Constrained Optimization Approaches to Estimation of Structural Models

Che-Lin Su[*]
The University of Chicago
Booth School of Business

Kenneth L. Judd[†]
Hoover Institution
and NBER

This version: November 14, 2010
Last version: April 2008. First version: February, 2006 [‡]

Abstract

Estimation of structural models is often viewed as computationally difficult. This impression is partly due to a focus on the nested fixed-point (NFXP) approach. We present a direct constrained optimization approach and show that it is significantly faster than NFXP when applied to the canonical Zurcher bus repair model. We also consider estimation of discrete-choice games of incomplete information, which can admit multiple equilibria. The NFXP approach might not be valid for estimating games because it requires finding all Nash equilibria for each vector of parameters. Furthermore, the resulting likelihood function can be nonsmooth or even discontinuous, which makes the outer-loop optimization problem extremely difficult. The constrained optimization approach preserves the smooth property of the likelihood function and structural equations and does not need to find all the equilibria. Monte Carlo results show that the finite sample properties of our method are superior to those of two-step estimators and of the nested pseudo-likelihood estimator.
keywords: structural estimation, constrained optimization, dynamic discrete choice models, games of incomplete information, multiple equilibria

[*]Contact Author. `E-mail:che-lin.su@chicagobooth.edu`

[†]`E-mail:kennethjudd@mac.com`

[‡]We have greatly benefited from the comments and suggestions of a co-editor and four anonymous referees. We thank Jaap Abbring, Pat Bajari, Steve Berry, John Birge, Richard W. Cottle, Jean-Pierre Dubé, Jeremy Fox, Ron Gallant, Phil Haile, Lars P. Hansen, James Heckman, Günter Hitsch, Panle Jia, Tobias Klein, Sven Leyffer, Todd Munson, Jorge Nocedal, Ariel Pakes, Peter Rossi, John Rust, and seminar participants at the University of Chicago, Harvard University, Stanford University, UIUC, Yale University and various conferences for helpful discussions and comments. We are grateful to John Rust for providing us the example of games of incomplete information studied in Section 5. We acknowledge the financial support from the NSF under Grant SES-0631622. Su is grateful for the financial support from the IBM Corporation Faculty Research Fund at the University of Chicago Booth School of Business.

1 Introduction

Structural estimation of economic models is an important approach for analyzing economic data. However, the computational burden of estimating structural models raises concern. It is commonly believed that computational demands make it difficult to implement the most powerful and efficient statistical methods. For example, Rust (1987) proposes a computational strategy for maximum likelihood estimation of single-agent dynamic programming models, an approach termed the nested fixed-point (NFXP) algorithm. Unfortunately, NFXP is computationally demanding because it repeatedly takes a guess for structural parameters and solves for the corresponding endogenous economic variables with high accuracy for each guess of these parameters. Furthermore, NFXP may not be valid for estimating games. One feature that distinguishes games from single-agent dynamic decision models is that games can admit multiple equilibria. When applying NFXP for maximum likelihood estimation of games, one needs to find all Nash equilibria for each vector of parameters considered in order to calculate the corresponding likelihood value. Finding all the equilibria in a game is a daunting computational task.[1] Even if one can solve for all the equilibria for every guess of structural parameters, the resulting likelihood function can be nonsmooth or even discontinuous, which makes the outer-loop optimization problem extremely difficult. This concern about the computational burden of implementing NFXP has led to the development of estimators that are computationally light, for example, the well-known two-step estimator by Hotz and Miller (1993) for estimating singe-agent dynamic discrete-choice models, and two-step estimators by Bajari, Benkard, and Levin (2007), Pakes, Ostrovsky, and Berry (2007), and Pesendorfer and Schmidt-Dengler (2008) for estimating dynamic games. However, two-step estimators are not asymptotically efficient and can be severely biased in finite samples. Recognizing these limitations, Aguirregabiria and Mira (2002, 2007) propose the nested pseudo-likelihood (NPL) estimator, a recursive two-step pseudo-likelihood estimator, to improve the asymptotic properties and finite-sample performance of two-step estimators.

We present a direct constrained optimization approach, termed the Mathematical Program with Equilibrium Constraints (MPEC; see Luo, Pang, and Ralph (1996)), for structural estimation. The idea behind the MPEC approach is simple: choose structural parameters and endogenous economic variables so as to maximize the likelihood (or minimize the moment conditions) of the data subject to the constraints that the endogenous economic variables

[1]See the discussion in Besanko, Doraszelski, Kryukov, and Satterthwaite (2010) and Judd, Renner, and Schmedders (2010).

are consistent with an equilibrium for the structural parameters. When formulating the estimation problem in this way, one can simply write down expressions that define the likelihood or the moment as the objective function and the equilibrium equations as constraints and use one of the state-of-the-art constrained optimization solvers to solve the estimation problem. The advantages of our approach are twofold. First, we do not repeatedly solve for an equilibrium for each guess of structural parameters; in fact, the only equilibrium that needed to be solved exactly is the one associated with the final estimate of parameters. Modern constrained optimization algorithms do not enforce constraints to be satisfied until the last iteration in the search process. This feature makes it possible for our approach to be faster than NFXP. Second and more important, in the context of estimating games of incomplete information, our approach preserves the smooth property of the objective (likelihood or moment) function and structural equations, while the objective function in NFXP might be nonsmooth or discontinuous. This feature allows constrained optimization solvers to be applied directly to estimation of games and makes estimating games conceptually no more difficult than estimating than estimating dynamic programming models.

In many ways, the MPEC approach is not new, since it builds on ideas and methods that have been developed and analyzed in the statistics and econometrics literature. In particular, examples that we will examine below are constrained estimation problems of the kind exposited in Aitchison and Silvey (1958), Silvey (1975), Gallant and Holly (1980), Wolak (1987, 1989), and Gallant and Tauchen (1989).[2] We demonstrate in this paper that the constrained optimization approach is computationally feasible. Researchers can also access state-of-the-art constrained optimization solvers used in our numerical implementation on the NEOS Server.[3]

The rest of the paper is organized as follows. In Section 2, we give a brief overview on numerical optimization. We describe the basic ideas behind algorithms for solving constrained optimization problems and explain why constrained optimization methods are efficient and capable of solving problems with tens or hundreds of thousands of variables and constraints on a laptop or millions of variables on a workstation. In Section 3, we use a simple demand example to illustrate the main idea of the MPEC approach. In Section 4, we study the standard single-agent, dynamic discrete-choice models as in Rust (1987). We present the MPEC formulation for estimating this model. We show the equivalence in solution be-

[2]We thank Ron Gallant for pointing us to the constrained estimation literature.

[3]The NEOS Server is at `http://www.neos.gov`. For reference, see Gropp and Moré (1997), and Czyzyk, Mesnier, and Moré (1998).

tween NFXP and MPEC, that is, a solution for NFXP is also a solution for MPEC and vice versa. The Monte Carlo results show that while estimates from NFXP and MPEC are almost identical, MPEC is significantly faster than NFXP, in particular when the discount factor in the dynamic programming model is close to 1. In Section 5, we consider a model of static discrete-choice game of incomplete information studied by Rust (2008). We show in an example that the likelihood function of NFXP for estimating this game is discontinuous. One can expect the outer-loop optimization problem in NFXP to be computationally difficult because there is no reliable algorithm for maximizing a discontinuous function. In contrast, the MPEC formulation, defined over the space of both structural parameters and equilibrium, gives a smooth constrained optimization problem. The advantage of using this formulation is that the presence of multiple equilibria has no impact on the smoothness of the optimization problem solved by the MPEC approach. As a result, the problem of estimating games is qualitatively no more difficult to solve than the standard single-agent dynamic models. In Monte Carlo experiments, we study small sample properties of the MPEC estimator, two-step pseudo maximum likelihood (2S-PML), two-step least squares (2S-LS), and nested pseudo likelihood (NPL) with data generated by using best-reply stable and/or best-reply unstable equilibria. The results show that MPEC estimator is superior to other estimators considered. When data are generated from best-reply stable equilibria, all estimators perform well, and the mean estimates are within 1 or 2 standard errors from true parameters values. In term of computational speed, NPL needs more than 120 NPL iterations to converge and is the slowest method: about 40 times slower than MPEC and 100 times slower than two-step estimators. When data is generated from equilibria that are not stable under best-reply iteration, the MPEC estimator still recovers the structural parameters successfully, while two-step estimators reject the true parameter values. NPL fails to converge in every run, suggesting that NPL is not reliable unless the underlying equilibria in the data generating process are stable under best-reply iteration; see also the discussion by Pesendorfer and Schmidt-Dengler (2010). We summarize our conclusions in Section 6.

2 Overview of Numerical Optimization Methods

We give an overview of numerical methods for solving unconstrained and constrained optimization. We do not provide a comprehensive review. Instead, we describe the main ideas in solving optimization problems and explain why modern numerical optimization methods and solvers can solve large-scale problems with tens or hundreds of thousands of variables and constraints on laptop computers or millions of variables and constraints on workstations. We refer readers to an excellent book by Nocedal and Wright (2006) and the reference within for further details on numerical algorithms and their implementations for solving nonlinear equations and optimization problems.

The central idea in numerical optimization is to apply a Newton-type method to solve the system of first-order conditions of the corresponding (unconstrained or constrained) optimization problem. For example, consider an unconstrained optimization problem

$$\max_{\theta} f(\theta), \tag{1}$$

where $\theta \in R^n$ and $f : R^n \to R$ is at least twice-differentiable. If θ^* is a local solution to this problem, then θ^* satisfies the following system of first-order conditions:

$$\nabla f(\theta) = 0,$$

where $\nabla f(\theta) \in R^n$ is the gradient of f. Since f is at least twice-differentiable, the second-order derivatives $\nabla^2 f(\theta)$ exist. We then can apply the Newton's method to find a solution to the system of first-order conditions.

The same idea applies to constrained optimization problems. Consider a constrained optimization problem

$$\begin{array}{rl} \max\limits_{(\theta,y)} & f(\theta, y) \\ \text{subject to} & g(\theta, y) \leq 0 \\ & h(\theta, y) = 0, \end{array} \tag{2}$$

where $\theta \in R^n, P \in R^m$ and $f, g,$ and h are at least twice-differentiable. Let (θ^*, y^*) be a local solution to the problem (2). Assuming that a constraint qualification (CQ) holds at (θ^*, y^*),[4] then there exists a vector of Lagrange multipliers (μ^*, λ^*) such that the following

[4]The simplest one is the linear-independence constraint qualification (LICQ); i.e., the gradients of *binding* constraints in $g(\theta^*, y^*) = 0$ and $h(\theta^*, y^*) \leq 0$ are linearly independent.

first-order conditions are satisfied at $(\theta^*, y^*, \mu^*, \lambda^*)$:

$$\begin{aligned}
\nabla_\theta f(\theta, y) - \lambda^{\mathrm{T}} \nabla_\theta g(\theta, y) - \lambda^{\mathrm{T}} \nabla_\theta h(\theta, y) &= 0 \\
\nabla_y f(\theta, y) - \lambda^{\mathrm{T}} \nabla_y g(\theta, y) - \lambda^{\mathrm{T}} \nabla_y h(\theta, y) &= 0 \\
\mu \geq 0, \quad g(\theta, y) \leq 0, \quad \mu^{\mathrm{T}} g(\theta, y) &= 0 \\
\lambda \text{ free}, \quad h(\theta, y) &= 0.
\end{aligned} \tag{3}$$

Again, we apply a Newton-type method to solve the system of first-order conditions (3) for $(\theta^*, y^*, \mu^*, \lambda^*)$. To accommodate the inequality constraints on g and μ, one can adopt a line-search strategy in a Newton's method to ensure that both the equations and inequalities in the first-order conditions are satisfied at a solution.

Practical implementations of constrained optimization algorithms incorporate several nice and important features that make state-of-the-art solvers efficient in speed and capable of solving large-scale problems. We summarize these features below.

Using first-order and second-order derivatives. Most optimization methods require knowledge of derivatives. In fact, in solving an optimization problem, much of the computational time is devoted to evaluating functions and obtaining first-order derivatives, such as gradients and Jacobians, and second-order derivatives, such as Hessians. For most solvers, users should provide at least first-order derivatives for the solver to perform well. Supplying exact derivatives, instead of using finite-differencing derivatives, will significantly decrease the number of iterations needed and improve the robustness of the search process. While the cost for calculating derivatives was a concern in the past, the development of automatic differentiation has eliminated this as a serious problem; see Griewank and Walther (2008). We give the basic ideas of automatic differentiation in Appendix B.

Allowing infeasibility during the search process. Solving nonlinear equations or equivalently finding a feasible solution to constraints is computationally costly. In the implementation of optimization algorithms, intermediate iterates are allowed to be infeasible, and constraints, $g(\theta, y) \leq 0, h(\theta, y) = 0$ do no need to be satisfied until the last few iterations. The advantage of allowing the constraints to be violated during the search process is that one does not waste effort in finding a feasible solution that is not optimal. Furthermore, allowing the algorithm to move into the infeasible region can result in needing few iterations to converge to an optimal solution.

Enabling quadratic convergence. Newton-typ methods can be shown to be quadratically convergent near a solution.[5] This nice property enables Newton-type methods to converge in a few iterations and hence reduce the number of function and gradient evaluations when the iterates are in the neighborhood of a solution.

