Obsolete Version

This document is an outdated version of "Strong Convergence and Dynamic Economic Models." I make this old draft public because (i) it better explains how to modify the nested fixed point (NFXP) and nested pseudo-likelihood (NPL) estimators to leverage strong convergence, and (ii) it provides additional Monte Carlo simulations.

Strong Convergence and the Estimation of Markov Decision Processes

Robert L. Bray*

Kellogg School of Management, Northwestern University

January 31, 2017

Abstract

The empirical likelihood of a Markov decision process depends only on the differenced value function. And the differenced value function depends only on the payoffs received until the underlying Markov chain reaches its stationary distribution. Thus, whereas the value function converges with Bellman contractions at the rate of cash flow discounting, the differenced value function—and hence the empirical likelihood—converges at the rate of cash flow discounting times the rate of Markov chain mixing (the spectral sub-radius of the state transition matrix). I use this strong convergence result to speed up Rust's (1987) nested fixed point (NFXP) and Aguirregabiria and Mira's (2002) nested pseudo-likelihood (NPL) estimators. The approach is especially useful when estimating high-frequency models.

Keywords: Markov decision process, dynamic discrete choice, nested fixed point, nested pseudo-likelihood, strong convergence.

*I would like to thank Victor Aguirregabiria, Achal Bassamboo, Thomas Bray, Seyed Emadi, Soheil Ghili, Arvind Magesan, John Rust, Dennis Zhang, and three anonymous referees for their helpful comments.

1 Introduction

This project stemmed from a bug in my code. I tried to implement Rust's (1987) nested fixed point (NFXP) dynamic program estimator, but I erred. I mistakenly changed the inner-loop stopping condition from value function convergence to policy function convergence—naïvely, I thought the two were equivalent. They were not. The policy functions converged sooner, leaving me with value functions that violated Bellman's equation. I fixed the bug, iterating the inner-loop until the value function reached its fixed point. But after implementing Rust's algorithm properly, my code took orders of magnitude longer to run. And the extra time didn't buying me anything: my utility estimates didn't change. In this manner, I accidentally found a faster way to estimate Markov decision processes.

My approach extends to models with multiple agents and continuous state and action spaces, but I will focus on the conical single-agent dynamic discrete choice problem. Here, an agent guides the evolution of a Markov chain in a utility-maximizing fashion with a sequence of discrete choices, and an empiricist aims to reverse engineer this agent's private utility function from its public actions. My estimators capitalize on three facts:

1. The empirical likelihood of a dynamic discrete choice problem depends only on the dynamic program's policy function, not its value function.

2. The policy function depends only on the value function's relative differences, not its absolute level.

3. The value function's relative differences converge faster than its level under Bellman contractions. This is called *strong convergence*.

Note, the differencing I refer to in the second point is not the differencing most are used to. I difference across states, not across actions. It is well known that we can normalize the utility of choosing action a_0 to zero, since $\arg\max_{a_t} u(a_t, x_t) = \arg\max_{a_t} u(a_t, x_t) -$

$u(a_0, x_t)$. It is less well known that we can normalize the value of being in state x_0 to zero, since $\arg\max_{a_t} u(a_t, x_t) + \beta\mathbb{E}(v(x_{t+1})|a_t, x_t) = \arg\max_{a_t} u(a_t, x_t) + \beta\mathbb{E}(v(x_{t+1}) - v(x_0)|a_t, x_t)$. It is these cross-state value differences, $v(x_{t+1}) - v(x_0)$, that experience strong convergence.

Strong convergence roughly implies that only the first 1% of Bellman contractions meaningfully affect the value function's shape; the latter 99% increase the value function rigidly. These uniform shifts don't change the value function's relative differences, and thus don't change the policy function, and thus don't change the empirical likelihood. Therefore, they are superfluous. My estimators disregard them.

My first estimator applies strong convergence to Rust's (1987) nested fixed point (NFXP) estimator; I call it strong nested fixed point (SNFXP). NFXP imposes the Bellman equation as a constraint, specifying that a Bellman contraction does not affect the value function; in contrast, SNFXP imposes the Bellman equation's first differences as a constraint, specifying that a Bellman contraction does not affect the value function's shape, only its level. NFXP calculates many value functions, but SNFXP calculates none. Accordingly, dynamic programs that require 458 processor hours to estimate with NFXP require only 21 processor hours with SNFXP.

My second estimator applies strong convergence to Aguirregabiria and Mira's (2002) nested pseudo-likelihood (NPL) estimator; I call it strong nested pseudo-likelihood (SNPL). NPL uses policy iteration, which calculates the value function under a given policy by solving a system of equations the size of the state space; in contrast, SNPL uses Morton's (1971) relative policy iteration, which solves the *differenced* value function under a given policy by applying successive approximations until a Bellman contraction does not affect the value function's shape, only its level. NPL calculates many value functions, but SNPL calculates none. Accordingly, dynamic programs that require 740 processor hours to solve with NPL require only 72 processor hours with SNPL.

Strong convergence is a timely topic, given the deluge of high-frequency data coming online. Cheap sensors now give us visibility to fast-changing environments, such as Uber's network of drivers. But we cannot rely on financial discounting to make high-frequency models tractable, since the per-period discount rate goes to one as the period length goes to zero. Fortunately, strong convergence can be a substitute for discounting in this case; strong convergence empowers the Bellman contraction in the high-frequency era.

2 Illustration

Rust's (1987) canonical engine replacement problem demonstrates the power of strong convergence. Mechanic Harold Zurcher faces a dynamic program with one state variable: engine mileage x_t. Each month Zurcher decides whether or not to replace his bus engine. He follows the replacement schedule that minimizes the expected net present value of his operating costs, discounting at rate β. In month t, an engine with mileage x_t costs $x_t/10,000 - e_{t1}$ to operate and $10 - e_{t0}$ to replace, where e_{t1} and e_{t0} are $i.i.d.$ standard Gumbel shocks. If he replaces the engine, the odometer goes to zero; otherwise it increases by 10,000 miles.

I solve this dynamic program with value iteration. As the algorithm progresses, I track my assessment of (i) the value function, Zurcher's expected discounted operating costs at each state; (ii) the differenced value function, the value function minus its first element; and (iii) the policy function, Zurcher's engine replacement probability at each state (unconditional on the Gumbel shocks). Figure 1 depicts the magnitude of these function changes for each iteration of the algorithm. The policy function and differenced value function converge within 32 value iteration steps, whereas the total value function converges in upwards of 100,000. Figure 2 illustrates that the value function moves in lock step after 32 value iterations: the expected cost of operating a new engine increases from 88.42 to

179.74 between iterations 32 and 64, but the expected discounted cost difference between operating a new engine and one with 100,000 miles only changes from $95.44 - 88.42 = 7.02$ to $186.75 - 179.74 = 7.01$.

Figure 3 demonstrates why the differenced value function stabilizes faster than total value function. The difference between the value function evaluated at $x_t = 20,000$ and at $x_t = 0$ is the difference in the expected discounted costs starting from $x_t = 20,000$ and from $x_t = 0$. Under the optimal policy, Zurcher's engine mileage follows an ergodic Markov chain that regresses to its stationary distribution after 32 state transitions: variable x_{t+32} has the same distribution from both $x_t = 20,000$ and $x_t = 0$. Accordingly, the expectation of costs incurred after month $t + 32$ is the same under both initial states, and thus the difference between the value function evaluated at $x_t = 20,000$ and $x_t = 0$ depends only on the costs faced between months t and $t + 31$. Calculating these 32 cash flows requires a mere 32 Bellman contractions.

3 Model

3.1 Dynamic Program

An agent follows a Markov decision process. In a given period, it chooses action $a \in \mathbb{a} = \{a_1, \cdots, a_{\bar{a}}\}$ in response to state variables $x \in \mathbb{x} = \{x_1, \cdots, x_{\bar{x}}\}$ and $e = \{e(a) \in \mathbb{R} | a \in \mathbb{a}\}$. I observe a and x, but not e, which serves as a statistical error term. Taking action a in state $\{x, e\}$ yields utility $u(a, x) = z(a, x)'\theta + e(a)$, where $z(a, x)$ is a length-$\bar{z}$ vector of factors and θ is a length-$\bar{z}$ vector of parameters. I observe z, but not θ, which I seek to estimate. The probability density function of tomorrow's state variables given today's action a and state $\{x, e\}$ is $f(x', e'|a, x, e) = f^e(e'|x')f^x(x'|x, a)$, where f^e is twice differentiable, specifies a finite first moment, and has full support over the real line. Since f^e has full support, all actions have strictly positive probability in all states, and thus the state variables' Markov

Figure 1: Function Changes with Value Iteration Step

These plots depict, for each step of the value iteration algorithm, the magnitudes of the value function, differenced value function, and policy function changes (measured under the ℓ_∞ norm). The value function provides expected future discounted costs; the differenced value function is the value function less its first element; and the policy function gives the probability of replacing the engine, unconditional on the Gumbel cost shocks. Parameter β is the cash flow discount factor.

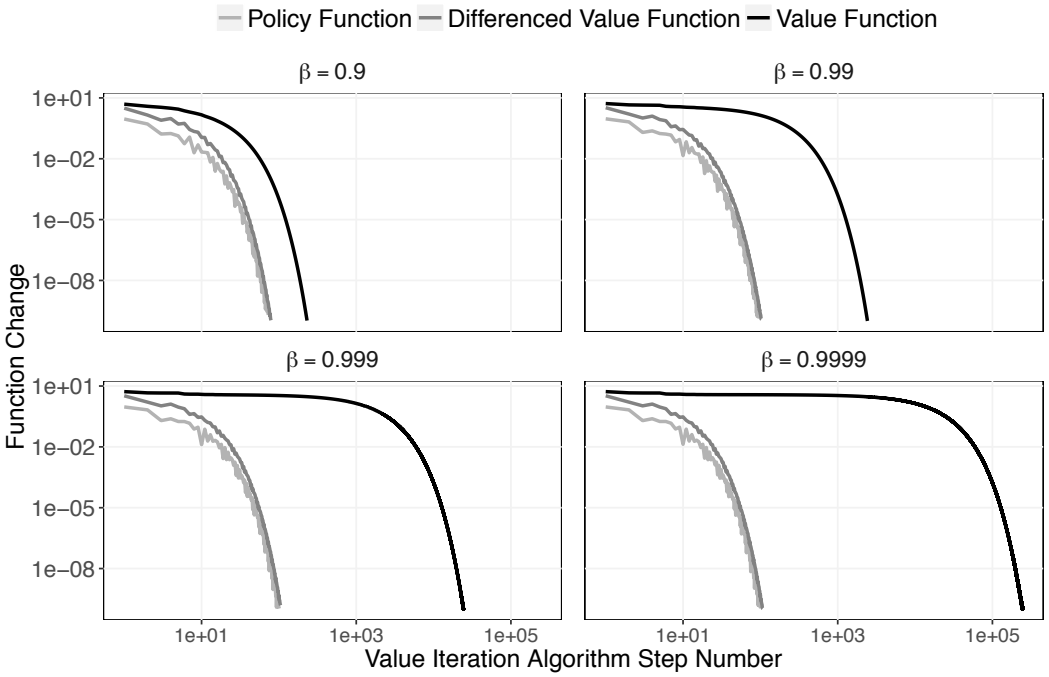


Figure 2: Value Function with Value Iteration Step

These plots depict my assessment of the value function at various steps of the value iteration algorithm. (For illustrative purposes, I set the elements of the initial value function to *i.i.d.* standard normal random variables.) The value function's shape converges by iteration 32, but the value function's level does not.

