Deep Learning Solutions to Master Equations for Continuous Time Heterogeneous Agent Macroeconomic Models

Zhouzhou Gu[*], Mathieu Laurière[†], Sebastian Merkel[‡], Jonathan Payne[§¶]

August 24, 2023

Abstract

We propose a new global solution algorithm for continuous time heterogeneous agent economies with aggregate shocks. First, we approximate the state space so that equilibrium in the economy can be characterized by one high, but finite, dimensional partial differential equation. Second, we approximate the value function using neural networks and solve the differential equation using deep learning tools. We refer to the solution as an Economic Model Informed Neural Network (EMINN). The main advantage of this technique is that it allows us to find global solutions to high dimensional, non-linear problems. We demonstrate our algorithm by solving two canonical models in the macroeconomics literature: the Aiyagari (1994) model and the Krusell and Smith (1998) model.

Keywords: Heterogeneous agents, computational methods, deep learning, inequality, mean field games, continuous time methods, aggregate shocks, global solution.

[*]Princeton, Department of Economics. Email: zg3990@princeton.edu

[†]NYU Shanghai, NYU-ECNU Institute of Mathematical Sciences. Email: mathieu.lauriere@nyu.edu

[‡]University of Exeter, Department of Economics. Email: s.merkel@exeter.ac.uk

[§]Princeton, Department of Economics. Email: jepayne@princeton.edu

[¶]We would like to thank Adam Rebei for outstanding research assistance throughout this project. We are also very grateful to the comments and discussion from Goutham Gopalakrishna and Yucheng Yang.

1 Introduction

There is great interest in understanding how inequality changes over time and reacts to government policy. This has led researchers to study macroeconomic models with complicated distributions of agents. A major difficulty with working on these models is that the agent distribution becomes a state variable and so the state space becomes infinite dimensional. In this paper, we demonstrate how deep learning techniques can relax the "curse of dimensionality" for continuous time heterogeneous agent models and allow global numerical solutions to be computed.

We develop solution techniques for a class of dynamic, stochastic, general equilibrium economic models with the following features. There is a large collection of price-taking agents who face uninsurable idiosyncratic and aggregate shocks. Given their belief about the evolution of aggregate state variables, agents choose control processes to solve dynamic optimization problems. When making their decisions, agents face financial frictions that constrain their behaviour and potentially break "aggregation" results that would allow the distribution of agents to be replaced by a "representative agent". We solve for a rational expectations equilibrium in which agent beliefs about the evolution of aggregate states are consistent with the dynamics that emerge in the economy. Solving for equilibrium reduces to solving a "master" partial differential equation (PDE) that summarizes both the agent optimization behaviour (from the Hamilton-Jacobi-Bellman equation) and the evolution of the distribution (from the Kolmogorov forward equation). A canonical example of this type of environment is the Krusell and Smith (1998) model, which is often used as a workhorse environment for testing solution methods in macroeconomics.

Our solution approach approximates the infinite dimensional master equation by a finite, but high, dimensional PDE and then uses deep learning to solve the high dimensional equation. We consider two different approaches for reducing the dimension of the master equation. The first approach approximates the distribution by a large, finite number of agents. We refer to this as the "finite-agent" approximation. The second approach approximates the distribution by discretizing the agent state space so the density becomes a collection of mass points at grid points. We call this the "finite-state" approximation.

We solve the finite dimensional approximation to the master equation using recent advances in deep learning. In particular, we adapt the Deep Galerkin Method (DGM) developed by the applied mathematics and analogous to the Physics Informed Neural Networks (PINNs) developed in the physics literature. This approach approximates the value function by a neural network and then uses stochastic gradient descent to train the neural network to minimise a loss function that combines the average error in master equation with the average error in the boundary conditions. We calculate average errors by randomly sampling over points in the state space and the boundary, with greater sampling applied to the regions with more curvature. We also need to handle inequality boundary conditions arising from financial constraints on the evolution of the state space. We do this by introducing penalty functions for when financial constraints bind. We refer to our approach as training Economic Model Informed Neural Networks (EMINNs).

We illustrate our technique by solving two canonical models in the macroeconomics literature that are frequently used for testing numerical techniques: Aiyagari (1994) and Krusell and Smith (1998). For the Aiyagari (1994) model, we show that we can match the finite difference solution to high accuracy and solve for transition dynamics without a shooting algorithm. For the Krusell and Smith (1998) model, we show that we generate solutions with a low error for the master equation. Traditional techniques only give approximate solutions to the Krusell and Smith (1998) model so we do not have an ideal benchmark for testing the model. However, we show that we get very similar results to contemporary approaches such as Fernández-Villaverde et al. (2018).

Literature Review: The economics literature has traditionally used three main approaches for solving heterogeneous agent models with aggregate shocks. One approach is to fit a statistical approximation to the law of motion for the key aggregate state variables (e.g. Krusell and Smith (1998), Den Haan (1997), Fernández-Villaverde et al. (2018)). As has been extensively discussed in the literature, this approach works well when the law of motion for the key state variables can be efficiently approximated as a function of key moments of the distribution (and so the economy is very close to permitting "aggregation"). By contrast, our approach can handle economies without near-aggregation results. A second approach is to take a type of linear perturbation in the aggregate state and then solve the resulting linear problem with matrix algebra (e.g. Reiter (2002), Reiter (2008), Reiter (2010), Winberry (2018), Ahn et al. (2018), Auclert et al. (2021), Bilal (2021), Bhandari et al. (2023)). By contrast, we solve the model globally and so can handle partial differential equations with extensive non-linearity. A final approach is to take a low dimensional projection of the distribution (e.g. Prohl (2017), Schaab (2020)). Our approach is complementary to these papers in that it allows for more general, higher dimensional projections through the use of deep neural networks.

Our paper is part of a growing computational economics literature using deep learning techniques to solve economic models and overcome the limitations of the traditional solution techniques. Many of these papers focus on solving heterogeneous agent macroeconomic models in discrete time (e.g. Azinovic et al. (2022), Han et al. (2021), Maliar et al. (2021), Kahou et al. (2021), Bretscher et al. (2022)) or using a discrete time approximation to a system forward and backward differential stochastic equations (e.g. Han et al. (2018), Huang (2022)). Our work is part of a less developed literature attempting to deploy deep learning techniques to solve the differential equations that arise in continuous time economic models (e.g. Duarte (2018), Gopalakrishna (2021), Fernandez-Villaverde et al. (2020), Sauzet (2021)). Our contribution to the continuous time literature is to understand how to handle a rich distribution of agents and directly solve the "master equation" for the economic system. This type of PDE has been introduced by Lions (2011) to describe the value function of a representative player in a mean field population of players at equilibrium. This necessitates resolving a collection of problems that are particular to the analytic characterization of continuous time problems, such as: approximating inequality boundary conditions, using the Kolmogorov Forward Equation to derive laws of motion for projection coefficients, and approximating derivatives with respect to the distribution.

There is also a growing mathematics literature that attempts to use neural networks to solve differential equations. Technically, our approach builds on the Deep Galerkin Method (DGM) and Physics Informed Neural Networks (PINNs) developed in Sirignano and Spiliopoulos (2018) and Raissi et al. (2017); see also Li et al. (2022). These papers train PINNs to solve systems of differential equations that arise in physics and finance for instance. We train Economic Model Informed Neural Networks (EMINNs) to solves systems of differential equations that arise in economics. A key difference is that economic models have forward looking optimizing agents and market clearing conditions. Our approach to handling borrowing constraints draws on Lu et al. (2021b) and Brzoza-Brzezina et al. (2015). We also build on the literature using neural networks to solve mean field games. Al-Aradi et al. (2022); Carmona and Laurière (2021) adapted the DGM to solve the PDE system arising in mean field games. Fouque and Zhang (2020); Carmona and Laurière (2022a); Germain et al. (2022b) proposed deep learning methods for mean field games and mean field control problems based either on direct approximation of the control or on an adaptation of the Deep BSDE method proposed by Han et al. (2018). We refer to e.g. Laurière (2021); Carmona and Laurière (2022b); Hu and Lauriere (2022) and the references therein for more details. However, these works are mostly focused on solving the problem at equilibrium and without aggregate shocks, so the equilibrium optimal control is learnt only for one distribution – the equilibrium distribution (or flow of distributions for non-stationary problems). Min and Hu (2021) proposed a deep learning method based on recurrent neural networks and signatures to solve mean field games with aggregate shocks, when the interactions are through moments and without solving the master equation. Perrin et al. (2022) introduced a deep reinforcement learning algorithm based on fictitious play to learn population-dependent policies for finite-state, finite-action mean field games. In the context of mean field control, Carmona et al. (2019); Gu et al. (2021); Germain et al. (2022a); Frikha et al. (2023) used deep learning to compute social optima with controls depending on the population distribution, but these methods do not solve Nash equilibria. Relative the mathematics literature, our focus is on solving master equations for a class of mean field games with aggregate shocks.

This document is organised as follows. Section 2 describes the general economic environment that we will be studying and derives the master equation. Section 3 describes the different approaches to solving the model. Section 4 applies our algorithm to the continuous time version of the Aiyagari (1994) described in Achdou et al. (2022). Section 5 applies our algorithm to continuous time version of Krusell and Smith (1998). Section 6 dicusses practical lessons. Section 7 concludes.

2 Economic Model

In this section, we outline the class of economic models for which our techniques are appropriate. At the high level, in economics terminology, we are solving continuous time, general equilibrium models with a distribution of optimizing agents who face idiosyncratic and aggregate shocks.[1] In mathematics terminology, we are solving mean field games with common noise.

2.1 Environment

Setting: The model is in continuous time with infinite horizon. There is an exogenous one-dimensional aggregate state variable, z_t, which evolves according to:

$$dz_t = \mu^z(z_t)dt + \sigma^z(z_t)dB_t^0, \quad z_0 \text{ given}, \tag{2.1}$$

where B_t^0 denotes a common Brownian motion process. We let $\mathcal{F}_t^0$ denote the filtration generated by B_t^0.

Agent Problem: The economy is populated by a continuum of agents, indexed by $i \in I = [0, 1]$. Each agent i has an idiosyncratic state vector, $x^i \in \mathcal{X}$, that evolves according to:

$$dx_t^i = \mu^x(c_t^i, x_t^i, z_t, q_t)dt + \sigma^x(x_t^i, z_t, q_t)dB_t^i + \gamma^x(x_t^i, z_t, q_t)dJ_t^i, \quad x_0^i \text{ given}, \tag{2.2}$$

where c_t^i is a one-dimensional control variable chosen by the agent, $q_t \in \mathcal{Q}$ is a collection of aggregate prices in the economy that will be determined endogenously in equilibrium, B_t^i denotes an N-dimensional idiosyncratic Brownian motion process, and J_t^i denotes an idiosyncratic Poisson jump process.[2] We let $\lambda(x)$ denote the rate at which Poisson jump shocks arrive given idiosyncratic state x.

Each agent i has a belief about the stochastic price process $\tilde{q} = \{\tilde{q}_t : t \geq 0\}$ adapted to $\mathcal{F}_t^0$. Given their belief, agent i chooses their control process, $c^i = \{c_t^i : t \geq 0\}$, to solve:

$$\begin{aligned} V(x_0^i, z_0) = \max_{c^i \in \mathcal{C}} \mathbb{E}_0 \left[\int_0^\infty e^{-\rho t} u(c_t^i) dt \right] \\ s.t. \quad (2.1),\ (2.2), \end{aligned} \tag{2.3}$$

where $\rho > 0$ is a discount parameter, $u(c_t^i)$ is the flow utility the agent gets and $\mathcal{C} = \{c_t^i \in \mathcal{C}(x_t^i, z_t, q_t) : t \geq 0\}$ is the set of admissible controls, where $\mathcal{C}(x, z, q)$ denotes the set of

[1]For ease of exposition, we restrict attention in the main text to models with 1-dimensional aggregate shocks. We set the master equation for a more general class of models in Appendix A.1.

[2]We do not consider the case where dB_t^0 directly impacts the evolution of idiosyncratic states. In principle, the techniques outlined in this paper still apply in this case but such cases typically involve long term asset pricing that leads to more complicated market clearing conditions. We leave the detailed discussion of such cases for future work.

possible actions for a player whose current state is x, when the aggregate state is z and the prices are q. We assume that u is increasing and concave. A classic example in economics is that the control must keep x_t^i positive. We let $\mathcal{X}$ denote the domain of x_t^i implicitly defined by the constraint that $c^i \in \mathcal{C}$.

Distributions: Market incompleteness means that idiosyncratic shocks potentially generate a non-degenerate cross sectional distribution of agent states. We let $G_t = \mathcal{L}(x_t^i|\mathcal{F}_t^0)$ and g_t denote the population distribution and density across x_t^i at time t, for a given history $\mathcal{F}_t^0$. Note that g_t should in principle depend on i, but by symmetry and homogeneity of the problem, all the agents are expected to use the same control at equilibrium and so it is sufficient to consider a single distribution flow.

Markets and distributions: We assume that the economy contains a collection of markets with finite dimensional price vector q_t and market clearing conditions allow us to solve for q_t explicitly in terms of z_t and g_t[3]:

$$q_t = Q(z_t, g_t), \quad \forall t \geq 0, \tag{2.4}$$

With an abuse of notation, we will write the constraint set as $\mathcal{C}(x, z, g)$ instead of $\mathcal{C}(x, z, Q(z, g))$. We assume that markets are incomplete in the sense that agents cannot trade claims directly on their idiosyncratic shocks dB_t^i and dJ_t^i.

Equilibrium: Given an initial density g_0, an equilibrium for this economy consists of a collection of $\mathcal{F}_t^0$-adapted stochastic processes, $\{c_t^i, g_t, q_t, z_t : t \geq 0, i \in I\}$, that satisfy the following conditions: (i) each agent's control process c^i solves problem (2.3) given their belief that the price process is $\tilde{q}$, (ii) the equilibrium prices q satisfy market clearing condition (2.4), and (iii) agent beliefs about the price process are consistent with the optimal behaviour of other agents in the sense that $\tilde{q} = q$.

2.2 Recursive Representation of Equilibrium

We assume that there exists an equilibrium that is recursive in the aggregate state variables: $\{z, g\}$. Observe that the price vector q can be expressed explicitly in terms of $\{z, g\}$ so beliefs about the price process can be characterized by beliefs about the evolution of the distribution and the aggregate state variable.

Hamilton Jacobi Bellman Equation (HJBE): Given a belief about the evolution of the dis-

[3]We will focus on numerical examples where $q_t = Q(z_t, \bar{g}_t)$, where $\bar{g}_t$ is the mean of g but this is not a constraint on the algorithm.

tribution, $dg_t(x) = \tilde{\mu}^g(z_t, g_t)dt$, the agent's optimal choice of control c solves the HJBE:

$$\begin{aligned}0 = \max_{c \in \mathcal{C}(x,z,g)} \Big\{ & -\rho V(x,z,g) + u(c) + D_x V(x,z,g)\mu^x(c,x,z,Q(z,g)) \\ & + \frac{1}{2}\operatorname{tr}\left\{\Sigma^x(x,z,Q(z,g))D_x^2 V(x,z,g)\right\} + \lambda(x)\left(V(x+\gamma^x(x,z,q),z,g) - V(x,z,g)\right) \\ & + \partial_z V(x,z,g)\mu^z(z) + \frac{1}{2}\left(\sigma^z(z)\right)^2 \partial_{zz} V(x,z,g) \\ & + \int_{\mathcal{X}} \tilde{\mu}^g(z,g)\frac{\partial V}{\partial g}(x,z,g)(y)dy\Big\} \end{aligned} \tag{2.5}$$

where $V(x,z,g)$ is the value function of the household, $\partial V/\partial g$ is the Frechet derivative of V with respect to the distribution, and $\Sigma^x(\cdot) := \sigma^x(\cdot)\left(\sigma^x(\cdot)\right)^\top$. The intuition behind this HJBE is that, although the agent can only influence x, the state is (x,z,g) and hence the value function should take these variables as inputs. From V, the optimal consumption, denoted by c^*, can be computed for every (x,z,g), which allows a representative player to react optimally to any population distribution. We focus on models where the constraint $c \in \mathcal{C}(x,z,g)$ only binds on the boundary so, for x in the interior of $\mathcal{X}$, the optimal control, c^*, is characterised by the first order condition:

$$0 = u'(c^*(x,z,g)) + D_x V(x,z,g)\partial_c \mu^x(c^*(x,z,g),x,z,Q(z,g))$$

and the HJBE is subject to a collection of boundary conditions denoted by

$$\Phi(x,z,g,V(x,z,g),D_x V(x,z,g)) \geq 0, \qquad x \in \partial\mathcal{X}.$$

See the applications in Sections 4 and 5 for examples of Φ.

