Exploiting Symmetry in High-Dimensional Dynamic Programming

Mahdi Ebrahimi Kahou[1] Jesús Fernández-Villaverde[2] Jesse Perla[1] Arnav Sood[3]

August 10, 2021

[1]University of British Columbia, Vancouver School of Economics

[2]University of Pennsylvania

[3]Carnegie Mellon University

Motivation

- Most models in macro (and other fields) deal with either:
  1. Representative agent (canonical RBC and New Keynesian models). Sometimes two agents (models of financial frictions, international business cycles).
  2. A continuum of agents (canonical Krusell-Smith model).
- However, many models of interest deal with a finite (but large) number of agents and aggregate uncertainty:
  1. Models with many locations (countries, regions, metropolitan areas, industries).
  2. Models with many households (e.g., overlapping generations, different types) instead of a continuum.
  3. Models of industry dynamics with many firms.
  4. Models of networks.
- Models with finite agents are increasingly popular as we accumulate more micro data.

Challenge

- In (most) models with a finite number of agents and aggregate shocks, agents need to keep track of their own states and the states of everyone else.
- Think about an N-locations business cycle model à la Backus, Kehoe, and Kydland (1992): the agent in location n needs to know her own 8 states and the 8 states in each of the other $N - 1$ locations.
- With $N = 50$ (U.S. states), number of state variables is 400.
- How do you solve a model with 400 state variables?
  1. What about perturbation à la Judd and Guu (1993)? Problems are often inherently nonlinear (e.g., occasionally binding constraints) or non-ergodic (e.g., no steady states, large transitional dynamics).
  2. What about sparse grids? Even the most aggressive approaches à la Brumm and Scheidegger (2017) cannot be pushed beyond 30 state variables in most real life applications.
  3. What about something that looks like Krusell and Smith (1998)? Wait for it!

Curse of dimensionality

Two components (Bellman, 1958, p. IX):

1. The cardinality of the state space is enormous: memory requirements, update of coefficients, ...
   - With 266 state variables, a grid of 2 points per state (low and high) or, equivalently, a tensor of 2 polynomials per state (a level and a slope) have more elements than the Eddington number, the estimated number of protons ($\approx 10^{80}$) in the universe.
2. It is difficult to evaluate highly-dimensional conditional expectations: continuation value function, Euler equations, ...
   - Computing integrals is an exponentially-hard function of their dimensions.

What do we do?

- We introduce permutation-invariant dynamic programming and the associated concept of permutation-invariant economies.
- Concepts are built around the idea of symmetry. Old tradition in economics that goes back decades (Samuelson, 1990, Mertens and Judd, 2018); let me skip the literature review.
- Common feature of many (most?) models of interest: the policy functions of the agents are the same, they are just evaluated at different points.
- In a multi-location model of the U.S.: if (the representative agent in) California had the same capital and productivity as Texas, it would behave as Texas and vice versa.
- Many forms of *ex-ante* heterogeneity are encompassed as pseudo-states (more on this later on).
- The solution of the model is invariant under all the permutations of other agents' states. In equilibrium, the Walrasian auctioneer removes indexes!

Why does permutation invariance help?

Permutation invariance tackles the two components of the "curse of dimensionality":

1. The value and policy functions belong to the family of permutation-invariant functions $\Rightarrow$ they can be represented (exactly!) using a latent dimension (possibly much lower than the number of states).
   - It helps to understand why the Krusell-Smith method works as $N \to \infty$.
2. A (fast) concentration of measure appears $\Rightarrow$ a "fancy" law of large numbers for equilibrium objects such as value and policy functions.
   - We can calculate conditional expectations with a single Monte Carlo draw from the distribution of idiosyncratic shocks (*no*, in general, you *do not want* to set the shocks to zero: e.g., the typical set of a Normal distribution is an orbit around the mode, not the mode) and quadrature for the aggregate shocks.

