Estimating Nonlinear Heterogeneous Agent Models with Neural Networks

Hanno Kase (ECB) Leonardo Melosi (EUI, CEPR) Matthias Rottner (BIS, Deutsche Bundesbank)

Frontier Methods for Monetary & Fiscal Policy — CEF 2026

Motivation

- Quantitative macro models can be:
  - **Nonlinear** (occasionally binding constraints, large shocks)
  - **High-dimensional** (many state variables, distributions in HANK)
  - **Richly parameterized**
- Classical solution methods:
  - Rely on local linearization, perturbation, grids ...
  - Cannot globally solve nonlinear, high-dimensional models

Motivation

- Quantitative macro models can be:
  - **Nonlinear** (occasionally binding constraints, large shocks)
  - **High-dimensional** (many state variables, distributions in HANK)
  - **Richly parameterized**
- Classical solution methods:
  - Rely on local linearization, perturbation, grids ...
  - Cannot globally solve nonlinear, high-dimensional models

This limits

- The class of models we can **solve globally**
- The models we can **estimate** with full-information likelihood methods

Motivation

- Quantitative macro models can be:
  - **Nonlinear** (occasionally binding constraints, large shocks)
  - **High-dimensional** (many state variables, distributions in HANK)
  - **Richly parameterized**
- Classical solution methods:
  - Rely on local linearization, perturbation, grids ...
  - Cannot globally solve nonlinear, high-dimensional models

This limits

- The class of models we can **solve globally**
- The models we can **estimate** with full-information likelihood methods

How to use deep learning to overcome these hurdles

- Propose a neural-network framework to **globally solve and estimate** nonlinear (HANK) models
- Apply it to a **HANK model with a zero lower bound** estimated on U.S. data.

Two key innovations

Two estimation bottlenecks, two innovations:

1. **Extended policy functions**
   - Approximate the policy function with a deep NN
   - Treat **parameters as pseudo-states**

Two key innovations

Two estimation bottlenecks, two innovations:

1. **Extended policy functions**
   - Approximate the policy function with a deep NN
   - Treat **parameters as pseudo-states**

Avoids repeatedly solving the model

Deep neural networks are great at approximating high-dimensional functions.

Two key innovations

Two estimation bottlenecks, two innovations:

1. **Extended policy functions**
   - Approximate the policy function with a deep NN
   - Treat **parameters as pseudo-states**

Avoids repeatedly solving the model

Deep neural networks are great at approximating high-dimensional functions.

2. **Neural Network Particle Filter**
   - Evaluating the likelihood of a complex model can be costly
   - The particle filter becomes the **bottleneck** for estimation

Two key innovations

Two estimation bottlenecks, two innovations:

1. **Extended policy functions**
   - Approximate the policy function with a deep NN
   - Treat **parameters as pseudo-states**

Avoids repeatedly solving the model

Deep neural networks are great at approximating high-dimensional functions.

2. **Neural Network Particle Filter**
   - Evaluating the likelihood of a complex model can be costly
   - The particle filter becomes the **bottleneck** for estimation

Reduces the cost of likelihood evaluations

- Use the particle filter to generate a dataset of parameters and likelihoods
- Fit a deep neural network on it: a **surrogate** for the particle filter

Solving the model with deep neural networks

Euler-residual minimization (Maliar et al. 2021):

0. Replace the continuum of agents by a finite but large number L of agents

1. **Parameterize** individual and aggregate policy functions with deep NNs:

$$\psi_t^i = \psi^I(S_t^i, S_t \mid \Theta) \approx \psi_{NN}^I(S_t^i, S_t \mid \Theta)$$
$$\psi_t^A = \psi^A(S_t \mid \Theta) \approx \psi_{NN}^A(S_t \mid \Theta)$$

2. **Construct a loss**: the weighted mean of squared Euler / equilibrium residuals

3. **Train** by stochastic optimization: minimize the loss, then **simulate the model forward** for new state points

Solving the model with deep neural networks

Euler-residual minimization (Maliar et al. 2021):

0. Replace the continuum of agents by a finite but large number L of agents

1. **Parameterize** individual and aggregate policy functions with deep NNs:

$$\psi_t^i = \psi^I(S_t^i, S_t \mid \Theta) \approx \psi_{NN}^I(S_t^i, S_t \mid \Theta)$$
$$\psi_t^A = \psi^A(S_t \mid \Theta) \approx \psi_{NN}^A(S_t \mid \Theta)$$

2. **Construct a loss**: the weighted mean of squared Euler / equilibrium residuals

3. **Train** by stochastic optimization: minimize the loss, then **simulate the model forward** for new state points

The bottleneck for estimation

Training from scratch for every parameter vector Θ is far too costly for estimation.

