Econ 202A Macroeconomics: Section 3

Kiyea Jin

November 6, 8, 2024

Section 3

Overview

1. Neoclassical Growth Model
   - Upwind Scheme
   - Boundary Conditions
   - Sparse Matrix Routines
2. Huggett (1993) Model
   - Model Overview
   - Markov Chain Generator

Section 3-1: Neoclassical Growth Model

Implicit Method: Matrix Representation

1. Define I discrete grid points for k, denoted as k_i for $i = 1, \dots, I$, and form an $I \times 1$ vector $\mathbf{k} = [k_1, k_2, \dots, k_I]'$.
2. Let $V_i = V(k_i)$. For each k_i on the grid, make an initial guess for the value function as an $I \times 1$ vector $\mathbf{V^0} = [V_1^0, V_2^0, \dots, V_I^0]'$.
3. Compute the derivative of the value function as an $I \times 1$ vector $(\mathbf{V^n})'$ using an $I \times I$ difference matrix operator $\mathbf{D}$ such that $\mathbf{DV^n} \simeq (\mathbf{V^n})'$.
4. Compute the optimal consumption as an $I \times 1$ vector $\mathbf{c^n}$ from $\mathbf{c^n} = (U')^{-1}(\mathbf{DV^n})$.
5. Compute the optimal savings as an $I \times 1$ vector $\mathbf{s^n}$ from $\mathbf{s^n} = f(\mathbf{k}) - \delta\mathbf{k} - \mathbf{c^n}$.
6. Find $\mathbf{V^{n+1}}$ from:
$$\frac{1}{\Delta}(\mathbf{V^{n+1}} - \mathbf{V^n}) + \rho\mathbf{V^{n+1}} = U(\mathbf{c^n}) + (\mathbf{DV^{n+1}}) \cdot \mathbf{s^n}$$
where the dot indicates element-wise multiplication.
7. If $\mathbf{V^{n+1}}$ is close enough to $\mathbf{V^n}$: stop. Otherwise, go to step 3.

Matrix Representation

Alternative matrix formulation:

$$\frac{1}{\Delta}(\mathbf{V^{n+1}} - \mathbf{V^n}) + \rho\mathbf{V^{n+1}} = U(\mathbf{c^n}) + \mathbf{S^n}\mathbf{D}\mathbf{V^{n+1}}$$

where $\mathbf{S^n} = \text{diag}(\mathbf{s^n})$ is an $I \times I$ diagonal matrix with diagnoals $\mathbf{s^n} = \{s_1^n, \cdots, s_I^n\}$.

Equivalently, solve the linear system:

$$\mathbf{V^{n+1}} = \left((\rho + \frac{1}{\Delta})\mathbf{I} - \mathbf{S^n}\mathbf{D}\right)^{-1}\left[U(\mathbf{c^n}) + \frac{1}{\Delta}\mathbf{V^n}\right] \tag{1}$$

Numerical Solution: Finite Difference Method

Exercise: Numerical Solution of the Neoclassical Growth Model

In this exercise, apply the finite difference method with a mixed method to numerically solve the Hamilton-Jacobi-Bellman (HJB) equation for the Neoclassical Growth Model:

$$\rho V(k) = \max_c \{U(c) + V'(k) \cdot (f(k) - \delta k - c)\} \tag{2}$$

Finite Difference Approximation

The finite difference approximations to HJB equation, associated with the FOC is:

$$\begin{aligned} &\rho V_i = U(c_i) + V_i'(k_i^\alpha - \delta k_i - c_i) \\ &\text{with } \; c_i = (U')^{-1}(V_i') \end{aligned} \tag{3}$$

where $i = 1, \cdots, I$, $V_i = V(k_i)$ with a uniform step size $\Delta k = k_{i+1} - k_i$.

Key Challenges

1. Approximating the derivative of the value function, V_i'.
   - Mixed Method
   - **Upwind Scheme**
2. Solving the system, which is highly non-linear, requires iterative schemes.
   - Explicit Method
   - Implicit Method

Mixed Method

The mixed method approximation for V_i' is defined as:

$$V_i' \simeq \begin{cases} V_{i,F}' = \dfrac{V_{i+1} - V_i}{\Delta k}, & i = 1 \\[2ex] V_{i,C}' = \dfrac{V_{i+1} - V_{i-1}}{2\Delta k}, & i \in \{2, 3, \cdots, I-1\} \\[2ex] V_{i,B}' = \dfrac{V_i - V_{i-1}}{\Delta k}, & i = I \end{cases} \tag{4}$$

Upwind Scheme

- Best finite difference approximation in this context: so-called "Upwind Scheme."

Upwind Scheme

- Best finite difference approximation in this context: so-called "Upwind Scheme."
- Rough idea:
  — Use forward difference whenever the drift of state variable is positive.
  — Use backward difference whenever the drift of state variable is negative.