Thus, we can handle models with thousands of state variables. Our application today has up to 10,000 states. Perfectly feasible (given enough memory) to handle millions of states.

A deep learning approach

- We show how to train a neural network that implements the permutation-invariant dynamic programming problem as dictated by our representation theorem.
- Strictly speaking, neural networks are not required. You only need a flexible functional basis to implement a projection.
- Neural networks have some numerical advantages, though:
  1. Universal nonlinear approximators that scale very well.
  2. Great libraries such as `PyTorch Lightning`.
  3. Massively parallel (our architectures are implemented in GPUs).

How do we pick our application to show how all this works?

- In terms of application, there are two routes:
  1. I can introduce a sophisticated application where our method "shines."
  2. Or, I can show you how our ideas work in a well-known example.
- In this video, I do not have the time to tell you about the methods *and* the application.
- Besides, if I tell you about a sophisticated application, how do you know our "solution" works?
- So, let me present a well-known example (with a twist)...
- ...and leave for another day the more sophisticated applications.

Our application

A variation of the Lucas and Prescott (1971) model of investment under uncertainty with N firms.

Why?

1. Ljungqvist and Sargent (2018), pp. 226-228, use it to introduce recursive competitive equilibria.
2. Simple model that fits in one slide.
3. Under one parameterization, the model has a known LQ solution, which gives us an exact benchmark:
   - 3.1 We can show that our solution will be extremely accurate.
   - 3.2 The classical control solution is of complexity $\mathcal{O}(N^3)$, whereas our solution is $\mathcal{O}(1)$ for reasonable N.
4. By changing one parameter, the model is nonlinear and, yet, our method handles the nonlinear case as easily as the LQ case and, according to the Euler residuals, with high accuracy.

A 'big X, little x' dynamic programming problem

Consider:

$$
\begin{aligned}
v(x, X) &= \max_{u} \left\{ r(x, u, X) + \beta \mathbb{E}\left[v(x', X') \right] \right\} \\
\text{s.t. } x' &= g(x, u) + \sigma w + \eta \omega \\
X' &= G(X) + \Omega W + \eta \omega \mathbf{1}_N
\end{aligned}
$$

where:

1. x is the individual state of the agent.
2. X is a vector stacking the individual states of all of the N agents in the economy.
3. u is the control.
4. w is random innovation to the individual state, stacked in $W \sim \mathcal{N}(\mathbf{0}_N, \mathbf{I}_N)$ and where, w.l.o.g., $w = W_1$.
5. $\omega \sim \mathcal{N}(0, 1)$ is a random aggregate innovation to all the individual states.

Some preliminaries

- A permutation matrix is a square matrix with a single 1 in each row and column and zeros everywhere else.
  - These matrices are called "permutation" because, when they premultiply (postmultiply) a conformable matrix A, they permute the rows (columns) of A.
- Let $\mathcal{S}_N$ be the set of all $n!$ permutation matrices of size $N \times N$. For example:

$$S_2 = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right\}$$

- (If you know about this): $\mathcal{S}_N$ is the *symmetric group* under matrix multiplication.
  - The algebraic properties of the symmetric group will be doing a lot of the heavy lifting in the proof of our theorems, but you do not need to worry about it.

Permutation-invariant dynamic programming

A 'big X, little x' dynamic programming problem is a permutation-invariant dynamic programming problem if, for all $(x, X) \in \mathbb{R}^{N+1}$ and all permutations $\pi \in \mathcal{S}_N$, the reward function r is permutation invariant:

$$r(x, u, \pi X) = r(x, u, X)$$

the deterministic component of the law of motion for X is permutation equivariant:

$$G(\pi X) = \pi G(X)$$

and the covariance matrix of the idiosyncratic shocks satisfies

$$\pi \Omega = \Omega \pi$$

Permutation invariance of the optimal solution

Proposition

The optimal solution of a permutation-invariant dynamic programming problem is permutation invariant. That is, for all $\pi \in \mathcal{S}_N$:

$$u(x, \pi X) = u(x, X)$$

and:

$$v(x, \pi X) = v(x, X)$$

Main result I: Representation of permutation-invariant functions

Proposition (based on Wagstaff et al., 2019)