Extended policy functions: parameters as pseudo-states

The **same recipe**, now with the parameters as inputs ($\bar{\Theta}$ fixed, $\tilde{\Theta}$ to estimate):

0. As before, a finite but large number L of agents

1. **Parameterize** the policy functions with *extended* deep NNs:

$$\psi_t^i = \psi^I(S_t^i, S_t, \tilde{\Theta} \mid \bar{\Theta}) \approx \psi_{NN}^I(S_t^i, S_t, \tilde{\Theta} \mid \bar{\Theta})$$
$$\psi_t^A = \psi^A(S_t, \tilde{\Theta} \mid \bar{\Theta}) \approx \psi_{NN}^A(S_t, \tilde{\Theta} \mid \bar{\Theta})$$

2. **Construct a loss**, as before

3. **Train** as before, now also drawing new parameters $\tilde{\Theta}$ during training

Extended policy functions: parameters as pseudo-states

The **same recipe**, now with the parameters as inputs ($\bar{\Theta}$ fixed, $\tilde{\Theta}$ to estimate):

0. As before, a finite but large number L of agents

1. **Parameterize** the policy functions with *extended* deep NNs:

$$\psi_t^i = \psi^I(S_t^i, S_t, \tilde{\Theta} \mid \bar{\Theta}) \approx \psi_{NN}^I(S_t^i, S_t, \tilde{\Theta} \mid \bar{\Theta})$$
$$\psi_t^A = \psi^A(S_t, \tilde{\Theta} \mid \bar{\Theta}) \approx \psi_{NN}^A(S_t, \tilde{\Theta} \mid \bar{\Theta})$$

2. **Construct a loss**, as before

3. **Train** as before, now also drawing new parameters $\tilde{\Theta}$ during training

Amortize the cost

Train the networks **once**, then evaluate the model for **any** parameters without re-solving.

Neural Network Particle Filter: a cheap likelihood

- For nonlinear models the likelihood is obtained with a **particle filter**
  - The model is evaluated for thousands of particles over many periods
  - The particle filter becomes the **bottleneck** for estimation

Train a neural network to map parameters to the log-likelihood

$$\Theta \longmapsto \log \mathcal{L}(\Theta)$$

- Build a dataset of parameter draws and particle-filter log-likelihoods (Sobol)
- Split into training / validation samples; train the NN, validate to avoid overfitting

Benefits

A single likelihood evaluation is almost instantaneous; it smooths out filtering noise; and it enables fast Bayesian (RWMH) estimation.

Proof of the pudding is in the eating

Four proofs of concept in the paper:

1. **Linearized 3-equation RANK**: analytical solution

   Compare with the closed-form solution $\longrightarrow$ *they coincide*

2. **RANK with a ZLB**: nonlinear, occasionally binding

   Solve with a standard global method and estimate $\longrightarrow$ *same posterior*

3. **One-asset HANK** (Auclert et al. 2021)

   Match the Sequence-Space Jacobian: distribution, policies, IRFs $\longrightarrow$ *handles heterogeneity*

4. **Quantitative HANK**: a model with capital and 7 aggregate shocks

   Recover parameters from simulated data $\longrightarrow$ *scales to more shocks and parameters*

...and an empirical application: a nonlinear HANK with a ZLB, estimated on U.S. data (1990–2019).

A Model We Can Check by Hand

Linearized three-equation NK model

Small linearized New-Keynesian model with a TFP shock:

$$\hat{X}_t = E_t\hat{X}_{t+1} - \sigma^{-1}(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t - E_t\hat{\Pi}_{t+1} - \hat{R}^*_t) \quad \text{(IS)}$$

$$\hat{\Pi}_t = \kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1} \quad \text{(NKPC)}$$

$$\hat{R}^*_t = \rho_A\hat{R}^*_{t-1} + \sigma(\rho_A - 1)\omega\sigma_A\epsilon^A_t$$

where $\hat{X}_t$ is the output gap, $\hat{\Pi}_t$ inflation, and $\hat{R}^*_t$ the natural rate.

