Accompanying Material: Learning Guide for "Estimating Nonlinear Heterogeneous Agent Models with Neural Networks"*

Hanno Kase
European Central Bank

Leonardo Melosi
European University Institute, CEPR

Matthias Rottner
Deutsche Bundesbank

August 18, 2026

1 Introduction

This accompanying material provides a step-by-step learning guide on how to use and extend our code, enabling researchers to solve and estimate nonlinear macroeconomic models using neural networks (NNs). It includes an open-source Python package together with illustrative example codes. The underlying conceptual framework is presented in our paper "Estimating Nonlinear Heterogeneous Agent Models with Neural Networks." When using the code or this guide, please cite the main paper as Kase et al. (2025).

The learning guide introduces our methods using four models with varying complexity:

1. Representative Agent New Keynesian (RANK) model
2. Heterogeneous Agent model based on Krusell and Smith (1998)
3. Two-asset extension of the Heterogenous Agent model
4. Heterogeneous Agent New Keynesian (HANK) model

The use of different models serves two purposes. First, using several models allows us to provide a clear, incremental learning path for researchers interested in solving and estimating nonlinear heterogeneous agent models with neural networks. For instance, the RANK model is substantially faster to solve and does not require access to a GPU. Thus, this model is very well suited to introduce the key features of our solution approach and the neural network setup. However, the inclusion of heterogeneity requires some adaptations to the code, making the Heterogeneous Agent (HA) model examples essential. Second, we provide the code for these models as a starting point for other researchers who wish to work with neural networks. For this reason, we focus on these seminal models.

The learning guide starts with the RANK model introduced in the first proof of concept of our paper, which is a linearized three-equation New Keynesian Model. We then extend

*Emails: hanno.kase@ecb.europa.eu, leonardo.melosi@eui.eu, matthias.rottner@bundesbank.de. The views in this note are those of the authors and should not be interpreted as reflecting the views of the Deutsche Bundesbank, the European Central Bank, or any other person associated with the Eurosystem.

the model step by step to explain the key features of our algorithm. We also vary the setup of the neural networks. Specifically, we cover the following changes: i) policy functions, ii) parameters, iii) endogenous state variables, iv) shocks, v) pseudo-state variables, vi) observation equation, vii) transformation of the observation equation, viii) options and settings of the neural network for the model solution, ix) options and settings of the neural network for the particle filter, and x) computation on CPU and GPU. We provide the code for the baseline model and for all adjustments.

The next model is the Heterogeneous Agent model based on Krusell and Smith (1998). In particular, we follow the specification of Auclert et al. (2021). We use this model to demonstrate how to incorporate heterogeneous agents in the model and how to specify the different neural networks. To solve this model, we adopt a two-step approach. First, we solve for the deterministic steady state of this economy and present key statistics such as the asset distribution and the individual policy functions. We also discuss the approximation error. Second, we solve the economy with aggregate risk, again computing key statistics, including impulse response functions and approximation error. Finally, we compare our results with those obtained using the Sequence Space Jacobian (SSJ) approach (Auclert et al., 2021).

We then extend the one-asset Heterogeneous Agent model based on Krusell and Smith (1998) to include a second asset, allowing households to hold both capital and government debt. This extension illustrates how our framework can accommodate multiple assets and how the corresponding neural-network architecture can be adapted. We deliberately adopt a parsimonious specification to make the implementation and potential extensions of the code as transparent as possible. As in the one-asset model, we first solve for the deterministic steady state and report key features of the solution. We then solve the model with aggregate risk and evaluate the resulting dynamics.

We conclude our learning guide with a canonical one-asset HANK model. As before, we follow the specifications of Auclert et al. (2021) to ensure comparability. The model is solved in two steps. First, we solve for the deterministic steady state of the economy. Second, we solve for the equilibrium with aggregate risk. Similar to the Krusell-Smith model, we present the asset distribution, individual policy functions, approximation errors, and impulse response functions. We then compare our model solution with the Sequence Space Jacobian under two alternative scenarios: one with low aggregate risk and another with substantially higher aggregate risk.

The remainder of this document is organized as follows. Section 2 describes the programming language and provides information on how to access the code. Section 3 uses the RANK model to provide a step-by-step learning guide. Section 4 introduces and explains the solution of the Krusell-Smith model. Section 5 discusses the HANK model and its solution. Section 6 concludes.

2 Code

The code is written in Python and uses the machine learning library PyTorch. The code can be downloaded either on GitHub or directly used as a notebook within Google Colab using the links below. While Colab requires a Google account, the code can be run directly there without other setup costs.

GitHub: To be added upon publication.

Google Colab: To be added upon publication.

3 Linearized Representative Agent New Keynesian (RANK) Model

The code is provided for the first proof of concept in the paper: a linearized small-scale DSGE model. The model is solved and estimated using neural networks. The model is then extended step-by-step to introduce new features that explain the structure of the code. Afterward, we vary various modifications to introduce the user to a set of possibilities to alter the code.[1]

Baseline Model

We consider for illustrational purposes an off-the-shelf three-equation New Keynesian model with only one shock (TFP shock). The model is log-linearized around its unique steady-state equilibrium, and the resulting equations are:

$$\hat{X}_t = E_t\hat{X}_{t+1} - \sigma^{-1}\left(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t - E_t\hat{\Pi}_{t+1} - \hat{R}_t^*\right), \tag{1}$$

$$\hat{\Pi}_t = \kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}, \tag{2}$$

$$\hat{R}_t^* = \rho_A\hat{R}_{t-1}^* + \sigma(\rho_A - 1)\omega\sigma_A\epsilon_t^A, \tag{3}$$

where output is defined as $\hat{Y}_t = (Y_t - Y)/Y$, inflation as $\hat{\Pi}_t = \Pi_t - \Pi$, and $\hat{R}_t^*$ is the natural rate of interest, which follows an exogenous process that is derived from the TFP process.

Neural Network Solution. We can map the linearized NK model in the general form of the solution method outlined in section 2.2 of the main paper. The state variable, the structural shock, and the control variables of the model are:

$$\mathbb{S}_t = \left\{\hat{R}_t^*\right\}, \quad \text{and} \quad \nu_t = \left\{\epsilon_t^A\right\}, \quad \text{and} \quad \psi_t = \left\{\hat{X}_t, \hat{\Pi}\right\}. \tag{4}$$

The parameters of the model are divided into the estimated set ($\tilde{\Theta}$) and the calibrated

[1]The code was tested with an Apple M4 Pro chip featuring a 14-core CPU and a 20-core GPU, with an AMD Ryzen 5 9600X 6-core CPU and Nvidia RTX 3090 GPU, and with NVIDIA RTX PRO 6000 Blackwell Server Edition (via Google Colab).

set ($\bar{\Theta}$):

$$\tilde{\Theta} = \{\beta, \sigma, \eta, \phi, \theta_\Pi, \theta_Y, \rho_A, \sigma_A\}, \tag{5}$$

$$\bar{\Theta} = \varnothing \tag{6}$$

Note that, for the moment, all parameters are estimated so that the set of calibrated parameters is empty. Extension 5, which covers pseudo-state variables, will relax this assumption.

The NN ψ_{NN} is trained to determine the output gap and inflation:

$$\begin{pmatrix} \hat{X}_t \\ \hat{\Pi}_t \end{pmatrix} = \psi_{NN}\left(\mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right). \tag{7}$$

We minimize the error in the following two equations

$$F_1 = \hat{X}_t - \left(E_t\hat{X}_{t+1} - \sigma^{-1}\left(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t - E_t\hat{\Pi}_{t+1} - \hat{R}_t^*\right)\right), \tag{8}$$

$$F_2 = \hat{\Pi}_t - \left(\kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}\right). \tag{9}$$

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.1 Extension 1: Policy Functions

We are interested in extending the number of policy functions. For this reason, we explicitly want to have the nominal interest rate as a policy function. Therefore, the Taylor rule is now part of the equations (instead of substituting it in the Euler equation). The system of equations can be written as:

$$\hat{X}_t = E_t\hat{X}_{t+1} - \sigma^{-1}\left(\hat{I}_t - E_t\hat{\Pi}_{t+1} - \hat{R}_t^*\right), \tag{10}$$

$$\hat{\Pi}_t = \kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}, \tag{11}$$

$$\hat{I}_t = \phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t, \tag{12}$$

$$\hat{R}_t^* = \rho_A\hat{R}_{t-1}^* + \sigma(\rho_A - 1)\omega\sigma_A\epsilon_t^A, \tag{13}$$

where $\hat{I}_t$ is the nominal interest rate. This requires defining an additional policy function and adjusting the set of equations that needs to be minimized.

Neural Network Solution. To map the adjusted model to the general form, we need to adjust the following objects. The neural network now additionally learns the mapping of state and pseudo-state variables to the nominal interest rate $\hat{I}_t$:

$$\begin{pmatrix} \hat{X}_t \\ \hat{\Pi}_t \\ \hat{I}_t \end{pmatrix} = \psi_{NN}\left(\mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right). \tag{14}$$

We also now include the Taylor rule in the set of equations that we minimize:

$$F_1 = \hat{X}_t - \left(E_t\hat{X}_{t+1} - \sigma^{-1}\left(\hat{I}_t - E_t\hat{\Pi}_{t+1} - \hat{R}^*_t\right)\right), \tag{15}$$

$$F_2 = \hat{\Pi}_t - \left(\kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}\right), \tag{16}$$

$$F_3 = \hat{I}_t - \left(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t\right). \tag{17}$$

All the other objects are unchanged.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.2 Extension 2: Parameters

We are interested in incorporating additional parameters. For this reason, we extend the model further by including an additional parameter in the Taylor rule, namely ρ_I. For the moment, the parameter enters as follows (the next extension includes the parameter in the usual approach):

$$\hat{X}_t = E_t\hat{X}_{t+1} - \sigma^{-1}\left(\hat{I}_t - E_t\hat{\Pi}_{t+1} - \hat{R}^*_t\right), \tag{18}$$

$$\hat{\Pi}_t = \kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}, \tag{19}$$

$$\hat{I}_t = \rho_I \cdot 0 + (1-\rho_I)\left(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t\right), \tag{20}$$

$$\hat{R}^*_t = \rho_A\hat{R}^*_{t-1} + \sigma(\rho_A - 1)\omega\sigma_A\epsilon^A_t, \tag{21}$$

We set the lower bound to -0.1 and the upper bound to 0.1 for ρ_I. Note that we change the upper and lower bound in the next extension.

Neural Network Solution. The only change is for the set of parameters. Note that we order the new parameter ρ_I for convenience before the parameters of the shock process:

$$\tilde{\Theta} = \{\beta, \sigma, \eta, \phi, \theta_\Pi, \theta_Y, \rho_I, \rho_A, \sigma_A\}. \tag{22}$$

All the other objects are unchanged. Note that we do not display the analytical solution anymore to compare the graphs, as we derived the analytical solution without a persistent Taylor rule.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.3 Extension 3: Endogenous State Variables

The next step is to extend the number of endogenous state variables. For this reason, we add the lagged interest rate in the Taylor rule:

$$\hat{X}_t = E_t\hat{X}_{t+1} - \sigma^{-1}\left(\hat{I}_t - E_t\hat{\Pi}_{t+1} - \hat{R}_t^*\right), \tag{23}$$

$$\hat{\Pi}_t = \kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}, \tag{24}$$

$$\hat{I}_t = \rho_I \cdot I_{t-1} + (1-\rho_I)\left(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t\right), \tag{25}$$

$$\hat{R}_t^* = \rho_A\hat{R}_{t-1}^* + \sigma(\rho_A - 1)\omega\sigma_A\epsilon_t^A, \tag{26}$$

Note that the bounds of ρ_I should be adjusted to 0.0 for the lower bound 0.75 for the upper bound.

Neural Network Solution. The state variables are changing. The interest rate is an additional endogenous state variable:

$$\mathbb{S}_t = \left\{\hat{I}_{t-1}, \hat{R}_t^*\right\} \tag{27}$$

All the other objects are unchanged.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.4 Extension 4: Shocks (Exogenous State Variables)

The next step is to include an additional shock (exogenous state variable). In particular, we include for demonstration purposes a monetary policy shock $\hat{mp}_t$ in the Taylor rule:

$$\hat{X}_t = E_t\hat{X}_{t+1} - \sigma^{-1}\left(\hat{I}_t - E_t\hat{\Pi}_{t+1} - \hat{R}_t^*\right), \tag{28}$$

$$\hat{\Pi}_t = \kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}, \tag{29}$$

$$\hat{I}_t = \rho_I \cdot I_{t-1} + (1-\rho_I)\left(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t\right) + \hat{mp}_t, \tag{30}$$

$$\hat{R}_t^* = \rho_A\hat{R}_{t-1}^* + \sigma(\rho_A - 1)\omega\sigma_A\epsilon_t^A, \tag{31}$$

$$\hat{mp}_t = \epsilon_t^{MP}. \tag{32}$$

Note that we included equation (32) to allow for a more general shock process, as the distinction between mp_t and ϵ_t^{MP} in the current setup is not necessary. However, we think it is helpful to write it up this way for our framework.

Neural Network Solution. The state variables and shocks are changing:

$$\mathbb{S}_t = \left\{\hat{I}_{t-1}, \hat{R}_t^*, \hat{mp}_t\right\}, \quad \text{and} \quad \nu_t = \left\{\epsilon_t^A, \epsilon_t^{MP}\right\}, \quad \text{and} \quad \psi_t = \left\{\hat{X}_t, \hat{\Pi}\right\}. \tag{33}$$

All the other objects are unchanged.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.5 Extension 5: Pseudo-State Variables

We are interested in dividing the set of parameters into parameters to be calibrated (fixed value) and parameters to be estimated (pseudo-state variables). All parameters were initially chosen to be pseudo-state variables. We now choose that the inflation and output gap response is calibrated to the following fixed values: $\phi_\Pi = 1.875$ and $\phi_Y = 0.25$.

Neural Network Solution. The estimated parameter set ($\tilde{\Theta}$) and the calibrated parameter set ($\bar{\Theta}$) are now as follows:

$$\tilde{\Theta} = \{\beta, \sigma, \eta, \phi, \rho_I, \rho_A, \sigma_A\}, \tag{34}$$

$$\bar{\Theta} = \{\theta_\Pi, \theta_Y\} \tag{35}$$

All the other objects are unchanged.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.6 Extension 6: Variables in the Observation Equation

This extension concerns the observation equation for the particle filter. We have so far implicitly included all five variables determined in the model: $\hat{\Pi}_t, \hat{X}_t, \hat{I}_t, \hat{R}^*_t, \hat{mp}_t$. These data is observed with some measurement error). The measurement equation is then given as:

$$\begin{bmatrix} \text{Output Gap} \\ \text{Inflation} \\ \text{Interest Rate} \\ \text{Natural Rate of Interest} \\ \text{Monetary Policy Shock} \end{bmatrix} = \begin{bmatrix} \hat{X}_t \\ \hat{\Pi}_t \\ \hat{I}_t \\ \hat{R}^*_t \\ \hat{mp}_t \end{bmatrix} + u_t, \tag{36}$$

where the measurement error follows a Gaussian distribution $u_t \sim \mathcal{N}(0, \Sigma_u)$. The variance of the measurement error for each time series is a fraction m_E of its own variance. We set $m_E = 0.1$.

In this extension, we are interested in only observing the output gap, inflation, and interest rate. The measurement equation is then given as:

$$\begin{bmatrix} \text{Output Gap} \\ \text{Inflation} \\ \text{Interest Rate} \end{bmatrix} = \begin{bmatrix} \hat{X}_t \\ \hat{\Pi}_t \\ \hat{I}_t \end{bmatrix} + u_t, \tag{37}$$

All the other objects are unchanged.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.7 Extension 7: Transformation of the Observation Equation

So far, we have used variables that we have already defined to solve the model. At the same time, estimation usually requires transforming the model variables to match them to the actual observed data. As a next step, we modify our code to incorporate the following observation equation:

$$\begin{bmatrix} \text{Output Gap} \\ \text{Inflation} \\ \text{Interest Rate} \end{bmatrix} = \begin{bmatrix} 100\hat{X}_t \\ 400\hat{\Pi}_t \\ 400\hat{I}_t \end{bmatrix} + u_t, \tag{38}$$

We also increase the measurement error so that $m_E = 0.25$.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.8 Extension 8: Options for the Neural Network Solution Step

We are now adapting various options, attributes, and hyperparameters for the training of the neural network for the extended policy functions. Note that the suggested changes here are not necessarily improvements or optimal choices. Instead, the idea is to show the vast array of options that are available to fine-tune the learning process. The focus is on the following points:

- **Activation function**: The default choice for the activation function is CELU (Continuously Differentiable Exponential Linear Unit):

  $$CELU(x) = \max(0, x) + \min(0, \alpha(\exp(x/\alpha) - 1)), \tag{39}$$

  where we set $\alpha = 1.0$. For more details, see Barron (2017).

However, the used activation function can be easily adapted to other common alternative functions such as ReLU (rectified linear unit function), PReLU (parameteric ReLU), sigmoid, SiLU (Sigmoid Linear Unit), Mish (self-regularized, non-monotonic activation function), Softlus, or Tanh (hyperbolic tangent), among others. The full list of options in PyTorch can be found at the PyTorch documentation under non-linear activation functions (https://pytorch.org/docs/stable/nn.html). Note that in the provided example code, we change the activation function to SiLU.

- **Width and depth**: The neural network specification has 64 neurons with 5 hidden layers in the baseline code. In this extension, we increase the number of neurons to 128 and decrease the number of hidden layers to 4. One way to cross-check if the neural network is sufficiently complex to capture the dynamics is to ensure that the results do not change when increasing the width and depth.
- **Optimizer**: Our default optimizer is AdamW (Loshchilov and Hutter, 2017), which is a modified version of the Adam (adaptive moment estimation) optimizer (Kingma and Ba, 2014). The AdamW optimizer modifies the weight decay relative to Adam. In the provided example code, we change the optimizer to Adam.
- **Loss function criterion**: The neural network minimizes a set of equations. The error in the equations can be evaluated by different criteria, such as the mean squared error. Our default loss function is the Huber loss, which is less sensitive to outliers than the mean squared error. The loss at batch b is evaluated as follows:

$$l_b = \begin{cases} 0.5(x_b - y_b), & \text{if } |x_b - y_b|, < \delta, \\ \delta(|x_b - y_b| - 0.5\delta), & otherwise. \end{cases} \tag{40}$$

  The parameter δ can be chosen. Note that for large values of δ, the criterion becomes close to the mean squared error. We now change the criterion from Huber loss to the mean squared error in the provided example code for this extension. Note that PyTorch allows for several different loss functions, e.g., SmoothL1Loss. See the PyTorch documentation for more information. In this extension, we change the setting to use manual weighting, assigning a weight of 1.0 to each equation in the loss function.
- **Policy function transformation**: The neural network provides an output at the last layer. We can transform this outcome. In our baseline code, we scale the output down by dividing it by 100. However, the output of the neural network could also be transformed using activation functions or constrained between some bounds. In the provided example code, we use hyperbolic tangent (TanH) to transform the neural network output. The function also bounds the output between -1.0 and 1.0. It is important to ensure that this transformation aligns with the underlying economic model. Note that an appropriate scaling/adjustment can help to find a solution.
- **Input normalization**: We normalize the input (our state variables) before feeding them into the neural network to facilitate the optimization.[2] While we initially set the

[2]We also normalize the parameters using their bounds. However, the scaling automatically uses the bounds

bounds of the state variables to $[-0.025, 0.025]$, we vary them in our adapted file to $[-1.0, 1.0]$

- **Number of iterations**: The default number of iterations is set to 10,000 in the example code. The main intention behind this choice is to get the neural network reasonably trained in a limited time. We now increase the training length to 20,000 iterations
- **Learning rate**: We impose a schedule for the learning rate, which is a tuning parameter for the optimization. Specifically, the learning rate determines the step size during each iteration. In our baseline, we use cosine annealing, which adjusts the learning rate from $1e-3$ to $1e-10$ using a cosine curve. In the modified version, we decay the learning rate by 0.1 for each 10,000 iterations using the StepLR scheduler.
- **Batch size** We initially use a batch size of 500. In the code for the extension, we lower the batch size to 100.
- **Draw new parameter values for pseudo-state variables** We redraw the parameter values for pseudo-state variables after 1 iteration. Note that, particularly for more complex models, there is a trade-off between varying the parameter values and ensuring to be in the right stochastic solution domain as it takes time to converge to the state space when simulating forward. In the provided example code, we increase the number to 10.
- **Stochastic solution domain and simulation steps:** We draw the state values of the parameters by simulating the model forward using our neural network solution. In the baseline code, we use 10 simulation steps. For the purpose of this exercise, we lower the steps to 5.
- **Monte Carlo integration**: We use Monte Carlo integration to approximate expectations. In the baseline version, we use 30 draws for the next period to approximate expectations. We alter this to 20 in this version. As we use the antithetic variate methods for drawing the shocks, the code requires an even number of draws.
- **Internal optimization steps** One option is to conduct several internal optimization steps for the expectations evaluation to increase precision. Specifically, we conduct several optimization rounds with new sets of drawn shocks to evaluate expectations while we fix the initial state variables. In our baseline model, we run five internal optimization steps. In the extended model, we reduce the internal optimization steps to 2.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

of the parameters, so we do not list this as an option here.

3.9 Extension 9: Options for the Filtering Step

We are now adapting various options, attributes, and hyperparameters of the setup and for the training of the neural network for the filtering step. Note that some options are the same as for the solution stop, for which we will keep the explanation short. As before, the suggested changes here are not necessarily improving or optimal. Instead, the idea here is to show the vast options that are available to fine-tune the learning process. The focus is on the following points:

- **Activation function**: While the activation function in the baseline is CELU, the extension uses the self-regularized, non-monotonic activation function Mish.
- **Width and depth**: We increase the number of neurons to 256 per layer and extend the number of hidden layers to 5.
- **Optimizer**: We modify the optimizer from AdamW to Adam, as in the previous extension.
- **Batch** We increase the batch size to 200 from 100.
- **Epoch** We lower the amount of training epochs, which is a complete pass through of the entire training data, to 3,000 (from 5,000 epochs in our baseline).
- **Size of draws** Initially, we have used 10,000 draws from the parameter space and calculated the likelihood using the particle filter. In this extension, we extend the number of draws to 15,000.
- **Size of validation sample** The sample of drawn points is divided into a training and validation sample, which helps to avoid overfitting. While we initially used 20% of our drawn points for the validation sample, we increase the percentage to 25% in this extension.
- **Learning rate**: As in the solution step, we alter the learning rate scheduler. We decay the learning rate by 0.1 each 1,000 epochs using the StepLR scheduler.

In addition to the neural network setup, there are a few options that are specific to the particle filter and the creation of the data:

- **Measurement error**: The measurement error u_t affects the observation equation and helps to avoid degeneracy of the likelihood. In this extension, we lower the variance of the measurement error m_E to 0.05.
- **Number of particles**: We change the number of particles to 250.
- **Length of data** We have worked initially with simulated data that covers 100 periods. We are increasing the length of the data to 125 periods.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

3.10 Extension 10: CPU and GPU

The code has been initially written to be solved with CPUs. The reason is that we want to make the initial code as widely accesible as possible. However to solve more challenging problems, it is necessary to have access to a decent GPU. We now adapt our code to work with GPUs, which are very commonly used in the deep learning. Note that the solution file for this extension is designed to give an error if there is no GPU. Our previous files allowed to continue with CPU even when GPU was specified.

Note that for MacOS devices with the Metal programming framework, PyTorch also offers the opportunity for GPU training directly on the Mac. While we observed that the speed is rather close to using the CPU of the MacBook in our specific applications, it is a great option for code testing.

Solution File.

- **GitHub:** To be added upon publication.
- **Google Colab:** To be added upon publication.

4 Heterogeneous Agent model based on Krusell and Smith (1998)

The next model in our learning guide is a heterogeneous agent model with aggregate risk. Specifically, it is the stochastic growth model with partially uninsurable income risk based on Krusell and Smith (1998). However, we abstract from cyclical income risk.

We solve the model using neural networks. In the benchmark specification, we use 100 agents ($L = 100$) to approximate the distribution. The exercise can also be conducted using a more parsimonious specification with fewer agents, e.g. $L = 25$. Under this specification, the model can be solved within a few hours on the CPU of a modern laptop, even when no GPU is available. The model itself is solved in two steps. Specifically, we first show how to solve the deterministic steady state. Afterwards, we solve the full equilibrium of the model. Finally, we compare the results to ones obtained when using the Sequence Space Jacobian approach of Auclert et al. (2021). For ease of comparison, we follow their setup of the model.

Solution File.

- **GitHub:** To be added upon publication.

4.1 Overview Model

The economy consists of a continuum of households. The households choose consumption C_t^i to maximize their utility:

$$E_0 \sum_{t=0}^{\infty} \beta^t \frac{\left(C_t^i\right)^{1-\sigma}}{1-\sigma}. \tag{41}$$

The budget constraint can be written as

$$C_t^i + K_t^i = (1 + r_t)K_{t-1}^i + w_t \bar{H} \exp(s_t^i), \tag{42}$$

where K_t^i is the household specific capital, r_t is the rental rate of capital and w_t is wage. Each household has a specific endowment of labor $\bar{H}$ that has an individual labor productivity s_t^i is stochastic and follows an AR(1) process: $s_t^i = \rho_s s_{t-1}^i + \epsilon_t^{s,i}$.

The agents can not have capital holdings below some limit, which implies:

$$K_t^i \geq \underline{K}. \tag{43}$$

The first-order conditions can be written as

$$1 = \beta \mathbb{E}_t \left[\left(\frac{C_{t+1}^i}{C_t^i} \right)^{-\sigma} (1 + r_{t+1}) \right] + \mu_t^i \tag{44}$$

where μ_t^i is the normalized Lagrange multiplier on the non-negativity constraint for individual capital holdings.

