Some "Conventional" Methods For Solving Spatial Models

(Quickly)

Jeffrey Sun

April 24, 2024

Introduction

Introduction

- A constructive theory of the model solution
- An attempt to generalize, package, and usefully share what I've learned
- All "conventional:" no machine learning, GPUs, sparse grids, continuous time

Background

- Coding up JMP quickly got too complicated
- Had to break up into clear, *computationally separate* blocks
- As each block got simpler, it became more general and optimized
- Over time: reusable, composable, optimized modules for writing fast models quickly
- No claims about novelty, only hopes about usefulness

Overview

1. A simple dynamic spatial model with migration, and wealth and income heterogeneity
2. High-level decomposition of model solution
3. Decomposition of intra-period household problem into "stages"
4. Comparison of benefits of some optimizations

Model

Model

- Discrete time t, locations $\ell \in L$, small open economy, perfect foresight (for now)
- Each location has exogenous wage $w_{\ell t}$ and amenity $\alpha_{\ell t}$, exogenous rental-only housing stock $H_{\ell t}$ and equilibrium rent $\rho_{\ell t}$
- Atomistic households i have state $x_{it} \in X$,

$$x_{it} = (\underbrace{k_{it-1}}_{\text{wealth}}, \underbrace{z_{it}}_{\text{income type}}, \underbrace{\ell_{it-1}}_{\text{location}}, \underbrace{a_{it}}_{\text{age}}).$$

(Boundary conditions and some details omitted.)

Each period t, household i:

- Begins with wealth k_{it-1}, income type z_{it}, location ℓ_{it-1}, age a_{it}
- Receives i.i.d. Gumbel location preference shocks $\{\varepsilon_{i\ell t}\}$
- Chooses location ℓ_{it}, goods consumption c_{it}, and housing consumption h_{it}, s.t.

$$k_{it} \equiv (1+r)k_{it-1} + w_{\ell_{it}} z_{it} - c_{it} - \rho_{\ell_{it}} h_{it} \geq 0$$

- Receives utility,

$$u_{it} = \frac{\alpha_{\ell_{it}t}^{1-\eta}(c_{it}^{\rho} + \gamma h_{it}^{\rho})^{\frac{1-\eta}{\rho}} - 1}{1-\eta} - D_{\ell_{it-1},\ell_{it}} + \varepsilon_{i\ell_{it}t}$$

- Realizes Markov income type shock $z_{it+1} \sim \Gamma(z_{it})$
- Ages $a_{it+1} = a_{it} + 1$ or receives bequest utility

Household maximizes expected lifetime utility, exponentially discounted at rate β

High-Level Solution Decomposition

Let the beginning and end-of-period household value functions and state distributions be,

$$V_t^{\text{start}} : X \to \mathbb{R}, \quad V_t^{\text{end}} : X \to \mathbb{R}, \lambda_t^{\text{start}} : \mathcal{P}(X) \to \mathbb{R}, \quad \lambda_t^{\text{end}} : \mathcal{P}(X) \to \mathbb{R}.$$

Computationally, these are just arrays (for today). A solution consists of:

1. An intra-period household's problem solution:

$$\mathcal{H} : (V_t^{\text{end}}, \lambda_t^{\text{start}}, \{\rho_{\ell t}\}_\ell, \theta) \mapsto (V_t^{\text{start}}, \lambda_t^{\text{end}}, \text{Moments}_t)$$

2. A set of defining equation functions:

$$\begin{aligned}
\text{(Market Clearing)} \quad & \mathcal{E} : (\text{Moments}_t, \{H_{\ell t}\}) \mapsto \text{ExcessDemand}_t \\
\text{(Period Boundaries)} \quad & \mathcal{B} : (V_t^{\text{end}}, V_{t+1}^{\text{start}}, \lambda_t^{\text{end}}, \lambda_{t+1}^{\text{start}}) \mapsto \text{B}_t \\
\text{(Calibration)} \quad & \mathcal{C} : (\text{Moments}_t, \text{DataMoments}_t) \mapsto \text{MomentError}_t
\end{aligned}$$

3. A solver, $\mathcal{S} : (\mathcal{H}, \mathcal{E}, \mathcal{B}, \mathcal{C}) \mapsto \left(\{V_t^{\text{end}}, V_t^{\text{start}}, \lambda_t^{\text{end}}, \lambda_t^{\text{start}}, \rho_{\ell t}\}_t, \theta\right)$

I build models by taking these components and composing them.