Closed-form solution (to check against):

$$\hat{X}_t = \frac{1-\beta\rho_A}{\Delta}\,\hat{R}^*_t, \qquad \hat{\Pi}_t = \frac{\kappa}{\Delta}\,\hat{R}^*_t, \qquad \Delta \equiv (\sigma(1-\rho_A)+\phi_Y)(1-\beta\rho_A)+\kappa(\phi_\Pi-\rho_A)$$

Why this example?

Simple enough to **explain the method** concretely, and with a **closed-form solution**.

Solving it with extended policy functions

0. **Parametrize** the controls with one deep network (1 state $+$ 8 parameters $=$ **9 inputs**):

$$\begin{pmatrix} \hat{X}_t \\ \hat{\Pi}_t \end{pmatrix} = \psi(\hat{R}_t^*, \tilde{\Theta}) \approx \psi_{\text{NN}}(\hat{R}_t^*,\ \beta, \sigma, \eta, \varphi, \phi_{\Pi}, \phi_Y, \rho_A, \sigma_A)$$

1. **Residuals** of the IS and Phillips curves, ERR_{IS} and ERR_{NKPC}

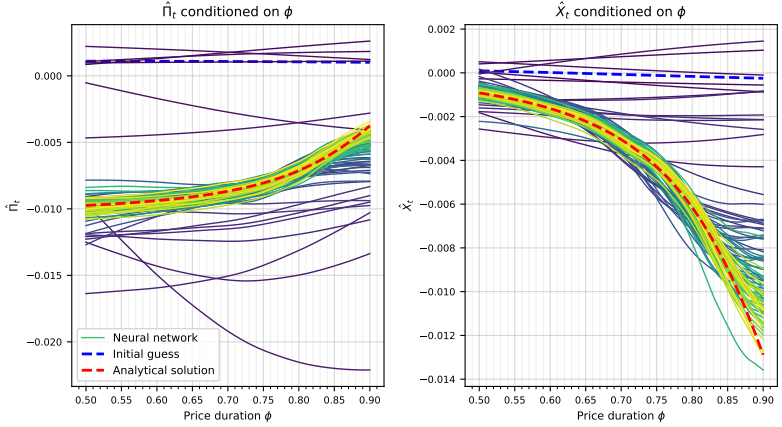
2. **Loss**, the weighted mean of squared residuals:

$$\mathcal{L} = w_1 \frac{1}{B} \sum_{i=1}^{B} \left(\text{ERR}_{\text{IS}}^i\right)^2 + w_2 \frac{1}{B} \sum_{i=1}^{B} \left(\text{ERR}_{\text{NKPC}}^i\right)^2$$

3. **Train** by drawing parameters, simulating $\hat{R}_t^*$, and taking Adam steps

training loss

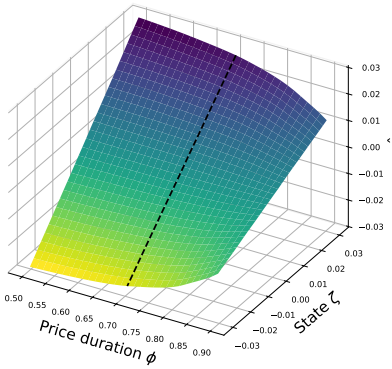
Watching the policy function converge



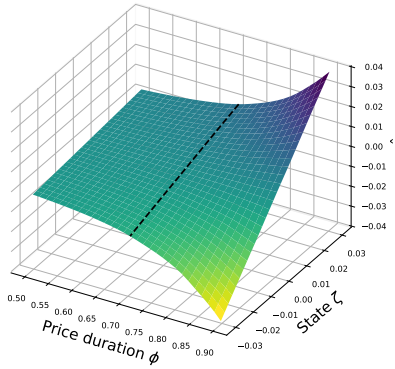
Inflation (left) and the output gap (right) against price duration ϕ, at successive training stages (dark to light): from the flat initial guess (blue) onto the analytical solution (red).

