Econ 202A Macroeconomics: Section 3

Kiyea Jin

November 5, 7, 2025

Section 3

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

Finite Difference Approximation

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

$$\rho V_i = U(c_i) + V_i' \cdot (k_i^\alpha - \delta k_i - c_i)$$
$$\text{with } c_i = (U')^{-1}(V_i') \tag{1}$$

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

Implicit Method: Matrix Representation

1. Define I discrete grid points for k, denoted as k_i for $i = 1, \ldots, I$, and form an $I \times 1$ vector $\mathbf{k} = [k_1, k_2, \ldots, k_I]'$.

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 $\mathbf{V^0} = [V_1^0, V_2^0, \ldots, V_I^0]'$.

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

3. Compute the derivative of the value function as an $I \times 1$ vector $(\mathbf{V^n})'$ using an $I \times I$ finite-difference 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:

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

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:

$$\mathbf{V^{n+1}} = \left((\rho + \frac{1}{\Delta})\mathbf{I} - \mathbf{S^n D} \right)^{-1} \left[U(\mathbf{c^n}) + \frac{1}{\Delta}\mathbf{V^n} \right] \tag{2}$$

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:

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

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."

- Select the *differencing method* based on the *direction of the drift* of state variable:
  - — Use **forward difference** whenever the drift of state variable is **positive**.
  - — Use **backward difference** whenever the drift of state variable is **negative**.

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

$$s_{i,F} = k_i^\alpha - \delta k_i - (U')^{-1}(V'_{i,F})$$
$$s_{i,B} = k_i^\alpha - \delta k_i - (U')^{-1}(V'_{i,B})$$

$$(4)$$

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

$$s_{i,F} = k_i^\alpha - \delta k_i - (U')^{-1}(V'_{i,F})$$
$$s_{i,B} = k_i^\alpha - \delta k_i - (U')^{-1}(V'_{i,B})$$

(4)

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

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

(5)

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

- 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.

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

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

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

- Intuitively, if saving is positive, what matters is how the value function changes when wealth increases by a small amount; and vice versa when saving is negative. (Achdou et al., 2022)

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).

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.

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

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

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

$$V'_{1,B} = U'(f(k_1) - \delta k_1) \tag{6}$$

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

$$V'_{1,B} = U'(f(k_1) - \delta k_1) \tag{6}$$

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

$$V'_{I,F} = U'(f(k_I) - \delta k_I) \tag{7}$$

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 (4), (5), (6), and (7).

4. Compute $\mathbf{c^n}$ from $c_i^n = (U')^{-1}[(V_i^n)']$.

5. Find V_i^{n+1} from (2).

6. If $\mathbf{V^{n+1}}$ is close enough to $\mathbf{V^n}$: stop. Otherwise, go to step 3.

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:

$$\rho V(k) = \max_c \left\{ U(c) + V'(k) \cdot (f(k) - \delta k - c) \right\} \tag{8}$$

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

Implicit Method: Matrix Representation

1. Define I discrete grid points for k, denoted as k_i for $i = 1, \ldots, I$, and form an $I \times 1$ vector $\mathbf{k} = [k_1, k_2, \ldots, k_I]'$.

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 $\mathbf{V}^0 = [V_1^0, V_2^0, \ldots, V_I^0]'$.

3. Construct the $I \times I$ forward and backward difference matrix operators $\mathbf{D_F}$ and $\mathbf{D_B}$ such that $\mathbf{D_F V}^n \simeq (\mathbf{V}^n)'_F$ and $\mathbf{D_B V}^n \simeq (\mathbf{V}^n)'_B$.

