The Econometrics of Inverse Reinforcement Learning

Colleen O'Briant[*]

October 31, 2024

Abstract

In this paper, I build a bridge between several methods to estimate models of Dynamic Discrete Choice (DDC): the Nested Fixed Point (NFXP) algorithm developed by (Rust, 1987) from the Econometrics literature, and two Max-Margin Inverse Reinforcement Learning (IRL) methods from the Machine Learning (ML) literature: the linear programming approach of Ng and Russell (2000) and the projection method of Abbeel and Ng (2004). This work is important because integrating Econometric and ML approaches can help address fundamental challenges in ML model reliability. While ML has made remarkable advances in prediction accuracy, Econometrics offers established frameworks for identifying potential sources of bias and establishing conditions for reliable estimation. Through Monte Carlo experiments on three different GridWorld environments, I demonstrate two key findings. First, I show that IRL's wind shock assumption creates a fundamental identification issue, which can be resolved using preference shocks from the Econometrics literature. I propose an extension to Projection IRL that incorporates these preference shocks to achieve full identification. Second, while Projection IRL's accuracy is slightly lower than NFXP's, it offers substantial computational advantages - requiring up to 20 times fewer dynamic programming solutions and running up to 27 times faster.

JEL Codes: C18, C61, C50, C45, C63

Keywords: Dynamic Discrete Choice, Inverse Reinforcement Learning, Nested Fixed Point, Monte-Carlo Simulations

[*]cobriant@uoregon.edu, 1285 University of Oregon, Eugene OR 97403, 541-346-8845. Thanks to Keaton Miller for his guidance, wisdom, and encouragement as an advisor. Mark Colas, Brittany Erickson, and David Evans gave helpful comments, as did the University of Oregon's Economics Micro brown-bag group, Micro-Metrics group, and job market candidate seminar series attendees.

1 Introduction

Dynamic Discrete Choice (DDC) modeling in Econometrics and Inverse Reinforcement Learning (IRL) in Machine Learning have developed largely in parallel, sharing the fundamental objective of inferring the underlying reward function faced by agents from observed behavior. Despite this common goal, these fields have remained largely separate. Except for Sanghvi et al. (2021), little work has been done to analyze IRL's varied approaches from an econometric perspective.

IRL emerges from the reinforcement learning (RL) framework. In RL, algorithms like value iteration are used to compute optimal policies for agents with known reward functions. Inverse Reinforcement Learning tackles the inverse problem: given observations of optimal behavior, what reward function was the agent optimizing? This approach has found applications across various domains, from teaching robots to navigate crowds (Ziebart et al. (2009), Kretzschmar et al. (2016)) to autonomous helicopter flight (Abbeel et al., 2006). The canonical environment for studying IRL methods is the GridWorld environment: a discrete state space where an agent navigates between cells, each associated with different rewards, while choosing from a fixed set of actions (typically movement in cardinal directions).

This environment closely parallels the discrete choice settings studied in econometrics, where DDC models have been employed to understand dynamic decision-making in contexts such as bus engine replacement (Rust, 1987) and patent renewal (Pakes, 1986). In both IRL and in DDC, the core task is to infer the reward function that makes the observed choice sequences most likely.

I uncover two critical insights about IRL in this paper. First, the linear programming approach proposed by (Ng and Russell, 2000) faces significant limitations in estimation capabilities, as demonstrated in Section 4. Second, and more fundamentally, the "wind shock" assumption in the (Abbeel and Ng, 2004) projection method (where agents randomly deviate from optimal policies) creates an identification problem detailed in Section 5. However, when modified to incorporate preference shocks from the Econometrics literature instead of wind shocks, the projection method emerges as a compelling alternative to the Nested Fixed Point (NFXP) algorithm. While maintaining comparable accuracy, it requires substantially fewer dynamic programming solutions, offering significant computational advantages in certain applications.

1.1 Related Work

This research stems from three main areas of study: the Dynamic Discrete Choice literature from Econometrics, the Inverse Reinforcement Learning literature from Machine Learning, and the efforts made to date to bridge the gap between these two areas of research. In this section, I summarize each of these three parts in turn.

1.1.1 Dynamic Discrete Choice (DDC)

In the field of Econometrics, Dynamic Discrete Choice (DDC) is concerned with estimating structural parameters and conducting counterfactual analysis in dynamic settings. These settings feature either competition among multiple agents (entry and exit; product repositioning), or decisions by a single agent, such as whether to renew a patent (Pakes, 1986), or when to replace capital equipment (Rust, 1987). In his seminal paper, John Rust (1987) modeled the capital replacement of city bus engines in order to do structural estimation of the demand for bus engines. Given the prolonged stability in bus engine prices, not enough variation in that variable existed in order to use a regression based method (Rust, 1994). Instead, Rust offered the Nested Fixed Point (NFXP) algorithm to estimate models of DDC, which has since become the standard DDC estimation strategy.

The NFXP algorithm features an outer loop and a nested inner loop. The outer loop does maximum likelihood estimation through iterative gradient search. The nested inner loop takes a guess for structural parameters, calculates the corresponding payoff function, solves the dynamic programming problem to get the value function, uses that value function combined with an assumption on the distribution of the shocks to calculate the choice probabilities, and combines the choice probabilities with the observed data to determine the likelihood of such a guess for parameter values.

The model assumptions focus on preference shocks: the state variables which are not observed but have an effect on the choice of the agent. As I show in this paper, preference shocks are crucial for the full identification of DDC estimation strategies. The effect of preference shocks must be additively separable from the effect of observed state variables and conditionally independent across time. If assumptions on these unobserved variables hold, NFXP has consistency guarantees and is the most efficient estimator asymptotically, but NFXP is limited by high computational demands from having to solve the dynamic programming problem in each iteration of the inner loop. This limitation has prompted further research on DDC methods (Hotz and Miller, 1993). In this paper, I show that two other algorithms exist from the Machine Learning literature that are computationally cheaper than NFXP. With some adjustments to incorporate preference shocks, they can be used to estimate DDC models. But without preference shocks, they are only partially identified.

1.1.2 Inverse Reinforcement Learning (IRL)

The IRL literature in Machine Learning originates from Ng and Russell (2000). In that paper, authors Andrew Ng and Stuart Russell proposed a *Linear Programming IRL* algorithm to estimate payoffs in a Markov Decision Process given an optimizing agent's trajectory data and a model of the environment. The authors' goal was that IRL would one day be able to solve the self-driving car problem. Their idea was that it is too difficult to write down a set of policy rules for a robot to do tasks as complicated as driving a car safely, but perhaps a robot could watch a driver handle a number of traffic situations and infer the underlying payoff function which the driver is maximizing. Then the robot would have the simple task of solving the Markov Decision Process (MDP) under that payoff function for its optimal policy rule. In this way, the robot might be able to drive safely even in situations which are very different from the training dataset.

The key idea in Ng and Russell's Linear Programming IRL algorithm is to match the given dataset with simulated data according to *feature expectations*: a discounted tally of the number of times the agent was observed with each feature. Each iteration of the Linear Programming IRL algorithm includes making a guess for the payoff function and solving the dynamic programming problem for the value function, just like the inner loop of the NFXP algorithm. From there, Linear Programming IRL diverges from NFXP: it uses the value function to simulate a synthetic dataset, calculates the feature expectations of that synthetic dataset, and defines the *margin* m as the difference between the synthetic feature expectations and the feature expectations of the original dataset. Over successive iterations of the algorithm, the margin is an accumuluted sum of all past margins, and the next guess for the payoff function $\pi(s)$ is the one which maximizes $m^T\pi(s)$ (a linear programming problem). In this paper, I show that only corner solutions are possible in the linear programming problem, and as a result, the algorithm is limited to making estimates on the set {-1, 0, 1}.

The Ng and Russell (2000) model differs from the model in Rust (1987) primarily in how it explains noise: idiosyncratic discrepancies between optimal policies predicted by the payoff function and those observed in the data. Rust models this noise as originating from preference shocks: random shocks drawn each period added to the payoff of taking each action. Ng and Russell, on the other hand, model noise through *wind shocks* with a wind coefficient that is a model parameter: under a wind coefficient of 0.7 for example, the agent takes an action according to its optimal policies with a probability of 0.7, and with a probability of 0.3, she samples randomly from all possible actions. The main contribution of my paper is to show that the distinction between wind shocks and preference shocks is significant and it has important implications for the full identification of the estimation strategy.

Andrew Ng then published a paper describing a second IRL algorithm with Pieter Abbeel (Abbeel and Ng, 2004), this time using projection instead of linear programming. In this paper, I refer to this algorithm as *Projection IRL*. Each iteration of this algorithm takes a guess for the payoff function and solves the dynamic programming problem for the value function, just like NFXP and Linear Programming IRL. Then just like Linear Programming IRL, Projection IRL uses simulation to create a synthetic dataset and calculates its feature expectations. From there, Projection IRL diverges from Linear Programming IRL: it uses projection and previous synthetic feature expectations to approach the expert feature expectations. The margin is the difference between the projection and the expert feature expectations, and the algorithm stops when the length of the margin becomes small enough. The next guess for the payoff function is the direction of the margin. Abbeel and Ng do not present an alternative to modeling noise through wind shocks. However, in this paper I demonstrate that by making certain adjustments to the Projection IRL algorithm, preference shocks can be effectively applied within Projection IRL. This adaptation significantly enhances its identification capabilities.

Recent studies in Inverse Reinforcement Learning (IRL) have expanded its applications across various fields. These include helicopter aerobatics (Abbeel et al., 2006), navigating crowds and pedestrians in a socially compliant way in order to avoid hindering pedestrians (Ziebart et al. (2009), Kretzschmar et al. (2016), and Kim and Pineau (2016)), playing video games (Tucker et al., 2018), boat sailing (Neu and Szepesvari, 2012), robotic legged locomotion (Ratliff et al., 2009), and energy efficient driving (Vogel et al. 2012).