One network to rule the whole parameter space

Policy function $\hat{\Pi}_t$ conditioned on ϕ and ζ

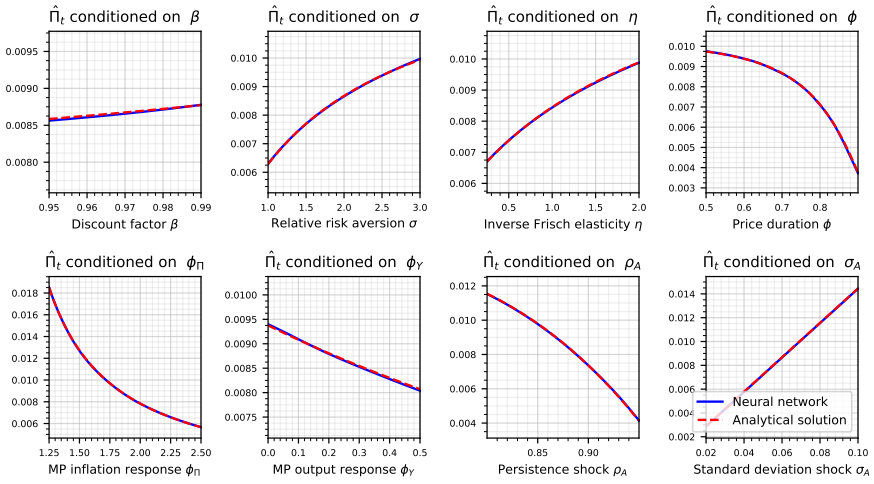


Policy function $\hat{X}_t$ conditioned on ϕ and ζ

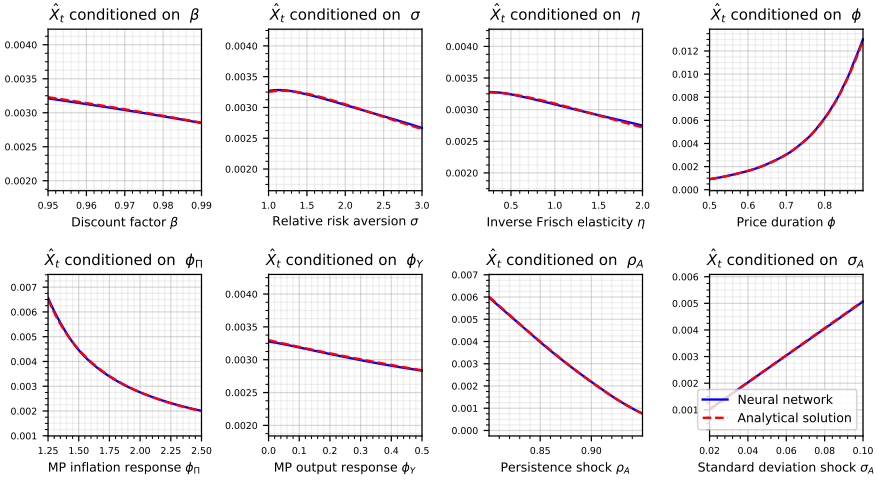


Extended policy functions for inflation $\hat{\Pi}_t$ (left) and the output gap $\hat{X}_t$ (right), over states *and* parameters, from a **single** trained network. Black dashed line: a single solution.

Inflation matches the closed form



Output gap matches the closed form



Deep learning for estimation

Evaluating the likelihood with a particle filter is the bottleneck

The model is run for thousands of particles over many periods.

Deep learning for estimation

Evaluating the likelihood with a particle filter is the bottleneck

The model is run for thousands of particles over many periods.

Neural Network Particle Filter: learn $\Theta \mapsto \log \mathcal{L}(\Theta)$

1. Generate a dataset of parameter draws and log-likelihoods — sampling
2. Train the NN on it; validate to avoid overfitting

Deep learning for estimation

Evaluating the likelihood with a particle filter is the bottleneck

The model is run for thousands of particles over many periods.

