Computational Methods for Heterogeneous-Agent OLG Models

Zhigang Feng

Fall 2025

© Zhigang Feng • Redistribution prohibited without permission

Lecture Overview

- **The Challenge:**
  - Many agents per generation: one per generation without idiosyncratic shocks, infinitely many with idiosyncratic shocks
  - Distribution as state variable with aggregate uncertainty
- **Part I: No Aggregate Uncertainty**
  - Model with idiosyncratic shocks only
  - Stationary recursive competitive equilibrium
  - Computational algorithm: nested fixed point
- **Part II: With Aggregate Uncertainty**
  - Model without idiosyncratic shocks (simplified OLG)
  - Distribution of assets by generation as state variable
  - Solution via machine learning (homotopy methods)
- **Extensions:** Models with both types of shocks (DeepHAM)

© Zhiyang Fe...
Redistribution prohibited without permission

References

Main Reference for This Lecture:

- Nishiyama, S. and Smetters, K. (2013): "Analyzing Fiscal Policies in a Heterogeneous-Agent Overlapping-Generations Economy," *Handbook of Computational Economics*, Vol. 3, Chapter 3

Machine Learning Approaches:

- Azinovic, M., Gaegauf, L., and Scheidegger, S. (2022): "Deep Equilibrium Nets," *International Economic Review*
- Zemlicka, J. and Azinovic, M. (2023): "Economics-Inspired Neural Networks with Stabilizing Homotopies"

Extensions:

- Zhang, Adam (2025): "Lifecycle Financial Portfolios in General Equilibrium"

© Zhigang Feng Redistribution prohibited without permission

The Challenge

© Zhigang Feng • Redistribution prohibited without permission

Why Heterogeneous-Agent OLG Models?

Advantages over Representative Agent Models:

- Life-cycle properties matter for labor supply and savings decisions
- Intra-generational heterogeneity essential for distributional analysis
- Inter-generational wealth distribution important for tax timing

The Computational Challenge:

- Many agents per generation
  - Without idiosyncratic shocks: one representative agent per generation
  - With idiosyncratic shocks: infinitely many heterogeneous agents per generation (different asset holdings, earnings history)
- Distribution evolves over time
- General equilibrium: prices affect decisions, decisions affect prices

© Zheng Fang Redistribution prohibited without permission

Literature: Foundations of OLG Models

Theoretical Foundations:

- **Samuelson (1958):** First formal OLG model
- **Diamond (1965):** Neoclassical growth with OLG structure
- **Auerbach & Kotlikoff (1987):** Dynamic Fiscal Policy
  - First computational general equilibrium OLG model
  - Transition dynamics for policy analysis
  - Became standard framework for fiscal policy evaluation

Adding Heterogeneity (Deterministic):

- **Fullerton & Rogers (1993):** Multiple lifetime income groups
- **Kotlikoff, Smetters & Walliser (1998, 1999, 2007):** Progressive taxation and Social Security
- **Altig et al. (2001):** Fundamental tax reform in the U.S.

These models allow heterogeneity by lifetime income but remain deterministic

© Thomas Fischer • Redistribution prohibited without permission

Literature: Heterogeneous Agents with Uncertainty

Idiosyncratic Risk (No Aggregate Uncertainty):

- **Bewley (1986), Huggett (1993), Aiyagari (1994):** Incomplete markets
  - Uninsurable idiosyncratic wage shocks
  - Precautionary savings
  - Wealth inequality
- **İmrohoroğlu et al. (1995):** Life cycle analysis of Social Security
- **Hubbard, Skinner & Zeldes (1995):** Precautionary saving
- **Conesa & Krueger (1999, 2006):** Social Security reform with heterogeneity
- **Nishiyama & Smetters (2005, 2007):** Consumption taxes and Social Security
- **Conesa, Kitao & Krueger (2009):** Capital taxation

© Zhigang Feng. Redistribution prohibited without permission

Key Feature: Aggregate variables (prices) are deterministic, but individuals face idiosyncratic shocks

Literature: Aggregate Uncertainty (Current Frontier)

OLG with Aggregate Shocks:

- **Krueger & Kubler (2004, 2006):** Stochastic production in OLG
  - Computing equilibrium with aggregate uncertainty
  - Key challenge: distribution of wealth is a state variable
- **Hasanhodzic & Kotlikoff (2013):** Generational risk
  - 80-period OLG model with aggregate shocks
  - Early application of advanced computational methods

Machine Learning Approaches:

- **Azinovic, Gaegauf & Scheidegger (2022):** Deep Equilibrium Nets
- **Maliar, Maliar & Winant (2021):** Deep learning for heterogeneous agents
- **Zemlicka & Azinovic (2023):** Homotopy methods

Current Challenge: Models with both aggregate and idiosyncratic uncertainty

- **Zhang (2025):** combines both types of shocks

© Zhigang Feng ... Redistribution prohibited without permission

Major Applications in Economics Research

1. **Fiscal Policy Analysis:**

   - Tax reform: income tax vs. consumption tax
   - Social Security reform and privatization
   - Government debt and deficit policies
   - Intergenerational redistribution

2. **Distributional Analysis:**

   - Wealth and income inequality
   - Winners and losers from policy changes
   - Welfare analysis across generations and income groups
   - Transition costs of reforms

3. **Social Insurance Programs:**

   - Social Security benefit design
   - Medicare and healthcare reform

© Zhigang Feng • Redistribution prohibited without permission

Major Applications (continued)

4. Life-Cycle Behavior:

- Savings and portfolio choice over the life cycle
- Labor supply decisions at different ages
- Retirement timing
- Precautionary savings against uninsurable risks

5. Asset Pricing and Macroeconomics:

- Equity premium puzzle (limited participation)
- Risk-free rate with incomplete markets
- Term structure of interest rates
- Business cycle implications

6. Climate and Environmental Policy:

- Carbon taxes and intergenerational equity
- Climate change mitigation costs

© Zhigang Feng Redistribution prohibited without permission

Two Types of Uncertainty

1. Idiosyncratic Shocks:

- Wage/productivity shocks that average out in aggregate
- Create heterogeneity within generations
- Example: Some workers have high income, others low, but aggregate labor income predictable

2. Aggregate Shocks:

- Economy-wide shocks affecting everyone (business cycles, TFP shocks)
- Factor prices (wages, interest rates) become stochastic
- Example: Recessions affect all workers simultaneously

Key Distinction:

- **Without aggregate shocks:** Can work with stationary distributions
- **With aggregate shocks:** Must track distribution as state variable

© Zhigang Feng • Distribution prohibited without permission

State Variables: Two Scenarios

Scenario 1: Aggregate Uncertainty, No Idiosyncratic Shocks

State variables: $\{Z_t, a_{1,t}, a_{2,t}, \ldots, a_{I,t}\}$

- Z_t: aggregate productivity shock
- $a_{i,t}$: asset holdings of generation i at time t
- One representative agent per generation
- Manageable: I continuous state variables (e.g., 80 generations)

Scenario 2: Aggregate Uncertainty + Idiosyncratic Shocks

State variables: $\{Z_t, \Lambda_{1,t}(\cdot), \Lambda_{2,t}(\cdot), \ldots, \Lambda_{I,t}(\cdot)\}$

- $\Lambda_{i,t}(a)$: distribution function of assets for generation i at time t
- Infinite-dimensional state space per generation!
- This is the "curse of dimensionality"

© Zhigang Feng • Redistribution prohibited without permission

State Variables: Two Scenarios

This Lecture:

1. Start: Idiosyncratic shocks only (stationary equilibrium, no aggregate uncertainty)
2. Then: Aggregate shocks only (distribution as state, ML solution)

© Zhigang Feng • Redistribution prohibited without permission

Part I: Model with Idiosyncratic Shocks Only

© Zhigang Feng
Redistribution prohibited without permission

Model Components

Households:

- Live $i = 1, \ldots, I$ periods (ages 21-100)
- Retire at age I_R (age 65)
- Face idiosyncratic wage shocks: $\ln z_i = \rho \ln z_{i-1} + \epsilon_i$
- Choose consumption c, labor h, savings a'
- Individual state: $s = (i, a, b, e)$
  - i: age, a: assets, b: earnings history, e: working ability

Firms:

- Perfectly competitive, CRS: $F(K, L) = AK^{\theta}L^{1-\theta}$

Government:

- Progressive income tax, Social Security, consumption tax

No Aggregate Uncertainty: Prices (r, w) are deterministic and constant

© Zhigang Feng • Redistribution prohibited without permission

Household Optimization Problem

Value Function:

$$v(s; r, w) = \max_{c,h,a'} \left\{ u(c, h) + \tilde{\beta}\phi_i \mathbb{E}[v(s'; r, w)|s] \right\}$$

where $s' = (i + 1, a', b', e')$ and prices (r, w) taken as given

Budget Constraint:

$$a' = \frac{1}{1+\mu}\Big[(1+r)a + weh - \tau_I(ra, weh) - \tau_P(weh) + tr_{SS}(i, b) + tr_{LS} + \mathbf{1}_{\{i<I_R\}}q - (1+\tau_C)c\Big]$$

Key Features:

- Progressive income tax $\tau_I(y)$, Social Security $tr_{SS}(i, b)$
- Idiosyncratic shock e with AR(1) process, transition matrix Π_i
- Prices (r, w) treated as parameters (partial equilibrium)

© Zhigang Feng
Redistribution prohibited without permission

Distribution of Households

Measure of Households: $\lambda(i, a, b, e)$

Interpretation: Fraction of population with state (i, a, b, e)

Properties:

- Non-negative: $\lambda(i, a, b, e) \geq 0$ for all (i, a, b, e)
- Integrates to one: $\sum_{i,a,b,e} \lambda(i, a, b, e) = 1$

Law of Motion:

Given decision rules $a'(s; r, w)$, $b'(s; r, w)$ and transition matrix $\Pi_i(e'|e)$:

$$\lambda(i+1, a', b', e') = \frac{\phi_i}{1+\nu} \sum_{a,b,e} \mathbf{1}_{\{a'(s)=a', b'(s)=b'\}} \Pi_i(e'|e) \lambda(i, a, b, e)$$

where:

- $\phi_i/(1+\nu)$: survival rate adjusted for population growth

© Zhigang Feng Redistribution prohibited without permission

Computing the Distribution: Two Methods

Goal: Find the distribution $\lambda(i, a, b, e)$ consistent with decision rules

Method 1: Forward Simulation

- Start with large population of agents (e.g., 50,000-100,000)
- Initialize: all agents start at age $i = 1$ with $a = 0$, $b = 0$, e drawn from $\Pi_1(e)$
- For each period, agents:
  - Make decisions using optimal policy functions
  - Age by one period, some die (survival rate ϕ_i)
  - Receive new shock e' from transition matrix $\Pi_i(e'|e)$
  - New cohort enters with $a = 0$, $b = 0$
- Run until distribution stabilizes (typically 500-1000 periods)
- Construct histogram to approximate $\lambda(i, a, b, e)$

© Zhigang Feng — Redistribution prohibited without permission

Advantages: Simple to implement, easy to parallelize

Disadvantages: Slower convergence, requires many agents for accuracy

Computing the Distribution: Transition Matrix Method

Method 2: Direct Iteration with Transition Matrix

Setup:

- Discretize state space: $(i, a, b, e) \in \{1, \ldots, I\} \times \{a_1, \ldots, a_J\} \times \{b_1, \ldots, b_K\} \times \{e_1, \ldots, e_L\}$
- Total nodes: $N = I \times J \times K \times L$ (e.g., 80 × 65 × 20 × 5 = 332,800 nodes)

Algorithm:

1. **Initialize:** Set $\lambda^{(0)}(1, 0, 0, e) = \Pi_1(e)$ (newborns), others to 0
2. **Iterate forward in age:** For $i = 1, \ldots, I - 1$, iterate until convergence

Key Challenge: Policy functions $a'(i, a, b, e)$ and $b'(i, a, b, e)$ typically do not lie on grid points (a_j, b_k)

Details on next slides...

© Zhigang Feng (Distribution prohibited without permission)

Bilinear Interpolation: Distributing Mass

Problem: Optimal choice (a', b') not on grid $\Rightarrow$ must distribute mass to neighboring grid points

Bilinear Interpolation Procedure:

For each state (i, a_j, b_k, e_l) with population mass $\lambda(i, a_j, b_k, e_l)$:

Step 1: Compute optimal next period assets:

$$a' = a'(i, a_j, b_k, e_l), \quad b' = b'(i, a_j, b_k, e_l)$$

Step 2: Find grid indices: $a_{j'} \leq a' < a_{j'+1}$, $b_{k'} \leq b' < b_{k'+1}$

Step 3: Compute interpolation weights:

$$\omega_a = \frac{a' - a_{j'}}{a_{j'+1} - a_{j'}} \in [0, 1]$$

$$\omega_b = \frac{b' - b_{k'}}{b_{k'+1} - b_{k'}} \in [0, 1]$$

Step 4: Distribute mass to 4 adjacent nodes:

© Zhigang Feng — Redistribution prohibited without permission

Updating the Distribution: Complete Formula

Full Update Equation:

For next period state $(i+1, a_{j'}, b_{k'}, e_{l'})$:

$$\lambda^{(n+1)}(i+1, a_{j'}, b_{k'}, e_{l'}) = \frac{\phi_i}{1+\nu} \sum_{j,k,l} W_{jk \to j'k'}(i) \cdot \Pi_i(e_{l'}|e_l) \cdot \lambda^{(n)}(i, a_j, b_k, e_l)$$

where $W_{jk \to j'k'}(i)$ is the interpolation weight matrix

For Each Age and Shock:

- Weights W depend on policy functions $a'(i, a_j, b_k, e_l)$, $b'(i, a_j, b_k, e_l)$
- Four non-zero weights per source node (bilinear interpolation)
- Transition matrix $\Pi_i(e_{l'}|e_l)$ for idiosyncratic shocks

© Zhigang Feng Redistribution prohibited without permission

Key Property: Bilinear interpolation preserves total measure:

$$\sum_{j',k',l'} \lambda^{(n+1)}(i+1, a_{j'}, b_{k'}, e_{l'}) = \frac{\phi_i}{1+\nu} \sum_{j,k,l} \lambda^{(n)}(i, a_j, b_k, e_l)$$

Eigenvector Approach: Matrix Formulation

Alternative: Solve as Eigenvector Problem

Key Insight: Stationary distribution satisfies $\lambda^* = T\lambda^*$ where T is the transition operator

Matrix Formulation:

Step 1: Stack distribution into vector $\vec{\lambda} \in \mathbb{R}^N$ where $N = I \times J \times K \times L$

For example: $\vec{\lambda} = [\lambda(1, a_1, b_1, e_1), \lambda(1, a_1, b_1, e_2), \ldots, \lambda(I, a_J, b_K, e_L)]^T$