Why Upwind Scheme

- One might wonder why we would want to use such approximations, since centered approximations are more accurate.

Why Upwind Scheme

- One might wonder why we would want to use such approximations, since centered approximations are more accurate.

  $\Rightarrow$ The upwind scheme is preferred for its stability and alignment with the directional dynamics of the HJB equation.

Why Upwind Scheme

Consider the following one-dimensional linear advection equation:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0$$

which describes a wave propagating along the x-axis with a velocity a.

In the advection equation, there is an asymmetry due to directional translation at speed a. If $a > 0$, the solution moves to the right; if $a < 0$, it moves to the left. Recognizing this asymmetry, it is often optimal to use one-sided differences in the relevant direction (LeVeque, 2007).

Upwind Scheme in This Context

Compute optimal savings according to both the forward and backward difference approximations $V'_{i,F}$ and $V'_{i,B}$:

$$
\begin{aligned}
s_{i,F} &= k_i^{\alpha} - \delta k_i - (U')^{-1}(V'_{i,F}) \\
s_{i,B} &= k_i^{\alpha} - \delta k_i - (U')^{-1}(V'_{i,B})
\end{aligned}
\tag{5}
$$

Upwind Scheme in This Context

Compute optimal savings according to both the forward and backward difference approximations $V'_{i,F}$ and $V'_{i,B}$:

$$\begin{aligned} s_{i,F} &= k_i^\alpha - \delta k_i - (U')^{-1}(V'_{i,F}) \\ s_{i,B} &= k_i^\alpha - \delta k_i - (U')^{-1}(V'_{i,B}) \end{aligned} \tag{5}$$

Then, use the following approximation for V'_i, assuming the value function is concave:

$$V'_i = V'_{i,F}\mathbf{1}_{\{s_{i,F}>0\}} + V'_{i,B}\mathbf{1}_{\{s_{i,B}<0\}} + \bar{V}'_i\mathbf{1}_{\{s_{i,F}\leq 0\leq s_{i,B}\}}, \tag{6}$$

where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function, and $\bar{V}'_i = U'(c_i) = U'(k_i^\alpha - \delta k_i)$.

Boundary Conditions

- All our work so far applies only within the interior of the state space.

Boundary Conditions

- All our work so far applies only within the interior of the state space.
- How many boundary conditions are required?

Boundary Conditions

- All our work so far applies only within the interior of the state space.
- How many boundary conditions are required?
- Key intuition: Two boundary inequalities do same job as one boundary equality.

Boundary Conditions

- All our work so far applies only within the interior of the state space.
- How many boundary conditions are required?
- Key intuition: Two boundary inequalities do same job as one boundary equality.
- In practice, the stability of the algorithm improves by imposing a state constraint $k_{min} \leq k \leq k_{max}$, where k_{min} and k_{max} represent the lower and upper bounds of the state space used in computations.

Boundary Conditions

We aim to keep households within the state space, so we enforce that they neither borrow as $k \to k_{min}$ nor save as $k \to k_{max}$.

Boundary Conditions

We aim to keep households within the state space, so we enforce that they neither borrow as $k \to k_{min}$ nor save as $k \to k_{max}$.

The constraint $k \geq k_{min}$ is enforced by setting

$$V'_{1,B} = U'(f(k_1) - \delta k_1) \tag{7}$$

The state constraint is applied whenever the forward difference approximation would yield negative savings, $s_{1,F} \leq 0$. If $s_{1,F} > 0$, the forward difference approximation $V'_{1,F}$ is used at the boundary, implying that the value function "never sees the state constraint."

Boundary Conditions

We aim to keep households within the state space, so we enforce that they neither borrow as $k \to k_{min}$ nor save as $k \to k_{max}$.

The constraint $k \geq k_{min}$ is enforced by setting

$$V'_{1,B} = U'(f(k_1) - \delta k_1) \tag{7}$$

The state constraint is applied whenever the forward difference approximation would yield negative savings, $s_{1,F} \leq 0$. If $s_{1,F} > 0$, the forward difference approximation $V'_{1,F}$ is used at the boundary, implying that the value function "never sees the state constraint."

The constraint $k \leq k_{max}$ is enforced by setting

$$V'_{I,F} = U'(f(k_I) - \delta k_I) \tag{8}$$

Implicit Method Algorithm

1. Construct I discrete grid points for k, denoted as k_i where $i = 1, \cdots, I$, and let $V_i = V(k_i)$.
2. For each k_i on the grid, guess for a value of $\mathbf{V^0} = (V_1^0, V_2^0 \cdots .V_I^0)$.