Neural Network Particle Filter: learn $\Theta \mapsto \log \mathcal{L}(\Theta)$

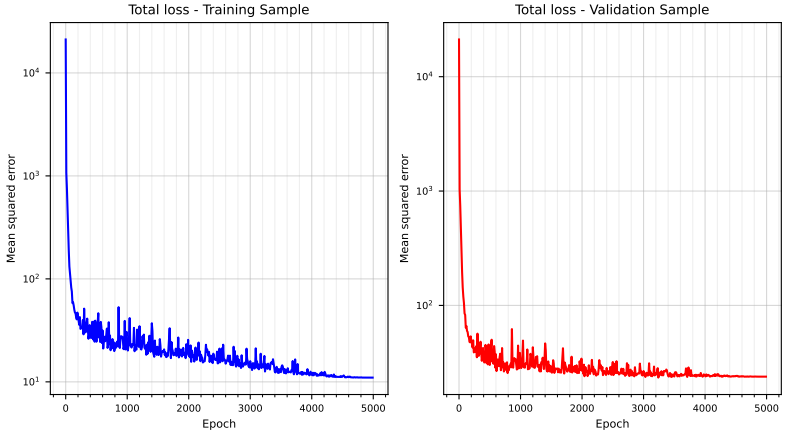
1. Generate a dataset of parameter draws and log-likelihoods
2. Train the NN on it; validate to avoid overfitting

sampling

Benefits

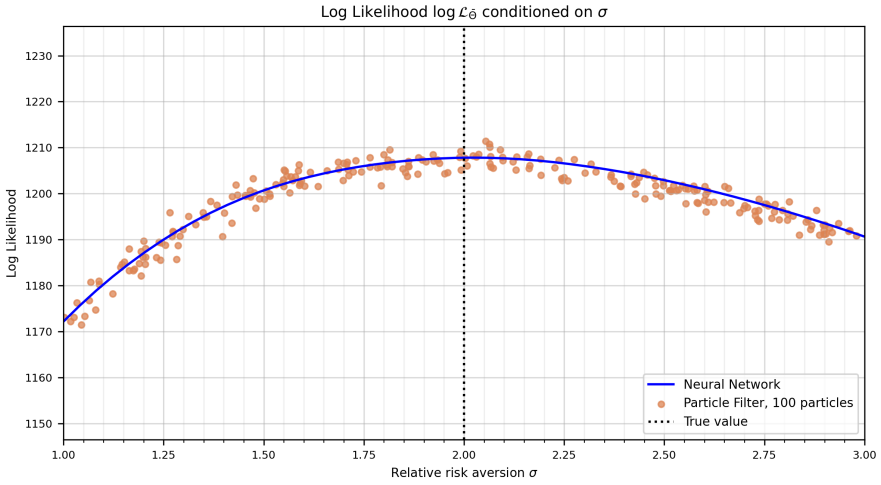
- ⚡ Single likelihood evaluation becomes almost instantaneous
- 💊 Smooths out noise from filtering
- 🔁 Enables efficient Bayesian estimation

Let's check the training and validation losses



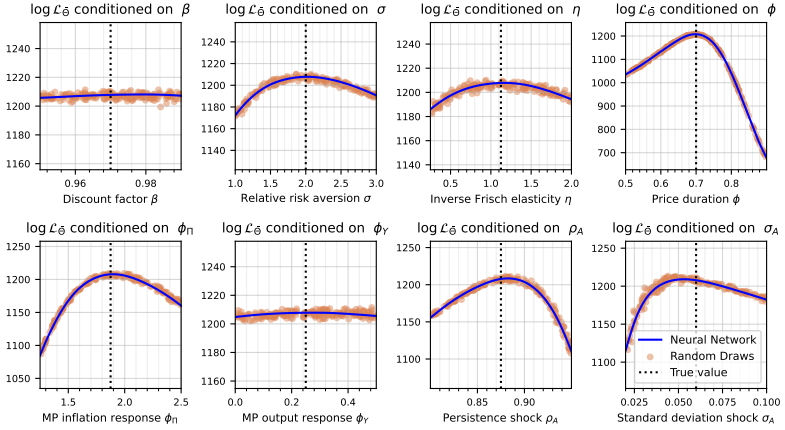
Validation loss tracks the training loss — no overfitting, despite the deliberately noisy 100-particle filter.

Cut through the noise 🎧



The NN smooths the particle-filter noise into a clean likelihood.

Recovering the truth 🔎



The approximated likelihood peaks at the **true parameter values** across the board.