Step 2: Construct transition matrix $\mathbf{T} \in \mathbb{R}^{N \times N}$:

Each element captures probability of transitioning from state (i, a_j, b_k, e_l) to $(i', a_{j'}, b_{k'}, e_{l'})$:

$$T_{(i',j',k',l'),(i,j,k,l)} = \begin{cases} \frac{\phi_i}{1+\nu} \cdot W_{jk \to j'k'}(i) \cdot \Pi_i(e_{l'}|e_l) & \text{if } i' = i+1 \\ 0 & \text{otherwise} \end{cases}$$

where $W_{jk \to j'k'}(i)$ are bilinear interpolation weights from previous slide

© Zhigang Feng Redistribution prohibited without permission

Solving the Eigenvector Problem: Algorithms

Method 1: Power Iteration

Simple iterative algorithm:

1. Start with initial guess: $\vec{\lambda}^{(0)} = [1/(N), 1/(N), \ldots, 1/(N)]^T$ (uniform)
2. Iterate: $\vec{\lambda}^{(n+1)} = \mathbf{T}\vec{\lambda}^{(n)}$
3. Normalize: $\vec{\lambda}^{(n+1)} = \vec{\lambda}^{(n+1)} / \|\vec{\lambda}^{(n+1)}\|_1$
4. Check convergence: if $\|\vec{\lambda}^{(n+1)} - \vec{\lambda}^{(n)}\| < \epsilon$, stop

Convergence rate depends on spectral gap: $|\lambda_2|$ where $|\lambda_1| = 1$

Method 2: Sparse Eigenvalue Solvers

Use specialized algorithms for sparse matrices:

- **Arnoldi iteration:** Build Krylov subspace, find eigenvector
- **Software:** ARPACK, SciPy's `sparse.linalg.eigs`, MATLAB's `eigs`
- Find eigenvector corresponding to eigenvalue closest to 1

© Zhiang Peng
Redistribution prohibited without permission

Eigenvector Approach: Practical Considerations

Advantages:

- Mathematically elegant: solves for exact stationary distribution
- Can use highly optimized sparse eigenvalue solvers
- No need for forward iteration loop

Disadvantages:

- **Memory:** Must store sparse matrix $\mathbf{T}$ ($\sim 4L \times N$ non-zero entries)
  - Example: $4 \times 5 \times 332,800 = 6,656,000$ entries
  - With 16 bytes per entry (index + value): ~100 MB
- **Construction cost:** Building $\mathbf{T}$ requires evaluating policy function at all grid points
- **Not needed in practice:** Forward iteration (Method 2) converges in 10-30 iterations, usually faster

When to Use:

- Research/debugging: verify forward iteration results
- Multiple distributions: if solving for stationary distribution many times with same policy functions

© Zhigang Feng | Reproduction prohibited without permission

Stationary Distribution

Definition: A distribution λ^* is stationary if:

$$\lambda^*(i+1, a', b', e') = \frac{\phi_i}{1+\nu} \sum_{a,b,e} \mathbf{1}_{\{a'(s)=a', b'(s)=b'\}} \Pi_i(e'|e) \lambda^*(i, a, b, e)$$

In words: The distribution reproduces itself over time when households follow optimal decision rules

Implications:

- Aggregate variables (K, L, Y) are constant
- Prices (r, w) are constant
- Distribution across (i, a, b, e) is time-invariant
- This simplifies computation: we seek a fixed point (λ^*, r^*, w^*)

Why does this work?

- No aggregate shocks $\Rightarrow$ aggregate quantities predictable
- Idiosyncratic shocks average out across many agents

© Zhigang Feng • Redistribution prohibited without permission

Market Clearing Conditions

Asset Market:

$$K^* = \sum_{i,a,b,e} a \cdot \lambda^*(i, a, b, e)$$

Capital demand (from firms): $K^d(r, w)$ from FOC $r = F_K(K, L) - \delta$

Labor Market:

$$L^* = \sum_{i,a,b,e} h(s; r, w) \cdot e \cdot \lambda^*(i, a, b, e)$$

Labor demand (from firms): $L^d(r, w)$ from FOC $w = F_L(K, L)$

Market Clearing:

- $K^* = K^d(r^*, w^*)$
- $L^* = L^d(r^*, w^*)$
- Prices must adjust so aggregate supply equals aggregate demand

© Zhigang Feng Redistribution prohibited without permission

Stationary Recursive Competitive Equilibrium (SRCE)

Definition: A SRCE consists of:

1. Prices: (r^*, w^*)
2. Value function: $v^*(s; r^*, w^*)$
3. Decision rules: $c^*(s; r^*, w^*)$, $h^*(s; r^*, w^*)$, $a'^*(s; r^*, w^*)$
4. Stationary distribution: $\lambda^*(i, a, b, e)$

such that:

1. **Household Optimization:** Given (r^*, w^*), (v^*, c^*, h^*, a'^*) solve the household problem

2. **Stationary Distribution:** λ^* is consistent with decision rules:

$$\lambda^*(i+1, a', b', e') = \frac{\phi_i}{1+\nu} \sum_{a,b,e} \mathbf{1}_{\{a'^*(s)=a', b'^*(s)=b'\}} \Pi_i(e'|e) \lambda^*(i, a, b, e)$$

3. **Market Clearing:**

$$\sum_{i,a,b,e} a \cdot \lambda^*(i, a, b, e) = K^d(r^*, w^*)$$

© Zhigang Feng Redistribution prohibited without permission

Beyond Steady State: Transition Dynamics

Why Study Transition Dynamics?

Stationary equilibrium analysis is useful but limited:

- Compares two steady states (before/after policy change)
- Ignores the path from one steady state to another
- Cannot analyze short-run vs. long-run effects
- Misses welfare costs of transition

© Yiyang Feng • Redistribution prohibited without permission

Key Questions Requiring Transition Analysis:

1. **Policy reforms:** What happens when government changes tax rates or Social Security?
2. **Welfare analysis:** Who wins/loses during transition?
3. **Aggregate shocks:** How does economy respond to unexpected TFP shocks?
4. **Demographic changes:** Effects of aging population over time

Example: Social Security privatization

Transition Dynamics: Problem Setup

Setting:

- Economy in initial steady state at $t = 0$
- At $t = 1$: policy change announced and implemented
- Economy gradually transitions to new steady state
- Need to solve for entire equilibrium path: $t = 1, 2, \ldots, T$

What Changes Over Time:

- Distribution $\lambda_t(i, a, b, e)$ evolves period by period
- Prices (r_t, w_t) adjust as capital and labor supplies change
- Policy variables adjust (if policy is phased in gradually)
- Households have perfect foresight about future prices

What We Need to Find:

Sequence of equilibria for $t = 1, \ldots, T$ where for each t:

- Decision rules: $c_t(s), h_t(s), a'_t(s)$ given anticipated future prices

© Zhigang Feng • Distribution prohibited without permission

Computing Transition Dynamics: Overview

Key Difference from Steady State:

- Steady state: Find single set of prices (r^*, w^*)
- Transition: Find sequence of prices $\{(r_t, w_t)\}_{t=1}^{T}$
- Much higher dimensional problem!

