Constrained Optimization Approaches to Estimation of Structural Models

Che-Lin Su* Chicago GSB

Kenneth L. Judd† Hoover Institution

February 2006
This version: April 12, 2008‡

Abstract

Maximum likelihood estimation of structural models is often viewed as computationally difficult. This impression is due to a focus on the Nested Fixed-Point approach. We present a direct optimization approach to the general problem and show that it is significantly faster than the NFXP approach when applied to the canonical Zurcher bus repair model. The NFXP approach is inappropriate for estimating games since it requires finding all Nash equilibria of a game for each parameter vector considered, a generally intractable computational problem. We formulate the problem of maximum likelihood estimation of games as a constrained optimization problem that is qualitatively no more difficult to solve than standard maximum likelihood problems. The direct optimization approach is also applicable to other structural estimation methods such as the methods of moments, and also allows one to use computationally intensive bootstrap methods to calculate inference. The MPEC approach is also easily implemented on software with high-level interfaces. Furthermore, all the examples in this paper were computed using only free resources available on the web.

*Graduate School of Business, The University of Chicago. E-mail:che-lin.su@chicagogsb.edu

†Hoover Institution and NBER. E-mail:judd@hoover.stanford.edu

‡The authors thank Sven Leyffer, James Heckman, Lars P. Hansen, Peter Rossi, Ron Gallant, Ali Hortacsu, Jeremy T. Fox, Jean-Pierre Dubé, Maria-Ana Vitorino, Güenter Hitsch, Lanier Benkard, Karl Schmedders, Felix Kubler, Patrick Bajari, Jorge Nocedal, Richard Waltz, Stephen J. Wright, Jong-Shi Pang, Daniel Ralph, John Birge, Harry J. Paarsch, Timothy Hubbard, Clinton Levitt, Frank Wolak, Raphael Thomadsen, Stephen Ryan, and seminar participants at the University of Chicago, Northwestern University, University of Wisconsin at Madison, Stanford University, the Chicago GSB, 2006 INFORMS Annual Conference, Stanford SITE 2007, the 2007 Argonne-Chicago ICE, the 2007 SAET in KOS, and ICCOPT II & MOPTA-07. The authors are thankful to Richard W. Cottle for his careful reading of this version and detailed suggestions and to John Rust for his comments. The authors are grateful for financial support from the NSF (award no. SES-0631622). The first author is indebted to Stanley Reiter for his host at the CMS-EMS (Math Center) at the Kellogg School of Management.

1 Introduction

Structural estimation of economic models is an important technique for analyzing economic data. However, it is commonly believed that computational demands make it difficult to implement the most powerful statistical methods. For example, maximum likelihood estimation is the gold standard in estimation and Rust (1987) provided a strategy for maximum likelihood estimation of single-agent dynamic programming models, an approach he dubbed the Nested Fixed-Point (NFXP) algorithm. Unfortunately, NFXP is computationally demanding since it repeatedly takes a guess for the structural parameters and solves for the corresponding endogenous economic variables with high accuracy. This has led to a large literature on deriving "computationally light" estimators. While it is clear that NFXP is impractical in many contexts, particularly in the estimation of games, this does not mean that maximum likelihood estimation is intractable. We present a direct optimization approach, called the MPEC (Mathematical Programming with Equilibrium Constraints (Luo, Pang and Ralph(1996)) approach to structural estimation that avoids repetitive solution of the structural model; in fact, the only equilibrium that needs to be solved exactly is the one associated with the final estimate of parameters. The idea behind the MPEC approach is simple: choose structural parameters and endogenous economic variables so as to maximize the likelihood of the data subject to the constraints that the endogenous economic variables are consistent with an equilibrium for the structural parameters. That's it. Nothing more. When formulated in this way, it is clear that all we do is write down expressions defining the likelihood (the objective) and the equilibrium equations (the constraints), and submit them to one or more of the state-of-the-art optimization solvers. These solvers treat the parameters and endogenous variables jointly and do not repeatedly solve for equilibria, making it possible for them to be much faster than NFXP. Furthermore, since our approach allows researchers to directly use state-of-the-art algorithms, the user need not make any decision about the algorithmic details (such as, for example, choosing between BHHH and BFGS, choosing between value function iteration or policy iteration, etc.) that make NFXP and related methods costly for a user to implement even when a model is tractable.

We first illustrate the idea in a simple demand example, related to the constrained maximum likelihood literature. We then show that the MPEC approach is significantly faster than the NFXP approach when applied to the canonical Zurcher bus repair model even though we use no special feature of the problem. To demonstrate the generality of the MPEC approach, we also implement a method of moments estimator to the bus problem. Furthermore, we use a parametric bootstrap method (Efron and Tibshirani (1993)) in both cases to compute standard errors, demonstrating that boostrap methods become much more practical when using the MPEC approach.

Our final examples show that the MPEC approach is immediately applicable to data from games, even when structural parameters do not imply a unique equilibrium. The NFXP approach is infeasible[1] for estimating all but the simplest games since it requires

[1]In the interest of precision, by "infeasible" we mean if we use the state of the art mathematical algorithms

finding all Nash equilibria of a game for each parameter vector considered[2]. In contrast, the MPEC approach just solves a constrained optimization problem that is qualitatively no more difficult to solve than standard maximum likelihood problems, since the key problem in both cases is the possibility of multiple local maxima. In fact, we illustrate in a figure that even if the NFXP approach is feasible for estimating a game, the likelihood function will often be ill-behaved, even discontinuous, if there are multiple equilibria, whereas the presence of multiplicity of equilibria has no impact on the smoothness of the optimization problem solved by the MPEC approach.

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 literatures. In particular, all of the examples below are constrained estimation problems of the kind exposited in Aitchison and Silvey (1958), Silvey (1970), Gallant and Holly (1980), Wolak (1987, 1989), and Gallant and Tauchen (1989)[3]. However, the current econometric literature seems to dismisses this approach as computationally infeasible. We argue in this paper that the constrained optimization approach is quite feasible if one uses standard methods in the mathematical programming literature.

Many readers, particularly practitioners and graduate students, will have practical questions about using the MPEC approach, such as "Do I have to learn Fortran or C?", and "Is my laptop powerful enough?". Our implementation of the MPEC approach in this paper demonstrates the low cost of implementing these ideas[4]. Applying NFXP typically involves writing a computer program using Gauss, Matlab, or some other commercial product, and executing the program on a researcher's computer. The financial costs of the software and hardware often limit what a researcher can do. Even if one has a very good desktop or laptop, it is still is just one computer and he does not want to devote it to these computations for extended periods of time. If one has a good commercial package, it will generally have just one solver that is probably not using state-of-the-art algorithms and possibly not one well-suited for his problem. In this paper, we take a very different approach. All of the computations performed in this paper were done using AMPL (Fourer, Gay and Kernighan (2003)), a modeling language, and NEOS Server (Gropp and Moré (1997), Czyzyk, Mesnier, Moré (1998), and Dolan et al. (2002)), a free internet service which gives the user access to several state-of-the-art solvers and allows him to use other people's computers. All one needs for the computations in this paper is a text editor and access to the internet. Furthermore, NEOS Server allows users to simultaneously submit multiple jobs to several computers at no cost. Therefore, the MPEC approach as implemented in this paper allows one to compute statistically efficient estimators using other people's software and hardware for free.

There is currently much concern about the computational burdens of structural estima-

and supercomputers running at the top known 2007 speed of 200 TFlops, then the NFXP approach could easily take days to solve the problem.

[2]See Judd and Schmedders (2006) for a case where finding all solutions is feasible and a general discussion of the relevant mathematics.

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

[4]In particular, the answers to the questions are clearly "No", and "YES".

tion. The basic idea in this paper is that the best approach to reduce these computational burdens is to solve the key numerical tasks with methods and software developed by computational scientists and mathematicians, and commonly used by scientists and engineers. The mathematical theory behind the numerical methods we use proves that these methods are far better in terms of the rates of convergence. Our examples demonstrate this superiority for well-known examples. Furthermore, our use of standard software and hardware available to anyone for free to compute efficient estimates shows that these methods will make structural models accessible to a larger set of researchers.

2 Current Optimization Methods in Econometrics

Optimization is used heavily in econometrics, but there is a large gulf between current practice in econometrics and the methods discussed in the mathematical programming literature. In this section we discuss the differences and highlight the critical mathematical details.

The econometrics literature contains much pessimism regarding full information maximum likelihood estimation of structural models. For example, Erdem et al. (2005) assert the following in a section titled "Reducing the Computational Burden of Structural Estimation":

> Estimating structural models can be computationally difficult. For example, dynamic discrete choice models are commonly estimated using the nested fixed point algorithm (see Rust (1994)). This requires solving a dynamic programming problem thousands of times during estimation and numerically minimizing a nonlinear likelihood function.....[S]ome recent research ... proposes computationally simple estimators for structural models including auctions, demand in differentiated product markets, dynamic discrete choice and dynamic games. The estimators ... use a two-step approach. In the first step, one flexibly estimates a reduced form for agents' behavior consistent with the underlying structural model. In the second step, the one recovers the structural parameters, by plugging the first-step estimates into the model.....The two-step estimators can have drawbacks. First, there can be a loss of efficiency. The parameters estimated in the second step will depend on a nonparametric first step. If this first step is imprecise, the second step will be poorly estimated. Second, stronger assumptions about unobserved state variables may be required. In a dynamic discrete choice model, accounting for unobserved heterogeneity by using random effects or even a serially correlated, unobserved state variable may be possible using a nested fixed point approach. However, two-step approaches are computationally light, often require minimal parametric assumptions and are likely to make structural models accessible to a larger set of researchers.

The prevailing pessimism about structural estimation is particularly strong when it comes to the discussion of games. For example, Aguirregabiria and Mira (2007) state that

[M]ost applications of empirical discrete games have estimated static models. Two main econometric issues have limited the scope of applications to relatively simple static games: the computational burden in the solution of dynamic discrete games, and the indeterminacy problem associated with the existence of multiple equilibria.....

The existence of multiple equilibria is a prevalent feature in most empirical games where best response functions are non-linear in other players' actions. Models with multiple equilibria do not have a unique reduced form and this incompleteness may pose practical and theoretical problems in the estimation of structural parameters. In particular, maximum likelihood and other extremum estimators require that we obtain all the equilibria for every trial value of the parameters. This can be infeasible even for simple models.

They make this more precise with the following discussion:

Define the following pseudo likelihood function:

$$Q_M(\theta, P) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T} \sum_{i=1}^{N} \ln \Psi_i(a_{imt}|x_{mt}; P, \theta) \tag{24}$$

where P is an arbitrary vector of players' choice probabilities. We call this function a pseudo likelihood because the choice probabilities are not necessarily equilibrium probabilities associated with θ, but just best responses to an arbitrary vector P. Consider first the hypothetical case of a model with a unique equilibrium for each possible value of $\theta \in \Theta$. Then, the maximum likelihood estimator (MLE) of θ^0 can be defined from the following constrained multinomial likelihood:

$$\hat{\theta}_{MLE} = \arg\max_{\theta \in \Theta} Q_M(\theta, P) \text{ subject to: } P = \Psi(\theta, P) \tag{25}$$

The computation of this estimator requires one to evaluate the mapping Ψ and the Jacobian matrix $\delta\Psi/\delta P'$ at many different values of P. Though evaluations of Ψ for different θ's can be relatively cheap because we do not have to invert the matrix $(I - \beta F)$ in (14), evaluations for different P imply a huge cost when the dimension of the state space is large because this matrix needs to be inverted each time. Therefore, this estimator can be impractical if the dimension of P is relatively large. For instance, that is the case in most models with heterogenous players because the dimension of the state space increases exponentially with the number of players. For that type of models this estimator can be impractical even when the number of players is not too large.

Aguirregabiria and Mira (2007) are correct in stating that the MLE problem is a con-

strained optimization problem[5]. However, the characterizations of the literature on numerical methods for constrained optimization problems found in Erdem et al. (2005) and Aguirregabiria and Mira (2007) are inaccurate on several dimensions.