HANK Model with a ZLB

A nonlinear HANK model with a ZLB

- **Households** face **idiosyncratic income risk** and a **borrowing limit**
  - Incomplete markets: precautionary saving and a non-degenerate wealth distribution
- **Firms**: competitive **final-good** producers and monopolistically competitive **intermediate** firms with **Rotemberg** price adjustment costs
- **Aggregate shocks**: preference, growth, and monetary policy
- **Monetary policy** follows a Taylor rule, constrained by the **zero lower bound**

Solving the HANK model

The same algorithm, now at scale:

0. **Discretize:** continuum $\to L = 100$ agents (≈ 213 states: 200 individual, 3 aggregate, 10 parameters)
1. **Parameterize** the policies with **split-input** networks (outputs: individual labor; aggregate inflation, wage)
2. **Loss:** the squared equilibrium residuals (Euler and borrowing limit, NKPC, market clearing)
3. **Train** by stochastic optimization, in two steps (transfer learning $+$ shock curriculum)

architecture

Two-step training

Monetary policy follows a Taylor rule, bounded by the ZLB:

$$R_t = \max\big[\,1,\ R\,(\Pi_t/\Pi)^{\theta_\Pi}(Y_t/Z_tY)^{\theta_Y}\exp(mp_t)\,\big]$$

The deterministic steady-state R and Y enter the rule, so we solve in two steps:

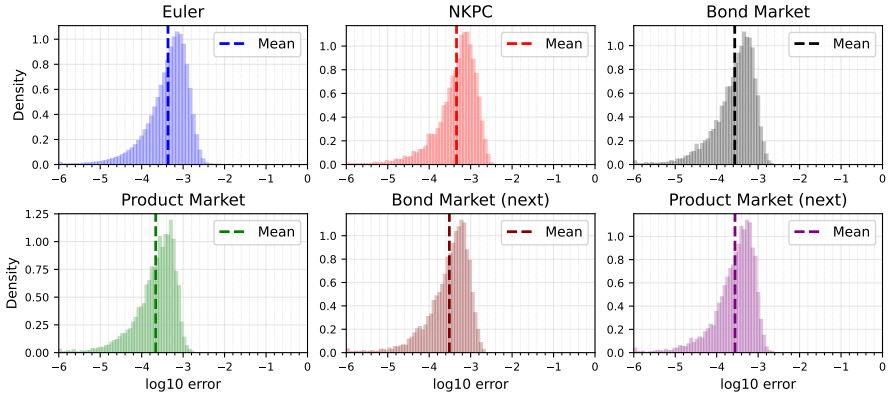
1. **Deterministic steady state** (no aggregate shocks): networks for R, Y and household labor
2. **Full nonlinear HANK with the ZLB**:
   - **transfer-learn** the household network from the DSS,
   - add the **aggregate** network,
   - and **introduce shocks gradually** (idiosyncratic $\rightarrow$ aggregate $\rightarrow$ ZLB)

Amortized solution

Trained **once** over the chosen parameter ranges, so we can simulate the model for any value of the **estimated** parameters in those ranges, without re-solving.

Solution accuracy

across parameters



Distribution of the absolute residual of each equilibrium condition ($\log_{10}$ scale) across simulated states.

Estimating on U.S. data

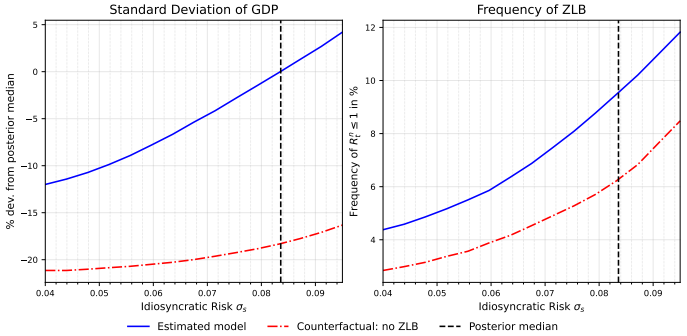
- **Sample**: 1990Q1–2019Q4 (120 quarters); observables: output growth, inflation, the policy rate (measurement error $=$ 10% of each series' variance)
- **Neural network particle filter** (the 2nd innovation):
  - $N = 10{,}000$ Sobol draws from parameter space,
  - compute likelihood with a bootstrap particle filter ($P = 10{,}000$);
  - train the surrogate $\Theta \mapsto \log \mathcal{L}(\Theta)$
- **RWMH**: one chain of 5M draws (100k burn-in)

We estimate 10 structural parameters

Including parameters that govern the household problem and the steady state (idiosyncratic risk and the borrowing limit).

posteriors table

How the ZLB and inequality interact



Idiosyncratic and aggregate interactions

Higher idiosyncratic risk $\longrightarrow$ more precautionary saving $\longrightarrow$ **lower interest rates** $\longrightarrow$ the **ZLB binds more often** $\longrightarrow$ **amplified output volatility**

- Output volatility rises with idiosyncratic risk, much more so with the ZLB *(left)*
- Higher idiosyncratic risk makes ZLB episodes more frequent *(right)*

Conclusion

Two innovations

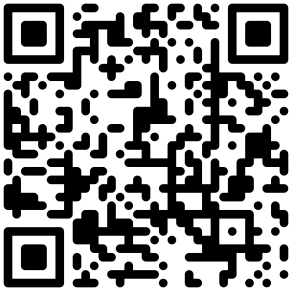
1. **Extended policy functions**: treat parameters as pseudo-states, solve once for all parameters
2. **Neural network particle filter**: fast, smooth likelihood evaluations

- Together they let us **solve and estimate** nonlinear heterogeneous-agent models with aggregate risk and occasionally-binding constraints (like the ZLB)
- We can now take such models to the data, and study how nonlinearity and heterogeneity interact for policy

Thank you!

Code & tutorial notebook (analytical example):

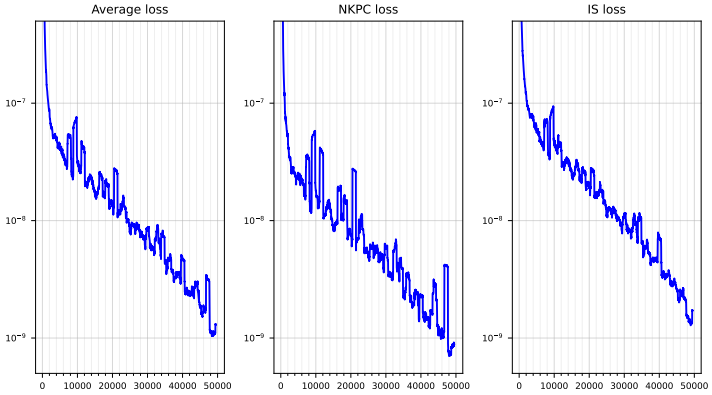
https://github.com/tseep/estimating-hank-nn



Appendix

Analytical example: training loss

back

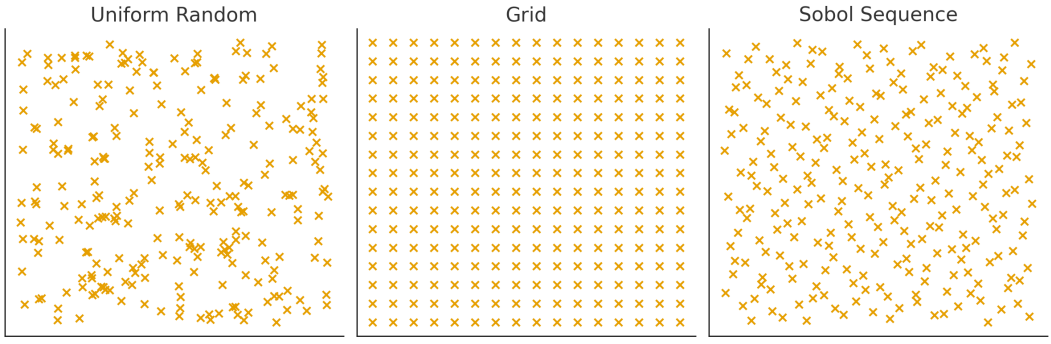


Mean squared residual (MSE): the average loss and its two components (NKPC, IS), over 50,000 iterations, down to the order of 10^{-9}.