1.1.3 Other Work on Bridging the Gap

To date, only one other paper has attempted to bridge the gap between IRL and DDC. Sanghvi et al. (2021) established that IRL and DDC share the same objective: to estimate the structural parameters of an MDP using data from an agent's optimal trajectories. They showed that one IRL approach, Maximum Causal Entropy IRL (Ziebart, 2010) is equivalent to the Nested Fixed Point algorithm (Rust, 1987) when payoffs are linear in features. Their research included a detailed comparison of this IRL method with two methods from DDC (Hotz and Miller 1993 and Aguirregabiria and Mira 2002). They found that when a lot of expert data was available, all methods perform similarly well. But when data was limited and feature counts were well approximated, NFXP had an edge over the two other DDC methods.

My research is unique because, unlike Sanghvi et al., I compare the Nested Fixed Point algorithm not with Maximum Causal Entropy IRL, but with two earlier fundamental IRL methods developed by Ng and Russell, and Abbeel and Ng. These methods are conceptually straightforward, easier to execute, and less computationally demanding, making them appealing for many applications.

2 Model (Rust, 1987) and Notation (Pesendorfer and Schmidt-Dengler, 2008)

Consider a single-agent discrete decision process where an expert is observed over time in states $s_t \in S = \{1, ..., L\}$ taking actions $a_t \in A = \{0, 1, ..., K\}$ so that the data takes the form $(a_t, s_t)_{t=1}^T$. Each period, the agent is endowed with a state variable s_t and a vector of pay-off shocks $\varepsilon_t \in \mathbb{R}^K$ (the pay-off shocks are ultimately not observable to the econometrician). The pay-off shocks are i.i.d. Normal. There are K pay-off shocks but $K+1$ actions because only the differences between pay-off shocks are relevant, so the first action's pay-off shock can be normalized to zero (McFadden, 1973).

The agent receives a pay-off each period according to a_t, s_t, and ε_t equal to

$$\pi(a_t, s_t) + \sum_{k=1}^{K} \varepsilon_{t,k} \cdot \mathbf{1}(a_t = k) \tag{1}$$

and then the state transition probability matrix G governs how the agent moves to s_{t+1}. G is made up of elements $g(a_t, s_t, s_{t+1})$, which is the probability the agent moves from s_t to s_{t+1} after having taken action a_t. The pay-off for the entire process is the discounted sum of per-period pay-offs where the discount factor is given by $\beta < 1$. I assume the transition process is Markovian so that I can abstract from calendar time. Let $p(s)$ be the vector of choice probabilities, so that $p(a|s)$ is the probability the agent will choose action a in state s. Then the value function is given by

$$V(s;p) = \sum_{a \in A} p(a|s)\pi(a,s) + \sum_{k=1}^{K} E_\varepsilon\left[\varepsilon_k | a = k\right] p(a = k|s) + \beta \sum_{s' \in S} g(a, s, s')V(s';p). \tag{2}$$

2.1 Nested Fixed Point Implementation Details

I use the NFXP algorithm outlined in Rust (1987). The objective is to estimate the model parameters θ which describe the payoff $\pi(a, s)$.

NFXP requires two sets of summary statistics from the discrete choice data: an approximation of the the probability transition matrix G and the observed choice frequency $n_{ks} \equiv \sum_t I(a_t = k, s_t = s)$.

The algorithm has an inner and outer loop, thus the name "nested". The outer loop uses maximum likelihood estimation to search for the values of θ which make the observed n_{ks} most likely. That log-likelihood is defined as

$$LL = \sum_{k,s} n_{ks} \cdot \log(p(a = k|s)), \tag{3}$$

where the choice probabilities of taking action a in state s, $p(a|s)$, are calculated for the values θ in an inner loop.

The inner loop takes θ and uses a value iteration technique to find the value function $V(s; p)$. Then to find the choice probabilities $p(a|s)$, recall that shocks ε are assumed to be Normally distributed. So after observing the realizations of shocks ε for a period, the probability that action a_1 for example has the highest value out of all other possible actions is

$$\begin{aligned} p(a|s) = P(&V(s; a = 1) + \varepsilon_1 > V(s; a = 2) + \varepsilon_2 \quad \cap \\ &V(s; a = 1) + \varepsilon_1 > V(s; a = 3) + \varepsilon_3 \quad \cap \\ &... \quad \cap \\ &V(s; a = 1) + \varepsilon_1 > V(s; a = K) + \varepsilon_K) \end{aligned} \tag{4}$$

And rearranging

$$
\begin{aligned}
p(a|s) = P(&\varepsilon_1 - \varepsilon_2 > V(s; a = 2) - V(s; a = 1) \quad \cap \\
&\varepsilon_1 - \varepsilon_3 > V(s; a = 3) - V(s; a = 1) \quad \cap \\
&... \quad \cap \\
&\varepsilon_1 - \varepsilon_K > V(s; a = K) - V(s; a = 1)
\end{aligned}
\tag{5}
$$

I assume that shocks ε are independent and normally distributed with equivalent means and variances of 1. Then the differences $\varepsilon_1 - \varepsilon_2$, $\varepsilon_1 - \varepsilon_3$, ..., and $\varepsilon_1 - \varepsilon_K$ are also normally distributed, they have mean zero, and they each have a variance of 2. I therefore find the joint probability $p(a|s)$ using a multivariate normal distribution package using the lower integration limits of $(V(s, a = 2), V(s, a = 3), ..., V(s, a = K)) - V(s, a = 1)$. Note that $Cov(\varepsilon_1 - \varepsilon_2, \varepsilon_1 - \varepsilon_3) = Cov(\varepsilon_1, \varepsilon_1) = Var(\varepsilon_1) = 1$, so the multivariate normal variance-covariance matrix I use is this $K \times K$ matrix.

$$
\begin{bmatrix}
2 & 1 & 1 & \cdots \\
1 & 2 & 1 & \cdots \\
1 & 1 & 2 & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\tag{6}
$$

If instead you assume that the shocks are, like a logit, distributed Extreme Value Type 1, computing the conditional probabilities is even simpler, following Rust (1987),

$$
p(a|s) = \frac{exp(\pi(a, s) + \beta \sum_{s' \in S} g(a, s, s')V(s'; p))}{\sum_{a \in A} exp(\pi(a, s) + \beta \sum_{s' \in S} g(a, s, s')V(s'; p)}.
\tag{7}
$$

2.2 Linear Programming IRL Implementation Details

The key to the IRL algorithms from Ng and Russell (2000) and Abbeel and Ng (2004) is that the true pay-off function $\pi(s)$ might be recovered by simulating synthetic trajectories under different guesses for payoff functions and trying to match feature expectations μ.

Assume the pay-off function $\pi(s)$ is a function of only s and not a function of action

a, and all preference shocks are zero. Instead of shocks, the agent is blown as if by wind a random direction w.p. 0.3. Otherwise, the agent moves in order to maximize value. Let $\{s_1, s_2, ...\}$ represent the process following policy $p(a|s)$. Then the value function evaluated at s_1 is

$$V(s_1; p) = \sum_{t=1}^{\infty} \beta^{t-1} \pi(s_t)$$

Let the *feature expectations* of $\{s_1, s_2, ..., s_T\}$ in the simplest case where each state is its own feature be $\mu(s) = \sum_{t=1}^{T} \beta^{t-1} 1(s_t = s)$. Then the key to IRL is that the $V(s_1, p)$ can be estimated by

$$\hat{V}(s_1; p) = \mu(s)^T \cdot \pi(s)$$

The expert's policy p^* should maximize $\hat{V}(s_1, p)$, so compared to any other policy p':

$$\begin{aligned} \hat{V}(s_1|p^*) &\geq \hat{V}(s_1|p') \\ \mu^*(s)^T \pi(s) &\geq \mu'(s)^T \pi(s) \\ \mu^*(s)^T \pi(s) - \mu'(s)^T \pi(s) &\geq 0 \\ \left(\mu^*(s) - \mu'(s)\right)^T \pi(s) &\geq 0 \end{aligned}$$

For Linear Programming IRL (Ng and Russell, 2000), let the feature expectation margin be $m(s) = \mu^*(s) - \mu'(s)$. The pay-off function $\pi(s)$ that best rationalizes $\mu^*(s)$ compared to $\mu'(s)$ is found by

$$\begin{aligned} \max_{\pi(s)} \quad & m(s)^T \pi(s) \\ \text{s/t} \quad & |\pi(s)| \leq 1 \quad \text{for } s = 1, ..., L \end{aligned}$$

Which is solved by

$$\pi(s) = \begin{cases} -1 & \text{if } m(s) < 0 \\ 1 & \text{if } m(s) > 0 \\ \text{any in } [-1, 1] & \text{if } m(s) = 0 \end{cases}$$

In order to estimate $\pi(s) = 0$ instead of only -1 and 1, add L1 regularization:

$$\begin{aligned} \max_{\pi(s)} \quad & m(s)^T \pi(s) - 1.2 \sum_s |\pi(s)| \\ \text{s/t} \quad & |\pi(s)| \leq 1 \quad \text{for } s = 1, ..., L \end{aligned}$$

Which makes the solution:

$$\pi(s) = \begin{cases} -1 & \text{if } m(s) < -1.2 \\ 1 & \text{if } m(s) > 1.2 \\ 0 & \text{if } -1.2 < m(s) < 1.2 \end{cases}$$

Using a constrained optimization package instead of this decision rule leads to artifacts which may estimate $\pi(s)$ falsely to be outside of the set $\{-1, 0, 1\}$. So to summarize, the Ng and Russell (2000) Linear Programming IRL algorithm is:

1. Use data $\{a_t, s_t\}_{t=1}^T$ to estimate the observed feature expectations μ^*. Generate a random pay-off function π_1 and set $i = 1$ and $m = 0$.

2. Use π_i to find the value function, generate expert trajectories, and feature expectations μ_i. Let $m = m + \mu^* - \mu_i$.

3. Set $i = i + 1$ and find π_i: $\max_{\pi_i} m\pi_i - 1.2 \sum_s |\pi_i|$. Jump to step 2, or quit after a large number of iterations (convergence is not guaranteed).

