Lecture Overview

- **Aiyagari–Bewley–Huggett Model:**
  - Existence and uniqueness of recursive stationary distributions
- **Solving Stationary Distributions:**
  - Methods and practical computation issues
- **Global Solution Approaches via ML and RL:**
  - Moving beyond stationary distributions, c.f. Feng et. al. (2025)
- **Krusell–Smith Model via DeepHAM:**
  - Application of advanced machine learning techniques to solve heterogeneous-agent models
- **Deep Equilibrium Nets (DEQN):**
  - Euler-equation-based approach with cloud simulation

Aiyagari-Bewley-Hugget Model

Aiyagari (1994, QJE): Heterogeneity among Households

- Continuum of infinitely lived households with total measure 1.
- Households are subject to uninsurable income shocks.
- Households are *ex-ante* identical, i.e. before income shocks are realized.
- Households differ *ex-post*, i.e. after income shocks have materialized, with respect to asset holdings, depending on the sequence of income shocks.
- Incomplete markets models endogenously generate wealth inequality and social mobility.

General Equilibrium: Ex-Post Heterogeneity

- Income shocks ϵ are idiosyncratic, i.e. *i.i.d.* across *households* (but not necessarily iid across *time*).
- Aggregate income is assumed to be constant.
- Households can trade a risk-free one-period bond a at market price.
- Labor supply is inelastic: income shocks are interpreted as shocks to labor productivity.
- Shocks follow Markov chain with transition matrix $\pi(\epsilon, \epsilon')$.

Household's Problem

- Recursive formulation:

$$v(a,\epsilon;\lambda) = \max_{c,a'}\{u(c) + \beta \sum_{\epsilon' \in E} v(a',\epsilon';\lambda')\pi(\epsilon,\epsilon')\}$$

s.t.

$$\begin{aligned} c + a' &= (1 + r(\lambda))a + w(\lambda)\epsilon \\ a' &\geq -\underline{a} \end{aligned}$$

where $r \equiv \tilde{r} - \delta$, ϵ are shocks to labor productivtiy and $\lambda(a,\epsilon)$ is the distribution of households across states

Budget Constraint

- Assets a are traded at price 1 today with return $1 + r$
- Assets contain claims to physical production capital
- Interest rate is determined by the marginal product of capital
- Timing assumption:
  - (i) production takes place,
  - (ii) income is distributed,
  - (iii) households consume
- If r is stationary over time, this is equivalent to specification given above (zero-bond)

Firms

- Competitive firms that produce a homogenous output good
- Firms act as price takers
- Production technology has constant returns to scale
- Hence, firm size is indeterminate: we consider a *single, representative firm*

Stationary Recursive Competitive Equilibrium

Stationarity

- We want prices r and w to be time-invariant
- This requires the distribution of households across states to be invariant: the probability distribution will permanently reproduce itself
- $\lambda(a, \epsilon) = \lambda'(a, \epsilon) = \lambda^*(a, \epsilon)$

A **Stationary Recursive Competitive Equilibrium (SRCE)** is a value function $v(a, \epsilon)$, policy functions $a'(a, \epsilon)$ and $c(a, \epsilon)$, a probability distribution $\lambda^*(a, \epsilon)$, and positive real numbers (K, L, r, w) such that

1. The prices (w, r) satisfy

$$\begin{aligned} w &= F_L(K, L) \\ r &= F_K(K, L) - \delta \end{aligned}$$

   where $F(K, N) = AK^{\alpha}L^{1-\alpha}$
2. Given r, w, the optimal policy functions $a'(a, \epsilon)$ and $c(a, \epsilon)$ solve the household's problem and v is the associated value function
3. The labor market clears: $L = \sum_{a,\epsilon} \epsilon \lambda^*(a, \epsilon)$
4. The capital market clears: $K = \sum_{a,\epsilon} a'(a, \epsilon)\lambda^*(a, \epsilon)$
5. The probability distribution $\lambda^*(a, \epsilon)$ is a stationary distribution associated with $a'(a, \epsilon)$ and $\pi(\epsilon, \epsilon')$; that is, it satisfies

$$\lambda^*(a', \epsilon') = \sum_{\epsilon} \sum_{a:a'=a'(a,\epsilon)} \lambda^*(a, \epsilon)\pi(\epsilon, \epsilon')$$

Existence and Uniqueness of SRCE

Existence and Uniqueness of SRCE

- General equilibrium requires market clearing in two markets: capital and labor (goods market redundant by Walras' law)
- Two prices: r and w
- Because labor supply is exogenous, w is a function of r only
- General equilibrium is a function of r only

Determination of r: Demand for Capital

- $K(r)$: capital demand of the firm
- Implicitly defined such that marginal product equals interest rate:

$$r = F_K(K, L) - \delta$$

- From Inada conditions on F, $\lim_{r \to -\delta} K(r) = \infty$ and $\lim_{r \to \infty} K(r) = 0$
- $K(r)$ is continuous

Determination of r: Supply of Capital

- $A(r)$: capital supply of households:

$$A(r) = \sum_{a,\epsilon} a'(a, \epsilon; r)\lambda^*(a, \epsilon; r)$$

- We know that $\lim_{r \to \frac{1}{\beta} - 1} A(r) = \infty$ and $\lim_{r \to -\delta} A(r) < \infty$
- If $A(r)$ is continuous, an equilibrium with $r \in (-\delta, \frac{1}{\beta} - 1)$ exists

Is $A(r)$ Continuous?

- $a'(a, \epsilon; r)$ is continuous in r by Corollary 6.1 in Acemoglu (2009). Also see Proposition 17.5 in Acemoglu (2009).
- $\lambda^*(a, \epsilon; r)$ is continuous in r by Theorem 12.13 in Stokey/Lucas (1989)
- Necessary pre-requisite: Existence and uniqueness of invariant probability distribution $\lambda^*(a, \epsilon; r)$

Probability Measure

- We are interested under which condition a unique stationary probability distribution $\lambda^*(a', \epsilon')$ exists
- Let $Q : \lambda \to \lambda'$ be transition function of λ:

$$\lambda'(a', \epsilon') = \underbrace{\sum_{\epsilon} \sum_{a : a' = a'(a, \epsilon)} \pi(\epsilon, \epsilon')\, \lambda(a, \epsilon)}_{\equiv Q}$$

Probability Measure

We require the Markov transition operator Q to satisfy three properties:

1. *Feller property*, for **existence** of an invariant measure,
2. *Mixing property*, for **uniqueness** of the invariant measure,
3. *Monotonicity*, for **convergence** to that invariant measure.

Result (Stokey & Lucas, 1989, Thm. 12.12)
If Q satisfies these conditions, there exists a unique invariant probability measure λ^*, and $Q^t(\lambda^0) \to \lambda^*$ for *any* initial measure λ^0.

Feller Property (Existence)

- The Feller property requires that Q sends continuous and bounded functions into continuous and bounded functions.
- In our setting, this is ensured if the policy rule $a' = a'(a, \epsilon)$ is continuous in both a and ϵ and remains bounded (the latter follows if $\beta(1 + r) < 1$, preventing asset levels from exploding).
- **Intuition:** The Feller property guarantees that Q is a "well-behaved" transition operator on the space of continuous functions, which is a key requirement for establishing the *existence* of an invariant measure.

Mixing Property (Uniqueness)

- Also referred to as the "American Dream, American Nightmare" condition. It stipulates that there is a positive probability of transitioning from the lowest state $(\underline{a}, \underline{\epsilon})$ to a higher state in a finite number of periods ("American Dream") *and* a positive probability of transitioning from the highest state $(\bar{a}, \bar{\epsilon})$ to a lower state ("American Nightmare").
- **Key idea:** No part of the state space is "isolated" forever. This prevents the chain from becoming trapped in extreme states, thereby ruling out multiple invariant measures.

Mixing Property (Intuition)

- In many macroeconomic models, income shocks ϵ are transitory and mean-reverting, so households do not stay in extreme states forever.
- Starting at $(\underline{a}, \underline{\epsilon})$, there is always some positive probability of drawing a long sequence of high income shocks, pushing savings (and thus assets) upward. Conversely, from $(\bar{a}, \bar{\epsilon})$, a sequence of negative shocks can drive assets down.
- Because no state is truly absorbing, the transition dynamics "mix" the distribution, forcing it toward a *unique* invariant measure.

Mixing Property: Counterexample

- If mixing fails, multiple invariant measures can arise. For instance, consider the transition matrix

$$Q = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

- Here, each state is absorbing. Whichever state you start in, you stay there.
- As a result, *any* probability distribution over the states is invariant, so uniqueness is lost.

Monotonicity (Convergence)

- The monotonicity condition requires that higher initial measures lead to higher measures after applying Q. Equivalently, the transition dynamics are "order-preserving" or exhibit "positive autocorrelation."
- Concretely, if $a'(a, \epsilon)$ is increasing in both a and ϵ, and the Markov chain on ϵ also preserves order, then Q is monotone.
- **Convergence intuition:** If Q is monotone, successive iterations of Q on any initial measure cannot "oscillate away" from an equilibrium distribution. This ensures convergence to the unique λ^*.

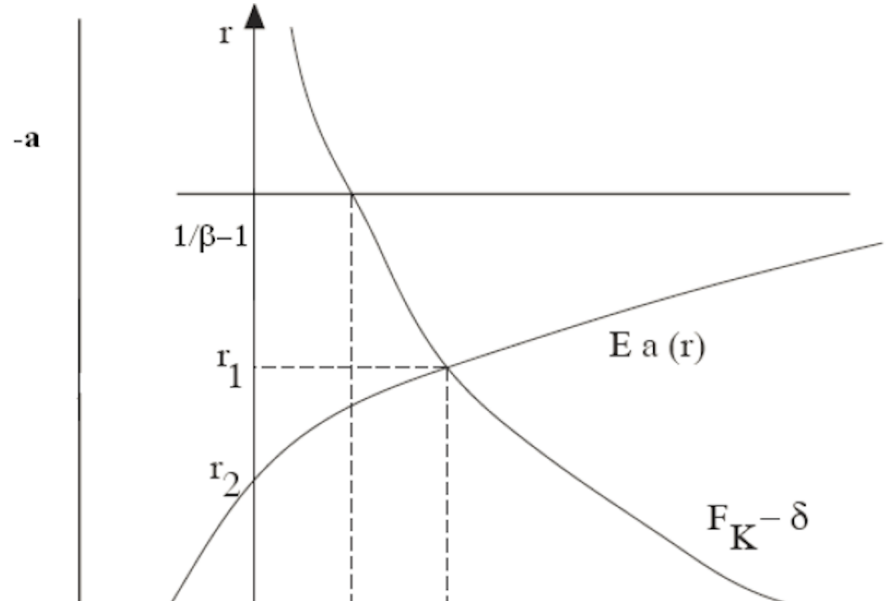
Monotonicity: Counterexample

- If monotonicity fails, one can get non-convergent dynamics. For example, consider the matrix:

$$Q = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

- This transition simply flips the state back and forth each period.
- Because of this persistent oscillation, there is no single long-run distribution to which the chain converges.