Let $f : \mathbb{R}^{N+1} \to \mathbb{R}$ be a continuous permutation-invariant function under $\mathcal{S}_N$, i.e., for all $(x, X) \in \mathbb{R}^{N+1}$ and all $\pi \in \mathcal{S}_N$:

$$f(x, \pi X) = f(x, X)$$

Then, there exist a latent dimension $L \leq N$ and continuous functions $\rho : \mathbb{R}^{L+1} \to \mathbb{R}$ and $\phi : \mathbb{R} \to \mathbb{R}^L$ such that:

$$f(x, X) = \rho\left(x, \frac{1}{N}\sum_{i=1}^{N} \phi(X_i)\right)$$

This proposition should remind you of Krusell-Smith!

Intuition: Take $f : 2^N \to \mathbb{R}$. If f is permutation invariant, it has at most $N + 1$ outputs. Thus, $\phi(\cdot)$ is the identity function (i.e., $L = 1$) and ρ maps the sum of 1's in the string into the $N + 1$ outputs. Using some results from the symmetric group, you can generalize the idea to the $\mathbb{R}^{N+1}$ domain.

Main result II: Concentration of measure

Concentration of measure when expected gradients are bounded in N

Suppose $z \sim \mathcal{N}(\mathbf{0}_N, \Sigma)$, where the spectral radius of Σ, denoted by $\rho(\Sigma)$, is independent of N and $f : \mathbb{R}^N \to \mathbb{R}$ is a function with expected gradient bounded in N. Then:

$$\mathbb{P}\left(\left|f(z) - \mathbb{E}\left[f(z)\right]\right| \geq \epsilon\right) \leq \frac{\rho(\Sigma)C}{\epsilon^2}\frac{1}{N}$$

As Ledoux (2001) puts it: "A random variable that depends in a Lipschitz way on many independent variables (but not too much on any of them) is essentially constant."

With concentration of measure, dimensionality is not a curse; it is a blessing!

A permutation-invariant economy

- Industry consisting of $N > 1$ firms, each producing the same good.
- A firm i produces output x with x units of capital.
- Thus, the vector $X \equiv [x_1, \dots x_N]^\top$ is the production (or capital) of the whole industry.
- The inverse demand function for the industry is, for some $\nu \geq 1$ (this is our twist!):

$$p(X) = 1 - \frac{1}{N} \sum_{i=1}^{N} x_i^\nu$$

- The firm does not consider the impact of its individual decisions on $p(X)$.
- Due to adjustment frictions, investing u has a cost $\frac{\gamma}{2} u^2$.
- Law of motion for capital $x' = (1 - \delta)x + u + \sigma w + \eta\omega$ where $w \sim \mathcal{N}(0, 1)$ an i.i.d. idiosyncratic shock, and $\omega \sim \mathcal{N}(0, 1)$ an i.i.d. aggregate shock, common to all firms.
- The firm chooses u to maximize $\mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t \left(p(X)x - \frac{\gamma}{2} u^2\right)\right]$.

Recursive problem

- The recursive problem of the firm taking the exogenous policy $\hat{u}(\cdot, X)$ for all other firms as given is:

$$
\begin{aligned}
v(x, X) = \max_{u} \left\{ p(X)x - \frac{\gamma}{2}u^2 + \beta \mathbb{E}\left[v(x', X')\right] \right\} \\
\text{s.t. } x' = (1-\delta)x + u + \sigma w + \eta\omega \\
X_i' = (1-\delta)X_i + \hat{u}(X_i, X) + \sigma W_i + \eta\omega, \quad \text{for } i \in \{1, ..., N\}
\end{aligned}
$$