2.3 Projection IRL Implementation Details

Projection IRL (Abbeel and Ng, 2004) is similar to Linear Programming IRL except in the way that it updates guesses for the payoff function. It uses Projection instead of Linear Programming to try to find payoff functions which generate simulated datasets with similar feature expectations compared to the expert. This is how the Projection algorithm works:

1. Initialize the algorithm by calculating expert feature expectations μ_E the same way as in Linear Programming IRL: $\mu_E = \sum_{t=1}^{T} \beta^{t-1} 1(s_t = s)$. Let $i = 0$ and make a random guess for the payoff function and call it π_0.

2. Solve the Dynamic Programming Problem: for the current payoff function guess π_i, solve the dynamic programming problem to find the value function. Use this value function to generate a synthetic dataset for an agent optimizing under π_i. Calculate the feature expectations for this synthetic dataset, denoted as μ_i.

3. For $i = 0$, calculate the margin and update the payoff function: compute the margin $m_0 = \mu_E - \mu_0$. Update the iteration counter $i = 1$ and make the next guess for the payoff function $\pi_1 = m_0$. Find μ_1 using the same procedure as in step 2.

4. Projection Step: Update the guess towards the expert's feature expectations by projecting onto the space of possible feature expectations. Specifically, adjust $\bar{\mu}_i$ as follows: $\bar{\mu}_i = \mu_{i-1} + projection(m_{i-1}, \mu_i - \mu_{i-1})$. The next guess for the payoff function then becomes $\pi_i = \mu_E - \bar{\mu}_i$.

5. Iterate: Update the iteration counter $i = i + 1$. Calculate μ_i, the feature expectations implied by the updated payoff function guess π_i, as done in step 2. Repeat steps 4 and 5 until the magnitude of π_i becomes smaller than a predefined threshold ε. The final estimate can undergo an affine transformation between -1 and 1.

The algorithm works smoothly under the assumption of wind shocks. But using preference shocks instead introduces a complication. Specifically, as the magnitude of the update to the payoff function π_i decreases toward zero, synthetic feature expectations μ_i converge to the feature expectations under a payoff function of all zeros. Under preference shocks with mean 0 and standard deviation 1, this looks like the agent is wandering the grid completely randomly. This undermines the algorithm's ability to align the synthetic agent's feature expectations with those of the expert. To address this, I included an affine transformation of the updated payoff function guess between -1 and 1 *every iteration*, not just for the final iteration (after comparing its magnitude to ε).

I also observed an improvement in performance if, at the end, Projection IRL returns the payoff function which generated feature expectations closest to the expert, instead of returning the smallest $\mu_E - \bar{\mu}_i$. The rationale is that $\mu_E - \bar{\mu}_i$ is a corrective adjustment to the search direction rather than the most accurate estimate of the payoff function that could replicate the expert's feature expectations.

3 Methods

Each experiment is on a 5x5 GridWorld where features are comprised of one or more cells, but for simplicity, no cells belong to more than one feature. Every cell in a feature has the same payoff.

The first step is to generate the expert data:

1. Select the expert agent's starting position randomly out of the 25 cells on the grid.

2. The expert moves according to either "wind shocks" or "preference shocks", depending on the experiment trial. If the trial uses preference shocks, then each time step, five i.i.d. Normal (0, 1) shocks are drawn and added to the value of taking each action and the agent takes the action with the highest value. If the trial uses wind shocks with a wind coefficient of 0.7, the agent takes the action with the highest value with probability 0.7, and with probability 0.3, she selects among all actions with equal probability. The expert always has five actions available: up, down, left,

right, or to stay in the same place. If the expert moves toward the outer boundary of the grid, they stay in the same place. The discount factor is 0.95.

3. Truncate the trajectory after 10 time steps.

4. Generate `nt` such trajectories and evaluate the performance of NFXP, Linear Programming IRL, and Projection IRL on each of these datasets.

Repeat the steps above for 20 trials per experiment using increasing amounts of data (let `nt` be 10, 50, and finally 100).

When the NFXP algorithm is used, I apply an affine transformation to bound the final NFXP estimate between -1 and 1. Note that the NFXP algorithm always models the expert data as having come from the preference shock process, so when the expert data is generated with the windy process, NFXP is misspecified. Linear Programming IRL and Projection IRL uses simulation in its estimation strategy, so when the expert is subject to wind shocks, both IRL algorithms simulate data subject to wind shocks. And when the expert is subject to preference shocks, both IRL algorithms simulate data subject to preference shocks.

4 Results

Table 1 presents summary statistics (means and standard deviations in parentheses) from the first experiment involving a 3-parameter GridWorld. For a more detailed illustration of each trial's estimate, see 9. Under preference shocks, NFXP is able to estimate the true values of the payoff function with accuracy, even with a limited dataset of only 100 points. As the size of the data increases, NFXP's estimations get even closer to the true payoff function values and estimates are more consistent. Projection IRL is almost as accurate and consistent as NFXP, and takes an average of 2-31 seconds to complete with an average of 20-51 iterations required, compared to NFXP's 55-65 seconds and 406-417 iterations required. Linear Programming IRL is not able to estimate values outside the set {-1, 0, 1}, yet under preference shocks, it is fairly accurate and its estimates improve with more data.

But under wind shocks (see the right hand side of Table 1), NFXP is no longer able to estimate the payoff function with accuracy: it estimates around (1, -.9, -.8) instead of the true values of (1, .95, -1). Projection IRL fairs a little better, estimating around (1, .6, -1), and LP IRL settles on (1, 0, -1) most frequently.

The same pattern presents in Table 2 for the 4-parameter GridWorld. Here, the true payoff function is (1, .99, .94, -1), which both NFXP and Projection IRL are able to estimate accurately and consistently under preference shocks, especially as the amount of data increases. Projection IRL takes, on average, between 10 and 55 seconds to complete, requiring 70 to 94 iterations. NFXP, on the other hand, takes 62 to 73 seconds to complete and does 479-501 iterations. Linear Programming IRL is less successful here with a tendency to estimate (0, 0, 1, -1) (see 10 for more detail). And under wind shocks, again NFXP is no longer able to make accurate estimates, and neither either of the IRL algorithms.

For the 5-parameter GridWorld, the same pattern appears again (see Table 3): under preference shocks, NFXP and Projection IRL do well and NFXP has a slight edge. Linear Programming IRL is a good estimator as well, estimating parameters rounded to the nearest integer: it estimates (1, 0, 0, 0, -1) instead of the true values of (1, .36, .36, .33, -1). And under wind shocks, Projection IRL is the only algorithm that gets close, but it still is not able to estimate the payoff function very precisely. One difference from the 3 and 4 parameter experiments is this: Projection IRL does not hold its time and number of iterations advantage over NFXP under preference shocks. NFXP takes 50-64 seconds to complete, doing 486-502 iterations, while Projection IRL takes 27-385 seconds to complete, doing 364-692 iterations.

5 Discussion

5.1 Estimates Under Preference Shocks

In the case that unobserved variables can be modeled as preference shocks, which method is recommended? Holding constant the data size and the number of parameters to estimate, NFXP has a slight advantage over Projection IRL when it comes to accuracy, across the board. That advantage decreases as the number of parameters to estimate increases or the data size increases. But in many cases, NFXP has a significant disadvantage over Projection IRL in terms of the number of iterations and the time it takes to converge.

So I recommend using Projection IRL in use cases with many parameters to estimate, and NFXP to use elsewhere. Linear Programming IRL can be used in cases where it is advantageous to assume the payoff function is limited to the set {-1, 0, 1}.

5.2 Identification Problem from Wind Shocks

In every experiment with wind shocks, I found that NFXP, Linear Programming IRL, and Projection IRL all appeared to be biased estimators of the payoff function when the expert data was generated with wind shocks. In order to explain why, I'll first explain a couple of differences between data generated with wind shocks and data generated with preference shocks. Then I'll explain why wind shocks create identification problems for estimators.

Consider a cliff GridWorld environment (Figure 4). The agent can collect a payoff of 50 and then 100, but only if they are willing to approach the cells with payoff -1000. When the agent is subject to preference shocks (see figure 4), there is no danger because every action that agent makes is purposeful. The agent collects 50 on their way to collecting 100 and only finds themselves in the cells with -1000 when that was their (random) starting position. Preference shocks are distributed N(0, 1), so because of the scale of payoffs on this grid, behavior here is identical to the behavior in the deterministic case where there are no shocks at all.

When the expert is instead subject to wind shocks set to 0.7 (see figure 5), then 30% of the time, the agent can not act with intention and is instead blown randomly by wind. The agent's behavior here is very different from the preference shock case. Note that this agent largely avoids collecting the payoff of 50 on their way to collecting 100 so that they don't have to risk being blown off the edge of the cliff.

When I change the wind coefficient to 0.5 (50% of the time, the agent is blown in a random direction: see figure 6), the agent's best policy shifts again. Now, the agent does not seem to have the goal of collecting 50 *or* 100, but instead, it stays near the bottom of the board, as far away as possible from the -1000 payoff cells, as if for safety. Preference shocks and wind shocks can create different behaviors from the agent. In the preference shock case, you need to know the variance of the preference shocks in order to estimate the scale of the payoffs correctly (assuming a variance of 1 will still allow you to correctly estimate relative values of the payoffs). But in the wind shock case, different wind coefficients can yield very different behaviors and it is crucial to either estimate it correctly or give your algorithm correct values for that wind coefficient.

Consider a new GridWorld environment with the same shape as experiment 1. In all three of figures 7a, 7b, and 7c, the agent is subject to preference shocks distributed N(0, 1). In figure 7a, the payoff function is (1, 0.5, 0). The result is that the agent has a slight preference for the center of the grid. As the scale of the payoff function increases to (2, 1, 0) in figure 7b, the agent's preference for the center cell and, sometimes, left and right of center intensifies. As the scale of the payoff function again increases to (10, 5, 0) in figure 7c, the agent works to get to the center and then never wanders away from it. Although the payoff functions in these three figures are similar, under preference shocks, the behaviors they elicit are very different.