A Possible Equilibrium



Aggregate Precautionary Savings

- Compare aggregate capital stock in the economy with earnings uncertainty (K_1) to economy without earnings uncertainty/complete markets (K_0)
- K_0 corresponds to the steady-state capital stock of the representative agent economy, where $\beta(1+r)=1$
- Uninsurable idiosyncratic shocks boosts aggregate savings and lowers r such that $\beta(1+r)<1$

Uniqueness

- Huggett (1997) shows that *any* steady-state capital stock in the neoclassical growth model with uninsurable labor endowment shocks lies above its deterministic counterpart
- So, r will be always such that $\beta(1+r) < 1$
- However, we do not know whether there is a unique r^* that clears the market
- This is due to the unknown relative strength of substitution/income effect: sign of $A_r(r)$ is unclear

Applications

Models with incomplete markets are used to understand

- Emergence of wealth inequality
- Quantitative importance of precautionary savings for aggregate capital stock
- Asset pricing
- Welfare implications of earnings uncertainty

See Heathcote, Storesletten and Violante (2009, Annual Review of Economics) and Guvenen (2011, NBER WP 17622)

Computation of SRCE

Computation

Involves three steps:

- Fix an $r \in \left(-\delta, \frac{1}{\beta} - 1\right)$. For a fixed r, solve household's recursive problem. This yields a value function v_r and decision rules a'_r,c_r.
- The policy function a'_r and π induce Markov transition function Q_r . Compute the unique stationary measure λ_r associated with this transition function.
- Compute excess demand for capital

$$d(r) = K(r) - \int a(r)$$

  If zero, stop, if not, adjust r.

Overview: Solving the Canonical Aiyagari Model

- **Step 1:** For a given interest rate r, solve the household's recursive problem to obtain policy functions $a'_r(a, \epsilon)$ and $c_r(a, \epsilon)$.
- **Step 2:** Use the policy function and the idiosyncratic income process $\pi(\epsilon, \epsilon')$ to compute the Markov transition function Q_r and the unique stationary distribution $\lambda_r(a, \epsilon)$.
- **Step 3:** Compute aggregate capital $K(r)$ and the excess demand $d(r)$, and adjust r until the capital market clears.

Step 1: Solving the Household Problem

- **Fix an interest rate:** Choose an $r \in \left(-\delta, \frac{1}{\beta} - 1\right)$.
- **Recursive formulation:**

$$v(a, \epsilon) = \max_{c, a'} \left\{ u(c) + \beta \sum_{\epsilon' \in E} v(a', \epsilon') \pi(\epsilon, \epsilon') \right\}$$

subject to:

$$c + a' = (1 + r)a + w\epsilon, \quad a' \geq -\underline{a}.$$

- **Outcome:** Obtain the value function v_r and the policy functions $a'_r(a, \epsilon)$ and $c_r(a, \epsilon)$.

Step 2: Computing the Stationary Distribution

- The optimal policy $a'_r(a, \epsilon)$ and the income process $\pi(\epsilon, \epsilon')$ induce a Markov transition function Q_r.
- **Stationary distribution:** Find $\lambda_r(a, \epsilon)$ satisfying:

$$\lambda_r(a', \epsilon') = \sum_{\epsilon \in E} \sum_{a:\, a'_r(a,\epsilon) = a'} \lambda_r(a, \epsilon) \pi(\epsilon, \epsilon').$$

- This distribution represents the long-run fraction of households in each state (a, ϵ).

Step 3: Market Clearing and Adjusting the Interest Rate

- **Aggregate capital:** Compute

$$K(r) = \sum_{a,\epsilon} a'_r(a,\epsilon)\, \lambda_r(a,\epsilon).$$

- **Excess demand:** Define

$$d(r) = K(r) - \int a\, d\lambda_r.$$

- **Adjustment:** Update r (using methods like bisection or fixed-point iteration) until $d(r) = 0$, so that the capital market clears.
- Once r is determined, compute the equilibrium wage via:

$$w = F_L(K, L), \quad r = F_K(K, L) - \delta,$$

where $F(K, L) = AK^{\alpha}L^{1-\alpha}$.

Algorithm Summary

1. **Initialize:** Choose an initial guess for r, say $r^{(0)}$.
2. **Iteration:** For each $r^{(i)}$:
   - 2.1 Solve the household's recursive problem to obtain a'_r and c_r.
   - 2.2 Compute the stationary distribution λ_r from the induced Markov transition function.
   - 2.3 Evaluate the aggregate capital $K(r^{(i)})$ and excess demand $d(r^{(i)})$.
   - 2.4 Adjust r to reduce $d(r^{(i)})$, setting $r^{(i+1)}$.
3. **Convergence:** Repeat until $d(r)$ is sufficiently close to zero.
4. **Equilibrium:** Output the equilibrium policy functions, stationary distribution, and prices (r, w).

Solving the Household's Recursive Problem

- Any valid solution method for recursive problems (e.g., value function iteration, projection methods, etc.) is acceptable.
- However, computational speed is critical.
- Plain value function iteration—without enhancements—may be too slow.
- Use standard refinements such as exploiting monotonicity and concavity.
- Make intelligent use of initial guesses to accelerate convergence.
- Calibrate by fixing variable values in the steady state rather than parameters.
- Consider multigrid schemes to improve efficiency.

Computing the Unique Stationary Measure

- Let $A = \{a_1, \ldots, a_M\}$ denote the discretized grid for assets and assume there are N discrete values for the idiosyncratic shock ϵ.
- The stationary measure λ is represented as an $M \times N \times 1$ column vector.
- Define the transition matrix

$$Q = (q_{i,j;k,l}),$$

where

$$q_{i,j;k,l} = \Pr\Big[(a', \epsilon') = (a_k, \epsilon_l) \mid (a, \epsilon) = (a_i, \epsilon_j)\Big].$$

- The stationary measure satisfies the matrix equation

$$\lambda = Q^T \lambda.$$

- Hence, λ is an eigenvector of Q^T associated with the eigenvalue 1.
- Since Q^T is a stochastic matrix, it always has at least one unit eigenvalue. If more than one exists, a continuum of stationary measures arises.

Transitional Dynamics: Motivation

- So far, we focused on a *stationary distribution*, which is the long-run, time-invariant steady state of the model.
- In many applications, we are also interested in how the economy adjusts *away* from one steady state to another after a parameter or policy change. This is what we call *transitional dynamics*.
- **Illustration:** Suppose a permanent capital income tax τ is introduced, with lump-sum rebates T. We want both:
  1. The new steady state after the tax, and
  2. The entire adjustment path (transitional dynamics) from the old to the new steady state.

Stationary Equilibria and Transitions

- A *stationary equilibrium* is one in which all time-indexed objects (e.g., prices, distributions) remain constant over time.
- In reality, transitions are often *asymptotic* and may not complete in a finite number of periods.
- A common *compromise* in computational work is to assume that after T periods, the economy has converged sufficiently to the new steady state. Then we set $v^T = v^\infty$ (value functions), and *solve backwards* given an assumed price path $\{r_t, w_t\}_{t=1}^T$.
- Later, we will discuss a *global solution* approach that does *not* rely on the assumption of completing the transition exactly in T periods. Instead, the equilibrium path *endogenously* emerges, and we track the economy's evolution without imposing artificial time cutoffs.

Computation of transitions

- Fix T.
- Compute stationary equilibrium $\lambda_0, v_0, r_0, w_0, K_0$ associated with $\tau_0 = 0$.
- Compute stationary equilibrium $\lambda_\infty, v_\infty, r_\infty, w_\infty, K_\infty$ associated with $\tau_\infty = \tau$. Assume that

$$\{\lambda_T, v_T, r_T, w_T, K_T\} = \{\lambda_\infty, v_\infty, r_\infty, w_\infty, K_\infty\}$$

- Guess sequence of capital stocks $\left\{\hat{K}_t\right\}_{t=1}^{T}$. The capital stock at time $t = 1$ is determined by decisions at time 0, $\hat{K}_1 = K_0$. Note that $L_t = L_0 = L$ is fixed. We also obtain

$$\begin{aligned}
\hat{w}_t &= F_L(\hat{K}_t, L) \\
\hat{r}_t &= F_K(\hat{K}_t, L) \\
\hat{T}_t &= \tau_t \hat{r}_t \hat{K}_t
\end{aligned}$$

- We need also guess sequence of labor supply $\left\{\hat{L}_t\right\}_{t=1}^{T}$ if there is labor-leisure choice.

Computation of transitions

- Since we know $v(a, \epsilon)$ and $\left\{\hat{r}_t, \hat{w}_t, \hat{T}_t\right\}_{t=1}^{T-1}$, we can solve for $\{\hat{v}_t, \hat{c}_t, \hat{a}_{t+1}\}_{t=1}^{T-1}$ backwards.
- With policy functions $\{\hat{a}_{t+1}\}$ define transition laws $\left\{\hat{\Gamma}_t\right\}_{t=1}^{T-1}$. We know $\lambda_0 = \lambda_1$ from the initial stationary equilibrium. Iterate the distributions forward

$$\hat{\lambda}_{t+1} = \hat{\Gamma}_t(\hat{\lambda}_t)$$

  for $t = 1, ..., T-1$.
- With $\left\{\hat{\lambda}_t\right\}_{t=1}^{T}$ we can compute $\hat{A}_t = \int a d\hat{\lambda}_t$ for $t = 1, ..., T$.

Computation of transitions

- If

$$\max_{1<t<T} |\hat{A}_t - \hat{K}_t| < tol_A$$

  go to the next step. If not, adjust your guesses for $\left\{\hat{K}_t\right\}_{t=1}^{T-1}$.
- If

$$\| \hat{\lambda}_T - \lambda_T \| < tol_\lambda$$

  the transition converges smoothly into the new steady state and we are done and should save

$$\left\{\hat{v}_t, \hat{a}_{t+1}, \hat{c}_t, \hat{\lambda}_t, \hat{r}_t, \hat{w}_t, \hat{K}_t\right\}$$

  If not, increase T.
- We can be smart with the initial guess: compute associated RA transition.

Welfare analysis

- This procedure determines aggregate variables such as $\{r_t, w_t, \lambda_t, K_t\}$ and individual decision rules $\{c_t, a_{t+1}\}$.
- The value functions enable us to make statements about the welfare consequences of the tax reform.
- We have value functions $\{v_t\}_{t=0}^{T}$.
- We can use v_0, v_1 and v_T to determine the welfare consequences from the reform.

Consumption equivalent variation

- Suppose that $U(c) = \frac{c^{1-\sigma}}{1-\sigma}$.
- Optimal consumption allocation in initial stationary equilibrium, in sequential formulation, $\{c_s\}_{s=0}^{\infty}$:

$$v_0(a, \epsilon) = \mathbb{E}_0 \sum_{s=0}^{\infty} \beta^s \frac{c_s^{1-\sigma}}{1-\sigma}$$

- If increase consumption in each date, in each state, in the old stationary equilibrium, by a fraction g. Then

$$\begin{aligned} v_0(a, \epsilon; g) &= \mathbb{E}_0 \sum_{s=0}^{\infty} \beta^s \frac{((1+g)c_s)^{1-\sigma}}{1-\sigma} \\ &= (1+g)^{1-\sigma} v_0(a, y) \end{aligned}$$

- By what percent g do we have to increase consumption in the old stationary equilibrium for agent to be indifferent between old stationary equilibrium and transition induced by policy reform?