   For iterations $n = 0, 1, 2, \cdots,$

3. Compute $(V_i^n)'$ using (5), (6), (7), and (8).
4. Compute $\mathbf{c^n}$ from $c_i^n = (U')^{-1}[(V_i^n)']$.
5. Find V_i^{n+1} from (1).
6. If $\mathbf{V^{n+1}}$ is close enough to $\mathbf{V^n}$: stop. Otherwise, go to step 3.

Numerical Solution: Finite Difference Method

Exercise: Numerical Solution of the Neoclassical Growth Model

In this exercise, apply the finite difference method with an **upwind scheme** to numerically solve the Hamilton-Jacobi-Bellman (HJB) equation for the Neoclassical Growth Model:

$$\rho V(k) = \max_{c} \{U(c) + V'(k) \cdot (f(k) - \delta k - c)\} \tag{9}$$

In this exercise, we also consider **boundary conditions** for the value function $V(k)$.

Sparse Matrix Routines

- The matrices $\mathbf{D}$, $\mathbf{I}$, and $\mathbf{S^n}$ are highly sparse, meaning they contain a large number of zero elements.
- Sparse matrices can be manipulated efficiently in MATLAB by storing only the non-zero elements, which significantly reduces memory usage and speeds up computations through optimized algorithms that skip operations on zero elements.

Useful Sparse Matrix Functions

- The spy function in MATLAB is useful for checking the sparsity pattern of matrices, displaying non-zero elements as dots. This visual verification ensures matrices are constructed as intended.

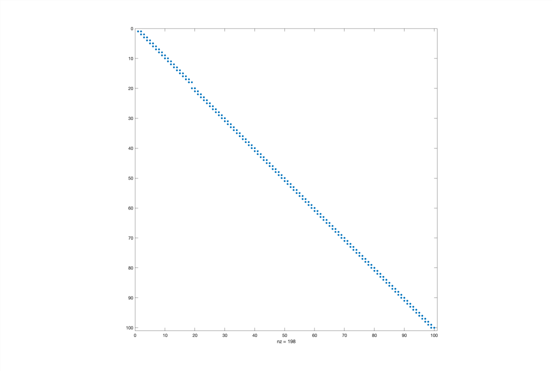

Figure 1: Visualization of Matrix SD

Useful Sparse Matrix Functions

- `sparse`: Creates a sparse matrix by storing only the non-zero elements.

```
sparse Create sparse matrix.
S = sparse(X) converts a sparse or full matrix to sparse form by squeezing
out any zero elements.
```

- `speye`: Generates a sparse identity matrix.

```
speye  Sparse identity matrix.
   speye(M,N) and speye([M N]) form an M-by-N sparse
   matrix with 1's on the main diagonal.  speye(N)
   abbreviates speye(N,N).
```

- `spdiags`: Constructs sparse matrices with specified diagonals.

```
A = spdiags(B,d,m,n) creates an m-by-n sparse matrix from the
    columns of B and places them along the diagonals specified by d.
```

Section 3-2: Huggett Model

Model Overview

- We solve a continuous-time version of Huggett (1993), which represents one of the simplest heterogeneous agent models capturing many key features of more complex models.
- It explores the behavior of agents under uninsurable idiosyncratic income risk with incomplete markets and borrowing constraints.

Model Overview

- The model features an exchange economy with a continuum of agents, where the total mass of agents equals one.
- Each period, households experience uninsurable idiosyncratic income shocks and can be either employed or unemployed.
- Each agent's income follows a Markov process: an employed individual becomes unemployed with probability $\lambda_e = \text{Prob}(z_{t+1} = z_u \mid z_t = z_e)$, and an unemployed individual becomes employed with probability $\lambda_u = \text{Prob}(z_{t+1} = z_e \mid z_t = z_u)$.
- When employed, they receive income $z_e = w(1-\tau)$; when unemployed, they receive unemployment benefits $z_u = \mu w$, with $z_e > z_u$.
- They cannot borrow beyond a certain limit.
- For partial equilibrium analysis, prices are assumed to be constant: $w_t = w$ and $r_t = r \ \forall t$.

Huggett Model in Discrete-Time Recursive Formulation

We start with the following Bellman equations for employed and unemployed households:

$$\begin{aligned}
V_e(a_t) &= \max_{c_t, a_{t+1}} \left\{ U(c_t) + (1-\rho)[(1-\lambda_e)V_e(a_{t+1}) + \lambda_e V_u(a_{t+1})] \right\} \\
\text{s.t.} \quad & c_t + a_{t+1} = z_e + (1+r_t)a_t \\
& a_t \geq \underline{a} \ \ \forall t
\end{aligned} \tag{10}$$

$$\begin{aligned}
V_u(a_t) &= \max_{c_t, a_{t+1}} \left\{ U(c_t) + (1-\rho)[(1-\lambda_u)V_u(a_{t+1}) + \lambda_u V_e(a_{t+1})] \right\} \\
\text{s.t.} \quad & c_t + a_{t+1} = z_u + (1+r_t)a_t \\
& a_t \geq \underline{a} \ \ \forall t
\end{aligned} \tag{11}$$