Equilibrium and Euler equation

Definition

A recursive permutation-invariant competitive equilibrium is a $v(x, X)$, $\hat{u}(x, X)$, and laws of motion for capital such that:

- Given $\hat{u}(x, X)$, $v(x, X)$ is the value function solving the recursive problem from previous slide for each agent and $u(x, X)$ is the optimal policy function.
- The optimal policy is symmetric, i.e., $u(x, X) = \hat{u}(x, X)$.
- The laws of motion for capital satisfy:

$$X_i' = (1 - \delta)X_i + \hat{u}(X_i, X) + \sigma W_i + \eta\omega, \quad \text{for } i \in \{1, ..., N\}$$

The Euler equation for the firm:

$$\gamma u(x, X) = \beta \mathbb{E}\left[p(X') + \gamma(1 - \delta)u(x', X')\right]$$

Solving the model

- We want to find a global solution that is accurate beyond a ball around some particular X^{ss} (usually the steady state of the model).
- Why? Compute transitional dynamics from far away the steady state, study large shocks, ...
- From the representation of permutation-invariant functions, we know that the policy function that satisfies the previous Euler equation has the form:

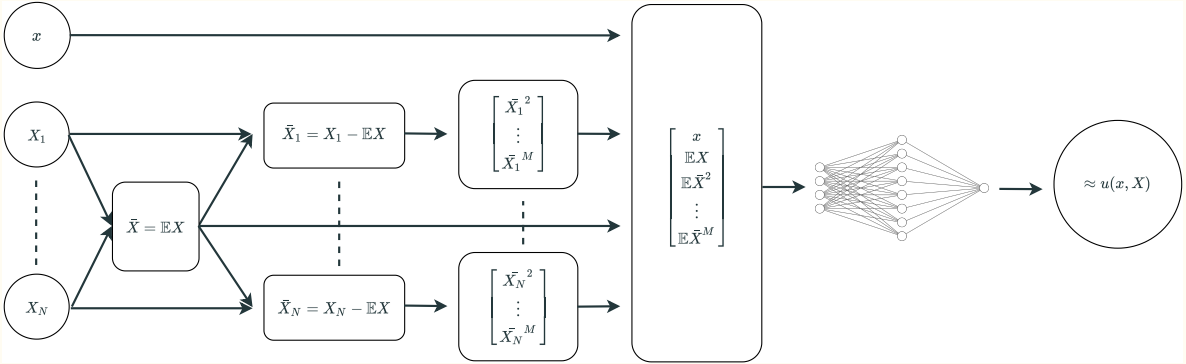
$$u(x, X) = \rho\left(x, \frac{1}{N}\sum_{i=1}^{N}\phi(X_i)\right)$$

- But, in general, we do not know $\rho(\cdot)$ or $\phi(\cdot)$.
- Thus, we will approximate $\rho(\cdot)$ and $\phi(\cdot)$ using deep learning.