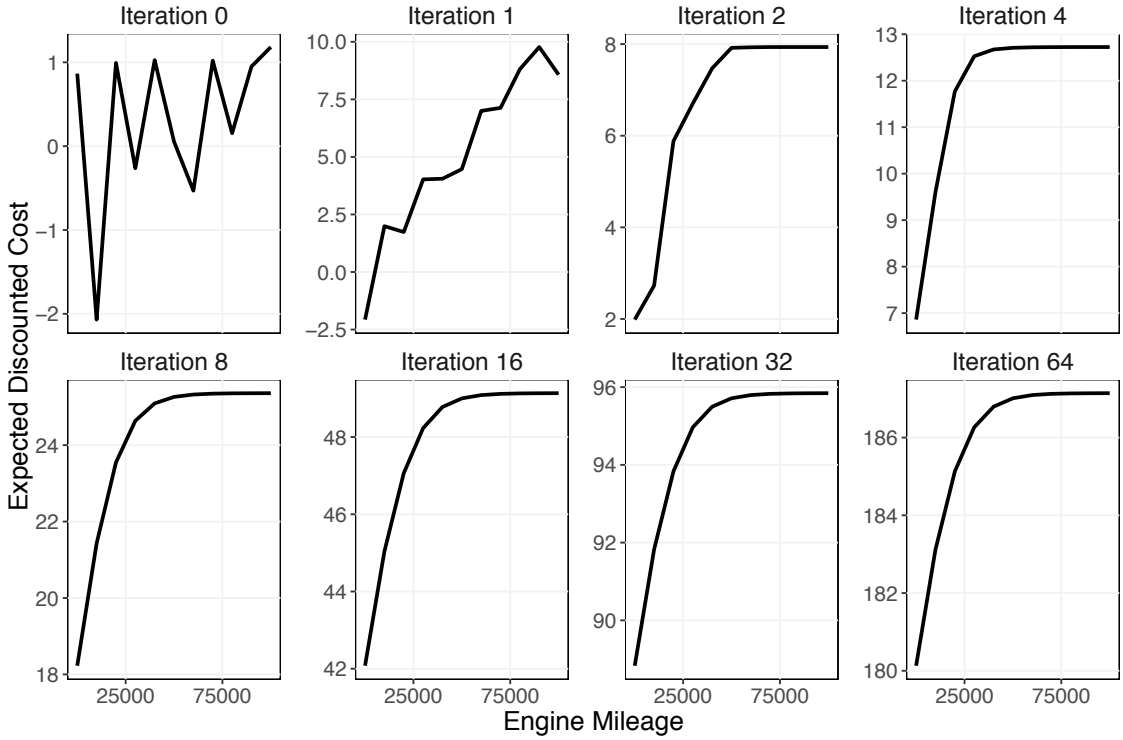
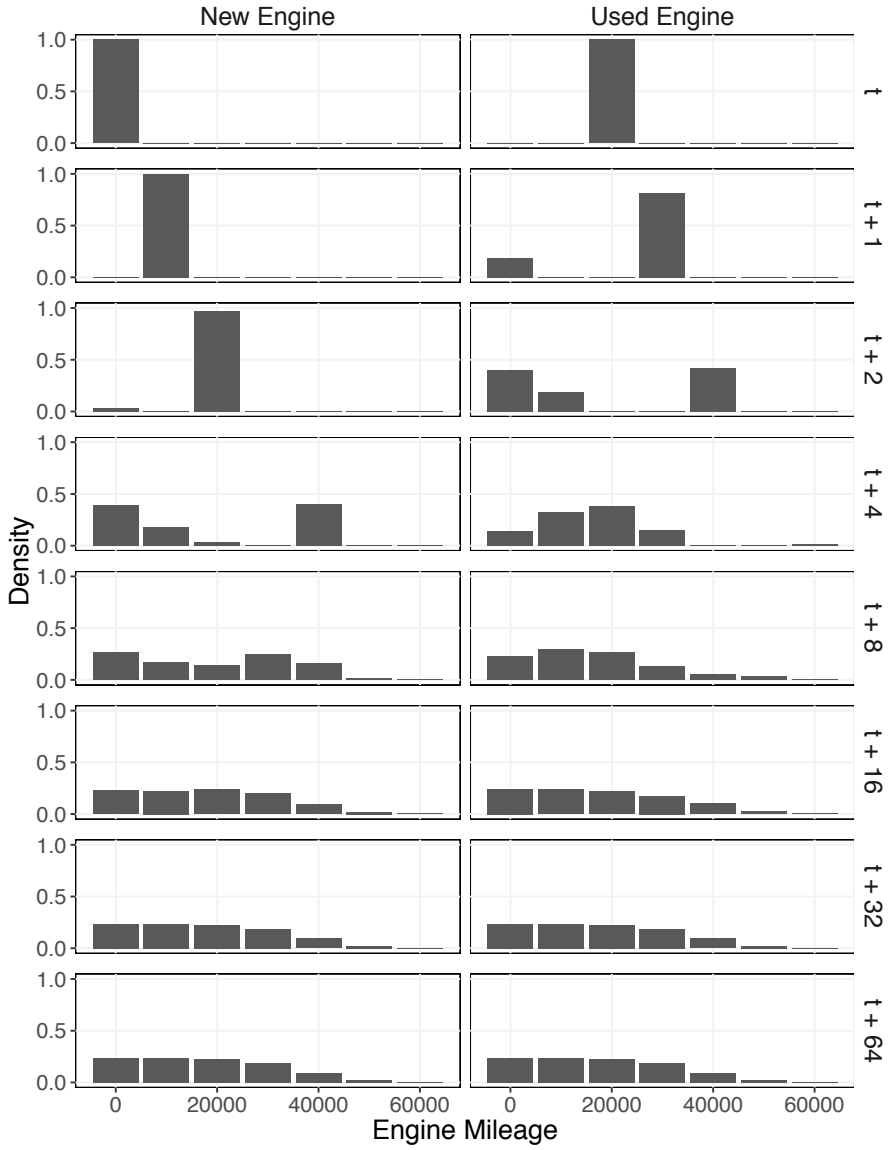


Figure 3: State Space Distribution with Value Iteration Step

These histograms depict the distribution of $x_{t+\tau}$ from the perspective of period t. The left-hand plots give the distribution conditional on $x_t = 0$, and the right-hand plots give the distribution conditional on $x_t = 20,000$. From both starting conditions, the distribution of x_{t+32} equals the Markov chain's stationary distribution.



chain is ergodic under all policies if it is ergodic under at least one policy; I suppose the state variables' Markov chain is ergodic under at least one policy.[1]

The agent chooses the action that maximizes its infinite-horizon expected discounted utility, under discount rate $\beta < 1$. A Bellman equation recursively defines its value function:

$$\breve{v}(x,e) = \max_{a \in \mathbb{a}} u(a,x) + \beta \int \sum_{x'} f(x',e'|a,x,e)\breve{v}(x',e')de'.$$

More useful, however, is the integrated value function, the expected value conditional on the observable state but not the unobservable state:

$$\begin{aligned} v(x) &= \int \breve{v}(x,e)de \\ &= \int \max_{a \in \mathbb{a}} \big(s(a,x) + e(a)\big) f^e(e|x)de, \\ \text{where} \quad s(a,x) &= z(a,x)'\theta + \beta \sum_{x'} f^x(x'|a,x)v(x'). \end{aligned}$$

Choice-specific value function $s(a,x)$ gives the expected value of choosing action a in state x, net the error term.

The conditional choice probability (CCP) of the agent choosing action a, given observable state x, is

$$p(a|x) = \int \mathbb{1}\{a = \arg\max_{\breve{a} \in \mathbb{a}} s(\breve{a},x) + e(\breve{a})\} f^e(e|x)de.$$

Hotz and Miller (1993) showed that these CCPs characterize the agent's policy by proving that there exists function g that satisfies $\mathbb{E}[e(a)|a,x] = g(a,p(x))$, where $p(x) = [p(a_1|x_1), \cdots, p(a_{\bar{a}}|x)]$. For example, when the e errors are $i.i.d.$, mean-zero Gumbels, $g(a,p(x)) = -\log(p(a|x))$. With this, and the law of iterated expectations, they derived

[1] My approach can accommodate transitory states as long as they do not feed into multiple communicating classes. The vast majority of economic models fit this specification.

alternate value function representation

$$\begin{aligned} v(x) &= \sum_{a \in \mathbb{a}} p(a|x)(s(a,x) + g(a, p(x))) \\ &= \max_{\breve{p} \in \Xi} \sum_{a \in \mathbb{a}} \breve{p}(a)(s(a,x) + g(a, \breve{p})), \end{aligned}$$

where simplex Ξ is the set of length-$\bar{a}$ probability vectors. This equation reformulates the problem: while the agent actually chooses action a in response to states x and e, I can pretend it chooses CCP p in response to state x. Casing the CCP as the policy function eliminates the unobserved state and transforms the optimization problem from discrete to continuous.

3.2 Vectorization

I now vectorize the model variables by the span of the state space.

