Econ 202A Macroeconomics: Section 6

Kiyea Jin

December 3, 5, 2025

Section 6

Overview

1. Transition Dynamics
    - Permanent shock
    - "MIT" shock (homework)
2. Andreas's Repository
    - Adaptive sparse grid

Section 6-1: Transition Dynamics

Transition Dynamics

- We have solved for the long-run stationary equilibrium of the Huggett economy under fixed parameters.
- Now, consider a permanent increase in unemployment risk, represented by λ_e.
- Our goal is to analyze the transition dynamics resulting from this parameter change.

Dynamics of Wealth Distribution

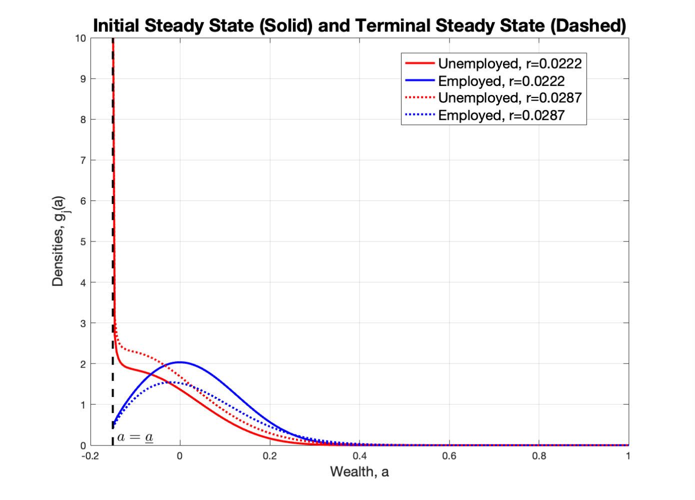

Figure 1: Initial and Terminal Wealth Distribution (T=0, ∞)

Dynamics of Wealth Distribution

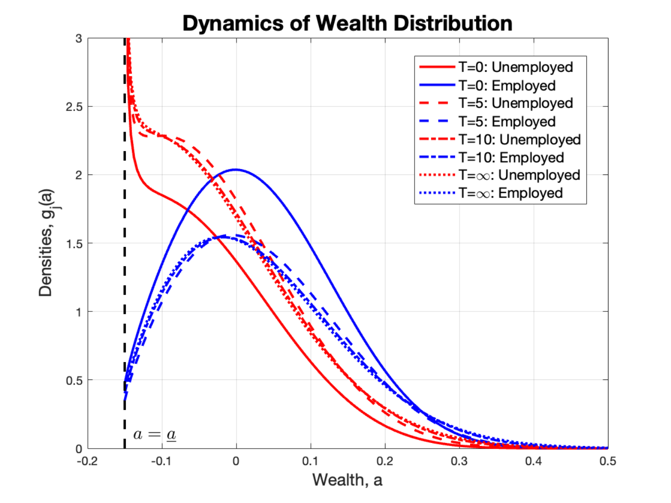

Figure 2: Dynamics of Wealth Distribution (T=0, 5, 10, ∞)

Go to Savings

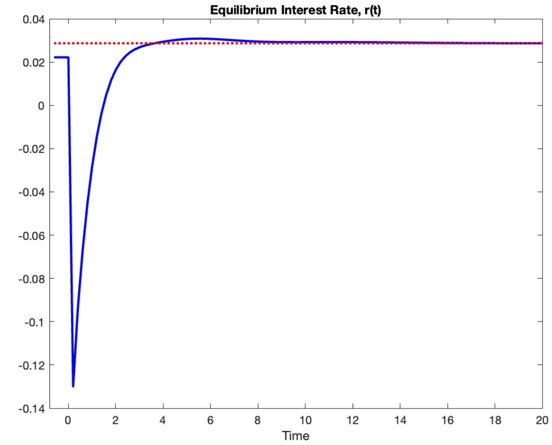

Figure 3: Time Path of Equilibrium Interest Rate

System of Equations

The system to be solved is:

$$\rho V_j(a,t) = \max_c \; u(c) + \partial_a V_j(a,t)\left[z_j + r(t)a - c\right] + \lambda_j \left[V_{-j}(a,t) - V_j(a,t)\right] + \partial_t V_j(a,t) \tag{1}$$

$$c_j(a,t) = \left(u'\right)^{-1}\left(\partial_a V_j(a,t)\right) \tag{2}$$

$$s_j(a,t) = z_j + r(t)a - c_j(a,t) \tag{3}$$

System of Equations

The system to be solved is:

$$\rho V_j(a,t) = \max_c \; u(c) + \partial_a V_j(a,t)\left[z_j + r(t)a - c\right] + \lambda_j\left[V_{-j}(a,t) - V_j(a,t)\right] + \partial_t V_j(a,t) \tag{1}$$

$$c_j(a,t) = \left(u'\right)^{-1}\left(\partial_a V_j(a,t)\right) \tag{2}$$

$$s_j(a,t) = z_j + r(t)a - c_j(a,t) \tag{3}$$

$$\partial_t g_j(a,t) = -\partial_a\left[s_j(a,t)g_j(a,t)\right] - \lambda_j g_j(a,t) + \lambda_{-j} g_{-j}(a,t) \tag{4}$$

System of Equations

The system to be solved is:

$$\rho V_j(a,t) = \max_c \; u(c) + \partial_a V_j(a,t)\left[z_j + r(t)a - c\right] + \lambda_j\left[V_{-j}(a,t) - V_j(a,t)\right] + \partial_t V_j(a,t) \quad (1)$$

$$c_j(a,t) = \left(u'\right)^{-1}\left(\partial_a V_j(a,t)\right) \quad (2)$$

$$s_j(a,t) = z_j + r(t)a - c_j(a,t) \quad (3)$$

$$\partial_t g_j(a,t) = -\partial_a\left[s_j(a,t)g_j(a,t)\right] - \lambda_j g_j(a,t) + \lambda_{-j} g_{-j}(a,t) \quad (4)$$

$$0 = dS(r(t)) \;\; \text{where} \;\; S(r(t)) = \int_{\underline{a}}^{\infty} a g_e(a,t)\,da + \int_{\underline{a}}^{\infty} a g_u(a,t)\,da \quad (5)$$

Algorithm

1. Solve for the long-run stationary equilibrium for both the initial and terminal states:
   — Set initial condition $g_j^n(a, t = 0)$ for all iterations n as $g_j(a)$ from initial equilibrium.
   — Set terminal condition $V_j^n(a, t = T)$ for all iterations n as $V_j(a)$ from terminal equilibrium.
2. Make an initial guess for the function $r^0(t)$. A good initial value is $r^0(t) = r_T$ for all t, based on the terminal equilibrium.

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

3. Given $r^n(t)$, solve the HJB equation backward in time with the terminal condition $V_j^n(a, T) = V_j(a)$. This yields the time path of $V_j^n(a, t)$ and the implied saving policy function $s_j^n(a, t)$. Ensure that the transition matrix $\mathbf{P}^n$ is computed and stored for each iteration.
4. Using $s_j^n(a, t)$, solve the Kolmogorov-Forward (KF) equation forward in time with the initial condition $g_j^n(a, 0)$ to compute the time path of $g_j^n(a, t)$.

Algorithm

5. Calculate the asset supply for all t:

$$S^n(t) = \int_{\underline{a}}^{\infty} a g_e^n(a,t)\, da + \int_{\underline{a}}^{\infty} a g_u^n(a,t)\, da$$

6. Update the guess for $r(t)$ using:

$$r^{n+1}(t) = r^n(t) - \xi \frac{dS^n(t)}{dt}$$

   where $\xi > 0$ is a step size.