Exploring sparsity structure in the constraint Jacobian and Hessian. While the number of variables and constraints may be large in an optimization problem, what really matters in the computation are the non-zeros in the constraint Jacobian and Hessian. Suppose that there are n variables and m constraints in an optimization problem. If the Jacobian and Hessian are dense, the memory required for a constrained optimization method is on the order of $(n+m)*(n+m)$. However, for sparse Jacobian and Hessian, the time and memory needed is usually on the order of the number of nonzeros in Jacobian and Hessian. In many applied models, sparsity structure in constraint Jacobian and Hessian arises naturally. For example, the constraint Jacobian of the Bellman equation in Rust's Zucher bus repair model has a sparse diagonal structure, as illustrated in Figure 1. The number of nonzeros in this constraint Jacobian is on the order of n, the number of variables. Modern solvers explore sparsity structure in matrices and utilize sparse numerical linear algebra techniques to speed the computation and save the memory needed in the computation.

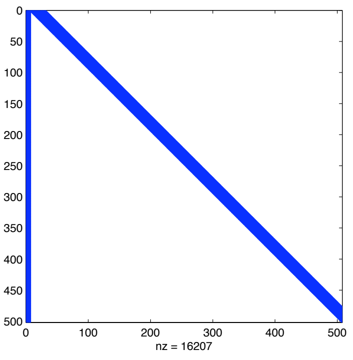


Figure 1: Sparsity pattern in constraint Jacobian of the Zucher model

[5]Quadratic convergence means that when the error is small (less than 1), the square of the error at one iteration is proportional to the error at the next iteration.

3 MPEC Approach to Estimation

In this section, we give a general formulation of MPEC approach for estimating structural models. We discuss the asymptotic distribution of the estimator using the MPEC approach.

3.1 Formulation

Suppose that an economic model is described by parameters θ, state x, and a vector of endogenous variables σ. The data is $X = \{x_m, d_m\}_{m=1}^{M}$, where x_m is the observed state, d_m is the observed equilibrium outcome of the underlying decision model and M is the number of data points. One example is a dynamic programming problem where θ are the parameters for costs, profits, laws of motion and stochastic shocks, and where σ is the policy function of a decisionmaker in an environment described by the structural parameters θ. In general, σ will depend on θ through a set of equilibrium conditions such as first-order conditions, Bellman equations, market balance conditions, etc., which are expressed in the system of equations denoted by

$$h\left(\theta, \sigma\right) = 0. \tag{4}$$

For any given θ, let $\Sigma\left(\theta\right)$ be the set of all σ such that $G\left(\theta, \sigma\right) = 0$, that is,

$$\Sigma(\theta) := \{\sigma : h(\theta, \sigma) = 0\}.$$

We let $\boldsymbol{\sigma}(\theta)$ denote the element in the solution set $\Sigma(\theta)$. For applications such as single-agent dynamic discrete choice models, $\boldsymbol{\sigma}(\theta)$, representing the expected value function, is uniquely determined and the set $\Sigma(\theta)$ is a singleton for any given θ. For other applications such as games, there can be multiple equilibria in $\boldsymbol{\sigma}(\theta)$ and hence, the set $\Sigma(\theta)$ can be multi-valued for some θ.

We use $L\left(\theta, \boldsymbol{\sigma}(\theta); X\right)$ to denote the log-likelihood of observing the data X conditional on parameter θ.[6] If $\boldsymbol{\sigma}(\theta)$ is unique for all θ, the maximum likelihood estimator is defined as

$$\theta = \underset{\theta}{\operatorname{argmax}} \frac{1}{M} L\left(\theta, \boldsymbol{\sigma}(\theta); X\right). \tag{5}$$

[6] If there are multiple solutions in $\boldsymbol{\sigma}(\theta)$, we assume that the data comes from a single equilibrium so that one can write the likelihood function.

If there are multiple solutions in $\boldsymbol{\sigma}(\theta)$, with the assumption that data comes from a single equilibrium, the maximum likelihood estimator is defined as

$$\theta = \underset{\theta}{\operatorname{argmax}} \frac{1}{M} \left\{ \max_{\boldsymbol{\sigma}(\theta) \in \Sigma(\theta)} L\left(\theta, \boldsymbol{\sigma}(\theta); X\right) \right\}. \tag{6}$$

Notice that the consistency requirement between θ and σ described in (4) is implicitly taken into account by $\boldsymbol{\sigma}(\theta)$ in the likelihood function.

This formulation implicitly expresses the dependence of L on σ. In many applications, σ is more directly related to the likelihood than some components of θ. For example, in a life-cycle problem, the likelihood of an observation depends on the consumption and labor decisions, not on, for example, the elasticity of labor supply which could be a component of θ. Elasticities affect likelihood only indirectly through their impact on decision rules.

We construct an *augmented likelihood function*[7], denoted by $L\left(\theta, \sigma; X\right)$, by simply replacing the equilibrium solution $\boldsymbol{\sigma}(\theta)$ in the likelihood function by any σ. This function explicitly expresses the dependence of the likelihood on σ and makes no requirement that θ and σ are consistent with the equilibrium conditions of the economic model. For example, in a life-cycle model, the policy function σ together with the components of θ related to exogenous shock processes defines a stochastic process for the observables. A stochastic process is well-defined even if σ and θ are not consistent with equilibrium. An augmented likelihood function allows us to treat σ and θ independently when we compute the likelihood.

When we estimate the structural parameters θ, we need σ to be consistent with θ. This is achieved by imposing the equilibrium conditions (4). Therefore, we can formulate maximum likelihood estimation problem as a constrained optimization problem

$$\begin{array}{ll} \max\limits_{(\theta,\sigma)} & \dfrac{1}{M} L\left(\theta, \sigma; X\right) \\ \text{subject to} & h\left(\theta, \sigma\right) = 0. \end{array} \tag{7}$$

One can show that the two formulations (6) and (7) (or (5) and (7) if $\boldsymbol{\sigma}(\theta)$ is unique) are mathematically equivalent. We state the equivalence result formally when we discuss specific applications in the sections below .

The MPEC approach described above is not limited to maximum likelihood estimation.

[7] We thank Peter Rossi for suggesting this term.

It can be applied to any loss (or gain) function, such as least squares, weighted GMM, simulated maximum likelihood, etc., as the objective. See, for example, Dubé, Fox and Su (2008) on GMM estimator for random-coefficient demand estimation.

3.2 Inference

Computing standard errors for maximum likelihood estimates requires the computation of the Hessian of the likelihood function $L\left(\theta, \boldsymbol{\sigma}(\theta); X\right)$. This Hessian matrix or an approximation of it can be obtained directly if one uses a numerical procedure to solve (5) or (6). While the MPEC approach is simply a different way to compute the maximum likelihood estimator, conducting inference is more complex with the MPEC approach. Aitchison and Silvey (1958) consider maximum likelihood estimation of parameters with equality constraints and study the conditions under which a maximum exists.[8] They also derive asymptotic distributions of estimates. Although their model does not include endogenous variables, the derivation and analysis there can be carried over to the our setting. Below we discuss the large sample properties of the maximum likelihood estimates based the MPEC approach without including the proof. We refer the reader to Aitchison and Silvey (1958) for detailed derivation and proof and Silvey (1975) for a concise discussion.

Proposition 1. Let $\theta_0 \in \Theta$ be the true parameter vector in the population and let σ_0 be the corresponding endogenous variables such that $h(\theta_0, \sigma_0) = 0$. Denote $\vartheta = (\theta, \sigma)$, $H_0 = \frac{\partial h(\vartheta_0)}{\partial \vartheta}$, and $B_0 = \mathbb{E}\left[\frac{\partial L(\vartheta_0)}{\partial \vartheta}\frac{\partial L(\vartheta_0)}{\partial \vartheta'}\right]$. Assume the following conditions hold:

(i) Θ is compact.

(ii) θ_0 is in the interior of Θ.

(iii) The observations $\{x_m, d_m\}$ are independent.

(iv) The likelihood function L is at least three-times differentiable and the constraint h is at least twice differentiable.

(v) The matrix B_0 is positive definite.

Under some regularity conditions, the random variable $\sqrt{M}(\vartheta^* - \vartheta_0)$ is asymptotically normally distributed with

$$\sqrt{M}(\vartheta^* - \vartheta_0) \xrightarrow{d} \mathcal{N}(0, P_0),$$

where $P_0 = B_0^{-1}[I - H_0(H_0' B_0^{-1} H_0^{-1})H_0' B_0^{-1}]$.

[8]Wolak (1987) analyzes problems with both equality and inequality constraints.

4 Single-Agent Dynamic Discrete Choice Models

We consider a single-agent dynamic discrete choice model as in Rust (1987).We choose the Rust model since it has become the standard framework for modeling dynamic discrete-choice problems. It is also often used to evaluate the performance of alternative estimators. We first describe the nested fixed-point algorithm (NFXP) proposed by Rust for estimating this model. As an alternative, we present the MPEC formulation for this model and show the equivalence in formulation and solution between NFXP and MPEC. We compare the numerical performance of these two methods in several Monte Carlo experiments.

4.1 Bus Engine Replacement Problem: The Model

The bus engine replacement problem in Rust (1987) considers the dynamic decisions faced by a repairman, Harold Zurcher. In each period, a bus arrives in state x, the accumulated mileage since the last engine replacement. Given the observed mileage state x and other unobserved state variables ε, Harold Zurcher decides whether to perform extensive repairs on a bus or to implement less costly activities. Formally, the discrete decision in time t is defined as follows:

$$d_t = \begin{cases} 1, & \text{replacing the engine} \\ 0, & \text{performing regular maintenance.} \end{cases}$$

If Zucher chooses to replace the engine, the mileage state x after the replacement will be reset to 0 before the bus goes out to travel for another period.

Let $c\left(x;\theta_1\right)$ be the expected operating costs per period at mileage x, where θ_1 is a vector of parameters to be estimated. Let RC denote the expected replacement cost to install a new engine, net of any scrap value of the old engine. Given the state (x, ε) and action d, the utility per period is

$$u(x,d,\varepsilon;\theta_1,RC) = \nu(x,d;\theta_1,RC) + \varepsilon(d),$$

where

$$\nu(x,d;\theta_1,RC) = \begin{cases} -c(x;\theta_1), & \text{if } d = 0 \\ -RC - c(0;\theta_1), & \text{if } d = 1, \end{cases}$$

and $\varepsilon = (\varepsilon(0), \varepsilon(1))$ represents utility shocks.

The state (x_t, ε_t) evolves after a decision d_t is chosen, and the evolution is described by the transition probability $p(x_{t+1}, \varepsilon_{t+1}|x_t, \varepsilon_t, d_t; \theta_2, \theta_3)$, where θ_2 and θ_3 are parameters to be estimated. Note that the transition probability p is *Markovian* since it depends only on current state (x_t, ε_t). Let $\beta \in (0, 1)$ be a discount factor. Given the current state (x_t, ε_t), the agent chooses a sequence of decision to maximize the total expected discounted utility over infinite horizons:

$$\max_{\{d_t, d_{t+1}, d_{t+2}, \ldots\}} \mathbb{E}_p \left[\sum_{\tau=t}^{\infty} \beta^{\tau-t} u(x, d, \varepsilon; \theta_1, RC) \right], \tag{8}$$

where the expectation is taken over the stochastic evolution of future states.

Define the value function V by

$$V(x_t, \varepsilon_t) = \max_{\{d_t, d_{t+1}, d_{t+2}, \ldots\}} \mathbb{E}_p \left[\sum_{\tau=t}^{\infty} \beta^{\tau-t} u(x_\tau, d_\tau, \varepsilon_\tau; \theta_1, RC) \right]. \tag{9}$$

Since the transition probability p is Markovian, the optimal decision rule for the infinite-horizon decision problem is time invariant and we can drop the time index t. The infinite-horizon optimal decision problem in (8) can be formulated as a solution to the Bellman equation

$$V(x, \varepsilon) = \max_d \left\{ \nu(x, d; \theta_1, RC) + \varepsilon(d) + \beta \int_{x'} \int_{\epsilon'} V(x', \varepsilon') p(x', \varepsilon'|x, \varepsilon, d; \theta_2, \theta_3) dx' d\varepsilon'. \right\}, \tag{10}$$

where (x', ε') denote the state variable in the next period.

Rust (1987) discusses the computational difficulties in estimating the model specified above and introduces the *conditional independence* assumption on the transition probability p to simplify the estimation problem.

Assumption 1: Conditional Independence. The Markov transition probability of state variables (x, ε) can be decomposed as $p(x', \varepsilon'|x, \varepsilon, d; \theta_2, \theta_3) = p_2(\varepsilon'|x'; \theta_2) p_3(x'|x, d; \theta_3)$.