Kolmogorov Forward Equation (KFE): Denote the recursive equilibrium optimal control of the individual agents by $c^*(x,z,g;\tilde{\mu}^g)$. Compared with the above notation, we add $\tilde{\mu}^g$ to stress the fact that the belief of the agent may differ from the true μ^g. Then, for a given z path, the evolution of the distribution under the optimal control can be characterized by the Kolmogorov Forward Equation (KFE):[4]

$$\begin{aligned} dg_t(x) = {} & \mu^g(c^*(x_t,z_t,g_t;\tilde{\mu}^g),x_t,z_t,g_t)dt, \quad \text{where} \qquad (2.6) \\ \mu^g(c_t^*,x_t,z_t,g_t) := {} & -\partial_x[\mu^x(c_t^*(x_t,z_t,g_t;\tilde{\mu}^g),x_t,z_t,q(x_t,g_t))g_t(x)] + \frac{1}{2}\partial_{xx}[(\sigma^x(z_t))^2 g_t(x)] \\ & + \lambda((1-\gamma'(x_t,z_t,q_t)g_t(x_t-\gamma(x_t,z_t,q_t)) - g_t(x_t)) \end{aligned}$$

Under this recursive characterization, the belief consistency condition becomes that $\mu^g = \tilde{\mu}^g$.

[4]Observe that there is no noise in the KFE because dB_t^0 does not directly impact the evolution of idiosyncratic states.

Master Equation: We follow the approach of Lions (2011) and characterize the equilibrium in one PDE, which is often referred to as the "master equation" of the "mean field game". This formulation is particularly convenient when the evolution of the economy is subject to aggregate shocks and the evolution of the aggregate state variables cannot be determined deterministically. Conceptually, the master equation is related to the HJBE (2.5) but it imposes the belief consistency by putting the equilibrium KFE into the HJBE. In equilibrium, the value function $V(x, z, g)$ is the solution to the following "master equation":

$$(\mathcal{L}V)(x, z, g) = 0 \tag{2.7}$$

where the operator $(\mathcal{L}V)(x, z, g) := (\mathcal{L}^h V + \mathcal{L}^g V)(x, z, g)$ is defined by:

$$\begin{aligned}
(\mathcal{L}^h V)(x, z, g) := & -\rho V(x, z, g) + u(c^*(x, z, g)) + D_x V(x, z, g)\mu^x(c^*(x, z, g), x, z, Q(z, g)) \\
& + \frac{1}{2}\mathrm{tr}\left\{\Sigma^x(x, z, Q(z, g)) D_x^2 V(x, z, g)\right\} \\
& + \lambda(x)\left(V(x + \gamma^x(x, z, q), z, g) - V(x, z, g)\right) \\
& + \partial_z V(x, z, g)\mu^z(z) + \frac{1}{2}\left(\sigma^z(z)\right)^2 \partial_{zz} V(x, z, g) \\
(\mathcal{L}^g V)(x, z, g) := & \int_{\mathcal{X}} \mu^g(c^*(x, z, g), z, g)\frac{\partial V}{\partial g}(x, z, g)(y)dy.
\end{aligned}$$

In this notation, the $\mathcal{L}^h$ operator reflects the optimization problem of the household, the $\mathcal{L}^g$ operator reflects how the evolution of the distribution affects the household value, and the master equation is subject to a problem-specific boundary condition related to the constraint $c \in \mathcal{C}(x, z, g)$.[5]

Intuitively, $V(x, z, g)$ can be interpreted as the optimal value of a representative player who starts at state x, with aggregate shock equal to z, and who faces a population that starts at the distribution g and then plays according to the Nash equilibrium control c. We refer to Cardaliaguet et al. (2015); Bensoussan et al. (2015) for more details. The goal of this paper is use deep learning techniques to find numerical solutions to equation (2.7). The challenge is that the master equation contains an infinite dimensional variable, g, and a derivative with respect to this variable. We thus need to work with numerical approximations of the distribution and solution methods that can handle high dimensions.

2.3 Model Generality

Our model set up nests many canonical models in macroeconomics such as Krusell and Smith (1998). However, we have also made a number of strong assumptions, which we discuss below:

(i). We assumed that the prices, q, can be expressed explicitly in closed form as functions

[5]We define the gradient $D_x V(\cdots)$ as a row vector, such that the product $D_x V(\cdots)\mu^x(\cdots)$ has to be interpreted as an inner product for multi-dimensional x.

of the aggregate state variables (z, g), see (2.4). In models with complicated market clearing conditions we need to solve an auxiliary fixed point problem in order to solve for the recursive representation of the price.

(ii). We assumed that the aggregate Brownian motion B^0 does not directly shock the evolution of idiosyncratic states. Instead it changes prices and so indirectly changes agent controls. This means that the KFE (2.6) does not have an aggregate noise term dB_t^0. The solution approach can easily be generalized to handle KFEs with aggregate noise so long as it does not lead to significantly more complicated market clearing conditions (a common extension studied is Fernández-Villaverde et al. (2018)).

3 Solution Approach

In this section, we outline a generic approach for applying a "Deep Galerkin" approach to solving the master equation (2.7). The first part of this approach is to find a finite dimensional approximation to the distribution so we can develop a finite, but high, dimensional approximation to the master equation. The second part is to approximate the solution to the finite dimensional master equation using a neural network. Finally, we use "deep learning" tools to solve the approximate master equation.

3.1 Finite Dimensional Master Equation

In order to proceed, we need to find a finite dimensional approximation to the density, which we denote by $\hat{g}$, and the associated finite dimensional master equation operator, $\hat{\mathcal{L}}$. In this section, we outline two possible approximation approaches. The first approach approximates the distribution with a finite collection of agents. The second approach projects the distribution onto a collection of mass points on a discretized grid. The difference between the approaches primarily appears in how the operator $\mathcal{L}^g$ is approximated.

3.1.1 Finite Agent Approximation

In this approach, we restrict the model so that the economy contains a large but finite number of agents $I < \infty$ agents. In this case, the density, g_t, is replaced by the individual states of the I agents, which we denote by $\hat{g}_t$:

$$\hat{g}_t := \{x_t^i : i \leq I\}.$$

With an abuse of notation, we identify $\hat{g}_t$ with the empirical distribution of its points, i.e., $\hat{g}_t := \frac{1}{I}\sum_{i=1}^{I} \delta_{x_t^i}$ where $\delta_{x_t^i}$ denotes a Dirac mass at x_t^i.

The market clearing condition now becomes (with some abuse of notation) $q_t = Q(z_t, \hat{g}_t)$, where the distribution in the clearing condition is approximated by the empirical distribution generated by the agent positions. However, to maintain the price taking assumption in the finite agent model, we impose that agent i behaves as if their individual actions do not influence prices. Formally, this means that agent i perceives the pricing function to be:

$$q_t = Q(z_t, \hat{g}_t^{-i})$$

where $\hat{g}_t^{-i} = \{x_t^j \in I^{-i}\}$ is the position of the other agents $I^{-i} := \{j \leq I : j \neq i\}$. Ultimately, this will ensure that the neural network trains the policy rules as if the agents believe that their assets do not influence the market prices. Aside from this change to the belief process, the optimization problem for household remains the same as in section 2.

Let $c^*(x^i, z, \hat{g})$ denote the equilibrium optimal control. Let $V(x^i, z, \hat{g})$ denote the value function for the master equation in the economy with I price taking agents. Then $V(x^i, z, \hat{g})$ solves $(\hat{\mathcal{L}}V)(x^i, z, \hat{g}) = 0$ subject to the boundary conditions, where the (approximate) master equation operator $\hat{\mathcal{L}}$ is given by:

$$\begin{aligned}
\hat{\mathcal{L}}V &= \hat{\mathcal{L}}^h V + \hat{\mathcal{L}}^g V, \quad \text{where} \\
(\hat{\mathcal{L}}^h V)(x^i, z, \hat{g}) &:= (\mathcal{L}^h V)(x^i, z, \hat{g}) \\
(\hat{\mathcal{L}}^g V)(x^i, z, \hat{g}) &:= \sum_{j \neq i} \frac{\partial V}{\partial \hat{g}^j}(x^i, z, \hat{g}) \mu^x(c^*(x^j, z, \hat{g}), x^j, z, Q(z, \hat{g}^{-j})) \\
&+ \sum_{j \neq i} \frac{1}{2} \operatorname{tr}\left\{\Sigma^x(x^j, z, Q(z, \hat{g}^{-j})) D^2_{g^j} V(x^i, z, \hat{g})\right\} \\
&+ \sum_{j \neq i} \lambda(x^j)\left(V(x^i, z, \{x^j + \gamma^x(x^j, z, Q(z, \hat{g}^{-j})), \hat{g}^{-j}\}) - V(x^i, z, \hat{g}^{-i})\right),
\end{aligned}$$

The operator for the household optimization problem, $\hat{\mathcal{L}}^h$, is the same as in the general problem but the market clearing conditions are calculated with the true distribution replaced by the empirical distribution for the finite collection of agents. The operator for the impact of distributional changes on the household, $\hat{\mathcal{L}}^g$, becomes finite dimensional because the economy only needs to track the evolution of a finite number of agents. The notation $\frac{\partial V}{\partial \hat{g}^j}(x^i, z, \hat{g})$ means that we take the partial derivative of V with respect to the j-th point in $\hat{g}$, i.e., x_t^j.

The solution $V(x^i, z, \hat{g})$ is expected to converge to the solution of the master equation (2.7) as the number of agents, I, grows to infinity. Intuitively, this relies on propagation of chaos result for McKean-Vlasov dynamics, see e.g. Sznitman (1991). Such results have been extended, by incorporating equilibrium conditions, to show the convergence of finite player games to mean field games using the master equation; see e.g. Cardaliaguet et al. (2015) or Lacker (2020) for proofs based respectively on PDE and stochastic arguments. Finer results have also been obtained, such as a large deviation principle by Delarue et al. (2020). Notice that these results do not apply readily to the approach we discuss above, because it does not approximate the mean field game by a finite player game: instead, it uses a finite population only to approximate the distribution. But the players are still behaving as if they were in an infinite population. However, we expect that similar proof techniques could be used to prove convergence of the approximate solution to the true solution.

3.1.2 Discrete State Space Approximation

In this section, we outline the second approach: projecting the distribution onto a finite collection of mass points so it can be approximated by the distribution by a histogram, i.e., a vector of weights on a finite number of points. This is a special case of a more general projection technique that we describe in Appendix E.

We consider N^x points, denoted by $x_1, \dots, x_{N^x}$, in $\mathcal{X}$. We will approximate g by a vector $\hat{g} \in \mathbb{R}^{N^x}$ of values at $x_1, \dots, x_{N^x}$. The KFE under optimal control (2.6) is replaced by an ordinary differential equation in dimension N^x of the form:

$$d\hat{g}_t = \hat{\mu}_{\hat{g}}(z_t, \hat{g}_t)dt \tag{3.1}$$

describing the evolution of mass at the values at $x_1, \dots, x_{N^x}$. The right-hand-side needs to be obtained using information from the Kolmogorov Forward Equation (2.6). In our numerical examples we use a finite difference approximation to the Kolmogorov Forward Equation to derive $\hat{\mu}_{\hat{g}}$. However, the technique can be applied to other types of projections.

The solution V to the master equation is replaced by a function $V : \mathcal{X} \times \mathcal{Z} \times \mathbb{R}^{N^x} \to \mathbb{R}$ which solves the PDE $(\hat{\mathcal{L}}V)(x, z, \hat{g}) = 0$ subject to the boundary conditions, where the (approximate) master equation operator $\hat{\mathcal{L}}$ is given by:

$$\begin{aligned}
\hat{\mathcal{L}}V &= \hat{\mathcal{L}}^h V + \hat{\mathcal{L}}^g V, \quad \text{where} \\
(\hat{\mathcal{L}}^h V)(x, z, \hat{g}) &:= (\mathcal{L}^h V)(x, z, \hat{g}) \\
(\hat{\mathcal{L}}^g V)(x, z, \hat{g}) &= \sum_{i=1}^{N^x} [\hat{\mu}_{\hat{g}}(z, \hat{g})]_i \frac{\partial V}{\partial \hat{g}_i}(x, z, \hat{g}).
\end{aligned}$$

We expect the solution of the approximate master equation $V(x, z, \hat{g})$ to converge towards the solution of the true master equation (2.7). Convergence of mean field games with finite state spaces to mean field games with continuous state spaces has been proved in Hadikhanloo and Silva (2019) for mean field games of first order, i.e., without idiosyncratic noise (nor common noise), and in Bertucci and Cecchin (2022) including in the case of common noise. This research direction is also related to numerical methods based on Markov chain approximations, as developed in Bayraktar et al. (2018). Notice that these results do not directly apply to our method because here we do not discretize the state space for x or z: we only approximate the distribution by a distribution over a finite subset of the state space. However, we expect that similar proof techniques could be used to show the convergence of the solution of the approximate master equation to the solution of the true master equation.

3.2 Neural Network Approximations

Both approaches in section 3.1 lead to finite approximations to the density, $\hat{g}$, and the master equation operator $\hat{\mathcal{L}}$. However, the resulting master equations are high dimensional and so cannot be solved by traditional numerical techniques. Instead, we replace the solution to the approximate master equation by a neural network and deploy tools from the "deep learning" literature to "train" the neural network to solve the approximate master equation.

A neural network is a type of parametric functional approximation that is built by composing affine and non-linear functions in a chain or "network" structure (see Goodfellow et al. (2016) for a detailed discussion). We let $\hat{X} := \{x, z, \hat{g}\}$ denote the collection of inputs into the approximate value function. We denote the neural network function to the value function by $V(\hat{X}) \approx \hat{V}(\hat{X}; \theta)$, where θ are the parameters in the neural network approximation that depend upon the architecture, i.e., the form of the approximation. There are many types of neural network approximations. The simplest form is a "feedforward" neural network which is defined by:

$$
\begin{aligned}
h^{(1)} &= \phi^{(1)}(W^{(1)}\hat{X} + b^{(1)}) && \ldots \text{Hidden layer 1} \\
h^{(2)} &= \phi^{(2)}(W^{(2)}h^{(1)} + b^{(2)}) && \ldots \text{Hidden layer 2} \\
&\vdots \\
h^{(H)} &= \phi^{(H)}(W^{(H)}h^{(H-1)} + b^{(H)}) && \ldots \text{Hidden layer } H \\
o &= W^{(H+1)}h^{(H)} + b^{(H+1)} && \ldots \text{Output layer} \\
\hat{V} &= \phi^{(H+1)}(o) && \ldots \text{Output}
\end{aligned}
\tag{3.2}
$$

where the $\{h^{(i)}\}_{i \le H}$ are vectors referred to as "hidden layers" in the neural network, $\{W^{(i)}\}_{i \le (H+1)}$ are matrices referred to as the "weights" in each layer, $\{b^{(i)}\}_{i \le (H+1)}$ are vectors referred to as the "biases" in each layer, $\{\phi^{(i)}\}_{i \le (H+1)}$ are non-linear functions applied element-wise to each affine transformation and referred to as "activation functions" for each layer. The length of hidden layer, $h^{(i)}$, is referred to as the number of neurons in hidden layer i. The total collection of parameters is denoted by $\theta = \{W^{(i)}, b^{(i)}\}_{i \le (H+1)}$. The goal of deep learning is to train the parameters, θ, to make $\hat{V}(\cdot; \theta)$ a close approximation to V.