On the other hand, in figures 8a, 8b, and 8c, the agent is subject to wind shocks with a wind coefficient of 0.7. As the scale of the payoff function increases again from (1, .5, 0), to (2, 1, 0) to (10, 5, 0), the behavior of the agent *does not change*. Here, 30% of the time, the agent is blown randomy by wind, and 70% of the time, the agent is able to act according to its best policy. And the best policy for the agent is independent of the scale of the payoff: as long as the payoff function is increasing toward a single maximum, the agent can't improve on a policy that includes reaching that maximum (from the left or right if it doesn't take extra steps to do so), and once in the center, to stay there. This is the root of the identification issue that comes with wind shocks. Because the agent's behavior does not change when the scale of the payoff function changes, under wind shocks, relative values for the payoff function can never be identified precisely: ony ordinal values can sometimes be estimated.

Wind shocks make identification impossible because when the expert takes non-optimal actions randomly, there is an upper bound to the amount of information the Econometrician can possibly extract from the data. Best policies may be identifiable (letting ordinal identification sometimes be possible), but if the agent takes all other actions in a random uniform way, then it is unknown which action may have been second- or third-best, and it is unknown how much worse the second-best (or third-best) option may have been compared to the best options. When the agent is blown by wind, actions give the Econometrician no information about the relative values for $\pi(s')$, and estimates will be biased no matter the estimation strategy.

The Econometrics literature (and NFXP) solves this identification problem by assuming that for each time step, actions have preference shocks which are not observable to the econometrician, but the expert always takes the action with the highest value *combined* with the shock. In this way, the best option is taken with the highest probability, and then the second best option is taken with the next highest probability, onward. And even more than that, the difference between the relative action frequencies gives the econometrician useful information about how much worse the second-best options are from the first, and so on. With preference shocks, the expert takes action a in a way that is proportional to $V(s';p)$, so actions give the econometrician the information required about relative values for $\pi(s')$ so that identification is possible.

6 Conclusion

This paper explores three methods for estimating payoffs in sequential discrete choice models, focusing on their respective advantages and disadvantages. I compared Inverse Reinforcement Learning (IRL) methods—Linear Programming IRL and Projection IRL—with the Nested Fixed Point (NFXP) algorithm from the Econometrics literature.

I found that wind shocks commonly used in Machine Learning lead to partial identification of the payoff function, whereas preference shocks from Econometrics allow for full identification in both Linear Programming IRL and Projection IRL. I showed that Linear Programming IRL is limited to identifying payoffs in the set {-1, 0, 1}. Despite Projection IRL's slightly lower accuracy compared to NFXP, it offers significant computational advantages, requiring up to 20 times fewer dynamic programming problems to be solved and delivering results up to 27 times faster. These findings suggest that Projection IRL may be preferable in situations with large state spaces, while Linear Programming IRL can be useful when the payoff function is limited to a specific set of values.

References

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 1, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-5.

P. Abbeel, A. Coates, M. Quigley, and A. Ng. An application of reinforcement learning to aerobatic helicopter flight. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL `https://proceedings.neurips.cc/paper_files/paper/2006/file/98c39996bf1543e974747a2549b3107c-Paper.pdf`.

V. Aguirregabiria and P. Mira. Swapping the nested fixed point algorithm: A class of estimators for discrete markov decision models. *Econometrica*, 70(4):1519–1543, 2002. doi: https://doi.org/10.1111/1468-0262.00340. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0262.00340`.

V. J. Hotz and R. A. Miller. Conditional Choice Probabilities and the Estimation of Dynamic Models. *Review of Economic Studies*, 60(3):497–529, 1993. URL `https://ideas.repec.org/a/oup/restud/v60y1993i3p497-529..html`.

B. Kim and J. Pineau. Socially adaptive path planning in human environments using inverse reinforcement learning. *International Journal of Social Robotics*, 8:51–66, 2016.

H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35, 01 2016. doi: 10.1177/0278364915619772.

D. McFadden. Conditional logit analysis of qualitative choice behaviour. In P. Zarembka, editor, *Frontiers in Econometrics*, pages 105–142. Academic Press New York, New York, NY, USA, 1973.

A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000.

A. S. Pakes. Patents as Options: Some Estimates of the Value of Holding European Patent Stocks. *Econometrica*, 54(4):755–784, July 1986. URL `https://ideas.repec.org/a/ecm/emetrp/v54y1986i4p755-84.html`.

M. Pesendorfer and P. Schmidt-Dengler. Asymptotic least squares estimators for dynamic games -super-1. *Review of Economic Studies*, 75(3):901–928, 2008. URL `https://EconPapers.repec.org/RePEc:oup:restud:v:75:y:2008:i:3:p:901-928`.

J. Rust. Optimal replacement of gmc bus engines: An empirical model of harold zurcher. *Econometrica*, 55:999–1033, 02 1987. doi: 10.2307/1911259.

J. Rust. Chapter 51 structural estimation of markov decision processes. volume 4 of *Handbook of Econometrics*, pages 3081–3143. Elsevier, 1994. doi: https://doi.org/10.1016/S1573-4412(05)80020-0. URL `https://www.sciencedirect.com/science/article/pii/S1573441205800200`.

N. Sanghvi, S. Usami, M. Sharma, J. Groeger, and K. Kitani. Inverse reinforcement learning with explicit policy estimates, 2021.

A. Tucker, A. Gleave, and S. Russell. Inverse reinforcement learning for video games. In *Proceedings of the Workshop on Deep Reinforcement Learning at NeurIPS*, 2018.

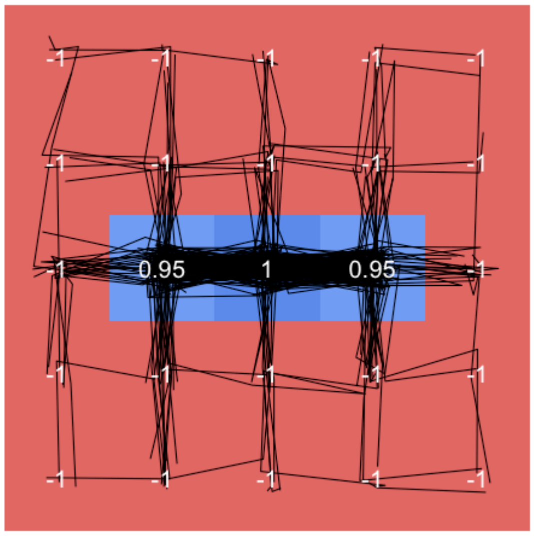
B. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. Bagnell, M. Hebert, A. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. pages 3931–3936, 12 2009. doi: 10.1109/IROS.2009.5354147.

B. D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy.* PhD thesis, Carnegie Mellon University, School of Computer Science, Machine Learning Dept., 2010.

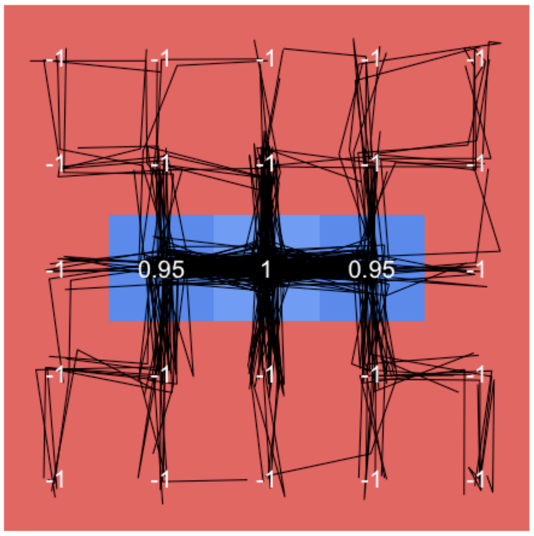
7 Tables and Figures

Figure 1: Simulated Data on a 3-Parameter GridWorld

(a) Preference Shock Trajectories



(b) Wind Shock Trajectories



Notes: Figures 1 (a) and (b) illustrate typical simulated paths of an agent on a 5x5 GridWorld where the agent is tasked with maximizing her cumulative discounted payoffs over time. The grid has a defined payoff structure that is known to the agent: the center cell has a payoff of 1 unit, the cells immediately to the left and right of the center have a payoff of 0.95 units, and all other cells have a payoff of -1 units. Each trajectory consists of 10 steps, taken by an agent that starts from a random location and who can move up, down, left, right, or who can stay in the same place. If the agent hits the grid's outer edge, she remains in place. Figures 1 (a) and (b) both illustrate 50 such trajectories.

The trajectories in **Figure 1 (a)** are subject to preference shocks, which means that at every time step, the agent receives a set of five independent and identically distributed shocks, drawn from a normal distribution with mean 0 and standard deviation 1, one for each potential action. The agent then chooses the action that is expected to yield the highest total discounted payoff, shock included. This process generates trajectories with the pattern that the agent is attracted toward the center cell and, once there, moves predominantly left or right of center, with fewer movements up or down from the center.

The agent in **Figure 1 (b)**, on the other hand, is subject to wind shocks with a wind coefficient of 0.7. That means that there is a 70 percent chance that the agent can move with her intention (selecting from optimal policies), and a 30 percent chance that a random gust of 'wind' will blow her in a random direction. The agent's strategy still involves moving toward the center, often via the cells to the left and right of the center. However, once in the center, she stays there unless blown by wind. The agent's movements to the left and right of the center therefore occur just as frequently as movements up and down from the center, which contrasts with Figure 1. Note that trajectory data in Figures 1 (a) and (b) seem similar on the surface, except for the activity going up and down from the center cell. Although wind shocks can sometimes create data that looks superficially similar to preference shock data, wind shocks prevent the full identification of all estimation strategies (see Table 1).