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

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

$$(\mathbf{V}_F^n)' = \mathbf{D_F}\mathbf{V}^n \qquad\qquad (\mathbf{V}_B^n)' = \mathbf{D_B}\mathbf{V}^n$$

5. Set the first element of $(\mathbf{V}_B^n)'$ as:

$$(V_{1,B}^n)' = U'(f(k_1) - \delta k_1)$$

and the last element of $(\mathbf{V}_F^n)'$ as:

$$(V_{I,F}^n)' = U'(f(k_I) - \delta k_I)$$

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

$$\mathbf{c}_F^n = (U')^{-1}[(\mathbf{V}_F^n)'] \qquad \mathbf{c}_B^n = (U')^{-1}[(\mathbf{V}_B^n)']$$

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

$$\mathbf{s}_F^n = f(\mathbf{k}) - \delta\mathbf{k} - \mathbf{c}_F^n \qquad \mathbf{s}_B^n = f(\mathbf{k}) - \delta\mathbf{k} - \mathbf{c}_B^n$$

8. Create indicator vectors:

$$\mathbf{I}_F^n = [I_{1,F}^n, I_{2,F}^n, \ldots, I_{I,F}^n]' \qquad \mathbf{I}_B^n = [I_{1,B}^n, I_{2,B}^n, \ldots, I_{I,B}^n]'$$

where $I_{i,F}^n = 1$ if $s_{i,F}^n > 0$ and $I_{i,B}^n = 1$ if $s_{i,B}^n < 0$ for $i = 1, \ldots, I$.

Implicit Method: Matrix Representation

9. Compute optimal consumption as:

$$\mathbf{c}^n = \mathbf{I}_F^n \cdot \mathbf{c}_F^n + \mathbf{I}_B^n \cdot \mathbf{c}_B^n + (1 - \mathbf{I}_F^n - \mathbf{I}_B^n) \cdot \mathbf{c}_0^n$$

where the dot indicates element-wise multiplication.

10. Compute optimal savings as:

$$\mathbf{s}^n = \mathbf{I}_F^n \cdot \mathbf{s}_F^n + \mathbf{I}_B^n \cdot \mathbf{s}_B^n$$

where the dot indicates element-wise multiplication.

11. Construct an $I \times I$ matrix:

$$\mathbf{S}^n\mathbf{D} = \text{diag}(\mathbf{I}_F^n \cdot \mathbf{s}_F^n)\mathbf{D_F} + \text{diag}(\mathbf{I}_B^n \cdot \mathbf{s}_B^n)\mathbf{D_B}$$

Implicit Method: Matrix Representation

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

$$\mathbf{V}^{n+1} = \left[(\rho + \frac{1}{\Delta})\mathbf{I} - \mathbf{S}^n\mathbf{D}\right]^{-1}\left[U(\mathbf{c}^n) + \frac{1}{\Delta}\mathbf{V}^n\right] \tag{9}$$

13. If $\mathbf{V}^{n+1}$ is close enough to $\mathbf{V}^n$, stop. Otherwise, go to step 4.

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.
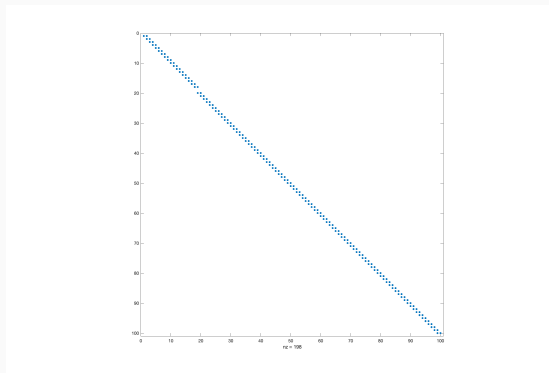

Figure 1: Visualization of Matrix SD

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 is one of the simplest heterogeneous agent models that captures key features of richer models.

- The model studies agents in incomplete markets facing (i) uninsurable idiosyncratic income risk and (ii) borrowing constraints.

Environment

- 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.

- The only asset available is a risk-free bond, whose interest rate is determined in equilibrium. For partial-equilibrium analysis, prices are assumed to be constant and exogenously given.