1. V is the length-$\bar{x}$ vector with i^{th} element $v(x_i)$. It denotes the value function.
2. P_a is the length-$\bar{x}$ vector with i^{th} element $p(a|x_i)$. It denotes the action-a CCPs.
3. $P = [P'_{a_1}, \cdots, P'_{a_{\bar{a}}}]'$ is a length-$\bar{a}\bar{x}$ vector that comprises all the CCPs. It denotes the full policy function.
4. $G_a(P)$ is the length-$\bar{x}$ vector with i^{th} element $g(a, p(x_i))$. It denotes the expected error terms, conditional on action a.
5. $G(P) = \sum_{a \in \mathbb{a}} \text{diag}(P_a) G_a(P)$ is the length-$\bar{x}$ vector with i^{th} element $\sum_{a \in \mathbb{a}} p(a|x_i) g(a, p(x_i))$. It denotes the expected error terms, unconditional on the action.
6. Z_a is the $\bar{x} \times \bar{z}$ matrix with i^{th} row $z(a, x_i)$. It denotes the utility statistics, conditional on action a.

7. $Z(P) = \sum_{a \in \mathbb{a}} \text{diag}(P_a) Z_a$ is the $\bar{x} \times \bar{z}$ matrix with i^{th} row $\sum_{a \in \mathbb{a}} p(a|x_i) z(a, x_i)$. It denotes the expected utility statistics, unconditional on the action.

8. $U(P) = Z(P)\theta + G(P)$ is the length-$\bar{x}$ vector with i^{th} element $\sum_{a \in \mathbb{a}} p(a|x_i) \big(z(a, x_i)'\theta + g(a, p(x_i)) \big)$. It denotes the expected utilities, unconditional on the action.

9. F_a is the $\bar{x} \times \bar{x}$ matrix with ij^{th} element $f^x(x_j|a, x_i)$. It denotes the joint state transition probabilities, conditional on action a.

10. $F(P) = \sum_{a \in \mathbb{a}} \text{diag}(P_a) F_a$ is the $\bar{x} \times \bar{x}$ matrix with ij^{th} element $\sum_{a \in \mathbb{a}} p(a|x_i) f^x(x_j|a, x_i)$. It denotes the expected state transition probabilities, unconditional on the action.

3.3 Operators

I now define several dynamic programming operators:

1. The Bellman contraction function is γ. It maps policies and values to values: the value today is $\gamma(\breve{V}, \breve{P}) = U(\breve{P}) + \beta F(\breve{P}) \breve{V}$ when the policy today is $\breve{P}$ and the value tomorrow is $\breve{V}$.

2. The policy improvement operator is π. It maps values to policies: the optimal policy today is $\pi \breve{V} = \arg\max_{\breve{P} \in \Xi \times \mathbb{x}} \gamma(\breve{V}, \breve{P})$ when the value tomorrow is $\breve{V}$. When the errors are *i.i.d.* Gumbels, $\pi \breve{V} = \left\{ \frac{\exp(R(a) + \beta F_a \breve{V})}{\sum_{\breve{a}} \exp(R(\breve{a}) + \beta F(\breve{a}) \breve{V})} \mid a \in \mathbb{a} \right\}$, with the exponents and division evaluated element-wise.

3. The value iteration operator is ν. It maps values to values: the value today is $\nu \breve{V} = \gamma(\breve{V}, \pi \breve{V})$ when the value tomorrow is $\breve{V}$. When the errors are *i.i.d.*, mean-zero Gumbels, $\nu \breve{V} = \log \left(\sum_a \exp \left(R(a) + \beta F_a \breve{V} \right) \right)$, with the exponents and logarithm evaluated element-wise.

4. The policy valuation operator is η. It maps policies to values: the value of following policy $\breve{P}$ forever is $\eta\breve{P} = \left(I - \beta F(\breve{P})\right)^{-1} U(\breve{P})$. This equation stems from isolating $\breve{V}$ in fixed point $\gamma(\breve{V}, \breve{P}) = \breve{V}$.

5. The policy iteration operator is $\rho = \pi\eta$. It maps policies to policies: the optimal policy today is $\rho\breve{P}$, given policy $\breve{P}$ from tomorrow onwards. When the errors are *i.i.d.* Gumbels, $\rho\breve{P} = \left\{ \frac{\exp(R(a)+\beta F_a \eta\breve{P})}{\sum_{\breve{a}} \exp(R(\breve{a})+\beta F(\breve{a})\eta\breve{P})} \;\middle|\; a \in \mathbb{a} \right\}$, with the exponents and division evaluated element-wise.

6. The difference operator is $\Delta = I - \iota e_1'$, where ι is a length-$\bar{x}$ vector of ones and e_1 is a length-$\bar{x}$ unit vector indicating the first position. Premultiplying a vector by Δ renormalizes it, setting its first element to zero. For example, the i^{th} element of ΔV is $v(x_i) - v(x_1)$.

4 Solution

4.1 Standard Algorithms

I now present two standard solution algorithms: value iteration and policy iteration. Both use convergence threshold $\delta = \frac{\epsilon(1-\beta)}{2\beta}$ to produce an ϵ-optimal policy, a policy that yields within ϵ of V when followed forever (Puterman, 2014, 161).

The value iteration algorithm applies the value iteration operator to convergence:

1. Set $t = 0$ and $V_t = 0$.

2. Increment t.

3. Set $V_t = \nu V_{t-1}$.

4. If $\|V_t - V_{t-1}\| \geq \delta$ go to 2.

5. Return πV_t.

And the policy iteration algorithm applies the policy iteration operator to convergence:

1. Set $t = 0$ and $P_t = \pi 0$.

2. Increment t.

3. Set $P_t = \rho P_{t-1}$.

4. If $\|\nu\eta P_{t-1} - \eta P_{t-1}\| \geq \delta$ go to 2.

5. Return P_t.

Policy iteration steps accomplish more than value iteration steps, but they are slower because calculating $\eta\breve{P}$ is difficult. The naïve approach—computing $(I - \beta F(\breve{P}))^{-1}$ and post-multiplying by $U(\breve{P})$—is cumbersome and numerically unstable. A better way to evaluate $\eta\breve{P}$ is to apply the LU decomposition or Gaussian elimination to the $\big(I - \beta F(\breve{P})\big)\eta\breve{P} = U(\breve{P})$ system of equations.

Alternatively, I can approximate $\eta\breve{P}$ with a series of Bellman contractions. The value of following policy $\breve{P}$ forever is $\eta\breve{P} = \big(I - \beta F(\breve{P})\big)^{-1} U(\breve{P}) = \sum_{t=0}^{\infty} \beta^t F(\breve{P})^t U(\breve{P})$. Define $\eta_\tau \breve{P} = \sum_{t=0}^{\tau} \beta^t F(\breve{P})^t U(\breve{P})$ as a finite approximation of this infinite sum. Operator η_τ returns the expected discounted utility amassed over the next τ periods under a given policy. Approximation error $\eta\breve{P} - \eta_\tau\breve{P} = \sum_{t=\tau+1}^{\infty} \beta^t F(\breve{P})^t U(\breve{P}) = \beta^{\tau+1} F(\breve{P})^{\tau+1} \sum_{t=0}^{\infty} \beta^t F(\breve{P})^t U(\breve{P}) = \beta^{\tau+1} F(\breve{P})^{\tau+1} \eta\breve{P}$ goes to zero as τ grows. So I can substitute $\eta_t\breve{P}$, with large τ, for $\eta\breve{P}$. And since $\eta_0\breve{P} = U(\breve{P})$ and $\eta_\tau\breve{P} = \gamma(\eta_{\tau-1}\breve{P}, \breve{P})$, I can evaluate $\eta_\tau\breve{P}$ with τ Bellman contractions.

4.2 Relative Algorithms

I will now present versions of value iteration and policy iteration that correspond not to the value function V, but to its relative differences $\bar{V} = \Delta V$. This relative approach

exploits two identities. First, an outer differencing makes an inner differencing redundant: $\Delta\gamma(\Delta\breve{V}, \breve{P}) = \Delta\gamma(\breve{V}, \breve{P})$.[2] And second, the policy function depends on the value function's shape but not its level: $\pi\Delta = \pi$.[3] These properties imply that the differenced value function identifies the optimal policy (i.e., $\pi\bar{V} = \pi\Delta V = \pi V = P$) and that repeatedly applying relative value function operator $\bar{\nu} = \Delta\nu$ identifies the differenced value function (i.e., $\lim_{\tau\to\infty} \bar{\nu}^{\tau} 0 = \bar{V}$).[4]

Leveraging this insight, the relative value iteration algorithm applies the relative value iteration operator to convergence:[5]

1. Set $t = 0$ and $\bar{V}_t = 0$.

2. Increment t.

3. Set $\bar{V}_t = \bar{\nu}\bar{V}_{t-1}$.

4. If $\|\bar{V}_t - \bar{V}_{t-1}\| \geq \delta$ go to 2.

5. Return $\pi\bar{V}_t$.

Analogously define relative policy iteration operator: $\bar{\rho} = \pi\bar{\eta}$, where $\bar{\eta} = \Delta\eta$. The relative policy iteration algorithm applies the relative policy iteration operator to convergence:[6]

1. Set $t = 0$ and $P_t = \pi 0$.

[2] First, $\Delta^2 = \Delta$, since Δ is a projection matrix. Second, for any Markovian matrix F, $F\iota = \iota$, and thus $F(I - \Delta) = F\iota e_1' = \iota e_1' = I - \Delta$. Third, $\Delta F(I - \Delta) = \Delta(I - \Delta) = 0$, which implies $\Delta F = \Delta F \Delta$, and thus $\Delta\gamma(\Delta\breve{V}, \breve{P}) = \Delta U(\breve{P}) + \beta\Delta F(\breve{P})\Delta\breve{V} = \Delta U(\breve{P}) + \beta\Delta F(\breve{P})\breve{V} = \Delta\gamma(\breve{V}, \breve{P})$.

[3] Footnote 2 shows that $I - \Delta = F(I - \Delta)$, which implies $\pi\Delta\breve{V} = \arg\max_{\breve{P}\in\Xi\times\mathbb{X}} U(\breve{P}) + \beta F(\breve{P})\Delta\breve{V} = \arg\max_{\breve{P}\in\Xi\times\mathbb{X}} U(\breve{P}) + \beta F(\breve{P})\Delta\breve{V} + \beta(I - \Delta)\breve{V} = \arg\max_{\breve{P}\in\Xi\times\mathbb{X}} U(\breve{P}) + \beta F(\breve{P})\Delta\breve{V} + \beta F(\breve{P})(I - \Delta)\breve{V} = \arg\max_{\breve{P}\in\Xi\times\mathbb{X}} U(\breve{P}) + \beta F(\breve{P})\breve{V} = \pi\breve{V}$ for all $\breve{V}$, and thus $\pi\Delta = \pi$.