7. Stop the iteration when $r^{n+1}(t)$ is sufficiently close to $r^n(t)$ for all t.

(Step 3) Solving the Time-Dependent HJB Equation

Approximate the value function at I discrete points in the wealth dimension and I_t discrete points in the time dimension, and use the shorthand notation $V_{i,j}^{i_t} = V_j(a_i, t_{i_t})$ where $i = 1, \cdots, I$ and $i_t = 1, \cdots, I_t$ with a uniform time step size $\Delta t = t(i_t + 1) - t(i_t)$. The discrete approximation to the time-dependent HJB equation (1) is:

$$\rho V_{i,j}^{i_t} = U(c_{i,j}^{i_t+1}) + (V_{i,j}^{i_t})' \cdot \left[z_j + r^{i_t+1} a_i - c_{i,j}^{i_t+1}\right] + \lambda_j \left[V_{i,-j}^{i_t} - V_{i,j}^{i_t}\right] + \frac{V_{i,j}^{i_t+1} - V_{i,j}^{i_t}}{\Delta t} \tag{7}$$

with terminal condition $V_{i,j}^{I_t} = V_j^{\text{s.s}}(a_i | p_{\text{terminal}})$.

(Step 3) Solving the Time-Dependent HJB Equation

Given $\mathbf{V^{i_t+1}}$, this system can be written in matrix notation as:

$$\rho \mathbf{V^{i_t}} = U(\mathbf{c^{i_t+1}}) + \mathbf{P^{i_t+1}} \mathbf{V^{i_t}} + \frac{1}{\Delta t}(\mathbf{V^{i_t+1}} - \mathbf{V^{i_t}}) \tag{8}$$

where $\mathbf{P^{i_t+1}}$ still has the interpretation of the transition matrix of the discretized stochastic process for (a_t, z_t).

(Step 3) Solving the Time-Dependent HJB Equation

Given $\mathbf{V^{i_t+1}}$, this system can be written in matrix notation as:

$$\rho \mathbf{V^{i_t}} = U(\mathbf{c^{i_t+1}}) + \mathbf{P^{i_t+1}} \mathbf{V^{i_t}} + \frac{1}{\Delta t}(\mathbf{V^{i_t+1}} - \mathbf{V^{i_t}}) \tag{8}$$

where $\mathbf{P^{i_t+1}}$ still has the interpretation of the transition matrix of the discretized stochastic process for (a_t, z_t).

Now each i_t has the *interpretation of a time step instead of an iteration* on the stationary value function. The reason for this similarity in the algorithm is that intuitively a stationary value function can be found by solving a time-dependent problem and going far enough back in time, i.e., as $t \to -\infty$.

(Step 3) Solving the Time-Dependent HJB Equation

Equivalently, solve the linear system:

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

(Step 4) Solving the Time-Dependent Kolmogorov Forward Equation

We approximate the density at I discrete points in the wealth dimension and I_t discrete points in the time dimension, and use the shorthand notation $g_{i,j}^{i_t} = g_j(a_i, t_{i_t})$. Given an initial condition $g_{i,j}^0 = g_j^{s.s}(a_i | p_{\text{initial}})$, the Kolmogorov Forward equation (4) is then easily solved.

One here has the option of using either an explicit method:

$$\frac{\mathbf{g}^{\mathbf{i_t}+1} - \mathbf{g}^{\mathbf{i_t}}}{\Delta t} = (\mathbf{P}^{\mathbf{i_t}})^{\mathsf{T}} \mathbf{g}^{\mathbf{i_t}} \quad \Longrightarrow \quad \mathbf{g}^{\mathbf{i_t}+1} = \Delta t (\mathbf{P}^{\mathbf{i_t}})^{\mathsf{T}} \mathbf{g}^{\mathbf{i_t}} + \mathbf{g}^{\mathbf{i_t}}$$

or an implicit method:

$$\frac{\mathbf{g}^{\mathbf{i_t}+1} - \mathbf{g}^{\mathbf{i_t}}}{\Delta t} = (\mathbf{P}^{\mathbf{i_t}})^{\mathsf{T}} \mathbf{g}^{\mathbf{i_t}+1} \quad \Longrightarrow \quad \mathbf{g}^{\mathbf{i_t}+1} = \left(\mathbf{I} - \Delta t (\mathbf{P}^{\mathbf{i_t}})^{\mathsf{T}}\right)^{-1} \mathbf{g}^{\mathbf{i_t}} \tag{10}$$

Both schemes preserve mass, but the implicit scheme is also guaranteed to preserve the positivity of g for arbitrary time steps Δt.