Computational Challenge:

State space expands dramatically:

- Not just individual state (i, a, b, e)
- Also need to track time t and entire future price path
- Households form expectations about $\{(r_s, w_s)\}_{s=t+1}^{T}$

Solution Approach: Backward Induction with Iteration

1. Choose T large enough that economy reaches new steady state
2. Backward solve household problems from T to 1
3. Forward simulate distribution from 1 to T

© Zhitang Feng • Redistribution prohibited without permission

Transition Dynamics Algorithm: Step-by-Step

Step 0: Initialization

1. Compute initial steady state (period 0): (r_0, w_0, λ_0^*)
2. Compute final steady state (period T): (r_T, w_T, λ_T^*)
3. Choose T large enough (typically 100-200 years)
4. Initialize price path: $\{r_t^{(0)}, w_t^{(0)}\}_{t=1}^{T-1}$
   - Simple approach: linear interpolation from initial to final steady state
   - Better: use prices from final steady state throughout

Iteration $n = 1, 2, \ldots$ until convergence:

1. **Backward solve household problems** (from $t = T - 1$ down to $t = 1$)
2. **Forward simulate distributions** (from $t = 1$ up to $t = T$)
3. **Compute market clearing errors**
4. **Update price path**

Details on next slides...

© Zhigang Feng • Redistribution prohibited without permission

Step 1: Backward Solving Household Problems

Goal: Find decision rules $d_t(s) = \{c_t(s), h_t(s), a'_t(s)\}$ for each t

Backward Induction from $t = T-1$ to $t = 1$:

At $t = T$: In final steady state, use steady state value function and decision rules

At $t = T-1, T-2, \ldots, 1$: For each age $i = I, I-1, \ldots, 1$:

1. Know value function for next period: $v_{t+1}(i+1, a', b', e')$ from previous iteration
2. Know prices: $r_t^{(n)}, w_t^{(n)}$ from current guess
3. Solve household problem:

$$v_t(i, a, b, e) = \max_{c,h,a'} \left\{ u(c, h) + \tilde{\beta}\phi_i \mathbb{E}[v_{t+1}(i+1, a', b', e')|e] \right\}$$

   subject to budget constraint with prices $(r_t^{(n)}, w_t^{(n)})$
4. Use Newton solver with interpolation for v_{t+1} (as in steady state)
5. Store decision rules: $c_t(i, a, b, e)$, $h_t(i, a, b, e)$, $a'_t(i, a, b, e)$

Key Point: Households have perfect foresight about future prices $\{r_s^{(n)}, w_s^{(n)}\}_{s=t+1}^{T}$

© Zhiheng Feng Redistribution prohibited without permission

Step 2: Forward Simulating Distributions

Goal: Compute $\{\lambda_t\}_{t=1}^T$ consistent with decision rules

Forward Iteration from $t = 1$ to $t = T$:

At $t = 1$: Start with initial steady state distribution $\lambda_1 = \lambda_0^*$

For $t = 1, 2, \ldots, T-1$:

Use law of motion with decision rules from Step 1:

$$\lambda_{t+1}(i+1, a', b', e') = \frac{\phi_i}{1+\nu} \sum_{a,b,e} \mathbf{1}_{\{a_t'(s)=a', b_t'(s)=b'\}} \Pi_i(e'|e) \lambda_t(i, a, b, e)$$

Use bilinear interpolation as before for $a_t'(i, a, b, e)$, $b_t'(i, a, b, e)$ not on grid

Verification:

- At $t = T$: Check if $\lambda_T \approx \lambda_T^*$ (final steady state distribution)
- If not close, need larger T

© Zhigang Feng Redistribution prohibited without permission

Step 3-4: Market Clearing and Price Updates

Step 3: Compute Excess Demands

For each $t = 1, \ldots, T$:

1. Aggregate supplies from distribution:

$$K_t^s = \sum_{i,a,b,e} a \cdot \lambda_t(i, a, b, e)$$

$$L_t^s = \sum_{i,a,b,e} h_t(i, a, b, e) \cdot e \cdot \lambda_t(i, a, b, e)$$

2. Compute factor demands from firm FOCs:

$$r_t^{(n)} = F_K(K_t^s, L_t^s) - \delta$$

$$w_t^{(n)} = F_L(K_t^s, L_t^s)$$

3. Compute excess demands:

$$ED_{K,t} = K_t^s - K_t^d(r_t^{(n)}, w_t^{(n)}), \quad ED_{L,t} = L_t^s - L_t^d(r_t^{(n)}, w_t^{(n)})$$

© Zhirang Feng
Redistribution prohibited without permission

Step 4: Update Price Path

Convergence and Computational Cost

Convergence Check:

Stop when price path is stable:

$$\max_{t=1,\ldots,T-1} \left| \frac{K_t^{(n+1)} - K_t^{(n)}}{1 + K_t^{(n)}} \right| < \epsilon$$

Typical tolerance: $\epsilon = 10^{-4}$

Computational Cost:

- Steady state: 6-10 iterations, ∼1 hour total
- Transition: 6-10 iterations, but each iteration:
  - Solve household problem for $T \times I \times J \times K \times L$ nodes
  - Example: $T = 150$, total nodes ≈ 50 million
  - Time per iteration: ∼1-2 hours on modern workstation
  - **Total: 10-20 hours** for full transition path
- Storage: Must store decision rules for all (t, i, a, b, e)
  - Example: 4 decision rules × 50 million nodes × 8 bytes ≈ 1.6 GB

© Zhigang Feng • Redistribution prohibited without permission

Why Transition Dynamics Matter: Example

Social Security Privatization (from Nishiyama & Smetters 2007):

Policy: Cut benefits by 50%, reduce payroll taxes

Long-Run Steady State Effects (t > 100):

- Capital stock: +24% (more private savings)
- GDP: +11% (higher capital and labor)
- Welfare of newborns: +6.9% (lower taxes, higher wages)

⇒ Looks great if we only compare steady states!

But Transition Effects (t=1-40):

- Current workers: -2.9% welfare (pay twice: support retirees + save for own retirement)
- Capital builds gradually over 40+ years
- Different cohorts win/lose at different times

© Zhiyang Jia • Redistribution prohibited without permission

Transition Dynamics: Connection to Part II

From Stationary to Aggregate Shocks:

The transition dynamics algorithm prepares us for models with aggregate uncertainty:

Similarities:

- Both require tracking distribution evolution over time
- Both involve sequences of prices and distributions
- Both require perfect foresight / rational expectations

Key Difference:

- **Transition:** Deterministic path, solve once
- **Aggregate shocks:** Stochastic paths, need policy functions $r(\mathbf{x}_t, Z_t)$, $w(\mathbf{x}_t, Z_t)$

The Challenge in Part II:

With aggregate uncertainty:

© Zhigang Feng
Redistribution prohibited without permission

High-Level Conceptual View:

Stationary equilibrium is a nested fixed point problem:

Outer Loop (Price Fixed Point):

- Guess prices (r, w)
- For given prices, solve inner loop
- Check if markets clear
- Update prices if not, iterate until convergence

Inner Loop (Distribution Fixed Point):

- Given prices (r, w), solve household problem $\Rightarrow$ get decision rules
- Given decision rules, find stationary distribution λ^*
- Distribution is a fixed point: $\lambda^* = T(\lambda^*)$ where T is the transition operator

Key Insight: Prices matter for household decisions, household decisions determine aggregate supplies, aggregate supplies must equal demands for equilibrium prices

Redistribution prohibited without permission
© Zheng Song

Algorithm Overview: Nested Fixed Point (Part 1)

Step 0: Initialization

- Initialize prices $(r^{(0)}, w^{(0)})$ using steady-state approximation
- Set convergence tolerance ϵ (typically 10^{-4})