Consumption equivalent variation

- This percent g solves

$$v_0(a, \epsilon; g) = v_1(a, \epsilon)$$

- Note that $v_1(a, \epsilon)$ has considered the transition dynamics and new stationary equilibrium.
- This yields

$$g(a, \epsilon) = \left[\frac{v_1(a, \epsilon)}{v_0(a, \epsilon)}\right]^{\frac{1}{1-\sigma}} - 1$$

  which varies with idiosyncratic shocks.

- Similarly, we can compute the welfare gain/loss in steady state.

$$g_{ss}(a, \epsilon) = \left[\frac{v_T(a, \epsilon)}{v_0(a, \epsilon)}\right]^{\frac{1}{1-\sigma}} - 1$$

A Simple Example to Understand How ML Helps

Why Approximate the Value Function?

- In many dynamic equilibrium models (e.g., the optimal growth model), we need to solve

$$V(k) \;=\; \max_{g(k)} \Big\{\, u\big(k, g(k)\big) \;+\; \beta\, V\big(f(k, g(k))\big) \Big\},$$

  where
  - k is the state (e.g., capital),
  - $g(k)$ is the policy (e.g., investment),
  - u is the utility function (e.g., $\log$ utility),
  - β is the discount factor.
- Traditional methods (Chebyshev polynomials, finite elements, grid-based collocation) approximate $V(k)$ by discretizing the state space, but often suffer from the **curse of dimensionality**.
- **Deep Neural Networks (DNNs)** can mitigate this by learning high-dimensional approximations more efficiently, given suitable training data (states and associated values).

Approximating the Value and Policy Functions with DNNs

- We let
$$V(k) \approx \Gamma_{\gamma_v}(k), \quad g(k) \approx \Gamma_{\gamma_g}(k),$$
where γ_v and γ_g are neural network parameters (weights and biases).
- The idea is analogous to value function iteration:
  1. Generate states $k \sim d(k)$ in the state space (where d is some distribution or weighting measure).
  2. Compute approximate target values or gradients for training:
$$\hat{V}(k) \quad \text{or} \quad \nabla_{\gamma_g} \left\{ u(k, \Gamma_{\gamma_g}(k)) + \beta \, \Gamma_{\gamma_v}[f(k, \Gamma_{\gamma_g}(k))] \right\}.$$
  3. Train Γ_{γ_v} or Γ_{γ_g} using standard gradient-based methods to solve the relevant expectation-based objectives.
- This framework naturally handles high-dimensional k.

Value Function Update (One-Step Look-Ahead)

- Fix a policy $g^{(n-1)}(k) = \Gamma_{\gamma_g}^{(n-1)}(k)$.
- The one-step approximation for the value is:

$$\hat{V}(k) \;=\; u\big(k, g^{(n-1)}(k)\big) \;+\; \beta\, \Gamma_{\gamma_v}^{(n-1)}\big(f(k, g^{(n-1)}(k))\big).$$

  - We are using $T_{sim} = 1$ look-ahead for simplicity: immediate payoff plus approximate next-state value.

- We then solve an *expectation-based* least squares problem:

$$\min_{\gamma_v^{(n)}} \; \mathbb{E}_{k\sim d(k)}\Big[\big(\Gamma_{\gamma_v}^{(n)}(k) - \hat{V}(k)\big)^2\Big],$$

  where $d(k)$ is the chosen state distribution (could be uniform, stationary, etc.).

- In practice, this expectation is approximated by sampling states $\{k_i\}_{i=1}^{N_v}$ from $d(k)$:

$$\min_{\gamma_v^{(n)}} \; \frac{1}{N_v} \sum_{i=1}^{N_v} \Big(\Gamma_{\gamma_v}^{(n)}(k_i) - \hat{V}(k_i)\Big)^2.$$

Policy Function Update (One-Step Look-Ahead)

- Given $\Gamma_{\gamma_v}^{(n)}$, we improve the policy by solving:

$$\max_{\gamma_g^{(n)}} \ \mathbb{E}_{k\sim d(k)}\Big[u\big(k,\Gamma_{\gamma_g}^{(n)}(k)\big)+\beta\,\Gamma_{\gamma_v}^{(n)}\big(f(k,\Gamma_{\gamma_g}^{(n)}(k))\big)\Big].$$

  - Again, $T_{sim}=1$ for simplicity.
  - We typically solve this via gradient ascent methods applied to the neural net parameters $\gamma_g^{(n)}$.

- Approximating by sample average:

$$\max_{\gamma_g^{(n)}} \ \frac{1}{N_g}\sum_{i=1}^{N_g}\Big[u\big(k_i,\Gamma_{\gamma_g}^{(n)}(k_i)\big)+\beta\,\Gamma_{\gamma_v}^{(n)}\big(f(k_i,\Gamma_{\gamma_g}^{(n)}(k_i))\big)\Big],$$

  where $\{k_i\}_{i=1}^{N_g}\sim d(k)$.

Choosing the Distribution $d(k)$ and Weights

- In the above, $\mathbb{E}_{k\sim d(k)}$ indicates an expectation with respect to the distribution or weighting measure $d(k)$.
- Common choices:
  - **Uniform distribution** on a bounded interval (simple, but may be inefficient for large state spaces).
  - **Stationary distribution** induced by an approximate policy (focuses on states the system is likely to visit).
  - **Adaptive or importance sampling** strategies to focus on high-value or high-uncertainty regions.
- In practice, we often approximate the integral by a sample average:

$$\mathbb{E}_{k\sim d(k)}[F(k)] \;\approx\; \sum_{i=1}^{N} w_i F(k_i), \quad \text{where} \sum_i w_i = 1.$$

- Weights w_i may be uniform ($w_i = 1/N$) or adjusted to account for different sampling schemes.

Bias-Variance Tradeoff for $T_{sim} = 1$

- $\mathbf{T_{sim} = 1}$ approach:

$$\hat{V}(k_i) \;\approx\; u\big(k_i, g^{(n-1)}(k_i)\big) + \beta\,\Gamma_{\gamma_v}^{(n-1)}\big(f(k_i, g^{(n-1)}(k_i))\big).$$

- **Pros:**
  - Easy to implement and fast to sample/compute.
  - Lower *variance* in target values $\hat{V}(k_i)$.
- **Cons:**
  - Higher *bias*: we rely heavily on the *previous* value function approximation for future returns.
  - May converge more slowly or to a suboptimal policy if the approximation is poor.
- A natural extension is to use $\mathbf{T_{sim}}$-step or multi-period look-ahead to reduce bias. This will, however, increase variance in the training targets.

Improvement 1: Focusing on the Stationary Distribution (1/2)

- **Motivation:** Uniformly sampling $\{k_i\}$ over a large domain can be highly inefficient:
  - Many sampled states may be very unlikely to occur in practice.
  - This wastes model capacity on unimportant regions of the state space.
- **Stationary Distribution Approach:**
  - *Simulate* the economy under the current or near-optimal policy for many periods.
  - Wait until the capital stock (or other state variables) converges to a *stationary distribution*.
  - *Draw training points* $\{k_i\}$ from that distribution.
- **Benefits**:
  - *Sample efficiency:* We focus on the part of the state space that truly matters for long-run outcomes.
  - Often converges faster because the DNN learns to approximate $V(\cdot)$ and $g(\cdot)$ well in the most relevant regions.
  - *Law of Large Numbers:* By simulating long enough, the empirical distribution of states stabilizes, giving reliable samples without needing to handcraft a grid.

Improvement 1: Focusing on the Stationary Distribution (2/2)

- **Why not Chebyshev or other grid-based methods here?**
  - Chebyshev and finite-element methods typically *must* approximate over the entire domain.
  - As dimensionality grows, covering the entire domain with a fine grid becomes infeasible (*curse of dimensionality*).
  - In contrast, stationarity-based sampling zooms in on the *likely* region of interest.
- **Limitations:**
  - *Rare but important events* (e.g., extreme shocks or tail risks) may be under-represented or entirely missed if they have low probability but high impact.
  - If the system has multiple steady states or complex dynamics, a single stationary distribution might not capture important transient states.
- **Possible Remedies:**
  - Combine stationary distribution sampling with targeted sampling of rare states or shocks (sometimes called *importance sampling*).
  - Conduct stress tests or out-of-distribution sampling separately to ensure the DNN's performance does not degrade drastically in rare scenarios.

Improvement 2: Extending to T_{sim}-Step Returns (1/3)

- **Recall One-Step Return:**

$$\hat{V}(k_i) \;=\; u\big(k_i, g^{(n-1)}(k_i)\big) \;+\; \beta\,\Gamma_{\gamma_v}^{(n-1)}\big(f(k_i, g^{(n-1)}(k_i))\big).$$

  $\Rightarrow$ *Fast* but *biased* if $V(\cdot)$ is inaccurate.

- T_{sim}**-Step Return Idea:**

$$\hat{V}(k_i) \;=\; \sum_{t=0}^{T_{sim}-1} \beta^t\, u\big(k_{i,t}, g^{(n-1)}(k_{i,t})\big) \;+\; \beta^{T_{sim}}\,\Gamma_{\gamma_v}^{(n-1)}\big(k_{i,n}\big),$$

  where $k_{i,t+1} = f\big(k_{i,t}, g^{(n-1)}(k_{i,t})\big)$.

- **Pros & Cons**:
  - $+$ *Lower bias*: uses actual simulated rewards over multiple steps.
  - $-$ *Higher variance*: the simulated path can vary significantly, especially under uncertainty.
- As $n \to \infty$, we approximate the infinite horizon directly. But in practice, large n can be costly and noisy.

Improvement 2: Extending to T_{sim}-Step Returns (2/3)

- **Policy Function Update with T_{sim}-Step:**
  - Similarly, when *improving* the policy, we could maximize the n-step return plus the value function at the end of the n-step path:

$$\max_{\gamma_g^{(n)}} \sum_{i=1}^{N_g} \left[\sum_{t=0}^{T_{sim}-1} \beta^t u\big(k_{i,t}, \Gamma_{\gamma_g}^{(n)}(k_{i,t})\big) \; + \; \beta^{T_{sim}} \Gamma_{\gamma_v}^{(n)}(k_{i,n}) \right].$$

  - This can potentially improve policy more significantly each iteration but also introduces more variability in gradients.

Improvement 2: Extending to T_{sim}-Step Returns (3/3)

- **Uncertainty and the Role of Monte Carlo:**
  - When the model includes stochastic elements or shocks, the future state evolves randomly.
  - Enumerating all possible future states or integrating over a high-dimensional shock space quickly becomes infeasible (another form of the curse of dimensionality).
  - *Monte Carlo simulation* circumvents this by generating a large sample of *realized* state-and-shock paths, and computing empirical averages to approximate expectations.
  - By drawing sufficiently many simulated paths $(k_{i,t}, \omega_{i,t}, \ldots)$, we can estimate the expected returns or value function without explicitly dealing with the full high-dimensional distribution.
  - DNNs then approximate the underlying functions (value or policy) from these simulated data points, relying on the law of large numbers for consistency.
- **In Practice**:
  - Choose T_{sim} to balance bias vs. variance.
  - Use parallel or GPU-based Monte Carlo to handle large simulation demands.
  - Combine with importance sampling if rare but influential shocks are of interest.