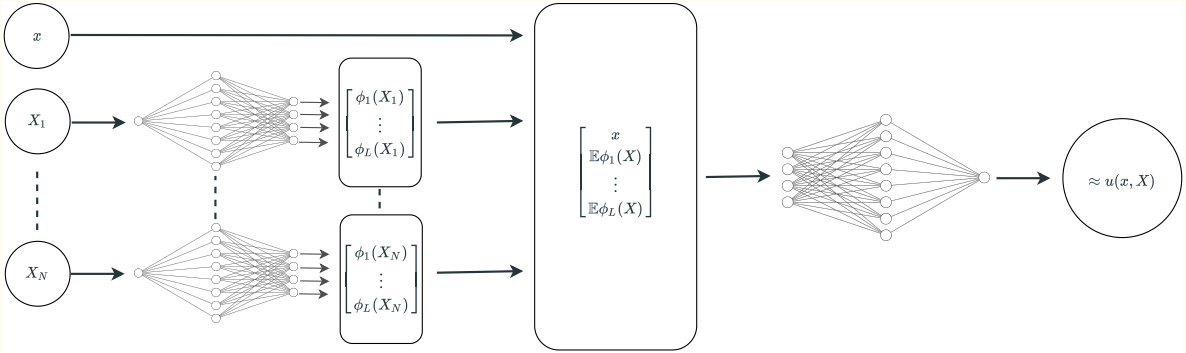
Our deep learning architectures

- We will specify several deep learning architectures $\mathcal{H}(\theta)$:
  1. ϕ is approximated as a function of a finite set of moments à la Krusell-Smith but in a fully nonlinear way as in Fernández-Villaverde et al. (2019). We use 1 and 4 moments.
  2. ϕ is approximated by a flexible ReLU network (with two layers each with 128 nodes).
- The baseline ϕ(Identity), ϕ(Moments), and ϕ(ReLU) have 49.4K, 49.8K, and 66.8K coefficients respectively regardless of N.
- In all cases, ρ is a highly parameterized neural network with four layers.
- A surprising benefit of a high-dimensional approximation is the "double-descent" phenomenon in machine learning (see Belkin et al., 2019, and Advani et al., 2020): more coefficients makes it easier to find minimum-norm solutions.
- All the code in `PyTorch Lightning` and run on GPUs.

Moments architecture



ReLU architecture



Training and calibration

Algorithm Network training

1: Given by network architecture $\mathcal{H}(\theta)$ for $u(x, X)$, define the Euler residuals:

$$\varepsilon(x, X; \theta) \equiv \gamma u(x, X) - \beta \mathbb{E}\left[P(X') + \gamma(1-\delta)u(x', X')\right]$$

2: Pick $\{X^m(0), ..., X^m(T)\}$ for $m = 1, .., M$ trajectories given some initial point of interest.

3: Evaluate $\varepsilon_{m,t}(x, X; \theta)$ for some or all the points above.

4: Solve using ADAM (a stochastic gradient descent with adaptive moment estimation):

$$\min_{\theta} \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{T} \left(\varepsilon_{m,t}(x, X; \theta)\right)^2$$

- Parameter values: $\beta = 0.95, \gamma = 90, \sigma = 0.005$, and $\eta = 0.001$. Idiosyncratic risk 5 times larger than aggregate risk.
- We will study two cases: linear ($\nu = 1$) and nonlinear ($\nu > 1$) demand functions.

Case I

- With $\nu = 1$, we have a linear demand function: $p(X) = 1 - \frac{1}{N}\sum_{i=1}^{N} x_i$.
- It generates an LQ dynamic programming problem (only the mean of x_i matters!).
- We can find the exact $u(x, X)$ using the linear regulator solution.
- The LQ solution gives us a benchmark against which we can compare our deep learning solution.
- The neural network "learns" very quickly that the solution is $u(x, X) = H_0 + \frac{1}{N} H_1 \sum_{i=1}^{N} x_i$.
- We also compute a modified linear regulator solution with *one* Monte Carlo draw instead of setting the individual shocks to zero: illustrates how concentration of measure works.
- Bonus point: we show how to implement this modified linear regulator solution. Useful for non-Gaussian LQ problems where certainty equivalence does not hold.

