \in Y} \pi(s_{i2}, y') \sum_{j=1}^{N} \phi([s_{i1}, y']) c_j$$

Linear system in $2N$ equations and $2N$ unknowns.

Stacking the System

- $V(s_i) = \sum_{j=1}^{N} \phi(s_i) c_j$ becomes $\mathbf{V}(s) = \Phi(s)c$
  - $\Phi(s)$ is the basis matrix of the interpolant evalueated at s
  - $c = (c_1, \ldots, c_N)'$ is the vector of coefficients
- So the first equation becomes

$$\Phi(s)c = \max_{a' \in \mathbf{B}(s)} \mathbf{u}(s, a') + \beta \Phi([a', s_2])c^e$$

- The second equation becomes

$$\Phi(s)c^e = (\Pi \otimes \mathbf{I}_{N_a})\Phi(s)c$$

System to Solve

$$\Phi(s)c = \max_{a' \in \mathbf{B}(s)} \mathbf{u}(s, a') + \beta \Phi([a', s_2])c^e$$

$$\Phi(s)c^e = (\Pi \otimes \mathbf{I}_{N_a})\Phi(s)c$$

How to implement this computationally?

1. Compute most of the basis matrices in advance
2. Given a guess for $\{c, c^e\}$, compute the max
3. Iterate on the equations, update coefficients until convergence

Computing Basis Matrices

- Create a `Basis` object to set up the type of interpolant, collocation nodes, etc.:

```
basis = Basis(SplineParams(agrid0, 0, order),
              LinParams(ygrid0, 0))
s, (agrid, ygrid) = nodes(basis)
```

- To compute the basis matrix, $\Phi(s)$:

```
bs = BasisMatrix(basis, Direct(), s, [0 0])
Φ = convert(Expanded, bs).vals[1]
```

Computing Basis Matrices

What else can be stored in advance?

- $(\Pi \otimes I_{N_a})\Phi(s)$

```
Emat = kron(Π, speye(Na))*Φ
```

- Part of $\Phi([a', s_2])$
  - This is really a tensor product: $\Phi_y(y)\Phi_{a'}(a')$
  - Since the values of y we want to evaluate at never change, we can compute just that part of the interpolation in advance, and then later on create the basis matrix for a'.

```
Φy = bs.vals[2]          # store this
...
bs_a = BasisMatrix(h.basis[1], Direct(), ap, 0)
Φa = bs_a.vals[1]
Φapy = row_kron(h.Φy, Φa)
```

  - Less basis matrices need to be computed by the solver

Optimization Step

$$\Phi(s)c = \max_{a' \in \mathbf{B}(s)} \mathbf{u}(s, a') + \beta \Phi([a', s_2])c^e$$

$$\Phi(s)c^e = (\Pi \otimes \mathbf{I}_{N_a})\Phi(s)c$$

- Now that we know how to compute basis matrices, it's straightforward to compute the objective function given a guess for the coefficients
- Golden search works well for computing the max (at least in one dimension)

```
lower_bound = zeros(size(s, 1), )
upper_bound = (1 + h.r).*s[:, 1] + s[:, 2]
f(ap::Vector{Float64}) = value(h, s, ap)
ap, v1 = golden_method(f, lower_bound, upper_bound)
```

Iteration

How to update the guesses for c, c^e?

1. Bellman iteration
2. Newton iteration (linearity of the system allows us to write down the Jacobian)

Bellman Iteration

Compute the right-hand side and invert $\Phi(s)$ to get the new coefficient guesses:

$$c_{k+1} = \Phi(s)^{-1}\left[\mathbf{u}(s, a'(s)) + \beta\Phi([a'(s), s_2])c_k^e\right]$$
$$c_{k+1}^e = \Phi(s)^{-1}\left[(\Pi \otimes \mathbf{I}_{N_a})\Phi(s)c_k\right]$$

- Do this a few times to get a decent initial guess for the Newton iterations
- Sometimes useful to let this run for a while to make sure you have a contraction mapping

Newton Iteration

View this as a root-finding problem

$$\mathbf{g}_1(c, c^e) = \Phi(s)c - \left[\mathbf{u}(s, a'(s)) + \beta\Phi([a'(s), s_2])c^e\right]$$
$$\mathbf{g}_2(c, c^e) = \Phi(s)c^e - [(\Pi \otimes \mathbf{I}_{N_a})\Phi(s)c]$$

Because the system is linear in the coefficients, we can easily write down the Jacobian:

$$\mathbf{D}(c, c^e) = \begin{bmatrix} \Phi(s) & -\beta\Phi([a'(s), s_2]) \\ -(\Pi \otimes \mathbf{I}_{N_a})\Phi(s) & \Phi(s) \end{bmatrix}$$

Updating:

$$\begin{bmatrix} c_{k+1} \\ c^e_{k+1} \end{bmatrix} = \begin{bmatrix} c_k \\ c^e_k \end{bmatrix} - \mathbf{D}(c_k, c^e_k)^{-1} \begin{bmatrix} \mathbf{g}_1(c_k, c^e_k) \\ \mathbf{g}_2(c_k, c^e_k) \end{bmatrix}$$

These converge **much** faster than the Bellman iterations!

Homework

Solve the following problem of firm investment:

$$V(k,z) = \max_{i,n} zk^{\alpha}n^{\nu} - i - \eta(k' - k)^2 - wn + \beta\mathbb{E}\left[V(k',z')\right]$$

where $k' = (1-\delta)k + i$.

- Eliminate n by solving static labor choice problem
- Parameters: $w = 0.99$; $w = 1.26$; $\alpha = 0.26$; $\nu = 0.64$; $\delta = 0.07$; $\eta = 0.5$
- z follows a Markov chain with values [0.7578 0.9117 1.0969 1.3196]′ and transition matrix:

$$\begin{bmatrix} 0.9269 & 0.0713 & 0.0018 & 0 \\ 0.0238 & 0.9281 & 0.0475 & 0.0001 \\ 0.0001 & 0.0475 & 0.9281 & 0.0238 \\ 0 & 0.0018 & 0.0713 & 0.9269 \end{bmatrix}$$