The representative firm produces output Y_t using total labor L_t and total capital K_t:

$$Y_t = (A_t) K_{t-1}^{\alpha} N_t^{\alpha}, \tag{45}$$

where total factor productivity A_t follows an AR(1) process: $A_t = (1 - \rho)A + \rho_A A_{t-1} + \epsilon_t^A$. The first-order conditions are

$$r_t = \alpha \frac{Y_t}{K_{t-1}} - \delta, \tag{46}$$

$$w_t = (1 - \alpha) \frac{Y_t}{N_t} \tag{47}$$

Note that market clearing for the labor market, capital market and goods market requires

$$N_t = \int \exp(s_t^i) \bar{H} di, \tag{48}$$

$$K_t = \int K_t^i di, \tag{49}$$

$$Y_t = \int C_t^i di + K_t - (1 - \delta) K_{t-1}. \tag{50}$$

Equilibrium conditions. The equilibrium conditions can be written as:

$$C_t^i + K_t^i = (1 + r_t)K_{t-1}^i + w_t \bar{H} \exp(s_t^i), \forall i, \tag{51}$$

$$K_t^i \geq \underline{K}, \forall i, \tag{52}$$

$$\lambda_t^i = \frac{1}{C_t^i}, \forall i, \tag{53}$$

$$1 = \beta \mathbb{E}_t \left[\left(\frac{C_{t+1}^i}{C_t^i} \right)^{-\sigma} (r_{t+1} + (1 - \delta)) \right] + \mu_t^i, , \forall i \tag{54}$$

$$r_t = \alpha \frac{Y_t}{K_{t-1}} - \delta \tag{55}$$

$$w_t = (1-\alpha)\frac{Y_t}{N_t}, \tag{56}$$

$$N_t = \int \exp(s_t^i)\bar{H}di, \tag{57}$$

$$K_t = \int K_t^i di, \tag{58}$$

$$Y_t = \int C_t^i di + K_t - (1-\delta)K_{t-1}, \tag{59}$$

$$s_t^i = \rho_s s_{t-1}^i + \epsilon_t^{s,i}, \forall i, \tag{60}$$

$$A_t = \rho_A A_{t-1} + \epsilon_t^A A_t = A + A_t \tag{61}$$

Calibration. We calibrate the parameters as follows: $\beta = 0.982$, $\sigma = 1.0$, $\alpha = 0.11$, $\delta = 0.025$, $\underline{K} = 0$ $\rho_s = 0.966$, $\sigma_s = 0.129$, $\rho_A = 0.9$, σ_A=0.02. We map the productivity level A to have an output level of 1 in the deterministic steady state. Note that this requires us to solve the DSS of the Krusell-Smith model, as we will explain below.

4.2 Mapping of the model to the general framework

The state variables and shocks of the stochastic growth model with partially uninsurable labor are given as:

$$\mathbb{S}_t = \left\{ \left\{ K_{t-1}^i \right\}_{i=1}^L, \left\{ s_t^i \right\}_{i=1}^L, A_t \right\}, \tag{62}$$

$$\nu_t = \left\{ \left\{ \epsilon_t^{s,i} \right\}_{i=1}^L, \epsilon_t^a \right\}. \tag{63}$$

The parameters of the model are divided into calibrated ($\bar{\theta}$) and estimated ones ($\tilde{\theta}$). For instance, in the case of our application with US data, this yields:

$$\bar{\Theta} = \{\beta, \sigma, \alpha, \delta, \rho_s, \sigma_s, \rho_A, \sigma_A, \underline{K}\}, \tag{64}$$

$$\tilde{\Theta} = \{\}. \tag{65}$$

In this specific application, we need a NN for the individual policy functions. The individual NN $\psi_{NN}^I(\cdot)$ solves for consumption:

$$\left\{ \left(C_t^i \right) = \psi_{NN}^I \left(\mathbb{S}_t^i, \mathbb{S}_t, \tilde{\Theta} | \bar{\Theta} \right) \right\}_{i=1}^L, \tag{66}$$

where $\mathbb{S}_t^i = \left\{ B_{t-1}^i, s_t^i \right\}$. We also have a neural network that provides the productivity level and government bond price at the deterministic steady state:

$$\left(A \right) = \psi_{NN}^{SS} \left(\tilde{\Theta} | \bar{\Theta} \right). \tag{67}$$

4.3 NN-based Solution Algorithm for the Deterministic Steady State

We first solve the deterministic steady state. We use this step to calibrate the productivity level of the model to match an output level of 1 in the DSS.

The DSS NN solves for the deterministic steady state values:

$$\left(A\right) = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{68}$$

To solve for this NN, we also need to solve for the individual policy functions of the agents in the DSS:

$$\left\{\left(C_t^i\right) = \psi_{NN}^{I,DSS}\left(\mathbb{S}_t^i, \mathbb{S}, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}. \tag{69}$$

where the superscript DSS denotes this as the inidivual policy functions for the DSS. Note that we let the DSS values of the aggregate variables still enter the network. While this is unnecessary for solving the DSS, it allows us to use this trained NN as starting point when we solve the full equilibrium with aggregate risk. Note that the aggregate state variable $\mathbb{S}$ is, of course, constant.

The steps of the algorithm are as follows:

1. Set up the NNs to approximate the policy functions and initialize their weights:

   The NN $\psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right)$ for the steady state parameters:

   $$\left(A\right) = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{70}$$

   The NN $\psi_{NN}^{I,DSS}\left(\mathbb{S}_t^i, \mathbb{S}, \tilde{\Theta}|\bar{\Theta}\right)$ for the individual policy functions

   $$\left\{\left(C_t^i\right) = \psi_{NN}^{I}\left(\mathbb{S}_t^i, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}. \tag{71}$$

2. Solve for all the current period individual variables. From the NN, we get a current guess for the deterministic steady value

   $$\left(A\right) = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{72}$$

   and individual policy functions:

   $$\{C_t^i\}_{i=1}^{L}. \tag{73}$$

   The next step is to calculate the following (aggregate) variables:

   $$C_t = \left(\frac{1}{L}\sum_{i=1}^{L} C_t^i)\right), \tag{74}$$

   $$K_{t-1} = \left(\frac{1}{L}\sum_{i=1}^{L} K_{t-1}^i)\right), \tag{75}$$

$$N_t = \left(\frac{1}{L} \sum_{i=1}^{L} \bar{H} \exp(s_t^i) \right), \tag{76}$$

$$Y_t = A K_{t-1}^{\alpha} N_t^{1-\alpha}, \tag{77}$$

$$r_t = \alpha \frac{Y_t}{K_{t-1}} - \delta, \tag{78}$$

$$w_t = (1-\alpha) \frac{Y_t}{N_t}, \tag{79}$$

where we use a subscript t here to indicate variables calculated based on individual shock realizations.

We pursue calculating each household's individual assets:

$$\left\{ K_t^i = (1+r) K_{t-1}^i + w \bar{H} \exp(s_t^i) - C_t^i \right\}_{i=1}^{L}. \tag{80}$$

We use a Monte Carlo approach to evaluate the expectations. We randomly draw M sets of next period individual shocks to evaluate the expectations. We use the antithetic variate method to increase the precision and reduce the number of necessary draws. The antithetic variates technique creates to a given path $\{\nu_1, \nu_2, \nu_3, \ldots, \nu_{M/2}\}$ also its antithetic path $\{-\nu_1, -\nu_2, -\nu_3, \ldots, -\nu_{M/2}\}$. The drawn shocks are given as:

$$\left\{ \left\{ \epsilon_{t+1}^{s,i,m} \right\}_{i=1}^{L} \right\}_{m=1}^{M}, \tag{81}$$

where the last superscript m indicates which shock draw the next period value is associated with.

We then proceed to calculate the next period values of the stochastic state variables for all M draws, that is:

$$\left\{ \left\{ s_{t+1}^{i,m} = \rho_s s_t^i + \epsilon_{t+1}^{s,i,m} \right\}_{i=1}^{L} \right\}_{m=1}^{M} \tag{82}$$

which then gives us the state variables for all M shock draws.

We can then use the NNs for the individual and aggregate policy functions to calculate the individual and aggregate control variables:

$$\left\{ \left\{ \left(C_{t+1}^{i,m} \right) = \psi_{NN}^{I,DSS} \left(\mathbb{S}_{t+1}^{i,m}, \mathbb{S}^m, \tilde{\Theta} | \bar{\Theta} \right) \right\}_{i=1}^{L} \right\}_{m=1}^{M} \tag{83}$$

The next step is to calculate the following (aggregate) variables for the period $t+1$ for all M draws:

$$K_t = \left(\frac{1}{L} \sum_{i=1}^{L} K_t^i \right), \tag{84}$$

$$\left\{N_{t+1}^m = \left(\frac{1}{L}\sum_{i=1}^{L} \bar{H}\exp(s_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{85}$$

$$\left\{Y_{t+1}^m = AK_t^{\alpha}(N_{t+1}^m)^{1-\alpha}\right\}_{m=1}^{M}, \tag{86}$$

$$\left\{r_{t+1}^m = \alpha\frac{Y_{t+1}^m}{K_t} - \delta\right\}_{m=1}^{M}, \tag{87}$$

$$K_{t+1} = Y_{t+1} - C_{t+1} \tag{88}$$

We need to ensure that the borrowing constraint for households $K_t^i \geq \underline{K}$ is satisfied.[3] It is convenient to define $\bar{\lambda}_t^i$ as follows:

$$\bar{\lambda}_t^i = 1 - \mu_t^i. \tag{89}$$

We calculate the multiplier for all agents L using their Euler equation:

$$\left\{\bar{\lambda}_t^i = \beta\frac{1}{M}\sum_{m=1}^{M}\left[\left(\frac{C_t^i}{C_{t+1}^{i,m}}\right)^{\sigma}(r_{t+1}^m + 1)\right]\right\}_{i=1}^{L}, \tag{90}$$

where we can now evaluate the expectations by taking the mean of the M draws.

Equipped with this object, we use the Fischer-Burmeister function Ψ^{FB}, a smooth way of representing the Kuhn-Tucker conditions, to ensure that these conditions hold.[4] This can be written as

$$\Psi^{FB}\left(\frac{1}{\bar{\lambda}_t^i} - 1, K_t^i - \underline{K}\right). \tag{91}$$

We can now calculate the error associated with the Fisher-Burmeister function:

$$\left\{L^{1,DSS,i} = \left(\Psi^{FB}\left(1 - \bar{\lambda}_t^i, B_t^i - \underline{B}\right)\right)^2\right\}_{i=1}^{L}, \tag{92}$$

where $L^{1,i}$ is the squared error of agent i. We also calculate the error for output normalization, that is

$$L^{2,DSS} = (Y_t - Y)^2. \tag{93}$$

3. Repeat step 2 for each element in a batch B, which gives the following loss components:

$$\left\{\left\{L^{1,DSS,i,b}\right\}_{i=1}^{L}, L^{2,DSS,b}\right\}_{b=1}^{B}. \tag{94}$$

[3]The Kuhn-Tucker conditions can be written as $\mu_t^i \geq 0,\ \left(B_t^i - \underline{B}\right) \geq 0,\ \mu_t^i \times \left(B_t^i - \underline{B}\right) = 0$.

[4]The Kuhn-Tucker conditions can be written as: $e \geq 0,\ f \geq 0,\ e \times f = 0$. The Fischer-Burmeister function representing them is defined as: $\Psi^{FB}(e, f) = e + f - \sqrt{e^2 + f^2}$. The conditions are satisfied when $\Psi^{FB}(e, f) = 0$.

4. Define the loss function:

$$\phi^L = \frac{1}{B}\sum_{b=1}^{B}\left[\sum_{i=1}^{L}\alpha_1^{i,DSS}L^{1,DSS,i,b} + \alpha_2^{DSS}L^{2,DSS,b}\right], \tag{95}$$

where the weights for the equations are determined by $\{\alpha_1^i\}_{i=1}^L$ and α_2.

5. Optimize the parameters of the NNs ψ_{NN}^{SS} $\psi_{NN}^{I,DSS}$ and ψ_{NN}^{A} to minimize the loss function ϕ^L with a stochastic gradient decent based optimizer.

6. Steps 2 - 5 are repeated N^{int} times to take several optimization steps. Redraw the shock realization in period $t+1$ for each internal optimization step.

7. Draw new parameters for each economy in the batch. This step is not required at each iteration.

8. Simulate each economy for T^{sim} periods using randomly drawn shocks. This creates then the state vector for the next iteration of the optimizer.

9. Steps 2 - 8 are repeated N^{iter} times until the NN converges on average to sufficiently low loss.

Note that we additionally add an upper bound for the capital holdings of the agents, that is $K_t^i \geq \bar{K}_t$. This upper bound provides stability to the algorithm, as it avoids an overaccumulation of assets during the training. To incorporate this upper bound, we use the Fisher-Burmeister condition. After having trained the neural network, we ensure that the upper bound is not binding.

4.4 NN Setup and Results: Deterministic Steady State

We present the setup and results for the deterministic steady state that our code generates.

Neural Network. We specify two neural networks. Both neural networks contain 5 hidden layers with 128 neurons each. The activation function for the hidden layers are CELU. The optimizer is AdamW. The learning rate starts at 1×10^{-4}, and is then reduced at four fixed intervals by a factor of 0.1. The batch size is set to 50. The number of Monte Carlo integrations is set to 50. We use 100 households to compute the residuals for the individual optimization problems, while 100 households are used to keep track of the distribution and compute aggregates.

Computational time. The computations are performed on an NVIDIA RTX PRO 6000 Blackwell Server Edition (via Google Colab). The computational time for the DSS is around 0h43min.[5]

[5]Note that the DSS with a computationally less demanding setup, with fewer Monte Carlo draws (30 instead of 50) and fewer agents (25 instead of 100), can be solved on an Apple M4 Pro chip with a 14-core CPU in around 1h19min.

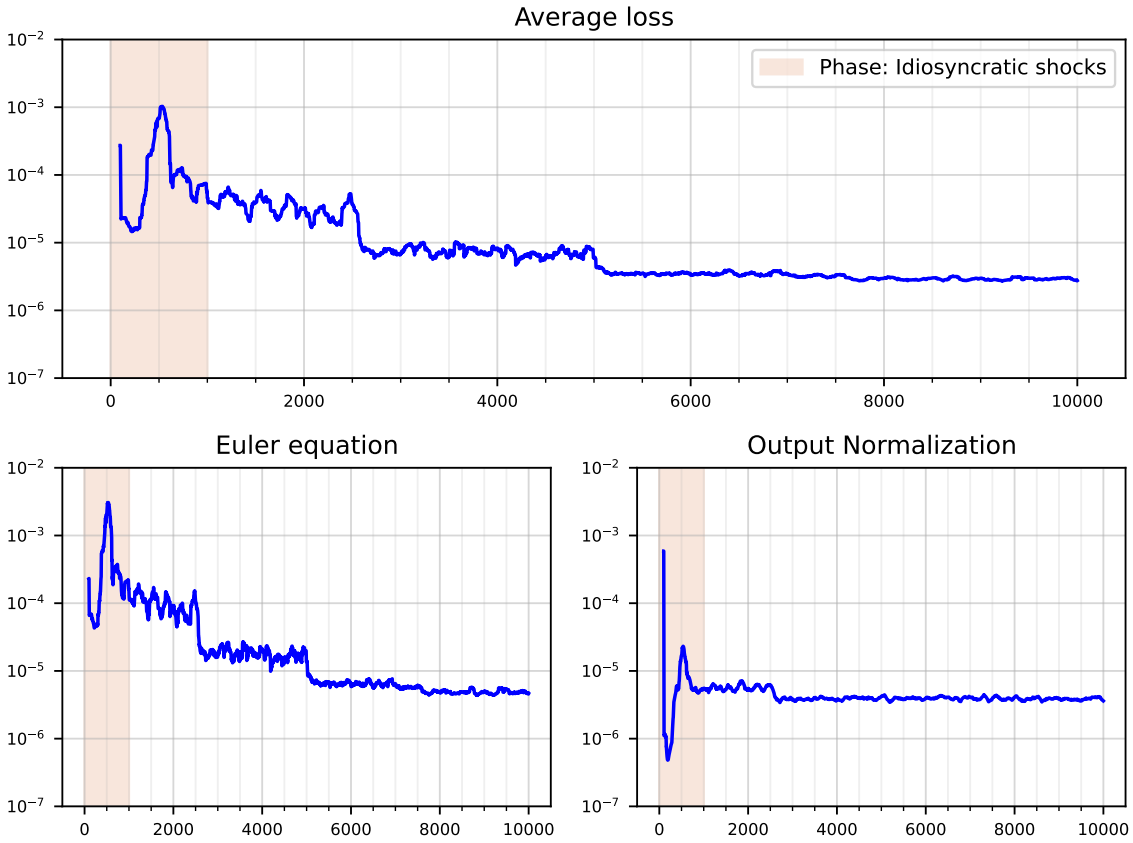

Figure 1: Convergence of the NN solution for the deterministic steady state. The figure shows the dynamics of the mean squared error during the training of the DSS. The upper plot shows the average loss, while the subcomponents are shown below. The shaded area indicates the period in which we scale up idiosyncratic risk. The vertical axis has a logarithmic scale.

Diagnostics: Training loss and approximation error. The top panel of Figure 1 displays the evolution of the average loss function ϕ^L over the training. In the lower panels of that figure, we plot the in-training evolution of the Euler equation and for the output normalization To facilitate convergence during training, we gradually introduce idiosyncratic shocks by increasing their variance from a small value to their target levels over the course of the 1,000 iterations. The red shaded area marks the stage at which this increase begins.

The training is successful, with clear convergence of the average loss. The breakdown of the loss into the individual residual errors confirms that the neural network learns all policy functions of interest effectively.

The next step is to define a diagnostic to assess how accurate the model solution is *after* the training of the neural networks has been completed. We use the Euler equation errors, following Judd (1992) and Aruoba et al. (2006). But, instead of focusing on one condition, we use the same set of equilibrium conditions used during training, which are the Euler equation and the output normalization:

$$err_1 = \left| \Psi^{FB} \left(\frac{1}{\lambda_t^i} - 1, B_t^i - \underline{B} \right). \right|, \tag{96}$$

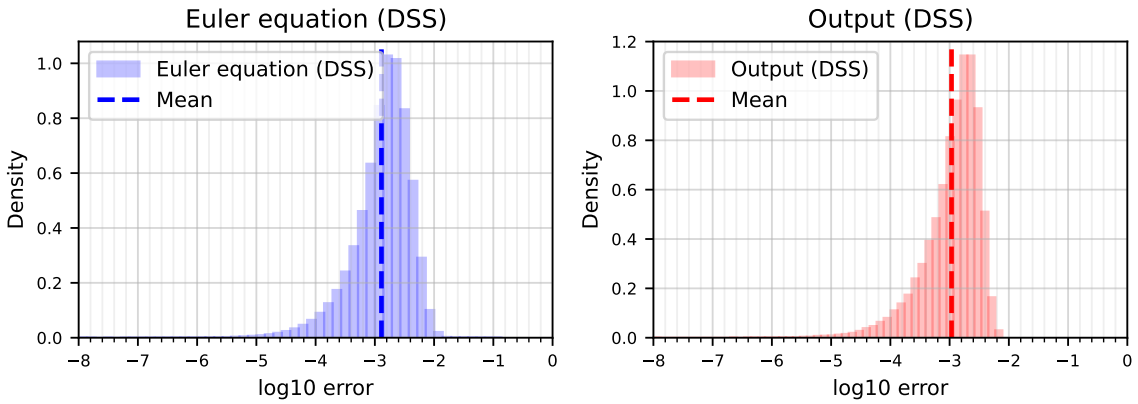

Figure 2: Distribution of post-training approximation errors. The figure shows the absolute error for the modified Euler equation (to account for the occasionally binding borrowing limit), and output normalization. Errors are evaluated after the training of the neural networks is complete. The distributions are based on a simulation of 1,000 periods for 100 economies, resulting in a total of 100,000 observations. We evaluate the expectations, relevant for the Euler equation using 1,000 Monte Carlo draws. The horizontal axis displays the log_{10} of the value.

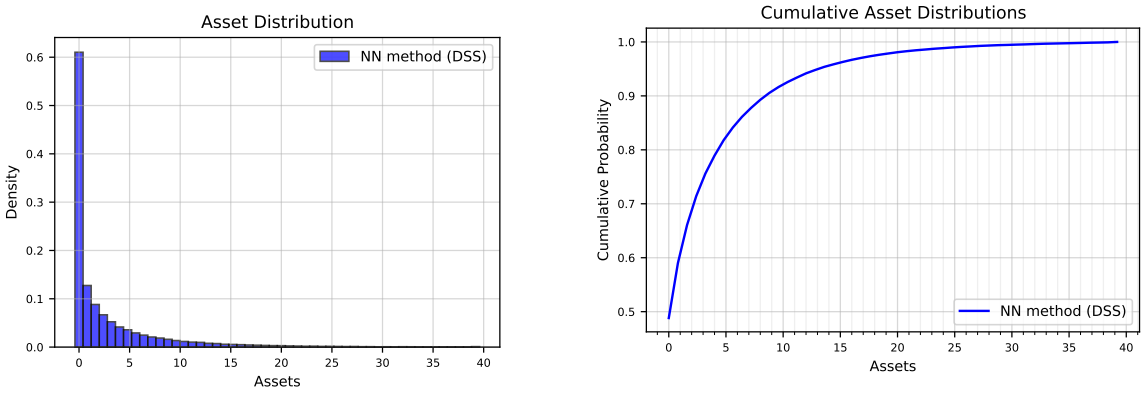

Figure 3: Asset distribution. The figure shows the histogram of asset holdings (left) and the corresponding cumulative distribution function (right). The distribution is evaluated at the deterministic steady state, averaged over 100 simulated economies.

$$err_2 = |Y_t - Y| \,. \tag{97}$$

We simulate our model for 1,000 periods for 100 economies, resulting in a total of 100,000 observed periods. We calculate the approximation error for each period.[6] The results of the validation exercise for the two conditions are shown in Figure 2. The residual errors are well contained, even in the right tails of the distributions. Each distribution is approximately bell-shaped, with modes close to the corresponding mean absolute errors. This suggests that large approximation errors are rare and unlikely to systematically affect the model's dynamics.

Moreover, the symmetry and concentration of the distributions around small values further reinforce the reliability of the neural network solution, indicating that the approximation performs consistently well across a wide range of simulated states.

[6]We draw $1,000$ shocks to evaluate expectations.

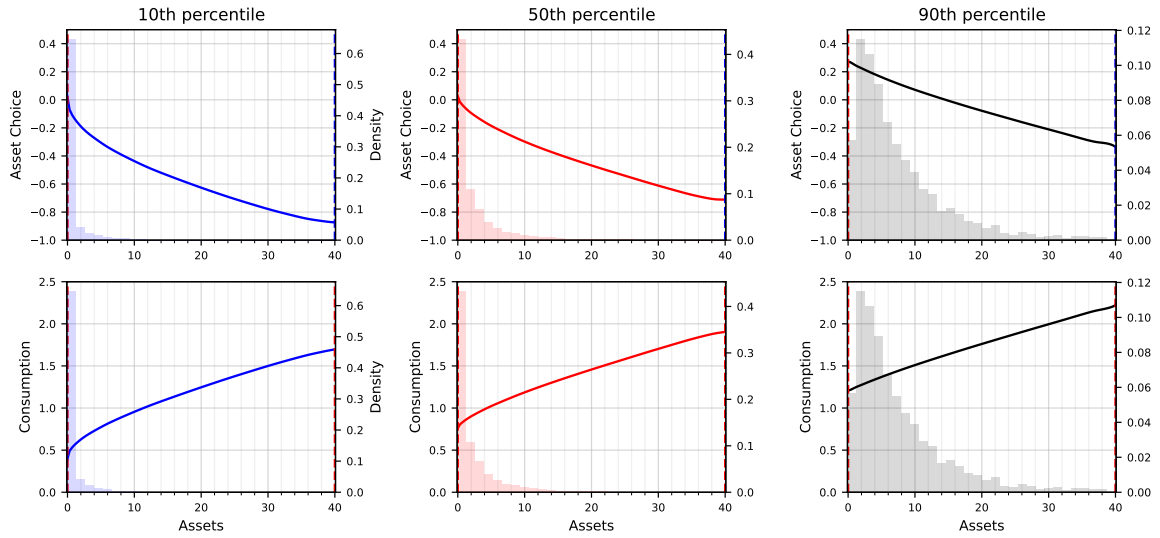

Figure 4: Policy functions. Policy functions (asset choice - upper panels - and consumption - lower panels) and asset distribution conditional on individual productivity (10th percentile, 50th percentile, 90th percentile).

Distribution. Figure 14 shows the distribution of asset holdings across different levels of discretization, with the left panel showing the histogram and the right panel the corresponding cumulative distribution function. Each distribution is obtained by averaging across 100 simulated economies (batches).

Individual policy functions. Figure 15 shows the bar graph of the asset distribution conditional on various levels of productivity (10th percentile, 50th percentile, and 90th percentile) along with the policy functions of assets and consumption.

4.5 NN-based Solution Algorithm for the Full Equilibrium

As a next step, we now solve the full equilibrium. The steps of the algorithm are as follows:

1. Set up the NNs to approximate the policy functions and initialize their weights:

   The NN $\psi^I_{NN}\left(\mathbb{S}^i_t, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)$ for the individual policy functions

   $$\left\{\left(C^i_t\right) = \psi^I_{NN}\left(\mathbb{S}^i_t, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)\right\}^L_{i=1}. \tag{98}$$

2. Solve for all time t variables for a given state vector of batch b. From the NN, we have a current guess for the policy functions:

   $$\{C^i_t\}^L_{i=1}. \tag{99}$$

The next step is to calculate the following (aggregate) variables:

$$C_t = \left(\frac{1}{L} \sum_{i=1}^{L} C_t^i) \right), \tag{100}$$

$$K_{t-1} = \left(\frac{1}{L} \sum_{i=1}^{L} K_{t-1}^i) \right), \tag{101}$$

$$N_t = \left(\frac{1}{L} \sum_{i=1}^{L} \bar{H} \exp(s_t^i) \right), \tag{102}$$

$$Y_t = A_t K_{t-1}^{\alpha} N_t^{1-\alpha}, \tag{103}$$

$$r_t = \alpha \frac{Y_t}{K_{t-1}} - \delta, \tag{104}$$

$$w_t = (1-\alpha) \frac{Y_t}{N_t}, \tag{105}$$

$$K_t = Y_t - C_t + (1-\delta) K_{t-1}. \tag{106}$$

After that, we calculate each household's asset holding:

$$\left\{ K_t^i = (1+r) K_{t-1}^i + w_t \bar{H} \exp(s_t^i) - C_t^i \right\}_{i=1}^{L}. \tag{107}$$

We use a Monte Carlo approach to evaluate the expectations. We randomly draw M sets of next period shocks to evaluate the expectations using the antithetic variates techniques. The drawn shocks are given as:

$$\left\{ \left\{ \epsilon_{t+1}^{s,i,m} \right\}_{i=1}^{L}, \epsilon_{t+1}^{A,m} \right\}_{m=1}^{M}, \tag{108}$$

where the last superscript m indicates which shock draw the next period value is associated with. Note that, with a slight abuse of notation, we also use the superscript m, in the first position, to denote the monetary policy shock ϵ_t^m.

We then proceed to calculate the next period values of the stochastic state variables for all M draws, that is:

$$\left\{ \left\{ s_{t+1}^{i,m} = \rho_s s_t^i + \epsilon_{t+1}^{s,i,m} \right\}_{i=1}^{L} \right\}_{m=1}^{M}, \tag{109}$$

$$\left\{ A_{t+1}^m = (1-\rho_A) A + \rho_A A_t + \epsilon_{t+1}^{A,m} \right\}_{m=1}^{M}, \tag{110}$$

which then gives us the state variables for all M shock draws.