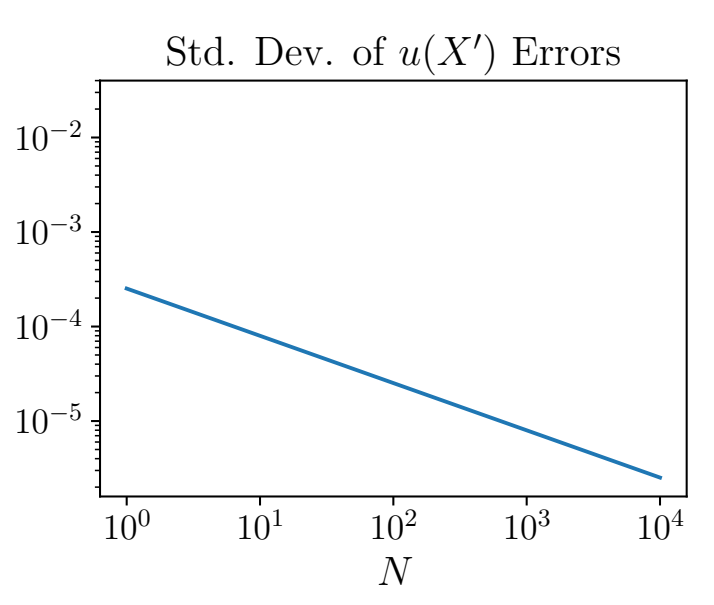


Figure 1: The concentration of the optimal policy $u(X')$ for $\nu = 1$.

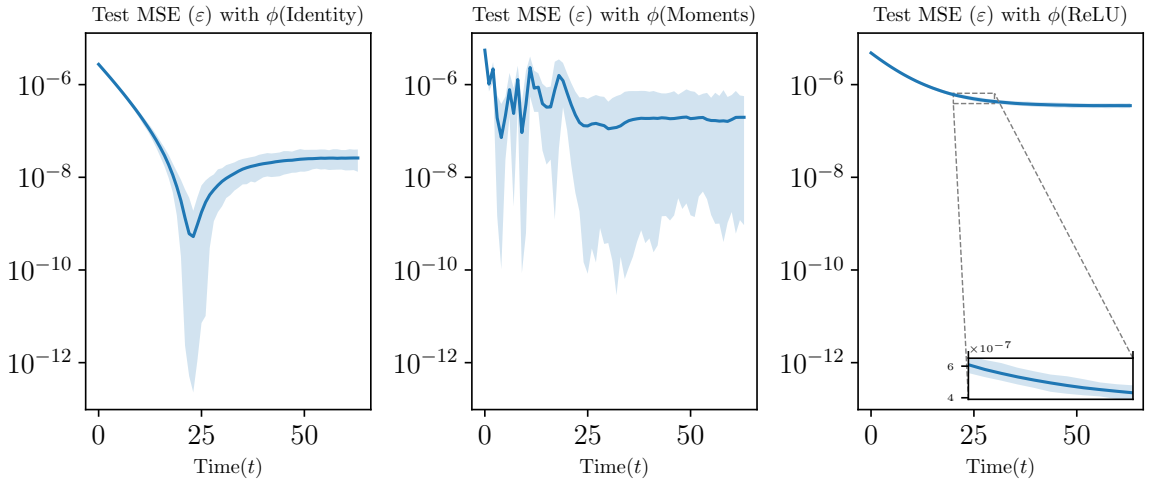


Figure 2: The Euler residuals for $\nu = 1$ and $N = 128$ for ϕ(Identity), ϕ(Moments), and ϕ(ReLU). The dark blue curve shows the average residuals along equilibrium paths for 256 different trajectories. The shaded areas depict the 2.5th and 97.5th percentiles.

