Continuation Value Is All You Need:

A Simple, General Deep Learning Method for Globally Solving Recursive Economic Models with Aggregate Uncertainty

[Click here for latest version]

Jeffrey E. Sun

November 24, 2023

Abstract

This is currently a work in progress. This version of the paper is identical to the computational appendix of my job market paper. It makes specific reference to the model which I build there and employs some optimizations which are tailored to that model.

1 Introduction

This method exploits the fact that if we know the current utility and continuation value resulting from every action in a household's choice set, then we can typically solve for the household's optimal choice, even under conditions of aggregate uncertainty. Existing deep learning techniques typically use a neural network to approximate both the policy function and the value function. This requires techniques to train the policy function approximator which are generally less robust than the training of the value function. Approximating the policy function with a neural network also imposes constraints such as, commonly, the differentiability of the continuation value with respect to the policy variable. Aided by optimized algorithms I develop to quickly solve for optimal household choices, I show that solving for the optimal policy "online" during training is a viable—and indeed more flexible and arguably simpler—alternative to approximating the policy function using a neural network. A relatively inexpensive restriction is that prices are pinned down by intratemporal market clearing conditions given household conditional continuation values. In principle, prices can also be solved "online," but in practice I use an auxiliary neural network to approximate prices, as in Azinovic et al. (2022). I use this method to solve the heterogeneous-agent dynamic spatial model with aggregate uncertainty in my job market paper.

2 Computational Method

The solution of the model consists of two parts. First, the solution of a complex heterogeneous-agent model without aggregate uncertainty, which does not require neural networks. Second, the solution of such a model with aggregate uncertainty, which does require a targeted application of neural networks. Part of my approach, however, is to use neural networks as little as possible. This reduces complexity and room for error in the model solution.

Section 2.1 describes the solution method in a stationary steady state. Section 2.2 describes the solution method when aggregate variables might be changing, but do so in a perfectly-foreseen way, without aggregate uncertainty. Section 2.3 describes how to use neural networks to incorporate aggregate uncertainty. Section 2.4 describes a proposal for the general use of these techniques in more challenging models where aggregate uncertainty persists forever and there is a nontrivial ergodic distribution.

2.1 Solving Discrete-Time Heterogeneous-Agent Models in Stationary Steady State

The model I describe in my job market paper has a relatively large number of model features (frictions, household choices, etc.). The majority of this complexity is in the household problem. In this section, let us consider the stationary steady state of the full model, in which emissions are equal to zero, $e_t = 0 \ \forall t$, and the economy has converged to stationary steady state.

In the stationary steady state, equilibrium quantities (housing prices, rents, and stocks) are fixed over time. The stationary steady state can be solved by first solving the household's problem given equilibrium quantities, then, in an outer loop, solving for equilibrium quantities to satisfy market-clearing conditions.

Algorithm 1 Stationary Steady State Outer Loop

1. `Guess housing prices and rents for each location`
2. `Solve household problem for all households`
3. `Compute market clearing conditions`
4. `Update housing price and rent guesses and repeat until convergence`

2.1.1 Solving the Household's Problem

In the stationary steady state, while household value functions do not depend on time, the household's problem is still quite complex. The household's idiosyncratic transition each period has many elements: choosing location, choosing housing investment, paying realtor's fees, etc. I break this complexity into

manageable pieces by treat these elements as occuring one at a time and solve for each one independently. That is, I treat a period of the household's problem as consisting of several *stages*, where the idiosyncratic transition happening in each stage is simple and manageable.

This leverages a useful property of discrete-time models. While, in a continuous-time model, all stages (elements of the household's idiosyncratic transition) are happening "simultaneously," in a discrete time model they can happen one at a time.