The neural network defined in (3.2) is called a "feedforward" because hidden layer i cannot depend on hidden layers $j > i$. This is in contrast to a "recursive" neural network where any hidden layer can be a function of any other hidden layer. It is called "fully connected" if all the entries in the weight matrices can be non-zero so each layer can use all the entries in the previous layer. In this paper, we will consider a fully connected "feedforward" network to be the default network. This is because these networks are the quickest to train and so we typically start by trying out this approach. However, there are applications where we find that more complicated neural network architectures are useful. In particular, we find that the type of recursive neural network suggested by the Deep Galerkin

Method in Sirignano and Spiliopoulos (2018) is helpful for finite state space approximations.

3.3 Solution Algorithm

We train the neural network to learn parameters θ that minimize the error in the master equation and boundary conditions. We describe the key steps in Algorithm 1.[6] Essentially, the algorithm generates random points in the discretized states space $\{x, z, \hat{g}\}$, then calculates the error made by the neural network in the master equation on those points, and updates the neural network parameters to decrease the error in the master equation. In fact, the loss consists of three terms: $\mathcal{E}^e$ for the PDE residual, $\mathcal{E}^b$ for the boundary condition, and a last one, $\mathcal{E}^s$, which is used to incorporate information about the shape of the solution (e.g., monotonicity or concavity). Specific examples of these functions will be discussed in the following sections.

In the deep learning literature, this approach is sometimes referred to as "unsupervised" learning (e.g. Azinovic et al. (2022)) because we do not have direct observations of the value function, $V(x, z, \hat{g})$, and instead have to learn the value function indirectly via the master equation.

There are some features of the algorithm that are typical to deep learning problems but less common in other economics techniques so we address them here:

(i). *Why do we draw a new sample each epoch rather than fixing a sample from the start?* We find that fixing the sample across epochs leads to overfitting problems where the neural network matches the master equation solution at the sample points well but interpolates poorly in the rest of the state space.

(ii). *Could the same algorithm be run with an alternative parametric approximation such as Chebyshev polynomials?* In principle, it is possible. But, a key feature of the training algorithm is that we need to be able to calculate automatic derivatives. Chebyshev polynomials have a smaller number of parameters but complicated basis functions whereas our projection approach with neural networks have a large number of parameters but simple basis functions. In addition, the machine learning literature has invested heavily in getting non-linear optimizers to work well with neural networks. For all these reasons, it is easier to run non-linear optimizers with neural networks.

(iii). *Does this algorithm solve for the global or local minimum?* The stochastic gradient descent algorithm (or one of its variants, such as Adam) calculates the loss on random collections of points and so has some ability to wander the parameter space looking for the global minimum. We do not always need to find the global minimum since, for suitable architectures, there are some local minima that are reasonable approximations.

[6]The generic pseudo-code given in Algorithm 1 can be modified in practice. For example, instead of fixing a precision threshold, one can fix a number of iterations, and instead of using a fixed sequence of learning rates, one can use an adaptive method, such as Adam.

(iv). *Why do we need shape constraints?* We find that deep learning algorithms can "cheat" by finding "bad" approximate solutions. For example, they are likely to find solutions where the value function has zero derivative with respect to dimensions where there is limited curvature. More generally, it seems that in some instances there are local minima which are quite easily learnt by the neural network and yet are very different from the true solution. We find that enforcing shape constraint such as monotonicity or concavity helps prevent the algorithm getting stuck at these solutions.

(v). *What about slowing down the updating?* For the projection methods, we use a version of a "Howard improvement algorithm" to slow down the rate of updating (by fixing the policy rule for some iterations and just updating the value function). Duarte (2018) and Gopalakrishna (2021) suggest introducing a "false" time-step to eliminating "cheat" solutions but so far we have not found this necessary (or computationally implementable for high dimensional models). Instead, we find that using shape constraints to eliminate bad approximate solutions works well in our case.

(vi). *What about imposing symmetry and/or dimension reduction?* Han et al. (2021) and Kahou et al. (2021) suggest feeding the distribution through a preliminary neural network that reduces the dimension and imposes symmetry; see also Germain et al. (2022a) in the case of social optima. In our numerical experiments, we find that we can solve the problem with and without this approach. We do find that when we impose dimension reduction the accuracy of our solution is very sensititve to how much dimension reduction is done.

At a high level, the algorithm is straightforward. However, successfully implementing the deep learning approach is not trivial and involves many problem specific adjustments. In the following sections, we work through canonical macroeconomic problems in detail and compare different ways of implementing the algorithm.

3.4 Advantages of Neural Networks For Continuous Time

There is a number of advantages to using neural networks to solve differential equations, some of which are particular to continuous time problems. One advantage, as we have mentioned, is that we can deal with high dimensional differential equations. This allows us to work with high dimensional approximations to distributions that traditional techniques (e.g. finite difference and spectral methods) cannot solve in reasonable time. As we show in the following examples, this allows us to solve models with heterogeneous agents and aggregate shocks.

A second advantage is that we are able to use automatic differentiation to calculate the derivatives to evaluate the differential equations. In continuous time, the master equation contains a differential operator and so numerical techniques need to focus on efficient ways of calculating derivatives. This is very different from discrete time, where the Bellman

Algorithm 1: Generic Solution Algorithm

Input : Initial neural network parameters θ^0, number of points M^e and M^b to sample respectively inside the domain and on the boundary, positive weights κ^e, κ^b, and κ^s on the master equation errors; sequence of learning rates $\{\alpha_n : n \geq 0\}$, precision threshold ϵ

Output: A neural network approximation $(x, z, \hat{g}) \mapsto \hat{V}(x, z, \hat{g}; \theta)$ of the value function $(x, z, \hat{g}) \mapsto V(x, z, \hat{g})$ solving the approximate master equation

1. Initialize the neural network with parameters θ^0.
2. At iteration $n = 0, 1, 2, \ldots$ with guess θ^n:
   (a) Generate M^e sample points, $S^{ne} = \{(x_m, z_m, \hat{g}_m)\}_{m \leq M^e}$ for evaluating the master equation error and M^b sample points, $S^{nb} = \{(\underline{x}_m, z_m, \hat{g}_m)\}_{m \leq M^b}$ for evaluating the boundary condition error at boundary points $\underline{x}_m \in \partial\mathcal{X}$. Let $S^n := \{S^{ne}, S^{nb}\}$ denote the collection of sample points at iteration n.

   (b) Calculate the weighted average error:

$$\mathcal{E}(\theta^n, S^n) = \kappa^e \mathcal{E}^e(\theta^n, S^{ne}) + \kappa^b \mathcal{E}^b(\theta^n, S^{nb}) + \kappa^s \mathcal{E}^s(\theta^n, S^{ne})$$

   where $\mathcal{E}^e$, and boundary condition error, $\mathcal{E}^b$, and penalty for a "wrong" shape, $\mathcal{E}^s$. The errors are typically taken to be mean-squared errors:

$$\mathcal{E}^e(\theta^n, S^{ne}) := \frac{1}{M^e} \sum_{m \leq M^e} |(\hat{\mathcal{L}}\hat{V}(\cdot; \theta))(x_m, z_m, \hat{g}_m)|^2$$

$$\mathcal{E}^b(\theta^n, S^{nb}) := \frac{1}{M^b} \sum_{m \leq M^b} |\Phi(\underline{x}, z_m, \hat{g}_m, \hat{V}(\underline{x}_m, z_m, \hat{g}_m; \theta), D_x\hat{V}(\underline{x}, z_m, \hat{g}_m; \theta))^-|^2$$

   where $(\cdot)^-$ denotes the negative part and the derivatives in the differential operator and boundary condition are calculated using automatic differentiation. The definition of $\mathcal{E}^s$ depends on the problem (see examples in the next sections).

   (c) Update the parameters using "deep learning" toolkit. We typically use a "stochastic gradient descent" style method: at each point:

$$\theta^{n+1} = \theta^n - \alpha_n D_\theta \mathcal{E}(\theta^n, S^n)$$

   where $D_\theta\mathcal{E}$ is the gradient (i.e., vector differential) operator.

   (d) Repeat until $\mathcal{E}(\theta^n, S^n) \leq \epsilon$.

equation has expectation operators and so numerical approaches need to focus on efficient ways of calculating expectations. Traditional continuous time techniques, especially finite difference methods, typically have difficulty working with regions where the solution has high curvature because the finite difference approximation to the derivative breaks down. Neural network techniques are particularly useful for continuous time because they can use automatic differentiation rather than working with discrete state space approximations to derivatives.

A third advantage is that neural networks work well with randomly sampling state space points. When using spectral methods, such as Chebyshev polynomials, the grid for the state space need to be carefully chosen to prevent synchronisation in the oscillations of the polynomials. By contrast, for neural networks, we can sample randomly from the state space and so use active learning to focus on regions with large errors.

4 Example: Uninsurable Income Risk

A canonical macroeconomic model with heterogeneous agents is Aiyagari (1994), which we refer to as the Aiyagari-Bewley-Huggett (ABH) model (following the terminology used in the continuous time formulation in Achdou et al. (2017)). We start by using our solution technique to solve this model because there are existing accurate solution techniques and so we can compare our solution technique in detail. In the next section we solve the Krusell and Smith (1998), which introduces aggregate productivity shocks.

4.1 Model Specification

In this subsection, we briefly explain how ABH model fits into our general framework. We have included a more detailed derivation of the master equations for the Aiyagari (1994) and Krusell and Smith (1998) models in our: "Online Appendix: Krusell and Smith (1998) Model".

Setting: In the ABH model, there is a perishable consumption good and a durable capital stock. The economy consists of a unit continuum $I = [0, 1]$ of households and a representative firm. The representative firm controls the production technology, which produces consumption goods according to the production function:

$$Y_t = e^z K_t^\alpha L_t^{1-\alpha}$$

where K_t is the capital rented at time t, L_t is the labour hired at time t, and z is productivity. Firm productivity is fixed so there is no aggregate shock process in the economy.

Households: Each household $i \in [0, 1]$ has discount rate ρ and gets flow utility $u(c_t^i) = (c_t^i)^{1-\gamma}/(1-\gamma)$ from consuming c_t^i consumption goods at time t. Each household has two idiosyncratic states $x_t^i = [a_t^i, n_t^i]$, where a_t^i is the household's net wealth and $n_t^i \in \{n_1, n_2\}$ is the household's labor endowment, where $n_1 < n_2$ so n_1 is interpreted as unemployment and n_2 is interpreted as employment. Labor endowments switch idiosyncratically between n_1 and n_2 at Poisson rate $\lambda(n_t^i)$. Households choose consumption c_t^i and their idiosyncratic state evolves according to:

$$dx_t^i = d\begin{bmatrix} a_t^i \\ n_t^i \end{bmatrix} = \begin{bmatrix} s(a_t^i, n_t^i, c_t^i, r_t, w_t) \\ 0 \end{bmatrix} dt + \begin{bmatrix} 0 \\ \check{n}_t^i - n_t^i \end{bmatrix} dJ_t^i$$

where r_t is the return on household wealth, w_t is the wage rate, $\check{n}_t^i$ is the complement of n_t^i, J_t^i is an idiosyncratic Poisson process with arrival rate $\lambda(n_t^i)$, and the agent's saving function is given by:

$$s(a, n, c, r, w) = wn + ra - c.$$

Assets, markets, and financial frictions: Each period, there are competitive markets for goods, capital rental, and labor. We use goods as the numeraire. We let r_t denote the rental rate on capital, w_t denote the wage rate on labor, and $q_t = [r_t, w_t]$ denote the price vector. Firm optimization and market clearing imply that, given g_t, the prices r_t and w_t solve:

$$\begin{aligned} r_t &= \partial_K F(K_t, L) - \delta, & w_t &= \partial_L F(K_t, L), \\ K_t &= \sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} a g_t(a, n_j) da & L &= \sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} n_j g_t(a, n_j) da. \end{aligned}$$

So, in the terminology of Section 2, we write the prices explicitly as functions of g_t:

$$q_t = \begin{bmatrix} r_t \\ w_t \end{bmatrix} = \begin{bmatrix} \partial_K F\left(\sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} a g_t(a, n_j) da, L\right) - \delta \\ \partial_L F\left(\sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} a g_t(a, n_j) da, L\right) \end{bmatrix} =: Q(g_t) \tag{4.1}$$

Asset markets are incomplete so households cannot insure their idiosyncratic labor shocks. Instead, households can trade claims to the aggregate capital stock in a competitive asset market. The original Krusell and Smith (1998) model imposes the "borrowing constraint" that each agent's net asset position, a_t^i, must satisfy $a_t^i \geq \underline{a}$, where $\underline{a}$ is an exogenous "borrowing limit". This generates an inequality boundary constraint and a mass point, as discussed in Achdou et al. (2022). However, this causes difficulties for the neural network. So, to make the problem more tractable, we instead follow Brzoza-Brzezina et al. (2015)

and introduce a penalty function ψ at the left boundary, replacing the agent flow utility by:

$$U(a_t, c_t) = u(c_t) + \mathbf{1}_{a_t \leq \underline{a}} \psi(a_t).$$

The penalty function we use here is the quadratic function: $\psi(a) = -\frac{1}{2}\kappa(a - \underline{a})^2$ where κ is a positive constant.

Master equation: Let $c^*(a, n, g)$ denote the equilibrium optimal household control. Then, the master equation for the ABH model is given by the following:

$$0 = (\mathcal{L}V)(a, n, g) = (\mathcal{L}^h V)(a, n, g) + (\mathcal{L}^g V)(a, n, g),$$

where the operators $\mathcal{L}^h$ and $\mathcal{L}^g$ are defined by:

$$\begin{aligned}
(\mathcal{L}^h V)(a, n, g) := & -\rho V(a, n, g) + u(c^*(a, n, g)) + \mathbf{1}_{a \leq \underline{a}} \psi(a) \\
& + \partial_a V(a, n, g) s(a, n, c^*(a, n, g), r(g), w(g)) \\
& + \lambda(n)(V(a, \check{n}, g) - V(a, n, g)) \\
(\mathcal{L}^g V)(a, n, g) := & \int_{-\infty}^{\infty} \partial_b \frac{\partial V}{\partial g}(a, n, g)(b) s\left(a, n, c^*(a, n, g), r(g), w(g)\right) g(b, n) db \\
& + \int_{-\infty}^{\infty} \frac{\partial V}{\partial g}(a, n, g)(b) \left(\lambda(\check{n}) g(b, \check{n}) - \lambda(n) g(b, n)\right) db,
\end{aligned}$$

where $\check{n}$ denotes the complement of n, where $r(g)$ and $w(g)$ solve the system of equations (4.1), and the optimal consumption policy satisfies the first order optimality condition:

$$\partial_a V(a, n, g) = u'(c^*(a, n, g)).$$

So the optimal control is a function of the $\partial_a V(a, n, g)$. It will thus be more convenient to solve the master equation for the partial derivative, which we denote by $W(a, n, g) := \partial_a V(a, n, g)$. This function is solution to the following PDE:

$$0 = (\mathfrak{L}W)(a, n, g) = (\mathfrak{L}^h W)(a, n, g) + (\mathfrak{L}^g W)(a, n, g)$$

with the boundary conditions, where the operators $\mathfrak{L}^h$ and $\mathfrak{L}^g$ are defined by:

$$\begin{aligned}
(\mathfrak{L}^h W)(a,n,g) &:= (r(g)-\rho)W(a,n,g) + \mathbf{1}_{a\le \underline{a}}\psi'(a) \\
&\quad + \partial_a W(a,n,g)s(a,n,c^*(a,n,g),r(g),w(g)) \\
&\quad + \lambda(n)(W(a,\check{n},g)-W(a,n,g)) \\
(\mathfrak{L}^g W)(a,n,g) &:= \int_{-\infty}^{\infty} \partial_b \frac{\partial W}{\partial g}(a,n,g)(b) s\left(a,n,c^*(a,n,g),r(g),w(g)\right) g(b,n) db \\
&\quad + \int_{-\infty}^{\infty} \frac{\partial W}{\partial g}(a,n,g)(b)\left(\lambda(\check{n})g(b,\check{n}) - \lambda(n)g(b,n)\right) db
\end{aligned} \tag{4.2}$$

with the first order optimality condition:

$$W(a,n,g) = u'(c^*(a,n,g)).$$

In the next sections, we solve this master equation numerically using a straightforward adaptation of Algorithm 1. The parameters that we use in numerical experiments are in Appendix B.1.