Discrete to Continuous-Time Transformation

Over a time interval of Δ units, the model can be expressed as:

$$
\begin{aligned}
V_j(a_t) = \max_{c_t, a_{t+1}} & \left\{ \Delta U(c_t) + (1 - \Delta\rho)\big[(1 - \Delta\lambda_j) V_j(a_{t+\Delta}) + \Delta\lambda_j V_u(a_{t+\Delta})\big] \right\} \\
\text{s.t.} \quad & \Delta c_t + a_{t+\Delta} = \Delta z_j + (1 + \Delta r_t) a_t \\
& a_t \geq \underline{a} \;\; \forall t
\end{aligned}
$$

where $j \in \{e, u\}$ represents employment states (employed e and unemployed u).

Discrete to Continuous-Time Transformation

Subtracting $V(a_t)$ from both sides and substituting the constraints into $V(a_{t+\Delta})$, we get:

$$\begin{aligned} 0 = \max_{c_t} \Big\{ & \Delta U(c_t) + \big[V_j(a_t + \Delta(z_j + r_t a_t - c_t)) - V_j(a_t)\big] \\ & - \Delta(\rho + \lambda_j - \Delta\rho\lambda_j) V_j(a_t + \Delta(z_j + r_t a_t - c_t)) \\ & + (1 - \Delta\rho)\Delta\lambda_j V_u(a_t + \Delta(z_j + r_t a_t - c_t)) \Big\} \end{aligned}$$

Dividing both sides by Δ:

$$\begin{aligned} 0 = \max_{c_t} \Big\{ & U(c_t) + \Big[\frac{V_j(a_t + \Delta(z_j + r_t a_t - c_t)) - V_j(a_t)}{\Delta}\Big] \\ & - (\rho + \lambda_j - \Delta\rho\lambda_j) V_j(a_t + \Delta(z_j + r_t a_t - c_t)) \\ & + (1 - \Delta\rho)\lambda_j V_u(a_t + \Delta(z_j + r_t a_t - c_t)) \Big\} \end{aligned}$$

Taking the limit as $\Delta \to 0$, we obtain:

$$0 = \max_{c_t} \Big\{ U(c_t) + V_j'(a_t)(z_j + r_t a_t - c_t) - \rho V_j(a_t) + \lambda_j (V_u(a_t) - V_j(a_t)) \Big\}$$

Hamilton-Jacobi-Bellman (HJB) Equation

Rearranging terms and dropping time notation leads to the HJB equation:

$$\begin{aligned} \rho V_e(a) &= \max_c \left\{ U(c) + V_e'(a)(z_e + ra - c) + \lambda_e(V_u(a) - V_e(a)) \right\} \\ \rho V_u(a) &= \max_c \left\{ U(c) + V_u'(a)(z_u + ra - c) + \lambda_u(V_e(a) - V_u(a)) \right\} \end{aligned} \tag{12}$$

Hamilton-Jacobi-Bellman (HJB) Equation

Rearranging terms and dropping time notation leads to the HJB equation:

$$\begin{aligned} \rho V_e(a) &= \max_c \left\{ U(c) + V_e'(a)(z_e + ra - c) + \lambda_e (V_u(a) - V_e(a)) \right\} \\ \rho V_u(a) &= \max_c \left\{ U(c) + V_u'(a)(z_u + ra - c) + \lambda_u (V_e(a) - V_u(a)) \right\} \end{aligned} \tag{12}$$

The optimal consumption, $c_j = c_j(a)$, is derived from the first-order condition:

$$U'(c_j) = V_j'(a) \tag{13}$$

Hamilton-Jacobi-Bellman (HJB) Equation

Rearranging terms and dropping time notation leads to the HJB equation:

$$\begin{aligned} \rho V_e(a) &= \max_c \left\{ U(c) + V_e'(a)(z_e + ra - c) + \lambda_e (V_u(a) - V_e(a)) \right\} \\ \rho V_u(a) &= \max_c \left\{ U(c) + V_u'(a)(z_u + ra - c) + \lambda_u (V_e(a) - V_u(a)) \right\} \end{aligned} \tag{12}$$

The optimal consumption, $c_j = c_j(a)$, is derived from the first-order condition:

$$U'(c_j) = V_j'(a) \tag{13}$$

Denote $s_j(a) = z_j + ra - c_j$, which represents optimal savings.