Creating the dataset

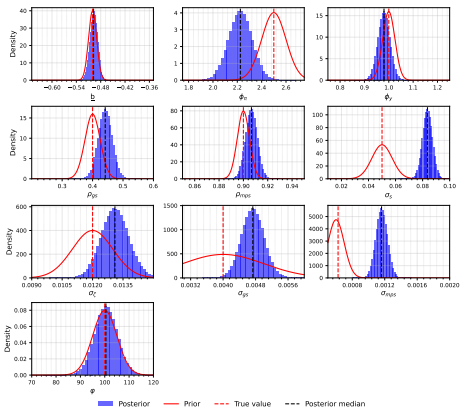
back

- Draw parameters with a **Sobol sequence** to cover the parameter space efficiently
- Run the particle filter at each draw to evaluate the (noisy) log-likelihood
- Split into training / validation sets, then fit the NN surrogate



Posterior distributions

back



Posterior densities (blue) against the priors (red); vertical lines: posterior median (black) and the baseline calibration (red dashed; labelled “true value’ ’ in the figure).

Estimation results

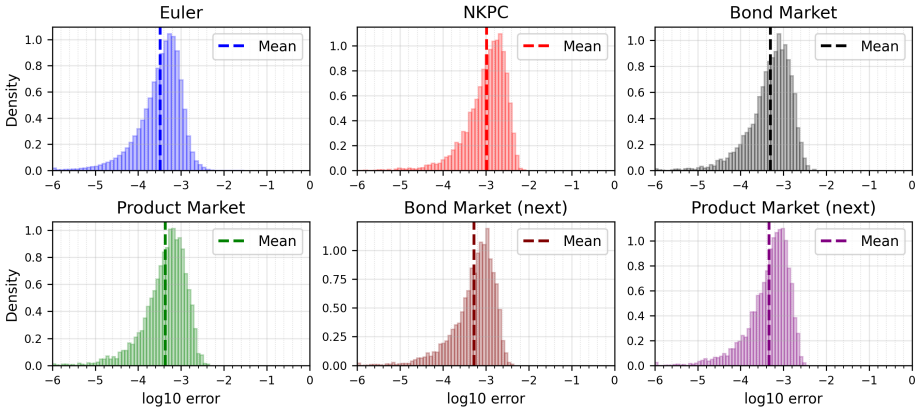
back

Parameter	Prior mean	Prior bounds	Post. mean	Post. [5%, 95%]
Borrowing limit $\underline{b}$	-0.50	$[-0.65, -0.35]$	-0.50	$[-0.51, -0.48]$
MP inflation ϕ_Π	2.50	$[1.75, 2.75]$	2.22	$[2.07, 2.38]$
MP output ϕ_Y	1.00	$[0.75, 1.25]$	0.98	$[0.94, 1.02]$
Persistence growth ρ_g	0.40	$[0.20, 0.60]$	0.44	$[0.40, 0.48]$
Persistence MP ρ_m	0.90	$[0.85, 0.95]$	0.91	$[0.90, 0.91]$
Idiosyncratic risk σ_s	0.050	$[0.010, 0.100]$	0.084	$[0.077, 0.090]$
Std. preference σ_ζ	0.012	$[0.009, 0.015]$	0.013	$[0.012, 0.014]$
Std. growth σ_g	0.0040	$[0.0030, 0.0060]$	0.0047	$[0.0043, 0.0052]$
Std. MP σ_m	0.0006	$[0.0005, 0.0020]$	0.0012	$[0.0010, 0.0013]$
Rotemberg cost φ	100	$[70, 120]$	100.5	$[92.8, 108.1]$

Truncated-normal priors; posterior from 5M RWMH draws (100k burn-in).

Solution accuracy across the parameter space

back



Absolute residuals ($\log_{10}$ scale) pooled across parameter draws spanning the estimated range, by equilibrium condition. The extended policy network stays accurate across the whole parameter space.

Network architecture and training

back

Policy networks (DSS rate & output, household labor, aggregate inflation & wage):

- each input block (individual state, distribution, aggregate state, parameters) is normalized and **projected separately** to 128 units, then **summed**
- 4 hidden layers of 128 units, CELU activation

Training: AdamW with cosine-annealing learning rate and gradient clipping; expectations via antithetic Monte-Carlo shocks; states sampled by simulating forward; a **curriculum** ramps idiosyncratic shocks, then aggregate, then the ZLB

NN particle-filter surrogate: MLP, 128 units, 4 layers, CELU; 1000 epochs, 90/10 train/validation