With Assumption CI, we define the expected value function

$$EV(x) = \int_{\varepsilon} V(x, \varepsilon) p_2(\varepsilon|x; \theta_2) d\varepsilon,$$

and the choice-specific value function

$$EV(x,d) = \nu(x,d;\theta_1,RC) + \beta \int_{x'} EV(x')p_3(x'|x,d;\theta_3)dx'.$$

Taking the expectation with respect to ε on both side of the Bellman equation (10) and using $EV(x)$ and $EV(x,d)$ defined above, one can obtain the integrated Bellman equation

$$EV(x) = \int_{\varepsilon'} \max_d \{EV(x,d)\}\, p_2(\varepsilon'|x;\theta_2)d\varepsilon'. \tag{11}$$

Let $\theta = (RC,\theta_1,\theta_2,\theta_3)$. Assuming $p_2(\varepsilon'|x;\theta_2)$ is a multivariate extreme value distribution, Rust derives the multinomial logit formula to characterize *conditional choice probability*:

$$P(d|x;\theta) = \frac{\exp\{\nu(x,d,\theta) + \beta EV(x,d)\}}{\displaystyle\sum_{d'\in\{0,1\}} \exp\{\nu(x,d',\theta) + \beta EV(x,d')\}}, \tag{12}$$

where $P(d|x;\theta)$ is the probability of choosing decision d given the mileage state x and parameter vectors θ. Furthermore, the choice-specific value function $EV(x,d)$ in (12) is the unique fixed point solution to the contraction mapping

$$EV(x,d) = \int_{x'=0}^{\infty} \log\left[\sum_{d'\in\{0,1\}} \exp\{\nu(x',d';\theta_1,RC) + \beta EV(x',d')\}\right] p_3(dx'|x,d,\theta_3). \tag{13}$$

Since the mileage state space x is continuous, $EV(x,d)$ is a functional. Following Rust (1987), we discretize the mileage state space x by K grid points $\hat{\mathbf{x}} = \{\hat{x}_1,\ldots,\hat{x}_K\}$ with $\hat{x}_1 = 0$. We require the fixed point equation (13) to be satisfied at those grid points. Given the current state $\hat{x}_k$, we assume that the mileage state in the next period x' can move up at most J grid points and specify the Markov transition probabilities as

$$p_3(x'|x_k,d,\theta_3) = \begin{cases} \Pr\{x' = \hat{x}_{k+j}|\theta_3\}, & \text{if } d=0 \\ \Pr\{x' = \hat{x}_{1+j}|\theta_3\}, & \text{if } d=1, \end{cases} \tag{14}$$

for $j = 0,1,\ldots,J$. With a slight abuse of notation, we now denote

$$EV = (EV(\hat{x}_1,0),\ldots,EV(\hat{x}_K,0),EV(\hat{x}_1,1),\ldots,EV(\hat{x}_K,1)) \in R^{2K}$$

as a vector of expected value function and rewrite the fixed point equation (13) as

$$EV(\hat{x}_k, d) = \sum_{j=0}^{J} \log \left[\sum_{d' \in \{0,1\}} \exp\{\nu(x', d'; \theta_1, RC) + \beta EV(x', d')\} \right] p_3(x' | \hat{x}_k, d, \theta_3). \quad (15)$$

To simplify the notation, we denote the fixed-point equation (15) as

$$EV = T(EV, \theta). \quad (16)$$

This characterization allows us to calculate the conditional choice probability $P(d|x;\theta)$ for a given vector of structural parameters θ by first solving the fixed-point equation (15) for $EV(x, d')$ for all d'. We refer readers to Rust (1987, 1994) for detailed derivation and discussions.[9]

4.2 Maximum Likelihood Estimation

The econometrician observes the mileage state and the corresponding decision in each period and for each bus in the data. Let $X^i = (x_t^i, d_t^i)_{t=1}^T$ be the time series data for bus $i = 1, \ldots, M$; x_t^i is the mileage state of bus i examined in period t and d_t^i is the replacement decision made for that bus. For simplicity, we assume x_t^i stays on the grid $\hat{\mathbf{x}} = \{\hat{x}_1, \ldots, \hat{x}_K\}$.[10] The full data including all buses is denoted by $X = (X^i)_{i=1}^M$.

Rust (1987) proposes to estimate structural parameters θ using maximum likelihood estimation. The likelihood of observing data X^i for bus i is

$$l_i(X^i; \theta) = \prod_{t=2}^{T} P(d_t^i | x_t^i; \theta) p_3(x_t^i | x_{t-1}^i, d_{t-1}^i; \theta_3),$$

where P is given by (12). The likelihood function of the full data $X = (X^i)_{i=1}^M$ is

$$l(\theta) = \prod_{i=1}^{M} l_i(X^i; \theta) = \prod_{i=1}^{M} \prod_{t=2}^{T} P(d_t^i | x_t^i; \theta) p_3(x_t^i | x_{t-1}^i, d_{t-1}^i; \theta_3), \quad (17)$$

[9]With the Markov transition probabilities $p_3(x'|x_k, d, \theta_3)$ defined in (14), we have $EV(\hat{x}_k, 1) = EV(\hat{x}_1, 0)$ for all $k = 1, \ldots, K$. As a result, one can define $EV = (EV(\hat{x}_1, 0), \ldots, EV(\hat{x}_K, 0)$. See Rust (2000, p. 14).

[10]If x_t^i is not on the grid $\hat{\mathbf{x}}$, then we around x_t^i up or down to the nearest grid point.

and the log-likelihood function is denoted by

$$L(\theta) = \log l(\theta) = \sum_{i=1}^{M} \sum_{t=2}^{T} \log \left(P(d_t^i | x_t^i; \theta) \right) + \sum_{i=1}^{M} \sum_{t=2}^{T} \log \left(p_3(x_t^i | x_{t-1}^i, d_{t-1}^i; \theta_3) \right) \}. \tag{18}$$

We infer the unknown parameter vector $\theta = (RC, \theta_1, \theta_3)$ by maximizing the log-likelihood function

$$\max_{\theta} \frac{1}{M} L(\theta). \tag{19}$$

To calculate the value of the log-likelihood function for a given θ, one needs to know the value of the term $P(d_t^i | x_t^i; \theta)$, which in turn is a solution of the fixed-point equation (15). This observation motivates the nested fixed-point algorithm proposed in Rust (1987). NFXP consists of an outer loop and an inner loop. In the outer loop, a maximization procedure is carried out to maximize the log-likelihood function $L(\theta)$ by changing the structural parameters θ; in the inner loop, for a given θ, one solves the fixed-point equation (15) for EV to calculate the conditional choice probabilities $P(d_t^i | x_t^i; \theta)$, and evaluate the log-likelihood function $L(\theta)$. Furthermore, Rust (1988) proves that the log-likelihood function $L(\theta)$ is a smooth function of θ, an attractive property that makes Newton's method (e.g., BHHH) a valid approach. Rust shows that one can compute the exact derivatives of $L(\theta)$ with respect to θ using implicit function theorem. Supplying exact derivatives in the implementation of Newton's method makes the algorithm faster and more robust than when finite-difference methods are used to approximate derivatives.

NFXP requires the fixed-point equation (15) to be solved very accurately for every vector of parameters θ considered in the maximization procedure.[11] Aguirregabiria and Mira (2002) point out that repeatedly solving the fixed-point equation is not computationally efficient since it is not necessary to enforce the fixed-point equation to be satisfied during the search process in NFXP's outer loop. They propose a nested pseudo-likelihood estimator to avoid repeated solution of the fixed-point equation. With the same motivation in mind, we propose a constrained optimization approach for solving the maximum likelihood estimation of Rust's bus model. The idea behind our constrained optimization approach is simple: reformulate the log-likelihood function as a function of both EV and θ, jointly maximize the (reformulated) log-likelihood function in both EV and θ space, and impose structural equations, the Bellman fixed-point equation in this case, as constraints. Modern numerical constrained optimization

[11]Rust (1987) starts with contraction mapping (with a tolerance level of $1.0e-13$) and then switches to Newton's method to solve the fixed-point equation with very high accuracy; see also Rust (2000).

solvers do not require the constraints to be satisfied during the search process; hence, they minimize the need for repeatedly solving the fixed-point equation.

To be more specific, we construct an *augmented* likelihood function as a function of both structural parameters θ and expected value function EV:

$$\begin{aligned}
&\mathcal{L}(\theta, EV) \\
= &\sum_{i=1}^{M}\sum_{t=2}^{T}\log\left(P(d_t^i|x_t^i;\theta)\right) + \sum_{i=1}^{M}\sum_{t=2}^{T}\log\left(p_3(x_t^i|x_{t-1}^i, d_{t-1}^i;\theta_3)\right) \\
= &\sum_{i=1}^{M}\sum_{t=2}^{T}\log\left(\frac{\exp\{\nu(x_t^i, d_t^i, \theta) + \beta EV(x_t^i, d_t^i)\}}{\sum\limits_{d'\in\{0,1\}}\exp\{\nu(x_t^i, d', \theta) + \beta EV(x_t^i, d')\}}\right) \\
&+\sum_{i=1}^{M}\sum_{t=2}^{T}\log\left(p_3(x_t^i|x_{t-1}^i, d_{t-1}^i;\theta_3)\right).
\end{aligned} \tag{20}$$

Notice that in defining the augmented likelihood function, we have substituted the conditional-choice probability $P(d|x;\theta)$ in the log-likelihood function $L(\theta)$ in (18) by the multinomial formula given in (12). The augmented likelihood function $\mathcal{L}(\theta, EV)$ is a well-defined smooth function of θ and EV, and its first- and second-order derivatives with respect to θ and EV exist.

To ensure that EV and θ are consistent in the structural equation, we impose the integrated Bellman equation (16) as constraints. Combining the augmented likelihood function as the objective function and the integrated Bellman equation as constraints gives the following constrained optimization problem:

$$\begin{aligned}
\max_{(\theta, EV)} \quad & \frac{1}{M}\mathcal{L}(\theta, EV) \\
\text{subject to} \quad & EV = T(EV, \theta).
\end{aligned} \tag{21}$$

As we illustrate in Figure 1, the constraint Jacobian of this model is highly sparse. Let $dim(z)$ denote the dimension of a vector z. One can verify that the number of non-zero elements in the Jacobian is around $(dim(\theta) + dim(\theta_3) + 1) * dim(EV)$. The Hessian is even more sparse; the number of non-zero elements in the Hessian is around $(2 * (dim(RC) + dim(\theta_1)) + dim(\theta_3)) * (dim(\theta) + dim(EV))$. Assuming dim($\theta$) $\ll dim(EV)$, which is the case for most dynamic discrete choice models, the memory needed for the constrained optimization approach on Zucher's model is on the order of $dim(\theta) + dim(EV)$, the number of variables in the MPEC estimator.

4.3 Equivalence in Formulation between NFXP and MPEC

In this section, we show that the maximum likelihood estimation problem defined in (19) and the constrained optimization problem (21) are mathematically equivalent. We state the results in Proposition 2.

Proposition 2. Define the maximum likelihood estimator

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \frac{1}{M} L(\theta).$$

Let $\hat{EV}$ be the unique solution of the Bellman fixed-point equation (16) given the structural parameters $\hat{\theta}$, i.e.,

$$\hat{EV} = T(\hat{EV}, \hat{\theta}).$$

Denote (θ^*, EV^*) a solution of the constrained optimization problem (21). The following results hold:

(i) $L(\theta) = \mathcal{L}(\theta, EV)$, $\forall\, \theta$ and EV such that $EV = T(EV, \theta)$.

(ii) $L(\hat{\theta}) = \mathcal{L}(\theta^*, EV^*)$.

(iii) If the model is uniquely identified, then $\hat{\theta} = \theta^*$, and $\hat{EV} = EV^*$.

Proposition 2 states that NFXP and MPEC give the same estimates and are equivalent in formulation. Dubé, Fox and Su (2009) further prove the equivalence in solutions to the first-order conditions of NFXP and MPEC. They also discuss the finite-sample and large-sample properties of NFXP in the presence of inner-loop error.[12] NFXP and MPEC solves the same MLE problem. The only difference between NFXP and MPEC is how they solve the MLE problem.

[12]Dubé, Fox and Su (2009) study properties of NFXP in the context of demand estimation. However, the results and proof can be applied directly to the single-agent dynamic models considered here.

We now compare the numerical implementation of the NFXP and MPEC approach. First, if the solution time for solving the fixed-point equation is large relative to the time for evaluating the likelihood function, then the MPEC approach results in a faster algorithm. For each θ that is considered in NFXP, there are two steps: solving the dynamic programming problem with high accuracy and evaluating the likelihood function. Value function iteration will evaluate the Bellman equation many times in NFXP each time a dynamic program is solved. Furthermore, this is repeated for nearby values of θ if finite-difference methods are used to compute the derivatives. In contrast, for each θ considered in MPEC, the Bellman equation is *evaluated* once, not *solved.* Essentially, MPEC does not solve the Bellman equations exactly until the last few iterations.

4.4 A Monte Carlo Study

We consider a specific example of Rust's bus engine replacement model. We follow the specifications used to report the estimates in Table X in Rust (1987, p. 1022). In particular, we choose the fixed point dimension to be 175 (i.e., discretizing the mileage state space into 175 grid points) and assume the maintenance cost function is linear, that is, $c(x, \theta_1) = 0.001\theta_{11}x$. The Markov transition probabilities in the mileage state are defined as $\theta_{3j} = \Pr\{x_{t+1} = x_t + j|x_t, i_t = 0\}$, for $j = 0, 1, \ldots, 4$. Hence, θ_{3j} is the probability that the mileage state will move up j grid points in the next period. The unknown structural parameters to be estimated are replacement cost RC, cost function parameter θ_{11}, and mileage transition probabilities $\theta_3 = (\theta_{3j})_{j=0}^{4}$. We do not estimate the discount factor β. Instead, we run five scenarios with β ranging from 0.975 to 0.995 in an interval of 0.005.[13]

We use the parameter values reported for bus groups 1, 2, and 3 in Table X in Rust (1987) as the true values in the data generating process:

$$\begin{aligned} RC^0 &= 11.7257, \\ \theta_{11}^0 &= 2.4569, \\ \theta_3^0 &= (0.0937, 0.4475, 0.4459, 0.0127, 0.0002). \end{aligned}$$

We simulate time series data for $T = 120$ time periods (10 years of monthly data points) and $M = 50$ buses. To simulate data for each of the five scenarios and the corresponding β, we fix the structural parameters at $\theta^0 = (RC^0, \theta_{11}^0, \theta_3^0)$ and solve the integrated Bellman equation

[13]Rust (1997) chooses $\beta = 0.9999$.

(15) for EV^0. We then use θ^0 and EV^0 to compute the conditional choice probabilities $P(d|x,\theta)$ and to simulate mileage transitions and decisions for 250 data sets for each β in our Monte Carlo study.

We have conducted three implementations of algorithms. The first implementation is to code the MPEC estimator in AMPL, an algebraic modeling language; see Fourer, Gay, and Kernighan (2003).[14] The advantages of using AMPL are twofold: (*i*) it will compute the exact first-order and second-order analytic derivatives using automatic differentiation, and (*ii*) it will analyzes the sparsity pattern of constraint Jacobian and Hessian and provide that information to the solvers. We denote this implementation as MPEC/AMPL.[15]

We also code both MPEC and NFXP in Matlab and provide hand-coded first-order derivatives of objective function and constraint and the sparsity pattern of constraint Jacobian. We denote these two implementations as MPEC/Matlab and NFXP/Matlab, respectively. In our implementation of the NFXP, we use only contraction mapping iterations to solve the Bellman equation (15) with a inner-loop tolerance level $1.e-10$.[16] Rust (1997) estimates the transition probabilities θ_3 from the mileage transition data in the first stage and then estimates the remaining parameters (RC, θ_{11}) in the second stage. In our implementation of NFXP, we estimate the parameters $(RC, \theta_{11}, \theta_3)$ jointly by solving the full likelihood functions (18). For all three implementations, we solve the estimation problem using the optimization solver, Knitro, a state-of-the-art optimization program; see Byrd, Nocedal, and Waltz (2006). For each of the 1,250 replications (250 replications for each of the five β's) , we use five different starting values to do a better job at finding a global solution. To make a fair comparison, all three implementations use the same starting values in parameters.

We report the Monte Carlo results in Table 1. These three implementations produce almost identical means and standard deviations of estimates in the five scenarios when we vary the discount factor β. We believe the differences are due to different local maximizers found in these three implementations. The estimation results show that the ML estimator works extremely well on this model, in particular on recovering the transition probabilities. For $\beta = 0.99$ and 0.995, all mean parameter estimates, except for θ_{30}, are within two standard

[14]We can not implement NFXP in AMPL because it does not allow nested function calls.

[15]For this example, the number of non-zeros in Jacobian is 2255 ($\approx (7+5+1)*175)$)and the number of non-zeros in Hessian is 1584 ($\approx (2*(1+1)+5)*182)$).

[16]NFXP will need more computational time if one uses tighter inner loop tolerance level such as $1.e-13$, or follows Rust's suggestion on switching to Newton's method after the contraction mapping iteration converges; see Footnote 7. Since we provide hand-coded first-order derivatives in our implementation, NFXP using contraction mapping iteration only in the inner loop and with inner loop tolerance $1.e-10$ is quite robust.

errors of the true values.

We report the numerical performance of these three implementations in Table 2. We can see that the MPEC/Matlab implementation is most robust in terms of number of runs converged. In terms of timing, MPEC/AMPL is the fastest, needing around only 0.15 seconds for each run and considerably fewer iterations and function evaluations than do MPEC/Matlab and NFXP/Matlab. The reason is that AMPL supplies both first-order and second-order derivatives to the solver Knitro while the other two implementations supply only first-order derivatives to the Knitro solver. This observation confirms that a solver runs most efficiently and needs fewer iterations and function evaluations when supplying both first-order and second-order derivatives. If we remove the effect of supply second-order derivatives and compare the performance of MPEC/Matlab and NFXP/Matlab, MPEC still dominates NFXP in all categories. As one can see, the number of major iterations and function evaluations for NFXP and MPEC do not increase when we increase the discount factor from 0.975 to 0.995; however, the computing time for NFXP increases steadily while the computing time for MPEC/Matlab (or MPEC/AMPL) stays the same for different β's. The reason for the increase in computing time for NFXP is that when the discount factor β increases, the number of contraction mapping iterations needed in solving the inner loop Bellman equations also increases.

4.5 Boostrap Inference

Instead of using the asymptotic properties of an estimator presented in Section 3.2 to construct confidence inference, one can use the bootstrap to construct confidence interval or test hypothesis. Furthermore, it has been shown that the bootstrap inference is often more accurate than the inference based on asymptotic theory when the sample size is small; see Horowitz (2001) and MacKinnon (2002). However, the feasibility of conducting bootstrap inference relies on having a computationally efficient method to perform parameter estimation of the underlying structural model since the bootstrap requires estimating parameters for each of the simulated samples. As we illustrate above, the MPEC approach provides a computational efficient method to estimate dynamic discrete-choice models and make the bootstrap inference possible. Indeed, the data sampling procedure that we describe in the Monte Carlo study in Section 4.4 is a parametric bootstrap procedure to generate simulated samples and the Monte Carlo study demonstrate the uses of parametric bootstrap to compute standard errors on structural parameters.

Table 1: Monte Carlo Results Varying the Discount Factor β

β	Implementation	Parameters							MSE
			RC	θ_{11}	θ_{30}	θ_{31}	θ_{32}	θ_{33}	
		True values	**11.726**	**2.457**	**0.0937**	**0.4475**	**0.4459**	**0.0127**	
0.975	MPEC/AMPL	Mean	12.212	2.607	0.0943	0.4473	0.4454	0.0127	3.111
		Std. dev.	(1.613)	(0.500)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
	MPEC/Matlab	Mean	12.212	2.607	0.0943	0.4473	0.4454	0.0127	3.111
		Std. dev.	(1.613)	(0.500)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
	NFXP/Matlab	Mean	12.213	2.606	0.0943	0.4473	0.4445	0.0127	3.123
		Std. dev.	(1.617)	(0.500)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
0.980	MPEC/AMPL	Mean	12.134	2.578	0.0943	0.4473	0.4455	0.0127	2.857
		Std. dev.	(1.570)	(0.458)	(0.0037)	(0.0057)	(0.0060)	(0.0015)	–
	MPEC/Matlab	Mean	12.134	2.578	0.0943	0.4473	0.4455	0.0127	2.857
		Std. dev.	(1.570)	(0.458)	(0.0037)	(0.0057)	(0.0060)	(0.0015)	–
	NFXP/Matlab	Mean	12.139	2.579	0.0943	0.4473	0.4455	0.0127	2.866
		Std. dev.	(1.571)	(0.459)	(0.0037)	(0.0057)	(0.0060)	(0.0015)	–
0.985	MPEC/AMPL	Mean	12.013	2.541	0.0943	0.4473	0.4455	0.0127	2.140
		Std. dev.	(1.371)	(0.413)	(0.0037)	(0.0057)	(0.0060)	(0.0015)	–
	MPEC/Matlab	Mean	12.013	2.541	0.0943	0.4473	0.4455	0.0127	2.140
		Std. dev.	(1.371)	(0.413)	(0.0037)	(0.0057)	(0.0060)	(0.0015)	–
	NFXP/Matlab	Mean	12.021	2.544	0.0943	0.4473	0.4455	0.0127	2.136
		Std. dev.	(1.368)	(0.411)	(0.0037)	(0.0057)	(0.0060)	(0.0015)	–
0.990	MPEC/AMPL	Mean	11.830	2.486	0.0943	0.4473	0.4455	0.0127	1.880
		Std. dev.	(1.305)	(0.407)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
	MPEC/Matlab	Mean	11.830	2.486	0.0943	0.4473	0.4455	0.0127	1.880
		Std. dev.	(1.305)	(0.407)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
	NFXP/Matlab	Mean	11.830	2.486	0.0943	0.4473	0.4455	0.0127	1.880
		Std. dev.	(1.305)	(0.407)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
0.995	MPEC/AMPL	Mean	11.819	2.492	0.0942	0.4473	0.4455	0.0127	1.892
		Std. dev.	(1.308)	(0.414)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
	MPEC/Matlab	Mean	11.819	2.492	0.0942	0.4473	0.4455	0.0127	1.892
		Std. dev.	(1.308)	(0.414)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–
	NFXP/Matlab	Mean	11.819	2.492	0.0942	0.4473	0.4455	0.0127	1.892
		Std. dev.	(1.308)	(0.414)	(0.0036)	(0.0057)	(0.0060)	(0.0015)	–

There are 250 replications for each experiment. Each replication uses five starting points to do a better job at finding a global maximum. MSE is calculated by summing over all structural parameters.

Table 2: Numerical Performance of Estimators in Monte Carlo Experiments

β	Implementation	Runs Converged (out of 1250 runs)	CPU Time (in sec.)	# of Major Iter.	# of Func. Eval.	# of Contraction Mapping Iter.
0.975	MPEC/AMPL	1240	0.13	12.8	17.6	–
	MPEC/Matlab	1247	7.9	53.0	62.0	–
	NFXP	998	24.6	55.9	189.4	$1.348e+5$
0.980	MPEC/AMPL	1236	0.15	14.5	21.8	–
	MPEC/Matlab	1241	8.1	57.4	70.6	–
	NFXP	1000	27.9	55.0	183.8	$1.625e+5$
0.985	MPEC/AMPL	1235	0.13	13.2	19.7	–
	MPEC/Matlab	1250	7.5	55.0	62.3	–
	NFXP	952	42.2	61.7	227.3	$2.658e+5$
0.990	MPEC/AMPL	1161	0.19	18.3	42.2	–
	MPEC/Matlab	1248	7.5	56.5	65.8	–
	NFXP	935	70.1	66.9	253.8	$4.524e+5$
0.995	MPEC/AMPL	965	0.14	13.4	21.3	–
	MPEC/Matlab	1246	7.9	59.6	70.7	–
	NFXP	950	111.6	58.8	214.7	$7.485e+5$

For each experiment, we solve the model once for the expected value function by fixing structural parameters at true values and then simulate and re-estimate the model 250 times. Each of the 250 replications uses five starting points to do a better job at finding a global maximum. CPU time, number of major iterations, number of function evaluations and number of contraction mapping iterations are the averages for each run.

5 Estimation of Games of Incomplete Information

Unlike the single-agent dynamic discrete-choice models considered in Section 4, in which a unique fixed point for the integrated Bellman equation there always exists, games can admit multiple equilibria. This distinct feature requires applying NFXP to estimating games computationally difficult because one needs to find all the equilibria for any given structural parameters in the inner loop of NFXP. Furthermore, the resulting likelihood function can be discontinuous, which makes the outer loop optimization problem extremely difficult.[17] In this section, we use an example of a discrete-choice game of incomplete information to demonstrate multiple equilibria, the discontinuity in the likelihood function when using NFXP, and the use of MPEC approach for estimating games. We also conduct Monte Carlo experiments to compare the finite sample properties of MPEC and those of two-step pseudo maximum likelihood (2S-PML), two-step least squares (2S-LS), and nested pseudo-likelihood.

[17]If one can prove a priori that the likelihood function is continuous in structural parameters, then NFXP is a valid approach for estimating games.

5.1 A Discrete-Choice Game of Incomplete Information

The description of the example presented here closely follows the derivation in Rust (2008).[18] For simplicity, we first describe the model for only one market and then generalize the model to accommodate data observed from multiple differentiated markets.

5.1.1 Model with One Market

Consider a two-player discrete-choice game with both *observed* and *unobserved* heterogeneity. Let the players be labeled a and b, and let d_a and d_b denote the choice of player a and b, respectively. For simplicity, we assume that each player has two possible choices; see, for example, entry and exit games in Berry (1992), Seim (2006), and Ciliberto and Tamer (2009). We define

$$d_a = \begin{cases} 1 & \text{if player } a \text{ is active} \\ 0 & \text{if player } a \text{ is not active;} \end{cases}$$

d_b is defined similarly.

The ex post utility functions for players a and b are

$$\begin{aligned} u_a(d_a, d_b, x_a, \varepsilon_a) &= \theta^a_{d_a,d_b} x_a + \varepsilon_a(d_a), \\ u_b(d_a, d_b, x_b, \varepsilon_b) &= \theta^b_{d_a,d_b} x_b + \varepsilon_b(d_b), \end{aligned}$$

where a scalar x_a is the observed type for player a and a (2×1) vector ε_a is player a's unobserved type. The parameter (to be estimated) $\theta^a_{d_a,d_b}$ measures the effect of the observed type x_a on player a's utility. Note that player a's utility is a function of joint decision of both player players, (d_a, d_b), and this is captured by the parameter $\theta^a_{d_a,d_b}$. The observed type x_b, unobserved type ε_b, and parameter $\theta^b_{d_a,d_b}$ are similarly defined for player b.

We assume that the observed types (x_a, x_b) are common knowledge among players and that unobserved types $(\varepsilon_a, \varepsilon_b)$ are private information. Furthermore, we assume that player a knows the distribution of ε_b, and similarly player b knows the distribution of ε_a. We also assume that ε_a and ε_b are independent. Let $\theta^a = \{\theta^a_{d_a,d_b}\}$ and $\theta^b = \{\theta^b_{d_a,d_b}\}$ denote the parameters to be estimated for player a and player b, respectively. We assume θ^a and θ^b are common knowledge among players.

[18]We are grateful to John Rust for providing us his lecture notes with a detailed derivation and discussion of this example.

Because of the presence of private information ε_a, player a's decision will be probabilistic from player b's point of view. Let p_a denote player b's belief of the probability that player a will be active. Similarly, p_b represents player a's belief of the probability that player b will be active. Given player b's belief p_a, the expected utility of player b for taking an action d_b is given by

$$\begin{aligned} U_b(d_b, x_b, \varepsilon_b) &= p_a u_b(1, d_b, x_b, \varepsilon_b) + (1-p_a) u_b(0, d_b, x_b, \varepsilon_b) \\ &= p_a \theta^b_{1,d_b} x_b + (1-p_a)\theta^b_{0,d_b} + \varepsilon_b(d_b). \end{aligned}$$

It follows that player b will be active ($d_b = 1$) if and only if

$$U_b(1, x_b, \varepsilon_b(1)) > U_b(0, x_b, \varepsilon_b(0)).$$

We assume that each component in the error terms ε_a and ε_b has an extreme value distribution. Given player b's belief p_a, the probability that player b will be active is given by the standard binomial logit formula

$$\begin{aligned} p_b &= Prob\{d_b = 1\} \\ &= Prob\left\{\varepsilon_b | U_b(d_b = 1, x_b, \varepsilon_b(1)) > U_b(d_b = 0, x_b, \varepsilon_b(0))\right\} \\ &= \frac{1}{1 + \exp\{(x_b(\theta^b_{0,0} - \theta^b_{0,1}) + p_a x_b(\theta^b_{0,1} - \theta^b_{0,0} + \theta^b_{1,0} - \theta^b_{1,1}))\}} \\ &\equiv \Psi_b(p_a, p_b, x_b; \theta^b). \end{aligned} \tag{22}$$

This formula can be thought of as a *best response function* for player b given b's belief p_a. Similarly, the best response for player a, given a's belief p_b, is

$$\begin{aligned} p_a &= \frac{1}{1 + \exp\{(x_a(\theta^a_{0,0} - \theta^a_{1,0}) + p_b x_a(\theta^a_{0,1} - \theta^a_{0,0} + \theta^a_{1,0} - \theta^a_{1,1}))\}} \\ &\equiv \Psi_a(p_a, p_b, x_a; \theta^a). \end{aligned} \tag{23}$$

A *Bayesian Nash equilibrium* is a pair of beliefs (p_a^*, p_b^*) that are mutual best responses:

$$\begin{aligned} p_a^* &= \Psi_a(p_a^*, p_b^*, x_a; \theta^a) \\ p_b^* &= \Psi_b(p_a^*, p_b^*, x_b; \theta^b). \end{aligned} \tag{24}$$

The structural parameters to be estimated in this model are $(\theta^a, \theta^b) = (\{\theta^a_{d_a,d_b}\}, \{\theta^b_{d_a,d_b}\})$. However, we cannot identify all of them. Rust (2008) shows that we can identify only four parameters:

$$\alpha^a = \theta^a_{0,0} - \theta^a_{1,0}, \qquad \alpha^b = \theta^b_{0,0} - \theta^b_{0,1},$$

$$\beta^a = \theta^a_{0,1} - \theta^a_{1,1}, \qquad \beta^b = \theta^b_{1,0} - \theta^b_{1,1}.$$

We further impose the symmetry assumptions on the players; that is, $\theta^a = \theta^b$ and hence, $\alpha^a = \alpha^b = \alpha$ and $\beta^a = \beta^b = \beta$. Substituting α and β into the best response functions (22) and (23), we have

$$\begin{aligned} p_a &= \frac{1}{1 + \exp\{(x_a\alpha + p_b x_a(\beta - \alpha)\}}, \\ p_b &= \frac{1}{1 + \exp\{(x_b\alpha + p_a x_b(\beta - \alpha)\}}. \end{aligned} \tag{25}$$

To simplify the notation, let $\boldsymbol{\theta} = (\alpha, \beta)$, $x = (x_a, x_b)$, $p = (p_a, p_b)$ and $\boldsymbol{\Psi} = (\Psi_a, \Psi_b)$. We rewrite the Bayesian Nash equilibrium equations (24) as

$$p = \boldsymbol{\Psi}(p, x; \boldsymbol{\theta}). \tag{26}$$

For given parameter values (α, β) and observed types (x_a, x_b), there are two equations and two unknowns, (p_a, p_b) in (25). However, there could exist multiple Bayesian Nash equilibria satisfying the equations (25). The example below illustrates such a case.

Example 1. We select parameter values $\boldsymbol{\theta}^0 = (\alpha^0, \beta^0) = (-5, 11)$ and players' observed types $x = (x_a, x_b) = (0.52, 0.22)$. Substituting these values into (25), we solve the following two equations for Bayesian Nash equilibria p^*:

$$\begin{aligned} p_a &= \frac{1}{1 + \exp(-4.48 + 8.32 p_b)}, \\ p_b &= \frac{1}{1 + \exp(-1.10 + 3.52 p_a)}. \end{aligned} \tag{27}$$

We use the state-of-the-art constrained optimization solver, Knitro, to solve this system of two equations and two unknowns. Since there could be multiple solutions, we use 100 different starting points in an attempt to find all equilibria. It turns out that there are three Bayesian Nash equilibria that solve the two equations above:

$$\begin{aligned} &\text{Equilibrium 1:} \quad (p_a, p_b) = (0.030100, 0.729886), \\ &\text{Equilibrium 2:} \quad (p_a, p_b) = (0.616162, 0.255615), \\ &\text{Equilibrium 3:} \quad (p_a, p_b) = (0.773758, 0.164705). \end{aligned}$$

Furthermore, one can verify that Equilibrium 2 is not stable under best-reply (BR) iteration, which means that the BR iteration will not be able to find Equilibrium 2 even if the starting point for BR is very close to the solution.[19]

5.1.2 Model with Multiple Markets

We generalize the model described above to accommodate multiple differentiated markets in the observed data. Assume that there are M markets and that players' decisions in one market do not depend on their decisions in other markets. However, the unknown parameters to be estimated $\boldsymbol{\theta} = (\alpha, \beta)$ stay the same across all markets. In our setting, the characteristics that differentiate markets are the players' observed types. Hence, two different players' observed types $(x_a, x_b) \neq (\hat{x}_a, \hat{x}_b)$ give rise to two different markets. We denote the players' observed types in market m as $x^m = (x_a^m, x_b^m)$, for $m = 1, \dots, M$.

For each market, there is a set of Bayesian Nash equilibrium equations, parameterized by the observed types in that market, that define the corresponding Bayesian Nash equilibrium. Denoted as p^m, the Bayesian Nash equilibrium for market m satisfies the following equation:

$$p^m = \boldsymbol{\Psi}(p^m, x^m; \boldsymbol{\theta}), \qquad \text{for } m = 1, \dots, M. \tag{28}$$

Let $P = (p^m)_{m=1}^M$ be the collection of equilibrium probabilities for all markets . We denote $\mathbf{x} = (x^m)_{m=1}^M$ the collection of the players' observed types for all markets. With a slight abuse of notation, we simplify the Bayesian Nash equilibrium equations for all markets (28) as

$$P = \boldsymbol{\Psi}(P, \mathbf{x}; \boldsymbol{\theta}). \tag{29}$$

From (28), it is clear that markets with different observed types might have different Bayesian Nash equilibria. Furthermore, the number of equilibria can be different for markets with different observed types, as we illustrate in the following example.

Example 2. We select the same parameter values $\boldsymbol{\theta}^0 = (\alpha^0, \beta^0) = (-5, 11)$ as in Example 1. We consider a case with 256 markets. We discretize the interval $[0.12, 0.87]$ into 16 equally spaced grid points. These 16 different values give 16 observed types for each player; jointly,

[19]Note that best-reply stable equilibrium is not among the common notions of stable equilibria, such as strategic stability in Kohlberg and Mertens (1986), and Mertens (1989, 1991), or evolutionarily stable strategy (ESS), studied in the game theory literature. See also Chapter 11 in Fudenberg and Tirole (1996).

there are 16*16 = 256 pairs of $(x_a^m, x_b^m), m = 1, \ldots, 256$. Each of these pairs defines a market. For each market m, we solve the corresponding Bayesian Nash equilibrium equations in (28) with 100 different starting values to find all Bayesian Nash equilibria $p^{m*} = (p_a^{m*}, p_b^{m*})$ for that market.

We do not report all equilibrium solutions. Instead, we report the number of equilibria in each market in Figure 2. As one can see, there are three equilibria in most markets. For markets with low x_a and/or x_b, the equilibrium is unique. However, the number of equilibria can be different even for markets with similar characteristics. For example, there are three equilibria for the market with observed types $(0.17, 0.87)$, but there is only one equilibrium for the market with the observed types $(0.12, 0.87)$.

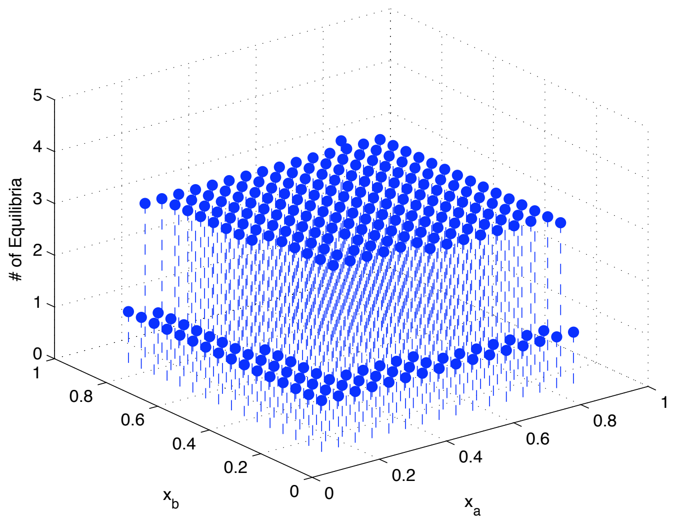


Figure 2: Number of equilibria in different markets

5.2 Estimation

In this section, we describe data generating process and discuss various estimators for estimating this discrete-choice game of incomplete information.

5.2.1 Data Generating Process

In the data, the researcher observes the players observed types $x^m = (x_a^m, x_b^m)$ and the players' decisions $d^{mt} = (d_a^{mt}, d_b^{mt})_{t=1}^T$ over T periods for the game played independently in that market. Let X^m denote the data observed in market m:

$$X^m = \{x^m = (x_a^m, x_b^m), d^{mt} = (d_a^{mt}, d_b^{mt}), \text{ for } t = 1, \ldots, T\}.$$

Data observed for all M markets over T periods are the collection of X^m for all m:

$$X = \{X^m, \text{ for } m = 1, \ldots, M\}. \tag{30}$$

Following a common assumption in the literature (Aguirregabiria and Mira 2007, Bajari, Benkard, and Levin 2007, and Pesendorfer and Schmidt-Dengler 2008), we assume that for each market, only one equilibrium is played in the data. Since equilibrium solutions are different in different markets, as we see in Example 2, in the data we allow different equilibria being played in different markets.

Assumption 2: For each market, only one equilibrium is played in the data. However, equilibria played across different markets are different.

5.2.2 Maximum Likelihood Estimation with NFXP

Given Assumption 2, the log-likelihood of observing data X^m in market m is

$$\begin{aligned} & L_i(X^m, \boldsymbol{p}^m(\boldsymbol{\theta}); \boldsymbol{\theta}) \\ = & \sum_{t=1}^{T} \left(d_a^{mt} * \log(\boldsymbol{p}_a(x_a^m, x_b^m, \boldsymbol{\theta})) + (1 - d_a^{mt}) * \log(1 - \boldsymbol{p}_a(x_a^m, x_b^m; \boldsymbol{\theta})) \right) \\ + & \sum_{t=1}^{T} \left(d_b^{mt} * \log(\boldsymbol{p}_b(x_a^m, x_b^m, \boldsymbol{\theta})) + (1 - d_b^{mt}) * \log(1 - \boldsymbol{p}_b(x_a^m, x_b^m; \boldsymbol{\theta})) \right), \end{aligned} \tag{31}$$

where $\boldsymbol{p}^m(\boldsymbol{\theta}) = (\boldsymbol{p}_a(x_a^m, x_b^m, \boldsymbol{\theta}), \boldsymbol{p}_b(x_a^m, x_b^m, \boldsymbol{\theta}))$ denote a Bayesian Nash equilibrium given $\boldsymbol{\theta}$ and (x_a^m, x_b^m). Let $\boldsymbol{P}(\boldsymbol{\theta}) = (\boldsymbol{p}(\boldsymbol{\theta})^m)_{m=1}^M$ and let $\Sigma(\boldsymbol{\theta})$ be the set of all Bayesian Nash equilibria. The log likelihood of observing the full data X for all markets is

$$L(\boldsymbol{\theta}, \boldsymbol{P}(\boldsymbol{\theta})) = \sum_{m=1}^{M} L_i(X^m, \boldsymbol{p}^m(\boldsymbol{\theta}); \boldsymbol{\theta}). \tag{32}$$

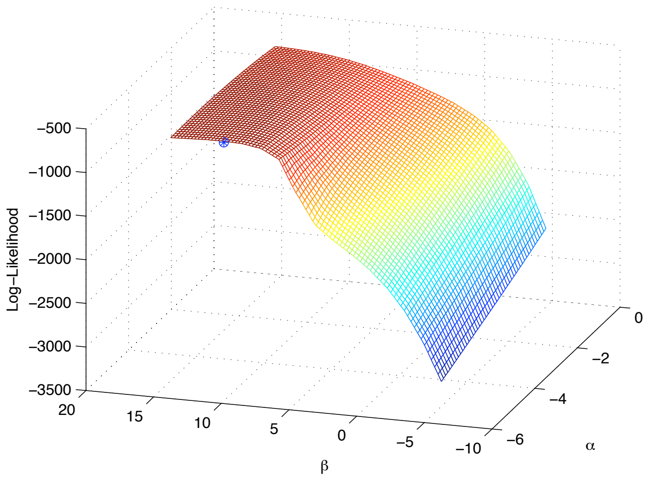
Since there can be multiple equilibria, the maximum likelihood estimator is defined as

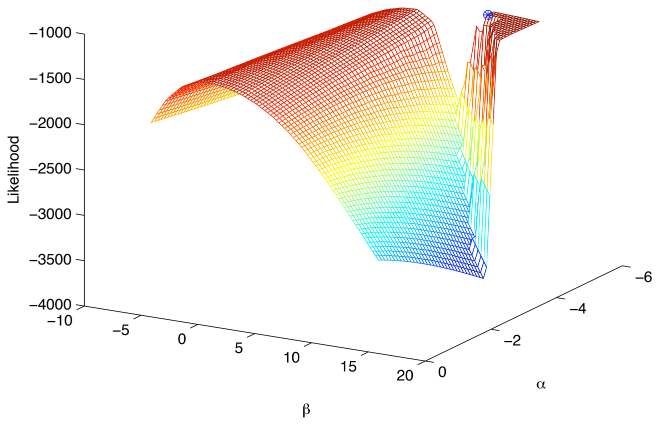
$$\boldsymbol{\theta}^{MLE} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{1}{M} \left\{ \max_{\boldsymbol{P}(\boldsymbol{\theta}) \in \Sigma(\boldsymbol{\theta})} L(\boldsymbol{\theta}, \boldsymbol{P}(\boldsymbol{\theta})) \right\} \tag{33}$$

The procedure for applying NFXP to find $\boldsymbol{\theta}^{MLE}$ is described as follows. In the outer loop, one searches over the structural parameter space $\boldsymbol{\theta}$ to maximize the log likelihood function (32). In the outer loop, for a given $\boldsymbol{\theta}$, one needs to find all the Bayesian Nash equilibria (BNE), evaluate the likelihood value at each such BNE and pick the BNE that gives the highest likelihood value. The algorithm then returns to the outer loop and repeats until it converges or fails.

Two main computational difficulties when applying NFXP to solve the ML estimator (33) are that (i) one needs to find all the equilibria for any given structural parameters $\boldsymbol{\theta}$, and (ii) the number of equilibria can be different from different values of structural parameters $\boldsymbol{\theta}$, which gives rise to the the discontinuity in the likelihood function. In the example below, we illustrate a case of discontinuous likelihood function using NFXP.

Example 3. Consider the setting in Example 1, in which there are three Bayesian Nash equilibria. First, we assume that both players use Equilibrium 1 to play the game 1,000 times and hence randomly generate 1,000 pairs of actions $(d_{a1}^t, d_{b1}^t)_{t=1}^{1000}$ independently based on Equilibrium 1. Denote this dataset as X_1. Repeat the same procedure. Equilibrium 2 and Equilibrium 3 are played 1,000 times to generate dataset X_2 and X_3, respectively. We plot the log-likelihood functions in the parameter $\boldsymbol{\theta} = (\alpha, \beta)$ space for each of the three datasets in Figure 3. As one can see, the log-likelihood function for X_1 is continuous in (α, β), while the log-likelihood function for X_2 and X_3 are discontinuous. Recall that Equilibrium 2 is unstable under best-reply iteration, whereas Equilibrium 3 is stable under best-reply. This example shows that the discontinuity in likelihood function does not depend on best-reply stability of an equilibrium.





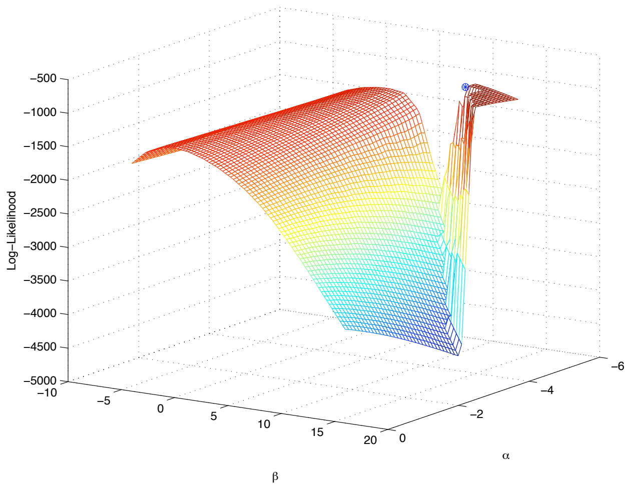


Figure 3: Log-likelihood function of NFXP when Equilibrium 1 (top figure), Equilibrium 2 (middle figure), and Equilibrium 3 (bottom figure) are played, respectively, in the data. The blue dot indicates the maximizer in the likelihood function.

5.2.3 Maximum Likelihood Estimation with MPEC

We now describe the MPEC approach for estimating games of incomplete information. The main advantage of using the MPEC approach in this setting is that the objective function and the constraints are smooth in our formulation, which gives rise to a smooth constrained optimization problem. As a result, one can use a state-of-the-art constrained optimization solver to solve the estimation problem.

Recall that $p^m = (p_a^m, p_b^m) \in R^2$ denotes any equilibrium probabilities in market m and that $P = (p^m)_{m=1^M}$. Given the observed data X_m, we define the augmented log-likelihood for observing X_m in market m as

$$\begin{aligned} \mathcal{L}_i(X^m, p^m, \boldsymbol{\theta}) &= \sum_{t=1}^{T} \left(d_a^{mt} * \log(p_a^m) + (1 - d_a^{mt}) * \log(1 - p_a^m) \right) \\ &+ \sum_{t=1}^{T} \left(d_b^{mt} * \log(p_b^m) + (1 - d_b^{mt}) * \log(1 - p_b^m) \right). \end{aligned} \tag{34}$$

The augmented log-likelihood of observing the full data X for all markets is

$$\mathcal{L}(\boldsymbol{\theta}, P) = \sum_{m=1}^{M} \mathcal{L}_i(X^m, p^m, \boldsymbol{\theta}). \tag{35}$$

The decision variables in our formulation are $\boldsymbol{\theta}$ and P. Notice from (34) that the structural parameters $\boldsymbol{\theta}$ does not enter the augmented function (35). Furthermore, the augmented likelihood function is a smooth function of equilibrium probabilities P and is well defined even when some component p^m in P is not a Bayesian Nash equilibrium.

To ensure that for all markets, the equilibrium probabilities p^m are Bayesian Nash equilibria, we impose the Bayesian Nash equilibrium equations (28) as constraints. The equilibrium probabilities p^m must be between 0 and 1. As a result, the constrained optimization formulation for the MPEC approach is given as follows:

$$\begin{aligned} \max_{(\theta, P)} \quad & \frac{1}{M} \mathcal{L}(\boldsymbol{\theta}, P) \\ \text{subject to} \quad & P = \boldsymbol{\Psi}(P, \mathbf{x}, \boldsymbol{\theta}). \end{aligned} \tag{36}$$

Similar to equivalence result stated in Proposition 2, one can show the equivalence of the MLE formulated in (33) and (36). We state this results below.

Proposition 3. Let $\bar{\boldsymbol{\theta}}$ be the solution of the maximum likelihood estimator defined in (33). Denote $\bar{\boldsymbol{P}}(\boldsymbol{\theta}) = \underset{\boldsymbol{P}(\boldsymbol{\theta})\Sigma(\boldsymbol{\theta})}{\operatorname{argmax}}\, L(\boldsymbol{\theta}, \boldsymbol{P}(\boldsymbol{\theta}))$. Let $(\boldsymbol{\theta}^*, P^*)$ a solution of the constrained optimization problem (36). The following results hold:

(i) $L(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{P}}(\bar{\boldsymbol{\theta}})) = \mathcal{L}(\boldsymbol{\theta}^*, P^*)$.

(ii) If the model is uniquely identified, then $\bar{\boldsymbol{\theta}} = \boldsymbol{\theta}^*$.

5.2.4 Two-Step Estimators

Pioneered by Hotz and Miller (1993) for estimating single-agent dynamic discrete-choice models, two-step estimators have become extremely popular among applied researchers and practitioners. One attraction of two-step estimators is that they are thought of as computationally light, compared with Rust's NFXP method or the MPEC approach. In this section, we describe two-step pseudo likelihood (2S-PML) estimator and two-step least squares estimator (2S-LS) for estimating the discrete-choice games described Section 5.1.

For any given vector of probabilities $\hat{P} = (\hat{p}^m)_{m=1}^M$, the pseudo likelihood of observing data X^m in market m is defined as

$$\begin{aligned}
& L_i^{PML}(X^m, \hat{p}^m; \boldsymbol{\theta}) \\
= & \sum_{t=1}^{T} \left(d_a^{mt} * \log(\Psi_a(\hat{p}^m, x^m, \theta)) + (1 - d_a^{mt}) * \log(1 - \Psi_a(\hat{p}^m, x^m, \theta)) \right) \\
+ & \sum_{t=1}^{T} \left(d_b^{mt} * \log(\Psi_b(\hat{p}^m, x^m, \theta)) + 1 - d_b^{mt}) * \log(1 - \Psi_b(\hat{p}^m, x^m, \theta)) \right),
\end{aligned}$$

The pseudo likelihood of observing the full data X for all markets is

$$L^{PML}(\boldsymbol{\theta}, \hat{P}) = \sum_{m=1}^{M} L_i(X^m, \hat{p}^m; \boldsymbol{\theta}). \tag{37}$$

Notice that in defining the pseudo likelihood function (37), the vector of probabilities

$\hat{P}$ does not need to be a Bayesian-Nash equilibrium. The pesudo likelihood function is well defined for any $\hat{P}$.

The two-step pseudo maximum likelihood (2S-PML) estimator is as follows:

$$\begin{aligned}
&\text{Step 1:} \quad \text{Find } \hat{P}_0\text{, a consistent estimate of the true choice probabilities } P^0. \\
&\text{Step 2:} \quad \text{Fixed } \hat{P}_0\text{. Solve the pseudo maximum likelihood estimator:} \\
&\qquad\qquad \boldsymbol{\theta}^{2S-PML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{1}{M} L^{PML}(\boldsymbol{\theta}, \hat{P}_0).
\end{aligned} \tag{38}$$

One can also use a two-step least squares (2S-LS) estimator (see Pesendorfer and Schimdt-Dengler (2008)). Instead of maximizing the pseudo likelihood function, one chooses structural parameters $\boldsymbol{\theta}$ to minimize the $l-2$ norm of the errors in the Bayesian Nash equilibrium equations in 2S-LS:[20]

$$\begin{aligned}
&\text{Step 1:} \quad \text{Find } \hat{P}_0\text{, a consistent estimate of the true choice probabilities } P^0. \\
&\text{Step 2:} \quad \text{Fixed } \hat{P}_0\text{. Solve the least square problem:} \\
&\qquad\qquad \boldsymbol{\theta}^{2S-LS} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\hat{P}_0 - \boldsymbol{\Psi}(\hat{P}_0, \mathbf{x}, \boldsymbol{\theta})\|_2^2.
\end{aligned} \tag{39}$$

Notice that in the two-step estimators, the Bayesian Nash equilibrium equations are not enforced in the second step and hence may not be satisfied at a solution. Furthermore, the optimization problem in the second step in unconstrained and only involves structural parameters. Because two-step estimators do not solve the Bayesian Nash equilibrium equations, they are considered computationally light. However, the trade-off is that they can be severely biased in finite-sample performance. Also, two-step estimators are asymptotically inefficient. We refer the readers to Aguirregabiria and Mira (2002, 2007) for further discussion.

5.2.5 Nested Pseudo Likelihood Estimator

Recognizing the limitations of two-step estimators, Aguirregabiria and Mira (2002, 2007) propose the nested pseudo likelihood (NPL) estimator for estimating dynamic discrete-choice

[20] Pesendorfer and Schimdt-Dengler (2008) proposes an asymptotic least-squares estimator and derive the asymptotically optimal weight matrix. Here, we do not derive the optimal weight matrix and use the identity matrix as the weight matrix in this example.

models and dynamic discrete games. NPL is a recursive procedure over two-step PML. The purpose of recursively iterating over 2S-PML is to enforce the Bayesian Nash equilibrium equations to be satisfied, and consequently reduce the finite-sample biases of two-step estimators. An outline of iterative procedure for the NPL estimator is given below.

```
**Outline of Nested Pseudo Likelihood (NPL) Algorithm:**
**Initialization:** Find an initial guess of choice probabilities $\hat{P}_0$. Set $K = 0$.
**repeat**
  Increase $K \leftarrow K + 1$.
  Fixed $\hat{P}_{K-1}$. Solve the pseudo maximum likelihood estimator for $\hat{\boldsymbol{\theta}}_K$:
      $\hat{\boldsymbol{\theta}}_K = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \frac{1}{M} L^{PML}(\boldsymbol{\theta}, \hat{P}_{K-1})$.
  Given $\hat{\boldsymbol{\theta}}_K$, obtain $\hat{P}_K$ by one best-reply iteration on the Bayesian Nash equilibrium
  equation (29):
      $\hat{P}_K = \boldsymbol{\Psi}(\hat{P}_{K-1}, \mathbf{x}, \hat{\boldsymbol{\theta}}_K)$.
**until** Converge
Let $\boldsymbol{\theta}^{NPL} = \hat{\boldsymbol{\theta}}_K$ and $P^{NPL} = \hat{P}_K$.
```

When NPL converges, the Bayesian Nash equilibrium equations are satisfied at the NPL solution $(\boldsymbol{\theta}^{NPL}, P^{NPL})$. However, there are serious drawbacks with NPL. First, there is no guarantee that NPL will converge, in particular for estimating games. Second, since NPL relies on best-reply iteration, it only finds equilibria that are best-reply stable. If the data were generated by equilibria that are not stable under best-reply iteration, NPL may fail to converge or converge to incorrect parameter values. See Pesendorfer and Schmidt-Dengler (2010) for such an example.