[4] For all $\breve{V}$, $\bar{\nu}\Delta\breve{V} = \Delta\gamma(\Delta\breve{V}, \pi\Delta\breve{V}) = \Delta\gamma(\breve{V}, \pi\breve{V}) = \bar{\nu}\breve{V}$, and thus $\bar{\nu}\Delta = \bar{\nu}$. By induction this implies $\bar{\nu}^{\tau} = \Delta\nu^{\tau}$, and thus $\lim_{\tau\to\infty} \bar{\nu}^{\tau}\breve{V} = \lim_{\tau\to\infty} \Delta\nu^{\tau}\breve{V} = \Delta V = \bar{V}$, for all $\breve{V}$.

[5] This algorithm also yields an ϵ-optimal policy (Puterman, 2014, 205).

[6] This algorithm's stopping condition is equivalent to relative value iteration's stopping condition, with $\bar{V}_{\tau-1} = \bar{\eta}P_{\tau-1}$ and $\bar{V}_{\tau} = \bar{\nu}\bar{\eta}P_{\tau-1}$. Thus, since relative value iteration yields an ϵ-optimal policy, this algorithm does as well.

2. Increment t.

3. Set $P_t = \bar{\rho} P_{t-1}$.

4. If $\|\bar{\nu}\bar{\eta} P_{t-1} - \bar{\eta} P_{t-1}\| \geq \delta$ go to 2.

5. Return P_t.

Since $\bar{\rho} = \pi\bar{\eta} = \pi\Delta\eta = \pi\eta = \rho$, relative policy iteration appears equivalent to policy iteration. But it is not, because $\bar{\eta}\breve{P}$ is more amenable to Bellman contraction approximation than $\eta\breve{P}$. That is, $\bar{\eta}_\tau\breve{P} = \Delta\eta_\tau\breve{P}$ converges to $\bar{\eta}\breve{P} = \Delta\eta\breve{P}$ faster than $\eta_\tau\breve{P}$ converges to $\eta\breve{P}$. Morton (1971) and Morton and Wecker (1977) dub this *strong convergence.*

5 Strong Convergence

Since $F(\breve{P})$ is a regular stochastic matrix, (i) its largest eigenvalue is one, (ii) its largest eigenvalue corresponds to eigenvector ι, and (iii) its second-largest eigenvalue is less than one (Puterman, 2014, 595). I will call this subdominant eigenvalue $\lambda(\breve{P})$. Note that approximation error $\eta\breve{P} - \eta_\tau\breve{P} = \beta^{\tau+1}F(\breve{P})^{\tau+1}\eta\breve{P}$ decreases with τ at rate β times the largest eigenvalue of $F(\breve{P})$. Since this eigenvalue is one, $\|\eta\breve{P} - \eta_\tau\breve{P}\|$ is $O(\beta^\tau)$ as $\tau \to \infty$. In contrast, differenced approximation error $\bar{\eta}\breve{P} - \bar{\eta}_\tau\breve{P} = \Delta(\eta\breve{P} - \eta_\tau\breve{P}) = \beta^{\tau+1}\Delta F(\breve{P})^{\tau+1}\eta\breve{P}$ decreases with τ at rate β times the largest eigenvalue of $F(\breve{P})$ whose eigenvector is not in the kernel of Δ. Since the kernel of Δ is the span of ι, and ι is the eigenvector corresponding to $F(\breve{P})$'s largest eigenvalue, $\|\bar{\eta}\breve{P} - \bar{\eta}_\tau\breve{P}\|$ is $O(\beta^\tau\lambda(\breve{P})^\tau)$ as $\tau \to \infty$. The differenced value function, therefore, is easier to approximate.

The intuition is straightforward: the distribution of the state t periods hence depends on the convolution of t state transitions. These convolutions scramble the state variables, returning them to steady state. Eigenvalue $\lambda(\breve{P})$ parameterizes this rate of mixing—smaller $\lambda(\breve{P})$ begets faster blending. The number of transitions required for a Markov

chain to return to its stationary distribution is called the *mixing time* (Levin et al., 2009). We can decompose the value function into expected payments received before the mixing time plus expected payments received after. But payments received after the mixing time contribute equally to each state's value, shifting the value function upwards in a uniform fashion. These uniform shifts wash out upon differencing. Thus, whereas the value function depends on all future payoffs not discounted to irrelevance, the differenced value function depends only on payoffs received before the mixing time. Strong convergence limits the set of cash flows we must consider.

To exploit strong convergence, Morton (1971) developed the relative policy iteration algorithm. Whereas policy iteration calculates $\eta\breve{P}$ by iterating $\eta_\tau\breve{P} = \gamma(\eta_{\tau-1}\breve{P}, \breve{P})$ to convergence (or by solving system of equations $(I - \beta F(\breve{P}))\eta\breve{P} = U(\breve{P})$ for $\eta\breve{P}$), relative policy iteration calculates $\bar{\eta}\breve{P}$ by iterating $\bar{\eta}_\tau\breve{P} = \Delta\gamma(\bar{\eta}_{\tau-1}\breve{P}, \breve{P})$ to convergence. This is a faster iterative scheme. Morton and Wecker (1977) extended this logic to relative value iteration, proving that $\|\bar{V} - \bar{\nu}^\tau 0\|$ is $O(\beta^\tau \lambda(P)^\tau)$ as $\tau \to \infty$, while $\|V - \nu^\tau 0\|$ is $O(\beta^\tau)$ as $\tau \to \infty$.

6 Estimators

6.1 Overview

I observe a sequence of states x and corresponding actions a. I store these data in $\bar{a}\bar{x}$-dimensional vector $N = [n(a_1, x_1), \cdots, n(a_{\bar{a}}, x_{\bar{x}})]'$, where $n(a, x)$ denotes the number of action-a, state-x observations in my dataset. Following Rust (1994, 3109), I take β and f^x as given (the state transition probabilities are estimable in reduced form). I will estimate utility vector θ—which resides in compact set $\Theta \in \mathbb{R}^M$—by maximizing the conditional

log-likelihood of the observed actions, given the observed states

$$\widehat{\theta} = \arg\max_{\breve{\theta}} N' \log(P(\breve{\theta})),$$

where $P(\breve{\theta})$ is the optimal policy under utility vector $\breve{\theta}$, and the logarithm evaluates element by element (as it does whenever applied to a vector).

I will present four algorithms that maximize this likelihood: nested fixed point (NFXP), strong nested fixed point (SNFXP), nested pseudo-likelihood (NPL), and strong nested pseudo-likelihood (SNPL). Since they maximize the same likelihood, all four estimators share the same identification and asymptotic properties. Indeed, SNFXP and SNPL are just NFXP and NPL with their value iteration and policy iteration steps replaced with relative value iteration and relative policy iteration steps. Since the policy function responds to relative value iteration and relative policy iteration in the same way it responds to value iteration and policy iteration, SNFXP always yields the same estimate as NFXP and SNPL always yields the same estimate as NPL. However, SNFXP and SNPL identify these estimates with less computation.

6.2 Nested Fixed Point

Rust's (1987) NFXP estimation algorithm proceeds as follows:

1. Define $P(\breve{\theta})$ as the output of the following subroutine:
   (a) Set $t = 0$ and $V_t = 0$.
   (b) Increment t.
   (c) Set $V_t = \nu_{\breve{\theta}} V_{t-1}$.
   (d) If $\|V_t - V_{t-1}\| \geq \delta_1$ or $\left| \frac{\|V_t - V_{t-1}\|}{\|V_{t-1} - V_{t-2}\|} - \beta \right| \geq \delta_2$ go to 1b.
   (e) Set $P_t = \pi_{\breve{\theta}} V_t$.

(f) Increment t.

(g) Set $P_t = \rho_{\breve{\theta}} P_{t-1}$.

(h) If $\|\nu_{\breve{\theta}}\eta_{\breve{\theta}}P_{t-1} - \eta_{\breve{\theta}}P_{t-1}\| \geq \delta$ go to 1f.

(i) Return P_t.

2. Return $\arg\max_{\breve{\theta}} N' \log(P(\breve{\theta}))$.

This algorithm uses both value iteration (in Steps 1b to 1d) and policy iteration (in Steps 1f to 1h). The policy iteration steps can calculate vector $\eta\breve{P}$ either by solving system of equations $(I - \beta F(\breve{P}))\eta\breve{P} = U(\breve{P})$ or by iterating $\eta_\tau\breve{P} = \gamma(\eta_{\tau-1}\breve{P}, \breve{P})$ to convergence.

6.3 Strong Nested Fixed Point

Rust's (2000, 18) NFXP estimator is predicated on the assumption "that one has to compute the fixed point $[V = \nu V]$ in order to evaluate the likelihood function." But this is incorrect. The empirical likelihood only depends on the value function's relative differences, not its absolute level. Therefor, I replace the value iteration and policy iteration steps, which solve the value function, with relative value iteration and relative policy iteration steps, which solve the differenced value function. This creates the SNFXP estimator:

1. Define $P(\breve{\theta})$ as the output of the following subroutine:

(a) Set $t = 0$ and $\bar{V}_t = 0$.

(b) Increment t.

(c) Set $\bar{V}_t = \bar{\nu}_{\breve{\theta}}\bar{V}_{t-1}$.

(d) If $\|\bar{V}_t - \bar{V}_{t-1}\| \geq \delta_1$ or $\left|\frac{\|\bar{V}_t - \bar{V}_{t-1}\|}{\|\bar{V}_{t-1} - \bar{V}_{t-2}\|} - \beta\right| \geq \delta_2$ go to 1b.

(e) Set $P_t = \pi_{\breve{\theta}}\bar{V}_t$.

(f) Increment t.

(g) Set $P_t = \bar{\rho}_{\breve{\theta}} P_{t-1}$.

(h) If $\|\bar{\nu}_{\breve{\theta}} \bar{\eta}_{\breve{\theta}} P_{t-1} - \bar{\eta}_{\breve{\theta}} P_{t-1}\| \geq \delta$ go to 1f.

(i) Return P_t.

2. Return $\arg\max_{\breve{\theta}} N' \log(P(\breve{\theta}))$.

Since Step 2 of each algorithm is the same, NFXP and SNFXP search the parameter space in the same way and yield the same estimates. However, by circumventing the solution of V, SNFXP evaluates this objective more quickly. First, satisfying Condition 1d requires fewer Bellman contractions under SNFXP than NFXP. And second, implementing policy iteration Step 1g is easier under SNFXP than NFXP, because evaluating $\bar{\eta}\breve{P}$ requires fewer Bellman contractions than evaluating $\eta\breve{P}$ (assuming $\eta\breve{P}$ is not solved via system of equations).