Household Problem Decomposition

An intra-period household problem solution (IPHP) is separable into two functions,

$$\mathcal{H}^{\text{back}} :(V_t^{\text{end}}, \{\rho_{\ell t}\}_\ell, \theta) \mapsto (V_t^{\text{start}})$$
$$\mathcal{H}^{\text{forward}} :(V_t^{\text{end}}, \lambda_t^{\text{start}}, \{\rho_{\ell t}\}_\ell, \theta) \mapsto (\lambda_t^{\text{end}}, \text{Moments}_t).$$

Each can be further decomposed into "stages" which occur in succession:

Choose Location
Receive Income
Choose Consumption
Income Shock

Each stage s has

$$\mathcal{H}_s^{\text{back}} : (V_{st}^{\text{end}}, \{\rho_{\ell t}\}_\ell, \theta) \mapsto (V_{st}^{\text{start}})$$
$$\mathcal{H}_s^{\text{forward}} : (V_{s-1,t}^{\text{end}}, \lambda_{st}^{\text{start}}, \{\rho_{\ell t}\}_\ell, \theta) \mapsto (\lambda_{st}^{\text{end}}, \text{Moments}_{st})$$

where

$$V_{s-1,t}^{\text{end}} = V_{st}^{\text{start}}$$
$$\lambda_{st}^{\text{start}} = \lambda_{s-1,t}^{\text{end}}$$

Build model from pre-built, optimized stages

Household Problem Code

```
function solve_period!(prealloc, V_next, params)
    V_preshock = get_V_preshock(prealloc, V_next)

    V_consume = get_V_preconsume(V_preshock, prealloc)

    V_income = get_V_preincome(V_consume, prealloc, params)
    enforce_borrowing_constraint!(V_preincome, prealloc)

    V_premove = get_V_premove(V_preincome, prealloc, params)

    V_end = YOUR_CODE_HERE(V_premove, prealloc, params)
end
```

Comparison of Optimizations

Benchmark

- 1000 locations, 129 wealth states, 5 income types, 6 age groups = 3.87m gridpoints
- Single-thread CPU
- Language: Julia
- Strawman: Jeffrey, May 2023
- One evaluation of household problem
- Initial time: 218s (3m38s)

Low-Hanging Fruit

- **Initial**: 218s
- **Memory Preallocation**: 218s → 152s
- **(Almost) Automatic Multithreading**: 152s → 49s
- **32 Bit Precision**: 49s → 31.9s

Individual Stage Optimizations

Choose Location	25.2s → 9.78s → 0.053s
Receive Income	0.38s → 0.019s
Choose Consumption	0.498s → 0.025s
Income Shock	4.52s → 0.028s

Overall: 31.9s → 0.353s (= 0.126s listed stages + 0.228s other)

Individual Stage Optimizations

Choose Location

$$\text{Let} \qquad V^{\text{start}}_{ts\iota}(\ell) = V^{\text{start}}_{ts}(k_{\iota t-1}, z_{\iota t}, \ell, a_{\iota t}), \quad V^{\text{end}}_{ts\iota}(\ell) \text{ similar}$$

where ι indexes all household types up to location.