Hamilton-Jacobi-Bellman (HJB) Equation

Rearranging terms and dropping time notation leads to the HJB equation:

$$\begin{aligned} \rho V_e(a) &= \max_c \left\{ U(c) + V_e'(a)(z_e + ra - c) + \lambda_e(V_u(a) - V_e(a)) \right\} \\ \rho V_u(a) &= \max_c \left\{ U(c) + V_u'(a)(z_u + ra - c) + \lambda_u(V_e(a) - V_u(a)) \right\} \end{aligned} \tag{12}$$

The optimal consumption, $c_j = c_j(a)$, is derived from the first-order condition:

$$U'(c_j) = V_j'(a) \tag{13}$$

Denote $s_j(a) = z_j + ra - c_j$, which represents optimal savings.

The state constraint $a \geq \underline{a}$ motivates a boundary condition:

$$V_j'(\underline{a}) \geq U'(z_j + r\underline{a}) \tag{14}$$

Generator of Markov Chain

- The generator $\mathcal{A}$ is a functional operator that describes how the stochastic process is expected to evolve.

Generator of Markov Chain

- The generator $\mathcal{A}$ is a functional operator that describes how the stochastic process is expected to evolve.
- It characterizes the expected instantaneous rate of change in the value of a function f as the process evolves over time. Mathematically, it is defined as:

$$\mathcal{A}f = \lim_{\Delta t \to 0} E_t \frac{f(t + \Delta t, X(t + \Delta t)) - f(t, X(t))}{\Delta t}$$

Generator of Markov Chain in This Context

In this model, each agent's endowment follows a Markov chain, where individual i experiences earnings (employment) shocks and transitions between states $z_t \in \{z_e, z_u\}$ — employed (z_e) and unemployed (z_u) — with transition rates λ_e and λ_u, respectively.

Generator of Markov Chain in This Context

In this model, each agent's endowment follows a Markov chain, where individual i experiences earnings (employment) shocks and transitions between states $z_t \in \{z_e, z_u\}$ — employed (z_e) and unemployed (z_u) — with transition rates λ_e and λ_u, respectively.

The associated generator is:

$$\mathcal{A}^z = \begin{pmatrix} -\lambda_e & \lambda_e \\ \lambda_u & -\lambda_u \end{pmatrix}.$$

This matrix represents the transitions between states. Households transition out of state j at rate λ_j, where $j \in \{e, u\}$.

Generator of Markov Chain in This Context

In this model, each agent's endowment follows a Markov chain, where individual i experiences earnings (employment) shocks and transitions between states $z_t \in \{z_e, z_u\}$ — employed (z_e) and unemployed (z_u) — with transition rates λ_e and λ_u, respectively.

The associated generator is:

$$\mathcal{A}^z = \begin{pmatrix} -\lambda_e & \lambda_e \\ \lambda_u & -\lambda_u \end{pmatrix}.$$

This matrix represents the transitions between states. Households transition out of state j at rate λ_j, where $j \in \{e, u\}$.

The rows of the matrix sum to 0, reflecting a conservation of mass in this continuous-time setting. Unlike discrete-time Markov chains, where row sums are 1 (representing probabilities), here, the row sums represent net transition rates.

Finite Difference Approximation

The finite difference approximation to the two HJB equations 12 and 13 is:

$$\begin{aligned}&\rho V_{i,j} = U(c_{i,j}) + V'_{i,j}(z_j + ra_i - c_{i,j}) + \lambda_j(V_{i,-j} - V_{i,j}), \;\; j = e, u \\ &\text{with} \;\; c_{i,j} = (U')^{-1}(V'_{i,j})\end{aligned} \tag{15}$$

where $i = 1, \cdots, I$ and $j \in \{e, u\}$, $V_{i,j} = V_j(a_i)$ with a uniform step size $\Delta a = a_{i+1} - a_i$.

Key Challenges

1. Approximating the **derivative** of the value function, V_i'.
    - Mixed Method
    - **Upwind Scheme**
2. Solving the system, which is highly **non-linear**, requires iterative schemes.
    - Explicit Method
    - **Implicit Method**

Upwind Scheme

Compute optimal savings according to both the forward and backward difference approximations $V'_{i,F}$ and $V'_{i,B}$:

$$\begin{aligned} s_{i,j,F} &= z_j + ra_i - (U')^{-1}(V'_{i,j,F}) \\ s_{i,j,B} &= z_j + ra_i - (U')^{-1}(V'_{i,j,B}) \end{aligned} \tag{16}$$

Then, use the following approximation for $V'_{i,j}$:

$$V'_{i,j} = V'_{i,j,F}\mathbf{1}_{\{s_{i,j,F}>0\}} + V'_{i,j,B}\mathbf{1}_{\{s_{i,j,B}<0\}} + \overline{V}'_i\mathbf{1}_{\{s_{i,j,F}\leq 0\leq s_{i,j,B}\}} \tag{17}$$

where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function, and $\bar{V}'_{i,j} = U'(c_{i,j}) = U'(z_j + ra_i)$.