This implementation of SNFX, however, inherits one of NFXP's problems: the difficultly in determining when to switch from value iteration to policy iteration. As expressed above, this issue amounts to choosing fine-tuning parameters δ_1 and δ_2. But Condition 1d is actually an abridged version of Rust's (2000) complex decision rule, which spans three pages and comprises a series of ad hoc conditions, such as "Case 3: if ftol is too small, say less than 10^{-7}, then if the last fixed point hit the upper limit of cstp iterations before switching to Newton Kantorovich iterations, then I decrease cstp by 10 and set minstp=2." Moreover, Rust (2000, 30) "makes no claim that these heuristics for adaptively determining the time to switch from contraction iterations to Newton Kantorovich iterations are in any sense 'optimal.' [And he] encourage[s] you to experiment with your own heuristics to get faster execution times." Iskhakov et al. (2015, 7) concur, explaining how this approach requires "devoting time to the special-purpose programming necessary to implement [NFXP]."

I can obviate these difficulties by using relative value iteration throughout for a simpler

SNFXP implementation:

1. Define $P(\breve{\theta})$ as the output of the following subroutine:
   (a) Set $t = 0$ and $\bar{V}_t = 0$.
   (b) Increment t.
   (c) Set $\bar{V}_t = \bar{\nu}_{\breve{\theta}} \bar{V}_{t-1}$.
   (d) If $\|\bar{V}_t - \bar{V}_{t-1}\| \geq \delta$ go to 1b.
   (e) Return $\pi_{\breve{\theta}} \bar{V}_t$.
2. Return $\arg\max_{\breve{\theta}} N' \log(P(\breve{\theta}))$.

This algorithm applies Bellman contractions until the differenced value function converges. It then stops. It never calculates the true value function because the empirical likelihood does not need it: if a Bellman contraction does not change the value function's curvature, then the policy is optimal and the likelihood true.[7]

6.4 Nested Pseudo-Likelihood

Aguirregabiria and Mira's (2002) NPL algorithm solves a series of problems linked by policy iteration steps. NPL does not evaluate the empirical likelihood perfectly for each parameter vector; instead, it only approximates the empirical likelihood well enough to identify a better parameter vector.

At a high level, the NPL estimation algorithm proceeds as follows:

1. Create reduced form CCP estimates $\widehat{P}$.
2. Set $t = 0$ and $P_t = \widehat{P}$.

[7] Note, since $\bar{\nu}^\tau = \Delta\nu^\tau$ (see Footnote 4), Step 1 of this algorithm is equivalent to iterating $V_t = \nu_{\breve{\theta}} V_{t-1}$ until stopping condition $\|\Delta(V_t - V_{t-1})\| < \delta$.

3. Increment t.

4. Set $\theta_t = \arg\max_{\breve{\theta}} N' \log(\rho_{\breve{\theta}} P_{t-1})$.

5. Set $P_t = \rho_{\theta_t} P_{t-1}$.

6. If $\|\nu_{\theta_t} \eta_{\theta_t} P_{t-1} - \eta_{\theta_t} P_{t-1}\| \geq \delta$ go to 3.

7. Return θ_t.

Note, implementing Step 4 requires evaluating $\rho_{\breve{\theta}} \breve{P}$ under many $\breve{\theta}$. There are two ways to do this efficiently. First, I can use the LU decomposition to express $I - \beta F(\breve{P})$ as the product of a lower triangular matrix and an upper triangular matrix, both amenable to Gaussian elimination (Horn and Johnson, 2012, 216). Doing so enables me to quickly solve $\eta_{\breve{\theta}} \breve{P}$ in system of equations $(I - \beta F(\breve{P})) \eta_{\breve{\theta}} \breve{P} = U_{\breve{\theta}}(\breve{P})$ under many $\breve{\theta}$. With this, NPL becomes:

1. Create reduced form CCP estimates $\widehat{P}$.

2. Set $t = 0$ and $P_t = \widehat{P}$.

3. Increment t.

4. Calculate lower triangular matrix $\mathcal{L}_t$, upper triangular matrix $\mathcal{U}_t$, and permutation matrix $\mathcal{P}_t$, such that $\mathcal{P}_t(I - \beta F(P_{t-1})) = \mathcal{L}_t \mathcal{U}_t$.

5. Define $V_t(\breve{\theta})$ as the output of the following subroutine:

   (a) Solve $\mathcal{L}_t x = \mathcal{P}_t U_\theta(P_{t-1})$ for x.

   (b) Solve $\mathcal{U}_t v = x$ for v.

   (c) Return v.

6. Set $\theta_t = \arg\max_{\breve{\theta}} N' \log(\pi_{\breve{\theta}} V_t(\breve{\theta}))$.

7. Set $P_t = \pi_{\theta_t} V_t(\theta_t)$.

8. If $\|\nu_{\theta_t} V_t(\theta_t) - V_t(\theta_t)\| \geq \delta$ go to 3.

9. Return θ_t.

The second approach to evaluating $\rho_{\breve{\theta}} \breve{P}$ under many $\breve{\theta}$ is to approximate $\eta_{\breve{\theta}} \breve{P}$ with a sequence of Bellman contractions in such a way that the final result is a linear function of $\breve{\theta}$. Note that (i) $\eta_{\breve{\theta}} \breve{P} = \left(I - \beta F(\breve{P})\right)^{-1} U_{\breve{\theta}}(\breve{P}) = \left(I - \beta F(\breve{P})\right)^{-1} Z(\breve{P}) \breve{\theta} + \left(I - \beta F(\breve{P})\right)^{-1} G(\breve{P})$; (ii) $\left(I - \beta F(\breve{P})\right)^{-1} Z(\breve{P}) = \lim_{\tau \to \infty} Z_\tau(\breve{P})$, where $Z_\tau(\breve{P}) = Z(\breve{P}) + \beta F(\breve{P}) Z_{\tau-1}(\breve{P})$ and $Z_0(\breve{P}) = Z(\breve{P})$; and (iii) $\left(I - \beta F(\breve{P})\right)^{-1} G(\breve{P}) = \lim_{\tau \to \infty} G_\tau(\breve{P})$, where $G_\tau(\breve{P}) = G(\breve{P}) + \beta F(\breve{P}) G_{\tau-1}(\breve{P})$ and $G_0(\breve{P}) = G(\breve{P})$. Thus, my alternate NPL implementation iterates the $Z_\tau(\breve{P})$ and $G_\tau(\breve{P})$ contractions to convergence to calculate $\left(I - \beta F(\breve{P})\right)^{-1} Z(\breve{P})$ and $\left(I - \beta F(\breve{P})\right)^{-1} G(\breve{P})$, with which it expresses $\eta_{\breve{\theta}} \breve{P}$ as a linear function of $\breve{\theta}$:[8]

1. Create reduced form CCP estimates $\widehat{P}$.

2. Set $t = 0$ and $P_t = \widehat{P}$.

3. Increment t.

4. Set $\tau = 0$, $Z_{t\tau} = Z(P_{t-1})$, and $G_{t\tau} = G(P_{t-1})$.

5. Increment τ.

6. Set $Z_{t\tau} = Z(P_{t-1}) + \beta F(P_{t-1}) Z_{t\tau-1}$ and $G_{t\tau} = G(P_{t-1}) + \beta F(P_{t-1}) G_{t\tau-1}$.

7. If $\max(\|Z_{t\tau} - Z_{t\tau-1}\|, \|G_{t\tau} - G_{t\tau-1}\|) \geq \delta$ go to 5.

8. Define $V_t(\breve{\theta}) = Z_{t\tau} \breve{\theta} + G_{t\tau}$.

[8] This implementation refines the approaches of Aguirregabiria and Mira (2010, 51) and Aguirregabiria and Alonso-Borrego (2014, 27). Their $A(\breve{P})_\tau = I + \beta F(\breve{P}) A(\breve{P})_{\tau-1}$ iterations multiply two $\bar{x} \times \bar{x}$ matrices. In contrast, my $Z_\tau(\breve{P}) = Z(\breve{P}) + \beta F(\breve{P}) Z_{\tau-1}(\breve{P})$ iterations multiply an $\bar{x} \times \bar{x}$ matrix with an $\bar{x} \times \bar{z}$ matrix, and my $G_\tau(\breve{P}) = G(\breve{P}) + \beta F(\breve{P}) G_{\tau-1}(\breve{P})$ iterations multiply an $\bar{x} \times \bar{x}$ matrix with a length-$\bar{x}$ vector. My approach should be faster when $\bar{z} < \bar{x} - 1$.

9. Set $\theta_t = \arg\max_{\breve{\theta}} N' \log(\pi_{\breve{\theta}} V_t(\breve{\theta}))$.

10. Set $P_t = \pi_{\theta_t} V_t(\theta_t)$.

11. If $\|\nu_{\theta_t} V_t(\theta_t) - V_t(\theta_t)\| \geq \delta$ go to 3.

12. Return θ_t.

6.5 Strong Nested Pseudo-Likelihood

My SNPL estimator replaces NPL's policy iterations steps with relative policy iteration steps. Whereas NPL solves $\eta\breve{P}$ by iterating $Z_\tau(\breve{P}) = Z(\breve{P}) + \beta F(\breve{P}) Z_{\tau-1}(\breve{P})$ and $G_\tau(\breve{P}) = G(\breve{P}) + \beta F(\breve{P}) G_{\tau-1}(\breve{P})$ (i.e., by iterating $\eta_\tau \breve{P} = \gamma(\eta_{\tau-1}\breve{P}, \breve{P})$), SNPL solves $\bar{\eta}\breve{P}$ by iterating $\bar{Z}_\tau(\breve{P}) = \Delta Z(\breve{P}) + \beta \Delta F(\breve{P}) \bar{Z}_{\tau-1}(\breve{P})$ and $\bar{G}_\tau(\breve{P}) = \Delta G(\breve{P}) + \beta \Delta F(\breve{P}) \bar{G}_{\tau-1}(\breve{P})$ (i.e., by iterating $\bar{\eta}_\tau \breve{P} = \Delta\gamma(\bar{\eta}_{\tau-1}\breve{P}, \breve{P})$):[9]

1. Create reduced form CCP estimates $\widehat{P}$.

2. Set $t = 0$ and $P_t = \widehat{P}$.

3. Increment t.

4. Set $\tau = 0$, $\bar{Z}_{t\tau} = \Delta Z(P_{t-1})$, and $\bar{G}_{t\tau} = \Delta G(P_{t-1})$.

5. Increment τ.

6. Set $\bar{Z}_{t\tau} = \Delta Z(P_{t-1}) + \beta \Delta F(P_{t-1}) \bar{Z}_{t\tau-1}$ and $\bar{G}_{t\tau} = \Delta G(P_{t-1}) + \beta \Delta F(P_{t-1}) \bar{G}_{t\tau-1}$.