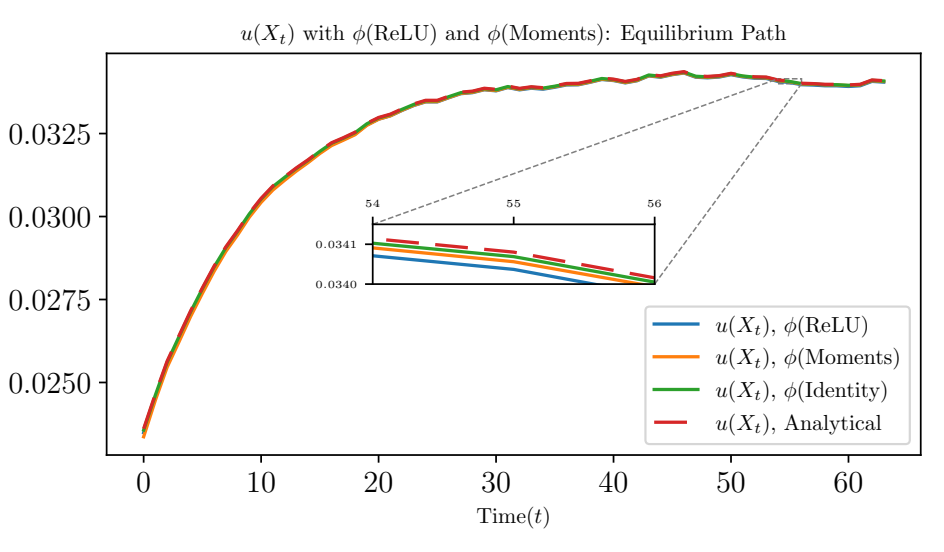


Figure 3: Comparison between baseline approximate solutions and the LQ-regulator solution for the case with $\nu = 1$ and $N = 128$.

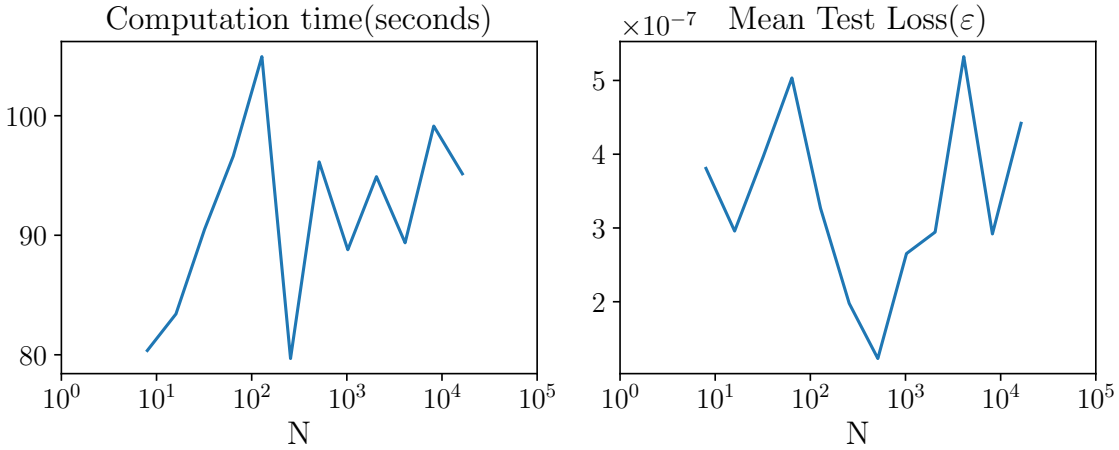


Figure 4: Performance of the ϕ(ReLU) for different N (median value of 21 trials).

Case II

- With $\nu > 1$, we have a nonlinear demand function: $p(X) = 1 - \frac{1}{N}\sum_{i=1}^{N} x_i^{\nu}$.
- Notice how, now, the whole distribution of of x_i matters!
- But we can still find the solution to this nonlinear case using exactly the same functional approximation and algorithm as before.
- We do not need change anything in the code except the value of ν.
- Since the LQ solution no longer holds, we do not have an exact solution to use as a benchmark.
- But we can always check the Euler residuals.