Section 6-2:
Andreas's Repository

Andreas's Repository

- Repository link: https://github.com/schaab-lab/SparseEcon
- Explore this repository to extend the code you've written so far!
- Clone the repository and add the local path to your MATLAB code.

Adaptive Sparse Grids

- This repository provides a toolbox for solving dynamic programming problems in continuous time using **adaptive sparse grids**. The method applies to a wide range of dynamic programming applications across various fields in economics (Schaab and Zhang, 2022).
- If you're interested, read Schaab and Zhang (2022).
- More resources: https://github.com/schaab-teaching/NumericalMethods

Adaptive Sparse Grids

- Uniform grids:

  — Points are placed equidistantly across the domain.

  — Suffer from the "curse of dimensionality," where the number of grid points grows exponentially with added dimensions.

Adaptive Sparse Grids

- Uniform grids:

  — Points are placed equidistantly across the domain.

  — Suffer from the "curse of dimensionality," where the number of grid points grows exponentially with added dimensions.

- Adaptive sparse grids:

  — Strategically remove points that contribute minimally to function approximation.

  — Use information like residual approximation error to dynamically refine the grid based on problem-specific needs.

Adaptive Sparse Grids

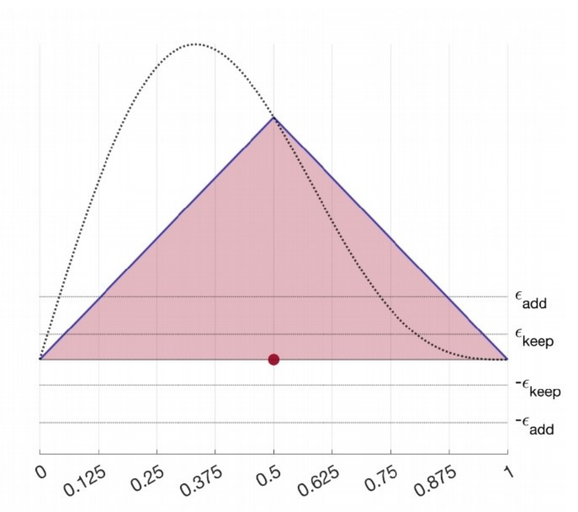

Figure 4: Adaptive Sparse Grids (Schaab and Zhang, 2022)

Adaptive Sparse Grids

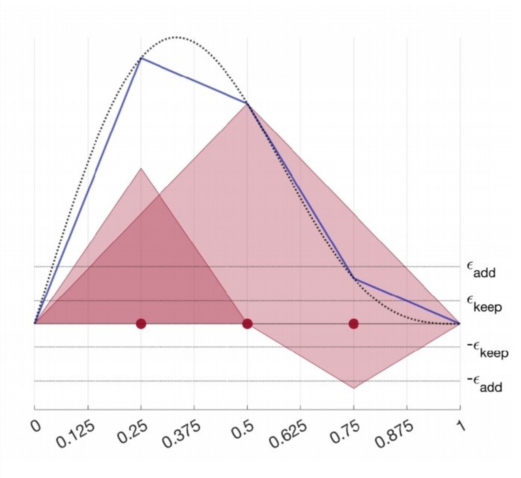

Figure 5: Adaptive Sparse Grids (Schaab and Zhang, 2022)