Implicit Method

V^{n+1} is now *implicitly* defined by:

$$\frac{V_{i,j}^{n+1} - V_{i,j}^{n}}{\Delta} + \rho V_{i,j}^{n+1} = U(c_{i,j}^{n}) + (V_{i,j}^{n+1})'(z_j - \delta a_i - c_{i,j}^{n}) + \lambda_j (V_{i,-j}^{n+1} - V_{i,j}^{n+1}) \tag{18}$$

The step size Δ can be arbitrarily large. (Achdou et al., 2022)

Upwind Scheme with Implicit Method

We use the following finite difference approximation:

$$\frac{V_{i,j}^{n+1} - V_{i,j}^{n}}{\Delta} + \rho V_{i,j}^{n+1} = U(c_{i,j}^{n}) + (V_{i,j,F}^{n+1})'[z_j + ra_i - c_{i,j,F}^{n}]^{+} + (V_{i,j,B}^{n+1})'[z_j + ra_i - c_{i,j,B}^{n}]^{-} + \lambda_j[V_{i,-j}^{n+1} - V_{i,j}^{n+1}] \tag{19}$$

with $c_{i,j} = (U')^{-1}(V'_{i,j})$.

where for any number x, $x^{+} = \max\{x, 0\}$ and $x^{-} = \min\{x, 0\}$

Upwind Scheme with Implicit Method

Equivalently:

$$\frac{V_{i,j}^{n+1} - V_{i,j}^{n}}{\Delta} + \rho V_{i,j}^{n+1} = U(c_{i,j}^{n}) + \frac{V_{i+1,j}^{n+1} - V_{i,j}^{n+1}}{\Delta a}(s_{i,j,F}^{n})^{+} + \frac{V_{i,j}^{n+1} - V_{i-1,j}^{n+1}}{\Delta a}(s_{i,j,B}^{n})^{-} + \lambda_j [V_{i,-j}^{n+1} - V_{i,j}^{n+1}] \tag{20}$$

Upwind Scheme with Implicit Method

Collecting terms with the same subscripts on the right-hand side:

$$\frac{V_{i,j}^{n+1} - V_{i,j}^{n}}{\Delta} + \rho V_{i,j}^{n+1} = u(c_{i,j}^{n}) + V_{i-1,j}^{n+1} x_{i,j} + V_{i,j}^{n+1} y_{i,j} + V_{i+1,j}^{n+1} z_{i,j} + V_{i,-j}^{n+1} \lambda_j \tag{21}$$

where

$$\begin{aligned} x_{i,j} &= -\frac{(s_{i,j,B}^{n})^{-}}{\Delta a} \\ y_{i,j} &= -\frac{(s_{i,j,F}^{n})^{+}}{\Delta a} + \frac{(s_{i,j,B}^{n})^{-}}{\Delta a} - \lambda_j \\ z_{i,j} &= \frac{(s_{i,j,F}^{n})^{+}}{\Delta a} \end{aligned} \tag{22}$$

Upwind Scheme with Implicit Method

Equation (21) with (22) represents a system of $2 \times I$ linear equations, which can be written in matrix notation as:

$$\frac{1}{\Delta}(\mathbf{V^{n+1}} - \mathbf{V^n}) + \rho \mathbf{V^{n+1}} = U(\mathbf{c^n}) + \mathbf{P^n}\mathbf{V^{n+1}}$$

where

$$\mathbf{P^n} = \left(\begin{array}{ccccc|ccccc}
y_{1,1} & z_{1,1} & 0 & \dots & 0 & \lambda_1 & 0 & 0 & \dots & 0 \\
x_{2,1} & y_{2,1} & z_{2,1} & 0 & \dots & 0 & \lambda_1 & 0 & 0 & \dots \\
0 & x_{3,1} & y_{3,1} & z_{3,1} & 0 & 0 & \lambda_1 & 0 & 0 & \dots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & 0 \\
0 & \ddots & \ddots & x_{I,1} & y_{I,1} & 0 & 0 & 0 & 0 & \lambda_1 \\
\hline
\lambda_2 & 0 & 0 & 0 & 0 & y_{1,2} & z_{1,2} & 0 & 0 & 0 \\
0 & \lambda_2 & 0 & 0 & 0 & x_{2,2} & y_{2,2} & z_{2,2} & 0 & 0 \\
0 & 0 & \lambda_2 & 0 & 0 & 0 & x_{3,2} & y_{3,2} & z_{3,2} & 0 \\
0 & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & 0 \\
0 & \dots & \dots & 0 & \lambda_2 & 0 & \dots & 0 & x_{I,2} & y_{I,2}
\end{array}\right)
\begin{array}{l}
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\end{array}}\right\} I \text{ elements} \\
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\end{array}}\right\} I \text{ elements}
\end{array}$$