One can show that under certain regularity conditions, the maximum likelihood estimator for discrete-choice games of incomplete information is consistent, asymptotically normal and efficient. Aguirregabiria and Mira (2007) show that asymptotically NPL is more efficient than the two-step PML estimator. Hence, we conclude that maximum likelihood estimator with MPEC is more efficient than NPL and two-step PML.

5.3 Monte Carlo

We conduct several Monte Carlo experiments to study finite-sample performance of four estimators: two-step pseudo maximum likelihood (2S-PML), two-step least squares (2S-LS), nested pseudo likelihood (NPL), and our maximum likelihood/MPEC estimator. We use Example 2 as the econometric model in our Monte Carlo experiments. Recall that the true values of parameters are $\boldsymbol{\theta}^0 = (\alpha^0, \beta^0) = (-5, 11)$ and there are $M = 256$ differentiated markets. We use different types of equilibria to generate the players' decisions in each market. Note that we still maintain the assumption that in each market, only one equilibrium is played in the data; however the equilibrium played in different can be different.

We describe the properties of equilibrium used in generating data in each of the three experiments below.

> Experiment 1: Only the best-reply stable equilibrium with lowest probabilities of being active for player a is played in each market for markets with multiple equilibria.[21]
>
> Experiment 2: Only one of the best-reply stable equilibria is played in each market for markets with multiple equilibria.
>
> Experiment 3: Only one equilibrium is played in each market. The equilibria chosen to be played can be stable or nonstable under best-reply iteration.

For Experiments 1 and 2, we generate the players' actions using a best-reply stable equilibrium in each market. For Experiment 3, we randomly pick an equilibrium in each market to simulate players' decisions; these equilibria may or may not be stable under best-reply iteration.

For each experiment, we simulate the static game for $T = 250$ periods and hence generate 250 pairs of actions, $p^m = (p_a^{mt}, p_b^{mt})_{t=1}^{250}$ in each market m, m = 1, ..., 256. We conduct 100 replications in each experiment to obtain a distribution of parameter estimates.

Since the players' decisions are observed in every market in the static game model, we use a frequency estimator to estimate the first-step choice probabilities for two-step estimators. The estimated choice probabilities $\hat{p}^m = (\hat{p}_a^m, \hat{p}_b^m)$ for each market m by the

[21]Rust (2008) uses the equilibrium that results in the highest probability of confession for player a in the Monte Carlo study.

frequency estimator is given as

$$\hat{p}_a^m = \frac{1}{T}\sum_{t=1}^{T} p_a^{mt}, \quad \hat{p}_b^m = \frac{1}{T}\sum_{t=1}^{T} p_b^{mt}, \text{ for } m = 1, \ldots, 256.$$

We also use the same estimated choice probabilities from the frequency estimator to initiate the NPL procedure. We set the maximum number of NPL iterations to be 500. If the number of NPL iterations reaches 500, we declare that NPL fails to converge in that run.

For MPEC, with 256 markets and two choice probabilities (p_a^m, p_b^m) in each market, there are $2 + 2 * 256 = 514$ decision variables and 512 equality constraints (2 Bayesian Nash equilibrium equations for each market) in the constrained optimization formulation. For each replication in the Monte Carlo experiments, we use 100 different starting points for $\boldsymbol{\theta}$ but fix the starting values for P at the estimated choice probabilities $\hat{p} = (\hat{p}^m)_{m=1}^{256}$ from the frequency estimator, in an attempt to find a global solution of the ML estimator.

Our Monte Carlo experiments are conducted in a Matlab environment. However, the optimization problem in each estimator is coded in AMPL and solved by Knitro. The reason that we choose AMPL as the programing platform for solving optimization problems in each estimator is that AMPL uses automatic differentiation to efficiently compute exact first-order and second-order derivatives and pass the derivative information together with sparsity structure of constrained Jacobian and Hessian to solvers. Modern optimization solvers usually perform the best with exact first-order and second-order derivatives. By using AMPL with its capability in automatic differentiation, we hope to give fair comparison of the performance of each estimator.

Monte Carlo results are given in Tables 3 – 5. We report the mean, standard deviation, and root mean squared error (RMSE) based on the empirical distribution of parameter estimates. We also report the average CPU time per run for each estimator and average NPL iterations in each run for the NPL estimator. For Experiments 1 and 2, in which the players' actions are simulated by using best-reply stable equilibria, all four estimators recover the true parameter values well. The standard deviation of MPEC is the smallest among all four estimators. In terms of computational time, not surprisingly, two-step estimators are faster than MPEC as two-step estimators do not need to solve for an equilibrium by solving the Bayesian Nash equilibrium equations. Both MPEC and NPL requires the equilibrium constraints to be satisfied and hence will need more computational time than two-step estimators. As we can see from Tables 3 and 4, MPEC needs only about 1 second of computing time per run. Surprisingly, NPL, on average, needs around 130 NPL iterations (i.e., iterating over two-step estimators for 130 times) and 40 seconds to converge per run. Hence, MPEC is about 40 times faster than NPL in computing time in these two Monte Carlo experiments.

In numerical computation, evaluating a function and its derivatives is computationally expensive. To reduce the computational burden, one also needs to reduce the number of function evaluations performed in the estimation algorithm. The fact that NPL needs around 130 NPL iterations (over the two-step PML) to converge illustrates that NPL is not computationally efficient.

The results for Experiment 3 are intriguing. Recall that in this experiment, we do not specify what type of equilibrium, best-reply stable or best-reply unstable, is to be played in the data. Among the four estimators, MPEC is the only method that recovers the true parameter values. Even though data are generated from some equilibria that are not best-reply stable, the mean estimates of MPEC are accurate: within one standard error of the true values $(-5, 11)$. This result is not surprisingly since algorithms in constrained optimization solver do not rely on best-reply iteration and hence the nature of the equilibria played in the data has no impact on the estimation results for MPEC. Both two-step ML and two-step LS perform much worse than MPEC. The biases in parameter estimates are indeed large. Both 2S-PML and 2S-LS will reject the truth $(\alpha^0, \beta^0) = (-5, 11)$. This result demonstrates that while two-step estimators are computationally light, they sacrifice the accuracy in parameter estimates. Researchers who choose to use two-step estimators should be aware of this trade-off between computing time and the quality of parameter estimates. Although NPL aims to decrease the biases in estimates, in Experiment 3 NPL fails to converge in all 100 replications.[22] This reason is that NPL implicitly requires that equilibria are stable under best-reply iteration. This is a very restricted assumption because there may not exist best-reply stable equilibria in a game. For example, Ferris, Judd, and Schmedders (2010) construct a simple Cournot game with a unique equilibrium. However, that equilibrium is not stable under best-reply. Even if best-reply equilibria exist, with multiple equilibria one does not know a priori which type, stable or unstable under best-reply, of equilibrium will be chosen by the players. There is no theoretical or empirical justification that only best-reply stable equilibria are adopted by firms or strategic agents in practical applications.

[22]In another experiment, NPL converges to $\boldsymbol{\theta}^{NPL} = (-1.1, 3.3)$. Pesendorfer and Schmidt-Dengler (2010) had another example that shows that NPL converges to wrong estimates.

Table 3: Experiment 1: Best-Reply Stable Equilibrium with Lowest Probabilities of Confess for Player a in Each Market

Estimator	Estimates		RMSE	CPU Time (sec)	Avg. NPL Iter.
	α	β			
Truth	-5	11	–	–	–
MPEC	-4.999 (0.031)	10.995 (0.062)	0.069	0.94	–
2-Step PML	-4.994 (0.037)	11.002 (0.092)	0.099	0.36	–
2-Step LS	-5.004 (0.042)	11.027 (0.151)	0.159	0.07	–
NPL	-5.001 (0.031)	10.999 (0.065)	0.072	40.26	125

For each experiment, we solve the model once for the Bayesian Nash equilibria by fixing structural parameters at true values and then simulate and re-estimate the model 100 times. For MPEC, we use 100 starting points to do a better job at finding a global maximum for each of the 100 replications. Standard deviations are reported in parentheses. CPU time and average number of NPL iterations are the averages for each run.

Table 4: Experiment 2: Best-Reply Stable Equilibrium in Each Market

Estimator	Estimates		RMSE	CPU Time (sec)	Avg. NPL Iter.
	α	β			
Truth	-5	11	–	–	–
MPEC	-5.002 (0.024)	10.994 (0.057)	0.062	1.06	–
2-Step PML	-4.997 (0.033)	11.001 (0.103)	0.108	0.36	–
2-Step LS	-5.007 (0.040)	11.027 (0.169)	0.175	0.06	–
NPL	-5.003 (0.028)	10.966 (0.227)	0.230	41.97	132

For each experiment, we solve the model once for the Bayesian Nash equilibria by fixing structural parameters at true values and then simulate and re-estimate the model 100 times. For MPEC, we use 100 starting points to do a better job at finding a global maximum for each of the 100 replications. Standard deviations are reported in parentheses. CPU time and average number of NPL iterations are the averages for each run.

Table 5: Experiment 3: Randomly Chosen Equilibrium in Each Market

Estimator	Estimates		RMSE	CPU Time (sec)	Avg. NPL Iter.
	α	β			
Truth	-5	11	–	–	–
MPEC	-4.999 (0.029)	10.999 (0.057)	0.063	1.02	–
2-Step PML	-4.906 (0.044)	10.828 (0.115)	0.231	0.37	–
2-Step LS	-4.767 (0.052)	10.625 (0.160)	0.472	0.06	–
NPL	Not Converged N/A	Not Converged N/A	N/A	152.30	500

For each experiment, we solve the model once for the Bayesian Nash equilibria by fixing structural parameters at true values and then simulate and re-estimate the model 100 times. For MPEC, we use 100 starting points to do a better job at finding a global maximum for each of the 100 replications. Standard deviations are reported in parentheses. CPU time and average number of NPL iterations are the averages for each run.

6 Conclusion

We illustrate that constrained optimization approaches can be applied directly to maximum likelihood estimation and possibly other problems, such as methods of moments and calibration. Our numerical results on single-agent dynamic discrete-choice models show that our approach is faster than NFXP. Our Monte Carlo results on estimating games of incomplete information show that finite-sample performance of our method are superior to those of two-step estimators, in particular when data are generated from equilibria that are not best-reply stable. In such cases, two-step estimators are severely biased. NPL either fails to converge or converge to incorrect parameter estimates. Overall, we find that NPL requires many outer iterations to converge (when it converges) and hence is not a computationally efficient procedure. Our findings are valuable because researchers should be aware of the trade-off between the computing time or computational convenience and the quality of parameter estimates. Our method can be easily implemented by using state-of-the-art constrained optimization solvers. We believe that constrained optimization approaches will be useful for estimating structural models in various contexts and applications. For future research, we plan to investigate the performance of MPEC approach on dynamic games and the use of the MPEC approach for estimating dynamic discrete-choice models with unobserved heterogeneity studied in Aguirregabiria and Mira (2007), Bajari, Benkard, and J. Levin (2007), and Arcidiacono and Miller (2010).

Appendix A: Proofs

Proof of Proposition 1. The results follow directly from Theorem 2 in Aitchison and Silvey (1958) or discussion in Section 4.7 in Silvey (1975, p. 79–81.). ■

Proof of Proposition 2. The proof for (i) follows directly from the definition of $L(\theta)$ and $\mathcal{L}(\theta, EV)$ in (18) and (20), respectively.

For (ii), by definition, $L(\hat{\theta}) \geq L(\theta)$, for all θ and EV such that $EV = T(EV;\theta)$. Since (θ^*, EV^*) is a solution of the constrained optimization problem and hence satisfies the integrated Bellman equation as a constraint, we have $L(\hat{\theta}) \geq L(\theta^*) = \mathcal{L}(\theta^*, EV^*)$. Conversely, $\mathcal{L}(\theta^*, EV^*) \geq \mathcal{L}(\theta, EV)$ for all θ and EV such that $EV = T(EV;\theta)$. Since $\hat{EV} = T(\hat{EV};\hat{\theta})$, we have $\mathcal{L}(\theta^*, EV^*) \geq \mathcal{L}(\hat{\theta}, \hat{EV}) = L(\hat{\theta})$.