Algorithm Summary

1. **Initialize:**
$$\Gamma_{\gamma_g}^{(0)},\ \Gamma_{\gamma_v}^{(0)}.$$
2. **Policy Evaluation (Value Function Update):**
   - Generate training data $\left\{k_i, \hat{V}(k_i)\right\}$ using either:
     - **One-step** approach: $T_{sim} = 1$,
     - or a $\mathbf{T}_{\text{sim}}$-period Monte Carlo simulation for lower bias.
   - Solve
$$\min_{\gamma_v^{(n)}} \sum_i \left[\Gamma_{\gamma_v}^{(n)}(k_i) - \hat{V}(k_i)\right]^2.$$
3. **Policy Improvement (Policy Function Update):**
   - Generate states $\{k_i\}$ (e.g., from the stationary distribution).
   - Solve
$$\max_{\gamma_g^{(n)}} \sum_i \left[u(k_i, \Gamma_{\gamma_g}^{(n)}(k_i)) + \beta\, \Gamma_{\gamma_v}^{(n)}(f(k_i, \Gamma_{\gamma_g}^{(n)}(k_i)))\right].$$
4. **Check convergence** and repeat until Γ_{γ_g} and Γ_{γ_v} stabilize.

Euler Equation Based + ML

Euler Equation

- Capital accumulation: $k' = g(k)$, consumption: $c = f(k) + (1-\delta)k - g(k)$.
- For CRRA utility $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$ and production $f(k) = k^{\alpha} + (1-\delta)k$, the equilibrium satisfies

$$u'(c) \;=\; \beta\, u'\big(c'\big)\,\Big[\alpha\, g(k)^{\,\alpha-1} + 1 - \delta\Big]. \tag{EE}$$

- Goal: learn a policy g that makes the Euler residual
  $\mathcal{R}(k\,;\,g) \;=\; u'(c) - \beta\, u'(c')\big[\alpha g(k)^{\alpha-1} + 1 - \delta\big]$ vanish for all economically relevant k.

From Euler Residual to Loss Function

- Approximate the policy by a neural net $g(k) \approx \Gamma_{\gamma_g}(k)$.
- Sample $k_i \sim d(k)$ (e.g. the stationary distribution used in the earlier value-iteration slides).
- Define the mean-squared Euler residual loss

$$\mathcal{L}_g(\gamma_g) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{R}\left(k_i\,;\,\Gamma_{\gamma_g}\right)\right]^2. \qquad \text{(ML objective)}$$

- Optimal policy parameters satisfy $\nabla_{\gamma_g}\mathcal{L}_g(\gamma_g^\star) = 0$.

Neural Network for the Policy Function

- Two-hidden-layer MLP (64–64–1) with ReLU non-linearities:

$$\boxed{\Gamma_{\gamma_g}(k) = \sigma_2\Big(W_2 \cdot \sigma_1\big(W_1\, k + b_1\big) + b_2\Big),}$$

- Output activation $\sigma_1 = \texttt{ReLU}$, and $\sigma_2 = \texttt{Sigmoid}$; we scale it to $[0, f(k) + (1-\delta)k - \varepsilon]$ so that $0 \le \Gamma_{\gamma_g}(k) \le f(k) + (1-\delta)k$.
- All derivatives $\partial \mathcal{L}_g / \partial \gamma_g$ needed in the Euler residual are provided automatically by `autograd`.

Minimising the Euler Residual Loss

- Initialise $\gamma_g^{(0)}$; choose learning rate α_g.
- **Repeat until convergence**
  1. Draw a mini-batch $\{k_i\}_{i=1}^{B}$.
  2. Compute loss $\mathcal{L}_g(\gamma_g)$ and its gradient via back-propagation.
  3. Gradient step
  $$\gamma_g^{(j+1)} \;=\; \gamma_g^{(j)} - \alpha_g \, \nabla_{\gamma_g} \mathcal{L}_g\big(\gamma_g^{(j)}\big).$$
  4. Optionally resample k_i from the *same* distribution $d(k)$ used earlier in value-iteration slides (uniform grid, simulated stationary, or importance-sampling mix).

PyTorch Implementation

- **Goal**: Solve optimal-growth model
  - `ValueNet` $\widehat{V}(k \mid \gamma_v)$
  - `PolicyNet` $\widehat{g}(k \mid \gamma_g)$
- **Key steps**:
  1. Network architecture
  2. Bellman MSE evaluation
  3. Policy gradient descent
  4. Training loop
- Single Python file + JSON config

1–Network Architecture

```
class ValueNet(nn.Module):
    def __init__(self, nh=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(1, nh), nn.ReLU(),
            nn.Linear(nh, nh), nn.ReLU(),
            nn.Linear(nh, 1))
    def forward(self, k): return self.net(k)

class PolicyNet(nn.Module):
    def __init__(self, alpha, eps, nh=64):
        super().__init__()
        self.alpha, self.eps = alpha, eps
        self.net = nn.Sequential(
            nn.Linear(1, nh), nn.ReLU(),
            nn.Linear(nh, nh), nn.ReLU(),
            nn.Linear(nh, 1), nn.Sigmoid())
    def forward(self, k):
        max_kp = torch.clamp(k**self.alpha - self.eps, 1e-9)
        return self.net(k) * max_kp
```

- `Sigmoid` ensures $g(k) \in [0, f(k) - \varepsilon]$

2–Policy Evaluation (Bellman MSE)

```
def value_update_step(self):

        self.value_optimizer.zero_grad()
        # Sample batch of capital states
        k_batch = torch.rand(self.n_batch, 1, device=self.device)*(self.k_max-self.k_min) + self.k_min

        # Current estimate: V(k)
        v_current = self.value_net(k_batch)

        # Next capital according to current policy
        kp_batch = self.policy_net(k_batch)

        # Bellman RHS: u + beta*V(k')
        u_batch = self.utility(k_batch, kp_batch)
        v_next = self.value_net(kp_batch).detach()  # detach so we don't backprop through V(k') here
        bellman_rhs = u_batch + self.beta * v_next

        # MSE loss
        loss = nn.functional.mse_loss(v_current, bellman_rhs)
        loss.backward()
        self.value_optimizer.step()

        return loss.item()
```

- $\min_{\gamma_v} \mathbb{E}\big[(V - (u + \beta V'))^2\big]$

3–Policy Improvement (Gradient Descent)

```
def policy_step(self):
       self.policy_optimizer.zero_grad()
        # Sample batch of capital states
        k_batch = torch.rand(self.n_batch, 1, device=self.device)*(self.k_max-self.k_min) + self.k_min

        # Proposed next capital
        kp_batch = self.policy_net(k_batch)
        # Evaluate the objective
        u_batch = self.utility(k_batch, kp_batch)
        v_next = self.value_net(kp_batch)

        # Mean objective
        objective = torch.mean(u_batch + self.beta * v_next)

        # We want to maximize -> so we minimize -objective
        loss = -objective
        loss.backward()
        self.policy_optimizer.step()

        # Return the positive objective for logging
        return objective.item()
```

- $\max_{\gamma_g} \mathbb{E}\big[u(k, \widehat{g}(k)) + \beta \widehat{V}(\widehat{g}(k))\big]$

4–Training Loop

```
def train(self):
        print(f"Training␣started␣on␣device={self.device}")
        pbar = tqdm(range(self.n_epoch), desc="Outer␣Loop")

        for _ in pbar:
            # Value net update
            for _ in range(self.n_inner_value):
                v_loss = self.value_update_step()
            self.value_losses.append(v_loss)

            # Policy net update
            for _ in range(self.n_inner_policy):
                policy_obj = self.policy_update_step()
            self.policy_objectives.append(policy_obj)

            pbar.set_postfix({
                "ValueLoss": f"{v_loss:.4e}",
                "PolicyObj": f"{policy_obj:.4e}"
            })
```

5–Visualization

```
def plot_results(self):
        with torch.no_grad():
            # Evaluate the value function
            V_eval = self.value_net(self.k_grid_eval).cpu().numpy()
            # Evaluate the policy function
            g_eval = self.policy_net(self.k_grid_eval).cpu().numpy()

        # Plot Value Function
        plt.figure(figsize=(12,5))
        plt.subplot(1,2,1)
        plt.plot(self.k_grid_eval.cpu().numpy(), V_eval, label="V(k)")
        plt.xlabel("Capital␣k")
        plt.ylabel("Value␣Function")
        plt.grid(True, alpha=0.3)
        plt.legend()

        # Plot Policy Function
        plt.subplot(1,2,2)
        plt.plot(self.k_grid_eval.cpu().numpy(), g_eval, label="g(k)", color="green")
        plt.xlabel("Capital␣k")
        plt.ylabel("Policy␣Function␣k'")
        plt.grid(True, alpha=0.3)
        plt.legend()

        plt.tight_layout()
        plt.show()
```

The Role & Advantage of Reinforcement Learning

Actor-Critic Perspective

- The combination of:
  - **Value function update**: $\Gamma_{\gamma_v}^{(n)} \leftarrow \text{argmin} \, \|\Gamma_{\gamma_v}^{(n)} - \hat{V}\|^2$.
  - **Policy function update**: $\Gamma_{\gamma_g}^{(n)} \leftarrow \text{argmax} \, \mathbb{E}[u + \beta V]$.
- $\Longrightarrow$ This is essentially an **Actor-Critic** algorithm:
  - Γ_{γ_g} is the *actor* (policy function),
  - Γ_{γ_v} is the *critic* (value function).
- In Reinforcement Learning (RL), actor-critic methods alternate between:

$$\text{Policy Evaluation (Critic)} \quad \longleftrightarrow \quad \text{Policy Improvement (Actor)}.$$

- We will discuss the RL perspective in more detail in a later **Lecture 10**.

Why (Deep) Reinforcement Learning Helps

- **Flexible Function Approximation with Neural Networks.**
  - Traditional dynamic programming methods typically rely on *functional approximation methods that require a structured grid*, such as polynomial expansions (e.g., Chebyshev) or piecewise polynomial interpolation.
  - As the dimension of the state space grows, the number of grid points needed for accurate approximation can increase exponentially—the so-called "curse of dimensionality." Even sophisticated methods that use sparse grids (e.g., Smolyak's algorithm) only partially mitigate this explosion of computational requirements, and can still become infeasible at moderate-to-high dimensions.
  - Deep neural networks, by contrast, do not require defining an exhaustive, structured grid. They can approximate high-dimensional value and policy functions directly, using parameters (weights/biases) that scale more gracefully with dimension.

Why Reinforcement Learning Helps

- **Data Efficiency Through Generative Process and Sampling.**
  - Reinforcement learning algorithms were originally developed for settings where the environment (i.e., model) is *not* fully known a priori.
  - As a result, sophisticated data generating process and sampling strategies (e.g., trajectory rollouts) emerged to *learn* both policies and value functions.
  - In macroeconomic models, we often do know or hypothesize the environment, but as the complexity of that environment increases—for example, in heterogeneous-agent settings—traditional solution methods can become computationally prohibitive or analytically intractable.
    - While a law of motion may be specified mathematically, it might not admit a closed-form solution for the evolution of the distribution–a state variable (e.g., except in special continuous-time cases that permit an analytic Kolmogorov Forward Equation).
    - In such circumstances, RL-style data genetative techniques are especially valuable, as they can approximate the relevant regions of the state space without requiring a fully tractable analytical representation or an exhaustive grid-based approach.
  - This targeted approach to sampling can save substantial computation time compared to covering the entire domain via a uniform grid.