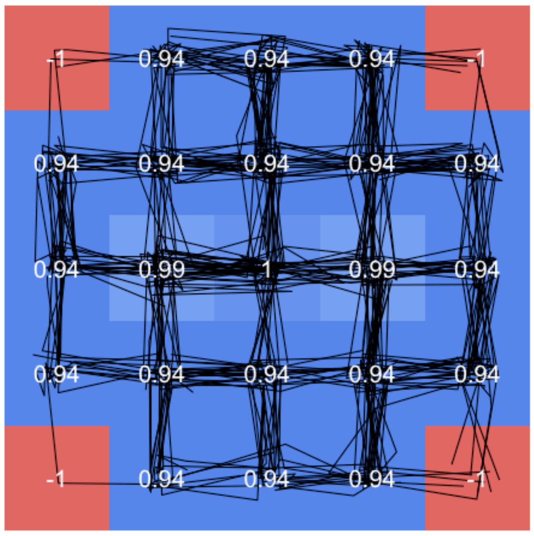
Table 1: Results for the Estimation of a 3-Parameter GridWorld

		Preference Shocks			Wind Shocks		
N = 100							
	True Values	NFXP	Projection IRL	LP IRL	NFXP	Projection IRL	LP IRL
Feature 1	1	0.95 (0.09)	0.89 (0.19)	0.8 (0.41)	1 (0)	0.97 (0.06)	1 (0)
Feature 2	0.95	0.9 (0.12)	0.88 (0.18)	0.9 (0.31)	-0.77 (0.25)	0.13 (0.79)	0.05 (0.69)
Feature 3	-1	-1 (0)	-1 (0)	-1 (0)	-0.82 (0.22)	-0.76 (0.52)	-0.95 (0.22)
Time elapsed (seconds)		55.2 (12.41)	2.06 (0.95)	21.53 (4.29)	57.32 (11.8)	0.66 (0.38)	6.92 (0.75)
Number of iterations		406.25 (49.04)	20.05 (8.44)	204.8 (8.37)	395.6 (88.89)	19.2 (11.51)	212.65 (16.59)
N = 500							
Feature 1	1	0.98 (0.04)	0.95 (0.08)	0.9 (0.31)	1 (0)	0.98 (0.04)	1 (0)
Feature 2	0.95	0.91 (0.11)	0.89 (0.14)	0.8 (0.41)	-0.76 (0.15)	0.64 (0.54)	0.55 (0.51)
Feature 3	-1	-1 (0)	-1 (0)	-1 (0)	-0.91 (0.17)	-1 (0)	-1 (0)
Time elapsed (seconds)		65.64 (8.77)	18.67 (18.09)	90.67 (4.06)	71.44 (5.06)	0.75 (0.75)	6.64 (0.57)
Number of iterations		408.35 (58.54)	40.65 (39.02)	201.25 (0.72)	435.25 (31.61)	22.45 (22.86)	207.05 (12.95)
N = 1000							
Feature 1	1	0.99 (0.03)	0.97 (0.05)	0.95 (0.22)	1 (0)	0.94 (0.07)	1 (0)
Feature 2	0.95	0.94 (0.05)	0.93 (0.09)	0.95 (0.22)	-0.87 (0.14)	0.58 (0.65)	0.1 (0.31)
Feature 3	-1	-1 (0)	-1 (0)	-1 (0)	-0.81 (0.2)	-1 (0)	-1 (0)
Time elapsed (seconds)		55.36 (8.26)	31.33 (30.12)	121.6 (12.15)	82.03 (3.18)	0.8 (0.96)	6.52 (0.16)
Number of iterations		417.85 (60.07)	51.05 (47.8)	201.15 (0.37)	431.2 (17.94)	24.35 (30.94)	204.75 (4.48)

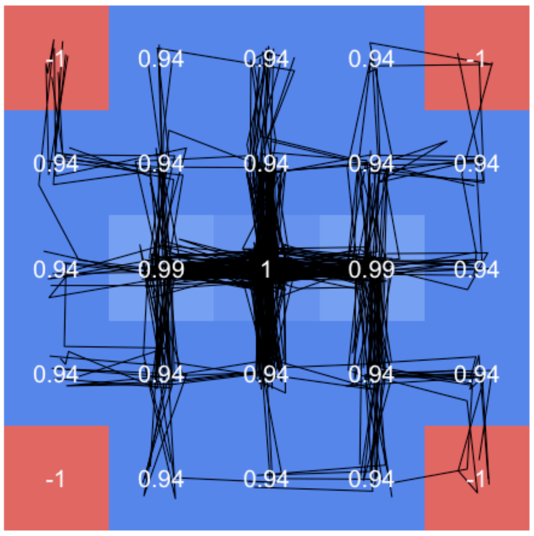
Notes: Table 1 reports the performance of three algorithms—Nested Fixed Point, Linear Programming IRL, and Projection IRL—in estimating the payoffs for cells in the GridWorld environment from Figures 1 (a) and (b). These algorithms are provided with a detailed model of the GridWorld environment, including its movement mechanics, a discount factor of 0.95, and a wind coefficient of 0.7. They also have access to data in the form of the agent's trajectories, and they are asked to use the model and the data to estimate the payoff function. The true payoff function is (1, 0.95, -1) by feature. Each algorithm's estimates undergo an affine transformation for standardization between 1 and -1. The table shows average estimates over twenty trials for each estimation method, data size (N = 100, 500, and 1000 observations), and using trajectories subject to preference shocks versus wind shocks. The sample standard deviations for estimates over the twenty trials is noted in parentheses. If the method produced an estimate of 1 for a certain feature in all twenty trials, the table will record 1 (0) because the standard deviation is zero.

Figure 2: Simulated Data on a 4-Parameter GridWorld

(a) Preference Shock Trajectories



(b) Wind Shock Trajectories



Notes: Figures 2 (a) and (b) illustrate typical paths of an agent on a 4-parameter GridWorld environment under preference shocks and then under wind shocks. The payoff function is (1, .99, .94, -1), defined on the set of features (center, left and right of center, all else except corners, corners). Under preference shocks, the agent wanders almost uniformly around the grid, avoiding only the corners. Under wind shocks, the agent focuses on the center cell unless blown by wind.

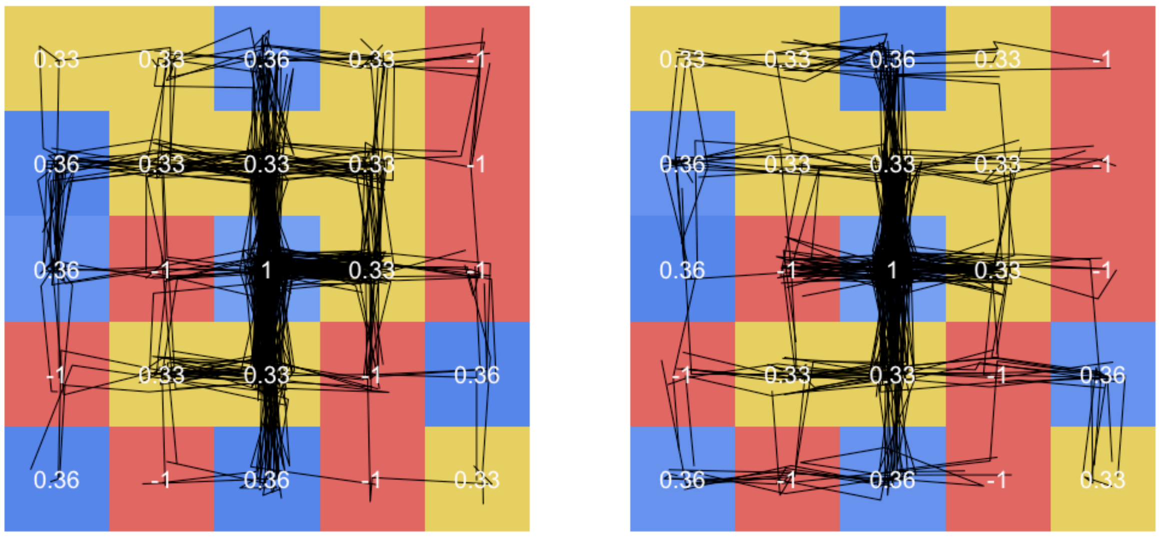
Table 2: Results for the Estimation of a 4-Parameter GridWorld

		Preference Shocks			Wind Shocks		
N = 100	True Values	NFXP	Projection IRL	LP IRL	NFXP	Projection IRL	LP IRL
Feature 1	1	0.8 (0.29)	0.41 (0.73)	0.2 (0.62)	1 (0)	0.99 (0.04)	1 (0)
Feature 2	0.99	0.81 (0.21)	0.57 (0.59)	0.1 (0.55)	-0.53 (0.44)	0.57 (0.52)	0.25 (0.55)
Feature 3	0.94	0.92 (0.09)	0.89 (0.14)	0.95 (0.22)	-0.69 (0.44)	0.07 (0.82)	-0.5 (0.69)
Feature 4	-1	-1 (0)	-0.8 (0.49)	-0.95 (0.22)	-0.24 (0.67)	-0.7 (0.54)	-0.55 (0.51)
Time elapsed (seconds)		68.55 (11.3)	10.32 (17.29)	22.97 (1.9)	71.29 (10.13)	2.42 (2.13)	6.72 (0.46)
Number of iterations		484.9 (76.25)	92.5 (156.52)	206.2 (8.79)	481.1 (71.28)	68 (60.25)	206 (6.85)
N = 500							
Feature 1	1	0.94 (0.08)	0.8 (0.19)	0.5 (0.51)	1 (0)	0.98 (0.03)	1 (0)
Feature 2	0.99	0.93 (0.08)	0.93 (0.08)	0.45 (0.51)	-0.73 (0.31)	0.68 (0.44)	0.35 (0.59)
Feature 3	0.94	0.93 (0.07)	0.92 (0.09)	0.9 (0.31)	-0.7 (0.22)	0.04 (0.77)	-0.7 (0.47)
Feature 4	-1	-1 (0)	-1 (0)	-1 (0)	-0.26 (0.51)	-0.84 (0.32)	-1 (0)
Time elapsed (seconds)		73.21 (11.94)	31.43 (17.39)	87.75 (8.77)	82.01 (6.84)	4.29 (4.81)	6.39 (0.23)
Number of iterations		479.3 (78.4)	70.6 (38.14)	202.35 (1.14)	490.3 (41.76)	131.6 (148.04)	201.25 (0.55)
N = 1000							
Feature 1	1	0.94 (0.09)	0.9 (0.16)	0.55 (0.51)	1 (0)	0.99 (0.02)	1 (0)
Feature 2	0.99	0.96 (0.05)	0.93 (0.09)	0.6 (0.5)	-0.8 (0.16)	0.56 (0.58)	0.35 (0.49)
Feature 3	0.94	0.96 (0.03)	0.93 (0.07)	0.9 (0.31)	-0.87 (0.19)	-0.08 (0.77)	-0.8 (0.41)
Feature 4	-1	-1 (0)	-1 (0)	-1 (0)	-0.36 (0.29)	-0.84 (0.3)	-0.6 (0.5)
Time elapsed (seconds)		62.77 (2.35)	55.47 (25.45)	116.44 (5.43)	86.09 (17.27)	1.74 (1.32)	6.12 (0.48)
Number of iterations		501.7 (3.45)	94.55 (40.33)	202.1 (1.02)	460.3 (104.61)	55.45 (41.26)	202.05 (1.73)