State Constraints

The borrowing constraint $a_t \geq \underline{a}$ is enforced by setting:

$$V'_{1,j,B} = U'(z_j + r\underline{a}) \tag{23}$$

In practice, the stability of the algorithm improves by imposing a state constraint $a \leq a_{max}$ where a_{max} is the upper bounds of the state space used in computations. This can be achieved by setting:

$$V'_{I,j,F} = U'(z_j + ra_I) \tag{24}$$

Initial Guess for the Value Function

A natural initial guess is the value function of "staying put":

$$V_{i,j}^{0} = \frac{U(z_j + ra_i)}{\rho} \tag{25}$$

Implicit Method: Matrix Representation

1. Define I discrete grid points for a, denoted as a_i for $i = 1, \dots, I$, and form an $I \times 1$ vector $\mathbf{a} = [a_1, a_2, \dots, a_I]'$.

2. Let $V_{i,j} = V_j(a_i)$. For each a_i on the grid, make an initial guess for the value function as two $I \times 1$ vectors $\mathbf{V_e^0} = [V_{1,e}^0, V_{2,e}^0, \dots, V_{I,e}^0]'$ and $\mathbf{V_u^0} = [V_{1,u}^0, V_{2,u}^0, \dots, V_{I,u}^0]'$.

3. Construct the stacked $2I \times 1$ vector $\mathbf{V^0} = [\mathbf{V_e^0}, \mathbf{V_u^0}]'$.

4. Construct the $I \times I$ forward and backward difference matrix operators $\mathbf{D_f}$ and $\mathbf{D_B}$ such that $\mathbf{D_F V^n} \simeq (\mathbf{V^n})'_\mathbf{F}$ and $\mathbf{D_B V^n} \simeq (\mathbf{V^n})'_\mathbf{B}$.

Implicit Method: Matrix Representation

5. Construct a $2I \times 2I$ matrix as follows:

$$\mathbf{A} = \left(\begin{array}{cccc|cccc}
-\lambda_e & 0 & \cdots & 0 & \lambda_e & 0 & \cdots & 0 \\
0 & -\lambda_e & 0 & 0 & 0 & \lambda_e & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \\
0 & 0 & 0 & -\lambda_e & 0 & 0 & 0 & \lambda_e \\
\hline
\lambda_u & 0 & \cdots & 0 & -\lambda_u & 0 & \cdots & 0 \\
0 & \lambda_u & 0 & 0 & 0 & -\lambda_u & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \\
0 & 0 & 0 & \lambda_u & 0 & 0 & 0 & -\lambda_u
\end{array}\right)
\begin{array}{l}
\left.\vphantom{\begin{array}{c}0\\0\\\vdots\\0\end{array}}\right\} I \text{ elements} \\
\left.\vphantom{\begin{array}{c}0\\0\\\vdots\\0\end{array}}\right\} I \text{ elements}
\end{array}$$

Implicit Method: Matrix Representation

For iterations $n = 0, 1, 2, \ldots$

6. Compute the derivative of the value function as an $I \times 1$ vector using both forward and backward difference matrix operators:

$$(\mathbf{V^n_{e,f}})' = \mathbf{D_f V^n_e} \qquad (\mathbf{V^n_{u,f}})' = \mathbf{D_f V^n_u}$$
$$(\mathbf{V^n_{e,b}})' = \mathbf{D_b V^n_e} \qquad (\mathbf{V^n_{u,b}})' = \mathbf{D_b V^n_u}$$

7. Set the first elements of

$$(\mathbf{V^n_{e,b}})' = U'(z_e + r\underline{a}) \qquad (\mathbf{V^n_{u,b}})' = U'(z_u + r\underline{a})$$

and the last elements of

$$(\mathbf{V^n_{e,f}})' = U'(z_e + r\overline{a}) \qquad (\mathbf{V^n_{u,f}})' = U'(z_u + r\overline{a})$$

Implicit Method: Matrix Representation

9. Compute the optimal consumption as an $I \times 1$ vector from:

$$\mathbf{c^n_{e,f}} = (U')^{-1}[(\mathbf{V^n_{e,f}})'] \qquad \mathbf{c^n_{u,f}} = (U')^{-1}[(\mathbf{V^n_{u,f}})']$$
$$\mathbf{c^n_{e,b}} = (U')^{-1}[(\mathbf{V^n_{e,b}})'] \qquad \mathbf{c^n_{u,b}} = (U')^{-1}[(\mathbf{V^n_{u,b}})']$$