We can then use the NNs for the individual and aggregate policy functions to calculate the individual and aggregate control variables:

$$\left\{ \left\{ \left(C_{t+1}^{i,m} \right) = \psi_{NN}^I \left(\mathbb{S}_{t+1}^{i,m}, \mathbb{S}_{t+1}^m, \tilde{\Theta} | \bar{\Theta} \right) \right\}_{i=1}^{L} \right\}_{m=1}^{M} \tag{111}$$

The next step is to calculate the following (aggregate) variables for the period $t+1$ for

all M draws:

$$\left\{C_{t+1}^m = \left(\frac{1}{L}\sum_{i=1}^{L} C_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{112}$$

$$\left\{N_{t+1}^m = \left(\frac{1}{L}\sum_{i=1}^{L} \bar{H}\exp(s_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{113}$$

$$\{Y_{t+1}^m = A_{t+1}K_t^{\alpha}(N_{t+1}^m)^{1-\alpha}\}_{m=1}^{M}, \tag{114}$$

$$\left\{r_{t+1}^m = \alpha\frac{Y_{t+1}^m}{K_t} - \delta\right\}_{m=1}^{M} \tag{115}$$

After that, we calculate each household's asset holding:

$$\left\{\left\{K_{t+1}^{i,m} = (1+r_{t+1}^m)K_t^i + w_{t+1}^m\bar{H}\exp(s_t^{i,m}) - C_{t+1}^{i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}. \tag{116}$$

We need to ensure that the borrowing constraint for households $K_t^i \geq \underline{K}$ is satisfied. It is convenient to define $\bar{\lambda}_t^i$ as follows:

$$\bar{\lambda}_t^i = 1 - \mu_t^i. \tag{117}$$

We calculate the multiplier for all agents L using their Euler equation:

$$\left\{\bar{\lambda}_t^i = \beta\frac{1}{M}\sum_{m=1}^{M}\left[\left(\frac{C_t^i}{C_{t+1}^{i,m}}\right)^{\sigma}(r_{t+1}^m+1)\right]\right\}_{i=1}^{L}, \tag{118}$$

where we can now evaluate the expectations by taking the mean of the M draws.

We can now calculate the error associated with the Euler equation using the Fisher-Burmeister function:

$$\left\{L^{1,i} = \left(\Psi^{FB}\left(1-\bar{\lambda}_t^i, K_t^i - \underline{K}\right)\right)^2\right\}_{i=1}^{L}, \tag{119}$$

where $L^{1,i}$ is the squared error of agent i.

We also calculate the squared error for the restrictions, which are overidentifying restrictions:

$$L^2 = (K_t - \left(\frac{1}{L}\sum_{i=1}^{L} K_t^i\right))^2, \tag{120}$$

$$L^3 = \frac{1}{M}\sum_{m=1}^{M}\left(\left(K_{t+1}^m - \frac{1}{L}\sum_{i=1}^{L} K_{t+1}^{i,m}\right)^2\right), \tag{121}$$

$$\tag{122}$$

3. Repeat step 2 for each element in a batch B, which gives the following loss components:

$$\left\{\left\{L^{i,b}\right\}_{i=1}^{L}, L^{2,b}, L^{3,b}\right\}_{b=1}^{B}. \tag{123}$$

4. Define the loss function:

$$\phi^L = \frac{1}{B}\sum_{b=1}^{B}\left[\sum_{i=1}^{L}\alpha_1^i L^{1,i,b} + \alpha_2 L^{2,b} + \alpha_3 L^{3,b}\right], \tag{124}$$

where $\left\{\alpha_1^i\right\}_{i=1}^{L}$, α_2, α_3 determine the weights for the different equations.

5. Optimize the parameters of the NNs ψ_{NN}^I and ψ_{NN}^A to minimize the loss function ϕ^L with a stochastic gradient decent based optimizer.

6. Steps 2 - 5 are repeated N^{int} times to take several optimization steps. Redraw the shock realization in period $t+1$ for each internal optimization step.

7. Draw new parameters for each economy in the batch. This step is not required at each iteration.

8. Simulate each economy for T^{sim} periods using randomly drawn shocks. This creates then the state vector for the next iteration of the optimizer.

9. Steps 2 - 8 are repeated N^{iter} times until the NN converges on average to sufficiently low loss.

4.6 NN Setup and Results: Full Equilibrium.

We present the setup and results for the full equilibrium that our code generates.

Neural Network. We specify one neural network, which contains 5 hidden layers with 128 neurons each. The activation function for the hidden layers are CELU. The optimizer is AdamW. The learning rate starts at 1×10^{-4}, and is then reduced at four fixed intervals by a factor of 0.1. The batch size is set to 50. The number of Monte Carlo integrations is set to 50. We use 100 households to compute the residuals for the individual optimization problems, while 100 households are used to keep track of the distribution and compute aggregates. Note that we choose a rather parsimonious specification on purpose to ensure that the model could still be solved in several hours on a modern laptop using the CPU.

Computational time The computations are performed on an NVIDIA RTX PRO 6000 Blackwell Server Edition (via Google Colab). The computational time for the full equilibrium is around 1h24min. [7]

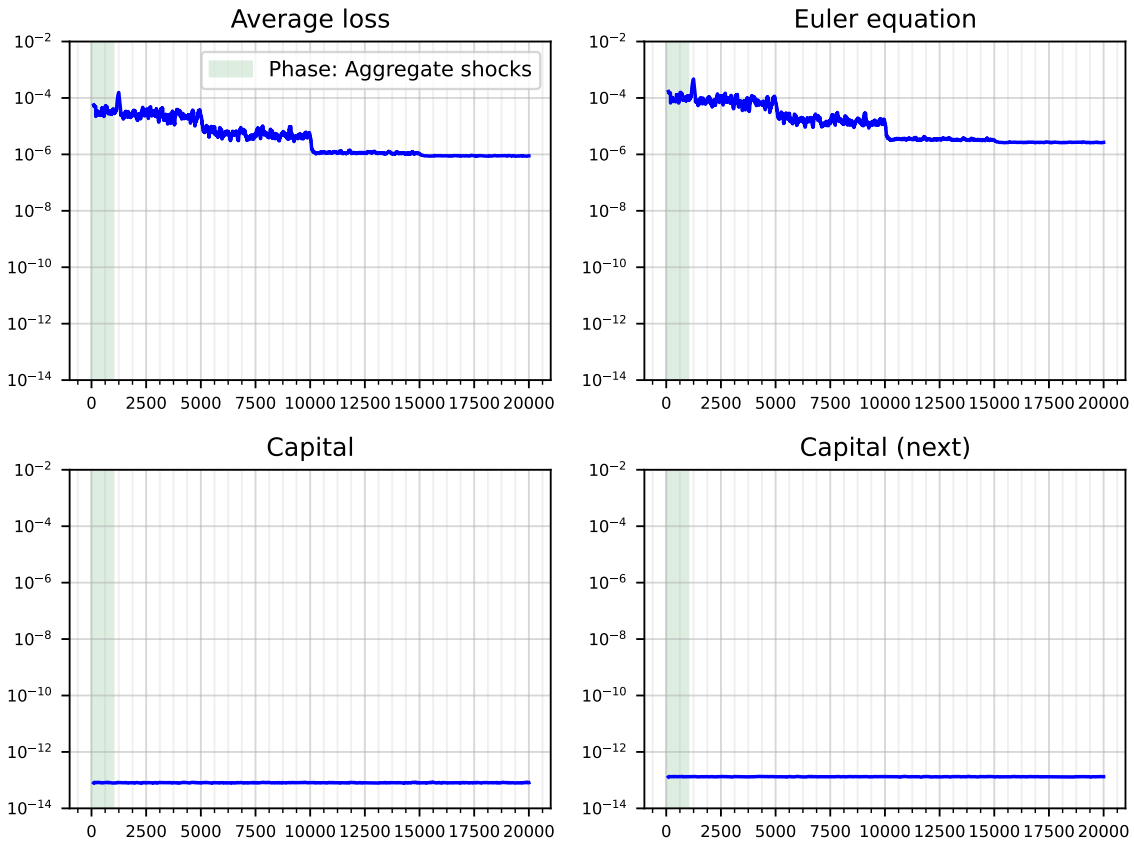

Figure 5: Convergence of the NN solution for the full equilibrium. The figure shows the dynamics of the mean squared error during the training of the full equilibrium. The shaded area indicates the periods in which we scale up aggregate risk. The vertical axis has a logarithmic scale.

Diagnostics: Training loss and approximation error. The left panel of Figure 16 displays the evolution of the average loss function ϕ^L over the training for 20,000 iterations. In the other panels of that figure, we plot the in-training evolution of the Euler equation and for the capital conditions.[8] To facilitate convergence during training, we gradually introduce aggregate risk by increasing their variance from a small value to their target levels over the course of the 1,000 iterations. The blue shaded area marks the stage at which this increase begins. The training is successful, with clear convergence of the average loss.

The next step is to calculate the approximation error after the training of the neural networks has been completed. The relevant equation is the absolute error of the Euler equation:

$$err_1 = \left| \Psi^{FB} \left(\frac{1}{\bar{\lambda}_t^i} - 1, B_t^i - \underline{B} \right) . \right|, \tag{125}$$

We simulate our model for 1,000 periods for 100 economies, resulting in a total of 100,000

[7]Note that the full equilibrium with a computationally less demanding setup, with fewer Monte Carlo draws (30 instead of 50) and fewer agents (25 instead of 100), can be solved on an Apple M4 Pro chip with a 14-core CPU in around 2h44min.

[8]Note that the capital conditions rae overidentified restrictions.

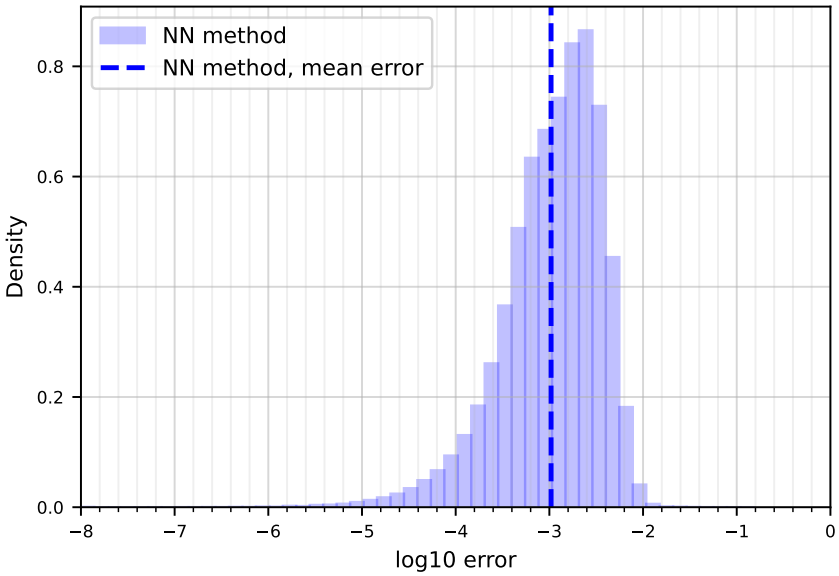

Figure 6: Distribution of post-training approximation errors. The figure shows the absolute error for the modified Euler equation (to account for the occasionally binding borrowing limit). Errors are evaluated after the training of the neural networks is complete. The distributions are based on a simulation of 1,000 periods for 100 economies, resulting in a total of 100,000 observations. We evaluate the expectations, relevant for the Euler equation using 1,000 Monte Carlo draws. The horizontal axis displays the log_{10} of the value.

observed periods. We calculate the approximation error for each period.[9] The results of the validation exercise are shown in Figure 17. The residual error of the Euler equation is well contained, even in the right tails of the distributions. The distribution is approximately bell-shaped, with the mode close to the corresponding mean absolute error. This suggests that large approximation errors are rare and unlikely to systematically affect the model's dynamics. Moreover, the symmetry and concentration of the distributions around small values further reinforce the reliability of the neural network solution, indicating that the approximation performs consistently well across a wide range of simulated states.

Distribution. Figure 18 shows the distribution of asset holdings across different levels of discretization, with the left panel showing the histogram and the right panel the corresponding cumulative distribution function. Each distribution is obtained by averaging the stochastic steady state across 100 simulated economies (batches).

Impulse response functions. Figure 19 shows the equilibrium effects of a positive TFP shock on selected aggregate variables. The responses show the deviations from the stochastic steady state (SSS). The impulse response are averaged over 1,000 simulated economies, to avoid that idiosyncratic shocks of the agents affect the results.[10]

[9] We draw $1,000$ shocks to evaluate expectations.

[10] For each simulated economy, the impulse response is measured relative to its own stochastic steady state under the same realization of idiosyncratic shocks.

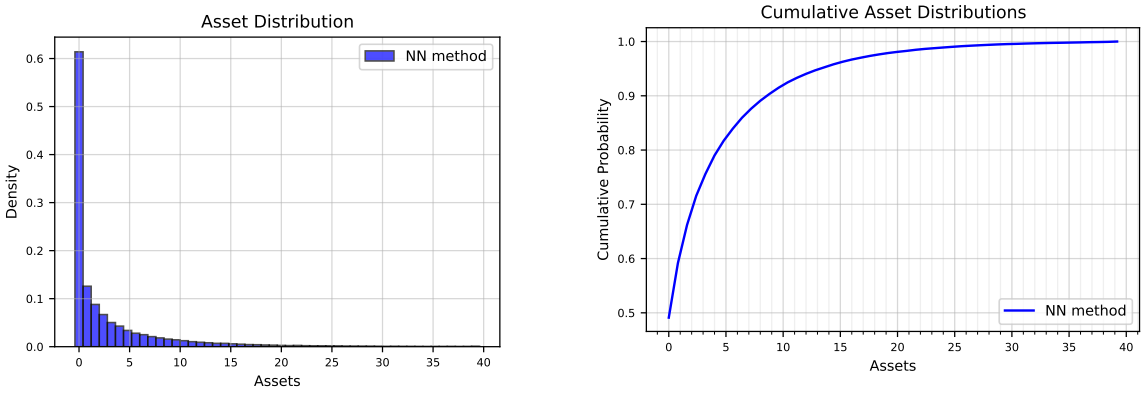

Figure 7: Asset distribution. The figure shows the histogram of asset holdings (left) and the corresponding cumulative distribution function (right). The distribution is evaluated at the stochastic steady state, averaged over 100 simulated economies.

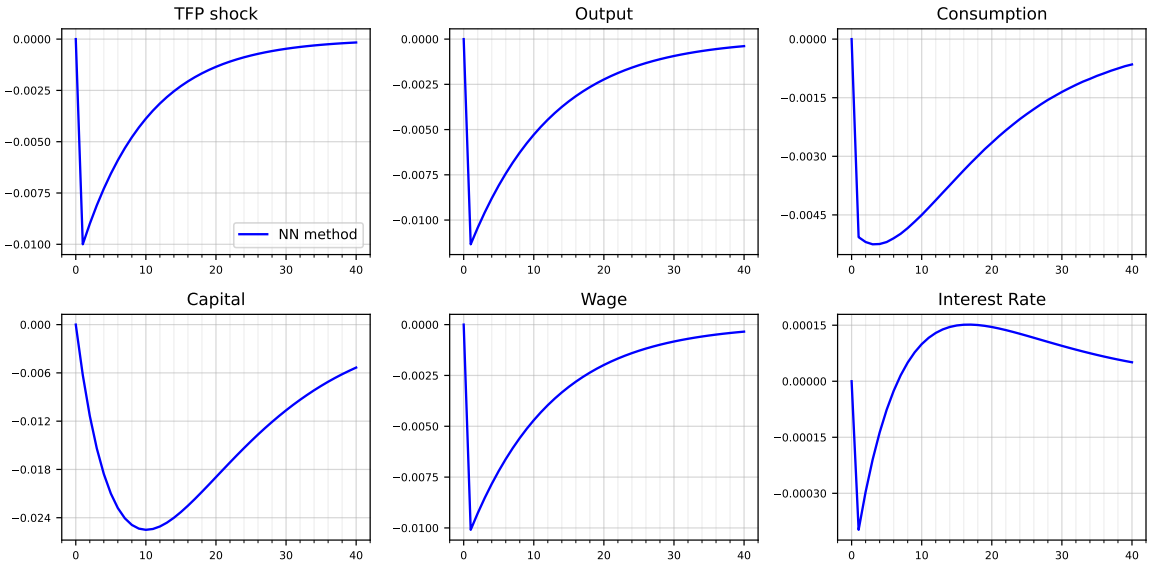

Figure 8: Impulse response functions for a one standard deviation shock. Deviations from the SSS are shown. The impulse response are averaged over 1,000 simulated economies.

4.7 Comparison with the Sequence Space Jacobian

In the next step, we want to compare our neural network based solution method to a conventional (linear) apprach, specifically the sequence space Jacobian (SSJ) approach of Auclert et al. (2021).[11] For this comparison, we use the conventional linearized SSJ solution. Note that the solution methods in general do not coincide as our approach incorporates nonlinear dynamics and aggregate uncertainty affects the equilibrium dynamics.[12] This version of the Krusell-Smith model features only one shock, and therefore aggregate uncertainty does not play a big role in affecting agents' decisions and equilibrium allocations. Even in its original

[11] We also choose the SSJ due to the availability of the Auclert et al. (2021) repository, which provides a natural benchmark for scholars and practitioners interested in comparing it with our code.

[12] There are also other differences, such as that we do not discretize the shocks.

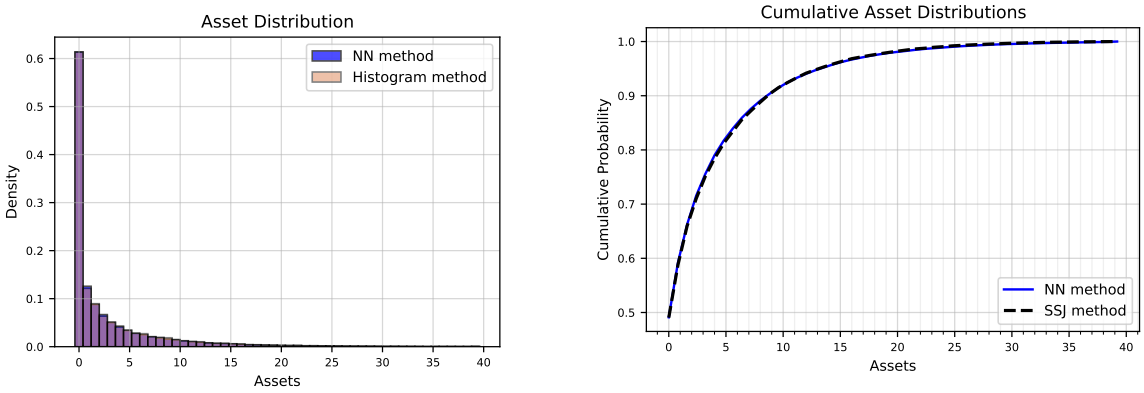

Figure 9: Asset distribution for our neural network (NN) method for comparison with sequence space Jacobian method. For the NN method, we evaluate the distribution at its stochastic steady state, averaging over 100 economies. For the SSJ, we use the deterministic steady state as it cannot account for aggregate uncertainty. LHS: The histogram of asset holdings in equilibrium. RHS: Cumulative density function for the asset holdings in equilibrium.

nonlinear specification, the equilibrium dynamics of aggregate variables in this model remain close to linear, making the solution obtained via the SSJ method a reliable standard for validation.[13]

As a next step, we compare our solution method based and along the following features: 1) the equilibrium asset distribution, 2) the individual policy functions, and 3) the impulse response functions to the monetary policy shock.

Distribution. Figure 9 compares the asset distribution generated by our method and by the SSJ approach. For our method, we report the distribution at the stochastic steady state, which accounts for the presence of aggregate risk. In contrast, the SSJ method by construction assumes away aggregate uncertainty, so the distribution it produces corresponds to the deterministic steady state. Despite this key difference, the two distributions are remarkably similar. Although minor deviations are visible in the tails, the overall shape and key moments align closely, suggesting that our method provides a highly accurate approximation.

Individual policy functions. The next step is to compare the two sets of policy functions that determine agents' consumption and capital decisions, as shown in Figure 10. The solid lines depict the policy functions approximated using our method, while the dashed lines correspond to the approximation obtained using the SSJ method. Each color represents one of the seven idiosyncratic labor productivity levels considered in the SSJ. The black lines correspond to the most productive agents (top productivity bin), while the blue lines represent the least productive ones (bottom productivity bin.) We display a shorter horizon for capital for a better visualization.

[13]Note that setting the volatility of aggregate shocks to be very small is problematic: our neural-network solver, which is designed to handle nonlinear models with meaningful aggregate risk, fails to learn effectively in that case. In practice, when aggregate volatility is reduced to negligible levels to suppress aggregate risk, we struggle to train the neural networks as the errors are too small

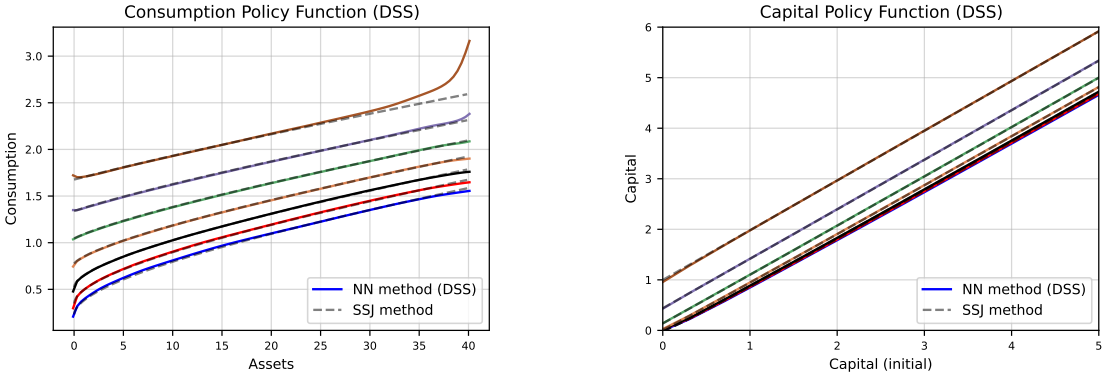

Figure 10: Individual policy functions comparison for the one-asset HA model. The figure illustrates how the policy functions for labor and consumption vary for different assets levels and different productivity levels. We compare the neural network method with the SSJ method. We use the 7 productivity levels that the discretization approach of the SSJ provides for comparison. We evaluate the policy functions at the deterministic steady state for both methods.

Note that these seven productivity levels are used solely for the purpose of comparing the results from our solution method with those from SSJ. It is worth recalling that our solution method does not rely on discretizing the productivity space. Instead, we treat idiosyncratic labor productivity as the outcome of a continuous AR(1) process, as specified in the model. For the purpose of this comparison, we evaluate the policy functions (solved for a continuous AR(1) process) at the values in line with the discretization in the code made available by Auclert and coauthors.

Our method successfully replicates the policy functions approximated by the SSJ approach. Second, the largest difference arises for the most productive agents with large asset holdings, who constitute only a very small fraction of the population. As our method imposes an upper bound on the amount of assets that agents can accumulate, these agents increase their consumption to avoid reaching the borrowing limit.[14] Third, another small discrepancy arises in the behavior of the most productive agents with zero assets—a group that represents an extremely small fraction of agents — and the unproductive agents with large asset holdings. These combinations require highly unlikely sequence of events. Because so few agents follow such a path, the neural network has limited data to learn from, which results in a less accurate approximation of their behavior. Nonetheless, the inaccuracies in these cases are fairly limited, highlighting the strong extrapolative properties of neural network. Moroever, since these "extreme" types of agents have almost zero measure in the population, the approximation error introduced by the neural network in this specific region is clearly inconsequential for aggregate outcomes.

Impulse response functions. The effects of a TFP shock on the equilibrium dynamics of aggregate variables predicted by the two methods are quite similar, as shown in Figure 11. The responses show the deviations from the stochastic steady state (SSS) for the nueral

[14] Note that increasing the upper bound reduces this effect.

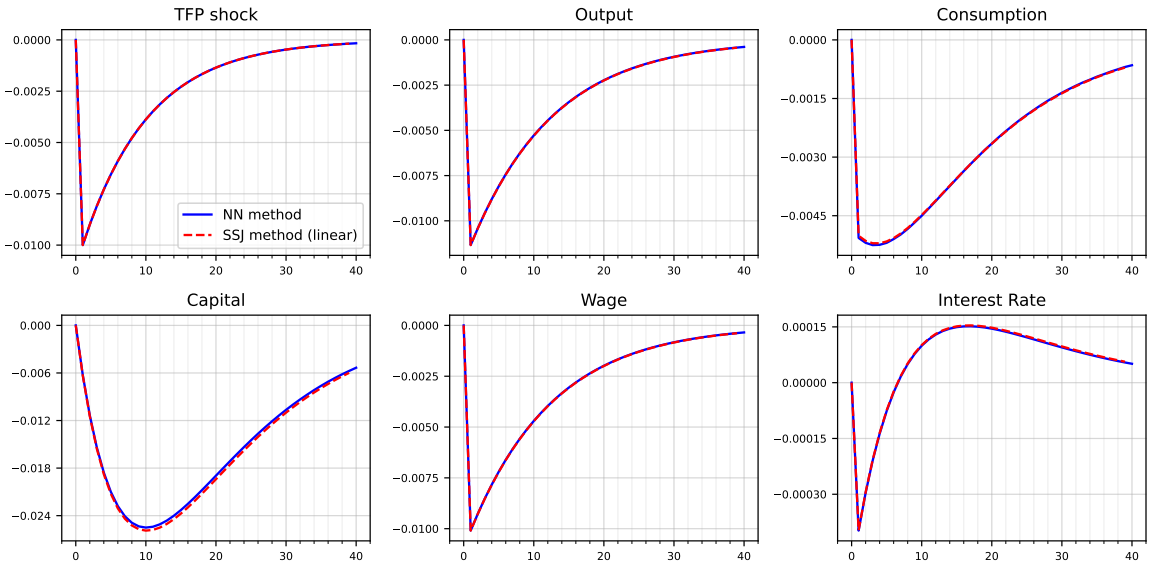

Figure 11: Impulse response functions comparison of neural network solution and sequence space Jacobian approach for a standard deviation positive shock. The deviations from SSS for the NN based method and from the DSS for SSJ are shown.

network-based method. As the SSJ does not feature the SSS, the deviations in that case are shown from the deterministic steady state (DSS). The impulse response are averaged over 1,000 simulated economies, to avoid that idiosyncratic shocks of the agents affect the results. The transmission of the shocks is very similar, as aggregate uncertainty plays only a very limited role.

5 Two-asset Heterogeneous Agent model

We now extend the one-asset Heterogenous Agent model of Section 4 by incorporating a second asset. Specifically, households can now also hold government debt. The introduction of the second asset makes the household problem substantially more demanding. Households now choose consumption, capital holdings, and bond holdings, subject to separate optimality conditions and potentially binding lower bounds for both assets. The individual state space is therefore larger now: two-dimensional in asset holdings, in addition to the idiosyncratic productivity state.

Solution File.

- **GitHub:** To be added upon publication.

5.1 Overview Model

Households consume C_t^i to maximize their utility:

$$E_0 \sum_{t=0}^{\infty} \beta^t \frac{\left(C_t^i\right)^{1-\sigma}}{1-\sigma}. \tag{126}$$

However, they can now also invest in government bonds. The budget constraint can be written as

$$C_t^i + K_t^i + Q_t B_t^i + \frac{\gamma}{2}\left(B_t^i - \overline{B}\right)^2 + T_t = B_{t-1}^i + (1+r_t)K_{t-1}^i + w_t \bar{H} \exp(s_t^i), \tag{127}$$

where B_t^i is the household-specific bond holdings that are purchased at price Q_t. The households have a target ratio for the bond holdings $\overline{B}$. We assume that the costs are given back to the household in a lump-sum manner. As a result, they do not affect individual households budget constraint directly, but only their asset choice. The agents can not have capital holdings as well as bond holdings below some limit, which implies:

$$K_t^i \geq \underline{K}, \tag{128}$$

$$B_t^i \geq \underline{B}. \tag{129}$$

The first-order conditions can be written as

$$1 = \beta \mathbb{E}_t \left[\left(\frac{C_{t+1}^i}{C_t^i} \right)^{-\sigma} (1 + r_{t+1}) \right] + \mu_t^i, \tag{130}$$

$$Q_t + \gamma\left(B_t^i - \overline{B}\right) = \beta \mathbb{E}_t \left[\left(\frac{C_{t+1}^i}{C_t^i} \right)^{-\sigma} \right] + \nu_t^i, \tag{131}$$

where μ_t^i is the normalized Lagrange multiplier on the non-negativity constraint for individual capital holdings and ν_t^i is the normalized Lagrange multiplier on the non-negativity constraint for individual bonds holdings.

The representative firm produces output Y_t using total labor L_t and total capital K_t:

$$Y_t = (A_t) K_{t-1}^{\alpha} N_t^{\alpha}, \tag{132}$$

where total factor productivity A_t follows an AR(1) process: $A_t = (1-\rho)A + \rho_A A_{t-1} + \epsilon_t^A$. The first-order conditions are

$$r_t = \alpha \frac{Y_t}{K_{t-1}} - \delta, \tag{133}$$

$$w_t = (1-\alpha)\frac{Y_t}{N_t} \tag{134}$$

The fiscal authority sets taxes T_t to keep the level of bonds B constant:

$$B = Q_t B + T_t. \tag{135}$$

Note that market clearing for the labor market, capital market and goods market requires

$$N_t = \int \exp(s_t^i)\bar{H}di, \tag{136}$$

$$K_t = \int K_t^i di, \tag{137}$$

$$B = \int B_t^i di, \tag{138}$$

$$Y_t = \int C_t^i di + K_t - (1-\delta)K_{t-1}. \tag{139}$$

Equilibrium conditions. The equilibrium conditions can be written as:

$$C_t^i + K_t^i + Q_t B_t^i + \gamma\left(B_t^i - \overline{B}\right)^2 + T_t = (1+r_t)K_{t-1}^i + B_{t-1}^i w_t \bar{H}\exp(s_t^i), \forall i, \tag{140}$$

$$K_t^i \geq \underline{K}, \forall i, \tag{141}$$

$$B_t^i \geq \underline{B}, \forall i, \tag{142}$$

$$\lambda_t^i = \frac{1}{C_t^i}, \forall i, \tag{143}$$

$$1 = \beta\mathbb{E}_t\left[\left(\frac{C_{t+1}^i}{C_t^i}\right)^{-\sigma}(r_{t+1}+(1-\delta))\right] + \mu_t^i, \forall i, \tag{144}$$

$$Q_t + \gamma\left(B_t^i - \overline{B}\right) = \beta\mathbb{E}_t\left[\left(\frac{C_{t+1}^i}{C_t^i}\right)^{-\sigma}\right] + \nu_t^i, \forall i, \tag{145}$$

$$r_t = \alpha\frac{Y_t}{K_{t-1}} - \delta \tag{146}$$

$$w_t = (1-\alpha)\frac{Y_t}{N_t}, \tag{147}$$

$$N_t = \int \exp(s_t^i)\bar{H}di, \tag{148}$$

$$T_t = (Q_t - 1)B \tag{149}$$

$$K_t = \int K_t^i di, \tag{150}$$

$$B = \int B_t^i di, \tag{151}$$

$$Y_t = \int C_t^i di + K_t - (1-\delta)K_{t-1}, \tag{152}$$

$$s_t^i = \rho_s s_{t-1}^i + \epsilon_t^{s,i}, \forall i, \tag{153}$$

$$A_t = \rho_A A_{t-1} + \epsilon_t^A A_t = A + A_t \tag{154}$$

Calibration. The calibration follows Section 4, that is: $\beta = 0.982$, $\sigma = 1.0$, $\alpha = 0.11$, $\delta = 0.025$, $\underline{K} = 0$ $\rho_s = 0.966$, $\sigma_s = 0.129$, $\rho_A = 0.9$, σ_A=0.02. We map the productivity level A to have an output level of 1 in the deterministic steady state. The additional parameters are set as follows: $\gamma = 0.025$, $\underline{B} = 0$, and $\bar{B} = 1$.

5.2 Mapping of the model to the general framework

The state variables and shocks of the stochastic growth model with partially uninsurable labor are given as:

$$\mathbb{S}_t = \left\{ \left\{ K_{t-1}^i \right\}_{i=1}^L, \left\{ B_{t-1}^i \right\}_{i=1}^L, \left\{ s_t^i \right\}_{i=1}^L, A_t \right\}, \tag{155}$$

$$\nu_t = \left\{ \left\{ \epsilon_t^{s,i} \right\}_{i=1}^L, \epsilon_t^a \right\}. \tag{156}$$

The parameters of the model are divided into calibrated ($\bar{\theta}$) and estimated ones ($\tilde{\theta}$). For instance, in the case of our application with US data, this yields:

$$\bar{\Theta} = \{\beta, \sigma, \alpha, \delta, \rho_s, \sigma_s, \rho_A, \sigma_A, \underline{K}, \gamma, \underline{B}, B\}, \tag{157}$$

$$\tilde{\Theta} = \{\}. \tag{158}$$

In this specific application, we need a NN for the individual policy functions. The individual NN $\psi_{NN}^I(\cdot)$ solves for consumption:

$$\left\{ \begin{pmatrix} C_t^i \\ B_t^i \end{pmatrix} = \psi_{NN}^I \left(\mathbb{S}_t^i, \mathbb{S}_t, \tilde{\Theta} | \bar{\Theta} \right) \right\}_{i=1}^L, \tag{159}$$

where $\mathbb{S}_t^i = \left\{ B_{t-1}^i, s_t^i \right\}$. We also have a neural network that provides the productivity level of the deterministic steady state:

$$\begin{pmatrix} A \\ Q \end{pmatrix} = \psi_{NN}^{SS} \left(\tilde{\Theta} | \bar{\Theta} \right). \tag{160}$$

We now have an aggregate neural network

$$\left(Q_t \right) = \psi_{NN}^A \left(\mathbb{S}_t, \tilde{\Theta} | \bar{\Theta} \right) \tag{161}$$

5.3 NN-based Solution Algorithm for the Deterministic Steady State

We first solve the deterministic steady state. We use this step to calibrate the productivity level of the model to match an output level of 1 in the DSS.

The DSS NN solves for the deterministic steady state values:

$$\begin{pmatrix} A \\ Q \end{pmatrix} = \psi_{NN}^{SS} \left(\tilde{\Theta} | \bar{\Theta} \right). \tag{162}$$

To solve for this NN, we also need to solve for the individual policy functions of the agents in the DSS:

$$\left\{ \begin{pmatrix} C_t^i \\ B_t^i \end{pmatrix} = \psi_{NN}^{I,DSS} \left(\mathbb{S}_t^i, \mathbb{S}, \tilde{\Theta} | \bar{\Theta} \right) \right\}_{i=1}^L. \tag{163}$$

where the superscript DSS denotes this as the inidivual policy functions for the DSS. Note

that we let the DSS values of the aggregate variables still enter the network. While this is unnecessary for solving the DSS, it allows us to use this trained NN as starting point when we solve the full equilibrium with aggregate risk. Note that the aggregate state variable $\mathbb{S}$ is, of course, constant.

The steps of the algorithm are as follows:

1. Set up the NNs to approximate the policy functions and initialize their weights:

   The NN $\psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right)$ for the steady state parameters:

   $$\left(A\right) = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{164}$$

   The NN $\psi_{NN}^{I,DSS}\left(\mathbb{S}_t^i, \mathbb{S}, \tilde{\Theta}|\bar{\Theta}\right)$ for the individual policy functions

   $$\left\{\left(C_t^i\right) = \psi_{NN}^{I}\left(\mathbb{S}_t^i, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}. \tag{165}$$

2. Solve for all the current period individual variables. From the NN, we get a current guess for the deterministic steady value

   $$\begin{pmatrix} A \\ Q \end{pmatrix} = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{166}$$

   and individual policy functions:

   $$\{C_t^i\}_{i=1}^{L}, \{B_t^i\}_{i=1}^{L}. \tag{167}$$

   The next step is to calculate the following (aggregate) variables:

   $$C_t = \left(\frac{1}{L}\sum_{i=1}^{L} C_t^i)\right), \tag{168}$$

   $$K_{t-1} = \left(\frac{1}{L}\sum_{i=1}^{L} K_{t-1}^i)\right), \tag{169}$$

   $$N_t = \left(\frac{1}{L}\sum_{i=1}^{L} \bar{H}\exp(s_t^i)\right), \tag{170}$$

   $$Y_t = AK_{t-1}^{\alpha}N_t^{1-\alpha}, \tag{171}$$

   $$r_t = \alpha\frac{Y_t}{K_{t-1}} - \delta, \tag{172}$$

   $$w_t = (1-\alpha)\frac{Y_t}{N_t}, \tag{173}$$

   $$T_t = (Q-1)B \tag{174}$$

   where we use a subscript t here to indicate variables calculated based on individual shock realizations.

We pursue calculating each household's individual assets:

$$\left\{K_t^i = (1+r)K_{t-1}^i + B_{t-1}^i + w\bar{H}\exp(s_t^i) - C_t^i - QB_t^i - \gamma(B_t^i - \bar{B})^2 - T_t\right\}_{i=1}^{L}. \tag{175}$$

We use a Monte Carlo approach to evaluate the expectations. We randomly draw M sets of next period individual shocks to evaluate the expectations. We use the antithetic variate method to increase the precision and reduce the number of necessary draws. The antithetic variates technique creates to a given path $\{\nu_1, \nu_2, \nu_3, \ldots, \nu_{M/2}\}$ also its antithetic path $\{-\nu_1, -\nu_2, -\nu_3, \ldots, -\nu_{M/2}\}$. The drawn shocks are given as:

$$\left\{\left\{\epsilon_{t+1}^{s,i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{176}$$

where the last superscript m indicates which shock draw the next period value is associated with.

We then proceed to calculate the next period values of the stochastic state variables for all M draws, that is:

$$\left\{\left\{s_{t+1}^{i,m} = \rho_s s_t^i + \epsilon_{t+1}^{s,i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M} \tag{177}$$

which then gives us the state variables for all M shock draws.

We can then use the NNs for the individual and aggregate policy functions to calculate the individual and aggregate control variables:

$$\left\{\left\{\left(C_{t+1}^{i,m}\right) = \psi_{NN}^{I,DSS}\left(\mathbb{S}_{t+1}^{i,m}, \mathbb{S}^m, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}\right\}_{m=1}^{M} \tag{178}$$

The next step is to calculate the following (aggregate) variables for the period $t+1$ for all M draws:

$$K_t = \left(\frac{1}{L}\sum_{i=1}^{L} K_t^i\right), \tag{179}$$

$$\left\{N_{t+1}^m = \left(\frac{1}{L}\sum_{i=1}^{L} \bar{H}\exp(s_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{180}$$

$$\left\{Y_{t+1}^m = AK_t^{\alpha}(N_{t+1}^m)^{1-\alpha}\right\}_{m=1}^{M}, \tag{181}$$

$$\left\{r_{t+1}^m = \alpha\frac{Y_{t+1}^m}{K_t} - \delta\right\}_{m=1}^{M}, \tag{182}$$

$$T_{t+1} = (Q-1)B, \tag{183}$$

$$K_{t+1} = Y_{t+1} - C_{t+1}. \tag{184}$$

We need to ensure that the borrowing constraint for households $K_t^i \geq \underline{K}$ is satisfied.[15] It is convenient to define $\bar{\lambda}_t^i$ as follows:

$$\bar{\lambda}_t^i = 1 - \mu_t^i, \tag{185}$$

$$\bar{\psi}_t^i = 1 - \nu_t^i. \tag{186}$$

We calculate the multiplier for all agents L using their Euler equation:

$$\left\{ \bar{\lambda}_t^i = \beta \frac{1}{M} \sum_{m=1}^{M} \left[\left(\frac{C_t^i}{C_{t+1}^{i,m}} \right)^{\sigma} (r_{t+1}^m + 1) \right] \right\}_{i=1}^{L}, \tag{187}$$

$$\left\{ \bar{\psi}_t^i = \left(\beta \mathbb{E}_t \left[\left(\frac{C_t^i}{C_{t+1}^{i,m}} \right)^{\sigma} \right] - \gamma \left(B_t^i - \overline{B} \right) \right) / Q \right\}_{i=1}^{L}, \tag{188}$$

where we can now evaluate the expectations by taking the mean of the M draws.

Equipped with this object, we use the Fischer-Burmeister function Ψ^{FB}, a smooth way of representing the Kuhn-Tucker conditions, to ensure that these conditions hold. This can be written as

$$\Psi^{FB} \left(\frac{1}{\bar{\lambda}_t^i} - 1, K_t^i - \underline{K} \right), \tag{189}$$

$$\Psi^{FB} \left(\frac{1}{\bar{\psi}_t^i} - 1, B_t^i - \underline{B} \right). \tag{190}$$

We can now calculate the error associated with the Fisher-Burmeister function:

$$\left\{ L^{1,DSS,i} = \left(\Psi^{FB} \left(1 - \bar{\lambda}_t^i, K_t^i - \underline{K} \right) \right)^2 \right\}_{i=1}^{L}, \tag{191}$$

$$\left\{ L^{2,DSS,i} = \left(\Psi^{FB} \left(1 - \bar{\psi}_t^i, B_t^i - \underline{B} \right) \right)^2 \right\}_{i=1}^{L}, \tag{192}$$

where $L^{1,i}$ is the squared error of agent i. We also calculate the error for output normalization, that is

$$L^{3,DSS} = (Y_t - Y)^2, \tag{193}$$

$$L^{4,DSS} = B - \frac{1}{L} \sum B_t^i \tag{194}$$

5.4 NN Setup and Results: Deterministic Steady State

We present the setup and results for the deterministic steady state that our code generates.

Neural Network. We specify two neural networks. Both neural networks contain 5 hidden layers with 128 neurons each. The activation function for the hidden layers are CELU. The optimizer is AdamW. The learning rate starts at 1×10^{-4}, and is then reduced at four fixed intervals by a factor of 0.1. The batch size is set to 100. The number of Monte Carlo

[15] The Kuhn-Tucker conditions can be written as $\mu_t^i \geq 0,\ \left(B_t^i - \underline{B}\right) \geq 0,\ \mu_t^i \times \left(B_t^i - \underline{B}\right) = 0$.

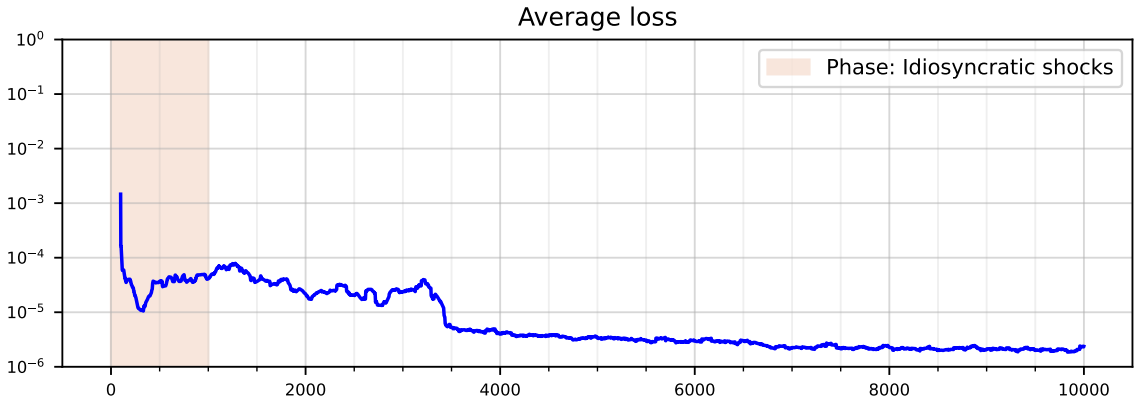

Figure 12: Convergence of the NN solution for the deterministic steady state. The figure shows the average mean squared error during the training of the DSS. The shaded area indicates the period in which we scale up idiosyncratic risk. The vertical axis has a logarithmic scale.

integrations is set to 50. We use 100 households to compute the residuals for the individual optimization problems, while 100 households are used to keep track of the distribution and compute aggregates.

Computational time. The computations are performed on an NVIDIA RTX PRO 6000 Blackwell Server Edition (via Google Colab). The computational time for the DSS is around 0h45min.

Diagnostics: Training loss and approximation error. Figure 12 displays the evolution of the average loss function ϕ^L over the training. As before, we gradually introduce idiosyncratic shocks, as marked by the red shaded area.

Because the household problem now involves two asset choices, we focus our diagnostics on the errors in the first-order conditions for capital and bonds. These conditions are more demanding than in the one-asset model because the neural network must jointly learn the householdâs optimal allocation across the two assets.

We consider the absolute errors in the two complementarity conditions:

$$err_1 = \left| \Psi^{FB}\left(\frac{1}{\bar{\lambda}_t^i} - 1, K_t^i - \underline{K} \right) \right|, \tag{195}$$

$$err_2 = \left| \Psi^{FB}\left(\frac{1}{\bar{\psi}_t^i} - 1, B_t^i - \underline{B} \right) \right|. \tag{196}$$

The first condition governs the optimal choice of capital, while the second governs the optimal choice of government bonds. The FischerâBurmeister function, Ψ^{FB}, allows us to represent the corresponding occasionally binding lower-bound conditions in a differentiable form suitable for neural-network training.

We simulate our model for $1,000$ periods for 100 economies (resulting in 100,000 ob-

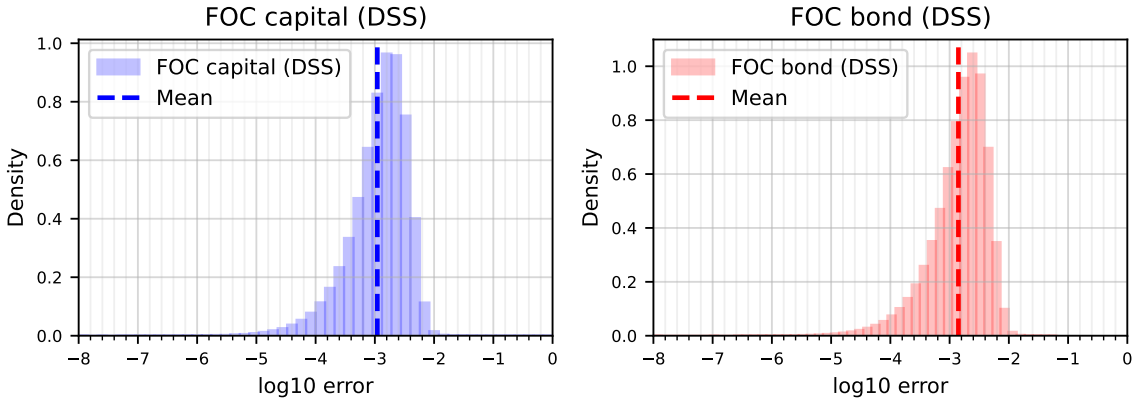

Figure 13: Distribution of post-training approximation errors. The figure shows the absolute error for the FOC for capital and bonds. Errors are evaluated after the training of the neural networks is complete. The distributions are based on a simulation of 1,000 periods for 100 economies, resulting in a total of 100,000 observations. We evaluate the expectations, relevant for the Euler equation using 1,000 Monte Carlo draws. The horizontal axis displays the log_{10} of the value.

served periods) and calculate for each period the approximation error.[16] Figure 13 reports the distributions of these post-training approximation errors. The residual errors remain well contained in this more elaborate two-asset environment, including in the right tails of the distributions. Both distributions are approximately bell-shaped, with most observations concentrated around the corresponding mean absolute errors. Large approximation errors are therefore rare and are unlikely to systematically affect the modelâs dynamics.

Importantly, the approximation error in the first-order condition for capital is virtually the same as in the one-asset heterogeneous-agent model. The mean post-training error is close to -3 in $\log_{10}$ units in both cases, and the two distributions have very similar shapes. Thus, adding a second asset does not materially reduce the accuracy with which the neural network solves the capital-choice problem. At the same time, the small errors in the bond condition show that the neural-network method can accurately learn the additional portfolio margin introduced by the second asset.

Distribution. Figure 14 shows the distribution of capital and bond holdings across different levels of discretization, with the left panel showing the histogram for capital and the right panel the corresponding histogram for bonds. Each distribution is obtained by averaging across 100 simulated economies (batches).

Individual policy functions. Figure 15 reports the individual consumption policy function over different combinations of capital and bond holdings. The left panel evaluates the policy function at the 10th percentile of the productivity distribution, while the right panel evaluates it at the 50th percentile.

These figures provide a direct illustration of the richer household problem. Consumption

[16] We draw $1,000$ shocks to evaluate expectations.

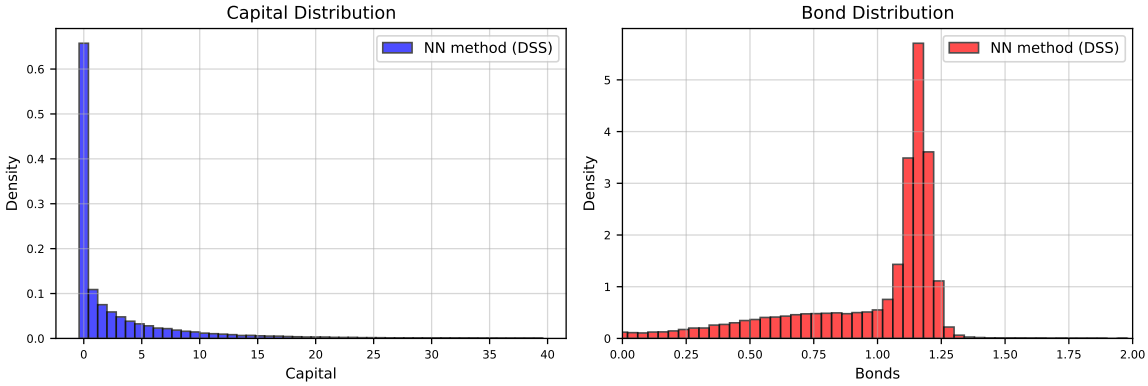

Figure 14: Asset distribution. The figure shows the histogram of capital holdings (left) and bond holdings (right). The distribution is evaluated at the deterministic steady state, averaged over 1000 simulated economies.

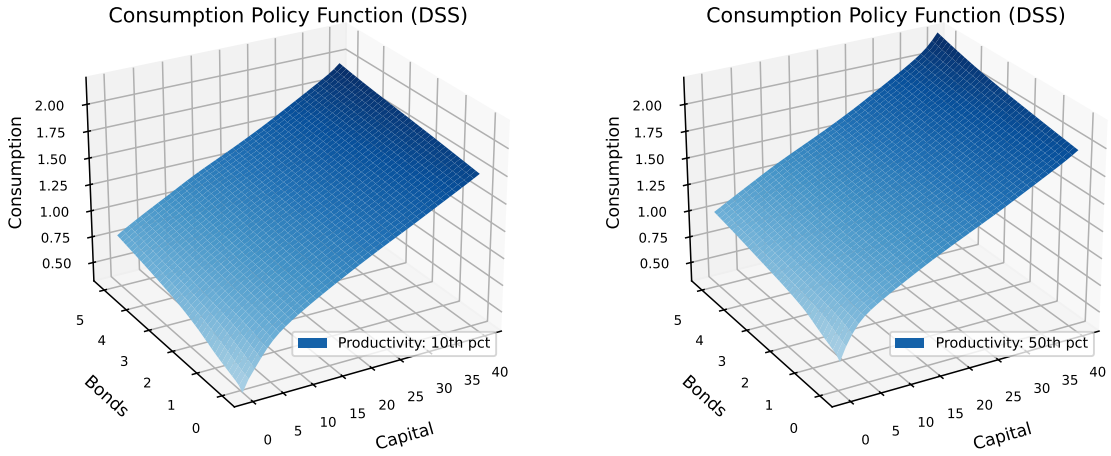

Figure 15: Policy functions. Individual consumption policy function for different levels of bond and capital holdings. The left panel reports the policy function at the 10th percentile of the productivity distribution, while the right panel reports the corresponding functions at the 50th percentile.

is now a function of two endogenous asset positions rather than a single wealth variable. The neural network must therefore learn how consumption varies jointly with capital and bonds and with idiosyncratic productivity.

The smoothness and economically coherent shape of the estimated policy functions provide an additional diagnostic of the quality of the solution. Consumption rises with household resources, while its sensitivity to either asset depends on the householdâs position in the joint asset distribution and on its productivity state.

5.5 NN-based Solution Algorithm for the Full Equilibrium

As a next step, we now solve the full equilibrium. The steps of the algorithm are as follows:

1. Set up the NNs to approximate the policy functions and initialize their weights:

The NN $\psi^I_{NN}\left(\mathbb{S}^i_t, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)$ for the individual policy functions

$$\left\{\begin{pmatrix} C^i_t \\ B^i_t \end{pmatrix} = \psi^I_{NN}\left(\mathbb{S}^i_t, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)\right\}^L_{i=1}, \tag{197}$$

and for the aggregate policy function:

$$\left\{\left(Q_t\right) = \psi^A_{NN}\left(\mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)\right\}^L_{i=1}, \tag{198}$$

2. Solve for all time t variables for a given state vector of batch b. From the NN, we have a current guess for the policy functions:

$$\{C^i_t\}^L_{i=1}. \tag{199}$$

The next step is to calculate the following (aggregate) variables:

$$C_t = \left(\frac{1}{L}\sum_{i=1}^{L} C^i_t)\right), \tag{200}$$

$$K_{t-1} = \left(\frac{1}{L}\sum_{i=1}^{L} K^i_{t-1})\right), \tag{201}$$

$$N_t = \left(\frac{1}{L}\sum_{i=1}^{L} \bar{H}\exp(s^i_t)\right), \tag{202}$$

$$Y_t = A_t K^{\alpha}_{t-1} N^{1-\alpha}_t, \tag{203}$$

$$r_t = \alpha\frac{Y_t}{K_{t-1}} - \delta, \tag{204}$$

$$w_t = (1-\alpha)\frac{Y_t}{N_t}, \tag{205}$$

$$K_t = Y_t - C_t + (1-\delta)K_{t-1}, \tag{206}$$

$$T_t = (Q_t - 1)B. \tag{207}$$

After that, we calculate each household's asset holding:

$$\left\{K^i_t = (1+r_t)K^i_{t-1} + B^i_{t-1} + w_t\bar{H}\exp(s^i_t) - C^i_t - QB^i_t - \gamma(B^i_t - \bar{B})^2 - T_t\right\}^L_{i=1}. \tag{208}$$

We use a Monte Carlo approach to evaluate the expectations. We randomly draw M sets of next period shocks to evaluate the expectations using the antithetic variates techniques. The drawn shocks are given as:

$$\left\{\left\{\epsilon^{s,i,m}_{t+1}\right\}^L_{i=1}, \epsilon^{A,m}_{t+1}\right\}^M_{m=1}, \tag{209}$$

where the last superscript m indicates which shock draw the next period value is associated with. Note that, with a slight abuse of notation, we also use the superscript m,

in the first position, to denote the monetary policy shock ϵ_t^m.

We then proceed to calculate the next period values of the stochastic state variables for all M draws, that is:

$$\left\{\left\{s_{t+1}^{i,m} = \rho_s s_t^i + \epsilon_{t+1}^{s,i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{210}$$

$$\left\{A_{t+1}^m = (1-\rho_A)A + \rho_A A_t + \epsilon_{t+1}^{A,m}\right\}_{m=1}^{M}, \tag{211}$$

which then gives us the state variables for all M shock draws.

We can then use the NNs for the individual and aggregate policy functions to calculate the individual and aggregate control variables:

$$\left\{\left\{\left(C_{t+1}^{i,m}\right) = \psi_{NN}^I\left(\mathbb{S}_{t+1}^{i,m}, \mathbb{S}_{t+1}^m, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}\right\}_{m=1}^{M} \tag{212}$$

The next step is to calculate the following (aggregate) variables for the period $t+1$ for all M draws:

$$\left\{C_{t+1}^m = \left(\frac{1}{L}\sum_{i=1}^{L} C_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{213}$$

$$\left\{N_{t+1}^m = \left(\frac{1}{L}\sum_{i=1}^{L} \bar{H}\exp(s_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{214}$$

$$\{Y_{t+1}^m = A_{t+1}K_t^{\alpha}(N_{t+1}^m)^{1-\alpha}\}_{m=1}^{M}, \tag{215}$$

$$\left\{r_{t+1}^m = \alpha\frac{Y_{t+1}^m}{K_t} - \delta\right\}_{m=1}^{M} \tag{216}$$

After that, we calculate each household's asset holding:

$$\left\{\left\{K_{t+1}^{i,m} = (1+r_{t+1}^m)K_t^i + w_{t+1}^m\bar{H}\exp(s_t^{i,m}) - C_{t+1}^{i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}. \tag{217}$$

We need to ensure that the borrowing constraint for households $K_t^i \geq \underline{K}$ is satisfied. It is convenient to define $\bar{\lambda}_t^i$ as follows:

$$\bar{\lambda}_t^i = 1 - \mu_t^i. \tag{218}$$

We calculate the multiplier for all agents L using their Euler equation:

$$\left\{\bar{\lambda}_t^i = \beta\frac{1}{M}\sum_{m=1}^{M}\left[\left(\frac{C_t^i}{C_{t+1}^{i,m}}\right)^{\sigma}(r_{t+1}^m+1)\right]\right\}_{i=1}^{L}, \tag{219}$$

, (220)

$$\left\{\bar{\psi}_t^i = \left(\beta\mathbb{E}_t\left[\left(\frac{C_t^i}{C_{t+1}^{i,m}}\right)^{\sigma}\right] - \gamma\left(B_t^i - \overline{B}\right)\right)/Q_t\right\}_{i=1}^{L}, \tag{221}$$

where we can now evaluate the expectations by taking the mean of the M draws.

We can now calculate the error associated with the Euler equation using the Fisher-Burmeister function:

$$\left\{L^{1,i} = \left(\Psi^{FB}\left(1 - \bar{\lambda}_t^i, K_t^i - \underline{K}\right)\right)^2\right\}_{i=1}^{L}, \tag{222}$$

$$\left\{L^{2,i} = \left(\Psi^{FB}\left(1 - \bar{\psi}_t^i, B_t^i - \underline{B}\right)\right)^2\right\}_{i=1}^{L}, \tag{223}$$

where $L^{1,i}$ is the squared error of agent i.

We also calculate the squared error for the restrictions, which are overidentifying restrictions:

$$L^3 = (K_t - \left(\frac{1}{L}\sum_{i=1}^{L} K_t^i\right))^2, \tag{224}$$

$$L^4 = \frac{1}{M}\sum_{m=1}^{M}\left(\left(K_{t+1}^m - \frac{1}{L}\sum_{i=1}^{L} K_{t+1}^{i,m}\right)^2\right), \tag{225}$$

$$L^5 = B - \frac{1}{L}\sum B_t^i \tag{226}$$

3. Repeat step 2 for each element in a batch B, which gives the following loss components:

$$\left\{\left\{L^{1,i,b}\right\}_{i=1}^{L}, \left\{L^{2,i,b}\right\}_{i=1}^{L} L^{3,b}, L^{4,b}, L^{5,b}\right\}_{b=1}^{B}. \tag{227}$$

4. Define the loss function:

$$\phi^L = \frac{1}{B}\sum_{b=1}^{B}\left[\sum_{i=1}^{L}\left(\alpha_1^i L^{1,i,b} + \alpha_2^i L^{2,i,b}\right) + \alpha_3 L^{3,b} + \alpha_4 L^{4,b} + \alpha_5 L^{5,b}\right], \tag{228}$$

where $\left\{\alpha_1^i\right\}_{i=1}^{L}$, α_2, α_3 determine the weights for the different equations.

5. Optimize the parameters of the NNs ψ_{NN}^I and ψ_{NN}^A to minimize the loss function ϕ^L with a stochastic gradient decent based optimizer.

6. Steps 2 - 5 are repeated N^{int} times to take several optimization steps. Redraw the shock realization in period $t+1$ for each internal optimization step.

7. Draw new parameters for each economy in the batch. This step is not required at each iteration.

8. Simulate each economy for T^{sim} periods using randomly drawn shocks. This creates then the state vector for the next iteration of the optimizer.

9. Steps 2 - 8 are repeated N^{iter} times until the NN converges on average to sufficiently low loss.

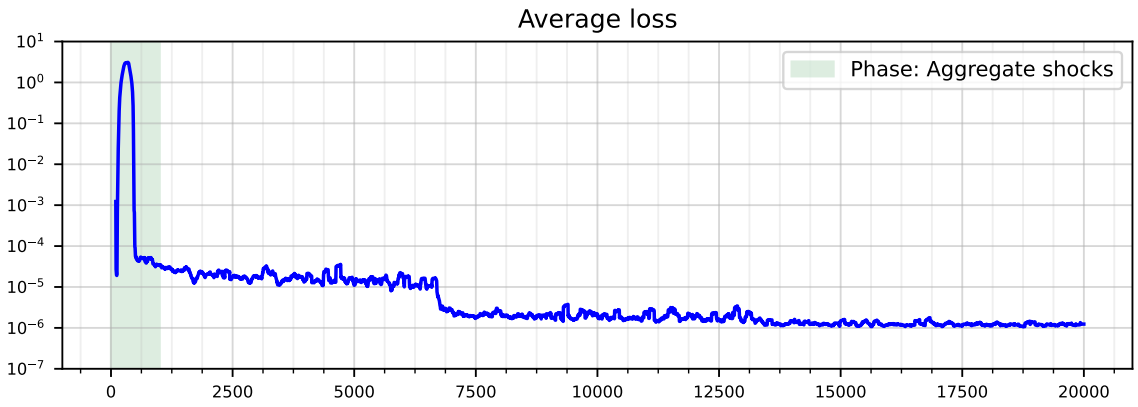

Figure 16: Convergence of the NN solution for the full equilibrium. The figure shows the dynamics of the mean squared error during the training of the full equilibrium. The shaded area indicates the periods in which we scale up aggregate risk. The vertical axis has a logarithmic scale.

5.6 NN Setup and Results: Full Equilibrium.

We present the setup and results for the full equilibrium that our code generates.

Neural Network. We specify one neural network, which contains 5 hidden layers with 128 neurons each. The activation function for the hidden layers are CELU. The optimizer is AdamW. The learning rate starts at 1×10^{-4}, and is then reduced at four fixed intervals by a factor of 0.1. The batch size is set to 50. The number of Monte Carlo integrations is set to 50. We use 100 households to compute the residuals for the individual optimization problems, while 100 households are used to keep track of the distribution and compute aggregates. Note that we choose a rather parsimonious specification on purpose to ensure that the model could still be solved in several hours on a modern laptop using the CPU.

Computational time The computations are performed on an NVIDIA RTX PRO 6000 Blackwell Server Edition (via Google Colab). The computational time for the full equilibrium is around 1h36min.

Diagnostics: Training loss and approximation error. Figure 16 displays the evolution of the average loss function, ϕ^L, over 20,000 training iterations. To facilitate convergence, we gradually introduce aggregate risk by increasing the variances of the aggregate shock from a small initial value to its calibrated level over the first 1,000 iterations. The shaded area marks the period during which aggregate risk is scaled up. The average loss declines and converges, indicating that the neural network successfully learns the full-equilibrium solution.

We then evaluate the approximation errors after the training of the neural networks is complete. As in the deterministic steady state, the relevant diagnostics are the absolute errors in the first-order conditions for capital and bonds. We simulate the model for 1,000

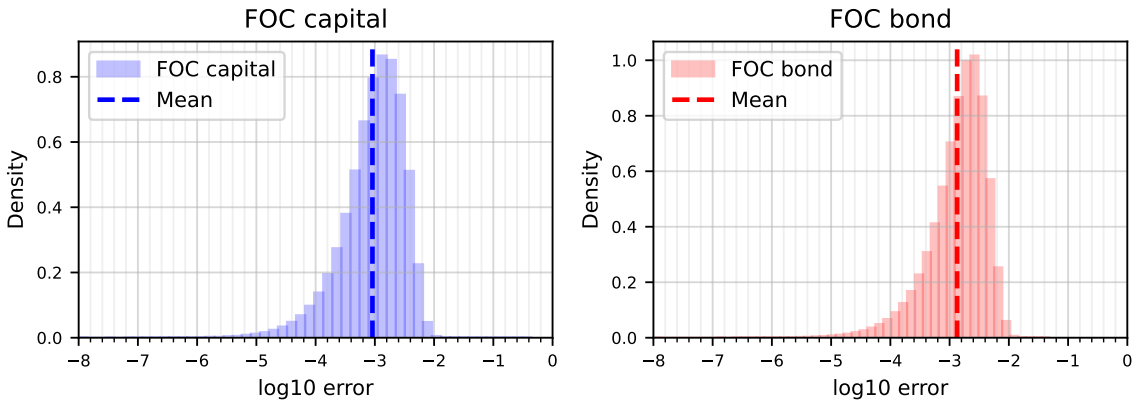

Figure 17: Distribution of post-training approximation errors. The figure shows the absolute error for the modified Euler equation (to account for the occasionally binding borrowing limit). Errors are evaluated after the training of the neural networks is complete. The distributions are based on a simulation of 1,000 periods for 100 economies, resulting in a total of 100,000 observations. We evaluate the expectations, relevant for the Euler equation using 1,000 Monte Carlo draws. The horizontal axis displays the log_{10} of the value.

periods across 100 economies and compute the approximation errors in each period.[17]

Figure 17 reports the resulting distributions. The residual errors in both conditions remain well contained, including in the right tails. The distributions are approximately bell-shaped, with their modes close to the corresponding mean absolute errors. This finding indicates that large errors are rare and unlikely to systematically distort the model dynamics generated by the approximate solution. Importantly, the approximation error for the capital condition is comparable to that obtained in the one-asset heterogeneous-agent model. In both cases, the mean post-training approximation error is close to -3 in $\log_{10}$ units, and the shapes of the error distributions are very similar.

In sum, although introducing a second asset materially increases the dimensionality and complexity of the household problem, the approximation diagnostics remain very well behaved, underscoring the reliability of our method.

Distribution. Figure 18 shows the distribution of asset holdings across different levels of discretization, with the left panel showing the histogram for capital and the left panel the for bonds. Each distribution is obtained by averaging the stochastic steady state across 100 simulated economies (batches).

Impulse response functions. Finally, Figure 19 shows the equilibrium responses of selected aggregate variables to a negative TFP shock. The responses are expressed as deviations from the stochastic steady state. To prevent idiosyncratic household shocks from affecting the reported responses, the impulse response functions are averaged across 1,000 simulated economies.

[17] We use 1,000 Monte Carlo draws to evaluate the expectations entering the household optimality conditions.

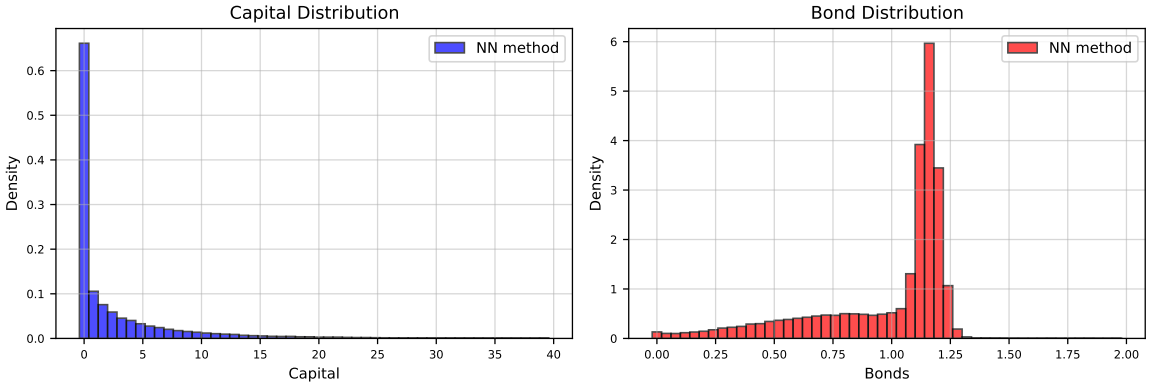

Figure 18: Asset distribution. The figure shows the histogram of capital holdings (left) and bond holdings (right). The distribution is evaluated at the stochastic steady state, averaged over 1000 simulated economies.

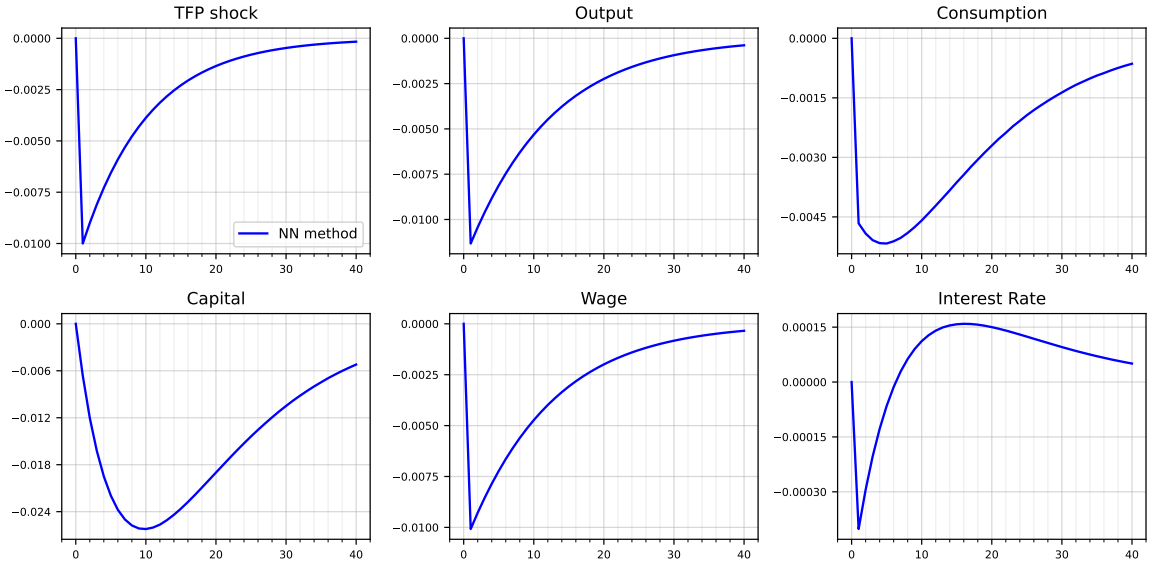

Figure 19: Impulse response functions for a one standard deviation shock. Deviations from the SSS are shown. The impulse response are averaged over 1,000 simulated economies.

The impulse responses show that the trained neural networks generate stable and economically meaningful aggregate dynamics in the two-asset environment. Thus, the exercise evaluates not only the accuracy of the household first-order conditions but also the ability of the method to accurately approximate the general equilibrium dynamics of the two-asset HANK model.

6 Heterogeneous Agent New Keynesian (HANK) model

The next model in our learning guide is a one-asset Heterogeneous Agent New Keynesian (HANK) model, following the specification of Auclert et al. (2021). We solve the model using neural networks. In our baseline specification, we use 100 agents ($L = 100$). However, we also

increase the number of agents up to 1,000, and show that in this case, 100 agents are already sufficient. Note that this setup now requires a decent GPU to solve the model.[18] The model itself is solved in two steps, as the Krusell-Smith model. Specifically, we first show how to solve the deterministic steady state. Afterward, we solve the full equilibrium of the model. Finally, we compare the results to ones obtained when using the Sequence Space Jacobian approach of Auclert et al. (2021). For ease of comparison, we follow their setup of the model.

Solution File.

- **GitHub:** To be added upon publication.

6.1 Model

The economy consists of a continuum of households. The households choose consumption C_t^i, labor H_t^i, and assets B_t^i to maximize their utility:

$$E_0 \sum_{t=0}^{\infty} \beta^t \left[\left(\frac{1}{1-\sigma} \right) \left(C_t^i \right)^{1-\sigma} - \chi \left(\frac{1}{1+\eta} \right) \left(H_t^i \right)^{1+\eta} \right].$$

The budget constraint in real terms can be written as:

$$C_t^i + B_t^i = W_t \exp(s_t^i) H_t^i + \frac{R_{t-1}}{\Pi_t} B_{t-1}^i + Div_t \exp(s_t^i) - T_t \exp(s_t^i) \tag{229}$$

where Div_t is the real dividends, W_t is the real wage, R_t is the gross nominal interest rate, T_t is the tax and Π_t is the gross inflation rate. The real taxation and real dividends are scaled with the productivity of the household. The agents individual labor productivity s_t^i is stochastic and follows an AR(1) process: $s_t^i = \rho_s s_{t-1}^i + \epsilon_t^{s,i}$. The agents face a borrowing limit $\underline{B}$, which implies:

$$B_t \geq \underline{B}. \tag{230}$$

The first-order conditions can be written as

$$1 = \beta R_t \mathbb{E}_t \left[\left(\frac{C_t^i}{C_{t+1}^i} \right)^{\sigma} \frac{1}{\Pi_{t+1}} \right] + \mu_t^i, \tag{231}$$

$$\chi (H_t^i)^{\eta} = \left(C_t^i \right)^{-\sigma} \exp(s_t^i) W_t, \tag{232}$$

where $\mu_t^i \geq 0$ is the normalized Lagrange multiplier on the individual borrowing limit in equation (230).

The New Keynesian Phillips curve, derived from Rotemberg pricing, is:

$$\varphi \log(\Pi_t) = (1-\epsilon) + \epsilon MC_t + \varphi \mathbb{E}_t \left[\frac{\Pi_{t+1}}{R_t} \log(\Pi_{t+1}) \frac{Y_{t+1}}{Y_t} \right]. \tag{233}$$

[18] To keep working with a CPU, we recommend reducing the number of households to 25.

Firms produce output using labor:

$$Y_t = N_t. \tag{234}$$

Labor is hired in competitive markets so that the wage is:

$$W_t = MC_t. \tag{235}$$

The real dividends of the firm sector can then be written as

$$Div_t = Y_t - W_t Y_t - 0.5\varphi \left[\log(\Pi_t)\right]^2 Y_t. \tag{236}$$

The central bank sets the nominal interest R_t using a Taylor rule that responds to inflation and output deviations from their targets Π and Y:

$$(R_t - 1) = (R - 1) + \theta_\Pi(\Pi_t - 1) + \theta_Y(Y_t - Y) + mp_t, \tag{237}$$

where the monetary policy shock mp_t follows an AR(1) process:

$$mp_t = \rho_m mp_{t-1} + \epsilon_t^m. \tag{238}$$

The fiscal authority sets T_t to keep their debts D_t on a constant level:

$$D = \frac{R_{t-1}}{\Pi_t} D - T_t \tag{239}$$

Market clearing for the labor market, bond market, and goods market requires

$$N_t = \int \exp(s_t^i) H_t^i di, \tag{240}$$

$$D = \int B_t^i di, \tag{241}$$

$$Y_t = \int C_t^i di + 0.5\varphi \left[\log(\Pi_t)\right]^2 Y_t, \tag{242}$$

$$s_t = \rho_s s_{t-1} + \epsilon_t^s, \tag{243}$$

$$\tag{244}$$

Equilibrium conditions. The equilibrium conditions can be written as:

$$C_t^i + B_t^i = W_t \exp(s_t^i) H_t^i + \frac{R_{t-1}}{\Pi_t} B_{t-1}^i + Div_t \exp(s_t^i) - T_t \exp(s_t^i), \quad \forall i \tag{245}$$

$$B_t^i \geq \underline{B}, \quad \forall i, \tag{246}$$

$$1 = \beta R_t \mathbb{E}_t \left[\left(\frac{C_t^i}{C_{t+1}^i} \right)^\sigma \frac{1}{\Pi_{t+1}} \right] + \mu_t^i, \quad \forall i, \tag{247}$$

$$\chi (H_t^i)^\eta = \left(C_t^i\right)^{-\sigma} \exp(s_t^i) W_t, \quad \forall i, \tag{248}$$

$$Y_t = N_t, \tag{249}$$

$$W_t = MC_t, \tag{250}$$

$$Div_t = Y_t - W_t Y_t - 0.5\varphi \left[\log(\Pi_t)\right]^2 Y_t, \tag{251}$$

$$\varphi \log(\Pi_t) = (1-\epsilon) + \epsilon MC_t + \varphi \mathbb{E}_t \left[\frac{\Pi_{t+1}}{R_t} \log(\Pi_{t+1}) \frac{Y_{t+1}}{Y_t}\right], \tag{252}$$

$$R_t = R + \theta_\Pi(\Pi_t - 1) + \theta_Y(Y_t - Y) + mp_t, \tag{253}$$

$$D = \frac{R_{t-1}}{\Pi_t} D - T_t, \tag{254}$$

$$N_t = \int \exp(s_t^i) H_t^i di, \tag{255}$$

$$D = \int B_t^i di, \tag{256}$$

$$Y_t = \int C_t^i di + 0.5\varphi \left[\log(\Pi_t)\right]^2 Y_t, \tag{257}$$

$$s_t = \rho_s s_{t-1} + \epsilon_t^s, \tag{258}$$

$$mp_t = \rho_m mp_{t-1} + \epsilon_t^m. \tag{259}$$

Calibration. We calibrate the model in line with Auclert et al. (2021): $\beta = 0.982$, $\sigma = 2.0$, $\eta = 2$, $\chi = 0.786$, $\underline{B} = 0$, $\epsilon = 6$, $\varphi = 60$, $\theta_\Pi = 1.5$, $\theta_Y = 0$, $D = 5.6$, $\rho_s = 0.966$, $\rho_{mp} = 0.6$, $\sigma_s = 0.129$, $\rho_{mp} = 0.9$, $\sigma_{mp} = 0.0025$.

6.2 Mapping of the model to the general framework

We can map this HANK model in the general form of the outlined estimation procedure. The state variable, the structural shocks, the individual and aggregate control variables, and the needed deterministic steady state values are:

$$\mathbb{S}_t = \left\{ \left\{B_{t-1}^i\right\}_{i=1}^L, \left\{s_t^i\right\}_{i=1}^L, R_{t-1}^N, mp_t \right\}, \tag{260}$$

$$\nu_t = \left\{ \left\{\epsilon_t^{s,i}\right\}_{i=1}^L, \epsilon_t^{mp} \right\}, \tag{261}$$

$$\psi_t^I = \left\{H_t^i\right\}, \tag{262}$$

$$\psi_t^A = \left\{\Pi_t, W_t\right\}, \tag{263}$$

$$\psi^{DSS} = \{R\} \tag{264}$$

The parameters of the model are

$$\Theta = \{\beta, \sigma, \eta, \underline{B}, \chi, \theta_\Pi, \theta_Y, D, \epsilon, \varphi, \rho_s, \rho_m, \sigma_s, \sigma_m\} \tag{265}$$

When solving the model, we treat all parameters as calibrated.

We use two NNs to separate individual and aggregate policy functions. The individual NN $\psi_{NN}^I(\cdot)$ solves for labor supply:

$$\left\{ \left(H_t^i\right) = \psi_{NN}^I \left(\mathbb{S}_t^i, \mathbb{S}_t, \tilde{\Theta} | \bar{\Theta}\right) \right\}_{i=1}^L, \tag{266}$$

where $\mathbb{S}_t^i = \left\{B_{t-1}^i, s_t^i\right\}$. The NN for the aggregate policy functions $\psi_{NN}^A(\cdot)$ determines the

wage and inflation:

$$\begin{pmatrix} \Pi_t \\ W_t \end{pmatrix} = \psi_{NN}^{A}\left(\mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right). \tag{267}$$

One challenge is that we must solve for the deterministic steady state values of the nominal rate, as the Taylor rule depends on them. This neural networks $\psi_{NN}^{SS}(\cdot)$ that maps the parameter values to the deterministic steady state values is:

$$\begin{pmatrix} R \\ Y \end{pmatrix} = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{268}$$

Note that for $\theta_Y = 0$, we would not need to solve for Y.