4.2 Details on the Finite Agent Approximation

We start with the finite agent approximation. In this case, we replace the distribution by the positions of the agents:

$$\hat{g} = \{(a^i, n^i)\}_{i\le I}$$

where $I = 41$ agents. We found that it was helpful to customize the generic Algorithm 1 in the following ways to solve the Aiyagari model.

Choice of Neural Network (Step 1): We use a fully connected feed-forward neural network with 5 layers and 64 neurons per layer. We use a tanh activation function between layers and an ELU activation at the output level. We initialize the neural network so that $W(a,\cdot)$ has an exponential shape with negative exponent. This is done through a pre-training phase. We tried setting up the network with and without the imposition of symmetry suggested by Kahou et al. (2021) and Han et al. (2021) but ultimately found that the solution was more robust without the symmetry. See Appendix B.4 for more details.

Sampling Approach (Step 3a): We sample points of the form $\{(a^i,n^i),\{a^j,n^j\}_{j\in(I-1)}\}$ on the interior of the state space. For the idiosyncratic variable, a^i, we sample using an active sampling technique similar to those developed by Gopalakrishna (2021) and Lu et al. (2021a). We divide the state space for $a \in [0,20]$ in a collection of regions. For each iteration of the algorithm, we calculate the average loss in each region. We then add extra points to the regions with the largest loss. We use the active sampling approach because it allows us to actively learn where the algorithm is having trouble minimizing the loss function. The interval $[\underline{a}, \overline{a}]$ is evenly partitioned into 2^4 subintervals. We do active sampling after 2,000 of epochs to make it efficient. We calculate the residual error in each subinterval, find where it is the largest and add $2^4, 2^3, 2^2$ points to it, its nearest, and second nearest neighboring subinvervals. For sampling the population of agents, $\{a^j, n^j\}_{j\in(I-1)}$, we generate a random interest rate, then generate a random distribution of agents and scale their individual wealth so the equilibrium interest rate is the randomly drawn interest rate. The interest rate r is drawn from a uniform distribution on $[r_{lb}, r_{rb}]$, with $r_{lb} = -0.05, r_{rb} = 0.05$. Thanks to the fact that we deal with the boundary condition through a penalty, we do not need to sample points on the boundary, nor to evaluate the error $\mathcal{E}^b$. This is quite robust to different model parameters.

Loss Function Specification (Step 3b): In order to help the neural network maintain monotonicity and avoid complex number consumption from the optimality condition $u'(c^*) = W$, we introduced a penalty for having W no less than a lower bound $\underline{W} > 0$ to ensure that W is strictly positive. This is taken into account through the $\mathcal{E}^s$ loss term.

4.3 Details on Discrete State Space Approximation

We now turn to the second approximation method. Algorithm 1 is implemented in the following way.

Choice of Neural Network (Step 1): We approximate $W = \partial_a V$ by a neural network $\hat{W}(\cdot; \theta_W)$ with parameters θ_W. We use an architecture similar to the one proposed by Sirignano and Spiliopoulos (2018). The main differences are two-fold. On the last layer, we use an activation function ensuring that the output is always positive. Furthermore, the first layers of the neural network realize an embedding of the distribution $\hat{g}$ in a space of lower dimension. This allows us to work with very fine discretizations of the state space for a.

Sampling Approach (Step 3a): We sample points of the form $(a, y, \hat{g})$ where (a, y) follows a uniform distribution over $[\underline{a}, \overline{a}] \times \{y_1, y_2\}$ and $\hat{g}$ is obtained as a randomly perturbed version of the equilibrium steady state $\hat{g}_{ss}$. More precisely, we take $\hat{g}_{i,j} = \frac{\omega_{i,j}\hat{g}_{ss,i,j}}{\sum_{i=1}^{N_x} \omega_{i,j}\hat{g}_{ss,i,j}}$ for $j = 1, 2$, where the $\omega_{i,j}$ are i.i.d. with uniform distribution over an interval of the form$[1-d_g, 1+d_g]$, with $d_g \in (0, 1)$. Thanks to the fact that we deal with the boundary condition through a penalty, we do not need to sample points on the boundary, nor to evaluate the error $\mathcal{E}^b$.

Loss Function Specification (Step 3b): To compute the error $\mathcal{E}^e$, the operators $\hat{\mathfrak{L}}^h$ and $\hat{\mathfrak{L}}^g$ approximating the operators introduced in (4.2) are represented as follows. $\hat{\mathfrak{L}}^h \hat{W}(a, y, \hat{g})$ is obtained by evaluating the neural network for $\hat{W}$ and by computing its partial derivative with respect to a using automatic differentiation. $\hat{\mathfrak{L}}^g \hat{W}$ is obtained by taking the product of the gradient of $\hat{W}$ with respect to $\hat{g}$, which is computed using automatic differentiation, and $\hat{\mu}^{\hat{g}}$, which is computed as follows. The ODE (3.1) which describes the evolution of $\hat{g}$ is obtained by using a finite-difference scheme proposed by Achdou et al. (2017). The right-hand side is written in the form $\hat{\mu}_{\hat{g}}(\hat{g}_t) = A^\top \hat{g}_t$, where $A^\top$ is a matrix which approximates spatial derivatives by finite differences. It relies on an upwind scheme for more stability. On top of this error $\mathcal{E}^e$, we also add an extra term $\mathcal{E}^s$ that penalizes the neural network when $\partial_a \hat{W}$ is positive. This extra term encourages $\hat{W} = \partial_a \hat{V}$ to be a decreasing function of a, which is consistent with the fact that $\hat{V}$ should be a concave function of a, for any given $(y, \hat{g})$.

After each gradient step updating the parameters θ_W of the neural network for $\hat{W}$, we update the parameters θ_c of the neural network for c^* by doing a few stochastic gradient descent steps in the direction of minimizing a Monte Carlo approximation of $\mathbb{E}_{a,y,\hat{g}}|c^*(a, y, \hat{g}; \theta_c) - (u')^{-1}(\hat{W}(a, y, \hat{g}; \theta_W))|^2$.

4.4 Comparison to Finite Difference Solution

In this section, we compare the output from our neural network solution to the output from our a finite difference solution to the Aiyagari model. We first compare the solution at the steady state and then compare the solution along transition paths following MIT shocks.

4.4.1 Steady State Comparison

Figure 1 plots the steady state consumption policy rule, derivative of the value function, probability density function (pdf), and cumulative distribution function (cdf) for the solutions from the finite agent code, the finite state space code, and finite difference code. As can be seen, the neural network solutions align very closely to the finite difference solution. The loss is:

	Master equation loss	MSE(NN, FD)
Finite Agent NN	3.135×10^{-5}	4.758×10^{-5}
Discrete State Space NN	2.548×10^{-4}	1.590×10^{-2}

Table 1: Neural Networks' results for solving master equations. The master equation loss is the mean squared error of residuals. MSE(NN,FD) is the mean squared difference of consumption solved by neural network and finite difference. The training loss at each iteration is available in Appendix D.

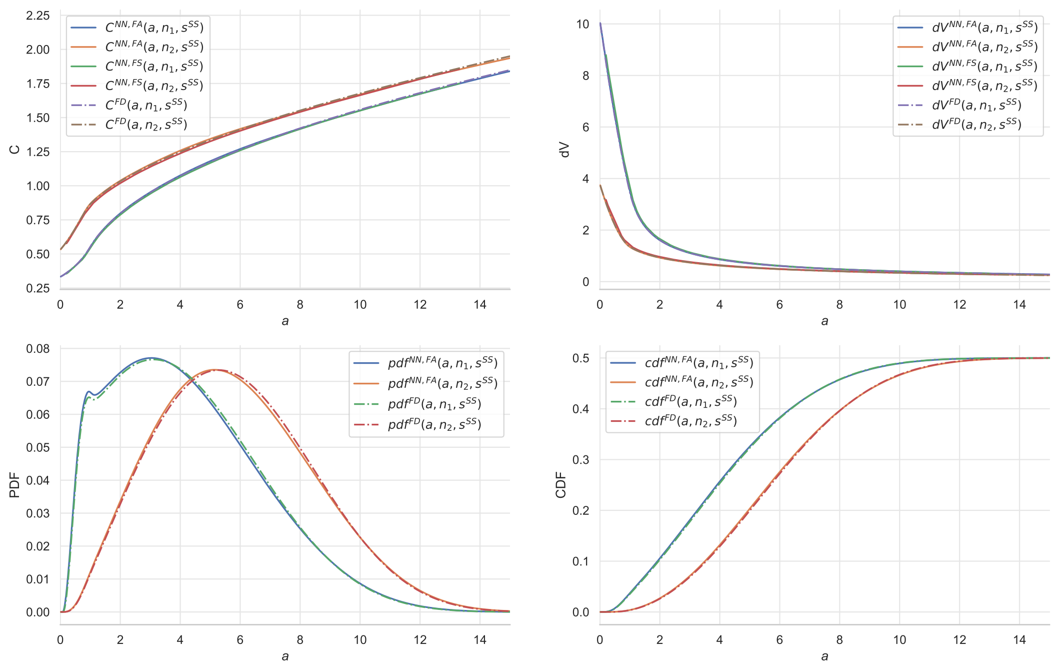


Figure 1: Comparison between neural network and finite difference solutions for the Aiyagari model. The top left plot shows the consumption policy rule. The top right shows the derivative of the value function, the bottom left shows the pdf, and the bottom right shows the cdf. The superscript NN, FA refers to the finite agent neural network code, the superscript NN, FS refers to the finite state space neural network code, and the superscript FD refers to the finite difference code.

4.4.2 Comparison to Finite Difference Solution: Transition Dynamics

We can also compare the transition path. An important advantage of having a global solution, i.e., c^* as a function of the distribution, is that we can solve for the transition path without a "shooting algorithm", as is commonly done in the finite difference literature. This is an advantage because shooting algorithms are often unstable, particularly for systems with a large number of prices that require complicated guess for the price path. Instead, with a full solution to the master equation, we can solve the Kolmogorov Forward Equation (KFE) directly as time-dependent consumption is a function of density $g(a, n)$.

$$\frac{\partial g_t(a,n)}{\partial t} = -\frac{\partial}{\partial a}(s(a,n,g_t)g_t(a,n)) + \lambda(g_t(a,\tilde{n}) - g_t(a,n)) \equiv \mathcal{A}^\top g$$

where $\mathcal{A}$ is transition matrix. We illustrate this for the finite agent approximation. The iterative procedure for solving the KFE in this case is summarized in Algorithm 2 below. We expand on the different approaches for calculating the transition path in Appendix B.3.

We compare the neural network and finite difference transition paths in Figure 2 below.

Algorithm 2: Finding Transition Path by Neural Network: Aiyagari case

```
Input  : Number of agents $N$, time step size $\Delta t$, number of time steps $N_T$, number
         of simulations $N_{sim}$
Output: A transition path $g = \{g_t : t = 0, \Delta t, \dots, N_T \Delta t\}$
for $n = 0, \dots, N_T - 1$ do
    for $k = 1, \dots, N_{sim}$ do
        Sample $N$ agents $\{s_i : i = 1, \dots, N\}$ from distribution $g_t$ at time $t = n\Delta t$.
        Given other agents' state $s_{-i}$, calculate the consumption $c(a, n, s_{-i})$,
          equilibrium capital return $r(s_{-i})$ and wage $w(s_{-i})$. Then construct $\mathcal{A}_k$ by
          finite difference scheme.
    end
    Take the average: $\bar{\mathcal{A}} = \frac{1}{N_{sim}} \sum_{k=1}^{N_{sim}} \mathcal{A}_k$.
    Update $g_t$ by implicit method: $g_{t+\Delta t} = (I - \bar{\mathcal{A}}^\top \Delta t)^{-1} g_t$.
end
```

We discuss how the finite difference solution is computed in Appendix B.3. In this numerical experiment, we train our neural network at $z = 0$ and we start from an economy in its steady state with productivity $z_t = -0.10$ for $t = 0$. At $t = 0^+$, an unexpected positive productivity shock brings z from -0.10 to $z_t = 0$ permanently. We solve distributional dynamics by Algorithm 2, and use steady state at $z = -0.10$ as the initial condition. We plot the percentage change of capital, capital return and wage evolution respectively in the first row and the second row of Figure 2. The difference between neural net's transition paths and finite difference's transition paths are less than 0.1%. The lower panels of Figure 2 compares neural network and finite difference probability density at time $t = 15$ and $t = 30$.

5 Example: Uninsurable Income Risk and TFP Shocks

A canonical extension of the previous model is the Krusell and Smith (1998) model, which introduces TFP shocks. This model is often used as a test model for new numerical techniques. We have included a more detailed derivation of the master equations for the Aiyagari (1994) and Krusell and Smith (1998) models in our: "Online Appendix: Krusell and Smith (1998) Model".