Intuitively, each stage of the household's problem consists of one thing happening to a household or one choice made by the household. We can then backwards induct through each stage in a period to solve the household's problem. Crucially, because these stages happen sequentially, the computation time needed to solve the household's problem is *linear* in the number of stages (but still exponential in the number of idiosyncratic state variables). In principle, this allows us to solve household's problems with arbitrarily many elements in the household's transition, as long as the number of idiosyncratic state variables remains manageable. I provide self-contained code implementing each stage, so that stages can be added, removed, and reordered at virtually zero cost of programmer time.[1]

I now formally define a "stage" and the way in which stages can be computationally approximated in a uniform, composable way. The formalism is a little abstract, but the basic idea is simple: each stage represents a simple household transition. For each stage, a Bellman equation implies a backwards induction function taking continuation values to beginning-of-stage values. The purpose of the abstraction is to show that every element of the household's problem can be represented in a unified way by the same general abstract structure that is amenable to computation.[2]

2.1.2 Decomposing a Complex Idiosyncratic Decomposition into Stages

Consider one element of the household's idiosyncratic transition, such as the arrival of income, the consumption-savings decision, or the application of a borrowing constraint. Assume that the

Definition 1. *Stage.* A *stage* $S = \left(\boldsymbol{X}^{pre}, \boldsymbol{X}^{post}, \Xi, \Pi\right)$ consists of a beginning-of-stage household state space $\boldsymbol{X}^{\text{pre}}$, an end-of-stage household state space $\boldsymbol{X}^{\text{post}}$, a "backward induction operator" taking an end-of-period

[1] These "stage functions" could even be written by different researchers and combined to easy program a large variety of models.

[2] This is true in my model but certainly not over the space of all possible models, and in fact relies on the property that household decisions do not interact with each other except through prices. There is some analogy here to the increasingly-popular mean-field-game approach.

continuation value function V^{post} to a beginning-of-period value function V^{pre},

$$\begin{aligned}\Xi : \mathbb{R}^{\boldsymbol{X}^{\text{post}}} &\longrightarrow \mathbb{R}^{\boldsymbol{X}^{\text{pre}}} \\ : V^{\text{post}} &\mapsto V^{\text{pre}} \\ : \left(V^{\text{post}} : \boldsymbol{X}^{\text{post}} \to \mathbb{R}\right) &\mapsto \left(V^{\text{pre}} : \boldsymbol{X}^{\text{pre}} \to \mathbb{R}\right),\end{aligned}$$

and a "forward simulation operator" taking a beginning-of-stage household distribution λ^{pre} and an end-of-stage value function V^{post} to an end-of-stage household distribution λ^{post},

$$\begin{aligned}\Pi : \Lambda\left(\boldsymbol{X}^{\text{pre}}\right) \times \mathbb{R}^{\boldsymbol{X}^{\text{post}}} &\longrightarrow \Lambda\left(\boldsymbol{X}^{\text{post}}\right) \\ : \left(\lambda^{\text{pre}}, V^{\text{post}}\right) &\mapsto \lambda^{\text{post}} \\ : \left(\left(\lambda^{\text{pre}} : \mathcal{P}\left(\boldsymbol{X}^{\text{post}}\right) \to \mathbb{R}^{+}\right), \left(V^{\text{post}} : \boldsymbol{X}^{\text{post}} \to \mathbb{R}\right)\right) &\mapsto \left(\lambda^{\text{post}} : \mathcal{P}\left(\boldsymbol{X}^{\text{post}}\right) \to \mathbb{R}^{+}\right),\end{aligned}$$

where $\Lambda(\mathbf{X})$ is the space of measures $\lambda : \mathcal{P}(\mathbf{X}) \to \mathbb{R}^{+}$ on $\mathbf{X}$.

Alternatively, we can think of a stage as a single operator iterating V^{post} backwards and λ^{pre} forwards:

$$\begin{aligned}S : \Lambda\left(\boldsymbol{X}^{\text{pre}}\right) \times \mathbb{R}^{\boldsymbol{X}^{\text{post}}} &\longrightarrow \Lambda\left(\boldsymbol{X}^{\text{post}}\right) \times \mathbb{R}^{\boldsymbol{X}^{\text{pre}}} \\ : \left(\lambda^{\text{pre}}, V^{\text{post}}\right) &\mapsto \left(\lambda^{\text{post}}, V^{\text{pre}}\right).\end{aligned}$$

Furthermore, we can compose two stages to form a composite stage.[3]

Definition 2. *Composition of Stages.* Let $S_1 = \left(\boldsymbol{X}_1^{\text{pre}}, \boldsymbol{X}_1^{\text{post}}, \Xi_1, \Pi_1\right)$ and $S_2 = \left(\boldsymbol{X}_2^{\text{pre}}, \boldsymbol{X}_2^{\text{post}}, \Xi_2, \Pi_2\right)$ be two stages. Their composition $S_{12} = S_1 \circ S_2 = \left(\boldsymbol{X}_1^{\text{pre}}, \boldsymbol{X}_2^{\text{post}}, \Xi_{12}, \Pi_{12}\right)$ is a stage given by,

$$\begin{aligned}\Xi_{12} \equiv \Xi_1 \circ \Xi_2 : \mathbb{R}^{\boldsymbol{X}_2^{\text{post}}} &\longrightarrow \mathbb{R}^{\boldsymbol{X}_1^{\text{pre}}} \\ \Pi_{12} : \Lambda\left(\boldsymbol{X}_1^{\text{pre}}\right) \times \mathbb{R}^{\boldsymbol{X}_2^{\text{post}}} &\longrightarrow \Lambda\left(\boldsymbol{X}_2^{\text{post}}\right) \\ \Pi_{12}\left(\lambda_1^{\text{pre}}, V_2^{\text{post}}\right) = \Pi_2\left(\Pi_1\left(\lambda_1^{\text{pre}}, \Xi_2\left(V_2^{\text{post}}\right)\right), V_2^{\text{post}}\right).\end{aligned}$$

Thus, an entire period can be thought of as a single stage. The purpose of this, however, is to decompose periods into the simplest possible stages.

This level of abstraction might seem unnecessary. but the essential point is that, for each stage, all we need to do is map a value function backwards and map a household distribution forwards. In particular, we can make these stages individually very simple and then compose stages to define complex models. We can

[3]This composition rule is associative and there is a natural "identity" stage on any state space $\mathbf{X}$ which changes neither V nor λ. It might be of some conceptual interest to note that we can therefore define a mathematical category **Stage** with the stages as the morphisms and the state spaces as the objects.

solve those models by composing the numerical approximations of each stage.

To make things more concrete, I describe how one period of the household problem of a simple Aiyagari model can be subdivided into five stages: receiving income, the consumption-savings decision, the application of a borrowing constraint, an idiosyncratic income shock, and the passage of time.

For each example stage, let an individual household's beginning-of-stage state be given by

$$x = (b, z) \in \mathbf{X} = \mathbf{X}^{\text{pre}} = \mathbf{X}^{\text{post}},$$

where b represents the household's bondholdings and z represents the household's income. The beginning-of-period state space $\mathbf{X}^{\text{pre}}$ and end-of-period state space $\mathbf{X}^{\text{post}}$ are equal. Let the end-of-state continuation value function be given by $V^{\text{post},s}(b, z)$ for $s \in \{\text{income, consume,constraint,shock,time}\}$.[4] To save on notation, I omit the stage-specific superscript within each example.

Example 1. Receiving Income.

Suppose that after receiving income, a household with initial state (b, z) has continuation state $(Rb + z, z)$. The household's backward induction operator is given by,

$$\Xi^{\text{income}} V^{\text{post}}(Rb, z) = V^{\text{post}}(Rb + z, z),$$

and the forward simulation operator is given by,

$$\Pi^{\text{income}} \lambda^{\text{pre}}(T \subseteq \mathbf{X}) = \lambda^{\text{pre}}(\{(Rb, z) \mid (Rb + z, z) \in T\}).$$

A similar thing is true for any deterministic household transition $f : \mathbf{X}^{\text{pre}} \to \mathbf{X}^{\text{post}}$.[5]

Example 2. Consumption-Savings Decision.

Suppose that a household with initial state (b, z) chooses continuation state (b', z) maximizing

$$\Xi^{\text{consumption}} V^{\text{post}}(b, z) = u(b'^* - b) + V^{\text{post}}(b'^*, z)$$
$$\text{where} \quad b'^*\left(b, z; V^{\text{post}}\right) \equiv \arg\max_{b'} u(b'^* - b) + V^{\text{post}}(b'^*, z)$$

This defines the backward induction operator of a consumption-savings decision stage.[6] The forward

[4] I abuse notation somewhat in not writing $V^{\text{post},s}((b, z))$.

[5] Specifically, for any such f, the backward induction operator is just the pullback or precomposition of V^{post} by f and the forward simulation operator is the pullback of λ^{pre} by the inverse image map $f^{-1} : \mathcal{P}(\mathbf{X}^{\text{post}}) \to \mathcal{P}(\mathbf{X}^{\text{pre}})$ associated with f.

[6] Note that b' is fully unconstrained here, and there is no time-discounting; these will occur in different stages.

simulation operator is given by

$$\Pi^{\text{consumption}}\lambda^{\text{pre}}\left(T \subseteq \mathbf{X}; V^{\text{post}}\right) = \lambda^{\text{pre}}\left(\left\{(b,z) \mid \left(b'^{*}\left(b,z;V^{\text{post}}\right),z\right) \in T\right\}\right).$$

Example 3. Borrowing Constraint.

Suppose that a household is subject to the borrowing constraint $b \geq 0$. We can then define a borrowing-constraint stage by,

$$\Xi^{\text{constraint}}V^{\text{post}}(b,z) = \begin{cases} V^{\text{post}}(b,z) & b \geq 0 \\ -\infty & b < 0 \end{cases}$$

$$\Pi^{\text{constraint}}\lambda^{\text{pre}} = \lambda^{\text{pre}}.$$

Example 4. Idiosyncratic Income Shock.

Suppose that a household's income z evolves according to

$$z' \sim \mathcal{D}(z)$$

for some distribution $\mathcal{D}$. Then the idiosyncratic income shock stage is given by,

$$\Xi^{\text{shock}}V^{\text{post}}(b,z) = \mathbb{E}_{z'}\left[V^{\text{post}}(b,z') \mid z\right]$$

$$\Pi^{\text{shock}}\lambda^{\text{pre}}\left(T;V^{\text{post}}\right) = \int_{(b,z)} P\left((b,z') \in T \mid z\right) d\lambda^{\text{pre}}$$

$$\text{where} \quad T \subseteq \mathbf{X}.$$

Example 5. Passage of Time.

Suppose that time passes between the beginning and end of the stage, with discount factor β. Then, the passage-of-time phase is given by,

$$\Xi^{\text{time}}V^{\text{post}}(b,z) = \beta V^{\text{post}}(b,z)$$

$$\Pi^{\text{time}}\lambda^{\text{pre}} = \lambda^{\text{pre}}.$$

At this point, this might well seem like a great deal of formalism with no interesting content. The purpose of this all, however, is just to show that solving a model with a complex household transition is not especially difficult, as long as we can break the period into stages and numerically approximate each stage. For example, the full list of stages in my full model are:

- Apply impact of house price changes on household net worth.
- Decide whether or not to move.
- Choose location if moving.
- Choose whether to sell owner-occupied housing.
- Choose whether to buy owner-occupied housing, and how much.
- Choose whether to sell rental real estate.
- Choose whether to buy rental real estate, and how much.
- Receive income and pay expenses.
- Make consumption-saving decision.
- Receive idiosyncratic income shock.
- Passage of time.
- End-of-life and bequest (if at maximum age).

2.1.3 Numerically Solving Each Stage

The first step in numerically solving a stage is to discretize the state space. My full model has six idiosyncratic state variables: bondholdings, income type, owner-occupied housing, rental real estate, location, and age. I use a separate array for each age group. For the other five variables, I define a five-dimensional array which is the product of portfolio value, income type, owner-occupied housing, rental housing, and location grids. I use portfolio value (bondholdings + real estate value) instead of bondholdings for the first dimension so that buying or selling real estate does not move households around the grid too dramatically. While I use the full product grid, significant savings might be possible with a dynamic spare grid approach, such as Brumm et al. (2022) use for continuous-time models.

I then represent both value functions V and household state distributions λ as arrays A_V and A_λ of this form, and construct a pair of arrays for each (more or less) stage and age group. A_V represents the interpolation nodes of the function V on the grid and A_λ represents a grid of point masses on those same grid points. I use the same grid for every stage to improve the reorderability and composability of the stages. I then implement each stage as a pair of functions taking arrays to arrays: one taking end-of-stage V^{post} to beginning-of-stage V^{pre} and one taking end-of-stage V^{post} and beginning-of-stage λ^{pre} to end-of-stage λ^{post}.

Four challenges arise in doing this in a performant manner. First, reinterpolating V^{pre} onto the grid points after each backwards induction step. Second, reinterpolating λ^{post} back onto the grid points after

each forward simulation step. Third, choosing optimal consumption for each grid point at the consumption-savings step. Fourth, simulating implementing the location choice stage, which requires a household at each grid point to choose between 1713 locations, with over 200 million gridpoints. I provide optimized algorithms implemented in Julia for overcoming each of these bottlenecks.

Because my model is a lifecycle model, I can begin by computing the bequest value at each terminal gridpoint, then solve the entire household's problem backwards from there. For age group a, the value function V_a^{end} at the end of each period is simply equal to the value function at the start of the following period for the next age group,

$$V_a^{\text{end}} = V_{a+1}^{\text{start}}.$$

To solve for prices, I apply the outer loop algorithm I describe above.

2.2 Transition Dynamics

Having solved for stationary steady state, we can now extend to transition dynamics. Consider a version of my main model with perfect foresight after the onset shock. That is, while the onset of climate change is unexpected, after the onset shock all households understand that climate climate will continue deterministically along the median path, where $\varepsilon_t^m = 0 \ \forall t$. To solve this, we can simply pick an end date far in the future (I choose 2300) at which we assume that the economy has returned to its stationary steady state. Then, given prices, we can simply use backwards induction to solve value functions from 2300 to 1990, then simulate the household state distribution forward from 1990 to 2300. We can then solve for prices in an outer loop.

Algorithm 2 Deterministic Transition Outer Loop

1. `Solve 2300 stationary steady state.`
2. `Guess housing prices and rents for each location and decade.`
3. `Solve household problem for all households across entire transition path, from 1990 onset shock to 2300 new steady state.`
4. `Compute market clearing conditions.`
5. `Update housing price and rent guesses and repeat from Step 3 until convergence.`

For age group a, the value function V_{at}^{end} at the end of period t, is simply equal to the value function for

the following age group at the start of the following period,

$$V_{at}^{\text{end}} = V_{a+1,t+1}^{\text{start}}.$$

2.3 Aggregate Risk

To solve the model with aggregate risk, I apply a similar strategy. The difference is that value function now depend on the aggregate state Γ, not only on the idiosyncratic state and a time index. The aggregate state changes between the end of the household's problem in one period and the beginning of the household's problem in the following period,

$$V_{at}^{\text{end}}(x;\Gamma) = \mathbb{E}_{\Gamma'}\left[V_{a+1,t+1}^{\text{start}}(x;\Gamma') \mid \Gamma\right].$$

Note that there is no need to solve the policy function. The policy function is calculated on-the-fly during the simulation of the model within each period. Because we do not need to solve for the policy function, we only need to use neural networks to approximate the value function (and, as an optimization, the price function). This is essentially a form of supervised learning, which is generally much easier and more stable than using a neural network to solve for a policy function.