First, Erdem et al. (2005) are correct in asserting that the nested fixed point algorithm "requires solving a dynamic programming problem thousands of times during estimation and numerically minimizing a nonlinear likelihood function" but this is only a property of NFXP and its related methods. In contrast, standard constrained optimization methods avoid this problem. To see this clearly, consider problem

$$\begin{array}{ll} \max\limits_{(\theta,y)} & f(\theta,y) \\ \text{subject to} & y = g(\theta,y). \end{array} \tag{1}$$

The NFXP is a nonlinear substitution of variables method. If we define $Y(\theta)$ implicitly in $Y(\theta) = g(\theta, Y(\theta))$, then this problem reduces to the unconstrained problem

$$\max_{\theta} f(\theta, Y(\theta)). \tag{2}$$

If $Y(\theta)$ is a smooth function then this unconstrained optimization problem can be solved by standard unconstrained optimization methods. In NFXP, $Y(\theta)$ is computed numerically for each value of θ if $Y(\theta)$ cannot be computed in closed form. The frequent computation of $Y(\theta)$ is the potential weakness.

PML (pseudo-maximum likelihood) is essentially a Gauss-Seidel method. Suppose that you have a guess for y, say y^i. Then the next guesses for y and θ are given by

$$\begin{array}{rcl} \theta^{i+1} & = & \arg\max\limits_{\theta} f(\theta, y^i) \\ y^{i+1} & = & g(\theta^{i+1}, y^i). \end{array} \tag{3}$$

The advantage of a Gauss-Seidel method is that one just evaluates the expression $g(\theta^{i+1}, y^i)$ at each iteration instead of solving $y = g(\theta^{i+1}, y)$ for y^{i+1}, but the disadvantage is the fact that Gauss-Seidel methods have at best linear convergence. More problematic is the fact that it is difficult to prove convergence, even local, of a Gauss-Seidel method for solving nonlinear equations.

In contrast, the constrained optimization methods we will use below are quadratically convergent since they are based on Newton's method. We describe the basic ideas behind most constrained optimization methods below. Suppose that you want to solve the constrained optimization problem (1). The first step is to define the Lagrangian $f(\theta,y) - \lambda^T(y - g(\theta,y))$, and then write down the first-order conditions

$$\begin{array}{rcl} \nabla_\theta f(\theta,y) - \nabla_\theta (y - g(\theta,y))^T \lambda & = & 0 \\ \nabla_y f(\theta,y) - \nabla_y (y - g(\theta,y))^T \lambda & = & 0 \\ y - g(\theta,y) & = & 0. \end{array} \tag{4}$$

[5]Rust (2000, Section 3.1, pp. 17) also describes the Zurcher bus repair problem as a constrained optimization problem. On this point, there is no disagreement. Therefore, the entire issue is how to best solve the constrained optimization problems that arise in structural estimation.

The next step is to apply Newton's method to solve the first-order conditions (4). This approach can be much faster than either Gauss-Seidel or nonlinear elimination of variables since Newton's method is quadratically convergent and the constraint $y - g(\theta, y) = 0$ only needs to be satisfied exactly at the solution (θ^*, y^*). By combining Newton's method with a variety of other techniques, the mathematical programming literature of the past 40 years has developed algorithms that are far more efficient and robust than both nonlinear elimination of variables and nonlinear Gauss-Seidel methods[6].

Second, it is not true that "evaluations for different P imply a huge cost when the dimension of the state space is large because this matrix needs to be inverted each time." Inverting large matrices is expensive, however linear equation solvers never invert a matrix. Instead, they compute an LU decomposition and then use backsolving to compute the solution. Moreover, these Jacobians are often sparse, in which case large linear systems are solvable even though inversions are infeasible.

Third, Aguirregabiria and Mira (2007) also appear to worry about the cost of computing $\delta\Psi/\delta P'$, the Jacobian matrix of the constraint $P = \Psi(\theta, P)$. The use of derivatives is an important feature in Newton's method. In fact, much of the computing time is devoted to obtaining first-order derivatives, such as gradients and Jacobians, and second-order derivatives, such as Hessians. While the computation time for derivatives was a concern in the past, the development of automatic differentiation has eliminated this as a serious problem.

We illustrate this with a function familiar to economists. 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. Economists and econometric software 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 intractable to compute the Hessian if n is large and Ψ is not trivial to compute.

[6]The mathematical programming literature has never considered nonlinear elimination of variables a serious option. The same is true of Gauss-Seidel methods. In very large problems, Gauss-Seidel iterative methods are often used to solve the linear equations that define a Newton step, but these iterative methods for linear problems are generally reliable, and the Newton character still implies quadratic convergence for the overall algorithm.

For sufficiently large problems, some nonlinear elimination of variables may be necessary due to space limitations. The definition of "large" depends, of course, on the available hardware. Currently, "large" means more than one hundred million unknowns, a size not common in economics.

If we instead compute the analytic gradient and Hessians efficiently, the 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 now use $\bigtriangledown f$ 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 2 divisions and 3 multiplications. This is because 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 3 extra multiplications and 2 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 only need 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 4 divisions and 5 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 which efficiently computes f and $\bigtriangledown f$.

$$
\begin{aligned}
x\alpha &= x^\alpha;\ y\alpha = y^\alpha;\ z\alpha = z^\alpha \\
s &= x\alpha + y\alpha + z\alpha \\
f &= s^\gamma \\
t &= \gamma\ \alpha\ f\ /\ s \\
f_x &= t\ x\alpha/x;\ f_y = t\ y\alpha/y;\ f_z = t\ z\alpha/z;
\end{aligned}
$$

This is just one example of how careful attention to the form of a function can improve the efficiency of derivative computation. Another example would be 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 which computes the function, its gradient, and Hessian and, furthermore, recognize common terms. This makes it 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; for example, see Kalaba et al. (1983) and Tesfatsion (1991). 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 Corliss (1991) and Berz et al. (1996) and incorporated into many software packages and optimization modeling language, such as AMPL and GAMS.

Fourth, while the Hessians may be large in terms of the number of elements, they are not really large if most entries are zero. Sparse Jacobians and Hessians occur very often in economic models, and computations involving these matrices automatically ignore the zero elements.

Of course, Aguirregabiria and Mira (2007) are correct in thinking that computing Jacobians and inverting matrices *could* be impediments to a constrained optimization approach, as would be the case if one were using finite differences and not exploiting any sparseness. But these considerations are not important today; in fact, sparse matrix methods were included in early releases of MINOS (Murtagh and Saunders (1977, 1982, 1983)) over 20 years ago. Their assertion that a constrained optimization approach "can be impractical if the dimension is relatively large" is vacuously true since any numerical method becomes impractical if the dimension becomes large such as would be the case if P had 10^9 dimensions. However, this observation is relevant only if the constrained optimization approach is impractical at sizes for which their alternative is practical. They chose to offer no evidence on that point.

Aguirregabiria and Mira (2007) continue their discussion on estimating games with multiple equilibria[7]:

> An important complication in the estimation of dynamic games is that for some values of the structural parameters the model can have multiple equilibria.

[7]Implicit in this discussion is that the problem is formulated in such a way that a maximum likelihood estimation procedure is valid even when there are multiple solutions to the Nash equilibrium equations. The example we present is one such case.

With multiple equilibria the restriction $P = \Psi(\theta, P)$ does not define a unique vector P but a set of vectors. In this case, the MLE can be defined as:

$$\hat{\theta}_{MLE} = \arg\max_{\theta \in \Theta} \left\{ \sup_{P \in (0,1)^{N|X|}} Q_M(\theta, P) \text{ subject to: } P = \Psi(\theta, P) \right\} \tag{26}$$

This estimator can be shown to be consistent, asymptotically normal and efficient. However, in practice, this estimator can be extremely difficult to implement. Notice that for each trial value of θ we have to compute all the vectors P which are an equilibrium associated with θ and then select the one with maximum value for $Q_M(\theta, P)$. Finding all the Markov Perfect equilibria of a dynamic game can be very difficult even for relatively simple models (see McKelvey and McLennan, 1996). Note also that with multiple equilibria the number of evaluations of Ψ for different values of P increases very importantly. These problems motivate the pseudo likelihood estimators we develop in the following subsections.

While the mathematical literature supports Aguirregabiria and Mira (2007) in their assessment of the tractability of computing all equilibria of a game, this does not imply that we should dismiss constrained optimization methods. In particular, their key assertion that the "we have to compute all the vectors P which are an equilibrium associated with θ" is contradicted by the constrained optimization literature.

Another reason why the pessisism about maximum likelihood estimation is not well-founded is that it appears to be based on the presumption that econometricians are limited to desktop computers or computing equipment of similar power. In modern scientific computing, many commonly used methods are implemented in much more powerful computing environment. For example, Ferreyya (2007) used over 200,000 cpu hours on the Condor cluster at UW-Madison.

While the negative portrayals of numerical optimization commonly seen in economics are false, there is always the possibility that economics problems have unique features that make possible ad hoc methods superior to standard numerical optimization methods. It is impossible to argue that standard numerical optimization methods will always do better than some ad hoc method which takes advantage of some special structure. However, we show below that standard numerical optimization methods, without any tweaking, can solve problems similar to those that have appeared in the economics literature and do so faster than methods that incorporate special features of underlying economic models. These examples will serve two purposes: first, to show that standard numerical optimization methods are practical and efficient, and second, to describe the critical features of economic models that will make these methods applicable.

3 MPEC Approach to Estimation

We first formulate the general MPEC method. Suppose that an economic model is described by parameters θ, and that in equilibrium the observable economic variables, X, are random variables from a distribution parameterized by both θ and a vector of endogenous variables, σ. One example is a dynamic programming problem where θ are the parameters for costs, benefits, 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, market balance conditions, etc., which are expressed in the system of equations[8] denoted by

$$G(\theta, \sigma) = 0. \tag{5}$$

This approach often fits into a variety of mathematical categories, such as bilevel optimization, inverse optimal problems, inverse parameter identification problems (Ferris and Tin-Loi (2001) and Pang and Tin-Loi (2001)), mathematical programs with equilibrium constraints (MPEC), or mathematical programs with complementarity constraints (MPCC). We refer readers to Facchinei and Pang (2003), Lou, Pang and Ralph (1996) and the references therein. We have chosen to use the term MPEC because it is a good fit for the ideas in this paper and in future generalizations.

If we denote the likelihood of a data set, X, conditional on parameter θ by $L(\theta; X)$, maximum likelihood estimation solves

$$\max_{\theta} L(\theta; X) \tag{6}$$

with the consistency requirement between θ and σ described in (5) being implicitly taken into account. This approach disguises many critical details by not explicitly expressing the dependence of L on σ. In many ways, 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.

To capture these details, we construct an *augmented likelihood function*[9], $\mathcal{L}(\theta, \sigma; X)$, which explicitly expresses the dependence of the likelihood on σ. This function 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

[8] In this paper, we stay with the simpler case where equilibrium is defined by a set of equations. More generally, equilibrium will be a system of complementarity constraints; see Facchinei and Pang (2003), and Luo, Pang and Ralph (1996) . That case is mathematically more challenging due to failures of constraint qualifications. However, those difficulties can be addressed by methods developed in the MPEC literature.

[9] We thank Peter Rossi for suggesting this term. We like this term since it describes the key idea and clearly distinguishes it from other adaptations of the likelihood approach, such as pseudo-maximum likelihood.

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 compute the maximum likelihood estimate for θ we want the σ we compute to be consistent with θ. This is achieved by imposing the equilibrium conditions (5). Therefore, maximum likelihood estimation is the constrained optimization problem

$$\begin{array}{ll} \max\limits_{(\theta,\sigma)} & \mathcal{L}(\theta,\sigma;X) \\ \text{subject to} & G(\theta,\sigma)=0. \end{array} \tag{7}$$

The two formulations, (6) and (7), are mathematically equivalent. Let $\Sigma(\theta)$ be the set of all σ such that $G(\theta,\sigma)=0$; Σ is implicitly defined by

$$G(\theta,\Sigma(\theta))=0.$$

Then the likelihood and augmented likelihood functions satisfy

$$L(\theta;X)=\mathcal{L}(\theta,\Sigma(\theta);X).$$

This equivalence also shows that the augmented likelihood function is a more fundamental expression of the problem since one can define L in terms of $\mathcal{L}$ and Σ, but not vice versa.

This is a very general formulation. The MPEC approach does not require that equilibrium be defined as a solution to a fixed-point equation. We do not need to specify an algorithm for computing $\sigma=\Sigma(\theta)$; it is doubtful that we could do better than a good solver. Gauss-Jacobi or Gauss-Seidel methods are often used in economics even though they are at best linearly convergent, whereas good solvers are at least superlinearly convergent locally (if not much better) and have better global convergence properties than GJ and GS typically do. Using a direct optimization approach allows one to take advantage of the best available methods and software from the mathematical programming literature.

One advantage of the MPEC approach is obvious: our augmented likelihood function, $\mathcal{L}(\theta,\sigma;X)$, uses *only single-valued functions!* When $\Sigma(\theta)$ is multivalued, $L(\theta;X)$ is multivalued and the maximum likelihood optimization problem is difficult to solve if not intractable.

Another obvious point is that there is nothing unique about the likelihood function as an objective. The MPEC approach could be applied using any loss (or gain) function, such as least squares, weighted GMM, simulated maximum likelihood, etc., as the objective. The sections below will present a variety of applications of the MPEC approach.

3.1 The Geometry of Structural Estimation

We next illustrate the basic ideas of NFXP and MPEC estimation in a graphical form that will help make clear various distinctions. Figure 1 with heading NFXP shows what the likelihood function may look like for the NFXP solver. The thin lines represent points where there is an equilibrium corresponding to parameter θ and with likelihood L. The true likelihood function picks the maximum of such points. The likelihood function could be discontinuous due to the presence of multiple solutions at some values of θ, the structural parameters. This can easily create difficulties for NFXP, particularly if θ is multidimensional.

Figure 2 illustrates the MPEC approach. We create an objective function that depends explicitly on both θ and σ. The solution set of equilibrium conditions imposed on the (θ, σ) vector is modeled as manifolds in (θ, σ)–space and is represented by the semi-transparent surfaces. The augmented likelihood function is defined for all θ and σ and is represented by the colored hills; the red color indicates high likelihood value and the blue indicates low values. The maximum likelihood problem then becomes one of finding the point on the equilibrium manifold that is a maximal with respect to the augmented likelihood function. The solution to the maximum likelihood problem is the one with the highest value. There are many obvious advantages to the MPEC approach. First, the constraints and the augmented likelihood function are continuous and differentiable as in most economic applications. Therefore, the MPEC approach does not introduce discontinuities and nondifferentiabilities that are not part of the structure of the problem. Second, the NFXP algorithm enforces equilibrium at each (θ, σ) examined, implying that each iterate must lie on the equilibrium manifold. In contrast, most optimization algorithms do not enforce feasibility at each iterate; during the solving process, iterates can move through infeasible region because feasibility only needs to be satisfied at the solution. This is one reason why methods that allows infeasible iterates are usually much faster than those that do enforce feasibility. This extra degree of freedom is made clear in Figure 2 where the feasible points are distinct from infeasible points.

3.2 Inference Computations

While the MPEC approach is equivalent to alternative approaches, implementing aymptotic inference methods is more complex with the MPEC approach. Computing standard errors for maximum likelihood estimates require the computation of the Hessian of the likelihood function, $L_{\theta\theta}$. This is a direct computation if one has a closed-form solution for the likelihood, or a numerical procedure as in the NFXP method. To use MPEC results to compute standard errors, we need to work through the implicit construction of the likelihood function. Recall that the likelihood and augmented likelihood functions are related by $L(\theta) = \mathcal{L}(\theta, \Sigma(\theta))$. To compute $L_{\theta\theta}$, we need to compute the Jacobian of Σ with respect to θ. The details of this computation and proofs concerning its validity for the case of equality constraints are contained in Aitchison and Silvey (1958). Wolak (1987) analyzes problems with both equality and inequality constraints. Since this piece of the MPEC approach is standard

statistics, we do not spend time discussing it nor do we implement these methods.

Instead, we use boostrap methods to construct standard errors. If it is not feasible to compute the estimate once, then bootstrap methods are also not feasible since they resolve the same problem many times. This infeasibility is particularly true if the maximum likelihood problem has multiple local maxima. In that case, one cannot just use the initial estimate as the only initial guess, but instead must resolve the analysis of each synthetic data set many times from significantly different initial guesses. Hence, using M resamplings will take M times as much time to compute as it took to compute the point estimate. This displays another advantage of the MPEC approach since it allows one to use bootstrap estimates of the standard error and avoid the finite sample bias that may arise with standard asymptotic methods.

4 MPEC Applied to a Simple Demand Example

We first apply the basic idea to a familiar example. Suppose that you have data on demand, q, and price p, but that demand is observed with error ε. Hence, true demand is $q-\varepsilon$. Suppose that you assume a parametric form for the utility function $u(c;\beta)$ where β is a vector of parameters. Economic theory says that $u_c(c;\beta) = u_c(q-\varepsilon;\beta) = p$, thereby providing a structural relation that can be used to estimate β.

Consider first the way you were taught how to do this. For example, you might assume a functional form for demand, such as

$$u(c) = c - \beta c^2.$$

In this case, you were told to compute the demand function

$$c = (1-p)/(2\beta),$$

which in turn implies that the i'th data point satisfies

$$q_i = (1-p_i)/(2\beta) + \varepsilon_i,$$

for some ε_i. To estimate β, you were told to minimize the sum of squared errors, as in

$$\sum_i \left(q_i - (1-p_i)/(2\beta)\right)^2.$$

The closed-form demand function approach has severe limitations. For example, suppose you wanted to examine the utility function

$$u(c) = c - \left(\beta c^2 + \gamma c^4 + \delta c^6\right),$$

which is as reasonable a utility function as the quadratic utility function. The problem is that the first-order condition implies

$$1 - \left(2\beta c + 4\gamma c^3 + 6\delta c^5\right) = p,$$

an equation which has no closed-form solution expressing demand in terms of price. The closed-form approach immediately breaks down, and it is impossible to continue!

If a closed-form solution does not exist, one can still proceed by dealing with the first-order condition directly. The first-order condition contains all the information you can have. The demand function is just another way of representing the same information, but is often difficult, if not impossible, to express in a compact and efficient manner. It is unnecessary as well as absurd to limit yourself to the few special cases for which there are closed-form solutions to express the demand function, or any other function that arises.

The proper way to proceed is to realize that all you want to do in least squares estimation is to find the errors that are smallest in terms of their sum of squares but can be consistent with the critical structural equation. For our consumption demand model, this is the problem

$$\begin{array}{ll} \min\limits_{(\varepsilon_i,\beta)} & \sum\limits_i \varepsilon_i^2 \\ \text{subject to} & u_c\left(q_i - \varepsilon_i; \beta\right) = p_i, \quad \forall\, i. \end{array} \tag{8}$$

In the case of the quadratic utility function, this reduces to

$$\begin{array}{lll} \min\limits_{(c_i,\varepsilon_i,\beta)} & \sum\limits_i \varepsilon_i^2 & \\ \text{subject to} & 1 - 2\beta c_i = p_i, & \forall\, i \\ & q_i = c_i + \varepsilon_i, & \forall\, i, \end{array} \tag{9}$$

and in the case of the degree-six utility function, the problem becomes

$$\begin{array}{lll} \min\limits_{(c_i,\varepsilon_i,\beta,\gamma,\delta)} & \sum\limits_i \varepsilon_i^2 & \\ \text{subject to} & 1 - (\beta 2c_i + 4\gamma c_i^3 + 6\delta c_i^5) = p_i, & \forall\, i \\ & q_i = c_i + \varepsilon_i, & \forall\, i \\ & -(2\beta + 12\gamma c_i^2 + 30\delta c_i^4) \leq 0, & \forall\, i, \end{array} \tag{10}$$

where the last constraint on (β, γ, δ) impose concavity on the estimated utility function.

Even when you can solve for the demand function, it is not clear you want to. Consider the case

$$\begin{aligned} u\left(q\right) &= q - \beta_1 q^2 - \beta_2 q^3 - \beta_3 q^4 - \beta_4 q^5 \\ u'\left(q\right) &= 1 - 2\beta_1 q - 3\beta_2 q^2 - 4\beta_3 q^3 - 5\beta_4 q^4. \end{aligned}$$

The demand function is one of the four solutions to the quartic polynomial in the first-order condition, where the four solutions are

$$\begin{aligned} q &= -\frac{c}{4d} \pm \frac{\sqrt{D + F/2}}{2} \pm \frac{\sqrt{-D + F - G}}{2} \\ G &= \frac{c^3 - 4bcd + 8ad^2}{\sqrt{8}d^3\sqrt{2D + F}},\ F = \frac{c^2}{2d^2} - \frac{4b}{3d} \\ D &= \frac{2^{1/3}B}{3C^{1/3}d} - \frac{C^{1/3}}{3\ 2^{1/3}d},\ C = A\sqrt{A^2 - 4B^3} \\ B &= b^2 - 3ac + 12d \\ A &= 2b^3 - 9abc + 27c^2 + 27a^2d - 72bd, \end{aligned}$$

and

$$a = 2\beta_1,\ b = 3\beta_2,\ c = 4\beta_3,\ d = 5\beta_4.$$

To compute demand, one needs to compute all four solutions and then pick the one that makes economic sense, such as being a positive real number, satisfying the second-order optimality condition, and being globally optimal.

Since the demand function involves many floating point operations, some of which, such as roots, are expensive to compute, it is much slower to compute than the first-order conditions. Also, the least-squares problem is "much more nonlinear" when we use the explicit demand function, involving cube and square roots, than the simper polynomial terms that arise in the constrained optimization approach. Mathematically, the MPEC approach includes the standard approach. Suppose there is a closed form solution for demand, $D(p, \beta)$, with parameters β. In this case, one way to implement the MPEC method is to solve

$$\begin{aligned} \min_{(c_i, \varepsilon_i, \beta)} \quad & \sum_i \varepsilon_i^2 \\ \text{subject to} \quad & c_i = D(p_i, \beta) \quad \forall\, i \\ & q_i = c_i + \varepsilon_i \quad \forall\, i, \end{aligned} \tag{11}$$

where the constraints $c_i = D(p_i, \beta)$ are just algebraically altered versions of $u_c(c_i; \beta) = p_i$. Which version is better depends on practical issues such as cpu time and the amount of nonlinearity. Therefore, the real task is writing the constrained optimization problem in a manner that minimizes the total computational burden given the finite speed and precision of computers.

These simple examples show that the habit of restricting models to cases with closed-form solutions is unnecessary. There is no reason for economists to impose this burden on themselves instead of writing down the models they really want to analyze and solving them.

5 MPEC applied to Zurcher

We apply the MPEC method to the bus repair model analyzed in Rust (1994). We choose the Rust model since it is often used to illustrate the ideas of maximum likelihood estimation of dynamic models and to evaluate alternative methods.

The bus repair problem considers the decisions faced by a repairman, Harold Zurcher, who must decide whether to perform extensive repairs on a bus when it comes into the bus barn and return it to excellent shape, or to implement less costly activities. The state variable is x, the accumulated mileage *since the last engine replacement*. Let $c\left(x,\theta^c\right)$ be the expected operating costs per period at mileage x, where θ^c is a vector of parameters of the cost function. The operating costs are the sum of observed maintenance, fuel, and insurance costs and unobserved loss costs (by the econometrician) due to unexpected bus breakdowns; hence, θ^c is to be estimated from the observations of when regular maintenance is performed. Let RC denote the expected replacement cost to install a new engine net of any scrap value of the old engine. Rust assumes that the mileage travelled by a bus during one month is exponentially distributed with parameters θ^p, independent of mileages driven in previous months. In each period, a bus arrives in state x and Zurcher decides between (i) "normal maintenance" which incurs operating costs $c(x,\theta^c)$, and (ii) replace the bus engine, with an expected replacement cost RC and incurring operating costs $c(0,\theta^c)$ for a bus with a new engine. Zurcher chooses a replacement policy to minimize the total expected discounted costs.

The data is the time series $(x_t^i, d_t^i)_{t=1}^T, i = 1,\ldots,M$, where x_t^i is state of bus i examined in period t and d_t^i is the replacement decision made for that bus, where $d_t^i = 1$ if the engine is replaced and $d_t^i = 0$ otherwise. The data set may also merge data from different busses. The objective is to find the parameter values that maximize the likelihood of that data. In particular, given $(x_t^i, d_t^i)_{t=1}^T, i = 1,\ldots,M$, we want to infer the unknown parameter vector $\theta = (\theta^c, RC, \theta^p)$ by maximizing the likelihood function

$$L(\theta) = \prod_{i=1}^{M}\prod_{t=2}^{T} P(d_t^i|x_t^i,\theta)p(x_t^i|x_{t-1}^i,d_{t-1}^i,\theta), \tag{12}$$

where $P(d|x,\theta)$, the probability of choosing decision d given the state x and parameter vectors θ, is given by the multinomial formula

$$P(d|x,\theta) = \frac{exp\{u(x,d,\theta)+\beta EV(x,d)\}}{\sum_{d'\in D(x)} exp\{u(x,d',\theta)+\beta EV(x',d)\}}. \tag{13}$$

The expected value function $EV(x,d)$ in (13) is the unique fixed point to the contraction mapping

$$\begin{aligned} EV(x,d) &= T_\theta(EV)(x,d) \\ &\equiv \int_{x'=0}^{\infty} \log\left[\sum_{d'\in D(x')} exp\{u(x',d',\theta)+\beta EV(x',d')\}\right] p(dx'|x,d,\theta), \end{aligned} \tag{14}$$

where

$$u(x,d,\theta) = \begin{cases} -RC - c(0,\theta), & \text{if } d = 1 \\ -c(x,\theta), & \text{if } d = 0. \end{cases}$$

We refer readers to Rust (1994) for detailed derivation on these equations.

We want estimate the unknown parameter vector $\theta = (\theta^c, RC, \theta^p)$ by solving the constrained optimization problem

$$\begin{aligned} \max_{(\theta,EV)} \quad & \prod_{i=1}^{M}\prod_{t=2}^{T} P(d_t^i|x_t^i,\theta)p(x_t^i|x_{t-1}^i,d_{t-1}^i,\theta) \\ \text{subject to} \quad & EV = T_\theta(EV). \end{aligned} \tag{15}$$

This is not possible since EV is a function on the real line $\mathbb{R}$, implying that the constraint is a functional equation. Therefore, we need to approximate the expected value function EV in a finite-dimensional manner. We take the same approach as in Rust and discretize the state space. Let N be the number of states for x, and Z the set of states. This approach requires that the data also live on the same finite grid. Therefore, the data is transformed, replacing each x_t^i with the nearest value in Z, which we denote $\hat{x}_t^i$. Hence, the computational approximation solves

$$\begin{aligned} \max_{(\theta,EV)} \quad & \prod_{i=1}^{M}\prod_{t=2}^{T} P(d_t^i|\hat{x}_t^i,\theta)p(\hat{x}_t^i|\hat{x}_{t-1}^i,d_{t-1}^i,\theta) \\ \text{subject to} \quad & EV(z) = T_\theta(EV)(z)\,,\ z \in Z. \end{aligned} \tag{16}$$

5.1 A Numerical Example

We consider a specific example of Rust's bus engine replacement model. We choose the state space $Z = \{1,\ldots,N\}$ and assume the cost function $c(x,\theta)$ is quadratic, i.e., $c(x,\theta^c) = \theta_1^c x + \theta_2^c x^2$. We do not estimate the discount factor β and let $\beta = 0.95$. The unknown parameters to be estimated are cost function parameters, θ^c, replacement cost RC, and θ^p, parameters defining the Markov transition probabilities. We simulate time series data for T time periods ($T = 10^3, 10^4$) by using the following parameter values: $\beta = 0.95$, $RC = 1.217$, $\theta^c = (0.06, 0.0005)$, and $\theta^p = (0.35, 0.60, 0.05)$. We submitted the problem, coded in AMPL, to the SNOPT solver via NEOS. The computer used on NEOS to solve the problem was an ordinary desktop computer.

We first estimate three parameters in transition probabilities θ_3. Rust (1987) estimated the Markov transition parameters in a first stage and then used those probabilities in his NFXP estimation of the remaining parameters. We report the results in Table 1.

A statistically more efficient approach is to estimate all parameters, including the Markov transition parameters, in one maximum likelihood estimation procedure since decisions help

reveal what Zurcher knows about the probabilities. We report the results of five parameter estimates in Table 2.

A variety of points are clear. First, the problems are solved quickly, usually around a second and requiring significantly more time only with the larger data sets and the finer grid in the dynamic programming approximation.

Second, notice how the estimates differ as we refine the approximation of the dynamic programming problem. The true state space is continuous. We discretize the state space as did Rust, but we examine a variety of discretizations. The cases where we used 1000 states are ten times as fine in their discretizations as in Rust. It appears that coarser discretizations produce nontrivial errors in the estimated parameters.

We also want to compare the speeds with those reported in Rust (1987). This comparison of speed cannot rely on clock time since different computers have different capabilities, particularly when they are of such different vintages. The key measure of time is the number of times the objective functions and constraints need to be computed. We see that very few evaluations are necessary, even when estimating five parameters. In NFXP, solving one dynamic programming problem would require dozens of evaluations of the Bellman equations (our constraints). Due to the use of finite differences for derivatives, each outer iteration requires solving four dynamic programming problems and evaluating the likelihood four times when estimating three parameters. These details all depend on the starting guess. We initially assumed that the value function is zero. If NFXP used that starting guess, our MPEC method would have finished the whole problem before NFXP has solved even one dynamic programming problem! All methods benefit from good initial guesses. The key fact is that NFXP solves many dynamic programming problems, whereas the MPEC method uses far fewer evaluations of the likelihood and the Bellman equations since it solves only one dynamic programming problem.

5.2 Parametric Bootstrap

Statistical inference relies on the computation of a standard error. Above we described how one could compute the Hessian of the likelihood function at the point estimate. Some prefer bootstrap methods for computing standard errors. Boostrapping may seem difficult for the MPEC method since we would need to resolve the estimation problem many times. We show that this is not a problem.

We used a parametric bootstrap method (Efron and Tibshirani (1993)) for our example. We took the solution to the policy function implied by our point estimate to generate several synthetic data samples of the same size as our original problem. This gives us some idea about precision of these estimates. The computational burden of a resampling approach would be the same since the key feature of bootstrap estimates is to repeat the estimation on several "similar" sets of data.

We should note that our application of bootstrap methods to the Zurcher bus model may not be statistically valid. The point of our example is to emphasize that computational burden is not a barrier to using bootstrap methods for this model, and to generally point out that one way to use the increased speed of MPEC methods is to apply inference methods that are not computationally tractable when using NFXP methods.

We generated 20 data sets of 1000 points each. We kept the number of new samples small for the purposes of this exercise. The computational demands scale linearly, so the time is increased by a factor of ten if one wanted to use 200 synthetic samples. Each data set takes the point estimates for the parameters, computes the implied decision rule, and simulates the data process. We then estimated the five parameters using 1001 grid points in the dynamic programming approximation.

The results are displayed in Table 3 where we report the standard errors generated by the bootstrap, and the computational costs. Note that the running times were all similar for the various data sets, indicating that the cost of the bootstrap method is relatively predictable and proportional to number of synthetic samples. We should note that that the standard errors we report are probably not statistically valid since many data sets produced corner solutions to the cost function parameters due to the monotonicity and convexity conditions we imposed. While the statistical results may not be correct, this example does make the point that bootstrap methods are more feasible when used in conjunction with an MPEC estimation approach.

5.3 Comparisons with NFXP

We next compare NFXP with the MPEC approach in several dimensions. We first consider the general differences, and then discuss the differences between how Rust implemented NFXP and how we implemented the MPEC approach.

First, if solution time is large relative to likelihood evaluation time, the MPEC approach results in a far faster algorithm. We do not solve the dynamic programming problem for each θ guess. 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. Furthermore, this is repeated for nearby values of θ if finite difference methods are used to compute derivatives. In contrast, for each θ considered in MPEC, the Bellman equation is *evaluated* once, not *solved*. Value function iteration will evaluate the Bellman equation many times in NFXP each time a dynamic program is solved. Essentially, MPEC nearly eliminates the time spent on solving the dynamic programming problem.

Even if one had a closed-form solution for the dynamic programming problem, it is not clear that it would be a good idea to use it. Nonlinear elimination of variables reduces the number of unknowns but possibly at the cost of increasing the nonlinearity of the objective function. If the equilibrium equations are sparse, as is true with the Zurcher dynamic

programming problem and many other problems, then little time is saved by substituting out variables. Actually, it is often easier to solve large optimization problems! Substituting out m variables from n squeezes all the nonlinearities of those m dimensions into the remaining $n - m$ dimensions and can easily create a highly nonlinear objective. In optimization, it is nonlinearity, not dimensionality, that makes a problem difficult.

The second kind of advantage of MPEC is the far greater flexibility and ease in implementation. Software implementations are much easier since it makes derivatives much easier to compute, and allows for the direct application of methods that produce efficient analytic derivatives. This would be very difficult to do in NFXP, which replies on the inferior and inefficient finite difference derivatives. Also, the MPEC approach allows one to experiment with many solvers to find the best one.

5.4 Comparison with Rust's Implementation of NFXP

Some of the reason for our superior results is that we use different software than Rust does. Also, the running times Rust reports cannot be compared to ours since computer technology has significantly improved in the past years. However, some comparisons can be made that are independent of technology. We next run through the points made in Rust (1987) about the advantages of Gauss, and compare his Gauss implementation of NFXP with our AMPL implementation of MPEC.

Ease of use. Rust used Gauss "because: 1. the GAUSS language is a high-level symbolic language which enables a nearly 1:1 translation of mathematical formulae into computer code. Matrix operations of GAUSS replace cumbersome do-loops of FORTRAN. 2. GAUSS has built-in linear algebra routines, no links to Lapack needed"

MPEC: AMPL is just as easy to use. All solvers include linear algebra routines in Lapack. AMPL does not have matrix notation, but its summation notation for matrices and tensors is more flexible than a matrix language.

Vectorization. Rust: "Efficient use of GAUSS requires the user to "vectorize" a program in a manner very similar to efficient vectorization on large-scale vector supercomputers."

MPEC: All vectorization is done automatically in AMPL. The user just expresses the problem using a modeling language with a user-friendly syntax. No need to think about vectorization.

Optimization Method. Rust: Outer iteration uses BHHH for a while then switches to BFGS, where the user chooses the switch point.

MPEC: The algorithms implemented in the solvers of the past 30 years are far superior to these methods. They are faster, and do not require user intervention regarding any switching points.

Derivatives. Rust: "The NFXP software computes the likelihood and its derivatives numerically in a subroutine. This implies that we can numerically compute the likelihood and its derivatives for each trial value encountered in the course of the process of maximizing. In order to do this, we need a very efficient and accurate algorithm for computing the fixed point."

MPEC: We use true analytic derivatives. This is done automatically and efficiently by AMPL, using ideas from automatic differentiation.

Dynamic programming method. Rust: "Inner Fixed Point Algorithm. Contraction mapping fixed point (poly)algorithm. The algorithm combines contraction iterations with Newton-Kantorovich iterations to efficiently compute the functional fixed point." In Rust, contraction iterations are linearly convergent; quadratic convergence is achieved only at the final stage.

MPEC: We use only Newton-style methods. They are faster than contraction mapping methods. This is particularly important if β is close to 1, representing short, but realistic, time periods. Rust starts with the contraction iterations because of instabilities using Newton's method, but this problem arises only with the pure Newton method. However, Newton's method works far better if combined with either a trust region or linesearch method, as is the practice now in all good optimization software. Moreover, Newton's method combined with one of these strategies will always converge to the true solution of the Bellman equations.

The MPEC approach could be implemented in Gauss using the CML module. However, the CML module does numerical derivatives (unless the user provides derivatives) and uses only one algorithm. The big advantages of AMPL are its implementation of automatic differentiation and it giving us access almost all of the state-of-the art solvers.

Some of the features used in our implementation of the MPEC approach could be implemented in NFXP. For example, one could use the version of Newton's method used in our MPEC implementation to reliably and rapidly solve the dynamic programming problem solved at each stage in NFXP. However, no implementation of NFXP can avoid the problem of resolving the dynamic programming problem at each guess of θ. This feature of NFXP makes it particularly inappropriate for application to games, a topic to which we next turn.

6 The MPEC Approach to games

NFXP cannot be used to estimate data from games except for very special cases. Suppose that the game has parameters θ representing payoffs, probabilities, and whatever else is not observed directly by the econometrician. Let σ denote the equilibrium strategy given θ. Suppose that σ is an equilibrium if and only if

$$G(\theta, \sigma) = 0$$

for some function G.[10]

Suppose that the augmented likelihood function of a data set X is $\mathcal{L}(\theta, \sigma; X)$. Therefore, the maximum likelihood estimator is the solution to

$$\begin{array}{ll} \max\limits_{(\theta,\sigma)} & \mathcal{L}(\theta, \sigma; X) \\ \text{subject to} & G(\theta, \sigma) = 0. \end{array}$$

Let $\Sigma(\theta)$ be the set of all σ such that $G(\theta, \sigma) = 0$. For each θ, NFXP requires one to find all the $\sigma \in \Sigma(\theta)$ that solve $G(\theta, \sigma)$, compute the likelihood at each $\sigma \in \Sigma(\theta)$, and report the maximum. Finding all equilibria is an intractable problem unless one has special structure. Also, the resulting likelihood function will often be discontinuous if there are multiple equilibria, and possibly be nondifferentiable even at points of continuity. Both of these problems will create difficulties for the outer loop in NFXP.

Aguirregabiria and Mira (2007) propose an alternative approach to this problem using their pseudo maximum likelihood (PML) approach. However, the Gauss-Seidel nature of their iteration produces convergence problems. Therefore, their Monte Carlo examples only looked at equilibria that were stable under Best Reply (BR) iteration. Their use of a selection criterion limits the range of equilibria. In fact, there are trivial examples of games with a unique equilibrium but where BR iteration is unstable.

In contrast, for MPEC, the augmented likelihood function is a smooth function of the parameters θ and the strategies σ, and the smooth constraints define a set of points in (θ, σ)–space. The resulting formulation gives a smooth constrained optimization problem. To implement the MPEC approach for estimating games, the user could simply use state-of-the-art optimization solvers to solve the estimation problem. There is no need for assuming that the data comes from a particular kind of equilibrium. Multiplicity of equilibria will not create discontinuities or lack of differentiability in the MPEC approach. Multiple equilibria may produce multiple local solutions, but that is a standard problem in maximum likelihood estimation, and would also be a problem for the NFXP approach.

We are not saying that solving this problem is easy. Of course, it will be more difficult and costly as the size and complexity of the game increase. The key advantage is that this

[10] In more general settings, an equilibrium is defined by complementarity constraints.

approach is often just a conventional optimization problem with continuous objective and constraint functions. In contrast, the NFXP approach is impossible to implement for all but the simplest games.

6.1 An Example of Games with Multiple Equilibria

We illustrate these points with a simple example. Consider a Bertrand pricing game between two firms in several cities, where each city is one of four types. There are two products, x and y, two firms with firm x (y) producing good x (y), and three types of customers. We assume that the equilibrium is the same in each city of the same type. Let p_{xi} (p_{yi}) be the price of good x (y) in a type i city. We use Dx_j to denote the demands for product x by customer type $j(=1,2,3)$ and similarly for Dy_j. Type 1 customers only want good x, and have a linear demand curve:

$$Dx_1(p_{x,i}) = A - p_{x,i};\ Dy_1 = 0,\ \text{ for } i = 1,\ldots,4.$$

Type 3 customers only want good y, and have a linear demand curve:

$$Dx_3 = 0;\ Dy_3(p_{y,i}) = A - p_{y,i},\ \text{ for } i = 1,\ldots,4.$$

Type 2 customers want some of both goods. Let n_i be the number of type 2 customers in a type i city. We assume that the two goods are imperfect substitutes for type 2 customers with a constant elasticity of substitution between the two goods and a constant elasticity of demand for a composite good. This implies the demand functions

$$\begin{aligned} Dx_2(p_{xi},p_{yi}) &= n_i p_{xi}^{-\sigma}\left(p_{xi}^{1-\sigma}+p_{yi}^{1-\sigma}\right)^{\frac{\gamma-\sigma}{-1+\sigma}} \\ Dy_2(p_{xi},p_{yi}) &= n_i p_{yi}^{-\sigma}\left(p_{xi}^{1-\sigma}+p_{yi}^{1-\sigma}\right)^{\frac{\gamma-\sigma}{-1+\sigma}}, \end{aligned}$$

where σ is the elasticity of substitution between x and y, and γ is the elasticity of demand for a composite good. Total demand for good x (y) in a type i city is given by

$$\begin{aligned} Dx(p_{xi},p_{yi}) &= Dx_1(p_{xi},p_{yi}) + Dx_2(p_{xi},p_{yi}) \\ Dy(p_{xi},p_{yi}) &= Dy_2(p_{xi},p_{yi}) + Dy_3(p_{xi},p_{yi}), \end{aligned}$$

Let m be the unit cost of production for each firm. Revenue for good x in a type i city, $R_x(p_{xi},p_{yi})$, is $(p_{xi}-m)Dx(p_{xi},p_{yi})$; R_y is similarly defined. Let MR_x be marginal profits

for good x; similarly for MR_y. The equations for MR_x and MR_y are

$$\begin{aligned} MR_x(p_{xi}, p_{yi}) &= A - p_{xi} + n_i \left(p_{xi}^{\sigma} \left(p_{xi}^{1-\sigma} + p_{yi}^{1-\sigma} \right)^{\frac{\gamma-\sigma}{\sigma-1}} \right)^{-1} \\ &\quad + (p_{xi} - m) \left(-1 + \frac{n_i(\sigma-\gamma)}{p_{xi}^{2\sigma} \left(p_{xi}^{1-\sigma} + p_{yi}^{1-\sigma} \right)^{1+\frac{\sigma-\gamma}{\sigma-1}}} - \frac{n_i \sigma}{p_{xi}^{1+\sigma} \left(p_{xi}^{1-\sigma} + p_{yi}^{1-\sigma} \right)^{\frac{\sigma-\gamma}{\sigma-1}}} \right) \\ MR_y(p_{xi}, p_{yi}) &= A - p_{yi} + n_i \left(p_{yi}^{\sigma} \left(p_{xi}^{1-\sigma} + p_{yi}^{1-\sigma} \right)^{\frac{\gamma-\sigma}{\sigma-1}} \right) \\ &\quad + (p_{yi} - m) \left(-1 + \frac{n_i(\sigma-\gamma)}{p_{yi}^{2\sigma} \left(p_{xi}^{1-\sigma} + p_{yi}^{1-\sigma} \right)^{1+\frac{\sigma-\gamma}{\sigma-1}}} - \frac{n_i \sigma}{p_{yi}^{1+\sigma} \left(p_{xi}^{1-\sigma} + p_{yi}^{1-\sigma} \right)^{\frac{\sigma-\gamma}{\sigma-1}}} \right). \end{aligned} \tag{17}$$

Equilibrium prices satisfy the system

$$\left. \begin{array}{l} MR_x(p_{xi}, p_{yi}) = 0 \\ MR_y(p_{xi}, p_{yi}) = 0 \end{array} \right\} \text{ for } i = 1, \ldots, 4. \tag{18}$$

Our example will assume that the four markets differ only in terms of population, n_i. The other parameters are common across markets; we assume they are

$$\sigma = 3;\ \gamma = 2;\ m = 1;\ A = 50.$$

Given these parameter values, the marginal revenue functions are

$$\begin{aligned} MR_x(p_{xi}, p_{yi}) &= 50 - p_{xi} + (p_{xi} - 1) \left(-1 + n_i p_{xi}^{-6} P_i^{-3} - 3 n_i p_{xi}^{-3} P_i^{-1} \right) + n_i p_{xi}^{-3} P_i^{-1} \\ MR_y(p_{xi}, p_{yi}) &= 50 - p_{yi} + (p_{yi} - 1) \left(-1 + n_i p_{yi}^{-6} P_i^{-3} - 3 n_i p_{yi}^{-4} P_i^{-1} \right) + n_i p_{yi}^{-3} P_i^{-1} \\ P_i &= \sqrt{p_{xi}^{-2} + p_{yi}^{-2}}. \end{aligned}$$

The key economic fact in our example is that each firm has two kinds of strategies it can follow. The *niche strategy* is to charge a high price and sell mainly to the relatively small number of customers that want only its good. The *mass market strategy* chooses a low price so that many of the type 2 customers will buy its product as well as the competing firm's product. The mass market customers will buy some of each, but the composition of their purchases is significantly price sensitive. Each firm has to choose between a high margin and low sales, or a low margin and high sales.

We choose the type 2 customer population in the four cities to be $(n_1, n_2, n_3, n_4) = (1500, 2500, 3000, 4000)$. For each city, we solve for the solutions to the above first-order conditions and check their second-order conditions. We also check global optimality for each firm in each potential equilibrium. After performing all the necessary checks, we find that the equilibrium is unique for type 1 and 4 cities:

$$\begin{aligned} \text{city 1:} \quad (p_{x1}, p_{y1}) &= (24.24, 24.24) \\ \text{city 4:} \quad (p_{x4}, p_{y4}) &= (1.71, 1.71)\,. \end{aligned}$$

The unique equilibrium in type 1 cities is for both firms to choose a niche strategy, whereas the unique equilibrium in type 4 cities is for both to pursue a mass market strategy. This is all very intuitive since the mass market is small in type 1 cities but large in type 4 cities.

The situation is more complex for type 2 and type 3 cities. The same procedure showed that there are two equilibria in type 2 cities and type 3 cities. In these equilibria, one firm chooses a niche strategy while the other chooses a mass market strategy. More precisely, the equilibria are:

$$\begin{array}{lll} \text{city 2:} & \left(p_{x2}^{I}, p_{y2}^{I}\right) & = \ (25.18,\ 2.19) \\ & \left(p_{x2}^{II}, p_{y2}^{II}\right) & = \ (2.19,\ 25.18) \\ & & \\ \text{city 3:} & \left(p_{x3}^{I}, p_{y3}^{I}\right) & = \ (2.15,\ 25.12) \\ & \left(p_{x3}^{II}, p_{y3}^{II}\right) & = \ (25.12,\ 2.15)\,. \end{array}$$

Essentially, the mass markets in type 2 and type 3 cities are large enough to attract one firm, but not large enough to attract the other firm.

We assume the econometrician observes price data for $4K$ cities, with K cities of each type; the data is denoted by $P = (p_{x,i}^k, p_{y,i}^k)_{i=1,2,3,4}^{k=1,..,K}$. We also assume that the same equilibrium is played in each city of a particular type; that is, each price $p_{x,i}^k$ in type k markets equals some common equilibrium price, $p_{x,i}$, plus some market-specific error, $\varepsilon_{x,i}^k$, and similarly for good y. Suppose that we want to estimate the unknown structural parameters (σ, γ, m, A) as well as equilibrium prices implied by the data in all four cities. To make the problem nonsingular, we assume that each observed price, $p_{y,i}^k$, is contains measurement error; hence,

$$p_{z,i}^k = p_{z,i} + \varepsilon_{z,i}^k,\ z = x, y;\ i = 1, 2, 3, 4;\ k = 1, ..., K;\ \varepsilon_{z,i}^k \sim N\left(0, 50\right).$$

The augmented (log) likelihood function, $\mathcal{L}(p_x, p_y, \sigma, \gamma, m, A, P)$ is

$$\mathcal{L}(p_x, p_y, \sigma, \gamma, m, A, P) = -\sum_{k=1}^{K}\sum_{i=1}^{4}\left((p_{xi}^k - p_{xi})^2 + (p_{yi}^k - p_{yi})^2\right).$$

We want to maximize the likelihood function subject to the requirement that our estimates of the type-specific equilibrium prices, $(p_{xi}, p_{yi})_{i=1}^4$, and our estimates of the structural parameters, (σ, γ, m, A), satisfy the first-order conditions (18). Because the firms are not solving concave maximization problems, the first-order conditions (18) are not sufficient for characterizing a global solution. Consequently, we also need to worry about global optimality. We suggest to impose the global optimality constraints for each firm's problem:

$$\begin{array}{ll} \forall\ p \geq 0: & (p_{xi} - m)Dx(p_{xi}, p_{yi}) \geq (p - m)Dx(p_j, p_{yi}), \quad i = 1, \ldots, 4 \\ \forall\ p \geq 0: & (p_{yi} - m)Dy(p_{xi}, p_{yi}) \geq (p - m)Dy(p_{xi}, p_j), \quad i = 1, \ldots, 4 \end{array}$$

The constrained optimization formulation for this price game is

$$\begin{array}{lll}
\min\limits_{(p_{xi},p_{yi},\sigma,\gamma,m,A)} & \sum_{t=1}^{K}\sum_{i=1}^{4}\left((p_{xi}^{k}-p_{xi})^2+(p_{yi}^{k}-p_{yi})^2\right) & \\
\text{subject to} & 0=MR_y(p_{xi},p_{yi})=MR_x(p_{xi},p_{yi}), & i=1,\ldots,4 \\
 & p_{xi}\geq 0,\ p_{yi}\geq 0, & i=1,\ldots,4 \\
\forall\ p\geq 0: & (p_{xi}-m)Dx(p_{xi},p_{yi})\geq (p-m)Dx(p_j,p_{yi}), & i=1,\ldots,4 \\
\forall\ p\geq 0: & (p_{yi}-m)Dy(p_{xi},p_{yi})\geq (p-m)Dy(p_{xi},p_j), & i=1,\ldots,4
\end{array}$$

This is an example of a semi-infinite programming problem. We do not want to invoke the complexities of that approach here. Therefore, we will adopt the sampling approach to solving semi-infinite problems, and just require that the equilibrium prices (p_{xi}, p_{yi}) satisfy a finite number of global optimality conditions on a pre-specified grid of J alternative prices, $\{p_1,\ldots,p_J\}=\{1,2,...,30\}$:

$$\left.\begin{array}{lcl}
(p_{xi}-m)Dx(p_{xi},p_{yi}) & \geq & (p_j-m)Dx(p_j,p_{yi}) \\
(p_{yi}-m)Dy(p_{xi},p_{yi}) & \geq & (p_j-m)Dy(p_{xi},p_j)
\end{array}\right\}\ \text{for } i=1,\ldots,4,\ j=1,\ldots,J.$$

The constrained optimization formulation for this price game now becomes

$$\begin{array}{lll}
\min\limits_{(p_{xi},p_{yi},\sigma,\gamma,m,A)} & \sum_{t=1}^{K}\sum_{i=1}^{4}\left((p_{xi}^{k}-p_{xi})^2+(p_{yi}^{k}-p_{yi})^2\right) & \\
\text{subject to} & 0=MR_y(p_{xi},p_{yi})=MR_x(p_{xi},p_{yi}), & i=1,\ldots,4 \\
 & p_{xi}\geq 0,\ p_{yi}\geq 0, & i=1,\ldots,4 \\
\text{for } j=1,\ldots,J: & (p_{xi}-m)Dx(p_{xi},p_{yi})\geq (p_j-m)Dx(p_j,p_{yi}), & i=1,\ldots,4 \\
\text{for } j=1,\ldots,J: & (p_{yi}-m)Dy(p_{xi},p_{yi})\geq (p_j-m)Dy(p_{xi},p_j), & i=1,\ldots,4.
\end{array} \qquad (19)$$

For the purpose of illustration, we assume that in this example the true parameter values are $(\sigma,\gamma,m,A)=(3,2,1,50)$, and that the city types are $(n_1,n_2,n_3,n_4)=(1500,2500,3000,4000)$. We assume that the equilibria in the four city types are

$$\begin{array}{rl}
(p_{x1},p_{y1})= & (24.24,24.24) \\
(p_{x2},p_{y2})= & (25.18,\ 2.19) \\
(p_{x3},p_{y3})= & (2.15,\ 25.12) \\
(p_{x4},p_{y4})= & (1.71,1.71)\,.
\end{array}$$

We used a normally distributed measurement error $\varepsilon\sim N(0,50)$ to simulate price data for 40,000 cities, with 10,000 cities of each type ($K=10{,}000$).

We studied three different cases by varying the number of structural parameters to be estimated. We coded the constrained optimization problem in AMPL and submitted it to the KNITRO (interior point option) solver via NEOS. We chose KNITRO to show that interior point methods could also be used for the constrained optimization approach to

estimation.[11] In Case 1, we estimated only two parameters, σ and γ and fixed $A_x = A_y = 50$ and $m_x = m_y = 1$ at the true parameter values. In Case 2, we estimated all six structural parameters but we also imposed the symmetry constraints on the two firms: $A_x = A_y$ and $m_x = m_y$. In Case 3, we estimated all six structural parameters without imposing the symmetry constraints. Since the estimation problem for this price game is a nonconvex program, there might exist multiple local solution. We performed a multi-start procedure with 25 different starting points and chose the solution taking the lowest object value as our final solution.

The estimated structural parameters and the implied equilibrium prices are reported in Table 4. The results clearly show that in each of the three cases, the estimated parameters are very accurate compared to the true values. Although there are multiple equilibrium prices in this pricing game, the implied equilibrium prices from our solutions correctly identify the equilibrium strategy played by the two firms in the data generating process. The implied equilibrium prices are also very close to the true equilibrium prices. The computational burden was low; most problems were solved in seconds with an average time of around 3 seconds. Except for a few instances, most problems only require around 50 to 60 major iterations and on average, 85 function evaluations. For Case 1, for 14 out of 25 different starting points, the Knitro solver converged to the global solution. For Case 2 (Case 3), for 20 (21) out of 25 starting points, Knitro converged to the global solution. This observation shows that solving a estimation problem with few structural parameters is not necessarily easier than that with more parameters.

7 The MPEC approach and the Method of Moments

There are many estimation methods that employ iterative methods of essentially a Gauss-Jacobi and Gauss-Seidel fashion to compute estimates. In general, computational efficiency could be substantially improved by pursuing a constrained optimization approach utilizing far more sophisticated methods. Moment-based estimators are examples where this approach could be applied. We demonstrate the advantage of the MPEC approach for a method of moments estimator using the bus example above.

Suppose that θ are unknown structural parameters, σ are decision rules and other endogenous variables, that $G(\theta, \sigma) = 0$ expresses equilibrium conditions, and that $M(\theta, \sigma)$ is a vector of moments implied by endogenous variables σ and parameters θ. As with the augmented likelihood function, we do not assume that σ and θ together satisfy the equilibrium conditions when we write $M(\theta, \sigma)$. Let $M(X)$ be the corresponding data moments. Then

[11] There are several more solvers that could be applied. In general, our impression is that SQP solvers such as SNOPT, Filter, KNITRO-SQP, and interior point solvers such as KNITRO-IP and IPOPT, will be reliable.

the moment estimator can be found by solving

$$\begin{array}{ll} \min\limits_{(\theta,\sigma)} & \|M(\theta,\sigma) - M(X)\|^2 \\ \text{subject to} & G(\theta,\sigma) = 0. \end{array} \tag{20}$$

Moment estimators are useful even in contexts where we want to use maximum likelihood estimation. For example, the range of values for θ for which the log likelihood function is defined could be fairly narrow and difficult to find, making maximum likelihood estimation impractical. Moment-based estimation is more robust and not subject to this difficulty.

We apply the method of moments to the bus example in Section 5.1. For notational simplicity, we only consider the case with one bus. We first compute various moments of the data. Let $(x_t, d_t)_{t=1}^T$ be the vector of mileage and decision. In our example, we compute the data moment vector

$$M = (M_x, M_d, M_{xx}, M_{xd}, M_{dd}, M_{xxx}, M_{xxd}, M_{xdd}, M_{ddd}),$$

where M_x is the mean of x, M_d is the mean of d, M_{xx} is the variance of x, M_{xd} is the covariance of x and d, M_{xxx} is the skewness of x, etc. Let

$$m = (m_x, m_d, m_{xx}, m_{xd}, m_{dd}, m_{xxx}, m_{xxd}, m_{xdd}, m_{ddd})$$

denote the vector of the corresponding moments implied by the model.

We next need to compute the moments predicted by the structural parameters θ. Let Π denote the transition matrix from state (x, d) to (x', d'). Π will depend on the expected value function EV and structural parameters θ. Let $\Pi = H(\theta, EV)$ denote that functional relation. This is an appropriate expression for the relation between Π and (θ, EV) since H is a direct computation once we fix θ and EV. Let $p = (p_{x,d})_{x\in Z, d\in\{0,1\}}$ denote the stationary distribution (which will be unique since all ergodic states communicate). Therefore, p and Π are the solutions to

$$\begin{aligned} &\Pi = H(\theta, EV) \\ &p^\top \Pi = p^\top, \ \sum_{x,d} p_{x,d} = 1. \end{aligned}$$

Once we have p, we can them compute the moments. For example,

$$\begin{aligned} m_x &= \sum_{x,d} p_{x,d}\, x, \ \ m_d = \sum_{x,d} p_{x,d}\, d \\ m_{xx} &= \sum_{x,d} p_{x,d}\, (x - m_x)^2, \ \ m_{xd} = \sum_{x,d} p_{x,d}\, (x - m_x)(d - m_d), \end{aligned}$$

and similarly for the other moments.

We are now ready to express the MPEC approach to moments estimation for the bus model. The objective function is the weighted squared differences between the data moments and the model moments

$$\begin{aligned}\mathcal{M}(m,M) &= (m_x - M_x)^2 + (m_d - M_d)^2 + (m_{xx} - M_{xx})^2 + (m_{xd} - M_{xd})^2 \\ &+ (m_{dd} - M_{dd})^2 + (m_{xxx} - M_{xxx})^2 + (m_{xxd} - M_{xxd})^2 \\ &+ (m_{xdd} - M_{xdd})^2 + (m_{ddd} - M_{ddd})^2 .\end{aligned}$$

When we combine the objective and constraints, the optimization problem is

$$\begin{aligned}\max_{(\theta,EV,\Pi,p,m)} \quad & \mathcal{M}(m,M) \\ \text{subject to} \quad & G(\theta,EV) = 0 \\ & \Pi = H(\theta,EV) \\ & p^\top \Pi = p^\top, \ \sum_{x,d} p_{x,d} = 1 \\ & m_x = \sum_{x,d} p_{x,d}\, x, \ m_d = \sum_{x,d} p_{x,d}\, d \\ & m_{xx} = \sum_{x,d} p_{x,d}\,(x - m_x)^2, \ m_{xd} = \sum_{x,d} p_{x,d}\,(x - m_x)(d - m_d) \\ & m_{dd} = \sum_{x,d} p_{x,d}\,(d - m_d)^2 \\ & m_{xxx} = \sum_{x,d} p_{x,d}\,(x - m_x)^3, \ m_{xxd} = \sum_{x,d} p_{x,d}\,(x - m_x)^2(d - m_d) \\ & m_{xdd} = \sum_{x,d} p_{x,d}\,(x - m_x)(d - m_d)^2, \ m_{ddd} = \sum_{x,d} p_{x,d}\,(d - m_d)^3\end{aligned} \tag{21}$$

We apply this formulation to the same 20 data sets used in Table 3. The constrained method of moments problem is coded in AMPL, and submitted to the SNOPT solver via NEOS. Table 5 displays the results. Again we see that the running time is fast, although it is not as fast as maximum likelihood. That is not surprising given the larger size of the moments problem and the nonlinearity introduced by the constraints related to moments, particularly the skewness equations. We made no attempt to insure that the problem was properly scaled; poor scaling is almost automatically assured by the fact that some constraints use cubic powers of the state variable x and some just use x.

Our example is a primitive approach to the method of moments. A more sophisticated implementation would repeat this with a weighting matrix implied by the initial solutions. Larger problems would have larger Markov transition matrices, but if we knew the sparseness structure then we could exploit it to efficiently represent the constraints that define the Markov transition matrix.

Simulated method of moments is often used in practice when it is difficult to compute the exact moments predicted by a parameterization of the model. To compute simulated moments, one could just define a procedure (subroutine) which takes a parameter vector, decision rules, and a fixed sequence of random draws and computes the implied moments.

Despite the simplicity of this example, it does show that moment estimators can be implemented within the MPEC framework.

8 Computational Notes

The examples we have used to illustrate the key idea are all infinitesimal relative to the optimization problems that are commonly solved. The computers we used were all desktop computers with a single processor. These problems can easily take advantage of the multiple processor environments used in high-power and high-throughput computing environments. The software we used is aimed at solving problems of moderate to large size. There are other methods that can be used to solve huge problems. Therefore, the small size and limited nature of our examples cannot be read as indications of size limits to the numerical optimization approach to solving structural estimation problems.

Our dynamic programming example discretized the state space. Such discretizations are impractical in many interesting problems. It is preferable to approximate value functions with linear combinations of basis functions, as is done in Judd (1992). In that case, the constraints become the projection conditions that solve the dynamic problem. The mathematical programming methods then become similar to a variety of problems currently solved in the engineering and science literatures, such as PDE–constrained optimization.

9 Conclusion

Maximum likelihood estimation, as well as other methods like the methods of moments, indirect inference, partial identification, and calibration are often just constrained optimization problems. We argue that nonlinear programming approaches can be applied directly to those problems, contradicting the pervasive pessimism in the econometrics literature. Our examples show that this theoretical superiority appears in many familiar econometric contexts. This is valuable for estimation purposes since many of the "computationally light" alternatives, such as two-step methods, are not asymptotically efficient. The fact that we used numerical software that can be easily obtained and commonly used in other fields and most basic hardware available today, clearly demonstrates that the MPEC approach will be very useful in many contexts.

References

[1] Agguiregabiria, V. and P. Mira (2007): Sequential estimation of dynamic discrete games, *Econometrica*, 75, 1–53.

[2] Aitchison, J. and S. D. Silvey (1958): Maximum likelihood estimation of parameters subject to restraints. *Annals of Mathematical Statistics*, 29, 813–828.

[3] Berz, M., C. Bischof, G. F. Corliss, and A. Griewank, eds. (1996): *Computational Differentiation: Techniques, Applications, and Tools, Proceedings of the SIAM Computational Differentiation Workshop, Santa Fe, New Mexico, 1996*, SIAM, Philadelphia.

[4] Burguete, J. F., A. R. Gallant, and G. Souza (1982): On unification of the asymptotic theory of nonlinear econometric models. *Econometric Reviews*, 1, 151–190.

[5] Byrd, R. H., J. Nocedal, and R. A. Waltz (2006): KNITRO: An integrated package for nonlinear optimization, in G. di Pillo and M. Roma, eds., *Large-Scale Nonlinear Optimization*, Springer Verlag.

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

[7] Dolan, E. D., R. Fourer, J. Moré, and T. S. Munson (2002): Optimization on the NEOS Server, *SIAM News*, 35, 8–9.

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

[9] Erdem, T., K. Srinivasan, W. Amaldoss, P. Bajari, H. Che, T. Ho, W. Hutchinson, M. Katz, M. Keane, R. Meyer, and P. Reiss (2005): Theory-driven choice models. *Marketing Letters*, 16, 225–237.

[10] Facchinei, F. and J.-S. Pang (2003): *Finite-Dimensional Variational Inequalities and Complementarity Problems*, Springer Verlag, New York.

[11] Ferreyra, M. M. (2007): Estimating the effects of private school vouchers in multidistrict economies, *American Economic Review*, 97, 789–817.

[12] Ferris, M. C. and F. Tin-Loi (2001): Limit analysis of frictional block assemblies as a mathematical program with complementarity constraints, *International Journal of Mechanical Sciences*, 43, 209–224.

[13] 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.

[14] Griewank, A. and G. F. Corliss, eds. (1991): *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, 157–165, SIAM, Philadelphia.

[15] Gallant, A. R. and A. Holly (1980): Stastical inference in an implicit, nonlinear, simultaneous equation model in the context of maximum likelihood estimation. *Econometrica*, 48, 697–420.

[16] Gallant, A. R. and G. Tauchen (1989): Seminonparametric estimation of conditionally constrained heterogeneous processes: asset pricing applications. *Econometrica*, 57, 1091–1120.

[17] Gill, P. E., W. Murray and M. A. Saunders (2005): SNOPT: An SQP algorithm for large-scale constrained optimization, *SIAM Review*, 47, 99–131.

[18] Gropp, W. and J. Moré (1997): Optimization environments and the NEOS Server, in M. D. Buhmann and A. Iserles, eds., *Approximation Theory and Optimization*, 167–182, Cambridge University Press.

[19] Judd, K. L. (1992): Projection methods for solving aggregate growth models. *Journal of Economic Theory*, 58, 410–452.

[20] Judd, K. and K. Schmedders (2006): A computational approach to proving uniqueness in dynamic games, Working paper, Kellogg School of Management, Northwestern University.

[21] Kalaba, R., L. Tesfatsion, and J.-L. Wang (1983): A finite algorithm for the exact evaluation of higher order partial derivatives of functions of many variables, *Journal of Mathematical Analysis and Applications*, 92, 552–563.

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

[23] Murtagh, B. A. and M. A. Saunders (1977): *MINOS User's Guide*, Report SOL 77-9, Dept of Operations Research, Stanford University, 127 pp.

[24] Murtagh, B. A. and M. A. Saunders (1982): A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints, *Mathematical Programming Study*, 16 (Constrained Optimization), 84–117.

[25] Murtagh, B. A. and M. A. Saunders (1983): *MINOS 5.0 User's Guide*, Report SOL 83-20, Dept of Operations Research, Stanford University, 118 pp.

[26] Pang, J. S. and F. Tin-Loi (2001): A penalty interior point algorithm for an inverse parameter identification problem in elastoplasticity, *Mechanics of Structures and Machines*, 29, 85–99.

[27] Rust, J. (1987): Optimal replacement of GMC bus engines: an empirical model of Harold Zurcher. *Econometrica*, 55, 999–1033.

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

[29] Silvey, S. D. (1970): *Statistical Inference*, Chapman & Hall, London.

[30] Tesfatsion, L. (1991): Automatic evaluation of higher-order partial derivatives for non-local sensitivity analysis, in A. Griewank and G. Corliss, eds., *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, 157–165, SIAM, Philadelphia.

[31] 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.

[32] Wolak, F. A. (1989): Testing inequality constraints in linear econometric models. *Journal of Econometrics*, 41, 205–235.

Table 1: Three-Parameter Estimates

T	N	Estimates RC	θ_1^c	θ_2^c	CPU (sec)	Major Iterations	Evals*	Bellman error
10^3	101	1.112	0.043	0.0029	0.14	66	72	3.0E−13
10^3	201	1.140	0.055	0.0015	0.31	44	59	2.9E−13
10^3	501	1.130	0.050	0.0019	1.65	58	68	1.4E−12
10^3	1001	1.144	0.056	0.0013	5.54	58	94	2.5E−13
10^4	101	1.236	0.056	0.0015	0.24	59	67	2.9E−13
10^4	201	1.257	0.060	0.0010	0.44	59	67	1.8E−12
10^4	501	1.252	0.058	0.0012	0.88	35	45	2.9E−13
10^4	1001	1.256	0.060	0.0010	1.26	39	52	3.0E−13

*Number of function and constraint evaluations

Table 2: Five-Parameter Estimates

T	N	Estimates RC	θ_1^c	θ_2^c	θ_1^p	θ_2^p	CPU (sec)	Major Iterations	Evals	Bellman error
10^3	101	1.107	0.039	0.0030	0.723	0.262	0.50	111	137	6.1E−12
10^3	201	1.140	0.055	0.0015	0.364	0.600	1.14	109	120	1.3E−09
10^3	501	1.129	0.050	0.0019	0.339	0.612	3.39	115	127	2.5E−11
10^3	1001	1.144	0.056	0.0014	0.360	0.608	7.56	84	116	5.1E−12
10^4	101	1.236	0.052	0.0016	0.694	0.284	0.50	76	91	5.3E−11
10^4	201	1.257	0.060	0.0010	0.367	0.053	0.86	85	97	3.6E−11
10^4	501	1.252	0.058	0.0012	0.349	0.596	2.73	83	98	2.9E−10
10^4	1001	1.256	0.060	0.0010	0.370	0.586	19.12	166	182	3.2E−10

Table 3: Maximum Likelihood Parametric Bootstrap Results

	Estimates						CPU	Major	Evals	Bell. EQ
	RC	θ_1^c	θ_2^c	θ_1^p	θ_2^p	θ_3^p	(sec)	Iter		Error
Mean	1.14	0.037	0.004	0.384	0.587	0.029	0.54	90	109	8.2E−09
S.E.	0.15	0.035	0.004	0.013	0.012	0.005	0.16	24	37	1.7E−08
Min	0.95	0.000	0.000	0.355	0.571	0.021	0.24	45	59	1.2E−13
Max	1.46	0.108	0.012	0.403	0.606	0.039	0.88	152	230	5.7E−08

Table 4: Game Estimation Results

Parameters	Case 1	Case 2	Case 3
(σ, γ)	$(3.008, 2.016)$	$(2.822, 1.987)$	$(3.080, 2.093)$
(A_x, A_y)		$(50.404, 50.404)$	$(50.236, 49.544)$
(m_x, m_y)		$(0.975, 0.975)$	$(1.084, 0.972)$
(p_{x1}, p_{y1})	$(24.292, 24.292)$	$(24.443, 24.443)$	$(24.694, 24.241)$
(p_{x2}, p_{y2})	$(25.192, 2.166)$	$(25.248, 2.139)$	$(25.434, 2.004)$
(p_{x3}, p_{y3})	$(2.127, 25.135)$	$(2.100, 25.163)$	$(2.243, 24.934)$
(p_{x4}, p_{y4})	$(1.718, 1.718)$	$(1.730, 1.730)$	$(1.814, 1.652)$

Table 5: Method of Moments Parametric Bootstrap Results

	Estimates						CPU	Major	Evals	Bell. EQ
	RC	θ_1^c	θ_2^c	θ_1^p	θ_2^p	θ_3^p	(sec)	Iter		Error
Mean	1.03	0.048	0.001	0.397	0.603	0.000	22.6	525	1753	6.7E−06
S.E.	0.31	0.032	0.002	0.040	0.040	0.001	16.9	389	1513	1.1E−05
Min	0.13	0.000	0.000	0.340	0.511	0.000	5.4	168	389	1.8E−10
Max	1.45	0.104	0.009	0.489	0.660	0.004	70.1	1823	6851	3.6E−05

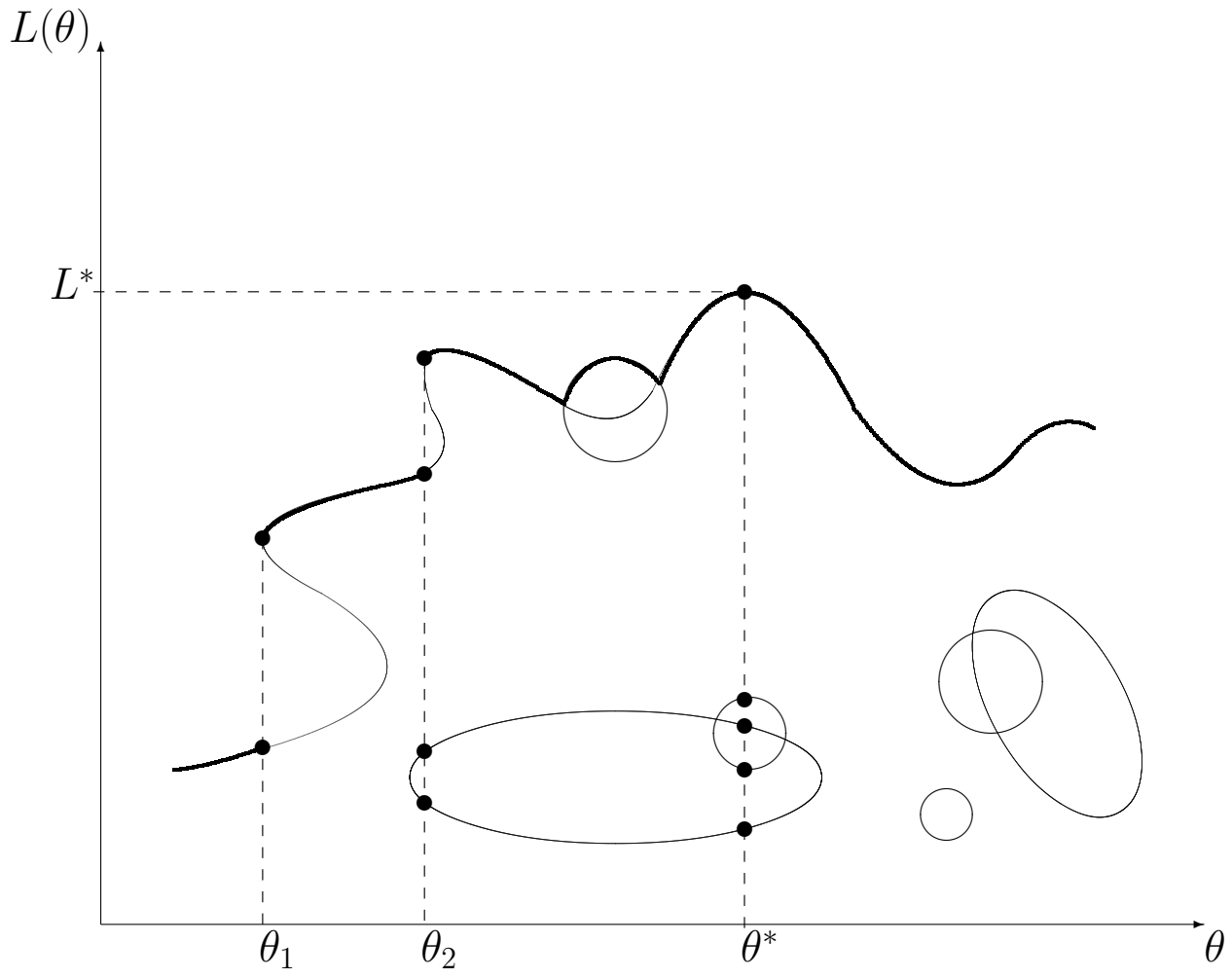

Figure 1: NFXP applied to games with multiple equilibria.

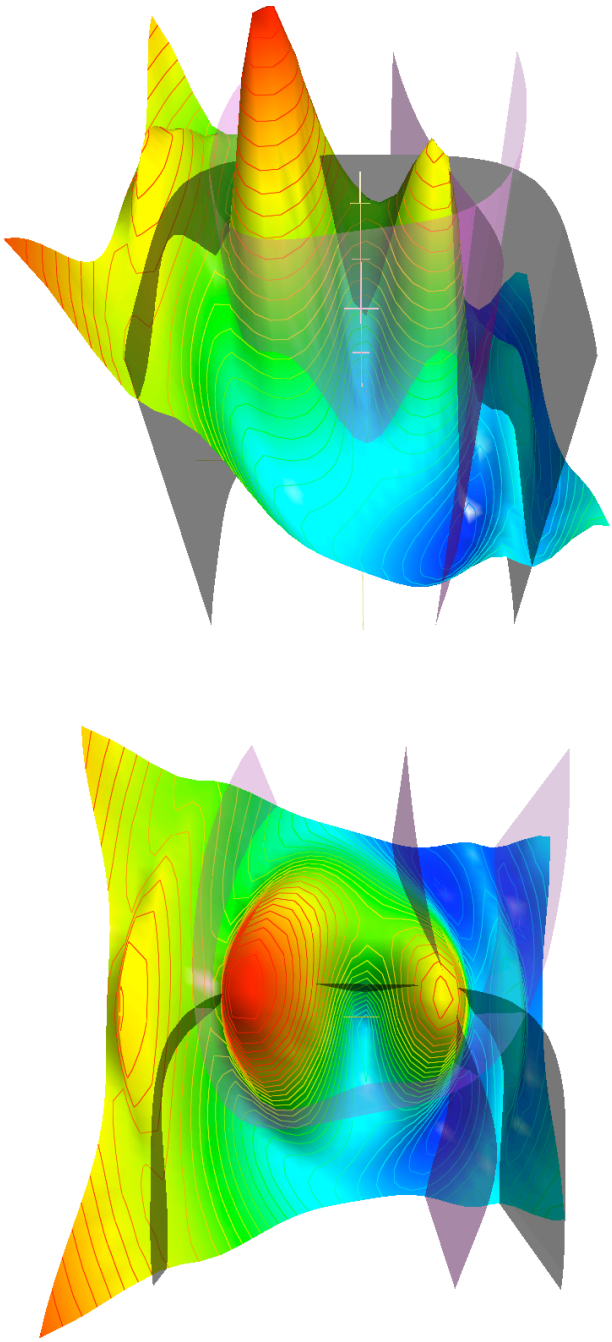

Figure 2: MPEC applied to games with multiple equilibria.
The bottom figure is the top-down view of the top figure.

Appledix: AMPL Code for MLE on Zucher's Model

We use the following AMPL model and command files to solve examples in this paper. We do not intend to include the AMPL code in the published version of this paper. Instead, we have made all the AMPL code and sample data sets used in this paper available for readers on the first author's webpage.

AMPL Model File: `GMCBusExampleMLE.mod`

```
# Title: Constrained Optimization Approaches for Estimation of Structural Models
# Authors: Che-Lin Su (Chicago GSB) and Kenneth L. Judd (Hoover Institution and NBER)
# November, 2006
# AMPL model file: GMCBusExampleMLE.txt

#####  HAROLD ZURCHER BUS REPAIR EXAMPLE  #####

# A constrained optimization formulation to compute maximum likelihood estimates
# of the Harold Zurcher bus problem in Rust (Econometrica, 1987).

#####  SET UP THE PROBLEM  #####

#  Define the state space used in the dynamic programming part
param    N;  # number of states used in dynamic programming approximation
set X := 1..N;  # X is the index set of states
#  x[i] denotes state i; the set of states is a uniform grid on the interval [xmin, xmax]
param xmin := 0;
param xmax := 100;
param x {i in X} := xmin + (xmax-xmin)/(N-1)*(i-1);

#  Define and process the data
param nT; # number of periods in data
set T := 1..nT; # T is the vector of time indices
param Xt {T};  # Xt[t] is the true mileage at time t
param dt {T}; # decision at time t
#  The dynamic programming model in the estimation lives on a discrete state
#  Binning process: assign true mileage Xt[t] to the closest state in X
param xt {t in T} := ceil(Xt[t]/(xmax-xmin)*(N-1)+0.5);

#  Define "known" structural parameters
#  We fix beta since data cannot identify it
param beta;       # discount factor

#####  DECLARE STRUCTURAL PARAMETERS TO BE ESTIMATED #####
#####  Parameters for cost function  #####
#  c(x, thetaCost) = thetaCost[1]*x + thetaCost[2]*x^2
var thetaCost {1..2} >= 0;

#####  Parameters and definition of transition process  #####
#  thetaProbs defines Markov chain
var thetaProbs {1..3} >= 0.00001;
```

```
#  Define the Markov chain representing the changes in mileage on the x[i] grid.

#  The state increases by some amount in [0,JumpMax] where
#  JumpMax is the maximum increase in mileage in one period and equals
#  a fraction JumpRatio of the range of mileage in the state space
param JumpRatio;
param JumpMax := (xmax-xmin) * JumpRatio;

#  We assume that the jumps are independent of the current state and its
#  transition process has three pieces, each a uniform distribution
#  Define 1st break point for stepwise uniform distribution in mileage increase
param M1 := ceil(1/4*JumpMax/(xmax-xmin)*(N-1)+0.5);
#  Define 2nd break point for stepwise uniform distribution in mileage increase
param M2 := ceil(3/4*JumpMax/(xmax-xmin)*(N-1)+0.5);
#  Define end point for stepwise uniform distribution in mileage increase
param M := ceil(JumpMax/(xmax-xmin)*(N-1)+0.5);

#  Y is the vector of elements in transition rule
set Y := 1..M;
var TransProb {i in Y} =
if  i <= M1  then  thetaProbs[1]/M1
else if  i > M1 and i <= M2  then  thetaProbs[2]/(M2-M1)
else  thetaProbs[3]/(M-M2);

#####  Scrap value parameter  #####
var RC >= 0;
##### END OF STRUCTURAL VARIABLES  #####

#####  DECLARE EQUILIBRIUM CONSTRAINT VARIABLES #####
# The NLP approach requires us to solve equilibrium constraint variables
var EV {X};         # Value Function of each state
#####  END OF EQUILIBRIUM CONSTRAINT VARIABLES  #####

#####  DECLARE AUXILIARY VARIABLES  ######
#  Define auxiliary variables to economize on expressions
#  Create Cost variable to represent the cost function;
#  Cost[i] is the cost of regular maintenance at x[i].
var Cost {i in X} = sum {j in 1..2} thetaCost[j]*x[i]^(j);
#  Let CbEV[i] represent - Cost[i] + beta*EV[i];
#  this is the expected payoff at x[i] if regular maintenance is chosen
var CbEV {i in X} = - Cost[i] + beta*EV[i];
#  Let PayoffDiff[i] represent -CbEV[i] - RC + CbEV[1];
#  this is the difference in expected payoff at x[i] between engine replacement and
#  regular maintenance
var PayoffDiff {i in X} = -CbEV[i] - RC + CbEV[1];
#  Let ProbRegMaint[i] represent 1/(1+exp(PayoffDiff[i]));
#  this is the probability of performing regular maintenance at state x[i];
var ProbRegMaint {i in X} = 1/(1+exp(PayoffDiff[i]));
var BellmanViola {i in 1..(N-M+1)} = sum {j in 0..(M-1)}
                log(exp(CbEV[i+j])+ exp(-RC + CbEV[1]))* TransProb[j+1] - EV[i];

#####  END OF AUXILIARY VARIABLES  #####
```

```
#####  OBJECTIVE AND CONSTRAINT DEFINITIONS  #####
## Define objective function:  Likelihood function
maximize Likelihood:
## #  The likelihood function contains two pieces
## #  First is the likelihood that the engine is replaced given time t state in the data.
sum {t in 2..nT} log(dt[t]*(1-ProbRegMaint[xt[t]]) + (1-dt[t])*ProbRegMaint[xt[t]])
## #  Second is the likelihood that the observed transition between t-1 and t would have occurred.
+ sum {t in 2..nT} log(dt[t-1]*(TransProb[xt[t]-1+1]) + (1-dt[t-1])*(TransProb[xt[t]-xt[t-1]+1]));

#  List the constraints

subject to

#  Bellman equation for states below N-M
Bellman_1toNminusM {i in X: i <= N-(M-1)}:
      EV[i] = sum {j in 0..(M-1)}
                log(exp(CbEV[i+j])+ exp(-RC + CbEV[1]))* TransProb[j+1];

#  Bellman equation for states above N-M, (we adjust transition probabilities
#  to keep state in [xmin, xmax])
Bellman_LastM {i in X: i > N-(M-1) and i <= N-1}:
      EV[i] = (sum {j in 0..(N-i-1)}
                log(exp(CbEV[i+j])+ exp(-RC + CbEV[1]))* TransProb[j+1])
                + (1- sum {k in 0..(N-i-1)} TransProb[k+1]) * log(exp(CbEV[N])+ exp(-RC + CbEV[1]));

#  Bellman equation for state N
Bellman_N:  EV[N] = log(exp(CbEV[N])+ exp(-RC + CbEV[1]));

#  The probability parameters in transition process must add to one
Probability: sum {i in 1..3} thetaProbs[i] = 1;

#  Put bound on EV; this should not bind, but is a cautionary step to help keep algorithm
#  within bounds
EVBound {i in X}: EV[i] <= 50;

####   DEFINE THE PROBLEM   #####
# Name the problem
problem MPECZurcher:

# Choose the objective function
Likelihood,

# List the variables
EV, RC, thetaCost, thetaProbs, TransProb, Cost, CbEV, PayoffDiff, ProbRegMaint, BellmanViola,

# List the constraints
Bellman_1toNminusM,
Bellman_LastM,
Bellman_N,
Probability,
EVBound;
```

AMPL Command File: GMCBusExampleMLEcommand.txt

```
# Title: Constrained Optimization Approaches for Estimation of Structural Models
# Authors: Che-Lin Su (Chicago GSB) and Kenneth L. Judd (Hoover Institution and NBER)
# November, 2006
# AMPL command file: GMCBusExampleMLEcommand.txt

# List true values for our reference
# let RC := 1.2170;
# let theta1 := 0.0851;

# Call solver and give it options

# default options for SNOPT 6.2
option snopt_options $snopt_options 'outlev=2 timing=1 ';  # output level

# Initial guesses set at trivial values; probably not good initial guess
let {i in X} EV[i] := 0;
let {i in 1..3} thetaProbs[i] := 1/3;


# Solve command
solve MPECZurcher;

display _solve_time;

# Output commands
option display_round 6, display_width 120;

# write the value function
display EV;

# write the structural parameters (remember beta was fixed)
display beta, RC, thetaCost, thetaProbs, TransProb;

# write errors in Bellman equations
display Bellman_1toNminusM.body;
display Bellman_LastM.body;
display Bellman_N.body;
display BellmanViola;
```