7. If $\max(\|\bar{Z}_{t\tau} - \bar{Z}_{t\tau-1}\|, \|\bar{G}_{t\tau} - \bar{G}_{t\tau-1}\|) \geq \delta$ go to 5.

8. Define $\bar{V}_t(\breve{\theta}) = \bar{Z}_{t\tau}\breve{\theta} + \bar{G}_{t\tau}$.

9. Set $\theta_t = \arg\max_{\breve{\theta}} N' \log(\pi_{\breve{\theta}} \bar{V}_t(\breve{\theta}))$.

[9] I derive iterative equations $\bar{Z}_\tau(\breve{P}) = \Delta Z(\breve{P}) + \beta \Delta F(\breve{P}) \bar{Z}_{\tau-1}(\breve{P})$ and $\bar{G}_\tau(\breve{P}) = \Delta G(\breve{P}) + \beta \Delta F(\breve{P}) \bar{G}_{\tau-1}(\breve{P})$ with identify $\Delta F(\breve{P}) = \Delta F(\breve{P}) \Delta$ from Footnote 2.

10. Set $P_t = \pi_{\theta_t}\bar{V}_t(\theta_t)$.

11. If $\|\nu_{\theta_t}\bar{V}_t(\theta_t) - \bar{V}_t(\theta_t)\| \geq \delta$ go to 3.

12. Return θ_t.

SNPL and the latter NPL version both comprise an outer loop, Steps 3 to 11, and an inner loop, steps 5 to 7. The outer loops are the same, since SNPL's $\pi_{\breve{\theta}}\bar{V}_t(\breve{\theta})$ equals NPL's $\pi_{\breve{\theta}}V_t(\breve{\theta})$. Accordingly, the estimators search the parameter space in the same way and yield the same estimates. However, SNPL uses fewer inner loops because $\bar{Z}_\tau(\breve{P})$ and $\bar{G}_\tau(\breve{P})$ converge to $\Delta\big(I-\beta F(\breve{P})\big)^{-1}Z(P)$ and $\Delta\big(I-\beta F(\breve{P})\big)^{-1}G(P)$ at rate $\beta\lambda(\breve{P})$, whereas $Z_\tau(\breve{P})$ and $G_\tau(\breve{P})$ converge to $\big(I-\beta F(\breve{P})\big)^{-1}Z(P)$ and $\big(I-\beta F(\breve{P})\big)^{-1}G(P)$ at rate β.

7 Monte Carlo Simulation

7.1 Experimental Design

A taxi driver traverses a random graph of neighborhoods $\mathbb{x} = [1, \cdots, \bar{x}]$. In each period, the driver ferries a customer from one neighborhood to another. The customer decides the destination, but the driver decides the customer class—whether to pick up a passenger from the street ($a = 0$) or the Uber app ($a = 1$). The probability that a neighborhood-i street passenger asks to go to neighborhood j is $f^x(x_j|0, x_i)$, and the probability that a neighborhood-i Uber passenger asks to go to neighborhood j is $f^x(x_j|1, x_i)$, where vectors $[f^x(x_1|0, x_i), \cdots, f^x(x_j|0, x_i)]$ and $[f^x(x_1|1, x_i), \cdots, f^x(x_j|1, x_i)]$ are symmetric Dirichlet random variables with dispersion parameter α. The expected utility from a neighborhood-i street ride is $z(0, x_i)'\theta + e(0)$, and the expected utility from a neighborhood-i Uber ride is $z(1, x_i)'\theta + e(1)$, where $z(0, x_i)$, $z(1, x_i)$, and θ are vectors of independent standard normals, and $e(0)$ and $e(1)$ are standard Gumbel shocks the taxi driver observes at the beginning

of the period. The taxi driver toggles between Uber rides and street rides to maximize expected discounted utility.

Why choose this specification? To specify a dynamic program, I must define three things: parameter vector θ, utility statistic matrix Z_a, and state transition matrix F_a. The simplest thing I can do for θ and Z_a is set their elements to independent standard normals; and the simplest thing I can do for F_a is set its rows to independent standard Dirichlets.[10] Since normal random variables have full support over the real line and Dirichlet random variables have full support over the unit simplex, my specification assigns strictly positive likelihood to all §3.1 models with two actions and Gumbel shocks. Indeed, it is the most parsimonious specification that can derive all two-action dynamic logit models; it is the simplest specification that can generate a wide variety of problems.

7.2 Implementation

I run 1,168 simulations in parallel across the 64 processors of an r4.16xlarge Amazon Web Services server. I use approximately 430 GiB of the machine's 488 GiB of RAM (I keep around 50 GiB of RAM idle to ensure no time is spent waiting for memory). The simulations comprise 1,511 processor hours of estimation time.

For a given simulation:

1. I set discount rate $\beta = 0.99$ per period.

2. I set $\bar{z} = 2$ utility statistics per state.

3. I draw state space cardinality $\bar{x}$ uniformly from $[100, \cdots, 20,000]$.

4. I draw Dirichlet dispersion parameter α with equal probability from $\{0.0001, 0.01, 1\}$.

   The $\alpha = 1$ specification yields dense state transition matrices, the $\alpha = 0.01$ specifi-

[10] Each row of a Markov matrix must lie in the unit simplex, and the Dirichlet is the standard unit simplex distribution.

cation yields semi-sparse transition matrices, and the $\alpha = 0.0001$ specification yield sparse state transition matrices.

5. I set the elements of Z_a to random normals and the rows of F_a to random Dirichlets.

6. I solve the agent's optimal policy with relative value iteration.

7. I use the optimal policy and state transition matrices to generate a sample comprising 300 state–action pairs.

8. I use these data to estimate the utility parameters with NPL, SNPL, SNFXP, and, if the state space is less than 8,000, NFXP (this estimator quickly becomes intractable).

I implement SNFXP without relative policy iteration steps. For NFXP's policy iteration steps, I use the LAPACK package to solve system of equations $\big(I - \beta F(\breve{P})\big)\eta\breve{P} = U(\breve{P})$, which I find faster than iterating $\eta_\tau\breve{P} = \gamma(\eta_{\tau-1}\breve{P}, \breve{P})$ to convergence. For NPL's policy iteration steps, I use the LAPACK package to perform an LU decomposition of $I - \beta F(P_\tau)$, which I find faster than iterating $Z_\tau(\breve{P}) = Z(\breve{P}) + \beta F(\breve{P})Z_{\tau-1}(\breve{P})$ and $G_\tau(\breve{P}) = G(\breve{P}) + \beta F(\breve{P})G_{\tau-1}(\breve{P})$ to convergence. I abstract away the estimation of state transitions, setting $\widehat{F}_a = F_a$, and I set the initial NPL and SNPL CCP estimates to the fitted values of a multinomial logistic regression of the actions on the $2\bar{a}$ variables in $\{Z_a | a \in \mathfrak{a}\}$. For all estimators, I set $\delta = 10^{-8}$, and for NFXP, I set $\delta_1 = \delta_2 = 0.1$ (which I find faster than setting $\delta_1 = \delta_2 = 0.01$).[11] I optimize across the parameter space with the BFGS algorithm.

7.3 Results

Table 1 depicts the distribution of the estimation error, $\|\widehat{\theta} - \theta\|$, across estimators. The four estimators are equally accurate because they maximize the same likelihood function. Since

[11] With $\delta_1 = \delta_2 = 0.1$, most empirical likelihood evaluations required exactly two Newton steps, satisfying Rust's (2000, 29) rule of thumb.

Table 1: Estimation Error

This table depicts the 10^{th}, 25^{th}, 50^{th}, 75^{th}, and 90^{th} percentiles of each estimator's estimation error, measured under the ℓ_∞ norm. I only consider the problems that I estimate with all four estimators (i.e., those with fewer than 8,000 states).

	10%	25%	50%	75%	90%
NFXP	0.04989	0.08431	0.14619	0.21743	0.28243
SNFXP	0.04989	0.08431	0.14619	0.21743	0.28243
NPL	0.04989	0.08438	0.14613	0.21743	0.28243
SNPL	0.04989	0.08438	0.14613	0.21743	0.28243

NFXP and SNFXP search the parameter space in the same fashion, their figures match perfectly; and since NPL and SNPL search the parameter space in the same fashion, their figures also match perfectly; but since NFXP and NPL search the parameter space in different ways, their figures differ slightly.

Figure 4 plots the estimation times as a function of the state space. The average estimation time in minutes is as follows:

	NFXP	SNFPX	NPL	SNPL
Fewer than 8,000 states	80	7.8	6.7	2.8
More than 8,000 states	–	14	124	9.8

Clearly SNFPX outperforms NFXP and SNPL outperforms NPL. But these results are hard to generalize, because they depend on spectral sub-radius $\lambda(P)$. And this scalar is small in my problems—around 0.02, 0.22, and 0.70 when transition matrices are dense, semi-sparse, and sparse, respectively.[12]

Figure 5 illustrates a more generalizable result. It plots the estimation times without strong convergence divided by the estimation times with strong convergence. As you see, strong convergence becomes more powerful as the problem scales. In fact, the time ratios increase *linearly* with the size of the state space. This result is universal: for any problem, the time ratios between NFXP and SNFXP and between NPL and SNPL will increase

[12]Further increasing $\lambda(P)$ would require imposing additional structure on the state transition matrices.

linearly with the state space cardinality so long as NFXP and NPL use traditional policy iteration. With traditional policy iteration steps, which calculate $\eta\breve{P}$ via a system of equations, NFXP's and NPL's estimation times increase cubicly with $\bar{x}$, whereas SNPL's and SNPL's estimation times increase quadratically.[13] Thus, for any problem, there is a state space large enough so that SNFXP and SNPL dominate NFXP and NPL; for any fixed $\lambda(P)$, there is an $\bar{x}$ that makes the estimation times look as they do in Figure 4.