Step 1.1: Solve Household Problem

Given $(r^{(n)}, w^{(n)})$:

- Use backward induction (value function iteration)
- For each age $i = I$ down to $i = 1$:
  - Solve optimization problem for all states (a, b, e)
  - Use Newton solver for complementarity problem
  - Interpolate marginal values for next period
- Obtain decision rules: $a'^*(s; r^{(n)}, w^{(n)})$, $h^*(s; r^{(n)}, w^{(n)})$

Step 1.2: Find Stationary Distribution (Inner Fixed Point)

- Guess $\lambda^{(0)}$, set $\lambda(1, 0, 0, e) = \Pi_1(e)$ (newborns)

© Zhe Fang • Redistribution prohibited without permission

Algorithm Overview: Nested Fixed Point (Part 2)

Step 1.3: Compute Aggregate Supplies

Using stationary distribution λ^*:

$$K^s = \sum_{i,a,b,e} a \cdot \lambda^*(i,a,b,e)$$

$$L^s = \sum_{i,a,b,e} h^*(s; r^{(n)}, w^{(n)}) \cdot e \cdot \lambda^*(i,a,b,e)$$

Step 1.4: Compute Excess Demands

$$ED_K = K^s - K^d(r^{(n)}, w^{(n)}) \quad \text{where } K^d \text{ from firm FOC}$$

$$ED_L = L^s - L^d(r^{(n)}, w^{(n)})$$

© Zhigang Feng • Redistribution prohibited without permission

Step 1.5: Check Convergence

- If $|ED_K| < \epsilon$ and $|ED_L| < \epsilon$:

Computational Details

Household Problem:

- Discretize (a, b, e) space (typically 100-500 points per dimension)
- Value function iteration with linear/bilinear interpolation
- Backward induction from age I to age 1

Distribution:

- Forward iteration from age 1 to age I
- Use bilinear interpolation to preserve measure
- Young cohort enters with $a = 0$, $b = 0$

Aggregation:

- Compute K^s, L^s by summing over all grid points
- Use numerical integration (trapezoidal rule)

Computational Cost:

© Zhigang Feng Redistribution prohibited without permission

Part II: Model with Aggregate Uncertainty Only

© Zhigang Feng • Redistribution prohibited without permission

Motivation: Why Machine Learning?

The Problem with Aggregate Uncertainty:

- State space: $(Z_t, a_{1,t}, \ldots, a_{I,t})$ where $I \approx 80$ generations
- 80+ continuous state variables
- Traditional grid methods: curse of dimensionality
  - 100^{80} grid points $\Rightarrow$ completely infeasible

Machine Learning Solution:

- Use neural networks to approximate policy functions and prices
- No need for explicit grids
- Scalable to high dimensions
- Based on: Azinovic, Gaegauf & Scheidegger (2022), Zemlicka & Azinovic (2023)

Key Innovation: Market clearing layer + stabilizing homotopy

© Zhigang Feng Redistribution prohibited without permission

Model Setup: Demographics and Preferences

Households:

- Live for H periods deterministically (e.g., $H = 7$ representing ages 20-90)
- No bequest motive: $k_t^{H-1} = b_t^{H-1} = 0$
- CRRA utility: $u(c) = \frac{c^{1-\gamma}-1}{1-\gamma}$ with $\gamma = 2$
- Patience: $\beta = 0.96$ (annual)

Labor Supply:

- Exogenous labor endowment l_t^h follows hump-shaped profile over life cycle
- Work for first $\sim$60% of life, retire afterwards
- Total labor: $L_t = \sum_{h=0}^{H-1} l_t^h$

No Idiosyncratic Shocks:

- One representative agent per generation
- All age-h households identical
- Distribution is degenerate: just track asset holdings by generation

© Zhigang Feng • Redistribution prohibited without permission

Model Setup: Technology and Uncertainty

Aggregate Uncertainty:

- TFP shock: $\log(Z_{t+1}) = \rho_Z \log(Z_t) + \sigma_Z \epsilon_t$
  - $\rho_Z = 0.85$, $\sigma_Z = 0.03$ (typical business cycle calibration)
- Depreciation depends on TFP: $\delta_t = \min\left\{\delta \frac{2}{1+Z_t}, 1\right\}$
  - Higher TFP $\Rightarrow$ lower depreciation
  - Increases dispersion of capital returns

Production:

- Cobb-Douglas: $Y_t = Z_t K_t^{\alpha} L_t^{1-\alpha}$ with $\alpha = 1/3$
- Firm FOCs determine factor prices:

$$r_t = \alpha Z_t K_t^{\alpha-1} L_t^{1-\alpha}$$
$$w_t = (1-\alpha) Z_t K_t^{\alpha} L_t^{-\alpha}$$

© Zhigang Feng Redistribution prohibited without permission

Model Setup: Budget Constraints and Assets

Two Assets:

1. **Capital** $k_t^h \geq 0$ (no short sales)
   - Risky: return depends on TFP shocks
   - Subject to adjustment costs: $\psi^k(k_t^h - k_{t-1}^{h-1})^2$
2. **Bonds** $b_t^h \geq \underline{b}$ (borrowing constraint)
   - Riskless: price p_t^b determined in equilibrium
   - Net supply: $B = 0$ (market clears)

Budget Constraint:

$$c_t^h + p_t^b b_t^h + k_t^h + \psi^k(k_t^h - k_{t-1}^{h-1})^2 = l_t^h w_t + b_{t-1}^{h-1} + k_{t-1}^{h-1}(1 - \delta_t + r_t)$$

where consumption c_t^h and savings (k_t^h, b_t^h) are chosen optimally

Euler Equations:

$$u'(c_t^h) = \beta \mathbb{E}_t \left[\frac{1}{p_t^b} u'(c_{t+1}^{h+1}) \right] \quad \text{(bonds)}$$

© Zhigang Feng Redistribution prohibited without permission

State Variables and Distribution (Part 1)

State of the Economy:

$$\mathbf{x}_t = (Z_t, k_{0,t}, k_{1,t}, \ldots, k_{H-1,t}, b_{0,t}, b_{1,t}, \ldots, b_{H-1,t})$$

- Z_t: aggregate TFP shock (1 scalar)
- $k_{h,t}$: capital holdings of age-h cohort (H scalars)
- $b_{h,t}$: bond holdings of age-h cohort (H scalars)
- **Total dimension:** $1 + 2H$ (e.g., $\sim$15 dimensions for $H = 7$, $\sim$160 for $H = 80$)

Interpretation as Distribution:

- No idiosyncratic shocks $\Rightarrow$ distribution is degenerate
- All agents of age h hold same amounts $(k_{h,t}, b_{h,t})$
- "Distribution" = vector of asset holdings by generation
- This vector evolves over time based on:
  - Optimal savings decisions
  - Aging (assets shift forward one generation)
  - TFP shocks

© Zhisheng Feng Redistribution prohibited without permission

State Variables and Distribution (Part 2)

Law of Motion:

The state evolves as:

$$\mathbf{x}_{t+1} = (Z_{t+1}, k_{0,t+1}, \ldots, k_{H-1,t+1}, b_{0,t+1}, \ldots, b_{H-1,t+1})$$

Components:

1. **TFP shock:** $\log(Z_{t+1}) = \rho_Z \log(Z_t) + \sigma_Z \epsilon_{t+1}$
2. **Asset holdings:** Assets of generation h at time $t+1$ are savings of generation $h-1$ at time t:

$$k_{0,t+1} = 0, \quad b_{0,t+1} = 0 \quad \text{(newborns)}$$
$$k_{h,t+1} = g^k_{h-1}(\mathbf{x}_t), \quad b_{h,t+1} = g^b_{h-1}(\mathbf{x}_t) \quad \text{for } h = 1, \ldots, H-1$$

where g^k_h, g^b_h are policy functions

Why track distribution?