10. Calculate the optimal savings as an $I \times 1$ vector from:

$$\mathbf{s^n_{e,f}} = z_e + r\mathbf{a} - \mathbf{c^n_{e,f}} \qquad \mathbf{s^n_{u,f}} = z_u + r\mathbf{a} - \mathbf{c^n_{u,f}}$$
$$\mathbf{s^n_{e,b}} = z_e + r\mathbf{a} - \mathbf{c^n_{e,b}} \qquad \mathbf{s^n_{u,b}} = z_u + r\mathbf{a} - \mathbf{c^n_{u,b}}$$

11. Create indicator vectors:

$$\mathbf{I^n_{e,f}} = [I^n_{1,e,f}, I^n_{2,e,f}, \cdots, I^n_{I,e,f}]' \qquad \mathbf{I^n_{u,f}} = [I^n_{1,u,f}, I^n_{2,u,f}, \cdots, I^n_{I,u,f}]'$$
$$\mathbf{I^n_{e,b}} = [I^n_{1,e,b}, I^n_{2,e,b}, \cdots, I^n_{I,e,b}]' \qquad \mathbf{I^n_{u,b}} = [I^n_{1,u,b}, I^n_{2,u,b}, \cdots, I^n_{I,u,b}]'$$

where $I^n_{i,j,f} = 1$ if $s^n_{i,j,f} > 0$ and $I^n_{i,j,b} = 1$ if $s^n_{i,j,b} < 0$ for $i = 1, \cdots, I$ and $j = e, u$.

Implicit Method: Matrix Representation

12. Compute optimal consumption as follows:

$$\mathbf{c_e^n} = \mathbf{I_{e,f}^n} \cdot \mathbf{c_{e,f}^n} + \mathbf{I_{e,b}^n} \cdot \mathbf{c_{e,b}^n}$$
$$\mathbf{c_u^n} = \mathbf{I_{u,f}^n} \cdot \mathbf{c_{u,f}^n} + \mathbf{I_{u,b}^n} \cdot \mathbf{c_{u,b}^n}$$

13. Compute optimal savings as follows:

$$\mathbf{s_e^n} = \mathbf{I_{e,f}^n} \cdot \mathbf{s_{e,f}^n} + \mathbf{I_{e,b}^n} \cdot \mathbf{s_{e,b}^n}$$
$$\mathbf{s_u^n} = \mathbf{I_{u,f}^n} \cdot \mathbf{s_{u,f}^n} + \mathbf{I_{u,b}^n} \cdot \mathbf{s_{u,b}^n}$$

14. Construct two $I \times I$ diagonal matrices as follows:

$$\mathbf{S_e^n D_e^n} = diag(\mathbf{I_{e,f}^n} \cdot \mathbf{s_{e,f}^n})\mathbf{D_f} + diag(\mathbf{I_{e,b}^n} \cdot \mathbf{s_{e,b}^n})\mathbf{D_b}$$
$$\mathbf{S_u^n D_u^n} = diag(\mathbf{I_{u,f}^n} \cdot \mathbf{s_{u,f}^n})\mathbf{D_f} + diag(\mathbf{I_{u,b}^n} \cdot \mathbf{s_{u,b}^n})\mathbf{D_b}$$

Implicit Method: Matrix Representation

15. Construct a $2I \times 2I$ matrix $\mathbf{S^nD^n}$ as follows:

$$\mathbf{S^nD^n} = \begin{pmatrix} \mathbf{S_e^nD_e^n} & 0 \\ 0 & \mathbf{S_u^nD_u^n} \end{pmatrix} \tag{26}$$

16. Construct the matrix $\mathbf{P^n}$ as $\mathbf{P^n} = \mathbf{S^nD^n} + \mathbf{A}$

17. Find $\mathbf{V^{n+1}}$ from:

$$\frac{1}{\Delta}(\mathbf{V^{n+1}} - \mathbf{V^n}) + \rho\mathbf{V^{n+1}} = U(\mathbf{c^n}) + \mathbf{P^nV^{n+1}} \tag{27}$$

18. If $\mathbf{V^{n+1}}$ is close enough to $\mathbf{V^n}$: stop. Otherwise, go to step 6.

Implicit Method: Matrix Representation

Alternatively, solve the linear system:

$$\mathbf{V^{n+1}} = \left((\rho + \frac{1}{\Delta})\mathbf{I} - \mathbf{P^n}\right)^{-1} \left[U(\mathbf{c^n}) + \frac{1}{\Delta}\mathbf{V^n}\right] \tag{28}$$

References

Achdou, Y., J. Han, J.-M. Lasry, P.-L. Lions, and B. Moll (2022). Income and wealth distribution in macroeconomics: A continuous-time approach. *The review of economic studies 89*(1), 45–86.

Huggett, M. (1993). The risk-free rate in heterogeneous-agent incomplete-insurance economies. *Journal of economic Dynamics and Control 17*(5-6), 953–969.

LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-dependent Problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics.