Advanced Tools in Macroeconomics

Continuous time models (and methods)

Pontus Rendahl

2023

Plan for these slides

1. Speeding things up while making them more robust
   1.1 Use the "implicit method" to somewhat bring back the contraction property
   1.2 Make use of sparsity

Why is the contraction property lost?

- The Ramsey growth model in continuous time is

$$\rho v(k) = \max_c \{u(c) + v'(k)(f(k) - \delta k - c)\}.$$

- The explicit method suggests

$$\rho v_{n+1}(k) = \max_c \{u(c) + v_n'(k)(f(k) - \delta k - c)\}.$$

- But this breaks down without a (very small) smoothing parameter

Why is the contraction property lost?

- Why? Consider the deterministic Ramsey growth model in discrete time

$$v(k) = \max_c \{u(c) + (1-\rho)v(f(k) + (1-\delta)k - c)\}.$$

- In discrete time we iterate as

$$v_{n+1}(k) = \max_c \{u(c) + (1-\rho)v_n(f(k) + (1-\delta)k - c)\},$$

- This is a contraction mapping and we know that $v_n \to v$.

Why is the contraction property lost?

- Let's, heuristically, convert this into continuous time

$$v_{n+1}(k) = \max_c \{\Delta u(c) + (1 - \Delta\rho) v_n(k + \Delta(f(k) - \delta k - c))\}.$$

$$0 = \max_c \{u(c) + \frac{v_n(k + \Delta(f(k) - \delta k - c)) - v_{n+1}(k)}{\Delta} \\ - \rho v_n(k + \Delta(f(k) - \delta k - c))\}.$$

Taking limits and rearranging

$$\rho v_n(k) = \max_c \{u(c) \\ + \lim_{\Delta \to 0} \frac{v_n(k + \Delta(f(k) - \delta k - c)) - v_{n+1}(k)}{\Delta}\}.$$

Why is the contraction property lost?

- **Problem 1:**

$$\lim_{\Delta \to 0} \frac{v_n(k + \Delta(f(k) - \delta k - c)) - v_{n+1}(k)}{\Delta} \neq v_n'(k)(f(k) - \delta - c)$$

  The right hand side of the HJB equation contains v_{n+1}.
- **Problem 2:** The left hand side of the HJB equation is v_n.

Why is the contraction property lost?

- Take a look at the previous equation

$$\rho v_n(k) = \max_c \{u(c) + \lim_{\Delta \to 0} \frac{v_n(k + \Delta(f(k) - \delta k - c)) - v_{n+1}(k)}{\Delta}\}.$$

- Subtract $v_n(k)/\Delta$ from both sides and rearrange

$$\lim_{\Delta \to 0} \frac{v_{n+1}(k) - v_n(k)}{\Delta} = \max_c \{u(c) + \lim_{\Delta \to 0} \frac{v_n(k + \Delta(f(k) - \delta k - c)) - v_n(k)}{\Delta} - \rho v_n(k)\}.$$

Why is the contraction property lost?

$$\lim_{\Delta \to 0} \frac{v_{n+1}(k) - v_n(k)}{\Delta} = \max_c \{u(c) + \lim_{\Delta \to 0} \frac{v_n(k + \Delta(f(k) - \delta k - c)) - v_n(k)}{\Delta} - \rho v_n(k)\}.$$

- Which is approximately

$$v_{n+1}(k) = v_n(k) + \Gamma \times \max_c \{u(c) + v_n'(k)(f(k) - \delta k - c) - \rho v_n(k)\}.$$

if Γ is sufficiently small!

Is the contraction property lost?

$$v_{n+1}(k) = v_n(k) + \Gamma \times \max_c \{u(c) + v_n'(k)(f(k) - \delta k - c) - \rho v_n(k)\}.$$

- This is just a reformulation of the explicit method.
- But it gives and indication why it can converge if the smoothing parameter is close to zero.
- In any case, there is a better better method available, and we will go through it next.

How to get it back

- Back to discrete time

$$v_{n+1}(k) = \max_c \{u(c) + (1-\rho)v_n(f(k) + (1-\delta)k - c)\},$$

- Call the optimal choice c_n (it's really a function of k but I'm saving some space)
- Howard's Improvement Algorithm says that we can then iterate on

$$v_{n+1}^{h+1}(k) = u(c_n) + (1-\rho)v_{n+1}^{h}(f(k) + (1-\delta)k - c_n)\},$$

  with $v_{n+1}^0 = v_n$.
- Until $v_{n+1}^{h+1} \approx v_{n+1}^{h}$. This can speed things up considerably, and preserves the contraction property

How to get it back

- Suppose that it holds exactly $v_{n+1}^{h+1} = v_{n+1}^{h}$, and let's just call this function v_{n+1}. Then it must satisfy

$$v_{n+1}(k) = u(c_n) + (1-\rho)v_{n+1}(f(k) + (1-\delta)k - c_n),$$

- In Δ units of time

$$v_{n+1}(k) = \Delta u(c_n) + (1-\Delta\rho)v_{n+1}(k + \Delta(f(k) - \delta k - c_n)).$$

How to get it back

Rearrange

$$0 = u(c_n) + \frac{v_{n+1}(k + \Delta(f(k) - \delta k - c_n)) - v_{n+1}(k)}{\Delta} \\ - \rho v_{n+1}(k + \Delta(f(k) - \delta k - c_n))\}.$$

and take limits

$$\rho v_{n+1}(k) = u(c_n) + v'_{n+1}(k)(f(k) - \delta k - c_n),$$

How to get it back

$$\rho v_{n+1}(k) = u(c_n) + v'_{n+1}(k)(f(k) - \delta k - c_n),$$

- Now the awkward discrepancy between v_{n+1} and v_n is gone!
- But the problem looks a bit hard to solve!
- Turns out it is not!
- This is where the "implicit method" comes in.