5.1 Model Specification

Environment Changes: The model is the same as the previous section, except that now the firm can produce consumption goods according to the production function:

$$Y_t = e^{z_t} F(K_t, L_t)$$

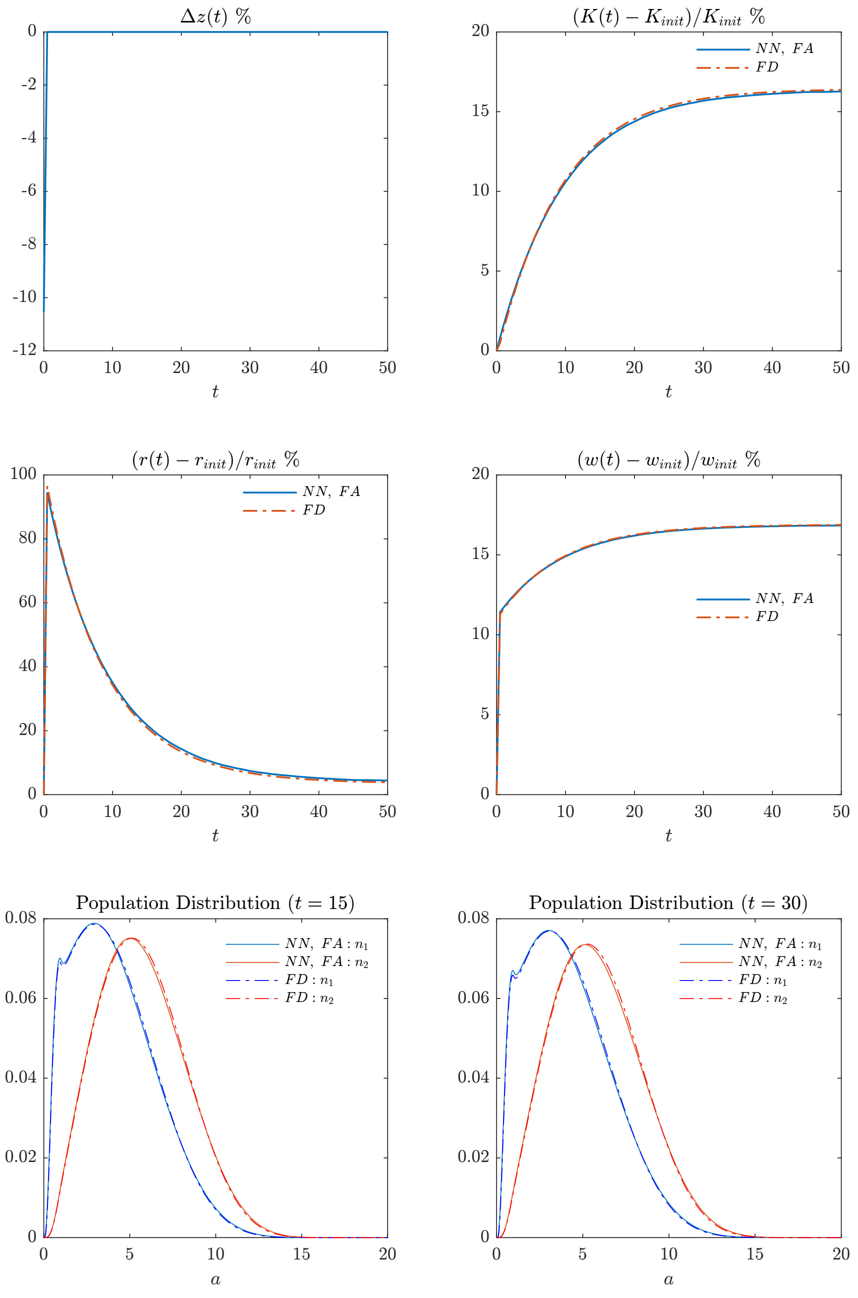


Figure 2: Comparison between neural network and finite impulse response for the Aiyagari model. The top left plot is the TFP shock path, the top right panel is the aggregate relative capital change, the middle left panel plots the relative capital return change, and the middle right panel plots the relative wage change. The bottom left and bottom right are snapshots of probability density at $t = 15$ and $t = 30$. *NN, FA* refers to the finite agent neural network code, and the *FD* refers to the finite difference code. Subscript *init* is the initial value at the steady state $z = z(0)$.

where z_t is the aggregate productivity, which follows an Ornstein-Uhlenbeck process:

$$dz_t = \eta(\overline{z} - z_t)dt + \sigma dB_t^0$$

with lower and upper reflecting boundaries at z_{min} and z_{max} respectively.

For this model, g_t now denotes the population density across $\{a_t^i, n_t^i\}$ at time t, given a filtration $\mathcal{F}_t^0$ generated by the sequence of aggregate shocks. Given g_t and z_t, the prices r_t and w_t solve (and so are implicitly functions of g_t and z_t):

$$\begin{aligned} r_t &= e^{z_t}\partial_K F(K_t, L) - \delta, & w_t &= e^{z_t}\partial_L F(K_t, L), \\ K_t &= \sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} a g_t(a, n_j) da & L &= \sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} n_j g_t(a, n_j) da. \end{aligned} \tag{5.1}$$

The master equation operators become:

$$\begin{aligned} (\mathcal{L}^h V)(a, n, z, g) :=& -\rho V(a, n, z, g) + u(c^*(a, n, z, g)) + \mathbf{1}_{a\leq \underline{a}}\psi(a) \\ &+ \partial_a V(a, n, z, g)s(a, n, c^*(a, n, z, g), r(g), w(g)) \\ &+ \lambda(y)(V(a, \check{n}, z, g) - V(a, n, z, g)) \\ &+ \partial_Z V(a, n, z, g)\eta(\overline{z} - z) + \frac{1}{2}\sigma^2 \partial_{zz} V(a, n, z, g) \\ (\mathcal{L}^g V)(a, n, z, g) :=& \int_{\overline{a}}^{\infty} \frac{\partial V}{\partial g}(a, n, z, g)(b)\left(\lambda(\tilde{n})g(b, \tilde{n}) - \lambda(y)g(b, n)\right) db \\ &+ \int_{\overline{a}}^{\infty} \partial_b \frac{\partial V}{\partial g}(a, n, g)(b) s\left(a, n, c^*(a, n, z, g), r(g, z), w(g, z)\right) g(b, n) db, \end{aligned}$$

where $r(g, z)$ and $w(g, z)$ solve the system of equations (5.1), the optimal control satisfies:

$$\partial_a V(a, n, z, g) = u'(c^*(a, n, z, g)).$$

As for the Aiyagari model, we apply the envelope theorem to get a PDE for $W(a, n, z, g) := \partial_a V(a, n, z, g)$, which is what we solve using deep learning.

5.2 Details on Finite Agent Algorithm Implementation

We customize the generic Algorithm 1 in the following ways to solve the Krusell-Smith model. As for the Aiyagari model, we solve a model with 41 agents.

Choice of Neural Network (Step 1): We use a fully connected feed-forward neural network with 5 layers and 64 neurons per layer. We use a tanh activation function between layers. We initialize the neural network so that $W(a, \cdot)$ has an exponential shape with negative exponent. This is done through a pre-training phase. The fact that we can use the same architecture as for the simpler ABH model illustrates the fact that for our method, solving the problem with aggregate shock is not much more difficult than solving the problem without.

Sampling Approach (Step 3a): We sample points of the form $\{(a^i, n^i), \{a^j, n^j\}_{j\in(I-1)}, z\}$. For idiosyncratic variable, (a^i, n^i), and population of agents $\{a^k, n^j\}_{j\in(I-1)}$, the sampling method is the same as in section 4.2. We sample aggregate variable z uniformly from interval $[z_{min}, z_{max}]$.[7]

Loss Function Specification (Step 3b): As for the Aiyagari model, we penalize $\partial_a V(a, \cdot) < 0$. In addition, to capture curvature in Z dimension, we introduce a penalty for having $d\bar{W}/dZ > 0$, which is equivalent to penalizing $dC/dZ < 0$, where C is the aggregate consumption. This is taken into account in the loss term $\mathcal{E}^s$ of Algorithm 1.

5.3 Details on Discrete State Space Algorithm Implementation

For the discrete state space approximation method, we customize Algorithm 1 in the following ways to solve the Krusell-Smith model. We approximate the distribution $\hat{g}$ with a 186-dimensional vector based on a 93-point discretization of the asset dimension a.

Choice of Neural Network (Step 1): We approximate $W = \partial_a V$ by $\hat{W}(a, n, \hat{g}; \theta_W) = (a_0 + a)^{-\tilde{\eta}} N(a, n, \hat{g}; \theta_W)$, where $N(\cdot; \theta_W)$ is a neural network with trainable parameter vector θ_W and non-trainable shape parameters $a_0, \tilde{\eta} \geq 0$. For N we use the same architecture based on Sirignano and Spiliopoulos (2018) as for the Aiyagari model. Our implementation of the Sirignano and Spiliopoulos (2018) network portion of N uses 3 layers with 100 neurons per layer. For the embedding of the distribution $\hat{g}$ in a space of lower dimension, we use a feed-forward network with 2 hidden 128-neuron layers and a 10-neuron output layer. The additional factor $(a_0 + a)^{-\tilde{\eta}}$ is motivated by the hyperbolic shape of the marginal value function. Its inclusion is not strictly necessary but helps improving the overall accuracy of the approximation for a given neural network size. Unfortunately, we also found that this additional factor can make training progress less stable. To balance stability and accuracy considerations, we start out with $\tilde{\eta} = 0$ and train the network until the training loss falls below 5×10^{-4}. We then gradually increase $\tilde{\eta}$ towards a target value of 0.5. We keep $a_0 = 10$ fixed throughout.

Sampling Approach (Step 3a): We sample points of the form $(a, n, \hat{g}, z)$. As for the Aiyagari model, we sample (a, n) based on a uniform distribution. We also sample the aggregate variable z uniformly from the interval $[z_{min}, z_{max}]$. For $\hat{g}$, we use a mixture of two sampling

[7]Theoretically, it is possible that z_t becomes lower than z_{min} or greater than z_{max}. In practice, letting $|z_{min/max}| = 4\sigma$ is good enough to approximate the economic relevant region. We cannot replace the step of having $r \in [r_{lb}, r_{rb}]$ by having $z \in [z_{min}, z_{max}]$ to let the distribution move around because z's scaling effect is weaker. In fact z's impact on interest rate is $\Delta r \approx (r + \delta_K)\Delta z$, which can hardly let the population distribution move around. See more discussions in Section 7.

approaches: (a) the sampling approach for the Aiyagari model and (b) ergodic sampling based on the current approximation for W. We increase the proportion of the $\hat{g}$-sample that uses approach (b) gradually from 0% to 90% during training.

Loss Function Specification (Step 3b): The implementation details are as for the Aiyagari model, except that we also include a penalty for having $\partial_Z W > 0$ exactly as for the finite agent algorithm.

5.4 Numerical Solution

We solve the Krusell-Smith model using the finite agent method. The error in the master equation is shown in Table 2 below.

	Master equation loss
Finite Agent NN	3.037×10^{-5}
Discrete State Space NN	9.639×10^{-5}

Table 2: Neural Networks' results for solving master equations with aggregate shocks.

Unlike for the Aiyagari model, we do not have a clear benchmark for Krusell-Smith model because there is no existing technique that provides an accurate solution to the model with aggregate shocks. However, we can compare to widely used approximation techniques in the literature. In particular, we compare to the approach suggested by Fernández-Villaverde et al. (2018), which uses a neural network to approximate a statistical law of motion rather than developing the fully global solution.

We compare to Fernández-Villaverde et al. (2018) by computing sample paths from both our solution approaches. We describe how we generate sample paths for the neural network solution in Algorithm 3 below. Essentially, we draw a series of productivity shocks from the Ornstein-Uhlenbeck process: $dz_t = \eta(\overline{z} - z)dt + \sigma dB_t^0$ and then evolve the population distribution by adding z_t to Algorithm 2 for the ABH model described in Section 4.4.2.

Figure 3 shows the comparison between our neural network solution and Fernández-Villaverde et al. (2018) for a particular path of productivity shocks. The upper-left panel shows the draw from the Ornstein-Uhlenbeck process: $dz_t = \eta(\overline{z} - z)dt + \sigma dB_t^0$. The upper left compares the evolution of capital stock. The middle plots compares the evolution of prices. The bottom plots compare the evolution of the population. As can be seen in Figure 3, we get a similar path for aggregate capital stock, interest rates, wage rates, and the population distribution.

In Figure 4, we generate multiple random TFP paths, z_t, and show the evolution of our Neural Network solution and solution in Fernández-Villaverde et al. (2018) in a "fan chart" that displays percentiles for the evolution of the population. In particular, we generate 1000

Algorithm 3: Finding Transition Path by Neural Network: Krusell Smith case

```
Input  : Number of agents $N$, time step size $\Delta t$, number of time steps $N_T$, number
         of simulations $N_{sim}$
Output: A transition path $g = \{g_t : t = 0, \Delta t, \dots, N_T \Delta t\}$
for $n = 0, \dots, N_T - 1$ do
    for $k = 1, \dots, N_{sim}$ do
        Sample $\Delta B_t^0$ from normal distribution $N(0, \Delta t)$, construct TFP shock path
          by: $z_{t+\Delta t} = z_t + \eta(\bar{z} - z_t) + \sigma \Delta B_t^0$.
        Sample $N$ agents $\{s_i : i = 1, \dots, N\}$ from distribution $g_t$ at time $t = n\Delta t$.
        Given other agents' state $s_{-i}$, calculate the consumption $c(a, y, s_{-i}, z_{t+\Delta t})$,
          equilibrium capital return $r(s_{-i}, z_{t+\Delta t})$ and wage $w(s_{-i}, z_{t+\Delta t})$. Then
          construct $\mathcal{A}$ by finite difference scheme.
    end
    Take the average: $\bar{\mathcal{A}} = \frac{1}{N_{sim}} \sum_{k=1}^{N_{sim}} \mathcal{A}_k$
    Update $g_t$ by implicit method: $g_{t+\Delta t} = (I - \bar{\mathcal{A}}^\top \Delta t)^{-1} g_t$
end
```

TFP paths starting from $z_0 = 0$ and calculate the corresponding aggregate capital evolution paths. We compute capital at different time t, sort to get the pth-quantile and plot the time series of the quantiles.

6 Calibration

In this section, we discuss how to use our algorithm to calibrate models. An additional benefit of deep learning algorithms is that we can include the parameters, ζ, as additional inputs into the neural network:

$$V(\hat{X}, \zeta) \approx \hat{V}(\hat{X}, \zeta; \theta)$$

We can then train the neural network using sampling from both $\hat{X}$ and ζ. This means that we can train the model once and get a solution across a range of parameter values. We can then use $\hat{V}$ to calculate the moments for different parameters and calibrate the model.

We illustrate this technique in our model by calibrating the Krusell and Smith (1998) model to match a stochastic steady state capital-to-labor ratio of 5.0. We do this by including the borrowing constraint $\underline{a}$ as an input into the neural network, training the model randomly sampling $\underline{a}$ from $[-0, 1.5]$, and then using the trained model to solve for $\underline{a}$ that generates a stochastic steady state capital-to-labor ratio of 5.0. We show the results in Table 3.

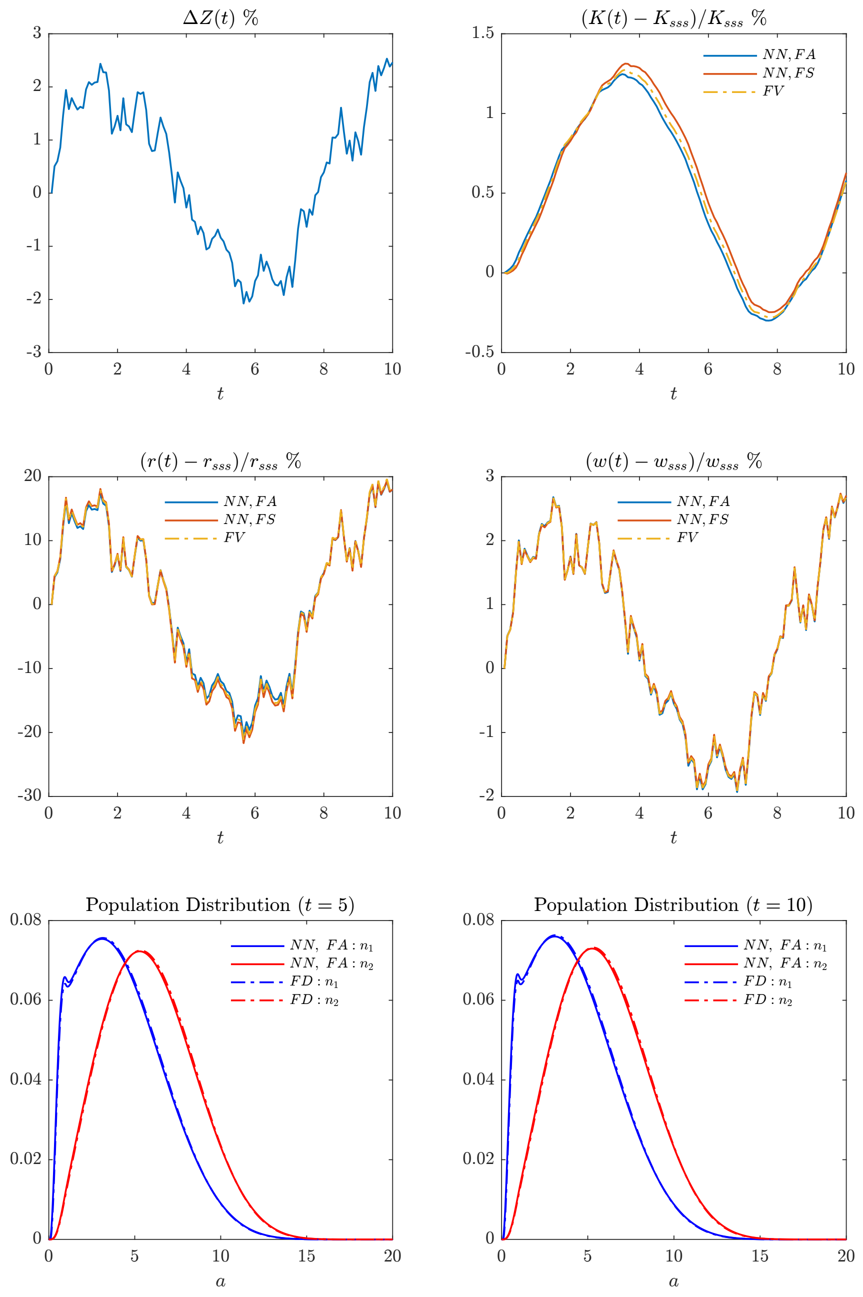


Figure 3: Impulse response functions for Krusell-Smith Model. The top left plot is the TFP shock path, the top right panel is the aggregate relative capital change, the middle left panel plots the relative average consumption change, and the middle right panel plots the relative capital return change. The bottom left is relative wage change, and the bottom right is the relative wealth change at different quantiles. *NN, FA* refers to the finite agent neural network, *NN, FV* refers to the finite state space neural network, and *FV* refers to the result generated from Fernández-Villaverde et al. (2018). Subscript *sss* refers to the stochastic steady state at $z = 0$.

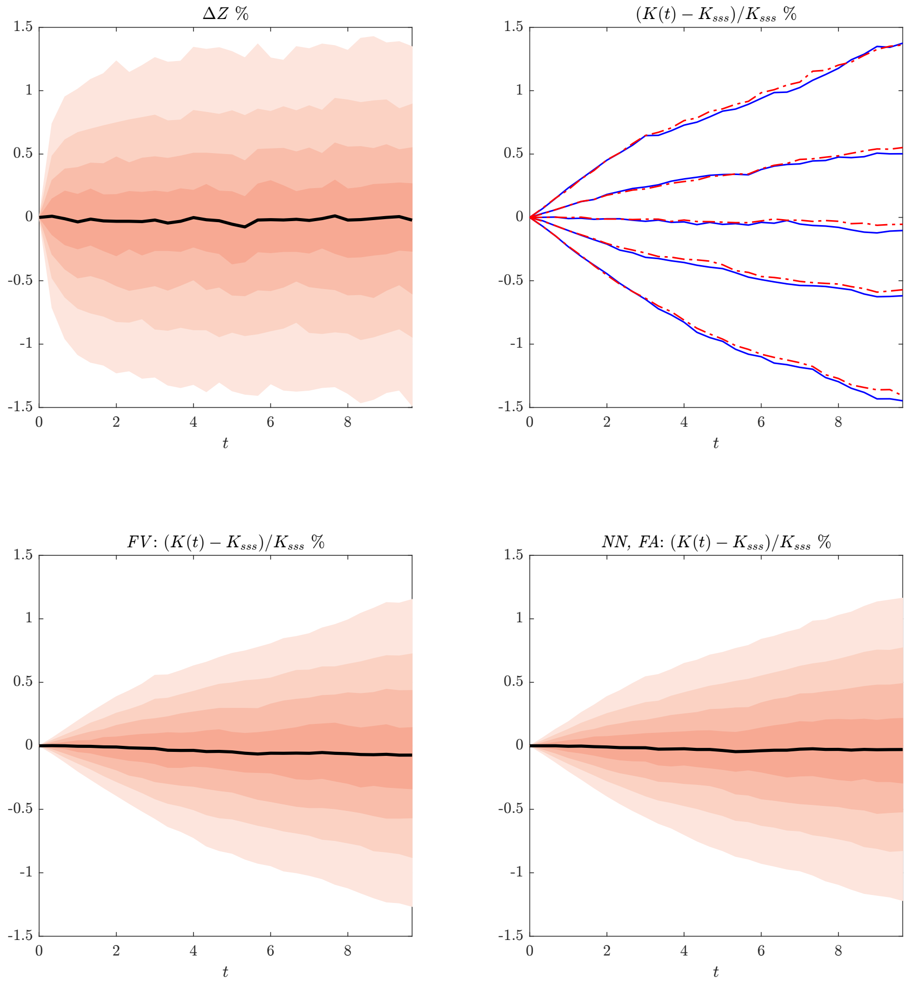


Figure 4: Forcasted aggregate capital dynamics starting from the stochastic steady state (sss) for the Krusell-Smith Model. The top left plot is the fan chart for the TFP shock path, generated from OU process with initial condition $z_0 = 0$. The bottom left panel and right panel are fan charts (capital quantile) of corresponding responses. The top right panel is the time series plot for relative change in aggregate capital at quantile 10%, 30%, 50%, 70%, 90% (from the lowest to the highest), in which the blue solid lines are generated by neural network solution and the red dashed lines are generated by Fernández-Villaverde et al. (2018). *NN, FA* referes to the finite agent technique, and *FV* refers to Fernández-Villaverde et al. (2018)'s technique.

K/L		
Target	Model	$\underline{a}$
5.0	5.0	1.082

Table 3: Neural Networks' results for Calibration

7 Practical Lessons

Although the deep learning algorithm is straightforward to describe, we find that implementing it successfully can be tricky. In this section, we collect some lessons that we believe are helpful for using deep learning to solve macroeconomic models.

Lesson 1: Working out the correct sampling approach is very important. A key feature of continuous time methods is that we must specify where in the state space to sample. This is very different from discrete time approaches which typically simulate the economy and so implicitly sample from the ergodic distribution. Like Gopalakrishna (2021), we find that choosing where to sample is both an advantage (because we can focus sampling in interesting, rarely visited regions of the state space with complicated curvature) but also a difficulty (because it can be hard to know where to sample). Many of our initial problems were resolved by adjusting how we were sampling the finite dimensional approximation to the distribution. In particular, we found it is essential to have significant variation in distribution so that the neural network can learn where there is curvature in the problem.

Lesson 2: Neural networks have difficulty dealing with inequality constraints. The Achdou et al. (2022) formulation of the Aiyagari (1994) model is written with a hard lower boundary that $a_t \geq \underline{a}$, which leads to an inequality boundary condition at $\underline{a}$ and a mass point in the distribution at $\underline{a}$. This is convenient for the finite difference solution technique because the inequality constraint does not impact the tractability of the upwind scheme. However, it causes problems for the neural network approach because the inequality constraint is too "easy" to satisfy, in the sense that many levels of the value function satisfy the constraint. Placing a low weight on the lower boundary error leads to solutions with the correct curvature but the wrong level. Placing a high weight on the lower boundary error leads to solutions with the right level but inaccurate curvature. So, there seems to only be a very small subset of weights that lead to accurate results for inequality boundary conditions. Ultimately, we find that replacing the hard constraint by a soft constraint and utility penalty makes the method much more robust.

Lesson 3: Enforcing shape constraints is very important. We find that the neural network is likely to converge to "cheat solutions" that approximately solve the differential equation by setting derivatives to zero. There are number of ways to help with this: (i) imposing loss functions on the curvature of the value function to enforce the correct shape (e.g. penalizing non-monotonicity or non-concavity), (ii) pre-training the neural network to match an initial guess satisfying known properties of the value function, and (iii) sampling from a sufficiently large part of the state space that the neural network realizes there is curvature in all dimensions.

Lesson 4: Mean squared errors can be misleading. We found that even if mean squared training errors are in the order of 10^{-2} or 10^{-3}, the neural network can give policy rules that are inaccurate. This suggests that overfitting can be a problem for neural network solutions to PDEs and so choosing the right "cross validation" sample is important. It also suggests that the threshold for convergence for neural network solutions to continuous time models might need to be higher than for other techniques, such as finite difference methods.

Lesson 5: Start with a simple model to tune hyperparameters. A benefit and cost with using neural networks is that they are very flexible approximations. This means that a difficult and time consuming part of training a neural network model is finding appropriate hyperparameters. We find that it is helpful to start with a simple model that can be solved with finite difference (e.g. a version of the model without aggregate shocks) and then tune the hyperparameters to make sure that the neural network approximation closely matches the finite difference solution. We can then introduce aggregate shocks starting with a set of hyperparameters that are good at capturing many features of the distribution. In other words, we suggest learning on a simple version of the problem, for which the solution is already known and then learning on the more complicated problem.

8 Conclusion

This paper proposed a new global solution algorithm for continuous time heterogeneous agent economies with aggregate shocks. We demonstrated our algorithm by solving two canonical models in the macroeconomics literature: the Aiyagari (1994) model and the Krusell and Smith (1998) model. Using our method on the first model, which can be solved using classical techniques, allows us to find tune hyperparameters, which can then be used on more complex models. One advantage of our method is that solving the second model with aggregate shocks is not much more difficult than solving the first problem without aggregate shocks. However, this is only the beginning of what is possible with this technique. Future work can deploy this solution technique to solve high dimensional economic models with non-linearities and aggregate shocks.

References

Achdou, Y., Han, J., Lasry, J.-M., Lions, P.-L., and Moll, B. (2022). Income and wealth distribution in macroeconomics: A continuous-time approach. *The Review of Economic Studies*, 89(1):45–86.

Achdou, Y., Han, J., Lasry, J.-M., and Moll, B. (2017). Heterogeneous Agent Models with Aggregate shocks in Continuous Time.

Ahn, S., Kaplan, G., Moll, B., Winberry, T., and Wolf, C. (2018). When inequality matters for macro and macro matters for inequality. *NBER macroeconomics annual*, 32(1):1–75.

Aiyagari, S. R. (1994). Uninsured idiosyncratic risk and aggregate saving. *The Quarterly Journal of Economics*, 109(3):659–684.

Al-Aradi, A., Correia, A., Jardim, G., de Freitas Naiff, D., and Saporito, Y. (2022). Extensions of the deep Galerkin method. *Applied Mathematics and Computation*, 430:127287.

Auclert, A., Bardóczy, B., Rognlie, M., and Straub, L. (2021). Using the sequence-space Jacobian to solve and estimate heterogeneous-agent models. *Econometrica*, 89(5):2375–2408.

Azinovic, M., Gaegauf, L., and Scheidegger, S. (2022). Deep equilibrium nets. *International Economic Review*, 63(4):1471–1525.

Bayraktar, E., Budhiraja, A., and Cohen, A. (2018). A numerical scheme for a mean field game in some queueing systems based on Markov chain approximation method. *SIAM Journal on Control and Optimization*, 56(6):4017–4044.

Bensoussan, A., Frehse, J., and Yam, S. C. P. (2015). The master equation in mean field theory. *Journal de Mathématiques Pures et Appliquées*, 103(6):1441–1474.

Bertucci, C. and Cecchin, A. (2022). Mean field games master equations: from discrete to continuous state space. *arXiv preprint arXiv:2207.03191*.

Bhandari, A., Bourany, T., Evans, D., and Golosov, M. (2023). A perturbational approach for approximating heterogeneous-agent models.

Bilal, A. (2021). Solving heterogeneous agent models with the master equation. Technical report, University of Chicago.

Bretscher, L., Fernández-Villaverde, J., and Scheidegger, S. (2022). Ricardian business cycles. *Available at SSRN*.

Brzoza-Brzezina, M., Kolasa, M., and Makarski, K. (2015). A penalty function approach to occasionally binding credit constraints. *Economic Modelling*, 51:315–327.

Cardaliaguet, P., Delarue, F., Lasry, J.-M., and Lions, P.-L. (2015). The master equation and the convergence problem in mean field games. *arXiv*.

Carmona, R. and Laurière, M. (2021). Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games I: the ergodic case. *SIAM Journal on Numerical Analysis*, 59(3):1455–1485.

Carmona, R. and Laurière, M. (2022a). Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: II—the finite horizon case. *The Annals of Applied Probability*, 32(6):4065–4105.

Carmona, R. and Laurière, M. (2022b). Deep learning for mean field games and mean field control with applications to finance. *Machine Learning in Financial Markets: A guide to contemporary practises, editors: A. Capponi and C.-A. Lehalle, Cambridge University Press.*

Carmona, R., Laurière, M., and Tan, Z. (2019). Model-free mean-field reinforcement learning: mean-field MDP and mean-field Q-learning. *arXiv preprint arXiv:1910.12802*.

Delarue, F., Lacker, D., and Ramanan, K. (2020). From the master equation to mean field game limit theory: Large deviations and concentration of measure. *Annals of Probability*, 48(1):211–263.

Den Haan, W. (1997). Solving Dynamic Models with Aggregrate Shocks and Heterogeneous Agents. *Macroeconomic Dynamics*, 1(2):355–386.

Duarte, V. (2018). Machine learning for continuous-time economics. *Available at SSRN 3012602*.

Fernández-Villaverde, J., Hurtado, S., and Nuño, G. (2018). Financial Frictions and the Wealth Distribution. *Working Paper*, pages 1–51.

Fernandez-Villaverde, J., Nuno, G., Sorg-Langhans, G., and Vogler, M. (2020). Solving high-dimensional dynamic programming problems using deep learning. *Unpublished working paper*.

Fouque, J.-P. and Zhang, Z. (2020). Deep learning methods for mean field control problems with delay. *Frontiers in Applied Mathematics and Statistics*, 6:11.

Frikha, N., Germain, M., Laurière, M., Pham, H., and Song, X. (2023). Actor-critic learning for mean-field control in continuous time. *arXiv preprint arXiv:2303.06993*.

Germain, M., Laurière, M., Pham, H., and Warin, X. (2022a). DeepSets and their derivative networks for solving symmetric PDEs. *Journal of Scientific Computing*, 91(2):63.

Germain, M., Mikael, J., and Warin, X. (2022b). Numerical resolution of mckean-vlasov fbsdes using neural networks. *Methodology and Computing in Applied Probability*, pages 1–30.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gopalakrishna, G. (2021). Aliens and continuous time economies. *Swiss Finance Institute Research Paper*, (21-34).

Gu, H., Guo, X., Wei, X., and Xu, R. (2021). Mean-field controls with Q-learning for cooperative MARL: convergence and complexity analysis. *SIAM Journal on Mathematics of Data Science*, 3(4):1168–1196.

Hadikhanloo, S. and Silva, F. J. (2019). Finite mean field games: fictitious play and convergence to a first order continuous mean field game. *Journal de Mathématiques Pures et Appliquées*, 132:369–397.

Han, J., Jentzen, A., and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510.

Han, J., Yang, Y., and E, W. (2021). DeepHAM: A global solution method for heterogeneous agent models with aggregate shocks. *arXiv preprint arXiv:2112.14377*.

Hu, R. and Lauriere, M. (2022). Recent developments in machine learning methods for stochastic control and games. *ssrn.4096569*.

Huang, J. (2022). A probabilistic solution to high-dimensional continuous-time macro-finance models. *Available at SSRN 4122454*.

Kahou, M. E., Fernández-Villaverde, J., Perla, J., and Sood, A. (2021). Exploiting symmetry in high-dimensional dynamic programming. Technical report, National Bureau of Economic Research.

Krusell, P. and Smith, A. A. (1998). Income and Wealth Heterogeneity in the Macroeconomy. *Journal of Political Economy*, 106(5):867–896.

Lacker, D. (2020). On the convergence of closed-loop Nash equilibria to the mean field game limit. *Annals of applied probability: an official journal of the Institute of Mathematical Statistics*, 30(4):1693–1761.

Laurière, M. (2021). Numerical methods for mean field games and mean field type control. *Mean Field Games*, 78:221.

Li, J., Yue, J., Zhang, W., and Duan, W. (2022). The deep learning Galerkin method for the general stokes equations. *Journal of Scientific Computing*, 93(1):1–20.

Lions, P.-L. (2007-2011). Lectures at College de France.

Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. (2021a). DeepXDE: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., and Johnson, S. G. (2021b). Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132.

Maliar, L., Maliar, S., and Winant, P. (2021). Deep learning for solving dynamic economic models. *Journal of Monetary Economics*, 122:76–101.

Min, M. and Hu, R. (2021). Signatured deep fictitious play for mean field games with common noise. In *International Conference on Machine Learning*, pages 7736–7747. PMLR.

Perrin, S., Laurière, M., Pérolat, J., Élie, R., Geist, M., and Pietquin, O. (2022). Generalization in mean field games by learning master policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9413–9421.

Prohl, E. (2017). Discetizing the Infinite-Dimensional Space of Distributions to Approximate Markov Equilibria with Ex-Post Heterogeneity and Aggregate Risk.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561.*

Reiter, M. (2002). Recursive computation of heterogeneous agent models. *manuscript, Universitat Pompeu Fabra*, (July):25–27.

Reiter, M. (2008). Solving heterogeneous-agent models by projection and perturbation. *Journal of Economic Dynamics and Control*, 33:649–665 Contents.

Reiter, M. (2010). Approximate and Almost-Exact Aggregation in Dynamic Stochastic Heterogeneous-Agent Models. Technical report, Vienna Institute for Advanced Studies.

Sauzet, M. (2021). Projection methods via neural networks for continuous-time models. *Available at SSRN 3981838.*

Schaab, A. (2020). Micro and macro uncertainty. *Available at SSRN 4099000.*

Sirignano, J. and Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364.

Sznitman, A.-S. (1991). Topics in propagation of chaos. *Lecture notes in mathematics*, pages 165–251.

Winberry, T. (2018). A method for solving and estimating heterogeneous agent macro models. *Quantitative Economics*, 9(3):1123–1151.

A Additional Theoretical Results for Section 2

In this section of the appendix, we outline additional theoretical results relating to section 2 of the main text. We first outline a more general version of the economic environment for which our solution techniques are appropriate.

A.1 General Economic Model

In this section of the appendix, we outline the more general version of the economic model for which the techniques in this paper are appropriate. Relative to Section 2, we generalize the model among three dimensions. First, we allow the aggregate state z_t^i to be of arbitrary dimension and driven by both Brownian and Poisson noise. Second, we allow the idiosyncratic state x_t^i to be driven by more than one Poisson processes. Third, we allow the control c_t^i to be multi-dimensional and to affect also the diffusion and jump terms of x_t^i. These generalizations complicate the notational burden required to present the model but do not pose any additional conceptual challenges for our solution algorithms. We present here only the aspects of the model that change relative to Section 2.

Aggregate State Dynamics: There is an exogenous aggregate state vector, $z_t \in \mathcal{Z}$, of arbitrary (finite) dimension. z_t evolves according to:

$$dz_t = \mu^z(z_t)dt + \sigma^z(z_t)dB_t^0 + \gamma^z(z_t)dJ_t^0, \tag{A.1}$$

where B^0 denotes a common N^{B0}-dimensional Brownian motion process and J^0 denotes a common N^{J0}-dimensional Poisson jump process. The aggregate history $\mathcal{F}_t^0$ is defined as the filtration generated by B^0 and J^0. We let $\nu^n(z)$, $n \leq N^{J0}$, denote the rate at which the n-th component of the Poisson jump shock arrives given aggregate state z.

Idiosyncratic State Dynamics: Each agent, $i \in I$, has an idiosyncratic state vector, $x^i \in \mathcal{X}$, that evolves according to:

$$dx_t^i = \mu^x(c_t^i, x_t^i, z_t, q_t)dt + \sigma^x(c_t^i, x_t^i, z_t, q_t)dB_t^i + \gamma^x(c_t^i, x_t^i, z_t, q_t)dJ_t^i, \tag{A.2}$$

where $c_t^i \in \mathcal{C}(x_t^i, z_t, q_t)$ is a control vector chosen by the agent, $q_t \in \mathcal{Q}$ is a collection of aggregate prices in the economy, B_t^i denotes an idiosyncratic N^{Bi}-dimensional Brownian motion process, and J^i denotes an idiosyncratic N^{Ji}-dimensional Poisson jump process. We let $\lambda^n(x, z)$, $n \leq N^{Ji}$, denote the rate at which the n-th component of the Poisson jump shock arrives given idiosyncratic state x and aggregate state z.

Agent Problem: Let $u(c_t^i)$ denote the flow utility from choosing the state vector c_t^i. The agent problem is in complete analogy to Section 2. Given their belief about prices q, agent

i chooses their control process, c^i, to solve:

$$V(x_0^i, z_0) = \max_{c^i \in \mathcal{C}} \mathbb{E}_0 \left[\int_0^\infty e^{-\rho t} u(c_t^i) dt \right] \tag{A.3}$$
$$s.t. \quad \text{(A.1)}, \ \text{(A.2)}.$$

Equilibrium: The definition of equilibrium is word by word identical to the definition given in Section 2 in the main text.

KFE: Because dB_t^0 and dJ_t^0 do not directly enter the idiosyncratic state evolution (A.2), the KFE is precisely as in the main text, equation (2.6).

Master Equation: The master equation can again be written in the form

$$(\mathcal{L}V)(x, z, g) = 0$$

with a differential operator $(\mathcal{L}V)(x, z, g) := (\mathcal{L}^h V + \mathcal{L}^g V)(x, z, g)$.

The distribution portion $\mathcal{L}^g V$ of this operator takes precisely the same form as in the main text. It is for this reason that the numerical solution approach readily generalizes to the model version presented here.

The optimization problem portion $\mathcal{L}^h V$ of the differential operator is structurally similar to the main text but requires slightly heavier notation:

$$\begin{aligned}
(\mathcal{L}^h V)(x, z, g) := & -\rho V(x, z, g) + u(c^*(x, z, g)) \\
& + D_x V(x, z, g) \mu^x(c^*(x, z, g), x, z, Q(z, \bar{g})) + D_z V(x, z, g) \mu^z(z) \\
& + \frac{1}{2} \operatorname{tr} \left\{ \Sigma^x(c^*(x, z, g), x, z, Q(z, \bar{g})) D_x^2 V(x, z, g) \right\} + \frac{1}{2} \operatorname{tr} \left\{ \Sigma^z(z) D_z^2 V(x, z, g) \right\} \\
& + \sum_{n \leq N^{Ji}} \lambda^n(x, z) \left(V(x + \gamma^{xn}(c^*(x, z, g), x, z, Q(z, \bar{g})), z, g) - V(x, z, g) \right) \\
& + \sum_{n \leq N^{J0}} \nu^n(z) \left(V(x, z + \gamma^{zn}(z), g) - V(x, z, g) \right)
\end{aligned}$$

Here, the notation is in analogy to the main text: $\Sigma^x(\cdot) = \sigma^x(\cdot) \left(\sigma^x(\cdot)\right)^\top$, $\Sigma^z(\cdot) = \sigma^z(\cdot) \left(\sigma^z(\cdot)\right)^\top$, and $c^*(x, z, g)$ denotes the (recursive) optimal control that solves the agent problem (A.3).

B Aiyagari Model

B.1 Parameters for Aiyagari Model

Parameter	Symbol	Value
Capital share	α	1/3
Depreciation	δ	0.1
Risk aversion	γ	2.1
Discount rate	ρ	0.05
Mean TFP	$\overline{Z}$	0.00
Reversion rate	η	0.50
Volatility of TFP	σ	0.01
Transition rate (1 to 2)	λ_1	0.4
Transition rate (2 to 1)	λ_2	0.4
Low labor productivity	n_1	0.3
High labor productivity	n_2	$1 + \lambda_2/\lambda_1(1 - n_1)$
Borrowing constraint	$\underline{a}$	10^{-6}
Maximum of asset	$\overline{a}$	20.0
Penalty Function	$\psi(a)$	$-\frac{1}{2}\kappa(a - a_{lb})^2$
Penalty parameters	a_{lb}	1.0
Penalty parameters	κ	3.0

B.2 Penalty Function Approximation

In this section, we compare the finite difference solutions with the hard and soft boundary constraints. The comparisons are summarized in Figure 5.

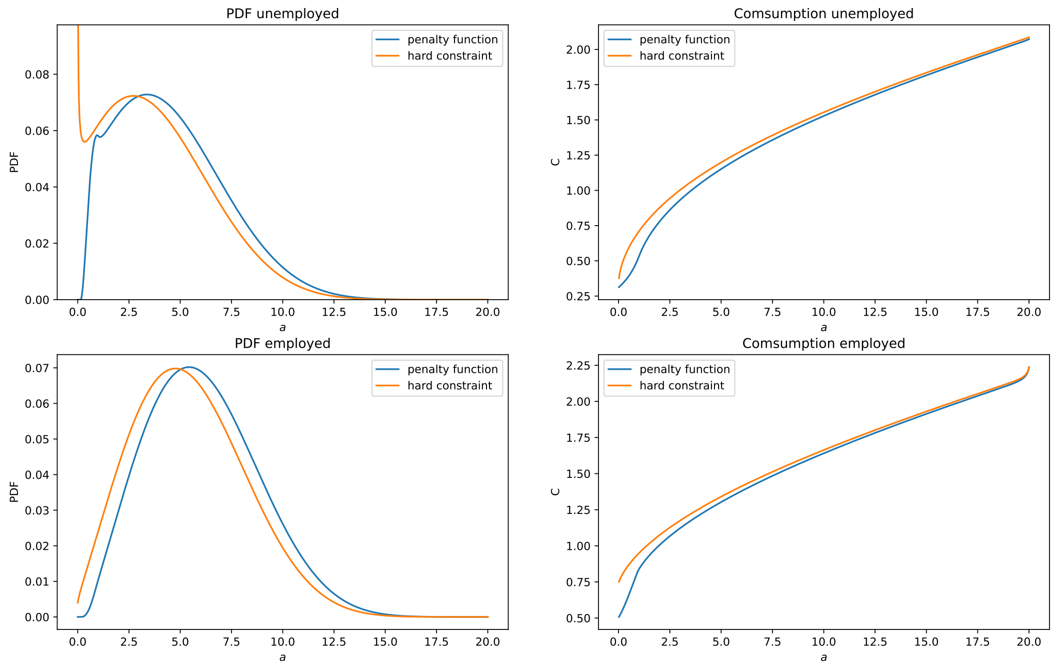


Figure 5: Comparison between penalty function approach and hard constraint in finite different exercise. $\psi(a) = -\frac{1}{2}\kappa(a - a_{lb})^2$, $a_{lb} = 1.0$, $\kappa = 3.0$.

B.3 Comparison of Techniques for Calculating Transition Dynamics without Aggregate Shock

After we obtained the neural network approximation of the value function in the master equation, $V(a_i, n_i, s_{-i})$, we are able to consider the transition path in two ways. The first way is to consider an evolving finite agent economy, the initial asset and employment status are sampled from the initial distribution: $g_0(a, n)$. This method gives the transition path of return on capital/labor directly. To solve the Kolmogorov Forward Equation, the key is to figure out $\mu(a, n) = ra + wn - c$, which can be approximated by the consumption policy. The algorithm can be summarized as follows in a high level.

Algorithm 4: Finding Transition Path by Neural Network

```
1. Sample $N$ agents from the initial condition $g_0(a, n)$.
2. Calculate the consumption, then find the transition path for every asset position.
3. After we've obtained $c_t, a_t, r_t$, solve $g_t(a, n)$ by finite difference of the forward
   equation
```

The second way is to update the distribution by Kolmogorov Forward Equation and resample agents from the updated distribution accordingly. Recall the forward equation in this economy:

$$\frac{\partial g_t(a,n)}{\partial t} = -\frac{\partial(\mu(a,n)g(a,n))}{\partial a} - \lambda_n g(a,n) + \lambda_{\check{n}} g(a,\check{n}).$$

In the this finite agents economy, the dynamic of equilibrium return on capital does not require another "guess-verify" loop, as in Achdou et al. (2022). The following algorithm modified to solve the dynamics of Aiyagari Economy.

Algorithm 5: Finding Transition Path by Finite Difference

```
1. Guess the path of equilibrium interest rate $r^o(t)$, then solve HJB, with terminal condition: $v(a,n,T) = \bar{v}(a,n)$
2. Solve the policy function $c_t(a,n)$.
3. Solve the forward equation, with initial condition $g_0(a,n)$.
4. Calculate the capital held by the whole economy: $\int_{\underline{a}}^{\infty} \sum_{n\in\{0,1\}} a g_t(a,n) = K_t$, then calculate the implied interest rate by $r^n(t) = \partial_K F(K_t, L) - \delta$, for every $t \in [0,T]$.
5. Update the path to be $\lambda r^o(t) + (1-\lambda) r^n(t)$, repeat 1-4 until $||r^o - r^n||_\infty < \epsilon$.
```

Lessons. When using iterative method to find steady state/distributional dynamics in the forward equation, we have:

$$g = (I - \mathcal{A}(g)dt)^{-1} g,$$

which means we essentially have a high dimensional *fixed-point* problem. As we are using simulation to find $\mathcal{A}$, we know $\mathcal{A}$ may be very kinky because of the sampling error, and the solution is to have N_{sim} very large to smooth $(I - \bar{\mathcal{A}}dt)^{-1}$. Otherwise, the distribution g can be trapped before it reaches the steady state, as shown in Figure 6. Initial distribution with total capital $K_{init} < K_{ss,new}$ will end up with a oscillating, but a steady state with slightly lower aggregate capital. Similarly, initial distribution with total capital $K_{init} > K_{ss,new}$ will end up with a oscillating, but a steady state with a slightly higher aggregate capital.

B.4 Comparison of neural networks with and without symmetry

- **Speed**: neural networks with symmetry is 10 times slower than without symmetry.
- **Convergence**: neural networks with symmetry has converged (without batch sampling), but it is extremely hard to converge without symmetry
- **Shape**: nn with symmetry has problem capturing the shape, nn without symmetry has problem capturing convexity for KS calibration.

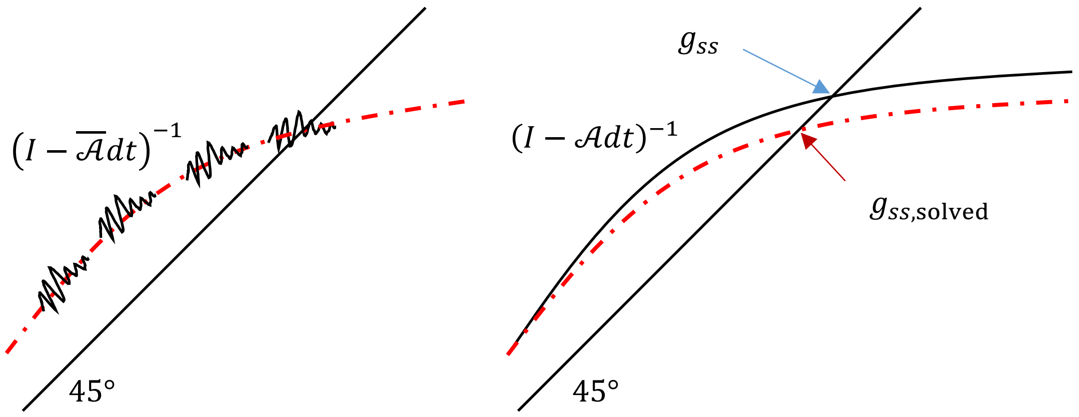


Figure 6: Fixed point problem in finding steady state and distributional dynamics. The fixed point is solved by the cross between $(I - \mathcal{A}dt)^{-1}$ and 45° line. The kinky black line on the left panel is the simulated operator $(I - \bar{\mathcal{A}}dt)^{-1}$, and the red dashed line is the smoothed approximation of $(I - \bar{\mathcal{A}}dt)^{-1}$. The concave black solid line on the right panel is the theoretical $(I - \bar{\mathcal{A}}dt)^{-1}$. g_{ss} and $g_{ss,\text{solved}}$ are the crossing points between the black solid line and 45° line, the red dashed line and 45° line, respectively.