6.3 NN-based solution algorithm for the deterministic steady state

We need the mapping from the parameters to the deterministic steady state values of the interest rate R and output Y, as these objects enter the Taylor rule. To obtain this mapping, we solve for the deterministic steady state (DSS) of our HANK model, in which agents face and experience idiosyncratic risk while aggregate risk is absent.

The DSS NN solves for the deterministic steady state values:

$$\begin{pmatrix} R \\ Y \end{pmatrix} = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{269}$$

To solve for this NN, we also need to solve for the individual policy functions of the agents in the DSS.

$$\left\{\left(H_t^i\right) = \psi_{NN}^{I,DSS}\left(\mathbb{S}_t^i, \mathbb{S}, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}, \tag{270}$$

where DSS enters the superscript of the NN for a clear distinction. Note that we let the deterministic steady state values of the aggregate state variables still enter this network. While this is unnecessary for solving for the DSS, it allows us to use this trained NN as a starting point when we solve for the economy with aggregate risk.

1. Set up the NNs to approximate the policy functions and initialize their weights:

   The NN $\psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right)$ for the steady state parameters:

   $$\begin{pmatrix} R \\ Y \end{pmatrix} = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{271}$$

   The NN $\psi_{NN}^{I,DSS}\left(\mathbb{S}_t^i, \mathbb{S}, \tilde{\Theta}|\bar{\Theta}\right)$ for the individual policy functions without aggregate risk:

   $$\left\{\left(H_t^i\right) = \psi_{NN}^{I,DSS}\left(\mathbb{S}_t^i, \mathbb{S}, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}. \tag{272}$$

2. Solve for all the current period individual variables. From the NN, we get a current guess for the deterministic steady state values

$$\begin{pmatrix} R \\ Y \end{pmatrix} = \psi_{NN}^{SS}\left(\tilde{\Theta}|\bar{\Theta}\right), \tag{273}$$

and individual policy functions

$$\{H_t^i\}_{i=1}^L. \tag{274}$$

Note that inflation is at target in the DSS, and the wage and marginal cost equal its value in the DSS without idiosyncratic risk. Thus, we know:

$$\Pi, W, MC. \tag{275}$$

The next step is to calculate the following variables:

$$N_t = \left(\frac{1}{L}\sum_{i=1}^{L} H_t^i \exp(s_t^i)\right), \tag{276}$$

$$Y_t = N_t, \tag{277}$$

$$T_t = \left(\frac{R_{t-1}}{\Pi_t} - 1\right) D \tag{278}$$

$$Div_t = Y_t - W_t N_t - 0.5\varphi \log(\Pi)^2 Y_t. \tag{279}$$

As before, we use a subscript t here to indicate variables calculated based on individual shock realizations.

We pursue calculating each household's individual variables:

$$\left\{C_t^i = \left(\frac{\exp(s_t^i)W}{\chi(H_t^i)^\eta}\right)^{1/\sigma}\right\}_{i=1}^{L}, \tag{280}$$

$$\left\{\omega_t^i = W\exp(s_t^i)H_t^i + \frac{R}{\Pi}B_{t-1}^i + Div_t\exp(s_t^i) - T_t\exp(s_t^i)\right\}_{i=1}^{L}, \tag{281}$$

$$\left\{B_t^i = \omega_t^i - C_t^i\right\}_{i=1}^{L}. \tag{282}$$

Aggregate consumption is given as:

$$C_t = \frac{1}{L}\sum_{i=1}^{L} C_t^i. \tag{283}$$

We use a Monte Carlo approach to evaluate the expectations, which requires randomly drawing M sets of next period shocks for the households (we also use the antithetic

variate method here). The drawn shocks are given as:

$$\left\{\left\{\epsilon_{t+1}^{s,i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{284}$$

where the superscript m indicates which shock draw the next period value is associated with.

We then proceed to calculate the next period values of the stochastic state variables for all M draws, that is:

$$\left\{\left\{s_{t+1}^{i,m}=\rho_s s_t^i+\epsilon_{t+1}^{s,i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}. \tag{285}$$

This gives us the household state variables for all M shock draws.

We can then use the NN for the individual policy functions in the DSS:

$$\left\{\left\{\left(H_{t+1}^{i,m}\right)=\psi_{NN}^{I,DSS}\left(\mathbb{S}_{t+1}^{i,m},\mathbb{S}^m,\tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}\right\}_{m=1}^{M}. \tag{286}$$

The next step is to calculate the following variables for the period $t+1$ for all M draws:

$$\left\{N_{t+1}^m=\left(\frac{1}{L}\sum_{i=1}^{L}H_{t+1}^{i,m}\exp(s_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{287}$$

$$\left\{Y_{t+1}^m=N_{t+1}^m\right\}_{m=1}^{M}, \tag{288}$$

$$\left\{T_{t+1}^m=\left(\frac{R_t}{\Pi_{t+1}^m}-1\right)D\right\}_{m=1}^{M}, \tag{289}$$

$$\left\{Div_{t+1}=Y_{t+1}^m-W_{t+1}^m N_{t+1}^m-0.5\varphi\log(\Pi)^2 Y_{t+1}^m\right\}_{m=1}^{M}. \tag{290}$$

We proceed to calculate each household's individual variables for the period $t+1$ across all M draws:

$$\left\{\left\{C_{t+1}^{i,m}=\left(\frac{\exp(s_{t+1}^{i,m})W}{\chi(H_{t+1}^{i,m})^{\eta}}\right)^{1/\sigma}\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{291}$$

$$\left\{\left\{\omega_{t+1}^{i,m}=W\exp(s_{t+1}^{i,m})H_{t+1}^{i,m}+\frac{R}{\Pi}B_t^i+Div_{t+1}^m\exp(s_{t+1}^{i,m})-T_{t+1}^m\exp(s_{t+1}^{i,m})\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{292}$$

$$\left\{\left\{B_{t+1}^{i,m}=\omega_{t+1}^{i,m}-C_{t+1}^{i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}. \tag{293}$$

Aggregate consumption is given as:

$$\left\{C_{t+1}^m=\frac{1}{L}\sum_{i=1}^{L}C_{t+1}^{i,m}\right\}_{m=1}^{M}. \tag{294}$$

We need to ensure that the borrowing constraint of the households $B_t^i \geq \underline{B}$ is satisfied. It is convenient to define $\bar{\lambda}_t^i$ as follows:

$$\bar{\lambda}_t^i = 1 - \mu_t^i. \tag{295}$$

We calculate the multiplier for all agents L using their Euler equation:

$$\left\{ \bar{\lambda}_t^i = \beta R \frac{1}{M} \sum_{m=1}^{M} \left[\left(\frac{C_t^i}{C_{t+1}^{i,m}} \right)^{\sigma} \frac{1}{\Pi} \right] \right\}_{i=1}^{L}, \tag{296}$$

where we can now evaluate the expectations by taking the mean over the M draws.

Equipped with this object, we use the Fischer-Burmeister function Ψ^{FB} to enforce the Kuhn-Tucker conditions again. This can be written as:

$$\Psi^{FB}\left(1 - \bar{\lambda}_t^i, B_t^i - \underline{B}\right). \tag{297}$$

We can now calculate the error associated with the Fischer-Burmeister function:

$$\left\{ L^{1,DSS,i} = \left(\Psi^{FB}\left(1 - \bar{\lambda}_t^i, B_t^i - \underline{B}\right) \right)^2 \right\}_{i=1}^{L}, \tag{298}$$

where $L^{1,DSS,i}$ is the squared Euler error of agent i.

We also calculate the squared error for aggregate output, the resource constraint in period t and $t+1$, and market clearing for the bond in period t and $t+1$:

$$\begin{aligned}
L^{2,DSS} &= \left(Y_t - Y\right)^2, \\
L^{3,DSS} &= \left(D - \frac{1}{L} \sum_{i=1}^{L} B_t^i \right)^2, && (299) \\
L^{4,DSS} &= \frac{1}{M} \sum_{m=1}^{M} \left(D - \frac{1}{L} \sum_{i=1}^{L} B_{t+1}^{i,m} \right)^2, && (300) \\
L^{5,DSS} &= \left(Y_t - C_t - 0.5\varphi \log(\Pi)^2 Y_t \right)^2, && (301) \\
L^{6,DSS} &= \frac{1}{M} \sum_{m=1}^{M} \left(Y_{t+1}^m - C_{t+1}^m - 0.5\varphi \log(\Pi)^2 Y_{t+1}^m \right)^2. && (302)
\end{aligned}$$

3. Repeat step 2 for each element in a batch B, which gives the following loss components for the entire batch:

$$\left\{ \left\{ L^{1,DSS,i,b} \right\}_{i=1}^{L}, L^{2,DSS,b}, L^{3,DSS,b}, L^{4,DSS,b}, L^{5,DSS,b}, L^{6,DSS,b} \right\}_{b=1}^{B}. \tag{303}$$

4. Define the loss function:

$$\phi^L = \frac{1}{B} \sum_{b=1}^{B} \left[\sum_{i=1}^{L} \alpha_1^{i,DSS} L_1^{1,DSS,i,b} + \sum_{j=2}^{6} \alpha_j^{DSS} L^{j,DSS,b} \right], \tag{304}$$

where the weights for the equations are determined by $\left\{\alpha_1^{i,DSS}\right\}_{i=1}^{L}$ and $\alpha_j^{DSS}, \forall j = 2,3,4,5,6$.

5. Optimize the parameters of the NNs ψ_{NN}^{SS} and $\psi_{NN}^{I,DSS}$ to minimize the loss function ϕ^L with a stochastic gradient decent based optimizer.

6. Steps 2 - 5 are repeated N^{int} times to take several optimization steps. Redraw the shock realization in period $t+1$ for each internal optimization step.

7. Draw new parameters for each economy in the batch. This step is not required at each iteration.

8. Simulate each economy for T^{sim} periods using randomly drawn shocks. This creates the state vector for the next iteration of the optimizer.

9. Steps 2 - 8 are repeated N^{iter} times until the NN converges on average to sufficiently low loss.

6.4 NN Setup and Results: Deterministic Steady State

We present the setup and results for the deterministic steady state that our code generates.

Neural Network. We specify two neural networks. Both neural networks contain 5 hidden layers with 128 neurons each. The activation function for the hidden layers are CELU. The optimizer is AdamW. The learning rate starts at 1×10^{-4} and is annealed to 1×10^{-8} following a cosine schedule. The batch size is set to 50. The number of Monte Carlo integrations is set to 100. We use 100 households to compute the residuals for the individual optimization problems, while 100 households are used to keep track of the distribution and compute aggregates. We use 10 internal optimization steps.

Computational time. The computations are performed on an NVIDIA GeForce RTX 5090 GPU with 32 GB of memory. The computational time for the DSS is around 0h43min. The GPU memory usage is 0.6 GB.

Diagnostics. The top panel of Figure 20 displays the evolution of the average loss function ϕ^L over the training. In the lower panels of that figure, we plot the in-training evolution of the Euler equation and for the output normalization To facilitate convergence during training, we gradually introduce idiosyncratic shocks by increasing their variance from a small value to their target levels over the course of the first 1,000 iterations. The red shaded area marks the stage at which this increase begins.

The training is successful, with clear convergence of the average loss. The breakdown of the loss into the individual residual errors confirms that the neural network learns all policy functions of interest effectively.

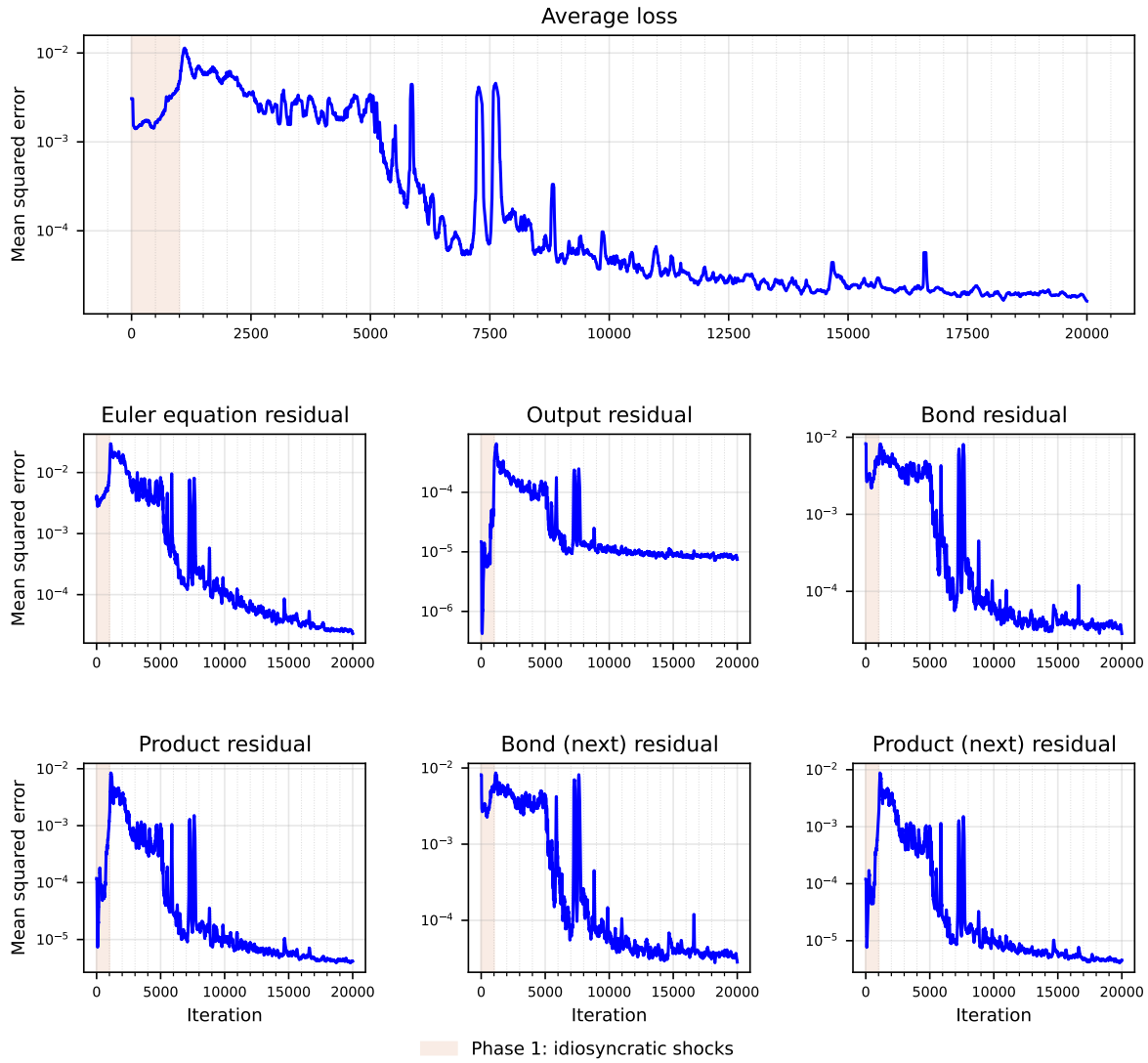

Figure 20: Convergence of the NN solution for the deterministic steady state. The figure shows the dynamics of the mean squared error during the training of the DSS. The upper plot shows the average loss, while the subcomponents are shown below. The shaded area indicates the period in which we scale up idiosyncratic risk. The vertical axis has a logarithmic scale.

Note that we skip the approximation error and the DSS to be concise. The accompanied codes calculates both.

Individual policy functions. Figure 21 shows the bar graph of the asset distribution conditional on various levels of productivity (10th percentile, 50th percentile, and 90th percentile) along with the policy functions of labor supply and consumption.

6.5 NN-based solution algorithm for the full equilibrium

The algorithm uses the outlined NN approach to solve the full equilibrium of the HANK model. We use two NNs to separate individual and aggregate policy functions. The individual

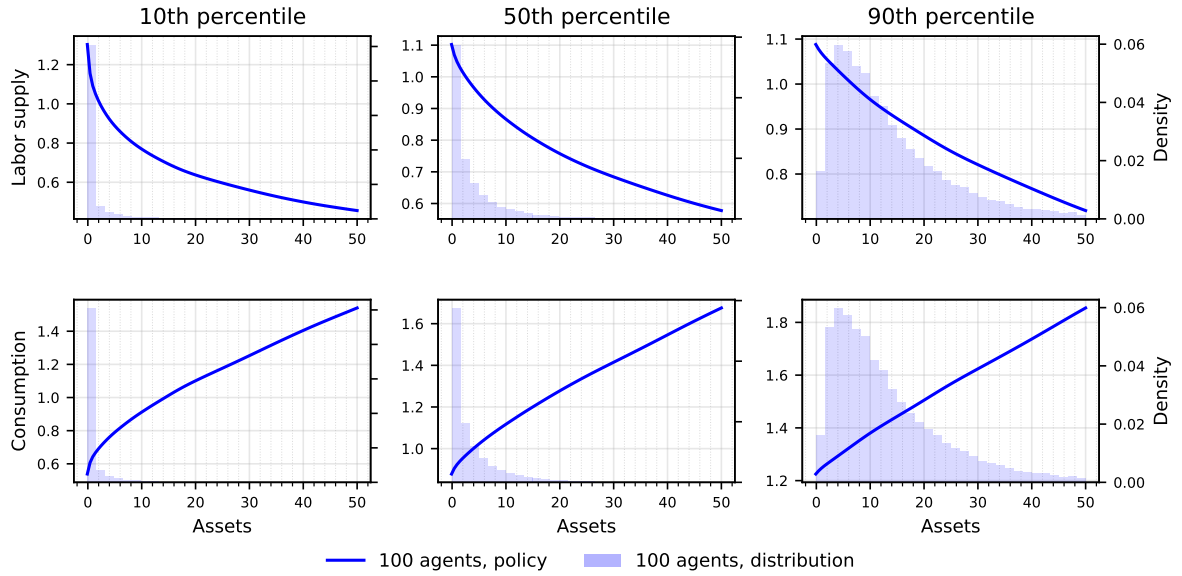

Figure 21: Policy functions. Policy functions (labor supply - upper panels - and consumption - lower panels) and asset distribution conditional on individual productivity (10th percentile, 50th percentile, 90th percentile).

NN $\psi^I_{NN}(\cdot)$ solves for labor supply:

$$\left\{\left(H^i_t\right) = \psi^I_{NN}\left(\mathbb{S}^i_t, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)\right\}^L_{i=1}, \tag{305}$$

where $\mathbb{S}^i_t = \left\{B^i_{t-1}, s^i_t\right\}$. The NN for the aggregate policy functions $\psi^A_{NN}(\cdot)$ determines the wage and inflation:

$$\begin{pmatrix}\Pi_t \\ W_t\end{pmatrix} = \psi^A_{NN}\left(\mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right). \tag{306}$$

We use our trained neural network for the deterministic steady state $\psi^{SS}_{NN}(\cdot)$, which maps the parameter values to the deterministic steady state values:

$$\begin{pmatrix}R \\ Y\end{pmatrix} = \psi^{SS}_{NN}\left(\tilde{\Theta}|\bar{\Theta}\right). \tag{307}$$

The steps of the algorithm are as follows:

1. Set up the NNs to approximate the policy functions and initialize their weights:

   The NN $\psi^I_{NN}\left(\mathbb{S}^i_t, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)$ for the individual policy functions

   $$\left\{\left(H^i_t\right) = \psi^I_{NN}\left(\mathbb{S}^i_t, \mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)\right\}^L_{i=1}. \tag{308}$$

   The NN $\psi^A_{NN}\left(\mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right)$ for the aggregate policy functions:

   $$\begin{pmatrix}\Pi_t \\ W_t\end{pmatrix} = \psi^A_{NN}\left(\mathbb{S}_t, \tilde{\Theta}|\bar{\Theta}\right). \tag{309}$$

2. Solve for all time t variables for a given state vector of batch b. From the NN, we have a current guess for the policy functions:

$$\{H_t^i\}_{i=1}^L, \Pi_t, W_t, \tag{310}$$

and the deterministic steady state values of R and Y are given by equation (307).

The next step is to calculate the following (aggregate) variables:

$$N_t = \left(\frac{1}{L}\sum_{i=1}^{L} H_t^i \exp(s_t^i)\right), \tag{311}$$

$$Y_t = N_t, \tag{312}$$

$$T_t = \left(\frac{R_{t-1}}{\Pi_t} - 1\right) D, \tag{313}$$

$$MC_t = W_t, \tag{314}$$

$$Div_t = Y_t - W_t N_t - 0.5\varphi \log(\Pi_t)^2 Y_t, \tag{315}$$

$$R_t = R + \theta_\Pi(\Pi_t - 1) + \theta_Y(Y_t - Y) + mp_t. \tag{316}$$

After that, we calculate each household's individual variables:

$$\left\{C_t^i = \left(\frac{\exp(s_t^i)W_t}{\chi(H_t^i)^\eta}\right)^{1/\sigma}\right\}_{i=1}^{L}, \tag{317}$$

$$\left\{\omega_t^i = W_t \exp(s_t^i) H_t^i + \frac{R_{t-1}}{\Pi_t} B_{t-1}^i + Div_t \exp(s_t^i) - T_t \exp(s_t^i)\right\}_{i=1}^{L}, \tag{318}$$

$$\left\{B_t^i = \omega_t^i - C_t^i\right\}_{i=1}^{L}. \tag{319}$$

Aggregate consumption is:

$$C_t = \frac{1}{L}\sum_{i=1}^{L} C_t^i. \tag{320}$$

To evaluate the expectations, we randomly draw M sets of next period shocks. We also use the antithetic variates technique here. The drawn shocks are:

$$\left\{\left\{\epsilon_{t+1}^{s,i,m}\right\}_{i=1}^{L}, \epsilon_{t+1}^{m,m}\right\}_{m=1}^{M}, \tag{321}$$

where the last superscript m indicates which shock draw the next period value is associated with. Note that, with a slight abuse of notation, we also use the superscript m, in the first position, to denote the monetary policy shock ϵ_t^m.

We then proceed to calculate the next period values of the stochastic state variables for all M draws, that is:

$$\left\{\left\{s_{t+1}^{i,m} = \rho_s s_t^i + \epsilon_{t+1}^{s,i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{322}$$

$$\left\{mp_{t+1}^{m} = \rho_m mp_t + \epsilon_{t+1}^{m,m}\right\}_{m=1}^{M}, \tag{323}$$

which then gives us the state variables for all M shock draws.

We can then use the NNs for the individual and aggregate policy functions to calculate the individual and aggregate control variables:

$$\left\{\left\{\left(H_{t+1}^{i,m}\right) = \psi_{NN}^{I}\left(\mathbb{S}_{t+1}^{i,m}, \mathbb{S}_{t+1}^{m}, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{324}$$

$$\left\{\begin{pmatrix}\Pi_{t+1}^{m} \\ W_{t+1}^{m}\end{pmatrix} = \psi_{NN}^{A}\left(\mathbb{S}_{t+1}^{m}, \tilde{\Theta}|\bar{\Theta}\right)\right\}_{m=1}^{M}. \tag{325}$$

The next step is to calculate the following (aggregate) variables for the period $t+1$ for all M draws:

$$\left\{N_{t+1}^{m} = \left(\frac{1}{L}\sum_{i=1}^{L} H_{t+1}^{i,m}\exp(s_{t+1}^{i,m})\right)\right\}_{m=1}^{M}, \tag{326}$$

$$\left\{Y_{t+1}^{m} = N_{t+1}^{m}\right\}_{m=1}^{M}, \tag{327}$$

$$\left\{T_{t+1}^{m} = \left(\frac{R_t}{\Pi_{t+1}^{m} a_{t+1}^{m}} - 1\right) D\right\}_{m=1}^{M}, \tag{328}$$

$$\left\{Div_{t+1}^{m} = Y_{t+1}^{m} - W_{t+1}^{m} N_{t+1}^{;} - 0.5\varphi \log D\right\}_{m=1}^{M}, \tag{329}$$

We proceed with calculating for each household their individual variables in period $t+1$ for all M draws:

$$\left\{\left\{C_{t+1}^{i,m} = \left(\frac{\exp(s_{t+1}^{i,m}) W_{t+1}^{m}}{\chi (H_{t+1}^{i,m})^{\eta}}\right)^{1/\sigma}\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{330}$$

$$\left\{\left\{\omega_{t+1}^{i,m} = W_{t+1}^{m}\exp(s_{t+1}^{i,m}) H_{t+1}^{i,m} + \frac{R_t}{\Pi_{t+1}^{m}} B_t^{i} + Div_{t+1}^{m}\exp(s_{t+1}^{i,m}) - T_{t+1}^{m}\exp(s_{t+1}^{i,m})\right\}_{i=1}^{L}\right\}_{m=1}^{M}, \tag{331}$$

$$\left\{\left\{B_{t+1}^{i,m} = \omega_{t+1}^{i,m} - C_{t+1}^{i,m}\right\}_{i=1}^{L}\right\}_{m=1}^{M}. \tag{332}$$

Aggregate consumption is given as:

$$\left\{C_{t+1}^{m} = \frac{1}{L}\sum_{i=1}^{L} C_{t+1}^{i,m}\right\}_{m=1}^{M}. \tag{333}$$

We need to ensure that the borrowing constraints for households $B_t^i \geq \underline{B}$ are satisfied, si that we define $\bar{\lambda}_t^i$ as follows:

$$\bar{\lambda}_t^i = 1 - \mu_t^i. \tag{334}$$

We calculate the multiplier for all agents L using their Euler equation:

$$\left\{\bar{\lambda}_t^i = \beta R_t \frac{1}{M}\sum_{m=1}^{M}\left[\left(\frac{C_t^i}{C_{t+1}^{i,m}}\right)^{\sigma}\frac{1}{\Pi_{t+1}^m}\right]\right\}_{i=1}^{L}, \tag{335}$$

where we can now evaluate the expectations by taking the mean of the M draws.

Using the Fischer-Burmeister function Ψ^{FB}, we write the Euler equation as:

$$\Psi^{FB}\left(1-\bar{\lambda}_t^i, B_t^i - \underline{B}\right). \tag{336}$$

We can now calculate the error associated with the Fisher-Burmeister function:

$$\left\{L^{1,i} = \left(\Psi^{FB}\left(1-\bar{\lambda}_t^i, B_t^i - \underline{B}\right)\right)^2\right\}_{i=1}^{L}, \tag{337}$$

where $L^{1,i}$ is the squared error of agent i.

We also calculate the squared error for the New Keynesian Phillips curve, the resource constraint in period t and $t+1$, and market clearing for the bond in period t and $t+1$:

$$\begin{aligned}
L^2 &= \left(\varphi\log(\Pi_t) - \left((1-\epsilon+\epsilon MC_t + \varphi\mathbb{E}_t\left[\frac{\Pi_{t+1}}{R_t}\log(\Pi_{t+1})\frac{Y_{t+1}}{Y_t}\right]\right)\right)^2, \\
L^3 &= \left(D - \frac{1}{L}\sum_{i=1}^{L}B_t^i\right)^2, \qquad (338)\\
L^4 &= \left(Y_t - \frac{1}{L}\sum_{i=1}^{L}C_t^i - 0.5\varphi\left[\log(\Pi_t)\right]^2 Y_t\right)^2 \qquad (339)\\
L^5 &= \frac{1}{M}\sum_{m=1}^{M}\left(\left(D - \frac{1}{L}\sum_{i=1}^{L}B_{t+1}^{i,m}\right)^2\right), \qquad (340)\\
L^6 &= \frac{1}{M}\sum_{m=1}^{M}\left(\left(Y_{t+1}^m - \frac{1}{L}\sum_{i=1}^{L}C_{t+1}^{i,m} - 0.5\varphi\left[\log(\Pi_{t+1}^m)\right]^2 Y_{t+1}^m\right)^2\right). \qquad (341)
\end{aligned}$$

3. Repeat step 2 for each element in a batch B, which gives the following loss components:

$$\left\{\left\{L^{1,i,b}\right\}_{i=1}^{L}, L^{2,b}, L^{3,b}, L^{4,b}, L^{5,b}, L^{6,b}\right\}_{b=1}^{B}. \tag{342}$$

4. Define the loss function:

$$\phi^L = \frac{1}{B}\sum_{b=1}^{B}\left[\sum_{i=1}^{L}\alpha_1^i L_1^{1,i,b} + \alpha_2 L^{2,b} + \alpha_3 L^{3,b} + \alpha_4 L^{4,b} + \alpha_5 L^{5,b} + \alpha_6 L^{6,b}\right], \tag{343}$$

where $\left\{\alpha_1^i\right\}_{i=1}^{L}, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6$ determine the weights for the different equations.

5. Optimize the parameters of the NNs ψ_{NN}^I and ψ_{NN}^A to minimize the loss function ϕ^L with a stochastic gradient decent based optimizer.

6. Steps 2 - 5 are repeated N^{int} times to take several optimization steps. Redraw the shock realization in period $t+1$ for each internal optimization step.

7. Draw new parameters for each economy in the batch. This step is not required at each iteration.

8. Simulate each economy for T^{sim} periods using randomly drawn shocks. This creates then the state vector for the next iteration of the optimizer.

9. Steps 2 - 8 are repeated N^{iter} times until the NN converges on average to sufficiently low loss.

6.6 NN Setup and Results: Full Equilibrium

We present the setup and results for the full equilibrium.

Neural Network. We specify two neural networks. Both neural networks contain 5 hidden layers with 128 neurons each. The activation function for the hidden layers are CELU. The optimizer is AdamW. The learning rate starts at 1×10^{-4} and is annealed to 1×10^{-8} following a cosine schedule. The batch size is set to 50. The number of Monte Carlo integrations is set to 100. We use 100 households to compute the residuals for the individual optimization problems, while 100 households are used to keep track of the distribution and compute aggregates. Note that we solve this model on a GPU. We use 10 internal optimization steps.

Computational time. The computations are performed on an NVIDIA GeForce RTX 5090 GPU with 32 GB of memory. The computational time for the full equilibrium is around 0h46min. The GPU memory usage is 0.6 GB.

Diagnostics. The left panel of Figure 22 displays the evolution of the average loss function ϕ^L over the training for 20,000 iterations. In the other panels of that figure, we plot the in-training evolution of the Euler equation, the New Keynesian Phillips Curve (NKPC), bond market clearing in the current as well as next period, and goods market clearing in the current as well as next period. To facilitate convergence during training, we gradually introduce idiosyncratic and aggregate shocks by increasing their variance from a small value to their target levels over the course of the 1,000 iterations. The red shaded area marks the stage at which this increase begins. Afterward, we introduce aggregate risk between iterations 1,000 and 2,000.

The training is successful, with clear convergence of the average loss. The breakdown of the loss into the individual residual errors confirms that the neural network learns all policy functions of interest effectively.

The next step is to calculate the approximation errors *after* the training of the neural networks has been completed. We use the same set of equilibrium conditions used during

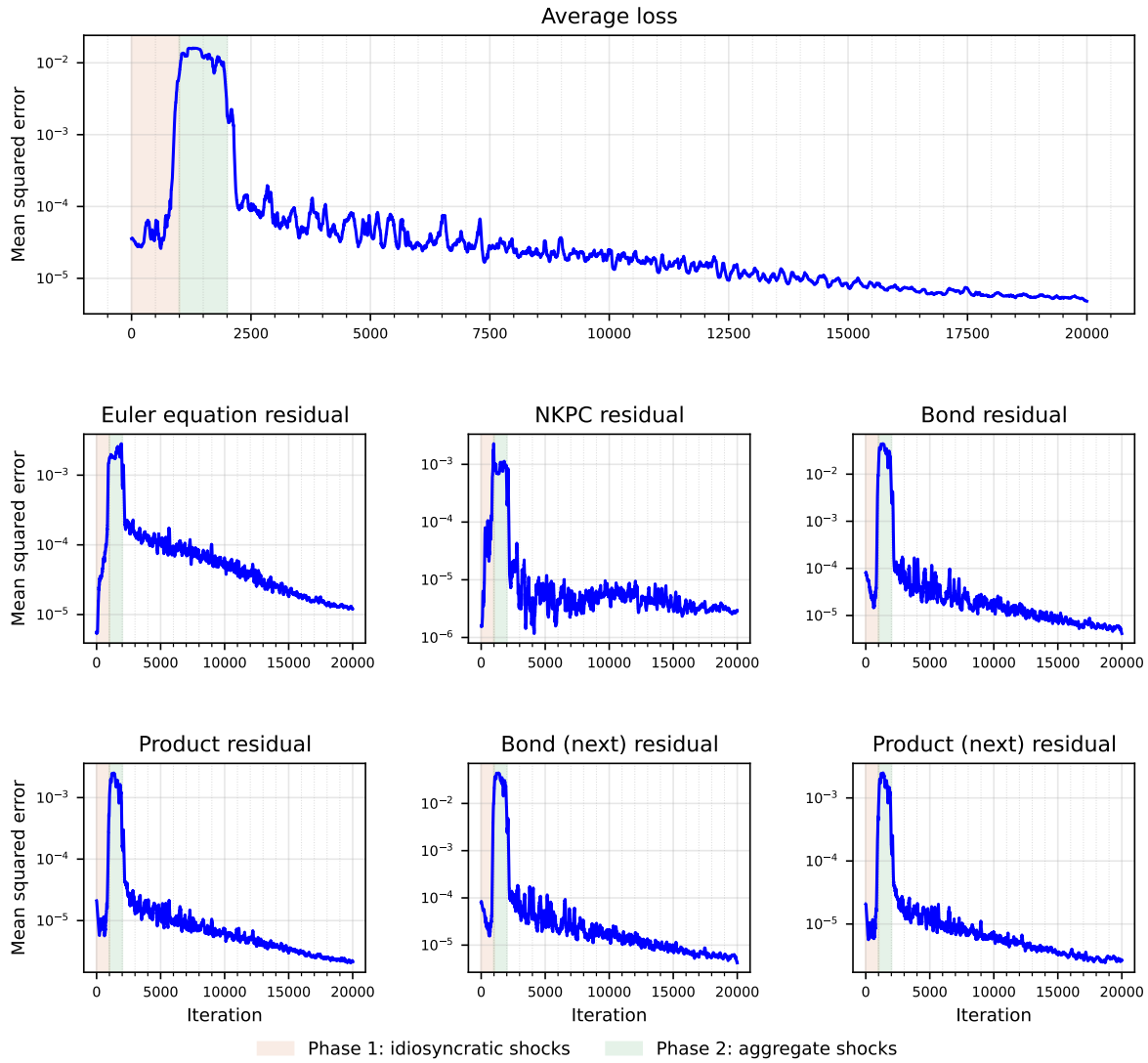

Figure 22: Convergence of the NN solution for the full equilibrium. The figure shows the dynamics of the mean squared error during the training of the full equilibrium. The upper plot shows the average loss, while the subcomponents are shown below. The shaded area indicates the periods in which we scale up aggregate risk. The vertical axis has a logarithmic scale.

training, which are the Euler equation, the New Keynesian Phillips Curve (NKPC), bond market clearing, and goods market clearing. The absolute error for these equations can be written as:

$$err_1 = \left| \Psi^{FB} \left(1 - \beta R_t \mathbb{E}_t \left[\left(\frac{C_t^i}{C_{t+1}} \right)^{\sigma} \frac{1}{\Pi_{t+1}} \right], B_t^i - \underline{B} \right) \right|, \tag{344}$$

$$err_2 = \left| \frac{\varphi \log(\Pi_t)}{1-\epsilon} - \left((1 + \frac{\epsilon}{1-\epsilon} MC_t + \frac{\varphi}{1-\epsilon} \mathbb{E}_t \left[\frac{\Pi_{t+1}}{R_t} \log(\Pi_{t+1}) \frac{Y_{t+1}}{Y_t} \right] \right) \right|, \tag{345}$$

$$err_3 = \left| 1 - \frac{\frac{1}{L} \sum_{i=1}^{L} B_t^i}{D} \right|, \tag{346}$$

$$err_4 = \left| 1 - \frac{\frac{1}{L} \sum_{i=1}^{L} C_t^i - 0.5\varphi \left[\log(\Pi_t) \right]^2 Y_t}{Y_t} \right| \tag{347}$$

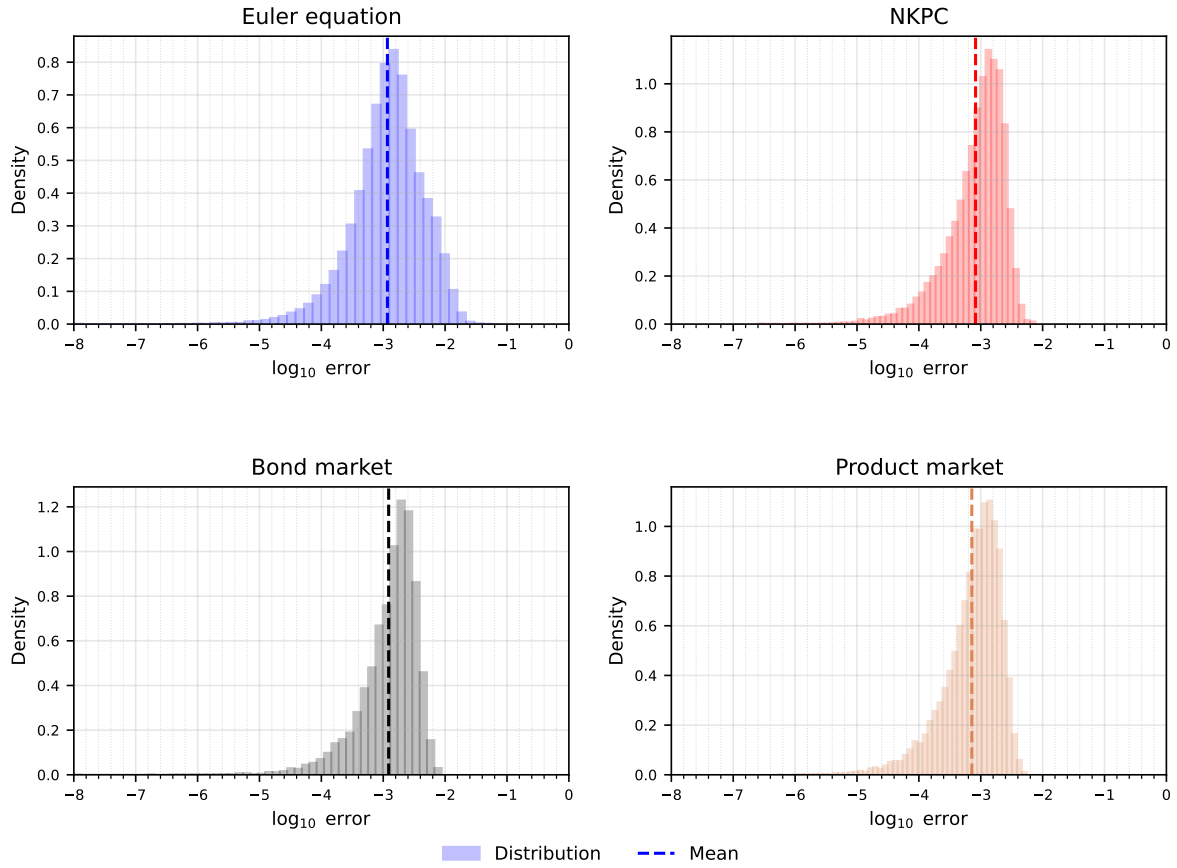

Figure 23: Distribution of post-training approximation errors. The figure shows the absolute error for the modified Euler equation (to account for the occasionally binding borrowing limit), the New Keynesian Phillips curve, bond market clearing and product market clearing. Errors are evaluated after the training of the neural networks is complete. The distributions are based on a simulation of 10,000 periods. We evaluate the expectations, relevant to the modified Euler equation and the NKPC using 1,000 Monte Carlo draws. The horizontal axis displays the log_{10} of the value.

where the err_1 is for the Euler equation, err_2 for the NKPC, err_3 for bond market clearing, and err_4 for product market clearing. Note that we do not include the bond and product market-clearing next period, as they are already covered by the current period counterparts.

We simulate our model for $10,000$ periods and calculate for each period the approximation error.[19] The results of the validation exercise for the four equilibrium conditions are shown in Figure 23. The residual errors are well contained. The symmetry and concentration of the distributions around small values further reinforce the reliability of the neural network solution, indicating that the approximation performs consistently well across a wide range of simulated states.

6.7 Comparison with Sequence Space Jacobian

We now compare again our neural network-based solution method to a conventional (linear) approach, specifically the sequence space Jacobian (SSJ) approach of Auclert et al. (2021).

[19] We draw $1,000$ shocks to evaluate expectations.

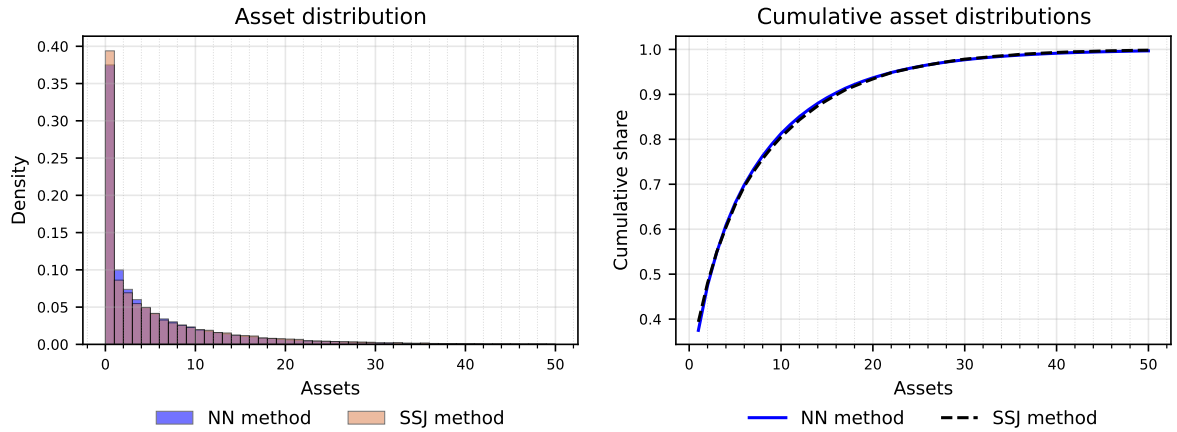

Figure 24: Asset distribution for our neural network (NN) method for comparison with sequence space Jacobian method. For the NN method, we evaluate the distribution at its stochastic steady state, averaging over 2,000 economies. For the SSJ, we use the deterministic steady state. Left panel: The histogram of asset holdings in equilibrium. Right panel: Cumulative distribution function for the asset holdings in equilibrium.

For this comparison, we use the conventional linearized SSJ solution.

Figure 24 compares the asset distribution generated by our method and by the SSJ approach. For our method, we report the distribution at the stochastic steady state, which accounts for the presence of aggregate risk. In contrast, the SSJ method by construction assumes away aggregate uncertainty, so the distribution it produces corresponds to the deterministic steady state. Despite this key difference, the two distributions are remarkably similar. Although minor deviations are visible in the tails, the overall shape and key moments align closely, suggesting that our method provides a highly accurate approximation. Note that, in the paper, we report the comparison using the deterministic steady state for both models. In this case, the resulting distributions are also remarkably similar, as expected.

Notably, these deviations cannot be attributed solely to inaccuracies in either our method or the SSJ approach. Instead, they stem from a fundamental difference in the economic environments captured by the two methods. Incorporating aggregate uncertainty, as our method does, leads poorer households to work more in response to increased precautionary motives, resulting in fewer households clustered at the bottom of the asset distribution, as reflected in the shorter blue bars in the lower asset bins. At the same time, this increased labor supply and precautionary saving among the poor depresses the equilibrium real interest rate, thereby reducing the incentive for wealthier households to save. As a result, rich households accumulate less wealth in equilibrium, leading to a thinner right tail in the asset distribution. This pattern is clearly illustrated by the crossing of the blue solid and black dashed lines in the right panel of Figure 24, which plots the cumulative asset distribution.

Even in this relatively stylized model, the differences between the asset distributions implied by the nonlinear (our method) and linearized (SSJ) solutions are already notable, despite being quite small. As we will show, these differences become even more pronounced if we increase aggregate risk.

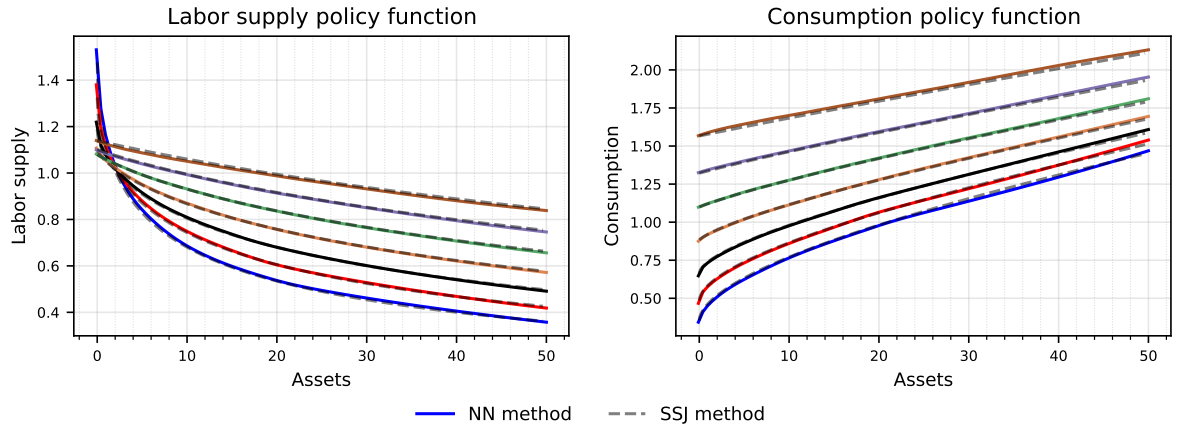

Figure 25: Individual policy functions comparison for the one-asset HANK model. The figure illustrates how the policy functions for labor and consumption vary for different asset levels and different productivity levels. We compare the neural network method with the SSJ method. We use the 7 productivity levels that the discretization approach of the SSJ provides for comparison. We evaluate the policy functions at the stochastic steady state (NN) and deterministic steady state (SSJ).

The next step is to compare the two sets of policy functions that determine agents' labor supply and consumption decisions, as shown in Figure 25. The solid lines depict the policy functions approximated using our method, while the dashed lines correspond to the approximation obtained using the SSJ method from Auclert et al. (2021). Each color represents one of the seven idiosyncratic labor productivity levels considered in Auclert et al. (2021). The brown lines correspond to the most productive agents (top productivity bin), while the blue lines represent the least productive ones (bottom productivity bin.)

Note that these seven productivity levels are used solely for the purpose of comparing the results from our solution method with those from SSJ. It is worth recalling that our solution method does not rely on discretizing the productivity space. Instead, we treat idiosyncratic labor productivity as the outcome of a continuous AR(1) process, as specified in the model. For the purpose of this comparison, we evaluate the policy functions (solved for a continuous AR(1) process) at the values in line with the discretization in the code made available by Auclert and coauthors.

The upshot of the comparison is twofold. First, our method successfully replicates the policy functions approximated by the SSJ approach. Second, the main discrepancy arises in the behavior of the most productive agents with zero assets—and low productive agents with a large amount of assets. Both groups represents a tiny fraction of agents. Because so few agents follow such a path, the neural network has limited data to learn from, which results in a less accurate approximation of their behavior.Nonetheless, the inaccuracies in these cases are fairly limited, highlighting the strong extrapolative properties of neural networks.

Indeed, the effects of a monetary policy shock on the equilibrium dynamics of aggregate variables predicted by the two methods are quite similar. See Figure 26.

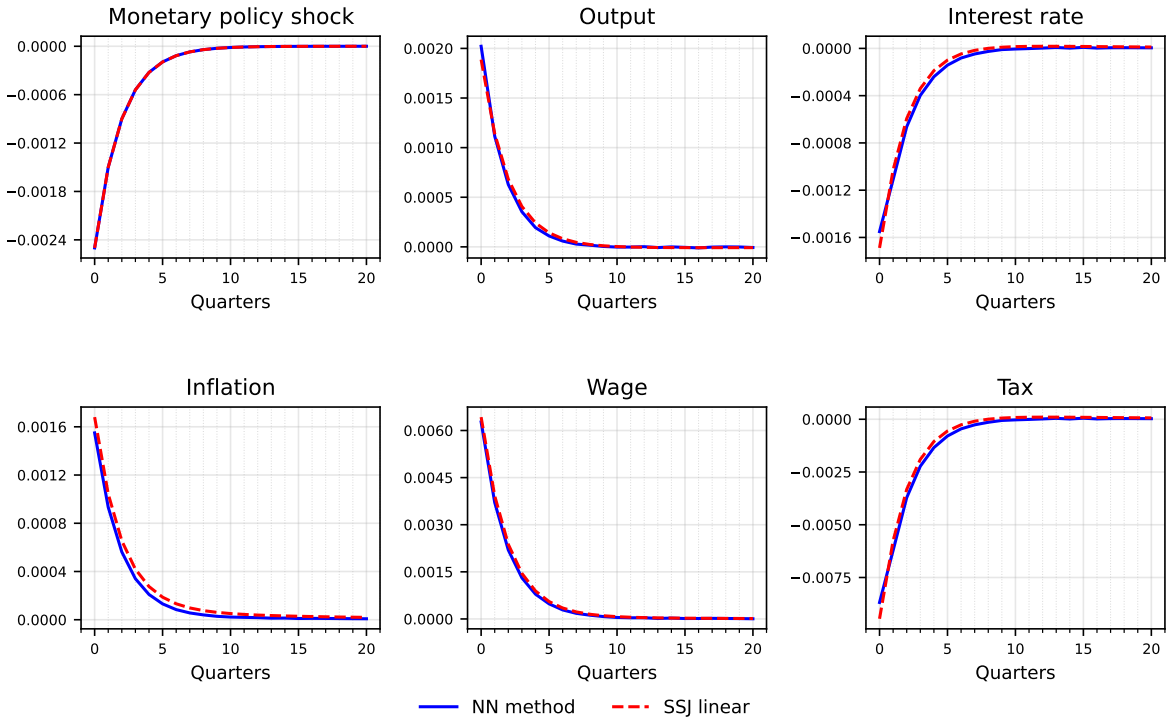

Figure 26: Impulse response functions comparison of neural network solution and sequence space Jacobian approach. The deviations from the stochastic steady state (for the NN method) and from the deterministic steady state (for the SSJ method) are reported for a one-standard-deviation negative monetary policy shock. L=100 and MC=100.

6.8 Number of Agents, Number of Draws for Monte Carlo Integration, Size of Shock Volatility

We are now testing how changing the number of agents and the number of draws for the Monte Carlo integration affects our results.

6.8.1 Number of Agents

Figure 27 shows the distribution of asset holdings across different levels of discretization, with the left panel showing the histogram and the right panel the corresponding cumulative distribution function. We compare the results for L=100, 200, 500, and 1,000 agents. Each distribution is obtained by averaging across 2,000 simulated economies (batches). The two figures illustrate that the benefit of increasing the number of tracked agents to capture heterogeneity is minimal, since the asset distribution changes only slightly.

Even more striking is the comparison based on the policy functions for labor and consumption. Figure 28 shows the bar graph of the asset distribution conditional on various levels of productivity (10th percentile, 50th percentile, and 90th percentile) along with the policy functions of labor and consumption for 100 agents (solid line) and 1,000 (dashed line).[20] The policy functions exhibit minimal change as the number of tracked agents increases from

[20]The other individual policy function determines the agent's asset demand, which is implied by the two individual policy functions and hence is not shown.

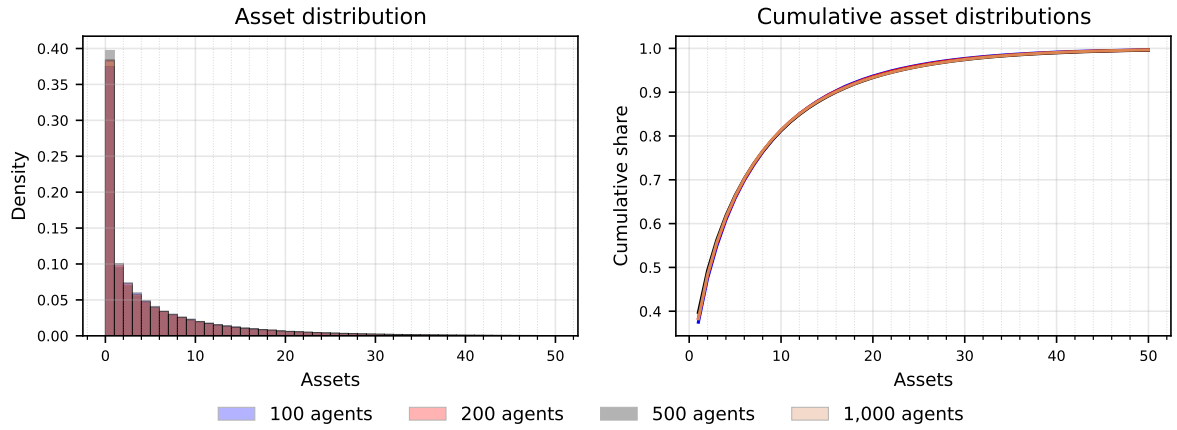

Figure 27: Asset distribution for variations in the number of agents L. The figure shows the histogram of asset holdings (left) and the corresponding cumulative distribution function (right) for different values of L used to solve the model with our NN methods. Specifically, we consider scenarios with 100, 200, 500, and 1,000 agents in equilibrium. The distribution is evaluated at the stochastic steady state, averaged over 2,000 simulated economies.