Why Reinforcement Learning Helps

- **Parallel and Hardware Acceleration.**
  - Neural network training is highly amenable to modern hardware such as GPUs and TPUs, since it relies on large-scale linear algebra (matrix multiplications) that can be parallelized efficiently.
  - While one can also parallelize standard DP algorithms (e.g., via MPI or multi-core CPU) by distributing grid points across workers, the underlying operations in grid-based DP may not leverage GPUs as effectively.
  - In contrast, RL-style methods combine batches of simulated data with gradient-based updates to the network, a pattern that fits well into GPU-accelerated frameworks such as TensorFlow or PyTorch.
  - Consequently, RL-based approaches can handle high-dimensional problems more smoothly, as the core update steps (forward/backward passes in the network) map directly onto GPU-friendly operations.

Computation of Equilibrium: Global Solution

Global Solution via Deep Neural Networks (DNN)

- **From Stationarity to Global Dynamics:**
  - The previous algorithm targets the stationary distribution—analogous to finding the steady state in representative agent models.
  - For a global solution, we must model the entire evolution of the distribution over time.
- **DNN for Value and Policy Functions:**
  - The value and policy functions now depend on the full distribution as an additional state variable.
  - Deep Neural Networks are well-suited to approximate such high-dimensional functions.
- **Simulating Distribution Dynamics:**
  - Begin with initial guesses for the policy and value functions.
  - Simulate the economy using the current policy to generate the evolution of the distribution.
  - Use these simulated paths to update the DNN approximations in an iterative fashion.

Household's Problem with Evolving Distribution

- **Extended State Space:** The state now includes:

$$(a, z, \lambda),$$

where $\lambda(a, \epsilon)$ is the distribution of households.

- **Recursive Formulation:**

$$v(a, \epsilon; \lambda) = \max_{c, a'} \left\{ u(c) + \beta \sum_{\epsilon' \in E} v(a', z'; \lambda') \pi(z, z') \right\},$$

subject to:

$$c + a' = (1 + r(\lambda))a + w(\lambda)z, \quad a' \geq -\underline{a}.$$

- **Law of Motion:** The distribution evolves via

$$\lambda' = G(a', z', \lambda, \pi),$$

where $\mathcal{T}$ captures the updating of λ based on individual decisions and shock realizations.

Global Solution via DNN: An Overview

- **Motivation:** The equilibrium above corresponds to a stationary solution. To study dynamics, we model the evolution of the distribution λ over time.
- **Extended State Variables:** Now the value and policy functions depend on (a, z, λ).
- **Simulation Setup:**
  - Index individual assets by i, shocks by j, and distribution by k.
  - for each (i, j, k), simulate a long sequence to capture the evolution of λ and associated prices.
  - the evolution of λ can be done via analytical characterization, or simulations based on the current policy function.
- **DNN Approximation:** Use deep neural networks (DNN) to approximate the high-dimensional functions:

$$\Gamma_{\gamma_v}(a, z, \lambda) \quad \text{and} \quad \Gamma_{\gamma_g}(a, z, \lambda).$$

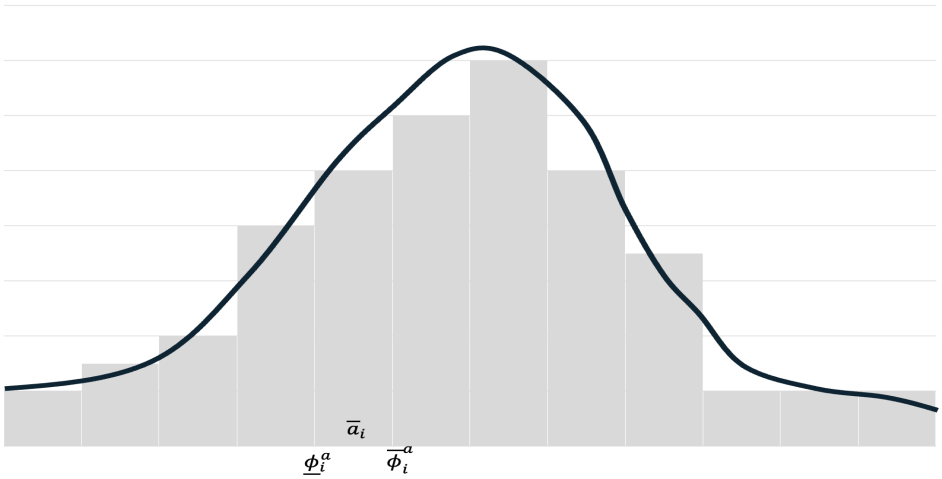
Approximation of Distribution

- The evolution of the distribution λ is driven by the individual asset choices.
- We discretize the space of $\mathbb{A}$ for individual asset holding a, with $\{\phi_i^a\}_{i=1}^{N_a}$, where $\bigcup_i \phi_i^a = \Omega_A$, and Ω_A represents the truncated space of $\mathbb{A}$.
- We approximate the distribution of asset using a vector notation,

$$\hat{f}_a = \begin{bmatrix} x_1 & \dots & \dots & x_N \end{bmatrix},$$

  where each element x_i signifies the density of a within the corresponding bin ϕ_i^a.
  - It is crucial to note that these densities must sum to 1, i.e., $\sum_i x_i = 1$.
  - Each bin ϕ_i^a is defined by an interval $\left[\underline{\phi}_i^a, \bar{\phi}_i^a\right]$, and represented by the mean value $\bar{a}_i$.

Approximation of Distribution



DNN-Representation

- During iteration n, we approximate both the policy function $g^{(n)}$ and the value function $V^{(n)}$ using multilayer deep neural networks (DNNs).
- Denote these approximators by $f_g^{(L_g,n)}$ and $f_V^{(L_V,n)}$, where the state is

$$S = \{a, z, \lambda\},$$

with a (assets), ϵ (idiosyncratic shock), and z (the joint distribution).

- Each DNN is characterized by a set of parameters:

$$\Theta_x = \left\{ \left[b_{x,i}^{(l)},\ \left(w_{x,i,j}^{(l)} \right)_{j=1}^{U_{l-1}} \right]_{i=1}^{U_l} \right\}_{l=1}^{L_x} \cup \left\{ \left[\tilde{b}_{x,i},\ \left(\tilde{w}_{x,i,j} \right)_{j=1}^{U_{L_x}} \right]_{i=1}^{N_x} \right\},$$

where $x \in \{g, V\}$, U_l is the number of nodes in layer l (with U_0 corresponding to the input dimension), and N_x is the output dimension.

Evolving the Distribution–Characterization via $\hat{G}$

- Start with an initial distribution and we focus on a particular agent. The agent draws an initial asset a_i and an idiosyncratic shock z_j.
- Simulate the economy forward for T periods.
- At each period t, record:
  - Each agent's asset $a_{i,t}$, the realized shock $z_{j,t}$;
  - The current distribution $\lambda_{k,t}$ of the joint distribution;
  - Individual decision based on the state variables above and the current policy function;
  - The distribution in the next period will be calculated based on the current policy functions and the law of motion, discussed later.

Update Value Function

- Given the current policy function $f_g^{(L_g,n)}(S)$, we generate a large number N_V of initial states

$$S_{i,0} = \left\{ \{a_{i,0}, z_{i,0}\}_{i=1}^{N}, \lambda_{i,0} \right\}$$

  sampled from the state space $\mathfrak{S}$.

- For each state $S_{i,0}$, we simulate forward for T_V periods. At each period t:
  - The agent's asset and shock evolve according to the policy g;
  - Consumption is determined via the budget constraint

$$c_{i,t} + a_{i,t+1} = (1 + r(\lambda_t))\, a_{i,t} + w(\lambda_t)\, z_{i,t}.$$

Update Value Function

- The realized utility (target value) is then given by

$$\hat{V}(S_{i,0}) = \sum_{t=0}^{T_V-1} \beta^t \, u\Big(c_{i,t}\Big),$$

  where $u(c)$ depends solely on consumption.

- We update the value network by mapping each $S_{i,0}$ to $\hat{V}(S_{i,0})$ via deep learning:

$$\min_{f_V^{(L_V,n+1)}} \mathbb{E}_{S\in\mathfrak{S}}\Big\{\mathcal{L}\Big(\hat{V}(S), f_V^{(L_V,n+1)}(S)\Big)\Big\},$$

  where $\mathcal{L}$ denotes a suitable loss function.

Update Policy Function

- Taking the updated value function $f_V^{(L_V,n+1)}$ as given, we update the policy network.
- For each state $S_{i,t} = \{a_{i,t}, z_{i,t}, \lambda_t\}$, the policy network $f_g^{(L_g,n+1)}(S_{i,t})$ determines the next period asset choice $a_{i,t+1}$ (and hence consumption via the budget constraint).
- The objective is to maximize the expected discounted utility:

$$\max_{f_g^{(L_g,n+1)}} \mathbb{E}_{S\in\mathfrak{S}} \left\{ \sum_{t=0}^{T_g-1} \beta^t \, u\Big(c_{i,t}\Big) + \beta^{T_g} \, f_V^{(L_V,n+1)}\Big(S_{i,T_g}\Big) \right\},$$

  where $c_{i,t}$ is determined by

$$c_{i,t} = (1 + r(\lambda_t))\, a_{i,t} + w(\lambda_t)\, z_{i,t} - f_g^{(L_g,n+1)}(S_{i,t}).$$

- The policy update is performed subject to the model's budget constraint.

Update Transition Function G

- The transition function G is represented by a matrix $\hat{G}$ whose element $\hat{G}_{i,j}$ is the probability that an agent moves from bin ϕ_i^a to ϕ_j^a according to the policy function g and the realized shock ϵ:

$$\hat{G}_{i,j} = \sum_k w_k \Pr\left(\underline{\phi}_j^a \leq a_+ \leq \bar{\phi}_j^a \,\middle|\, a \in \phi_i^a\right),$$

where $a_+ = f_g^{(L_g, n+1)}(a, z, \lambda)$ is the next-period asset choice and w_k are weights for the discretized shock values.

Update Transition Function G

- Given a policy function $a_+(S)$, we define a transitional kernel (conditional on A) as follows:

$$\lambda_+(z_+ \in \mathcal{Z}, a_+ \in \mathcal{A}) = \int_{\mathbb{Z}\times\mathbb{A}} \mathbf{1}_{\{a_+(a,z,\lambda)\in\mathcal{A},\ z_+(a,z,\lambda)\in\mathcal{Z}\}} \mathbf{d}\lambda(z,a), \tag{1}$$

  where $\mathcal{A}, \mathcal{Z}$ are two arbitrary Borel measurable sets in $\mathbb{A}$ and $\mathbb{Z}$.