The i.i.d. Gumbel location preference shocks imply:

$$\exp\left(\psi V^{\text{start}}_{ts\iota}(\ell)\right) = \sum_{\ell'} \exp\left(\psi\left(V^{\text{end}}_{ts\iota}(\ell') - D_{\ell\ell'}\right)\right)$$

$$P(\ell' = \ell_0 \mid \ell) = \frac{\exp\left(\psi\left(V^{\text{end}}_{ts\iota}(\ell') - D_{\ell\ell'}\right)\right)}{\exp\left(\psi V^{\text{start}}_{ts\iota}(\ell)\right)}$$

$$\lambda^{\text{end}}_{ts\iota}(\ell) = \sum_{\ell'} P(\ell' = \ell \mid \ell)\lambda^{\text{start}}_{ts\iota}(\ell')$$

First optimization: Precompute $\exp\left(\psi V^{\text{start}}_{ts\iota}(\ell)\right)$, then $P(\ell' = \ell_0 \mid \ell)$, then $\lambda^{\text{end}}_{ts\iota}(\ell)$

Time: 25.2s $\rightarrow$ 9.78s

Choose Location

Second optimization: observe that (with $\otimes$ and $\oslash$ elementwise mult. and div.)

$$
\begin{aligned}
\widetilde{V}_{ts}^{\text{start}} &= D\widetilde{V}_{ts}^{\text{end}} \\
\Lambda_{ts}^{\text{end}} &= \widetilde{V}_{ts}^{\text{end}} \otimes \left(D'\Lambda_{ts}^{\text{start}} \oslash \widetilde{V}_{ts}^{\text{start}}\right) \\
\text{where matrices} \quad D_{\ell\ell'} &= \exp\left(-\psi D_{\ell\ell'}\right) \\
\left(\widetilde{V}_{ts}^{\text{start}}\right)_{\ell\iota} &= \exp\left(\psi V_{ts\iota}^{\text{start}}(\ell)\right) \\
\left(\widetilde{V}_{ts}^{\text{end}}\right)_{\ell\iota} &= \exp\left(\psi V_{ts\iota}^{\text{end}}(\ell)\right) \\
\left(\Lambda_{ts}^{\text{end}}\right)_{\ell\iota} &= \lambda_{ts\iota}^{\text{end}}(\ell)
\end{aligned}
$$

No matter the size of the state space, just two matrix multiplications!

Time: 9.78s $\rightarrow$ 0.053s

The Power of Matrix Multiplication

Why is matrix multiplication 200 faster than an explicit loop?

- Surprising algorithms exist to multiply two matrices in as little as $O(n^{2.371552})$ time
- Most CPUs have specialized hardware for matrix multiplication
- Pretty much exactly the same thing works for CES production functions, etc.
- Similar approach to optimizing income shocks, or any finite-state Markov process

Choose Consumption

- Strategy: Gridsearch
- If $MPC \geq 0$, then my optimal saving is between my wealth-neighbors'
- Don't need to search over entire axis!
- By "sharing" information between wealth-neighbors: $O(N^2) \rightarrow O(N \log N)$
- Incompatible with vectorization (Python, Matlab) but fast in Julia
- Similar approach for linear interpolation of many gridpoints

Outer Loop Optimization

- Currently, takes 396 iterations to solve for prices, using tatonnement. 146s total.
- With autodiff + LBFGS (fancy improvement over Newton's method), hope to get under $\sim$ 10s for steady state solution

Global Solution

- These “conventional” methods enable the global solution to my JMP model
- Global solution: a solver and boundary conditions equations on $V^{\text{start}}, V^{\text{end}}$ which take the IPHP *as given*
- A neural network is trained to predict V^{end}. Everything else is conventional
- In particular, no neural network used to approximate policy function

Next Steps

- Write papers!
- Create ensemble models – standard in climate science
- Package up stages and solver modules
- Write some tutorials for users
- Rewrite modules for GPU, cluster platforms

Conclusion

- Attempt to standardize *one* class of model solutions, and pre-write necessary modules
- "Conventional methods" have some life in them yet
- Hard to see how this becomes a paper per se, but already useful to me
- A bunch of "obvious" stuff with non-obvious power
- No one part is super exciting/novel. But their usefulness compounds exponentially