To use a neural network to approximate $V_{at}^{\text{end}}(x;\Gamma)$ requires two steps. First, I must reduce the dimension of Γ somehow. Second, I must train the neural network.

Dimension Reduction There are many ways to reduce the dimension of Γ. One fairly robust way to do it is introduced in Han, Yang, and E (2023) with the DeepHAM method. However, I simply exploit the fact that the aggregate state in my model is simply a function of initial stationary steady state (which does not change and thus does not need to be inputted into the neural network) and the history of aggregate shocks ε_t^m following the onset shock.[7] Thus, the neural net approximator for $V_{at}^{\text{end}}(x;\Gamma)$ requires only $5+t$ inputs, where t is the number of periods since the onset shock. Let $\mathcal{N}_{at}^{\text{end}}$ represent this neural net approximator. Then,

$$V_{at}^{\text{end}}(x;\Gamma) = \mathcal{N}_{at}^{\text{end}}(x;v).$$

I simply define a separate neural network approximator $\mathcal{N}_{at}^{\text{end}}$ for each age a and period t.

Training To train this neural network, I first discretize the aggregate shock into 5 possible states for each ε_t^m, representing the median and each one- or two-standard deviation surprise. Let $\{\eta_k\}_{k=1}^5$ represent each

[7] I thank Robert Wagner for this idea.

of these five realizations. Then the following holds with equality:

$$V_{at}^{\text{end}}(x;\Gamma) = \sum_{k=1}^{5} V_{a+1,t+1}^{\text{start}}(x;\Gamma'(\Gamma,k))\, P(\varepsilon_t^m = \eta_k). \tag{1}$$

I can then train the neural network by the following algorithm:

Algorithm 3 Neural Network Training Loop

1. `Solve 2300 stationary steady state.`
2. `Initialize a neural network` $\mathcal{N}_{at}^{\texttt{end}}$ `for each age` a `and time` t.
3. `Draw a path of aggregate shocks` $\{\varepsilon_t^m\}_{t=1}^T$ `and compute the resulting temperature path` $\{\texttt{SST}_t^m\}_{t=1}^T$.
4. Working forwards, for each age a and time τ:
   (a) Guess period-t prices $\{\hat{\rho}_{it}(\varepsilon_1^m,\ldots,\varepsilon_t^m), \hat{q}_{it}(\varepsilon_1^m,\ldots,\varepsilon_t^m)\}_{i\in I}$.
   (b) Compute from neural network $\widehat{V}_{at}^{\text{end}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m) = \mathcal{N}_{at}^{\text{end}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m)$ at each grid point x.
   (c) Simulate $\lambda_{at}^{\text{start}}$ forward based on this $\widehat{V}_{at}^{\text{end}}$.
   (d) Compute market clearing conditions within period t.
   (e) Repeat from Step 4a until market clearing conditions satisfied within tolerance.
5. Working backwards, for each age a and time τ:
   (a) For each $k \in \{1,\ldots,5\}$:
      i. Guess period-$t+1$ prices $\{\rho_{it+1}(\varepsilon_1^m,\ldots,\varepsilon_t^m,\eta_k), q_{it+1}(\varepsilon_1^m,\ldots,\varepsilon_t^m,\eta_k)\}_{i\in I}$ for each location i.
      ii. Compute $\widehat{V}_{at+1}^{\text{end}}(x;\Gamma,\varepsilon_1^m,\ldots,\varepsilon_t^m,\eta_k) = \mathcal{N}_{at}^{\text{end}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m,\eta_k)$.
      iii. Simulate $\lambda_{at+1}^{\text{start}}$ forward based on this $\widehat{V}_{at+1}^{\text{end}}$.
      iv. Compute market clearing conditions within period $t+1$
      v. Repeat from step 5a(i) until market clearing conditions satisfied within tolerance.
      vi. Iterate backwards to obtain $\widehat{V}_{at+1}^{\text{start}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m,\eta_k)$.
   (b) Compute
   $$\widetilde{V}_{at}^{\text{end}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m) = \sum_{k=1}^{5} \widehat{V}_{at+1}^{\text{start}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m,\eta_k)\, P(\varepsilon_t^m = \eta_k)$$
   (c) Update $\mathcal{N}_{at}^{\text{end}}$ with error target $\mathcal{N}_{at}^{\text{end}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m) - \widetilde{V}_{at}^{\text{end}}(x;\varepsilon_1^m,\ldots,\varepsilon_t^m)$.