- Similarly, we have the conditional transition kernel

$$\lambda_+(z_+ \in \mathcal{Z}', a_+ \in \mathcal{A}' | z \in \mathcal{Z}, a \in \mathcal{A}) = \int_{\mathcal{Z}\times\mathcal{A}} \mathbf{1}_{\{a_+(a,z,\lambda)\in\mathcal{A}',\ z_+(a,z,\lambda)\in\mathcal{Z}'\}} \mathbf{d}\lambda(z,a), \tag{2}$$

Update Transition Function G

- Under mild regularity condition, the optimal saving decision of the household is continuous, differentiable in a. Hence, for each $z > 0$ and $\varsigma_a > \underline{a}$, there exists $\hat{a} > \underline{a}$ such that $a_+(\lambda, z, \hat{a}) = \varsigma_a$. Let $H(\lambda, z, \varsigma_a) = \hat{a}$. Thus, we have:

$$\lambda_+(\varsigma_z, \varsigma_a | z, a) = \Pr(z_+ \leq \varsigma_z, a_+ \leq \varsigma_a), \tag{3}$$

$$= \int_{(0,\infty)\times[\underline{a},\infty)} \Pr(z_+ \leq \varsigma_z | z) \cdot \Pr(a_+ \leq \varsigma_a | z, a) \mathbf{d}\lambda(z, a | A) \tag{4}$$

- This recursive relationship induces a stochastic process $\{\lambda_t\}_{t=0}^{\infty}$ in the Wasserstein space, which consists of all probability measures.

Approximation of the Transition of λ

- When z is i.i.d, policy a_+ is monotone and thus invertible w.r.t. z for all (a, A, λ),

$$\begin{aligned}\hat{G}_{i,j}^{\lambda,\eta_x} &= \mathrm{P}\left(\underline{\phi}_j^a \le a_+\left(\lambda, z, \phi_i^a\right) \le \bar{\phi}_j^a \middle| \underline{\phi}_i^a \le a \le \bar{\phi}_i^a\right) \\ &= \int_{\mathbb{Z}\times\mathbb{A}_i} \mathbf{1}_{\left\{\underline{\phi}_j^a \le a_+(\lambda,z,a) \le \bar{\phi}_j^a\right\}} \mathbf{d}\lambda(z,a) \qquad (5) \\ &= F_z\left(\bar{z}_{i,j}\right) - F_z\left(\underline{z}_{i,j}\right) \qquad (6)\end{aligned}$$

- Given the current state of the economy (λ), and the updated policy functions $\left\{\eta_x^{(L_x,n+1)}\right\}$, the transition function G is approximated by the matrix $\hat{G}^{\lambda,\eta_x}$. Each element of $\hat{G}_{i,j}^{\lambda,\eta_x}$ represents the probability that asset holding moves from bin ϕ_i^a to ϕ_j^a.

Approximation of the Transition of λ

- F_z denotes the CDF of z, and $\underline{z}_{i,j}$ and $\bar{z}_{i,j}$ are determined by solving the following two equations.

$$\eta_{a_+}^{(L_{a_+},n+1)}\left(\lambda, \underline{z}_{i,j}, \bar{a}_i\right) = \underline{\phi}_j^a \tag{7}$$

$$\eta_{a_+}^{(L_{a_+},n+1)}\left(\lambda, \bar{z}_{i,j}, \bar{a}_i\right) = \bar{\phi}_j^a. \tag{8}$$

- It is important to note that $\hat{G}^{\lambda,\eta_x}$ is contingent upon:
  - the specific realization of (λ);
  - as well as the current $\left\{\eta_x^{(L_x,n+1)}\right\}$.

Evolving the Distribution–Simulation Based

- Start with N agents. Each agent draws an initial asset a_i and an idiosyncratic shock z_j.
- The joint distribution λ is approximated by a histogram. For simplicity, assume that z takes N_z values; then the joint distribution can be characterized by the histogram over the asset grid for each shock value.
- Simulate the economy forward for T periods.
- At each period t, record:
  - Each agent's asset $a_{i,t}$, the realized shock $z_{j,t}$;
  - The current distribution $\lambda_{k,t}$, which will be approximated by histogram for the N agents;
  - Their decisions based on the state variables above and the current policy function.

Evolution of the Distribution–Simulation Based

- **Monte Carlo Simulation:** Standard ML methods approximate the evolution of the distribution λ by simulating forward to obtain λ_+ using candidate equilibrium policies Azinovic et. al (2024), Han, Yang and E (2025).
- **Efficiency vs. Differentiability:** Although computationally efficient, this simulation introduces a discontinuous mapping between λ and λ_+ due to discrete transitions (agents moving across bins), leading to a loss of differentiability.
- **Impact on Optimization:** The differentiability of the transition kernel Φ is crucial for backpropagation in the government's optimization problem; without it, the Euler equation constraints become problematic.

Analytical Transition Kernel for Efficient Learning

- **Histogram Approximation:** The asset distribution is approximated via a histogram, discretizing the state space into finite bins and tracking the number of agents in each bin.
- **Analytical Derivation:** The transition kernel is defined analytically:

$$\lambda'(z_+ \in \mathcal{Z}, a_+ \in \mathcal{A}) = \int_{\mathbb{Z}\times\mathbb{A}} \mathbf{1}_{\{a_+(a,z,\lambda)\in\mathcal{A},\, z_+(a,z,\lambda)\in\mathcal{Z}\}}\, d\lambda(z,a)$$

- **Maintaining Differentiability:** This kernel Φ is differentiable with respect to state variables, ensuring efficient backpropagation.
- **Online Update:** The transition kernel is updated online alongside policy functions $\Lambda(S_{agg})$, maintaining consistency between distribution dynamics and equilibrium policies.
- **Outcome:** This approach marries the flexibility of machine learning with analytical tractability to efficiently solve for the Markov Perfect Equilibrium in heterogeneous agent models.

Actor-Critic Perspective for the Global Solution

- **Critic (Value Network):**

$$\eta_v^{(n+1)} \leftarrow \arg\min \mathbb{E}_{S\in\mathfrak{S}} \left\| \eta_v^{(n)}(a_i, \lambda_j, z) - \hat{V}(a_i, \lambda_j, z) \right\|^2.$$

- **Actor (Policy Network):**

$$\eta_g^{(n+1)} \leftarrow \arg\max_{\eta_g} \mathbb{E}_{S\in\mathfrak{S}} \Big[u\big(a_i, \eta_g^{(n)}(a_i, \lambda_j, z)\big) + \beta\, \eta_v^{(n)}\big(f(a_i, \eta_g^{(n)}(a_i, \lambda_j, z)), \lambda', z'\big) \Big].$$

- The procedure alternates between:

$$\text{Policy Evaluation (Critic)} \quad \longleftrightarrow \quad \text{Policy Improvement (Actor)},$$

similar to standard actor-critic algorithms in reinforcement learning.

Krusell and Smith (1998)

Krusell-Smith Model

- There is a continuum of households, and each one is endowed with one unit of time.
- Households can provide ϵ units of labor. ϵ can take value 0 (unemployed) or 1 (employed).
- There is also a stochastic shock to the aggregate productivity of the only firm, z_t. This shock can take two values z_g, z_b.
- Individual and aggregate shocks are correlated. Shocks are Markov processes. Individual shocks are such that the fraction of unemployed is u_g when aggregate productivity is good, and u_b when it is bad.
- There is only one asset. Capital, and each agent is constrained to hold a positive amount at any given state.

Krusell-Smith Model

- Households collect income from working and from renting their capital to the firm.
- Let Γ be the current measure of consumers over holdings of capital and employment.
- The firm employs an aggregate measure of time $\bar{l}$ and capital $\bar{k}$.
- The production function is Cobb-Douglas and thus $w = (1-\alpha)\bar{k}^{\alpha}\bar{l}^{-\alpha}; r = \alpha\bar{k}^{\alpha-1}\bar{l}^{1-\alpha}$

Krusell-Smith Model

- Production economy with a continuum of households: each HH i solves

$$\max_{c_{i,t}\geq 0, a_{i,t+1}\geq \underline{a}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u\left(c_{i,t}\right)$$

  subject to budget constraint

$$a_{i,t+1} = w_t \bar{\ell} y_{i,t} + R_t a_{i,t} - c_{i,t}$$

- Idiosyncratic shocks on employment status $y_{i,t}$.
- Representative firm produces $Y_t = Z_t F(K_t, \bar{L})$.
- Aggregate shock $Z_t \sim$ two-state Markov, and enters HH's problem through competitive factor prices:

$$R_t = Z_t \partial_K F(K_t, \bar{L}) - \delta,\ w_t = Z_t \partial_L F(K_t, \bar{L})$$

Krusell-Smith Model

Curse of dimensionality shows up in recursive form of HH i's problem:

$$V(a_i, y_i, Z, \Gamma) = \max_{c_i, a_i'} \left\{ u(c_i) + \beta \mathbb{E} V \left(a_i', y_i', Z', \Gamma' | y_i, Z \right) \right\} \tag{9}$$

subject to budget and borrowing constraints. : distribution of all HHs' states.

Krusell-Smith method (KS, 1998; Maliar et al., 2010):

1. Approximate state vector: $\hat{s}_i = (a_i, y_i, Z, m_1)$, where m_1 is first moment of individual asset distribution.
2. Log linear law of motion for m_1:

$$\log(m_{1,t+1}) = A(Z) + B(Z) \log(m_{1t}).$$

Very costly in complex HA models with multiple assets or multiple shocks.

Traditional Aggregation Approximation (Finite Moments)

- **Key idea**: Approximate the distribution of individual states (e.g., assets) using a small set of aggregate moments (e.g., the mean or mean and variance of asset holdings).
- **Motivation**: Storing and updating the full distribution Γ is typically very costly or infeasible.
- **Approach**:
  - Choose a finite number of moments $\{m_1, m_2, \ldots, m_J\}$ of the asset distribution (e.g., $m_1 =$ mean, $m_2 =$ variance, etc.).
  - Postulate (guess) a *law of motion* for these moments that depends on current moments and the aggregate state Z. For instance, for a single moment m_1, one might guess:

$$\log(m_{1,t+1}) = A(Z_t) + B(Z_t)\,\log(m_{1,t}).$$

  - Use these guessed laws of motion to solve the household's dynamic programming problem (they serve as an expectation of future aggregates).

Solving the Model via Moment Approximation: Steps

- **Initial Guess**:
  - Specify the functional form of the law of motion for each chosen moment.
  - Initialize the parameters $\{A(Z), B(Z)\}$ (and so on, if more moments are used).
- **Household Problem**:
  - Given the guessed laws of motion for the moments (and hence implied aggregate prices R_t, w_t), solve the household's dynamic programming problem. This yields policy functions $a' = g(a, y, Z, m_1, \ldots, m_J)$.
- **Simulation**:
  - Simulate a large number of households (or a representative sample), each following the derived policy rules.
  - For each period t, compute the realized moments $\hat{m}_{1,t+1}, \ldots, \hat{m}_{J,t+1}$.

Solving the Model via Moment Approximation: Steps

- **Update the Law of Motion**:
  - Regress the realized moments on the previous period's state:

$$\log(\hat{m}_{j,t+1}) \approx A_j(Z_t) + B_j(Z_t)\,\log(m_{j,t}), \quad j = 1, \ldots, J.$$

  - Obtain new estimates $\{A_j(Z), B_j(Z)\}$.
- **Iteration and Convergence**:
  - Use the updated parameters as the new guess for the law of motion.
  - Repeat until convergence of the law of motion parameters (i.e., when they change little from one iteration to the next).
- **Limitations of Moment Approximations**:
  - Approximating the distribution with moments does not yield an explicit law of motion for the full distribution, requiring simulation-based methods (e.g., see the upcoming DeepHAM algorithm).
  - A key drawback: these methods are often inadequate for studying optimal policy, since a social planner typically needs to know how the entire distribution responds to policy changes (Feng et al., 2025).