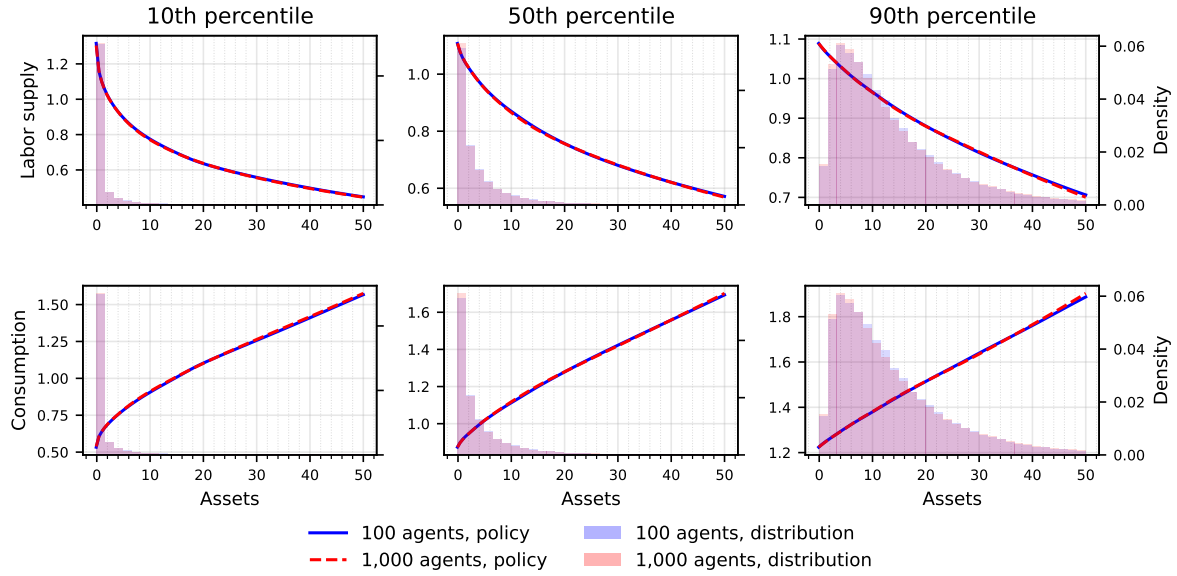

Figure 28: Policy functions for variations in number of agents L. Policy functions (labor supply - upper panels - and consumption - lower panels) and asset distribution conditional on individual productivity (10th percentile, 50th percentile, 90th percentile) for $L = 100$ and $L = 1,000$.

100 to 1,000, supporting our claim that tracking more than 100 agents yields little additional benefit in approximating the asset distribution in this one-asset HANK model.

Nevertheless, computing time increases rapidly as the number of agents tracked L grows, as we will now show. The runtime ranges from 1h31min for $L = 100$ to 6h12min for $L = 1,000$, as reported in Table 1.[21] Notably, even the version of the model with 1,000 agents required only 5.3 GB of memory (compared to just 0.6 GB for $L = 100$). This reflects the new memory-efficient design of our solution code, which makes it feasible to run the nonlinear one-asset

[21] These computations are performed on an Nvidia RTX 5090 GPU with 32 GB of memory.

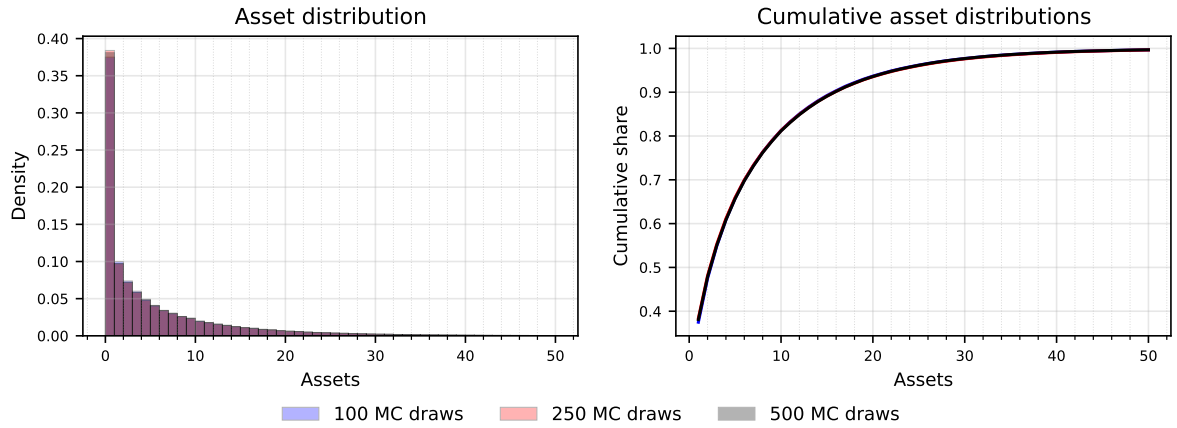

Figure 29: Asset distribution for variations in number of Monte Carlo draws MC. The figure shows the histogram of asset holdings (left) and the related cumulative density function (right) for an alternative number of Monte Carlo draws for the approximation of expectations. Specifically, we consider scenarios with 100, 250, and 500 Monte Carlo draws. We evaluate the distribution at its stochastic steady state, averaging over 2,000 economies.

HANK model on a desktop machine. For instance, solving the model with $L = 100$ took 30h38min on a recent MacBook.[22]

Number of Agents	GPU Computational Time	GPU Memory Usage (GB)
L=100	1h31min	0.6
L=200	2h05min	1.1
L=500	3h35min	2.7
L=1,000	6h12min	5.3

Table 1: Run time and memory usage for alternative number of agents. The calculation were computed with a Nvidia RTX 5090 with 32 GB of memory as GPU. Memory usage is the peak GPU memory allocated by the solver.

6.8.2 Number of Monte Carlo Draws

We adopt a Monte Carlo approach to approximate expectations. In general, the accuracy of this approximation depends on using a sufficiently large number of Monte Carlo draws for next-period shock realizations. Here, we show that this number does not need to be large when using our neural network approach; in fact, a relatively modest number of draws is sufficient to achieve accurate approximations. The underlying reasons are similar to those that explain why only a limited number of agents need to be tracked in our framework.

In the baseline case with 100 agents, we vary the number of Monte Carlo draws used to approximate next-period variables—testing with 100, 250, and 500 draws. Importantly, the number of agents is fixed at L=100 across these exercises. These results, as shown in Figures 29 and 30, are reported for the one-asset version of the Auclert et al. (2021) model. The

[22]This computation used an Apple M4 Pro chip with a 14-core CPU and a 20-core GPU. We employed the Metal programming framework for macOS devices. When run on the CPU, the same task took approximately twice as long.

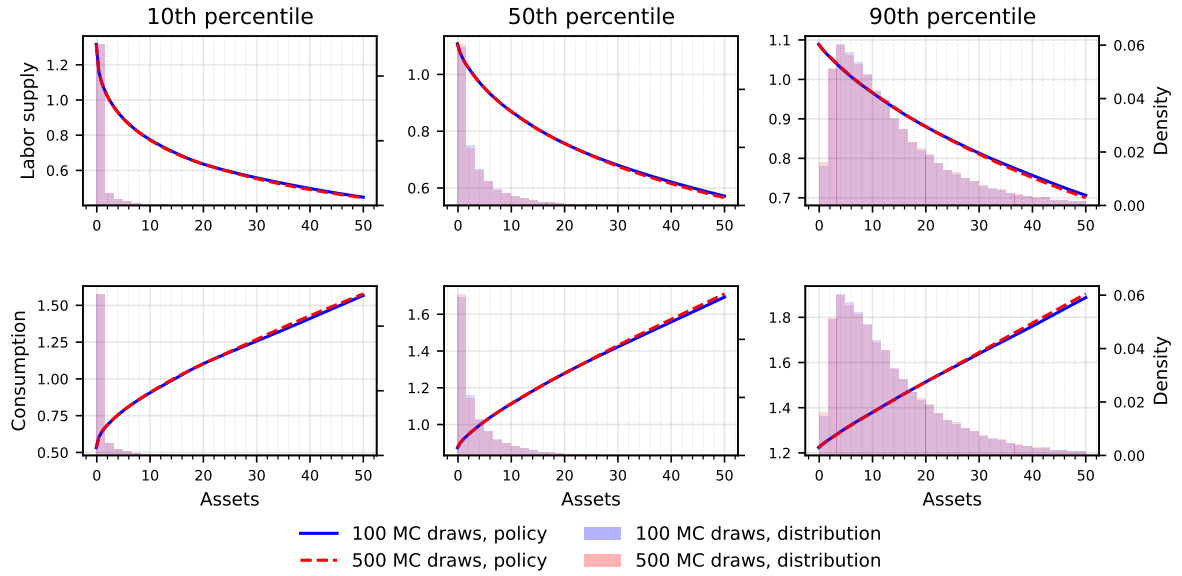

Figure 30: Policy functions for variations in number of Monte Carlo draws MC. Policy functions (labor supply - upper panels - and consumption - lower panels) and asset distribution conditional on individual productivity (10th percentile, 50th percentile, 90th percentile) for $MC = 100$ and $MC = 500$.

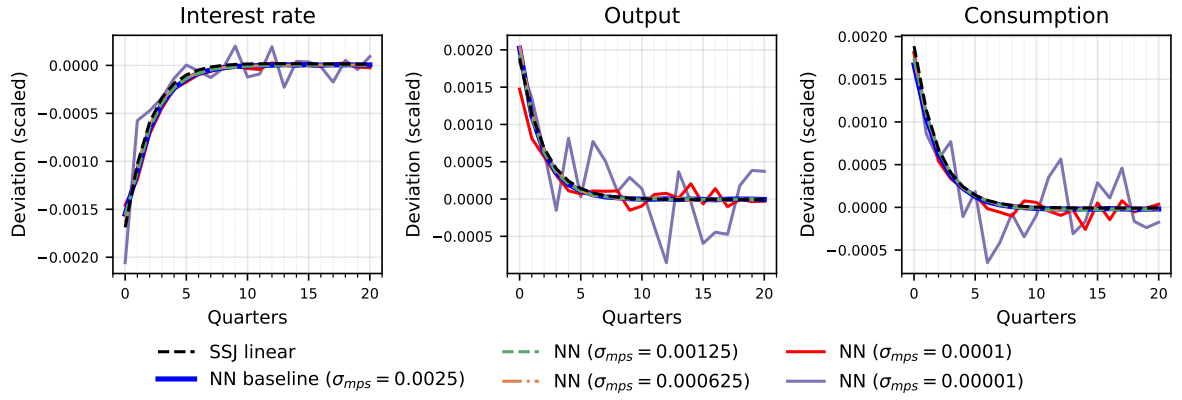

Figure 31: Monetary policy responses under low aggregate uncertainty in the one-asset HANK model. The figure compares the linearized SSJ impulse responses with those generated by neural networks trained under progressively lower volatilities of the monetary policy shock. To facilitate comparison, all neural-network responses are rescaled to correspond to the same shock size.

results show that 100 Monte Carlo draws approximate the expectations very good here. One reason that they perform so well is that we draw a new set of Monte Carlo draws during the training process for each iteration and batch, resulting in a very good approximation over the training process.

6.8.3 Size of Shock Volatility

The accuracy of the neural-network approximation can deteriorate when the volatility of aggregate shocks becomes extremely small. In this case, the variation induced by the aggregate shock may become difficult to distinguish from the numerical noise inherent in the training

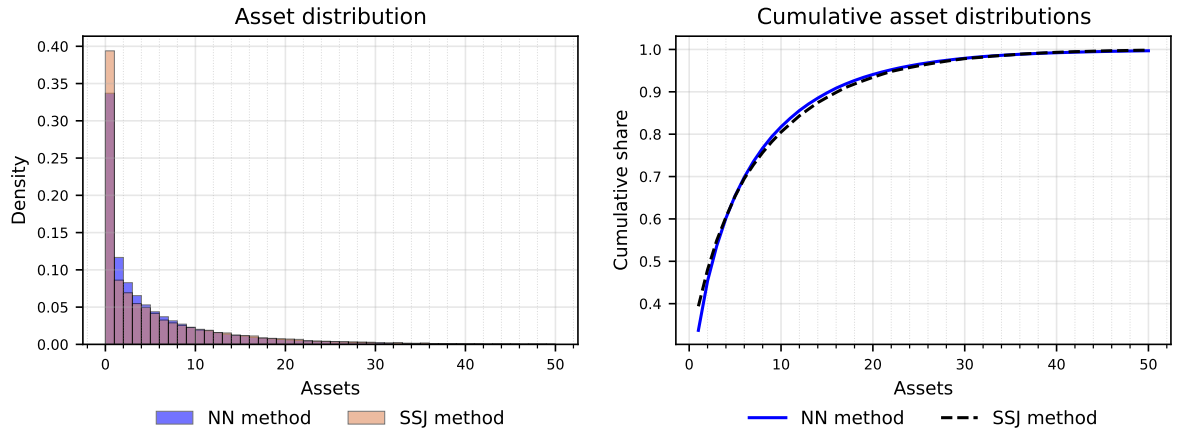

Figure 32: Asset distribution in an economy with large aggregate uncertainty. For the NN method, we evaluate the distribution at its stochastic steady state, averaging over 2,000 economies. For the SSJ, we use the deterministic steady state as it cannot account for aggregate uncertainty. Left panel: The histogram of asset holdings in equilibrium. Right panel: Cumulative density function for the asset holdings in equilibrium.

procedure.

Figure 31 illustrates the case of very low volatility of the monetary policy shock. It shows how the accuracy of the neural-network solution changes as we progressively reduce the volatility of the monetary policy shock.[23] In the benchmark specification, the standard deviation of this shock is 25 basis points. We then lower it to the values reported in the legend at the bottom of the figure. The neural-network solution remains stable when the standard deviation is reduced to 12.5 or 6.25 basis points. However, at implausibly low volatility levelsâ1 basis point or lessâthe neural network begins to fail to some extent capture the impulse responses to monetary policy shocks accurately.

The relevant issue is not the magnitude of the impulse responses themselves, which are rescaled to correspond to the same shock size, but the signal-to-noise ratio encountered during training. Our neural-network method relies on simulated data generated using a finite number of agents, Monte Carlo integration, and a finite sample of aggregate states. These ingredients inevitably introduce numerical noise into the training procedure. When aggregate volatility becomes sufficiently small, the variation generated by the aggregate shock becomes comparable to, or smaller than, this numerical noise. The neural network can then no longer reliably distinguish the aggregate signal from noise in the simulated environment.

6.9 The Role of Large Aggregate Uncertainty

The next step is to illustrate how, even in a HANK model with only one aggregate shock, an increase in aggregate risk leads to a substantial divergence between the solutions approximated with our code and the SSJ method. At higher levels of aggregate risk, precautionary saving behavior becomes more prominent, especially when intertwined with the model's nonlinear features.

[23] The monetary policy shock follows the AR(1) process $mp_t = \rho_m mp_{t-1} + \epsilon_t^m, \epsilon_t^m \sim \mathcal{N}(0, \sigma^2_{mps})$..

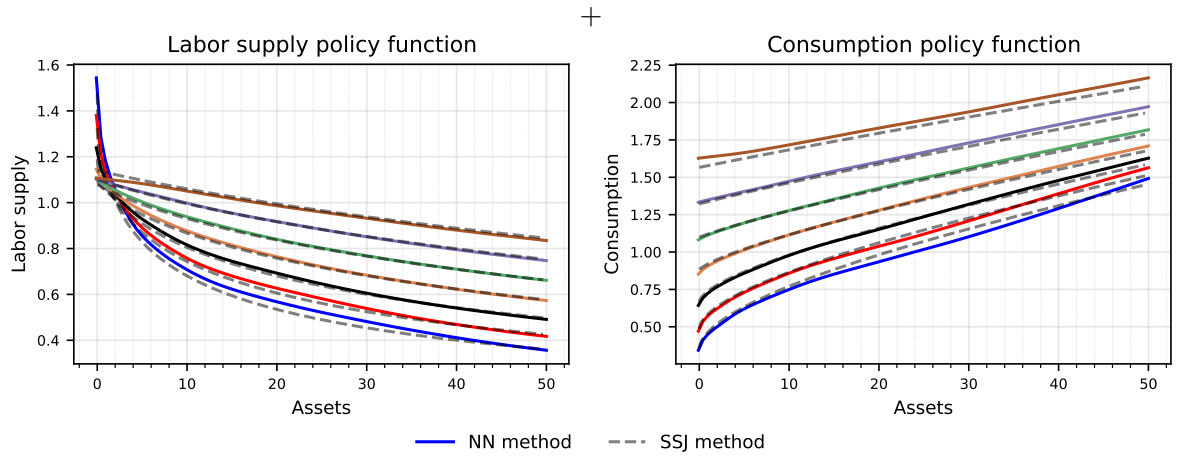

Figure 33: Individual policy functions in an economy with large aggregate uncertainty. The figure illustrates how the policy functions for labor and consumption vary for different assets levels and different productivity levels. We compare the neural network method with the SSJ method. We use the 7 productivity levels that the discretization approach of the SSJ provides for comparison. We evaluate the policy functions at the stochastic steady state (NN) and deterministic steady state (SSJ).

To illustrate the effect, we scale the size of the monetary policy shock by a factor of ten. With a larger standard deviation of monetary policy shocks, aggregate uncertainty increases and nonlinearities become more pronounced. This has substantial effects on the asset distribution, households' policy functions, and the aggregate transmission of shocks. As shown in Figure 32, the share of households at the borrowing limit is now significantly lower. As our method accounts for aggregate risk, we can now see a substantial difference compared to SSJ.

The underlying mechanism, which we already presented when we discussed Figure 32, is now amplified by the higher aggregate volatility: the precautionary savings motive leads households in low productivity states to work more and consume less, relative to a scenario with little or no aggregate risk, in order to accumulate buffer stocks of savings. Indeed, our NN solution method shows that there is now a much lower share of households at the borrowing limit because households have a larger precautionary motive due to larger aggregate risks. SSJ cannot capture these effects on the equilibrium wealth distribution.

As these households increase savings, the equilibrium interest rate falls. In response, high-productivity households benefit less from working, reduce their labor supply, and increase their consumption. These responses explain why the neural network solution now diverges more substantially from the SSJ solution, which does not account for uncertainty.

In Figure 33 the two policy functions of agents now diverge more markedly and for all levels of asset holdings. This is in large part explained by the fact that abstracting now from aggregate uncertainty is conseuqential.

Figure 34 shows the impulse response functions to a monetary policy shock under high aggregate volatility, approximated using our method (solid red line) and the SSJ method (dashed black line). The responses differ markedly both in the short run and in the long run.

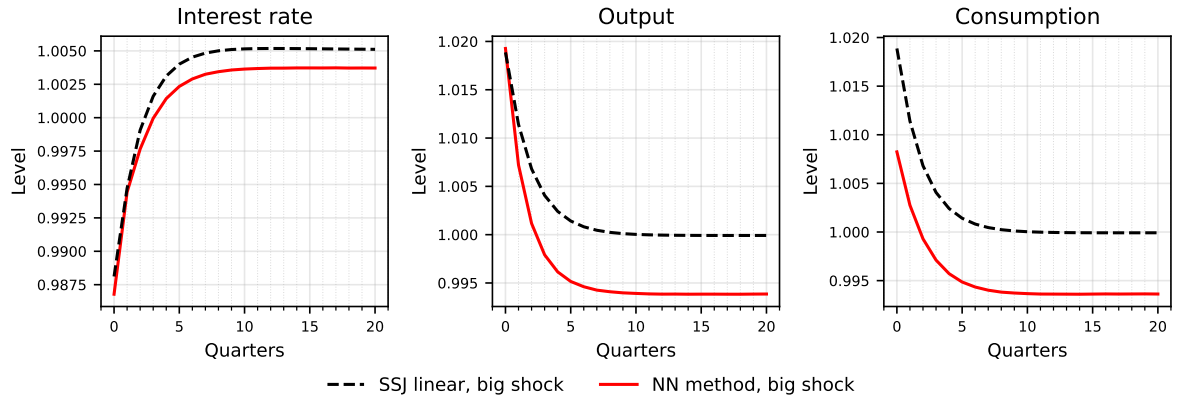

Figure 34: Impulse response functions in an economy with large aggregate uncertainty. The figure compares the responses generated by our neural-network solution (red solid) and the linearized SSJ solution (black dashed) following a large monetary policy shock.

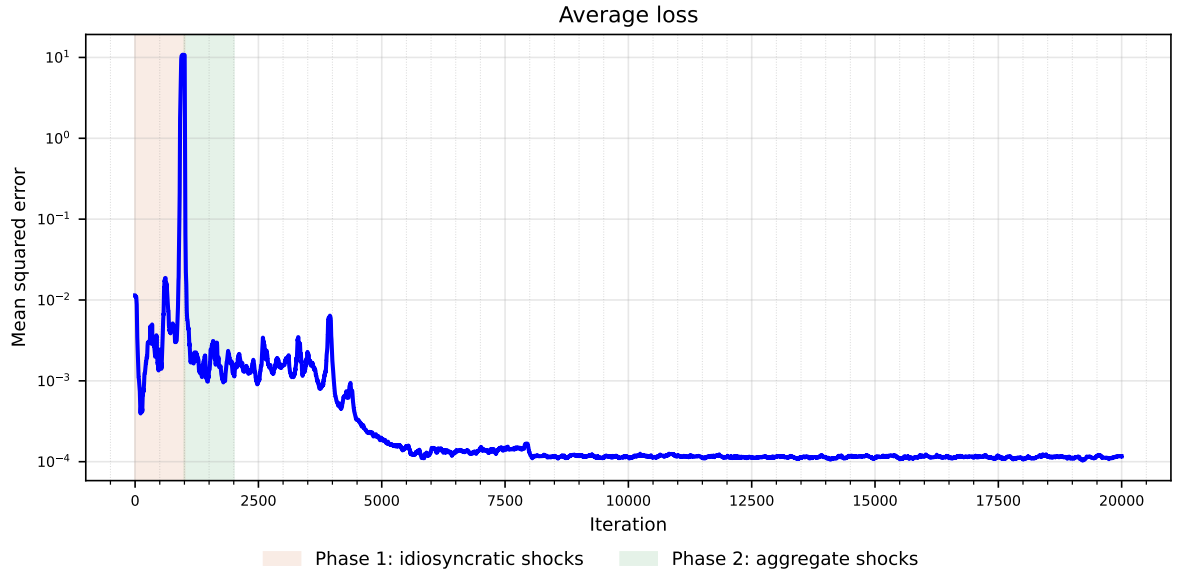

Figure 35: Convergence of the NN solution in an economy with large aggregate uncertainty in the one-asset HANK model. The figure shows the mean squared error during the joint training of the individual and aggregate policy functions for the one-asset HANK model with a large monetary policy shock ($\sigma_{mps} = 0.025$). The line reports the loss averaged across the equilibrium conditions that we minimize. The series is calculated as a rolling median over 100 iterations. The shaded areas indicate the periods during which the idiosyncratic and aggregate shocks are gradually introduced. The vertical axis is shown on a logarithmic scale.

The long-run differences can be attributed to the distinct stochastic (risky) steady states, which are captured by our method but not by the SSJ approach.

The differences reported in Figure 34 suggest that when aggregate risk is substantial –as in empirical models– existing methodologies that rely on linearizing aggregate dynamics may fail to accurately capture the impact of uncertainty on the long-run behavior of macro variables and the propagation of shocks. This misrepresentation introduces inaccuracies in the short-run and long-run dynamics of macro observables, which can distort parameter estimates and model's predictions.

Figure 35 reports the evolution of the training loss to provide further evidence that the neural-network solution is accurate and reliable. The line reports the loss averaged across the equilibrium conditions that we minimize. The series is calculated as a rolling median over 100 iterations.

We facilitate training by initializing the standard deviations of both idiosyncratic and aggregate shocks below their calibrated values and then gradually increasing them to those levels. The shaded areas indicate the phases during which idiosyncratic and aggregate shock volatility, respectively, are gradually scaled up. After the temporary increases associated with these phases, the average loss declines and stabilizes at approximately 10^{-4}.

6.10 Practical Advice in the Case of Limited Access to a GPU

A practical coding strategy leverages the portability of the new code architecture through a two-stage approach if there is only limited access to a GPU. During the development stage, the model can be designed and tested on a standard laptop (e.g., a MacBook, even without using a GPU) using a relatively small number of agents, which enables fast experimentation and debugging. Once the model's structure and performance are validated, the same code can be executed on a high-performance desktop or server equipped with a GPU. In this implementation stage, the greater computational power allows training neural networks with a much larger number of agents and additional features, thereby improving the precision of the solution and estimation results. This two-stage workflow is particularly useful in practice, as it combines the flexibility and convenience of local model development with the computational efficiency of large-scale implementation, greatly streamlining the research process.

7 Conclusion

This learning guide offers a hands-on introduction to implementing and adapting our NN-based solution and estimation framework for nonlinear heterogeneous agent models. The learning guide introduces our methods using three models with varying complexity: 1) Representative Agent New Keynesian (RANK) model, 2) Heterogeneous Agent model based on Krusell and Smith (1998), and 3) Heterogeneous Agent New Keynesian (HANK) model. By progressing through a series of structured extensions, users can gradually deepen their understanding of both economic modeling and the computational methods involved.

References

Aruoba, S.B., Fernández-Villaverde, J., Rubio-Ramirez, J.F., 2006. Comparing solution methods for dynamic equilibrium economies. Journal of Economic dynamics and Control 30, 2477–2508.

Auclert, A., Bardóczy, B., Rognlie, M., Straub, L., 2021. Using the sequence-space jacobian to solve and estimate heterogeneous-agent models. Econometrica 89, 2375–2408.

Barron, J.T., 2017. Continuously differentiable exponential linear units. arXiv preprint arXiv:1704.07483 .

Judd, K.L., 1992. Projection methods for solving aggregate growth models. Journal of Economic theory 58, 410–452.

Kase, H., Melosi, L., Rottner, M., 2025. Estimating nonlinear heterogeneous agent models with neural networks. mimeo.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Krusell, P., Smith, A., 1998. Income and wealth heterogeneity in the macroeconomy. Journal of Political Economy 106, 867–896.

Loshchilov, I., Hutter, F., 2017. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 .