Notes for Table 2: Table 2 reports the performance of three algorithms—Nested Fixed Point, Linear Programming IRL, and Projection IRL—in estimating the payoffs for cells in the Grid-World environment from Figures 2 (a) and (b). The true payoff function is (1, 0.99, .94, -1) by feature. The table shows average estimates and standard deviations in parentheses over twenty trials for each estimation method, data size (100, 500, and 1000 observations), and using trajectories subject to preference shocks versus wind shocks.

Figure 3: Simulated Data on a 5-Parameter GridWorld

(a) Preference Shock Trajectories (b) Wind Shock Trajectories



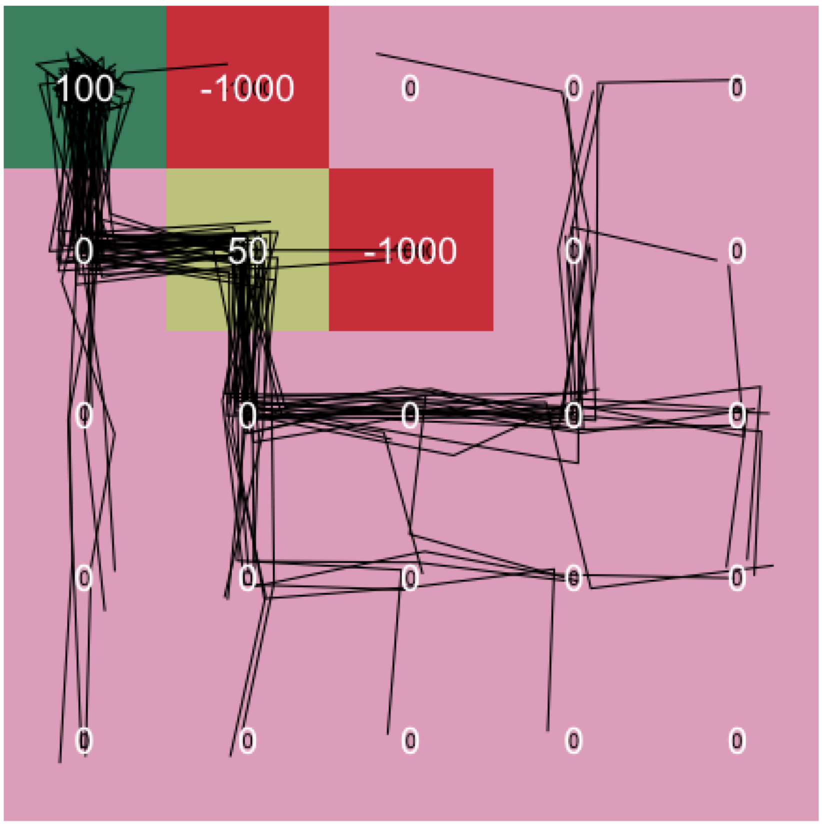
Notes: Figures 3 (a) and (b) illustrate typical paths of an agent on a 5-parameter GridWorld environment under preference shocks and then under wind shocks. The payoff function is (1, .36, .36, .33, -1). Under preference shocks, the agent wanders around the grid, often avoiding cells with a payoff of -1 and focusing slightly more on the center cell with a payoff of 1. Under wind shocks, the agent focuses only on the center cell, and once there, is blown to the left of center (receiving a payoff of -1) as many times as she is blown other directions.

Table 3: Results for the Estimation of a 5-Parameter GridWorld

		Preference Shocks			Wind Shocks		
N = 100	True Values	NFXP	Projection IRL	LP IRL	NFXP	Projection IRL	LP IRL
Feature 1	1	1 (0)	1 (0)	0.95 (0.22)	1 (0)	1 (0)	1 (0)
Feature 2	0.36	0.38 (0.36)	0.47 (0.32)	-0.1 (0.31)	0.08 (0.53)	-0.24 (0.68)	-0.35 (0.49)
Feature 3	0.36	0.43 (0.42)	0.14 (0.58)	-0.3 (0.47)	0.04 (0.58)	0.17 (0.74)	-0.35 (0.49)
Feature 4	0.33	0.38 (0.17)	0.48 (0.25)	0.3 (0.47)	-0.38 (0.42)	0.54 (0.53)	-0.35 (0.49)
Feature 5	-1	-1 (0)	-0.96 (0.1)	-1 (0)	-0.83 (0.36)	-0.26 (0.65)	-0.9 (0.31)
Time elapsed (seconds)	NA	50.81 (3.27)	27.35 (27.12)	14.68 (0.83)	75.97 (10.25)	6.19 (3.74)	6.63 (0.37)
Number of iterations	NA	502.3 (1.42)	364.05 (344.2)	204.05 (8.15)	484.85 (82.13)	183.15 (112.98)	202.3 (3.23)
N = 500							
Feature 1	1	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)
Feature 2	0.36	0.3 (0.22)	0.2 (0.36)	0 (0)	0.1 (0.21)	0.24 (0.58)	-0.15 (0.37)
Feature 3	0.36	0.33 (0.16)	0.31 (0.25)	0 (0)	0.15 (0.23)	0.18 (0.4)	0 (0)
Feature 4	0.33	0.34 (0.08)	0.36 (0.07)	0 (0)	-0.61 (0.11)	0.25 (0.34)	-0.15 (0.37)
Feature 5	-1	-1 (0)	-1 (0)	-1 (0)	-1 (0)	-0.86 (0.43)	-1 (0)
Time elapsed (seconds)	NA	54.81 (7.02)	180.41 (100.35)	57.87 (1.5)	87.64 (1.26)	3.22 (2.26)	6.38 (0.16)
Number of iterations	NA	486.85 (73.44)	631.45 (352.21)	201.2 (0.52)	502.4 (1.6)	99.25 (70.78)	201.8 (1.85)
N = 1000							
Feature 1	1	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)
Feature 2	0.36	0.37 (0.1)	0.35 (0.22)	0 (0)	-0.02 (0.15)	0.41 (0.37)	-0.1 (0.31)
Feature 3	0.36	0.38 (0.15)	0.41 (0.09)	0 (0)	0.18 (0.13)	0.19 (0.37)	0 (0)
Feature 4	0.33	0.32 (0.06)	0.3 (0.08)	0 (0)	-0.58 (0.09)	0.16 (0.43)	-0.1 (0.31)
Feature 5	-1	-1 (0)	-1 (0)	-1 (0)	-1 (0)	-0.96 (0.19)	-1 (0)
Time elapsed (seconds)	NA	65.98 (0.64)	385.58 (227.73)	112.59 (0.68)	92.78 (9.11)	3.68 (4.58)	5.73 (0.58)
Number of iterations	NA	501.1 (6.16)	692.15 (408.74)	201.4 (0.99)	502.95 (2.28)	128.7 (167.17)	201.15 (0.49)

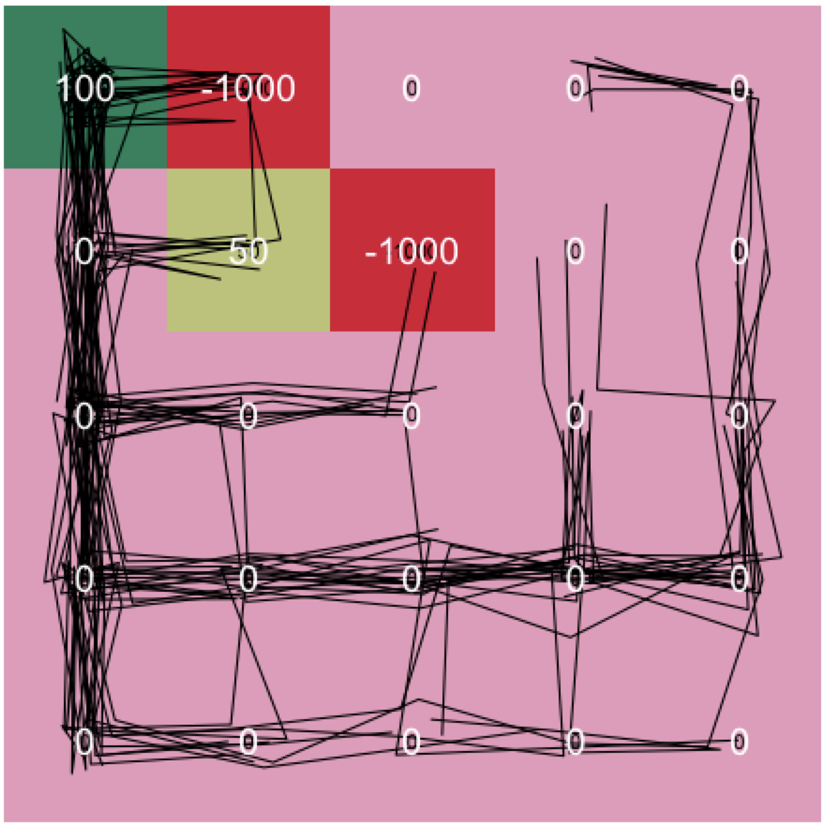
Notes for Table 3: Table 3 reports the performance of three algorithms—Nested Fixed Point, Linear Programming IRL, and Projection IRL—in estimating the payoffs for cells in the 5-parameter GridWorld environment from Figures 3 (a) and (b). The true payoff function is (1, 0.36, .36, .33, -1) by feature. The table shows average estimates and standard deviations in parentheses over twenty trials for each estimation method, data size (100, 500, and 1000 observations), and using trajectories subject to preference shocks versus wind shocks.