B.5 Active Learning

To efficiently train our neural net, we use the active learning method to track where losses are larger. We start from a uniform sampler in domain $[\underline{a}, \overline{a}]$. To capture the curvature where penalty function is applied, we add additional points, which is also a hyper parameter can be tuned, to the interval $[\underline{a}, a_{lb}]$. It is crucial to have a better approximation of policies in this subdomain to avoid wrong solutions. In addition, we dynamically adjust the sampler's distribution according to reported losses, similar to Gopalakrishna (2021). We increase the probability density for the subdomains where the loss are larger, and vice versa.

In practice, we partition the domain into N subdomains as in section 2.8 of Lu et al. (2021a). After approximately 2,000 epochs of training, we start to implement active learning. We calculate the master equation's relative residual in each subdomain, and re-weight sampler's distribution accordingly. We find it's better to also increase density to the subdomains close to where the relative residual is the largest. The loss has a steeper drop after we start active learning in the 2000th epoch, as shown in Figure 8.

C Krusell-Smith Model

C.1 Parameters for Krusell-Smith Model

Parameter	Symbol	Value
Capital share	α	1/3
Depreciation	δ	0.1
Risk aversion	γ	2.1
Discount rate	ρ	0.05
Mean TFP	$\overline{Z}$	0.00
Reversion rate	η	0.50
Volatility of TFP	σ	0.01
Transition rate (1 to 2)	λ_1	0.4
Transition rate (2 to 1)	λ_2	0.4
Low labor productivity	n_1	0.3
High labor productivity	n_2	$1 + \lambda_2/\lambda_1(1 - n_1)$
Borrowing constraint	$\underline{a}$	10^{-6}
Maximum of asset	$\overline{a}$	20.0
Penalty Function	$\psi(a)$	$-\frac{1}{2}\kappa(a - a_{lb})^2$
Penalty parameters	a_{lb}	1.0
Penalty parameters	κ	3.0
Drift in O-U Process	η	0.5
Volatility in O-U Process	σ	0.01
Maximum TFP	z_{max}	0.04
Minimum TFP	z_{min}	-0.04

D Additional Plots

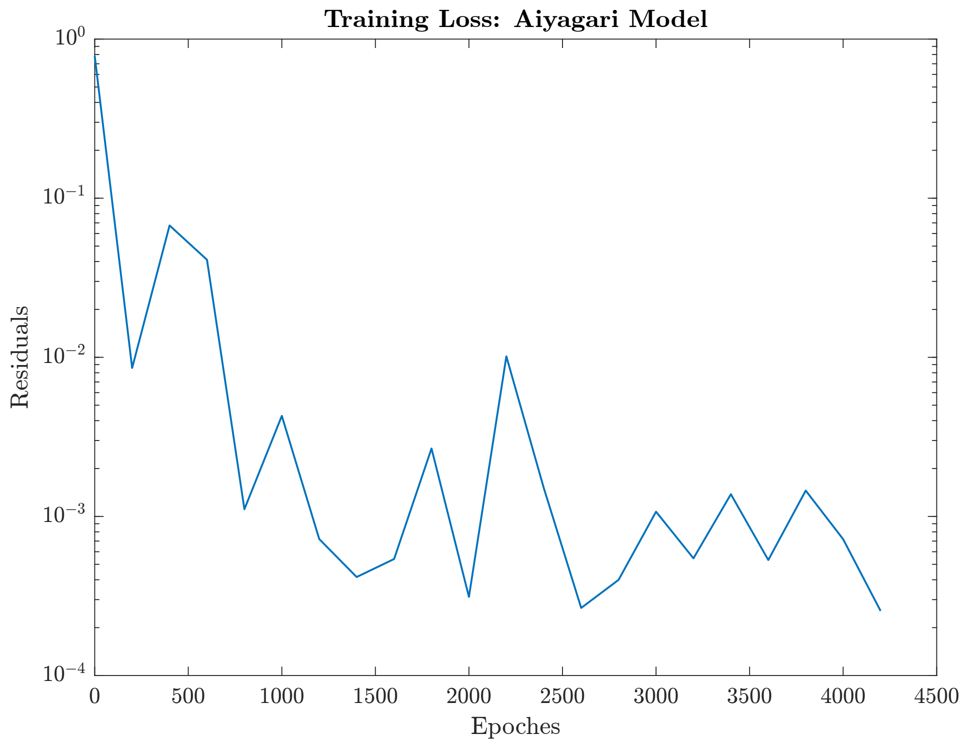
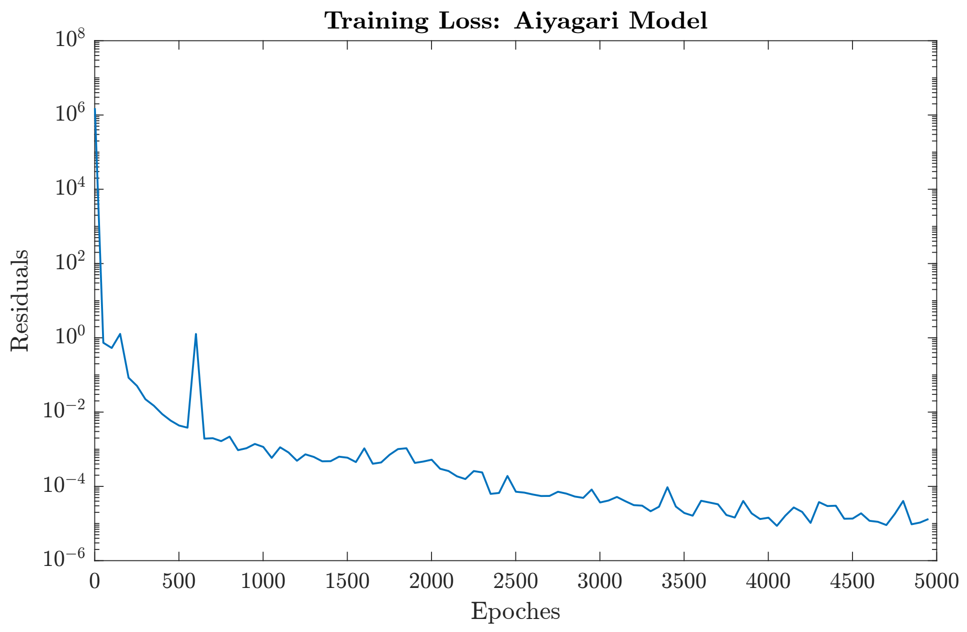


Figure 7: Training Loss vs Interation Plots (Discrete State Space Method)



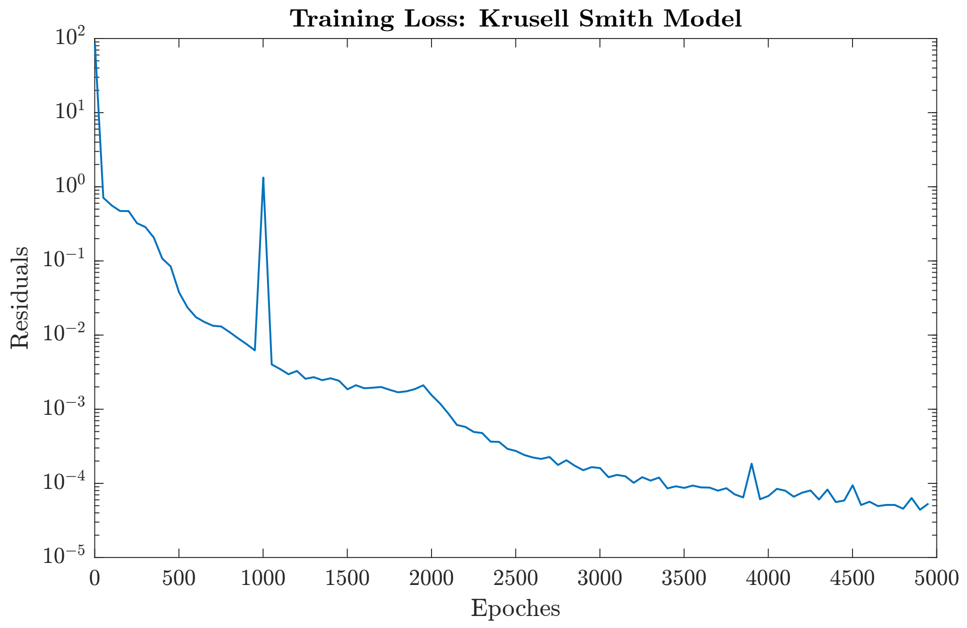


Figure 8: Training Loss vs Interation Plots (Finite Agent Method)

E Projection technique: General case

In this section, we describe a general projection technique to approximate the distribution using a finite-dimensional vector. The discrete state space approximation method described in subsection 3.1.2 is an implementation of the generic projection method described below, in which the eigenfunctions are the canonical basis functions for the discrete state space.

E.1 General description of the technique

The main idea of the projection method as we discuss it below is to represent the state distribution g by the coefficients of its projection on a set of suitable basis functions. Even though the exact representation of $\{g_t : t \geq 0\}$ might require an infinity of basis functions, we would like that just a few basis functions are enough to provide a good approximation. We are going to relate this idea to the eigenfunctions of the differential operator characterizing the KFE (2.6) at equilibrium, i.e., with $\tilde{\mu}^g = \mu^g$. We can rewrite this equation as follows:

$$dg_t(x) = (\mathcal{L}_t^g g_t)(x)dt, \tag{E.1}$$

where, for generic distribution g and point x,

$$(\mathcal{L}_t^g g)(x) := \mu^g(c^*(x, z_t, g), x, z_t, g).$$

Notice that $\mathcal{L}_t^g$ depends on time and is stochastic due to the implicit dependence on z_t.

Consider a fixed value of z_t, say $\bar{z}$, and let:

$$(\mathcal{L}^{ss} g)(x) := \mu^g(c^*(x, \bar{z}, g), x, \bar{z}, g).$$

Let us consider the eigenfunctions $\{g_i : i \geq 0\}$ of $\mathcal{L}^{ss}$ with corresponding eigenvalues denoted by $\{\lambda_i \in \mathbb{R} : i \geq 0\}$. They satisfy:

$$\mathcal{L}^{ss} g_i = \lambda_i g_i, \qquad i \geq 0.$$

We must have $\lambda_i < 0$, so that the derivative with respect to time decreases to 0. Furthermore, the more λ_i is negative, the faster the time derivative goes to 0. As such, if we were to decompose the stationary distribution g^{ss} satisfying $(\mathcal{L}^{ss} g^{ss})(x) = 0$ using a finite number of eigenfunctions of g_i^{ss}, we would probably give more importance to the ones corresponding to eigenvalues that are (negative but) close to 0. Indeed, these corresponds to eigenfunctions for which the decay is slower. Up to relabelling the eigenfunctions and eigenvalues, we thus assume for ease of presentation that the eigenvalues are in decreasing order: $0 > \lambda_1 > \lambda_2 > \dots$. The eigenfunctions $\{g_i : i \geq 0\}$ form a basis, so there exists a unique collection of coefficients $\{\alpha_i : i \geq 0\}$ such that we can decompose the stationary distribution as:

$$g^{ss}(x) = \sum_{i=1,2,\ldots} \alpha_i g_i(x).$$

Using this basis of functions, we can decompose the solution $\{g_t : t \geq 0\}$ to (E.1) at every t as:

$$g_t(x) = \sum_{i=1,2,\ldots} \alpha_i(t) g_i(x),$$

where the coefficients $\alpha_i(t)$ are real-valued random variables.

Let $\mathcal{H}_t := \mathcal{L}_t - \mathcal{L}^{ss}$, i.e.: for any distribution g,

$$\mathcal{H}_t g = (\mathcal{L}_t^g - \mathcal{L}^{ss})g.$$

This operator is time-dependent and stochastic due to the presence of z_t.

We can now write:

$$\partial_t g_t(x) = \mathcal{L}_t^g g_t(x) = \left(\mathcal{L}^{ss} + \mathcal{H}_t\right) g_t(x) \tag{E.2}$$

In the left-hand side, we have:

$$\partial_t g_t(x) = \sum_{i=1}^{\infty} \frac{d\alpha_i(t)}{dt} g_i(x),$$

while in the right-hand side, we have:

$$\begin{aligned}
\left(\mathcal{L}^{ss} + \mathcal{H}_t\right) g_t(x) &= \sum_{i=1}^{\infty} \left(\mathcal{L}^{ss}(\alpha_i(t) g_i(t,x))\right) + \mathcal{H}_t g_t(x) \\
&= \sum_{i=1}^{\infty} \alpha_i(t)\lambda_i g_i(x) + \left(\mathcal{H}_t \left(\sum_{i=1}^{\infty} \alpha_i(t) g_i\right)\right)(x).
\end{aligned}$$

Notice that, since $\mathcal{H}_t$ is not a linear operator, we cannot take the sum out of the operator. However, we expect that truncating the sum will provide a good approximation of the infinite sum if there are only a few "significant" eigenvalues.

Going back to (E.2), multiplying both sides by g_j for a given index j, and integrating of the state space yields,

$$\sum_i \frac{d\alpha_i(t)}{dt}\langle g_i, g_j\rangle = \sum_i \alpha_i(t)\lambda_i \langle g_i, g_j\rangle + \int g_j(x)\left(\mathcal{H}_t\left(\sum_{i=1}^{\infty} \alpha_i(t) g_i\right)\right)(x)dx, \tag{E.3}$$

where $\langle g_i, g_j\rangle = \int g_i(x) g_j(x) dx$, which is equal to 1 if $i = j$ (and otherwise it is not necessarily

0). From here, we want to fix a number of basis functions, say N_b, and use the approximation:

$$\sum_{i=1}^{\infty} \alpha_i(t) g_i \approx \sum_{i=1}^{N_b} \alpha_i(t) g_i.$$

E.2 Approximation using finite differences

We now explain how to implement this generic projection technique using a discretization of the state space.

We first replace the basis functions $\{g_i : i = 1, \dots, N_b\}$ by d-dimensional vectors approximating their values on a grid of points $x_k = x_{lb} + k\Delta x$: $\hat{g}_{i,k} \approx g_i(x_k)$, such that $\sum_{k,\ell} \hat{g}_{i,k} \Delta x = 1$. Here the state space is discretized using $K+1$ points $\{x_k : k = 0, 1, \dots, K\}$ with $x_0 = x_{lb}$ and $x_K = x_{ub}$.

Then:

- $\sum_{i=1}^{N_b} \alpha_i(t) \hat{g}_i$ is a d-dimensional vector approximating a probability; note that its coefficients does not necessarily sum to 1 because we have truncated the infinite sum;
- We replace $\mathcal{H}_t$ by a finite-difference operator $\widehat{\mathcal{H}}_t$; this is specific to the problem but one can use for example a finite difference scheme such as Achdou et al. (2022) for the ABH model;
- The term $\int g_j(x) \left(\mathcal{H}_t \left(\sum_{i=1}^{\infty} \alpha_i(t) g_i \right) \right)(x) dx$ is replaced by:

$$\sum_{k=1}^{K} \hat{g}_{j,k} \left(\widehat{\mathcal{H}}_t \left(\sum_{i=1}^{N_b} \alpha_i(t) \hat{g}_i \right) \right)_k \Delta x$$

- Then (E.3) rewrites:

$$\sum_i \frac{d\alpha_i(t)}{dt} \hat{g}_i \cdot \hat{g}_j = \sum_i \alpha_i(t) \lambda_i \hat{g}_i \cdot \hat{g}_j + \sum_{k=1}^{K} \hat{g}_{j,k} \left(\widehat{\mathcal{H}}_t \left(\sum_{i=1}^{N_b} \alpha_i(t) \hat{g}_i \right) \right)_k \Delta x$$

  for $j = 1, \dots, N_b$ and $t \geq 0$, which can be computed.