For (iii), if the model is uniquely identified, $\hat{\theta}$ is unique, and hence $\hat{\theta} = \theta^*$. ■

Proof of Proposition 3. For (i), since $(\boldsymbol{\theta}^*, P^*)$ satisfies the Bayesian Nash equilibrium equation, we have $P^* \in \Sigma(\boldsymbol{\theta}^*)$ and hence,

$$L(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{P}}(\bar{\boldsymbol{\theta}})) \geq L(\boldsymbol{\theta}^*, P^*) = \mathcal{L}(\boldsymbol{\theta}^*, P^*).$$

Conversely, since $\bar{\boldsymbol{P}}(\boldsymbol{\theta}) \in \Sigma(\bar{\boldsymbol{\theta}})$, the pair $(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{P}}(\boldsymbol{\theta}))$ satisfies the Bayesian Nash equilibrium equation and hence is a feasible solution for the constrained optimization problem (36). Hence,

$$\mathcal{L}(\boldsymbol{\theta}^*, P^*) \geq \mathcal{L}(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{P}}(\boldsymbol{\theta})) = L(\bar{\boldsymbol{\theta}}, \bar{\boldsymbol{P}}(\bar{\boldsymbol{\theta}})).$$

For (ii), if the model is uniquely identified, $\bar{\boldsymbol{\theta}}$ is unique, and hence $\bar{\boldsymbol{\theta}} = \boldsymbol{\theta}^*$. ■

Appendix B: Automatic Differentiation

We illustrate the basic idea of automatic differentiation with a function familiar to economists. Griewank and Walther (2008) provide an comprehensive introduction on this subject.

Suppose that we want to compute both the value of $f(x, y, z) = (x^\alpha + y^\alpha + z^\alpha)^\gamma$ and its gradient $\bigtriangledown f = (f_x, f_y, f_z)$. One way is to just analytically derive the derivatives and evaluate them separately. This is typically called *symbolic differentiation.* In this case we would evaluate the three functions $\gamma\alpha x^{\alpha-1}(x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}$, $\gamma\alpha y^{\alpha-1}(x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}$, and $\gamma\alpha z^{\alpha-1}(x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}$, each of which is more costly to compute than the original function. The Hessian becomes much worse. For example, the cross-partial $f_{xy}(x, y, z)$ is

$$\alpha^2\gamma(\gamma - 1)x^{\alpha-1}y^{\alpha-1}(x^\alpha + y^\alpha + z^\alpha)^{\gamma-2},$$

which is even more complicated than gradient elements. Furthermore, the Hessian contains nine elements of such second-order derivatives. Practitioners typically resort to finite differences to compute derivatives. The time required for finite difference methods for a Hessian is proportional to the size of the Hessian matrix, n^2, making it inefficient to compute the Hessian if n is large. Furthermore, finite difference methods produce numerical errors. As a result, optimization or nonlinear equation solvers may produce inaccurate solutions or fail to converge when inaccurate derivatives are used in the iteration process.

If we instead compute the analytic gradient and Hessians efficiently, the computational cost will be much lower. The key insight is that by using the chain rule of differentiation, the analytic computation of gradients and Hessians can take advantage of algebraic relations among individual terms. We use the gradient $\bigtriangledown f$ as an example to illustrate the basic ideas. First, we compute the original function, $f(x, y, z) = (x^\alpha + y^\alpha + z^\alpha)^\gamma$. Note that this computation produces values for the individual terms x^α, y^α, z^α, $x^\alpha + y^\alpha + z^\alpha$, as well as $(x^\alpha + y^\alpha + z^\alpha)^\gamma$. As we compute $f(x, y, z)$, we store these values for later use. With these values in hand, the computation of $f_x = (x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}\gamma\alpha x^{\alpha-1}$ needs only two divisions and three multiplications. The reason is that x^α, $x^\alpha + y^\alpha + z^\alpha$, and $(x^\alpha + y^\alpha + z^\alpha)^\gamma$ are known from the $f(x, y, z)$ computation, and $f_x = (x^\alpha + y^\alpha + z^\alpha)^\gamma/(x^\alpha + y^\alpha + z^\alpha) * \ \gamma * \alpha * x^\alpha/x$ just involves three extra multiplications and two extra divisions. Note also how we have used division to compute the necessary exponentiations. In general, if one knows f, f', and f^a, then $(f^a)' = a * f^a * f'/f$. This has the extra advantage of using division to compute an exponentiation, a replacement producing considerable time savings on many computers.

When we move to the other derivatives, we are able to realize even more economies of scale. Since $(x^\alpha + y^\alpha + z^\alpha)^{\gamma-1}\gamma\alpha$ is a common factor among the partial derivatives, we need only one more multiplication and one more division to compute f_y, and similarly for f_z. Hence $\bigtriangledown f$ can be computed at a marginal cost of four divisions and five multiplications. In contrast, the finite-difference method uses 12 exponentiations, 3 divisions, and 9 addition/subtractions. The savings increase as we move to higher dimensions, since the marginal cost of computing a derivative is one multiplication and one division. The following is the pseudocode that efficiently computes f and $\bigtriangledown f$.

```
$x\alpha = x^\alpha;\ y\alpha = y^\alpha;\ z\alpha = z^\alpha$
$s = x\alpha + y\alpha + z\alpha$
$f = s^\gamma$
$t = \gamma\ \alpha\ f\ /\ s$
$f_x = t\ x\alpha/x;\ f_y = t\ y\alpha/y;\ f_z = t\ z\alpha/z;$
```

This is one example of how careful attention to the form of a function can improve the efficiency of derivative computation. Another example is the exploitation of separability in any $f(x, y, z)$ of the form $f_1(x) + f_2(y) + f_3(z)$. The savings are even greater when we consider computing the Hessian, which can make use of the computations already performed for the gradient as well as the many algebraic relations among the second derivatives. It is also clear that the savings relative to finite-difference methods increase as we examine larger problems.

The ideas behind efficiently computing derivatives can be extended for arbitrary functions. The key insight is that by using the chain rule of differentiation, we can build a sequence of simple operations that take advantage of algebraic relations among individual terms to arrive at efficient ways to compute derivatives. The study of methods to exploit these relations and create efficient differentiation code is called *automatic differentiation*.

Arguing against using symbolic and automatic differentiation are the considerable costs of doing the algebra. Many functions are difficult to differentiate. Finite-difference methods avoid any errors that may arise from human differentiation of the objective. Even if one could reliably compute derivatives, much of the savings of automatic differentiation comes from recognizing common terms across derivatives, another process that challenges human algebraic abilities. Fortunately we can now eliminate much of the human error. Derivative computations can be done by symbolic software; in fact much of that software was created to do just this sort of computation. Some symbolic software, such as Maple and Macsyma, can form Fortran and C code that computes the function, its gradient, and Hessian and, furthermore, recognizes common terms. Thus, it is easy to exploit the economies of computation which are available through the analytical computation of gradients and Hessians.

The extensive literature on automatic differentiation formalizes these ideas and their application to real problems. While these methods are potentially very useful, we will not develop them in this paper. These ideas have been systematized in the automatic differentiation literature, which is reviewed in Griewank and Walther (2008) and incorporated into many software packages and optimization modeling language, such as AMPL.

Appendix C: Code

The AMPL and Matlab code used to generate data and to implement the MPEC approach in the Monte Carlo studies will be available at:
`http://faculty.chicagobooth.edu/che-lin.su/research/code.html`

References

[1] Arcidiacono, P. and R. A. Miller (2010): "CCP Estimation of Dynamic Discrete Choice Models with Unobserved Heterogeneity," Working paper, Duke University.

[2] Aguirregabiria, V., and P. Mira (2002): "Swapping the Nested Fixed Point Algorithm: A Class of Estimators for Discrete Markov Decision Models," *Econometrica*, 70, 1519–1543.

[3] Aguirregabiria, V., and P. Mira (2007): "Sequential Estimation of Dynamic Discrete Games," *Econometrica*, 75, 1–53.

[4] Aitchison, J., and S. D. Silvey (1958): "Maximum Likelihood Estimation of Parameters subject to Restraints," *Annals of Mathematical Statistics*, 29, 813–828.

[5] Bajari, P., C. L. Benkard, and J. Levin (2007): "Estimating Dynamic Games of Imperfect Competition," *Econometrica*, 75, 1331–1370.

[6] Berry, S. (1992): "Estimation of a Model of Entry in the Airline Industry," *Econometrica*, 60, 889–917.

[7] Besanko, D., U. Doraszelski, Y. Kryukov, and M. Satterthwaite (2010): "Learning-by-Doing, Organizational Forgetting, and Industry Dynamics," *Econometrica*, 78, 453–508.

[8] Byrd, R. H., J. Nocedal, and R. A. Waltz (2006): "KNITRO: An Integrated Package for Nonlinear Optimization," in *Large-Scale Nonlinear Optimization* ed. by G. di Pillo and M. Roma. Springer Verlag, 35–59.

[9] Ciliberto, F., and E. Tamer (2009): "Market Structure and Multiple Equilibria in Airline Markets," *Econometrica*, 77, 1791–1828.

[10] Czyzyk, J., M. Mesnier, and J. Moré (1998): "The NEOS Server," *IEEE Journal on Computational Science and Engineering*, 5, 68–75.

[11] Dubé, J.-P., J. T. Fox, and C.-L. Su (2009): "Improving the Numerical Performance of BLP Static and Dynamic Discrete Choice Random Coefficients Demand Estimation," Working paper, Booth School of Business, University of Chicago.

[12] Efron, B., and R. J. Tibshirani (1993): *An Introduction to the Bootstrap*, Chapman & Hall/CRC Press.

[13] Ferris, M., K. Judd, and K. Schmedders (2009): “Solving Dynamic Games with Newtons Method,” Working paper, Kellogg School of Management, Northwestern University.

[14] Fourer, R., D. M. Gay, and B. W. Kernighan (2003): *AMPL: A Modeling Language for Mathematical Programming*, Brooks/Cole – Thomson Learning, Pacific Grove, California, second edition.

[15] Fudenberg, D., and J. Tirole (1991): *Game Theory.* Cambridge: MIT Press.

[16] Griewank, A., and A. Walther (2008): *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia, second edition.

[17] Gallant, A. R., and A. Holly (1980): “Statistical Inference in an Implicit, Nonlinear, Simultaneous Equation Model in the Context of Maximum Likelihood Estimation,” *Econometrica*, 48, 697–420.

[18] Gallant, A. R., and G. Tauchen (1989): “Seminonparametric Estimation of Conditionally Constrained Heterogeneous Processes: Asset Pricing Applications,” *Econometrica*, 57, 1091–1120.

[19] Gropp, W., and J. Moré (1997): “Optimization Environments and the NEOS Server,” in *Approximation Theory and Optimization* ed. by M. D. Buhmann and A. Iserles, Cambridge University Press, Cambridge, UK, 167–182.

[20] Horowitz, J. L. (2001): “The Bootstrap,” in *Handbook of Econometrics*, Vol. 5, ed. by J. J. Hackman and E. Leamer, Elsevier Science, Amsterdam, 3159–3228.

[21] Hotz, J., and R. A. Miller (1993): “Conditional Choice Probabilities and the Estimation of Dynamic Models,” *Review of Economic Studies*, 60, 497–529.

[22] Judd, K., P. Renner, and K. Schmedders (2010): “Finding All Pure Strategy Equilibria in Dynamic and Static Games with Continuous Strategies,” Working paper, Kellogg School of Management, Northwestern University.

[23] Kohlberg, E., and J.-F. Mertens (1986): “On the Strategic Stability of Equilibria,” *Econometrica*, 54, 1003–1038.

[24] Luo, Z.-Q., J.-S. Pang, and D. Ralph (1996): *Mathematical Programs with Equilibrium Constraints*, Cambridge University Press, Cambridge, UK.

[25] MacKinnon, J.-G. (2002): “Bootstrap Inference in Econometrics,” *Canadian Journal of Economics*, 35, 615–645.

[26] Mertens, J.-F. (1989): “Stable Equilibria–a Reformulation: Part I. Definition and Basic Properties,” *Mathematics of Operations Research*, 14, 575–625.

[27] Mertens, J.-F. (1991): “Stable Equilibria–a Reformulation: Part II. Discussion of the Definition, and Further Results,” *Mathematics of Operations Research*, 16, 694–753.

[28] Nocedal, J. and S. J. Wright (2006): *Numerical Optimization*, Springer, New York, NY, second edition.

[29] Pakes, A., M. Ostrovsky, and S. Berry (2007): “Simple Estimators for the Parameters of Discrete Dynamic Games, with Entry/Exit Examples,” *RAND Journal of Economics*, 38, 373–399.

[30] Pesendorfer, M., and P. Schmidt-Dengler (2008): “Asymptotic Least Squares Estimators for Dynamic Games,” *Review of Economic Studies*, 75, 901–928.

[31] Pesendorfer, M., and P. Schmidt-Dengler (2010): “Sequential Estimation of Dynamic Discrete Games: A Comment,” *Econometrica*, 78, 833–842.

[32] Rust, J. (1987): “Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher,” *Econometrica*, 55, 999–1033.

[33] Rust, J. (1988): “Maximum Likelihood Estimation of Discrete Control Processes,” *SIAM Journal on Control and Optimization*, 26, 1006–1024.

[34] Rust, J. (1994): “Structural Estimation of Markov Decision Processes,” in *Handbook of Econometrics*, Vol. 4, ed. by R. F. Engle and D. L. McFadden, Elsevier Science, Amsterdam, 3082–3139.

[35] Rust, J. (2000): *Nested Fixed Point Algorithm Documentation Manual: Version 6*, Department of Economics, Yale University.

[36] Rust, J. (2008): Private communication.

[37] Seim, K. (2006): “An Empirical Model of Firm Entry with Endogenous Product-Type Choices,” *RAND Journal of Economics*, 37, 619–640.

[38] Silvey, S. D. (1975): *Statistical Inference*, Chapman & Hall, London.

[39] Wolak, F. A. (1987): "An Exact Test for Multiple Inequality and Equality Constraints in the Linear Regression Model," *Journal of American Statistical Association*, 82, 782–793.

[40] Wolak, F. A. (1989): "Testing Inequality Constraints in Linear Econometric Models," *Journal of Econometrics*, 41, 205–235.