Of course NFXP's and NPL's policy iteration steps could calculate $\eta\breve{P}$ via Bellman contraction, iterating $\eta_\tau\breve{P} = \gamma(\eta_{\tau-1}\breve{P}, \breve{P})$ to convergence. But in this case, SNFXP and SNPL mechanically dominate NFXP and NPL: the only difference is that the former estimators use relative policy iteration steps, which require roughly $\log(\delta)/\log(\beta\lambda(P))$ Bellman contractions, whereas the latter use regular policy iteration steps, which require roughly $\log(\delta)/\log(\beta)$ Bellman contractions. Thus, SNFXP and SNPL are roughly $\frac{\log(\delta)/\log(\beta)}{\log(\delta)/\log(\beta\lambda(P))} = 1 + \frac{\log(\lambda(P))}{\log(\beta)}$ times as fast as NFXP and NPL with iterative policy iteration steps. Consider the extreme case $\lambda(P) = 0.999$; if the data are annual, then $\beta \approx 0.95$ and using strong convergence is only around $1 + \frac{\log(0.999)}{\log(0.95)} = 1.02$ times as fast; however, if the data are daily, then $\beta \approx \exp(\log(.95)/365) = 0.99986$ and using strong convergence is $1 + \frac{\log(0.999)}{\log(0.99986)} = 8.1$ times as fast; and if the data are hourly, then $\beta \approx \exp(\log(.95)/365/24) = 0.9999941$ and using strong convergence is $1 + \frac{\log(0.999)}{\log(0.9999941)} = 171$ times as fast.

In sum, my estimators are competitive in problems with large state spaces and high sampling frequencies. High-frequency sampling enervates the traditional Bellman contraction by abating the per-period discount rate; and an expansive state space cripples the policy iteration step by situating the problem in the steep part of the cubic computational

[13] The slowest step in a Bellman contraction is the product of the state transition matrix with the value function vector; this operation increases quadratically with the state space cardinality. Note, the number of Bellman contractions required to evaluate $\eta\breve{P}$ generally does not increase with the size of the state space; it is always on the order of $\log(\delta)/\log(\beta\lambda(P))$.

cost curve. In these problems, strong convergence is useful.

7.4 External Validation

Aguirregabiria and Magesan (2016) replicate these findings, confirming that strong convergence shortens the solution time of an entry-exit model similar to Ericson and Pakes's (1995). They develop a new means to estimate dynamic programs based on Euler equations. Initially, they only compared their estimation times to NPL. But after I shared a preliminary version of this work with them, they added relative value iteration as a benchmark. Aguirregabiria and Magesan (2016, 25) report that relative value iteration solves their 200,000-state problem in 1,934 seconds, whereas value iteration does so in 23,270 seconds, and policy iteration does so in upwards of a million seconds.

8 Related Work

However, relative value iteration is not the fastest solution method for this problem. Aguirregabiria and Magesan's (2016) Euler equations solve it in only 274 seconds. Aguirregabiria and Magesan's (2016) estimator is elegant and fast—probably faster than mine in most cases. But theirs is not a maximum likelihood estimator, whereas mine is. Moreover, it is impossible to translate legacy NFXP or NPL code into an Euler equation estimator; in contrast, upgrading NFXP to SNFXP or NPL to SNPL requires only one line of code: `value.fn = value.fn - value.fn[1]`. Finally, while the math behind my approach is well understood, it's not clear what makes Aguirregabiria and Magesan's estimator so fast. Their numerical analysis demonstrates a powerful convergence, but their proof only guarantees a rate-β contraction.

My work relates to several other recent developments in the dynamic discrete choice literature (see Aguirregabiria and Mira (2010) and Arcidiacono and Ellickson (2011) for

Figure 4: Estimation Times

These scatter plots and corresponding LOESS regression curves depict how the estimation times vary with the size of the state space. The gray points correspond to NFXP and NPL estimation times, and black points to SNFXP and SNPL estimation times.

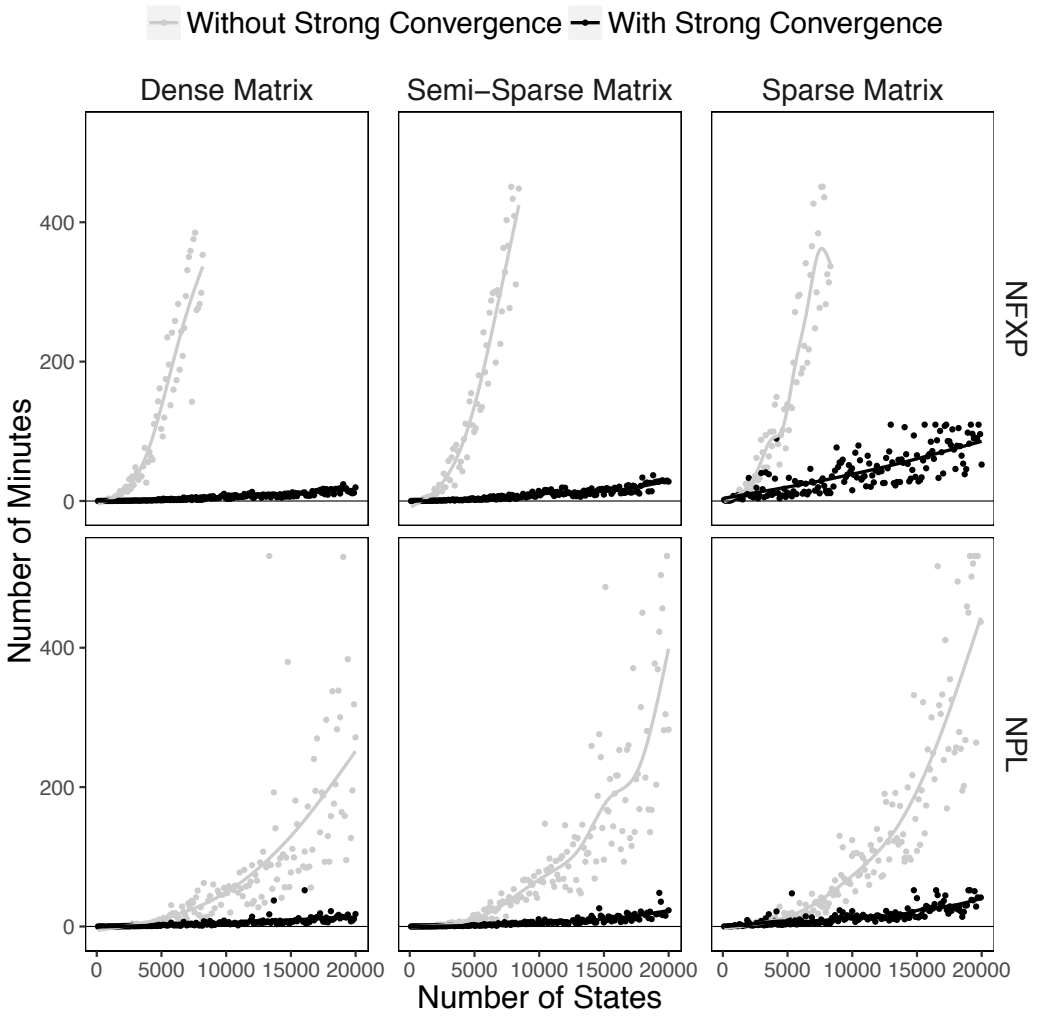
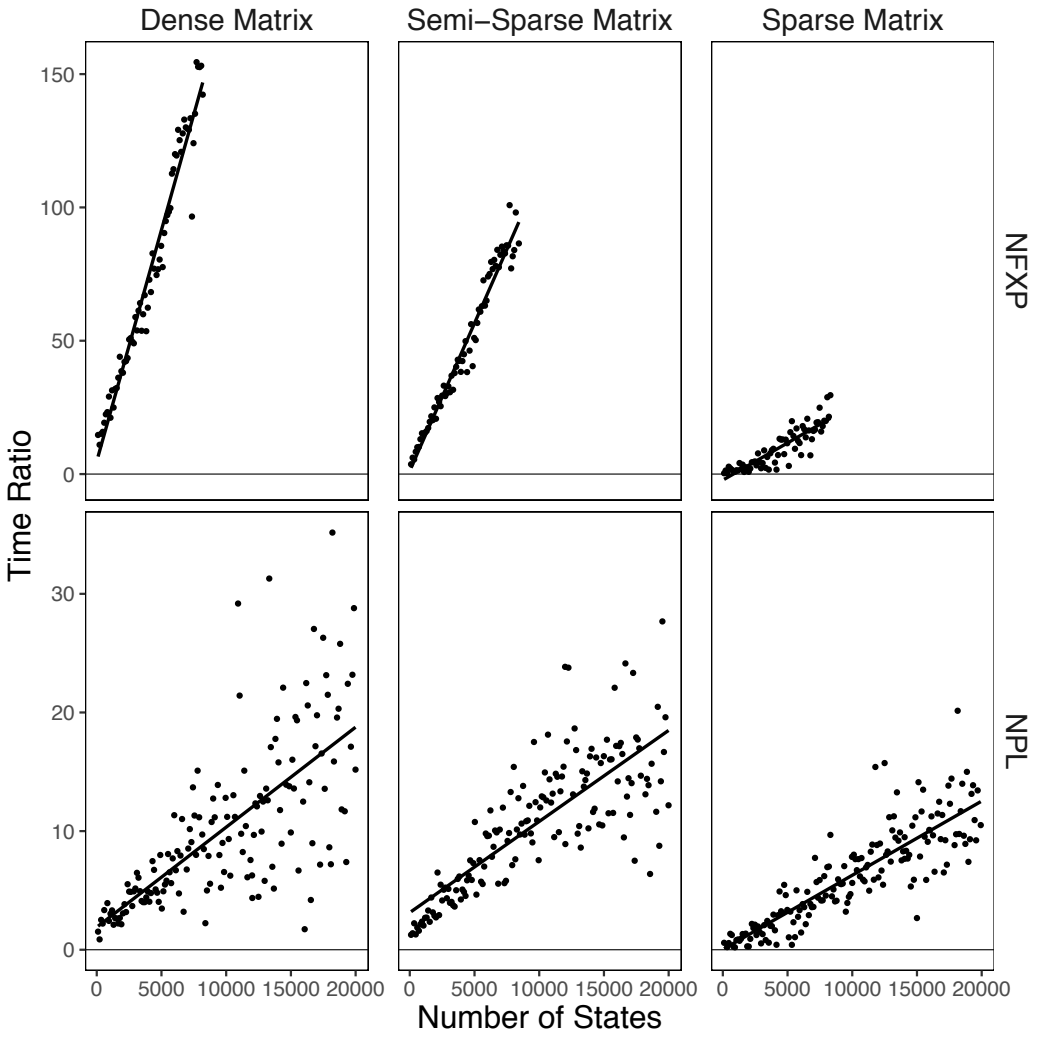


Figure 5: Estimation Time Ratios

These scatter plots and corresponding OLS regression lines depict the NFXP estimation times divided by the SNFXP estimation times, and the NPL estimation times divided by the SNPL estimation times.



more comprehensive surveys). Su and Judd's (2012) mathematical program with equilibrium constraints (MPEC) approach reframes the dynamic program as a constrained optimization problem. With Monte Carlo simulations, Su and Judd found that models that take 112 seconds to estimate with NFXP only take 0.14 seconds to estimate with MPEC. However, they use "an inefficient version of the NFXP algorithm that results in misleading conclusions about the relative speed of MPEC"; when NFXP is properly implemented "MPEC and [NFXP] are similar in performance when the sample size is relatively small. However, in problems with larger sample sizes, [NFXP] outperforms MPEC by a significant margin" (Iskhakov et al., 2015, 0-2). Also, Su and Judd only consider 175 states; I consider 20,000.