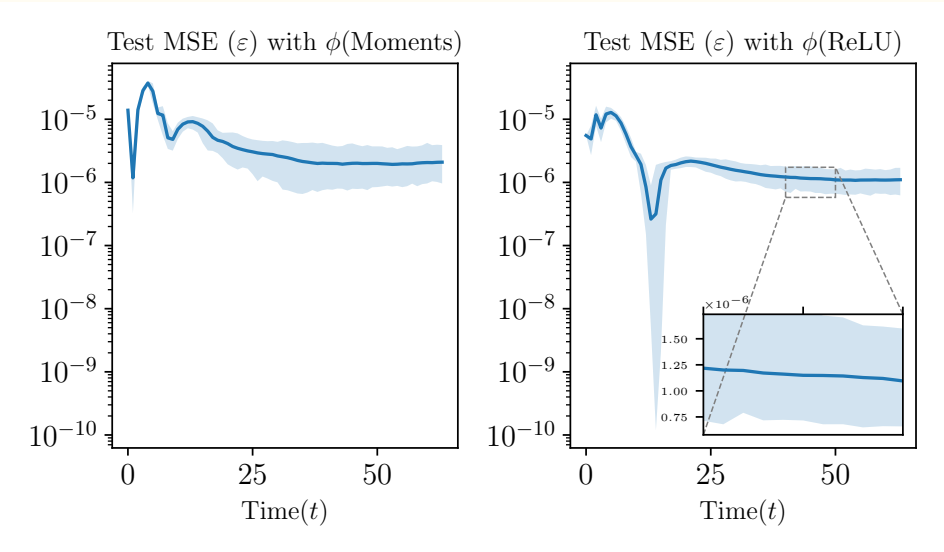


Figure 5: The Euler residuals for $\nu = 1.5$ and $N = 128$ for ϕ(Moments) and ϕ(ReLU). The dark blue curve shows the average residuals along equilibrium paths for 256 different trajectories. The shaded areas depict the 2.5th and 97.5th percentiles.

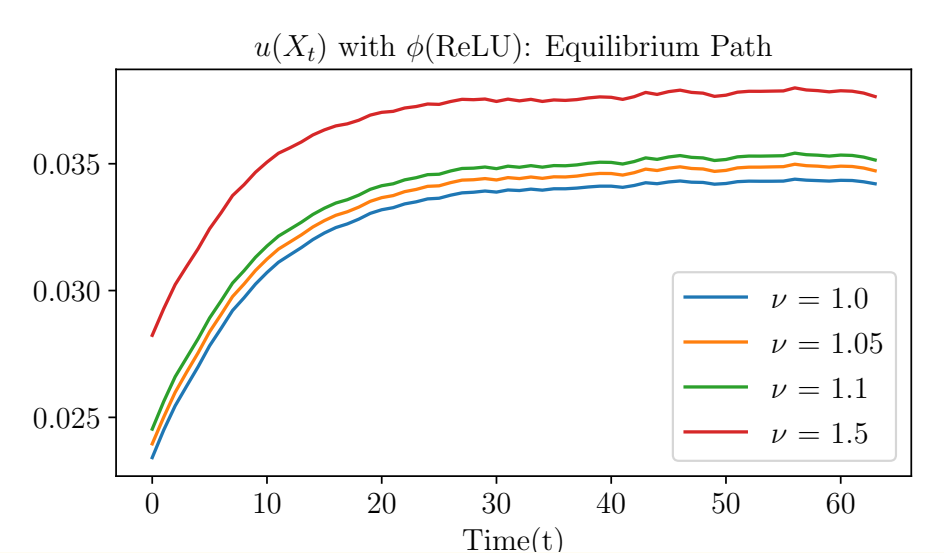


Figure 6: The optimal policy u along the equilibrium paths for $\nu = [1.0, 1.05, 1.1, 1.5]$ and $N = 128$. Each path shows the optimal policy for a single trajectory.

Extensions

1. Decreasing returns to scale: the policy becomes a function of x.
2. Multiple productivity types.
3. Complex idiosyncratic states.
4. Global solutions with transitions and aggregate shocks.
5. "Non-atomic" agents.
6. Many different network architectures.

Examples of models one can compute now

1. Models with rich ex-ante heterogeneity and aggregate shocks (OLG, many different household types).
2. Models of firm dynamics.
3. Open economy models with many locations.
4. Closed economy business cycle models with multisectors.
5. Network models.
6. Search and matching models.