Figure 4: Cliff GridWorld with Preference Shocks



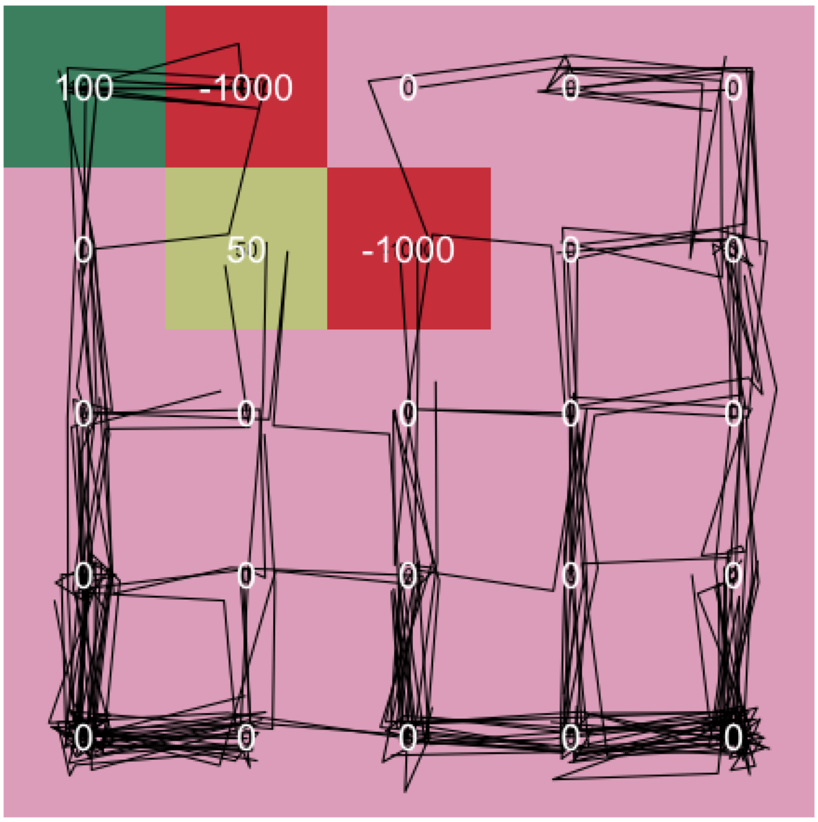
Notes: Figure 4 illustrates typical paths of an agent on a GridWorld "cliff" environment with the payoffs of (100, 50, 0, -1000), subject to N(0, 1) preference shocks each time step and a discount factor of 0.95. The large scale of the payoff function makes trajectories very close to the deterministic case with no shocks at all. Navigating the GridWorld optimally, the agent receives a random starting position and moves toward the cell with a payoff of 100, collecting 50 along the way if it doesn't take an extra time step to do so, and avoiding cells with -1000 unless that was her starting position. Once she has arrived at the cell with a payoff of 100, she stays there.

Figure 5: Cliff GridWorld with Wind Shocks (Coefficient = 0.7)



Notes: Figure 5 illustrates typical paths of an agent on a GridWorld "cliff" environment with the payoffs of (100, 50, 0, -1000), subject to wind shocks with a wind coefficient of 0.7 each time step and a discount factor of 0.95. Navigating the GridWorld optimally, the agent receives a random starting position and seeks to move toward the cell with a payoff of 100. Contrasting with Figure 4 though, she does not collect 50 along the way because she is forward thinking and knows that with a probability of 0.3, she is blown by wind and may find herself collecting the payoff of -1000. Getting to the cell with the payoff of 100 is worth it, but collecting 50 along the way is not.

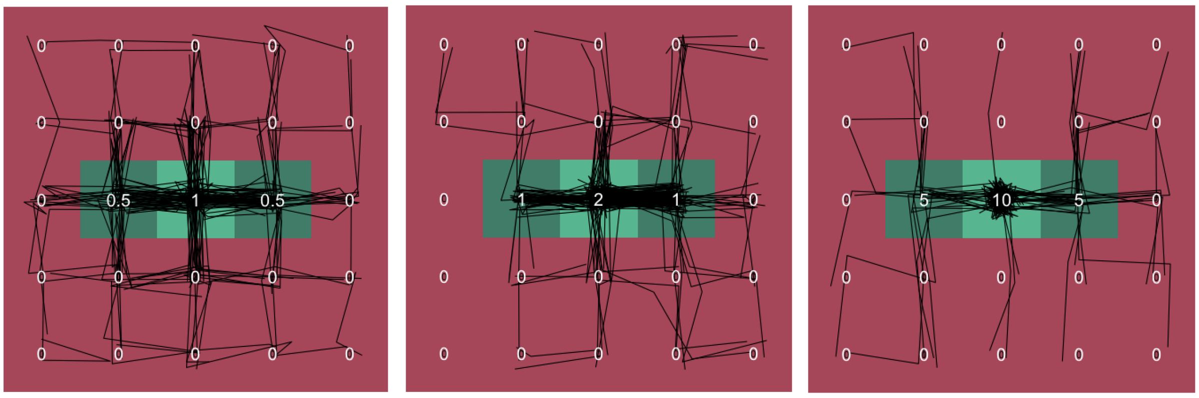
Figure 6: Cliff GridWorld with Wind Shocks (Coefficient = 0.5)



Notes: Figure 6 illustrates typical paths of an agent on a GridWorld "cliff" environment with the payoffs of (100, 50, 0, -1000), subject to wind shocks with a wind coefficient of 0.5 each time step and a discount factor of 0.95. Navigating the GridWorld optimally, the agent receives a random starting position and seeks to maximize the expected sum of discounted payoffs. Here, what is interesting is that her optimal behavior does not include moving toward cells with payoffs of 50 or 100. The wind coefficient of 0.5 means that half the time, the agent is blown by wind and can not expect to be able to select actions according to her optimal policy. Because of the high probability she will repeatedly receive payoffs of -1000 if she travels to the top of the grid, she finds it preferable to stay at the bottom of the grid, collecting zero.

Figure 7: Preference Shocks

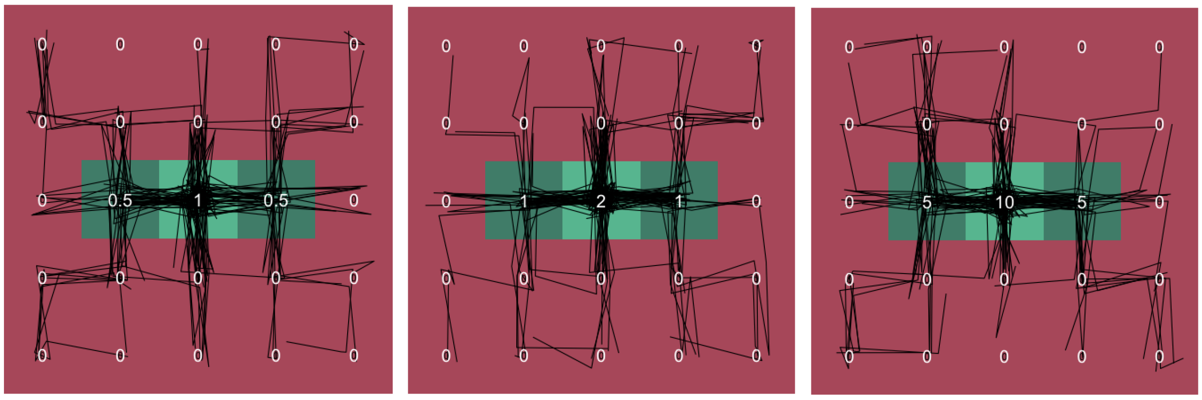
(a) Payoffs of 1, 0.5, 0 (b) Payoffs of 2, 1, 0 (c) Payoffs of 10, 5, 0



Notes: Figure 7 illustrates typical paths of an expert agent subject to N(0, 1) preference shocks on three GridWorlds with payoff functions that increase in scale: first (1, 0.5, 0), second (2, 1, 0), and then third (10, 5, 0). As the scale of the payoff function increases, the agent's behavior changes to focusing more and more on the center cell with the highest payoff because N(0, 1) shocks become less and less capable of justifying giving up the opportunity to receive the payoff in the center cell.