Adaptive Sparse Grids

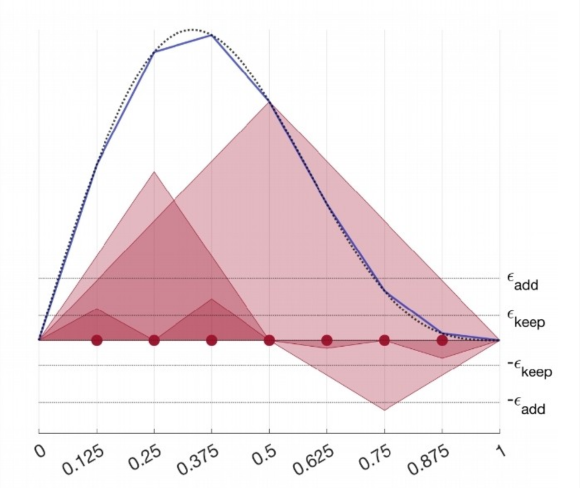

Figure 6: Adaptive Sparse Grids (Schaab and Zhang, 2022)

Adaptive Sparse Grids

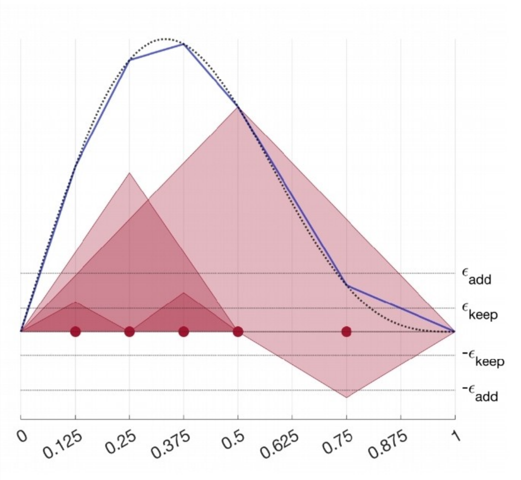

Figure 7: Adaptive Sparse Grids (Schaab and Zhang, 2022)

Back

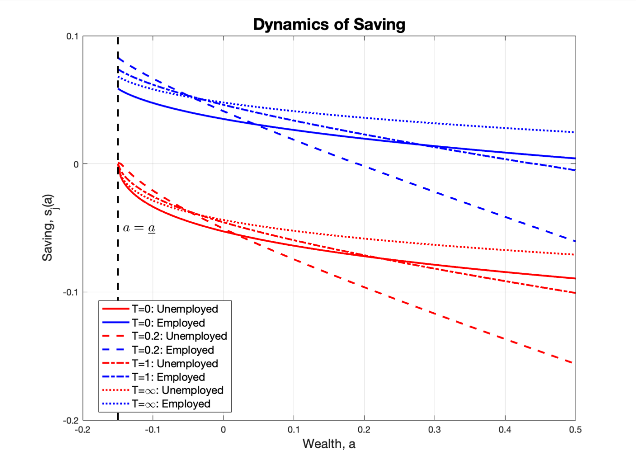

Figure 8: Time Path of Saving

Dynamics of Consumption

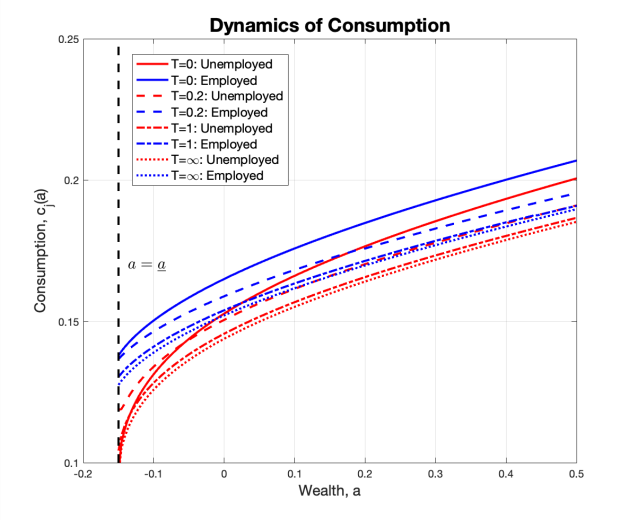

Figure 9: Time Path of Consumption

References

Schaab, A. and A. Zhang (2022). Dynamic programming in continuous time with adaptive sparse grids. *Available at SSRN 4125702*.