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

$$V_e(a_t) = \max_{c_t, a_{t+1}} \left\{ U(c_t) + (1 - \rho)[(1 - \lambda_e)V_e(a_{t+1}) + \lambda_e V_u(a_{t+1})] \right\}$$

$$\text{s.t.} \quad c_t + a_{t+1} = z_e + (1 + r_t)a_t \tag{10}$$

$$a_t \geq \underline{a} \ \forall t$$

$$V_u(a_t) = \max_{c_t, a_{t+1}} \left\{ U(c_t) + (1 - \rho)[(1 - \lambda_u)V_u(a_{t+1}) + \lambda_u V_e(a_{t+1})] \right\}$$

$$\text{s.t.} \quad c_t + a_{t+1} = z_u + (1 + r_t)a_t \tag{11}$$

$$a_t \geq \underline{a} \ \forall t$$

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

$$V_j(a_t) = \max_{c_t, a_{t+1}} \left\{ \Delta U(c_t) + (1 - \Delta\rho)\left[(1 - \Delta\lambda_j)V_j(a_{t+\Delta}) + \Delta\lambda_j V_u(a_{t+\Delta})\right] \right\}$$

$$\text{s.t.} \quad \Delta c_t + a_{t+\Delta} = \Delta z_j + (1 + \Delta r_t)a_t$$

$$a_t \geq \underline{a} \ \forall t$$

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:

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

Dividing both sides by Δ:

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

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:

$$\rho V_e(a) = \max_c \left\{ U(c) + V_e'(a)(z_e + ra - c) + \lambda_e(V_u(a) - V_e(a)) \right\}$$

$$\rho V_u(a) = \max_c \left\{ U(c) + V_u'(a)(z_u + ra - c) + \lambda_u(V_e(a) - V_u(a)) \right\}$$

(12)

Hamilton-Jacobi-Bellman (HJB) Equation

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

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

(12)

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

$$U'(c_j) = V_j'(a)$$

(13)

Hamilton-Jacobi-Bellman (HJB) Equation

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

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

(12)

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

$$U'(c_j) = V_j'(a)$$

(13)

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

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

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

$$(12)$$

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 a Markov Process

- The *(infinitesimal) generator* $\mathcal{A}$ maps a function $f(t, x)$ to the **instantaneous expected rate of change** of $f(t, X_t)$.

$$(\mathcal{A}f)(t, x) \;=\; \lim_{\Delta t \downarrow 0} \frac{\mathbb{E}[f(t + \Delta t, X(t + \Delta t)) - f(t, x) \,|\, X(t) = x]}{\Delta t}.$$

- Intuition: look an instant ahead, average the increment in f over the randomness of X, then scale by the step length.

Consider the Itô diffusion

$$dX_t = \mu(X_t, t)\, dt + \sigma(X_t, t)\, dW_t,$$

with $f \in C^{1,2}$.

Using Itô's formula, the generator is

$$(\mathcal{A}f)(t, x) \;=\; f_t(t, x) + \mu(x, t)\, f_x(t, x) + \tfrac{1}{2}\sigma^2(x, t)\, f_{xx}(t, x).$$

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:

$$\rho V_{i,j} = U(c_{i,j}) + V'_{i,j} \cdot (z_j + ra_i - c_{i,j}) + \lambda_j(V_{i,-j} - V_{i,j}), \ \ j = e, u$$
$$\text{with } c_{i,j} = (U')^{-1}(V'_{i,j})$$

(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**

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 $V'_{i,j}$:

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

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

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

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

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

We use the following finite difference approximation:

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

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

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

Equivalently:

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

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

$$\mathbf{P^n} = \begin{pmatrix} y_{1,1} & z_{1,1} & 0 & \ldots & 0 & \lambda_1 & 0 & 0 & \ldots & 0 \\ x_{2,1} & y_{2,1} & z_{2,1} & 0 & \ldots & 0 & \lambda_1 & 0 & 0 & \ldots \\ 0 & x_{3,1} & y_{3,1} & z_{3,1} & 0 & 0 & 0 & \lambda_1 & 0 & \ldots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & 0 \\ 0 & \ddots & \ddots & x_{I,1} & y_{I,1} & 0 & 0 & 0 & 0 & \lambda_1 \\ \lambda_2 & 0 & 0 & 0 & 0 & y_{1,2} & z_{1,2} & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & x_{2,2} & y_{2,2} & z_{2,2} & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 & 0 & 0 & x_{3,2} & y_{3,2} & z_{3,2} & 0 \\ 0 & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & 0 \\ 0 & \ldots & \ldots & 0 & \lambda_2 & 0 & \ldots & 0 & x_{I,2} & y_{I,2} \end{pmatrix} \begin{array}{l} \left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\end{matrix}}\right\} I \text{ elements} \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\end{matrix}}\right\} I \text{ elements} \end{array}$$

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:

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

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

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

Implicit Method: Matrix Representation

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

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 $\mathbf{V_e^0} = [V_{1,e}^0, V_{2,e}^0, \ldots, V_{I,e}^0]'$ and $\mathbf{V_u^0} = [V_{1,u}^0, V_{2,u}^0, \ldots, V_{I,u}^0]'$.

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 $\mathbf{D_B V^n} \simeq (\mathbf{V^n})'_{\mathbf{B}}$.

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

$$\mathbf{A} = \left(\begin{array}{cccc|cccc} -\lambda_e & 0 & \cdots & 0 & \lambda_e & 0 & \cdots & 0 \\ 0 & -\lambda_e & 0 & 0 & 0 & \lambda_e & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \\ 0 & 0 & 0 & -\lambda_e & 0 & 0 & 0 & \lambda_e \\ \hline \lambda_u & 0 & \cdots & 0 & -\lambda_u & 0 & \cdots & 0 \\ 0 & \lambda_u & 0 & 0 & 0 & -\lambda_u & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots \\ 0 & 0 & 0 & \lambda_u & 0 & 0 & 0 & -\lambda_u \end{array}\right) \begin{array}{l} \left.\rule{0pt}{3.5em}\right\} I \text{ elements} \\ \left.\rule{0pt}{3.5em}\right\} I \text{ elements} \end{array}$$

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:

$$(\mathbf{V_{e,F}^n})' = \mathbf{D_F V_e^n} \qquad\qquad (\mathbf{V_{u,F}^n})' = \mathbf{D_F V_u^n}$$

$$(\mathbf{V_{e,B}^n})' = \mathbf{D_B V_e^n} \qquad\qquad (\mathbf{V_{u,B}^n})' = \mathbf{D_B V_u^n}$$

7. Set the first elements of $\mathbf{V_{j,B}^n}$ as:

$$(V_{1,e,B}^n)' = U'(z_e + r\underline{a}) \qquad\qquad (V_{1,u,B}^n)' = U'(z_u + r\underline{a})$$

and the last elements of $\mathbf{V_{j,F}^n}$ as:

$$(V_{I,e,F}^n)' = U'(z_e + r\overline{a}) \qquad\qquad (V_{I,u,F}^n)' = U'(z_u + r\overline{a})$$

Implicit Method: Matrix Representation

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

$$\mathbf{c}_{\mathbf{e,F}}^{\mathbf{n}} = (U')^{-1}[(\mathbf{V}_{\mathbf{e,F}}^{\mathbf{n}})'] \qquad\qquad \mathbf{c}_{\mathbf{u,F}}^{\mathbf{n}} = (U')^{-1}[(\mathbf{V}_{\mathbf{u,F}}^{\mathbf{n}})']$$
$$\mathbf{c}_{\mathbf{e,B}}^{\mathbf{n}} = (U')^{-1}[(\mathbf{V}_{\mathbf{e,B}}^{\mathbf{n}})'] \qquad\qquad \mathbf{c}_{\mathbf{u,B}}^{\mathbf{n}} = (U')^{-1}[(\mathbf{V}_{\mathbf{u,B}}^{\mathbf{n}})']$$