- Aggregate capital: $K_t = \sum_{h=0}^{H-1} k_{h,t}$ determines factor prices
- Prices affect all households' decisions

© Zhigang Feng Redistribution prohibited without permission

Recursive Competitive Equilibrium: Definition

A recursive competitive equilibrium consists of:

1. Policy Functions:

- Capital savings: $k_t^h = g_h^k(\mathbf{x}_t)$ for $h = 0, \ldots, H-2$
- Bond savings: $b_t^h = g_h^b(\mathbf{x}_t)$ for $h = 0, \ldots, H-2$
- Bond price: $p_t^b = p^b(\mathbf{x}_t)$
- Terminal generation: $k_t^{H-1} = b_t^{H-1} = 0$ (no bequest motive)

2. Value Functions:

- $V^h(\mathbf{x}_t)$ for each generation $h = 0, \ldots, H-1$

3. Aggregate Quantities:

- Aggregate capital: $K_t = \sum_{h=0}^{H-1} k_t^h = \sum_{h=0}^{H-1} g_h^k(\mathbf{x}_t)$
- Aggregate labor: $L_t = \sum_{h=0}^{H-1} l_t^h$ (exogenous labor supply profile)

Redistribution prohibited without permission
© Zhigang Feng

Recursive Competitive Equilibrium: Law of Motion

4. Law of Motion of State:

The state vector evolves as:

$$\mathbf{x}_{t+1} = (Z_{t+1}, k_{0,t+1}, \ldots, k_{H-1,t+1}, b_{0,t+1}, \ldots, b_{H-1,t+1})$$

Components:

(a) TFP Evolution:

$$\log(Z_{t+1}) = \rho_Z \log(Z_t) + \sigma_Z \epsilon_{t+1}$$

(b) Asset Holdings Evolution:

Asset holdings of generation h at time $t+1$ are the savings of generation $h-1$ at time t:

$$k_{0,t+1} = 0 \quad \text{(newborns enter with no assets)}$$
$$k_{h,t+1} = g^k_{h-1}(\mathbf{x}_t) \quad \text{for } h = 1, \ldots, H-1$$
$$b_{0,t+1} = 0 \quad \text{(newborns)}$$
$$b_{h,t+1} = g^b_{h-1}(\mathbf{x}_t) \quad \text{for } h = 1, \ldots, H-1$$

© Zhigang Feng
Redistribution prohibited without permission

Recursive Competitive Equilibrium: Equilibrium Conditions

Equilibrium Conditions (must hold for all states $\mathbf{x}_t$):

(i) Household Optimization:

Policy functions satisfy first-order conditions (Euler equations):

$$u'(c_t^h) = \beta \mathbb{E}_t \left[\frac{1}{p_t^b} u'(c_{t+1}^{h+1}) \right] \quad \text{(bonds)}$$

$$u'(c_t^h) = \beta \mathbb{E}_t \left[u'(c_{t+1}^{h+1}) \frac{1 - \delta_{t+1} + r_{t+1} - 2\psi^k(k_{t+1}^{h+1} - k_t^h)}{1 + 2\psi^k(k_t^h - k_{t-1}^{h-1})} \right]$$

plus complementarity conditions for constraints $b_t^h \geq \underline{b}$, $k_t^h \geq 0$

(ii) Market Clearing:

$$K_t = \sum_{h=0}^{H-1} k_t^h = \sum_{h=0}^{H-1} g_h^k(\mathbf{x}_t) \quad \text{(capital market)}$$

$$0 = \sum_{h=0}^{H-1} b_t^h = \sum_{h=0}^{H-1} g_h^b(\mathbf{x}_t) \quad \text{(bond market, net supply = 0)}$$

© Zhigang Feng Redistribution prohibited without permission

Neural Network Approximation (Part 1)

Approach: Use a single neural network to approximate all functions

Network Architecture:

- Input: state vector $\mathbf{x}_t \in \mathbb{R}^{1+2H}$
- Hidden layers: 2 layers, N neurons each (typically $N = 64$ or 128)
- Activation: SELU (Scaled Exponential Linear Unit)
- Output: $(2H - 2)$ savings policies + 1 bond price
  - $[g_0^k(\mathbf{x}_t), \ldots, g_{H-2}^k(\mathbf{x}_t), g_0^b(\mathbf{x}_t), \ldots, g_{H-2}^b(\mathbf{x}_t), p^b(\mathbf{x}_t)]$
- Total parameters: ∼10,000 for moderate size network

Why Single Network?

- Share learned representations across all functions
- More efficient than separate networks for each age
- Hidden layers capture common economic relationships
- Reduces total parameters needed

© Zhigang Feng • Redistribution prohibited without permission

Neural Network Approximation (Part 2)

Output Transformation (Market Clearing Layer):

Raw network outputs are transformed to satisfy economic constraints:

1. Capital Policies (No Short Sales):

$$\hat{k}_h = 0.1 \cdot \text{softplus}(\text{network output}_h) \quad \Rightarrow \quad k_h \geq 0$$

Softplus ensures non-negativity smoothly: $\text{softplus}(x) = \log(1 + e^x)$

2. Bond Policies (Market Clearing):

Enforce $\sum_h b_h = 0$ by construction:

$$\tilde{b}_h = m_b \cdot (\text{network output}_{H-1+h}) \quad \text{(homotopy mask)}$$

$$b_h = \tilde{b}_h - \frac{1}{H-1} \sum_{h'=0}^{H-2} \tilde{b}_{h'} \quad \text{(center to zero)}$$

where $m_b \in [0, 1]$ is homotopy parameter

© Zhigang Feng Redistribution prohibited without permission

Loss Function: Equilibrium Residuals

Goal: Train network to minimize equilibrium condition errors

Equilibrium Residuals (for generation $h < H-1$):

Fisher-Burmeister reformulation of complementarity conditions:

$$\epsilon_h^k = \psi^{FB}\left(\frac{(u')^{-1}\left(\beta\mathbb{E}_t[u'(c_{t+1}^{h+1})R_{t+1,h}^k]\right)}{c_t^h} - 1, \frac{k_t^h}{c_t^h}\right)$$

$$\epsilon_h^b = \psi^{FB}\left(\frac{(u')^{-1}\left(\beta\mathbb{E}_t[\frac{u'(c_{t+1}^{h+1})}{p_t^b}]\right)}{c_t^h} - 1, \frac{b_t^h - \underline{b}}{c_t^h}\right)$$

where $\psi^{FB}(a,b) = a + b - \sqrt{a^2+b^2}$ and $R_{t+1,h}^k$ includes adjustment costs

Loss Function:

$$\mathcal{L}(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x}}\left[\sum_{\cdot}^{H-2}(\epsilon_h^k)^2 + w_b\sum_{\cdot}^{H-2}(\epsilon_h^b)^2 + 100\cdot(\max\{0,-c_t^h\})^2\right]$$

© Zhenglong Feng · Redistribution prohibited without permission

Training Algorithm: Overview

Key Idea: Alternate between:

1. **Training:** Update neural network weights using gradient descent
2. **Simulation:** Generate new training data by simulating economy forward

Why This Works:

- Cannot pre-specify training data (equilibrium unknown)
- Generate data on-the-fly using current policy approximation
- Gradually improve approximation through iteration
- Similar to fitted Q-iteration in reinforcement learning

Main Loop Structure:

0. Initialize network parameters $\theta^{(0)}$, state cloud $\{\mathbf{x}_i^{(0)}\}_{i=1}^{N_{\text{cloud}}}$

For episode $k = 1, \ldots, K$:

1. **Train:** Update $\theta^{(k-1)} \to \theta^{(k)}$ using gradient descent on loss
2. **Simulate:** Use policy $\theta^{(k)}$ to update state cloud

© Zhiyang Feng • Redistribution prohibited without permission

Step 1: Generating Training Samples

Goal: Create diverse set of economic states to train on

Initialization (Episode 0):

- Start from deterministic steady state (no shocks)
- Add small perturbations: $\mathbf{x}_i^{(0)} = \mathbf{x}_{ss} + \mathcal{N}(0, \sigma_{\text{init}})$
- Number of clouds: $N_{\text{cloud}} \approx 10,000$ parallel trajectories

Forward Simulation (Each Episode):

For each state $\mathbf{x}_i^{(k-1)}$ in cloud:

1. **Evaluate policy network:**
   - Obtain policies and prices: $(k_0, \ldots, k_{H-2}, b_0, \ldots, b_{H-2}, p^b) = \text{Network}(\mathbf{x}_i^{(k-1)}; \theta^{(k-1)})$
   - Terminal generation: $k_{H-1} = b_{H-1} = 0$ (no bequest)
2. **Draw shock:** $\epsilon_i \sim \mathcal{N}(0, \sigma_Z^2)$ (can use shock multiplier > 1 for more dispersion)
3. **Compute next state:**

$$Z_{i,t+1} = \exp(\rho_Z \log Z_{i,t} + \sigma_Z \epsilon_i)$$

© Zhaoeng Fang Redistribution prohibited without permission

Step 2: Computing the Loss Function

Goal: Measure how well network satisfies equilibrium conditions

For Each State $\mathbf{x}_i$ in Cloud:

1. **Evaluate current period:**
   - Run network: get policies $(k_{h,t}, b_{h,t})$ and price p_t^b
   - Compute aggregate capital: $K_t = \sum_h k_{h,t}$
   - Compute factor prices: $r_t = \alpha Z_t K_t^{\alpha-1} L_t^{1-\alpha}$, $w_t = (1-\alpha) Z_t K_t^{\alpha} L_t^{-\alpha}$
   - Compute consumption from budget:
     $c_t^h = l_t^h w_t + b_{t-1}^{h-1} + k_{t-1}^{h-1}(1 - \delta_t + r_t) - p_t^b b_t^h - k_t^h - \psi^k (k_t^h - k_{t-1}^{h-1})^2$
2. **Simulate next period states:**
   - Draw multiple shocks: $\{\epsilon_j\}_{j=1}^{N_{\text{quad}}}$ from Gauss-Hermite quadrature (typically $N_{\text{quad}} = 5 - 11$)
   - For each shock, compute next state $\mathbf{x}_j'$ and evaluate network
   - Get next period consumption: $c_{t+1,j}^{h+1}$, returns: $R_{t+1,j,h}^k$
3. **Compute Euler equation errors:**

$$\epsilon_t^{k,h} = \psi^{FB}\left(\frac{(u')^{-1}\left(\beta \sum_j w_j u'(c_{t+1,j}^{h+1}) R_{t+1,j,h}^k\right)}{c_t^h} - 1, \frac{k_t^h}{c_t^h} \right)$$

© Zhigang Feng Redistribution Prohibited without permission

Step 2: Computing the Loss (continued)

Aggregate Loss Over Cloud:

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{cloud}}} \sum_{i=1}^{N_{\text{cloud}}} \left[\frac{1}{2} \sum_{h=0}^{H-2} (\epsilon_{i,t}^{k,h})^2 + \frac{w_b}{2} \sum_{h=0}^{H-2} (\epsilon_{i,t}^{b,h})^2 + 100 \cdot \sum_{h} (\max\{0, -c_{i,t}^{h}\})^2 \right]$$

Components:

- **Euler equation errors:** Penalize violations of optimality conditions
- **Homotopy weight w_b:** Gradually introduce bond market ($0 \to 1$)
- **Consumption non-negativity:** Large penalty for negative consumption
- **Optional pretraining term:** $w_{\text{pre}} \cdot (p^b - p^b_{\text{target}})^2$ for bond price

Why This Loss Works:

- Minimizing Euler errors $\Rightarrow$ households optimize
- Market clearing enforced by network architecture (market clearing layer)
- At $\mathcal{L} = 0$, we have found an equilibrium

© Zheng Feng • Redistribution prohibited without permission

Step 3: Updating the Network

Gradient Computation:

- Compute gradient: $\nabla_\theta \mathcal{L}(\theta^{(k-1)})$ using automatic differentiation
- JAX/PyTorch provide automatic differentiation through all operations
- Gradient flows through:
  - Network evaluation
  - Budget constraint calculations
  - Euler equation computations
  - Market clearing layer

© Zhigang Feng • Redistribution prohibited without permission

Optimization with ADAM:

ADAM update rule:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta^{(k-1)}) \quad \text{(momentum)}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2)[\nabla_\theta \mathcal{L}(\theta^{(k-1)})]^2 \quad \text{(second moment)}$$

$$\theta^{(k)} = \theta^{(k-1)} - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$$

Stabilizing Homotopy: Capital-Only Start

Problem: Training is unstable when starting from scratch with bonds

Solution: Use homotopy method to gradually introduce bonds

Stage 1: Capital-Only Economy

- Set $m_b = 0$ (mask bond policies)
- Set $w_b = 0$ (zero weight on bond Euler equation)
- Set $\underline{b} = 0$ (no borrowing allowed yet)
- Train network to solve capital accumulation only
- Bonds are shut down: $b_h = 0$ for all h

Result: Network learns to approximate capital policies and equilibrium in simpler economy

© Zhigang Feng • Redistribution prohibited without permission

Why This Helps:

- Capital-only economy is easier to solve (fewer dimensions)

Stabilizing Homotopy: Bond Pretraining

Stage 2: Bond Price Pretraining

- Keep $m_b = 0$ (bonds still masked)
- Set $w_b = 1$ (turn on bond price training)
- Add pretraining term to loss: $w_{\text{pre}} \cdot (\hat{p}^b - p^b_{\text{target}})^2$
- Target price: $p^b_{\text{target}} = \mathbb{E}[1/(1 + r_{t+1})]$ (zero-liquidity limit)

Result: Network learns correct bond pricing before bonds are actively traded

Why This Helps:

- Bond price must be consistent with capital returns when $\underline{b} = 0$
- Learning correct price first stabilizes subsequent introduction of bonds
- Avoids price explosions or crashes when bonds are introduced

Zero-Liquidity Limit: When borrowing constraint binds for all, bond and capital must have same return

© Zhiralla Feng • Redistribution prohibited without permission

Stabilizing Homotopy: Gradual Bond Introduction

Stage 3: Homotopy on Borrowing Constraint

Gradually relax borrowing constraint: $\underline{b}^{(k)} = 0 \to \underline{b}_{\text{target}}$ (e.g., -0.05)

For each homotopy step $k = 1, \ldots, K_{\text{hom}}$:

1. Set $\underline{b}^{(k)} = k/K_{\text{hom}} \cdot \underline{b}_{\text{target}}$
2. Set $m_b^{(k)} = k/K_{\text{hom}}$ (gradually unmask bond policies)
3. Train network for a few episodes
4. Continue to next homotopy step

Result: Network smoothly adapts to borrowing and bond trading

Why This Helps:

- Avoids large policy function changes
- Network adapts incrementally to new behavior
- Maintains stability throughout training

© Zhigang Feng • Redistribution prohibited without permission

Algorithm Summary: Fixed Point Structure

Conceptual View: Similar nested fixed point as Part I, but implicit

In Stationary Case (Part I):

- Outer loop: iterate on prices (r, w) until market clearing
- Inner loop: iterate on distribution λ until stationary

In ML Approach (Part II):

- No explicit loops, but same economic logic:
  1. Network learns to output policies that satisfy household Euler equations
  2. Market clearing layer enforces bond market clearing by construction
  3. Training iteration implicitly finds equilibrium prices that clear capital market
  4. Simulation ensures policies are consistent across different states
- Loss function penalizes deviations from equilibrium conditions
- Gradient descent + automatic differentiation finds the fixed point
- Homotopy provides stability by gradually approaching full equilibrium

© Zhiyang Jia | Unauthorized distribution prohibited without permission

Results: Policy Functions

Validation:

- Check Bellman residuals: should be $< 10^{-4}$
- Verify budget constraints: $c_t^h \geq 0$ always
- Simulate economy: aggregate variables should be stable
- Market clearing: $\sum_h b_h = 0$ by construction, check K_t consistent with production

Policy Function Properties:

- Capital savings k_t^h increases with own assets, decreases with aggregate capital
- Life-cycle pattern: hump-shaped accumulation
- Bond holdings b_t^h adjust to smooth consumption
- Higher TFP $\rightarrow$ more capital accumulation (precautionary savings)

Computational Cost:

- Training time: 2-4 hours on GPU (with homotopy)
- Network size: $\approx$10,000 parameters

© Zhigang Feng Redistribution prohibited without permission

Comparison: Traditional vs. ML Methods

Aspect	Traditional (Part I)	ML (Part II)
Uncertainty	Idiosyncratic only	Aggregate only
State Space	(i, a, b, e) per individual	$(Z_t, a_{1,t}, \ldots, a_{I,t})$
Distribution	Stationary λ^*	Time-varying, vector of assets by generation
Method	Grid + iteration	Neural networks
Equilibrium	Nested fixed point (explicit)	Nested fixed point (implicit via gradient descent)
Scalability	Up to $\sim$5 states	Up to $\sim$100 dimensions

© Zhigang Feng Redistribution prohibited without permission

Next Frontier: Combine both (DeepHAM for aggregate + idiosyncratic shocks)

Extensions: DeepHAM

© Zhigang Feng • Redistribution prohibited without permission

DeepHAM: Both Types of Shocks

Goal: Handle aggregate + idiosyncratic uncertainty simultaneously

Challenge:

- State space: (Z_t, Γ_t) where Γ_t is the full cross-sectional distribution
- Each Γ_i (distribution for generation i) is infinite-dimensional
- Cannot enumerate all households
- Traditional approximations (moments, histograms) may not capture enough information

DeepHAM Solution:

1. **Generalized moments:** Represent distribution via learned statistics

$$\Gamma \approx \left\{ \frac{1}{N} \sum_i Q_1(a_i, y_i), \ldots, \frac{1}{N} \sum_i Q_J(a_i, y_i) \right\}$$

   where $Q_j(\cdot)$ are neural sub-networks that learn which moments matter
2. **Fictitious play:** Iterate between individual optimization and aggregate consistency

© Zhigang Feng
Redistribution prohibited without permission

DeepHAM: Key Ideas

Representation of Distribution:

- Instead of tracking full distribution $\Gamma_t(a, y)$, learn low-dimensional summary
- Use neural networks to define moment functions $Q_j(a, y)$
- State becomes: $(Z_t, m_1, \ldots, m_J)$ where $m_j = \mathbb{E}[Q_j(a, y)]$
- Network learns which moments are relevant for equilibrium

Training Algorithm:

1. Simulate panel of heterogeneous agents
2. Each agent optimizes taking aggregate state as given (fictitious play)
3. Update value function using supervised learning on realized utility
4. Update moment networks to better predict aggregate outcomes
5. Iterate until convergence

© Zhirang Feng. Redistribution prohibited without permission

Advantages:

- Handles both types of uncertainty

Conclusions

© Zhigang Feng • Redistribution prohibited without permission

Summary

Part I: Idiosyncratic Shocks Only

- Work with stationary distribution λ^*
- Nested fixed point: prices (outer) and distribution (inner)
- Traditional methods: grid + iteration
- Feasible for $\sim$300,000 nodes, $<$1 hour computation

Part II: Aggregate Shocks Only

- Distribution $(a_{1,t}, \ldots, a_{I,t})$ becomes state variable
- High-dimensional: 80+ continuous states
- ML solution: Neural networks + market clearing layer + homotopy
- Handles much higher dimensions than traditional methods

Extension: DeepHAM

- Combines both types of shocks

© Zhigalov Feed ... redistribution prohibited without permission

Key Computational Insights

1. **Distribution vs. State Variable:**
   - No aggregate shocks → stationary distribution suffices
   - Aggregate shocks + no idiosyncratic → track vector of assets by generation
   - Both shocks → track full distribution (infinite-dimensional)

2. **Nested Fixed Points:**
   - Equilibrium requires consistency at two levels: prices and distribution
   - Traditional: explicit outer/inner loops
   - ML: implicit via gradient descent on loss function

3. **Dimensionality:**
   - Idiosyncratic shocks: infinitely many agents, finite state space per agent
   - Aggregate shocks: finite agents per generation, high-dimensional aggregate state
   - Both: infinite agents and infinite-dimensional distribution

4. **Machine Learning Benefits:**
   - Universal function approximation without grids
   - Automatic differentiation through equilibrium
   - Scalable to high dimensions with proper initialization (homotopy)

© Zhiguang Feng Redistribution prohibited without permission

Practical Advice

For Traditional Methods:

- Start with coarse grid, refine gradually
- Monitor K/L ratio (more stable than prices)
- Use damping ($\eta \approx 0.6$) in price updates
- Bilinear interpolation for distributions (preserves measure)

For ML Methods:

- Start simple: fewer generations, fewer shock states
- Use homotopy: adjust one parameter at a time
- Validate: check Bellman residuals, budget constraints
- Monitor training: loss should decrease smoothly
- Experiment with network architecture (depth, width)

Software:

© Zdeněk Fiala – Redistribution prohibited without permission

References

Traditional Methods:

- Nishiyama & Smetters (2013): Handbook of Computational Economics, Vol. 3
- Auerbach & Kotlikoff (1987): Dynamic Fiscal Policy
- Huggett (1993): Risk-free rate in incomplete markets
- Aiyagari (1994): Uninsured idiosyncratic risk and saving

Machine Learning Approaches:

- Azinovic, Gaegauf & Scheidegger (2022): Deep Equilibrium Nets, IERE
- Zemlicka & Azinovic (2023): Economics-Inspired NNs with Homotopies
- Feng, Han, Liu & Lu (2025): DeepHAM
- Maliar, Maliar & Winant (2021): Deep learning for HA models

Code:

- Homotopy notebook: `https://arxiv.org/abs/2303.14802`
- Nishiyama-Smetters replication: CBO website

© Zhigang Feng Re-distribution prohibited without permission