Krusell-Smith Model: DeepHAM

- Based on Han, Yang and E (2025), "**DeepHAM: A global solution method for heterogeneous agent models with aggregate shocks**"
- In N-agent Krusell-Smith problem (N finite but large), define cumulative utility for HH i up to t:

$$\tilde{U}_{i,t} = \sum_{\tau=0}^{t} \beta^{\tau} u\left(c_{i,\tau}\right)$$

- Solve Markov Nash equilibrium: optimal policy $\mathcal{C}(s)$ solve HH i's problem:

$$V(s_{i,0}) = \max_{c_{i,t}\geq 0, a_{i,t+1}\geq \underline{a}} \mathbb{E}_0 \tilde{U}_{i,\infty} = \max_{\{c_{i,t}\geq 0, a_{i,t+1}\geq \underline{a}\}_{t=0}^{T}} \mathbb{E}_0 \left(\tilde{U}_{i,T} + \beta^T V(s_{i,T})\right)$$

  subject to the constraint that all other HHs also follow $\mathcal{C}(s)$, where $s_i = (a_i, y_i, Z, \Gamma)$, and Γ is distribution of all HHs' states.

- Value function $V(s)$ and policy function $\mathcal{C}(s)$ are parameterized by deep neural networks.

DeepHAM: Outline of Algorithm

Initialization: initial policy $\mathcal{C}^{(0)}$, initial value and policy neural networks with parameters Θ^V and Θ^C. In the k-th iteration:

1. Prepare the stationary distribution $\mu(\mathcal{C}^{(k-1)})$ according to the policy $\mathcal{C}^{(k-1)}$
2. Given policy Θ^C, update the value function by solving

$$\min_{\Theta^V} \mathbb{E}_{\mu(\mathcal{C}^{(k-1)})} \left[V_{\mathsf{NN}}(s_i; \Theta^V) - \widehat{V}_i \right]^2,$$

   where $\widehat{V}_i = \sum_{\tau=0}^{T_v} \beta^\tau u(c_\tau^i)$ is truncated lifetime utility with large T_v.
3. Given value Θ^V, optimize the policy function by solving

$$\max_{\Theta^C} \mathbb{E}_{\mu(\mathcal{C}^{(k-1)})} \left[\sum_{t=0}^{T} \beta^t u(c_{i,t}) + \beta^T V_{\mathsf{NN}}(s_{i,T}; \Theta^V) \right],$$

   following the spirit of fictitious play.

DeepHAM: Value Function Learning

In iteration k, given policy function $\mathcal{C}^{(k-1)}(s)$:

1. Simulate the economy many times for T_v periods ($T_v \gg T$), sample states s from simulations. Then the value of each state s can be approximately calculated as cumulative utility in the following T periods:

$$\tilde{V}^{(k)}(s) \approx \mathbb{E}\tilde{U}_T = \mathbb{E}\sum_{\tau=0}^{T} \beta^{\tau} u\left(c_{i,\tau}\right)$$

2. Learn value function $V^{(k)}(s)$ parameterized by deep neural networks with supervised-learning.

DeepHAM: Policy Function Iteration

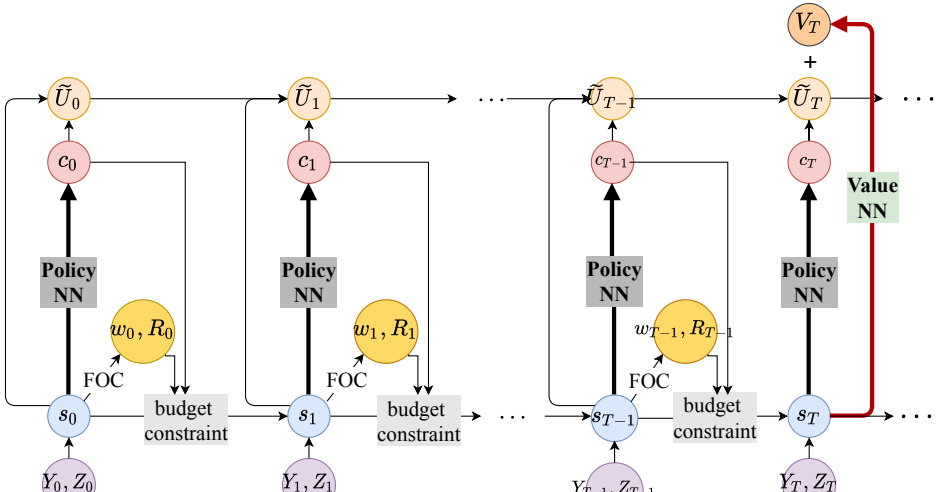
In iteration k, given value function $V^{(k)}(s)$, optimize policy function $\mathcal{C}^{(k)}(s)$ in a fictitious play.

- Fictitious play: separate N-agent problem into N individual problems where other agents strategies are fixed and follow the policy in the past "play".
- Solve the following problem iteratively:
  1. At round $\ell + 1$, everybody's policy $\mathcal{C}^{(k,\ell)}(s)$ is known.
  2. For agent $i = 1$, solve for her optimal policy $\mathcal{C}^{(k,\ell+1)}(s)$:
  $$\max_{\mathcal{C}^{(k,\ell+1)}(s)} \mathbb{E}_{\mu(\mathcal{C}^{(k-1)})} \left(\tilde{U}_{i,T} + \beta^T V^{(k)}(s_{i,T}) \right)$$
  subject to others all following $\mathcal{C}^{(k,\ell)}(s)$ in the first T periods (T large).

Optimization is solved on Monte Carlo simulation with N agents on a large number of sample paths in a computational graph.

Computational Graph Formulation for DeepHAM

$$\max_{\Theta^C} \mathbb{E}_{\mu(C^{(k-1)})} \left(\widetilde{U}_{i,T} + \beta^T V_{\mathsf{NN}}(s_{i,T}; \Theta^V) \right)$$



DL-Based Model Reduction and Generalized Moments

- General form of value & policy functions are like (ignore y):

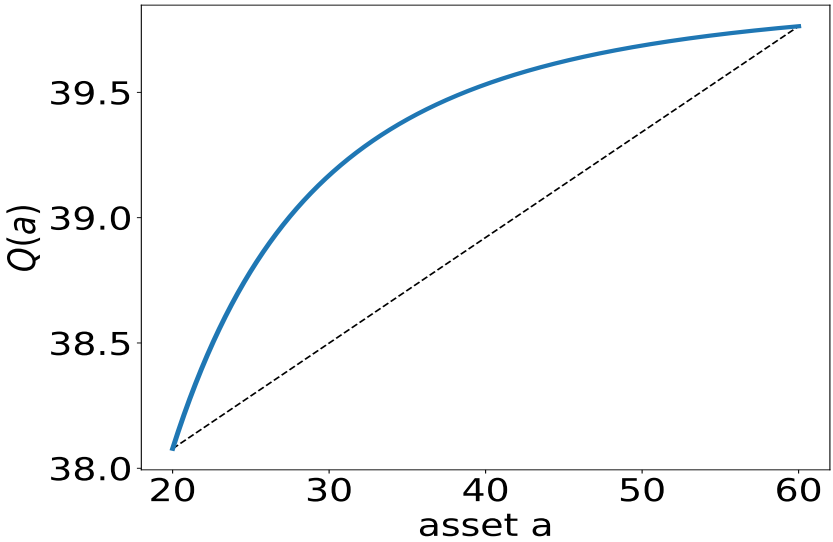
$$V(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z), \quad c(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z)$$

- For permutational invariance, specify its approximated form with symmetry preserving generalized moments $\frac{1}{N}\sum_i Q_j(a_i)$, with basis function $Q_1(\cdot), ..., Q_J(\cdot)$ parameterized by (sub) neural networks:

$$V(a_i; \frac{1}{N}\sum_i Q_1(a_i), \ldots, \frac{1}{N}\sum_i Q_J(a_i); Z)$$
$$c(a_i; \frac{1}{N}\sum_i Q_1(a_i), \ldots, \frac{1}{N}\sum_i Q_J(a_i); Z)$$

- Special case: $Q(a) = a$ yields the first moment.
- The algorithm optimally solve generalized moments that matters most for policy and value functions. ("numerically determined sufficient statistics")
- Generalized moments provide interpretabilitiy on how heterogeneity matters in model.

Interpretation of Generalized Moments



Deep Learning for High Dimensional Models

- Deep learning has achieved success in high dimensional problems.
- **Key idea**: use deep learning to "learn" policy function, value function, and distribution representation in high dimensional HA models.
- Three key elements to "learn" high-dim functions:
  1. Deep neural networks to represent function:

$$f(x) = \mathcal{L}^{\text{out}} \circ \mathcal{L}^{N_h} \circ \mathcal{L}^{N_h-1} \circ \cdots \circ \mathcal{L}^{1}(x),$$

$$h_p = \mathcal{L}^{p}(h_{p-1}) = \sigma\Big(W_p\, h_{p-1} + b_p\Big),$$

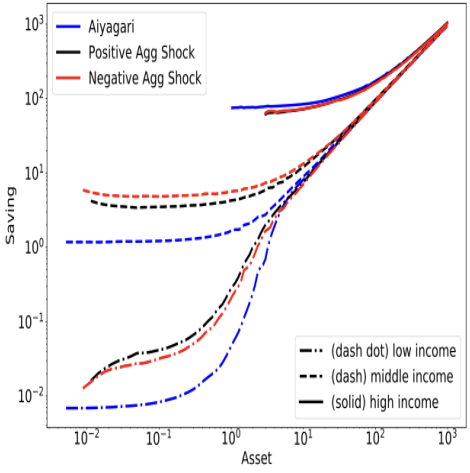
     σ : element-wise nonlinear activation function: e.g. $\max(0, x)$.