Figure 8: Wind Shocks

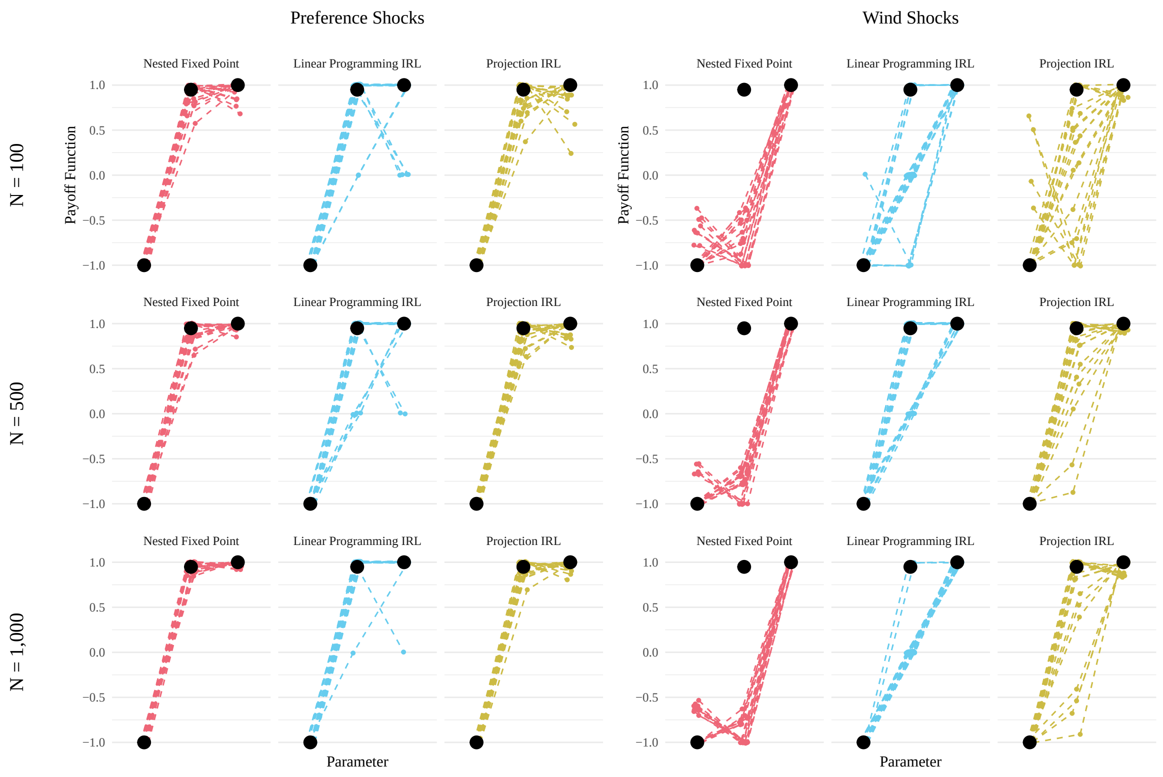
(a) Payoffs of 1, 0.5, 0 (b) Payoffs of 2, 1, 0 (c) Payoffs of 10, 5, 0



Notes: Figure 8 illustrates typical paths of an expert agent subject to wind shocks with a wind coefficient of 0.7 on three GridWorlds with payoff functions that increase in scale. Contrasting with Figure 7, the agent's behavior does not change as the scale of the payoff function increases because the agent's best policies do not change as the scale of the payoff function changes.

7.1 Appendix

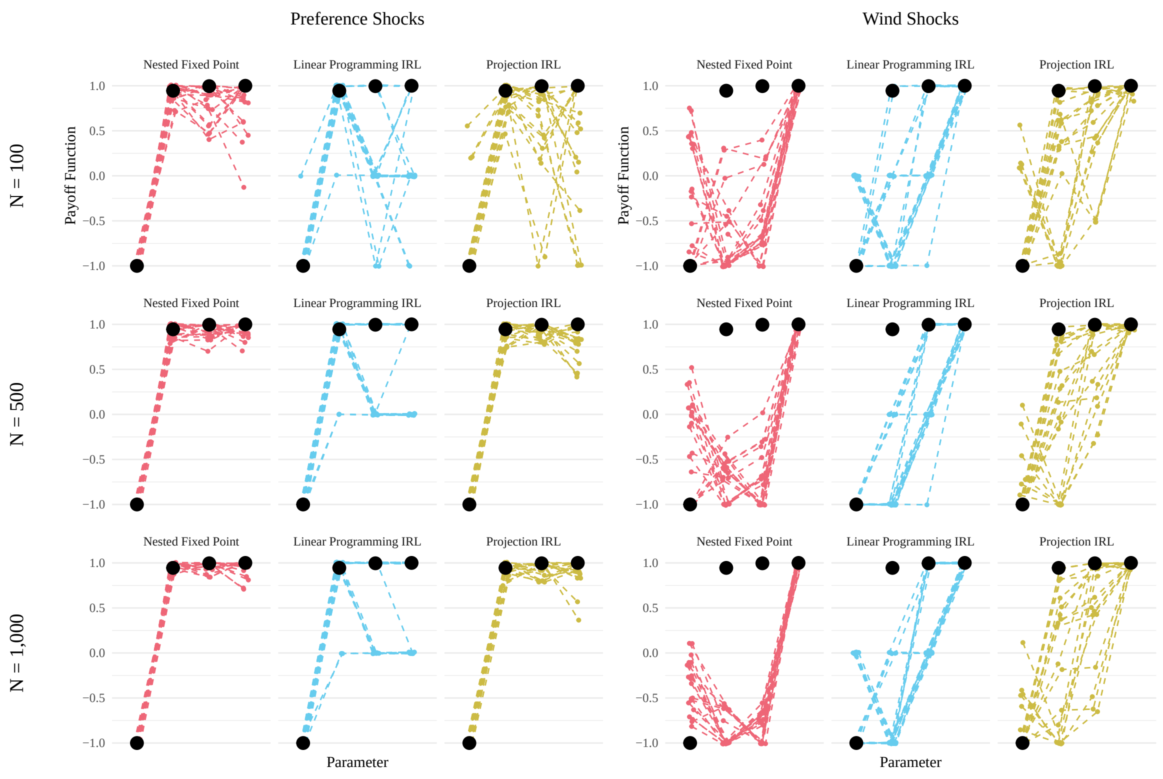
Figure 9: GridWorld Simulation Data and Results with 3 Parameters



Notes: Figure 9 shows the results for all twenty trials of each of the eighteen experiments for the 3-parameter GridWorld environment. Each dashed line represents the estimated payoff function for one trial of the experiment. Black dots represent the true payoff function of (1, .95, -1). For summaries of these results, see Table 1. Under preference shocks, Nested Fixed Point (NFXP) is able to accurately estimate the true reward function, and its error quickly approaches zero as the data size increases from 100 to 500 to 1000. Under preference shocks, Projection IRL's accuracy is almost as good as NFXP but lags slightly behind. Linear Programming IRL has the limitation that it can only make estimates in the set (1, 0, -1), but under preference shocks, it does not perform badly.

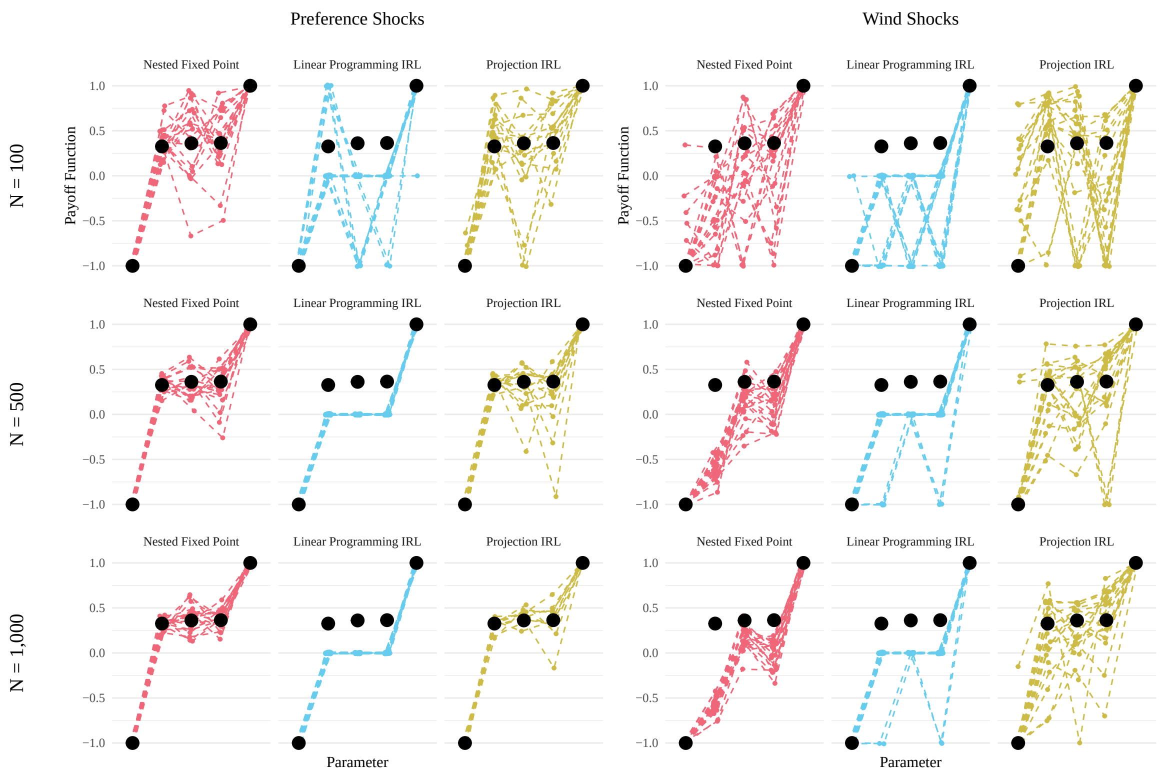
Under wind shocks however, the NFXP estimate converges toward (1, -1, -.7), which is badly biased compared to the true values of (1, .95, -1). Linear Programming IRL settles on estimating (1, 0, -1), and Projection IRL is noisy but seems to prefer (.8, 1, -1), which gets the rank incorrect but the bias is not as severe as the other methods.

Figure 10: GridWorld Simulation Data and Results with 4 Parameters



Notes: Figure 10 shows the results for all twenty trials of each of the eighteen experiments for the 4-parameter GridWorld environment. Each dashed line represents the estimated payoff function for one trial of the experiment. Black dots represent the true payoff function of (1, .99, .94, -1). For summaries of these results, see Table 2.

Figure 11: GridWorld Simulation Data and Results with 5 Parameters



Notes: Figure 11 shows the results for all twenty trials of each of the eighteen experiments for the 5-parameter GridWorld environment. Each dashed line represents the estimated payoff function for one trial of the experiment. Black dots represent the true payoff function of (1, .36, .36, .33, -1). For summaries of these results, see Table 3.