The implicit method

1. Start with a grid for capital $\mathbf{k} = [k_1, k_2, \ldots, k_N]$.
2. For each grid point for capital you have a guess for $v_0(k_i)$, $\forall k_i \in \mathbf{k}$
3. So you have a vector of N values of v_0. Call this $\mathbf{v}_0$
4. You should also have a difference operator (an $N \times N$ matrix) $\mathbf{D}$ such that

$$\mathbf{D}\mathbf{v} \approx \mathbf{v}'(k), \quad \forall k_i \in \mathbf{K}$$

The implicit method

5. Optimal consumption choice given by FOC

$$u'(\mathbf{c}_0) = \mathbf{D}\mathbf{v}_0$$

   reasonable to call this $c(\mathbf{v}_0)$ – an $N \times 1$ vector

6. This implies another $N \times 1$ vector of savings

$$\mathbf{s}_0 = (f(\mathbf{k}) - \delta\mathbf{k} - c(\mathbf{v}_0))$$

   (This vector can be used to improve on $\mathbf{D}$ – more on that in a second).

The implicit method

- So far, this implies that we can write

$$\rho \mathbf{v}_1 = u(c(\mathbf{v}_0)) + (\mathbf{D}\mathbf{v}_1) \cdot \mathbf{s}_0$$

  where the dot indicates element-by-element multiplication.
- We are close to a linear system though!

Thus we will use this trick:

7. Create the $N \times N$ matrix $\mathbf{S}_0 = diag(\mathbf{s_0})$

The implicit method

That is

$$\mathbf{S} = \begin{pmatrix} s_1 & 0 & \dots & 0 \\ 0 & s_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & s_N \end{pmatrix},$$

The implicit method

8. Then we have

$$\begin{aligned} \rho \mathbf{v}_1 =& u(c(\mathbf{v}_0)) + (\mathbf{D}\mathbf{v}_1) \cdot \mathbf{s}_0 \\ =& u(c(\mathbf{v}_0)) + \mathbf{S}_0 \mathbf{D}\mathbf{v}_1 \end{aligned}$$

9. This is a linear system. Thus, manipulate

$$(\rho \mathbf{I} - \mathbf{S}_0 \mathbf{D})\mathbf{v}_1 = u(c(\mathbf{v}_0))$$

10. Lastly

$$\mathbf{v}_1 = (\rho \mathbf{I} - \mathbf{S}_0 \mathbf{D})^{-1} u(c(\mathbf{v}_0))$$

11. Generally

$$\mathbf{v}_{n+1} = (\rho \mathbf{I} - \mathbf{S}_n \mathbf{D})^{-1} u(c(\mathbf{v}_n))$$

The implicit method

Or even more generally

$$\mathbf{v}_{n+1} = ((\rho + 1/\Gamma)\mathbf{I} - \mathbf{S}_n\mathbf{D})^{-1}[u(c(\mathbf{v}_n)) + \mathbf{v}_n/\Gamma]$$

for Γ very large (my experience: $\Gamma = \infty$ is fastest, but set lower if convergence issues arise)

- In matlab always use backslash operator to calculate $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. I.e. $\mathbf{x} = \mathbf{A}\backslash\mathbf{b}$

We are talking very substantial speed/robustness gains here. Perhaps by a factor of 1,000.

The implicit method: Improvement trick I

- Yesterday we created the matrix $\mathbf{D}$ as central differences
- We can do better. In particular, $\mathbf{s}_n$ tells us where the economy is drifting for each $k_i \in \mathbf{k}$
- So trick one is to use forward differences for all

$$\{k_i \in \mathbf{k} : s_i > 0\}$$

- and backward differences for all

$$\{k_i \in \mathbf{k} : s_i < 0\}$$

- This leads to

$$\mathbf{v}_{n+1} = ((\rho + 1/\Gamma)\mathbf{I} - \mathbf{S}_n\mathbf{D}_n)^{-1}[u(c(\mathbf{v}_n)) + \mathbf{v}_n/\Gamma]$$

The implicit method: Improvement trick II

- Inspect the matrix

$$((\rho + 1/\Gamma)\mathbf{I} - \mathbf{S}_n\mathbf{D}_n),$$

  and notice that all matrices are super sparse!
- So declaring them as sparse will free up a lot of memory and give you enormous speed gains too (this is particularly true for problems with $N > 200$ or so. Below that it doesn't really matter).
- **Never** declare any of these matrices as anything else than sparse! Use commands as `speye` and `spdiags`
- **Don't** be too concerned about loops. That doesn't seem to be what can clog these systems.