Arcidiacono and Miller's (2011) conditional choice probability (CCP) estimator employs the "finite dependence property" that there exists a sequence of actions that makes future state variables independent of the current action. The conical problem is one where the agent can regenerate the Markov chain within a fixed amount of time. Arcidiacono and Miller's estimator only considers payments received until the process reaches its regeneration state. Similarly, my estimators only consider payments received until the process reaches its stationary distribution. But whereas every ergodic Markov chain has a stationary distribution, most do not have a regeneration state. And even when a model exhibits finite dependence, there is no guarantee it will yield reasonable moment conditions: e.g., applying Arcidiacono and Miller's estimator to Harold Zurcher's engine replacement problem requires a sharp estimate of the logarithm of the probability that Zurcher replaces a brand new engine.

9 Conclusion

I will close with a few miscellaneous points that may be of interest to future researchers:

1. Traditionally, economists have observed economic agents every quarter or every month. Now we observe them every hour or every minute. For example, Uber has recently shared its trip-level data with academics (Uber, 2017); consider estimating an Uber driver's dynamic program: the time between decisions is around 15 minutes, so the per-period discount rate is around $\exp(\log(.95)/365/24/4) = 0.9999985$. Standard Bellman contractions are powerless with so little discounting. Relative Bellman contractions, on the other hand, converge steadily at the rate of Markov chain mixing. In the age of negligible discounting, strong convergence can power the Bellman contraction.

2. Aguirregabiria and Mira (2007) showed that NPL can estimate Markov perfect equilibria; SNPL generalizes to games in an equivalent fashion.

3. Iterating SNPL's outer loop to convergence is not necessary: stopping after one iteration yields a strong convergence analog to Hotz and Miller's (1993) original CCP estimator, and stopping after K iterations yields a strong convergence analog to Aguirregabiria and Mira's (2002) K-step NPL estimator. Both of these estimators are asymptotically efficient.

4. Estimators NFXP and NPL require $\beta < 1$, but estimators SNFXP and SNPL only require $\beta\lambda(P) < 1$. Since $\lambda(P) < 1$, my estimators therefore permit $\beta = 1$ (see Morton and Wecker, 1977). Strong convergence relaxes the assumption of cash flow discounting.

5. Empiricists usually fix the discount factor before estimating a dynamic discrete choice model. This seems dubious, because the value function changes dramatically with the discount factor near unity: e.g., the value function under $\beta = 0.999$ is roughly twice that under $\beta = 0.998$. However, whereas the value function asymptotes as β

goes to one, the differenced value function converges to a stable limit, making the policy function continuous in the discount factor through $\beta = 1$ (Puterman, 2014, 205). Thus, the empirical likelihood is largely insensitive to small perturbations in β near one, making the number of trailing nines inconsequential: changing β from 0.99 to 0.999999 affects the value function substantially, but not the policy function. This result justifies the tradition of calibrating the discount factor in dynamic discrete choice estimation.

6. Since $V = \bar{V} + (1-\beta)^{-1}(\nu\bar{V} - \bar{V})$, the differenced value function identifies the total value function.[14]

7. Strong convergence can facilitates the calculation of counterfactual policies and counterfactual value functions. Arcidiacono and Ellickson (2011, 391) explain that "the computational advantages of the CCP approach stem from avoiding the solution of the full DP problem when estimating the structural parameters of the underlying model. This is sometimes perceived as a weakness when it comes to conducting counterfactual policy simulations, which typically involves fully resolving the DP." In this sense, the counterfactual analysis can be the limiting aspect of the researcher's workflow. Strong convergence helps alleviates this bottleneck.

8. My SNFXP and SNPL estimators permit any compact action space. For example, in §3.1 I re-defined the decision variable from discrete choice $a \in \mathbb{a}$ to continuous choice $p \in \Xi$.

9. Strong convergence is ineffectual when $\lambda(P) \approx 1$. Hartfiel and Meyer (1998) prove that this only happens when the state transition matrix is nearly block diagonal.

[14] First, $\nu\bar{V} - \bar{V} = \alpha\iota$ for some scalar α. Second, $\nu^\tau\bar{V} = \sum_{t=0}^{\tau-1}\beta^t F(P)^t U(P) + \beta^\tau F(P)^\tau\bar{V}$, and thus $\nu^{\tau+1}\bar{V} - \nu^\tau\bar{V} = \beta^\tau F(P)^\tau U(P) - \beta^\tau F(P)^\tau(I - \beta F(P))\bar{V} = \beta^\tau F(P)^\tau(\nu\bar{V} - \bar{V}) = \alpha\beta^\tau F(P)^\tau\iota = \alpha\beta^\tau\iota = \beta^\tau(\nu\bar{V} - \bar{V})$. Third, $V = \bar{V} + \sum_{\tau=0}^{\infty}\nu^{\tau+1}\bar{V} - \nu^\tau\bar{V} = \bar{V} + \sum_{\tau=0}^{\infty}\beta^\tau(\nu\bar{V} - \bar{V}) = \bar{V} + (1-\beta)^{-1}(\nu\bar{V} - \bar{V})$.

10. Porteus (1975) demonstrate that it is often possible to transform a problem with $\lambda(P) \approx 1$ to one with $\lambda(P) \ll 1$. For example, if the smallest diagonal element of $F(P)$ is α, then the dynamic program with $\breve{F}(P) = (F(P) - \alpha I)/(1 - \alpha)$ and $\breve{U}(P) = U(P)/(1 - \alpha)$ has the same solution but a smaller spectral sub-radius.

11. When designing a model, we must keep tractability in mind. Most of us have been taught that we have two ways to facilitate computation, limiting the number of state variables and the discount rate. But we have a third degree of freedom: the rate at which the state variables gravitate back to equilibrium. We should encourage this Markov chain mixing. Due to the curse of dimensionality, we must invariably leave out some variables when designing a dynamic model. When deciding which variables to include and which to exclude, we must keep in mind that not all variables are equally burdensome. When choosing between two variables of equal economic merit, we should pick the one with a lower persistence. Indeed, it could be easier to accommodate two low-persistent variables than one high-persistent variable. Overall, I predict $\lambda(P)$ will proved to be quite elastic in most contexts; I suspect economists will find creative ways to promote Markov chain mixing once they acquire a taste for strong convergence.

References

Aguirregabiria, Victor, Cesar Alonso-Borrego. 2014. Labor contracts and flexibility: Evidence from a labor market reform in Spain. *Economic Inquiry* **52**(2) 930–957.

Aguirregabiria, Victor, Arvind Magesan. 2016. Solution and Estimation of Dynamic Structural Models Using an Euler Equations Mapping .

Aguirregabiria, Victor, Pedro Mira. 2002. Swapping the Nested Fixed Point Algorithm : A Class of Estimators for Discrete Markov Decision Models. *Econometrica* **70**(4) 1519–1543.

Aguirregabiria, Victor, Pedro Mira. 2007. Sequential estimation of dynamic discrete games. *Econometrica* **75**(1) 1–53.

Aguirregabiria, Victor, Pedro Mira. 2010. Dynamic discrete choice structural models: a survey. *Journal of Econometrics* **156**(1) 38–67.

Arcidiacono, Peter, Paul B. Ellickson. 2011. Practical Methods for Estimation of Dynamic Discrete Choice Models. *Annual Review of Economics* **3**(1) 363–394.

Arcidiacono, Peter, Robert Miller. 2011. Conditional choice probability estimation of dynamic discrete choice models with unobserved heterogeneity. *Econometrica* **79**(6) 1823–1867.

Ericson, Richard, Ariel Pakes. 1995. Markov-perfect industry dynamics: A framework for empirical work. *The Review of Economic Studies* **62**(1) 53–82.

Hartfiel, D.J., Carl D. Meyer. 1998. On the structure of stochastic matrices with a subdominant eigenvalue near 1. *Linear Algebra and its Applications* **272** 193–203.

Horn, Roger A, Charles R Johnson. 2012. *Matrix Analysis*. Cambridge university press.

Hotz, VJ, RA Miller. 1993. Conditional choice probabilities and the estimation of dynamic models. *The Review of Economic Studies* **60**(3) 497–529.

Iskhakov, Fedor, Jinhyuk Lee, John Rust, Bertel Schjerning, Kyongwon Seo. 2015. Constrained Optimization Approaches to Estimation of Structural Models: Comment.

Levin, David Asher, Yuval Peres, Elizabeth Lee Wilmer. 2009. *Markov chains and mixing times*. American Mathematical Soc.

Morton, Thomas E. 1971. On the Asymptotic Convergence Rate of Cost Differences for Markovian Decision Processes. *Operations Research* **19**(1) 244–248.

Morton, Thomas E, William E Wecker. 1977. Discounting, Ergodicity and Convergence for Markov Decision Processes. *Management Science* **23**(8) 890–900.

Porteus, E.L. 1975. Bounds and transformations for discounted finite Markov decision chains. *Operations Research* **23**(4) 761–784.

Puterman, Martin L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rust, J. 1987. Optimal replacement of GMC bus engines: an empirical model of Harold Zurcher. *Econometrica: Journal of the Econometric Society* **55**(5) 999–1033.

Rust, John. 1994. Structural estimation of Markov decision processes. *Handbook of econometrics* **4** 3081–3143.

Rust, John. 2000. Nested Fixed Point Algorithm Documentation Manual **06520**(October) 1–43.

Su, Che-Lin, Kenneth L Judd. 2012. Constrained Optimization Approaches to Estimation of Structural Models. *Econometrica* **80**(5) 2213–2230.

Uber. 2017. Introducing Uber Movement.