In practice, I move the determination of prices to the outer loop and use a neural network to guess prices once for each draw of the aggregate shock path. I then update the price network using excess supply of rental real estate and rental properties for each location: the errors in the market clearing conditions. This is the approach of Azinovic, et al. (2023)

2.4 Greater Generality

2.5 Optimized Algorithms for Interpolations, Choice Probabilities, and Optimal Choice

The primary numerical methods contribution in this paper is the use of deep learning (neural networks) to solve for prices and household conditional continuation values, as a function of aggregate uncertainty, given the aggregate state.

The key insight of the deep learning method is that, if a household knows the current utility and continuation values resulting from any action, then conventional methods can be used to solve the policy function. With one or two dimensions of heterogeneity, this is generally no problem. But because the model has so many individual states (213 million), it was necessary to develop a few custom algorithms for performance.

These supplementary algorithms overcome some computational bottlenecks when scaling up discrete-time heterogeneous-agent models to large numbers of individual states and locations. In the hope that someone finds them useful, I have made a separate Github repository for them, in highly-optimized parallelized Julia code.

2.5.1 All-at-once interpolating functions and distributions on endogenous gridpoints

Speedup factor: log n (n = #(gridpoints), relative to individual lookups)

When solving value functions using backwards induction or simulating distributions forward, if the gridpoints are defined in terms of endogenous quantities (e.g. wealth), you often have to reinterpolate the function or distribution back onto the grid. Existing interpolation libraries look up each gridpoint independently. With n gridpoints, each query is O(log n) and the whole reinterpolation is O(n log n). But if the grid is monotonic, you only need to iterate over each grid once, which you can do in O(n) time.

Reinterpolating is slightly different for functions and distributions, since distribution approximations depend on the density of gridpoints.

2.5.2 Computing choice probabilities for heterogeneous agents with a single matrix multiplication

Speedup factor: up to $n^{0.62}$ or so, with big improvement on constant (if n = #(locations), relative to naive approach)

With n locations and m possible within-location agent states, if agents receive Gumbel-distributed location preferences, then to solve for the value function and choice probabilities, you have to solve for n*n*m possible agent-choice pairs. Luckily, this can be constructed as a single matrix multiplication, which can even be done non-allocatingly if set up properly. Highly-optimized linear algebra libraries can then be used.

2.5.3 All-at-once solving monotone decision function for heterogenous agents

Speedup factor: n (n = #(gridpoints), relative to naive individual lookups. Speedup factor log n over optimized individual lookups)

In many models, agents must make consumption savings decisions. Assuming a discretized wealth grid, the most flexible solution is to maximize over all possible continuation values for each agent, but this requires O(n^2) total computations. If both continuation value and optimal saving are increasing in wealth, each agent needs only consider possible choices that lie between the choices of adjacent agents. By using binary search for individual lookups and a form of modified binary search to constrain the search space for individual agents, we can compute the optimal decisions of n agents who differ only by wealth using only O(n) computations.