  2. Cast high-dim function into an objective function:

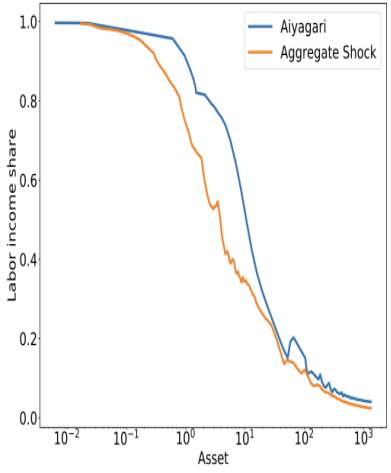
$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(f(x_i; \theta)).$$

  3. Efficient optimization: stochastic gradient descent (SGD).

Some plots

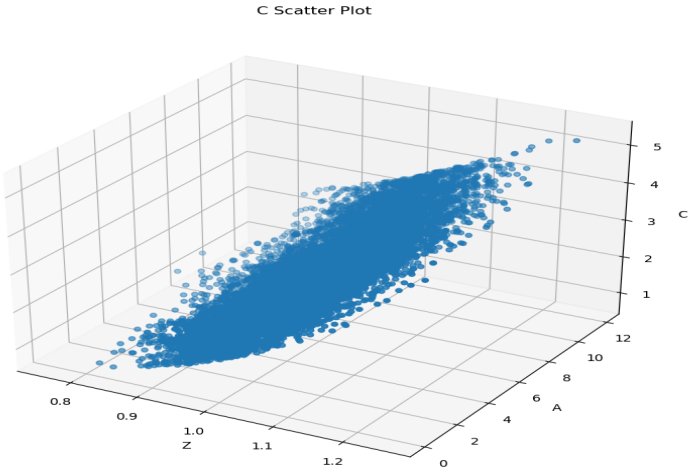


(a) Saving policy

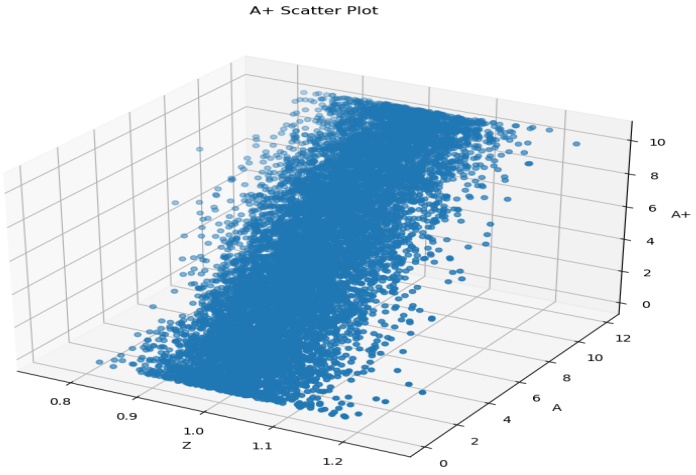


(b) Labor share

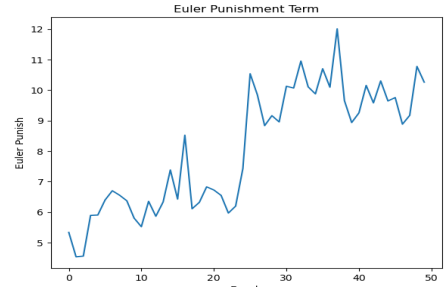
Some plots

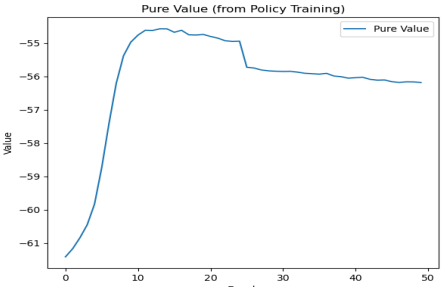


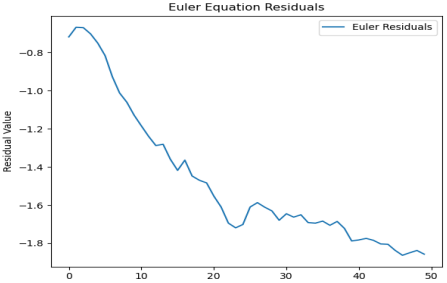
Some plots

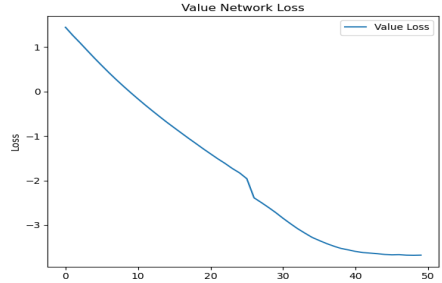


Some plots


Euler Punishment Term
Euler Punish
Epoch


Pure Value (from Policy Training)
Pure Value
Value
Epoch


Euler Equation Residuals
Euler Residuals
Residual Value


Value Network Loss
Value Loss
Loss

Exercise: Krusell–Smith Model with the Distribution as a State Variable

- **Task:** Download and examine the code provided at the link below, which solves a classical Krusell–Smith model treating the distribution of assets as a state variable.
- **Implementation:**
  1. Run the code on your own computing setup.
  2. Carefully trace the workflow to understand how the algorithm tracks the evolution of the distribution.
  3. Note any computational or conceptual bottlenecks you encounter.
- **Reflection and Feedback:**
  - Suggest possible improvements or extensions to the algorithm (e.g., handling additional shocks, alternative solution methods, or parallelization).
  - Consider how changes in the model specification (preferences, technology, market structure) might affect the solution approach.
- **Code and Materials:**
  `https://www.dropbox.com/scl/fo/81cgih9wkjb3r0f6zryxk/ANAs7AOIkJ4H3pmaQQ6W2E0?rlkey=rpctb1sp0nuq9dnz5adoeqp7k&dl=0`

Deep Equilibrium Nets (DEQN)

An Alternative: Deep Equilibrium Nets

- Based on Azinovic, Gaegauf, and Scheidegger (2022), "**Deep Equilibrium Nets**"
- Solves the **same Krusell-Smith model** as DeepHAM, but with a fundamentally different approach.
- **Key difference from DeepHAM**:
  - DeepHAM: two networks (value + policy), trained via fictitious play and supervised learning of the value function.
  - DEQN: **one network** (policy only), trained by minimizing **Euler equation residuals**.
- No value function is estimated — optimality is enforced directly through the household's first-order condition.
- Uses a "**cloud**" of N parallel economies to generate training data.

DEQN: Policy Network

A single neural network approximates the household's **savings rate**:

$$\sigma_\theta(a, e, Z, K) \;\in\; (0, 1)$$

Given cash-on-hand $\text{coh} = (1 + r)\, a + w\, e$, the savings decision is:

$$a' = \underline{a} + \sigma_\theta \cdot (\text{coh} - \underline{a})$$

- **Inputs**: individual state (a, e) and aggregate state (Z, K) where $K = \int a \, d\mu$.
- **Output**: logit passed through sigmoid $\rightarrow$ savings rate $\sigma \in (0, 1)$.
- **Architecture**: feedforward net with 3 hidden layers, 64 units each, Mish activation.
- The sigmoid guarantees $a' \in [\underline{a}, \text{coh}]$, automatically satisfying the borrowing constraint and budget feasibility.

Contrast with DeepHAM
DeepHAM parameterizes both $V_{\text{NN}}(s; \Theta^V)$ and $\mathcal{C}_{\text{NN}}(s; \Theta^C)$. DEQN needs only the policy network — the Euler equation replaces the role of the value function.

DEQN: The Euler Equation Loss

The household's optimality condition (Euler equation) states:

$$u'(c_t) = \beta\, \mathbb{E}_t \left[(1 + r_{t+1})\, u'(c_{t+1}) \right]$$

with equality when $a_{t+1} > \underline{a}$, and $\geq$ at the constraint.

Define the **Euler residual**:

$$\varepsilon(a, e, Z, K) \;=\; 1 \;-\; \frac{\beta\, \mathbb{E}\left[(1 + r')\, u'(c') \right]}{u'(c)}$$

The **training loss** is the mean squared residual over sampled agents:

$$\mathcal{L}(\theta) = \frac{1}{M} \sum_{m=1}^{M} \varepsilon_m^2$$

where M agents are randomly sampled from the cloud of economies.

Why Euler equations?
The Euler equation is a *necessary and sufficient* condition for optimality in this class of models. By driving $\varepsilon \to 0$, the network learns the optimal policy without ever computing a value function.

DEQN: Cloud Simulation

Problem: To evaluate the Euler loss, we need aggregate states (K, Z). But $K = \int a \, d\mu$ depends on the distribution μ, which evolves endogenously.

Solution: Simulate N independent copies (a "cloud") of the economy in parallel:

$$(\mu_{n,0},\, Z_{n,0}) \;\to\; (\mu_{n,1},\, Z_{n,1}) \;\to\; \cdots \;\to\; (\mu_{n,T},\, Z_{n,T}), \quad n = 1, \ldots, N$$

Each copy advances **one period per training epoch**.

- **Epoch 0**: All N copies start at μ_{ss} (stationary distribution from SSJ). Only Z_n varies.
- **As training progresses**: Each copy's $\mu_{n,t}$ evolves under its own TFP shock sequence $\Rightarrow$ the set of aggregate capitals $\{K_n\}$ spreads across a realistic range.
- **Result**: A diverse, economically consistent batch $\{(K_n, Z_n)\}_{n=1}^{N}$ for computing the Euler loss.

Contrast with DeepHAM
DeepHAM simulates N agents within *one* economy and uses T_v-step rollouts to estimate the value function. DEQN simulates N *economies*, each carrying its own full distribution μ_n, and uses the Euler equation directly.

DEQN: Distribution Transport

At each epoch, the distribution $\mu_{n,t}$ is a histogram over the grid (a_j, e_k). The policy network determines how mass moves:

1. **Compute savings** $a'_{j,k} = \underline{a} + \sigma_\theta(a_j, e_k, Z_n, K_n) \cdot (\text{coh}_{j,k} - \underline{a})$ for every grid point (a_j, e_k).
2. **Linear interpolation**: Since $a'_{j,k}$ falls between grid points $a_\ell \le a'_{j,k} \le a_{\ell+1}$, split the mass $\mu(a_j, e_k)$ proportionally:
$$w_{\ell+1} = \frac{a' - a_\ell}{a_{\ell+1} - a_\ell}, \quad w_\ell = 1 - w_{\ell+1}$$
3. **Markov transition for** e: Distribute mass to (a_ℓ, e') and $(a_{\ell+1}, e')$ weighted by $\Pi_e(e_k, e')$.
4. **Update aggregate capital**: $K_{n,t+1} = \sum_{j,k} a_j \cdot \mu_{n,t+1}(a_j, e_k)$.

This transport step is performed inside `torch.no_grad()` — it simulates the economy forward but gradients flow only through the Euler loss.

DEQN: Outline of Algorithm

Initialization: Solve steady state via SSJ $\rightarrow$ obtain K_{ss}, grids (a_j, e_k, Z_ℓ), stationary distribution μ_{ss}, and Markov matrices Π_e, Π_Z. Initialize policy network σ_θ.

Training loop (repeat for T epochs):

1. **Aggregate state**: For each cloud copy n, compute $K_n = \int a \, d\mu_n$ and read Z_n.
2. **Sample agents**: Draw M random (a, e) pairs per cloud copy.
3. **Forward pass**: Evaluate $\sigma_\theta \rightarrow (a', c)$ for sampled agents.
4. **Euler loss**: Compute ε^2 using expectations over (Z', e').
5. **Backpropagate**: Update θ via Adam.
6. **Transport**: Move $\mu_n \rightarrow \mu'_n$ using the full-grid policy (no grad). Draw next Z_n.

Convergence
Loss should decrease steadily over 2000 epochs; policy functions should stabilize. Euler errors below 10^{-3} indicate a well-solved model.

DEQN vs. DeepHAM: Summary

	DeepHAM	DEQN
Reference	Han, Yang & E (2025)	Azinovic, Gaegauf & Scheidegger (2022)
Networks	Two: value V_{NN} + policy $\mathcal{C}_{\text{NN}}$	One: policy σ_θ only
Optimality	Value iteration + fictitious play	Euler equation residuals
Distribution	Generalized moments $\frac{1}{N}\sum_i Q_j(a_i)$ learned by sub-networks	Full histogram μ on grid, transported explicitly
Training data	N agents, one economy, T_v-step roll-outs	N economies ("cloud"), each with own μ_n
Scalability	Generalized moments handle high-dim distributions	Adding states = changing NN input dimension

Both methods avoid the curse of dimensionality inherent in traditional grid-based approaches. DeepHAM provides richer distributional information via learned generalized moments; DEQN offers a simpler single-network architecture grounded in the Euler equation.