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

$$\mathbf{s}_{\mathbf{e,F}}^{\mathbf{n}} = z_e + r\mathbf{a} - \mathbf{c}_{\mathbf{e,F}}^{\mathbf{n}} \qquad\qquad \mathbf{s}_{\mathbf{u,F}}^{\mathbf{n}} = z_u + r\mathbf{a} - \mathbf{c}_{\mathbf{u,F}}^{\mathbf{n}}$$
$$\mathbf{s}_{\mathbf{e,B}}^{\mathbf{n}} = z_e + r\mathbf{a} - \mathbf{c}_{\mathbf{e,B}}^{\mathbf{n}} \qquad\qquad \mathbf{s}_{\mathbf{u,B}}^{\mathbf{n}} = z_u + r\mathbf{a} - \mathbf{c}_{\mathbf{u,B}}^{\mathbf{n}}$$

11. Create indicator vectors:

$$\mathbf{I}_{\mathbf{e,F}}^{\mathbf{n}} = [I_{1,e,F}^n, I_{2,e,F}^n, \cdots, I_{I,e,F}^n]' \qquad\qquad \mathbf{I}_{\mathbf{u,F}}^{\mathbf{n}} = [I_{1,u,F}^n, I_{2,u,F}^n, \cdots, I_{I,u,F}^n]'$$
$$\mathbf{I}_{\mathbf{e,B}}^{\mathbf{n}} = [I_{1,e,B}^n, I_{2,e,B}^n, \cdots, I_{I,e,B}^n]' \qquad\qquad \mathbf{I}_{\mathbf{u,B}}^{\mathbf{n}} = [I_{1,u,B}^n, I_{2,u,B}^n, \cdots, I_{I,u,B}^n]'$$

where $I_{i,j,F}^n = 1$ if $s_{i,j,F}^n > 0$ and $I_{i,j,B}^n = 1$ if $s_{i,j,B}^n < 0$ for $i = 1, \cdots, I$ and $j = e, u$.

12. Compute optimal consumption as follows:

$$\mathbf{c_e^n} = \mathbf{I_{e,F}^n} \cdot \mathbf{c_{e,F}^n} + \mathbf{I_{e,B}^n} \cdot \mathbf{c_{e,B}^n} + (1 - \mathbf{I_{e,F}^n} - \mathbf{I_{e,B}^n}) \cdot \mathbf{c_{e,0}^n}$$
$$\mathbf{c_u^n} = \mathbf{I_{u,F}^n} \cdot \mathbf{c_{u,F}^n} + \mathbf{I_{u,B}^n} \cdot \mathbf{c_{u,B}^n} + (1 - \mathbf{I_{u,F}^n} - \mathbf{I_{u,B}^n}) \cdot \mathbf{c_{u,0}^n}$$

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:

$$\mathbf{S_e^n D_e^n} = \mathrm{diag}(\mathbf{I_{e,F}^n} \cdot \mathbf{s_{e,F}^n})\,\mathbf{D_F} + \mathrm{diag}(\mathbf{I_{e,B}^n} \cdot \mathbf{s_{e,B}^n})\,\mathbf{D_B}$$
$$\mathbf{S_u^n D_u^n} = \mathrm{diag}(\mathbf{I_{u,F}^n} \cdot \mathbf{s_{u,F}^n})\,\mathbf{D_F} + \mathrm{diag}(\mathbf{I_{u,B}^n} \cdot \mathbf{s_{u,B}^n})\,\mathbf{D_B}$$

Implicit Method: Matrix Representation

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

$$\mathbf{S^n D^n} = \begin{pmatrix} \mathbf{S_e^n D_e^n} & 0 \\ 0 & \mathbf{S_u^n D_u^n} \end{pmatrix} \tag{26}$$

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

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

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

where $\mathbf{c^n} = [\mathbf{c_e^n}, \mathbf{c_u^n}]'$ is the stacked $2I \times 1$ vector.

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.