COMPUTING NASH EQUILIBRIA FOR STOCHASTIC GAMES

ARUN SEN
DEPT. OF ORFE, PRINCETON UNIVERSITY, PRINCETON, NJ 08544,
ASEN@PRINCETON.EDU
AND DAVID F. SHANNO
RUTCOR, RUTGERS UNIVERSITY, 640 BARTHOLOMEW RD.,
PISCATAWAY, NJ 08854,
SHANNO@RUTCOR.RUTGERS.EDU

Abstract. We describe two new algorithms for computing Nash equilibria of stochastic games. One is a global random-start algorithm based on nonlinear programming. The other combines a dynamical systems approach with nonlinear programming to find stable equilibria. Promising numerical results are presented.

1. Introduction

Since its introduction by Von Neumann and Morgenstern (see [12]), game theory has been a subject of much interest, attracting the attention of researchers in a variety of fields, such as economics, operations research and computer science. Despite all its promise, however, there has long been the limitation that solution techniques have been incapable of handling a vast number of problems of interest. There are a variety of reasons for this. It is sometimes difficult, if not impossible, to model complex human interactions with any reasonable degree of accuracy. The values of many quantities and parameters must be known, and the number of variables required can be intolerably large. But another reason is, even if a reasonable game-theoretic model can be built, its solution has often proved to be intractable. The primary solution concept in game theory is the Nash equilibrium. A Nash equilibrium is guaranteed to exist for almost any game (subject to broad qualifications), so it is potentially an attractive way to analyze games. But finding a Nash equilibrium for a given game turns out to be a difficult mathematical problem, especially if one moves beyond the realm of games played by two players only once. And many realistic problems entail multiple players, or multiple periods and states, or what

Date: June 24, 2005.
is known as *stochastic games*. The equations defining a Nash equilibrium for stochastic games are particularly challenging to solve, as they become nonlinear even with just two players.

Despite the drawbacks, game theory is now being employed much more widely, due to technological and economic changes. The growth in importance of electronic markets and auctions has spawned entirely new applications. Game theory is now being used actively to analyze and design auctions (see [13]), which are the engine behind a variety of transactions, including internet auctions for supply-chain management, energy markets and even corporate take-over battles. Beyond these, there are even further possible applications of game theory. In corporate finance, new quantitative methods, borrowed from the financial world, are being used to value projects and even entire businesses, known as real options analysis. The real options paradigm is essentially a single-player game, really an optimization problem. But real options have had their own shortcomings, one of which is, few markets have only one player! As soon as two players are introduced to make the situation more realistic, one has a stochastic game instead of an (easy) optimization problem (see [3]).

The major purpose of this paper is to present two hybrid methods for finding solutions of stochastic games which could not be solved before. The most common numerical approaches for computing Nash equilibria up to this point have been based on homotopy methods, of which the best known example is the PATH solver (see [4]). However, PATH is only locally convergent, and is normally used for general equilibrium problems. It has been found before, as we shall demonstrate here, that PATH is not particularly effective at solving stochastic games. Recently, some interesting theoretical work has been done in [9] to prove global convergence for a different homotopy method. However, the only numerical examples provided in [9] are of small models, with 12 variables or less. If there has been more numerical experience with this method, we are not aware of it; nor have we had access to the software to test it for ourselves.

A different approach which has started to attract interest is based on mathematical programming. Some work along these lines, though only for the restricted case of pure-strategy equilibria, may be found in [8]. More relevant to this paper is a locally convergent nonlinear programming approach that is described in [2]. In this paper we provide two ways to globalize this method, and show that it seems to work well in practice. This, we believe, distinguishes our paper from past work on the subject. To this end, in section 2 we introduce the specifics of stochastic games, and the concept of a Nash equilibrium in this context.
In section 3 we describe why the Nash equilibrium problem is particularly nasty, and propose a method based on nonlinear programming which can get around the difficulties encountered and compute a Nash equilibrium. In section 4 we describe a new hybrid method, which combines ideas from dynamical systems with nonlinear programing, that goes further in refining the equilibrium concept by giving a way to find a "reasonable" equilibrium, thereby making the very idea of a Nash equilibrium more meaningful to begin with. And in section 5 we provide some promising numerical results.

2. Stochastic Games

In this paper we shall deal with n-player, general-sum stochastic games with finite state and actions spaces, as originally defined by Shapley (see [16]). In this context, a stochastic game is a multi-period game (usually with an infinite horizon) played in discrete time. At any point in time, the game resembles the standard one-shot game. From one time period to the next, the game evolves within a finite set of states, with the transition between states being determined by a combination of the players’ moves and some probability law.

More specifically, let the set of states be $S = \{1, \ldots, N\}$. Let $A^i(s)$ be the (finite) set of actions available to player $i$ when the game is in state $s$. Let $\Pi^i(s, a^1, \ldots, a^n)$ be the payoff to player $i$ when the game is in state $s$ and the players choose the actions $a^1 \in A^1(s), \ldots, a^n \in A^n(s)$. Let $p$ be the transition vector between states, which is defined as follows: $p(s_1, s_2, a^1, \ldots, a^n)$ is the probability that, if the game is currently in state $s_1$ and the players’ actions are $a^1 \in A^1(s_1), \ldots, a^n \in A^n(s_1)$, the game’s next state will be $s_2$. Note, these assumptions impose a Markovian structure on the game— the only information needed to determine the transition between states is the current state, and the actions taken by the players. If we let $\delta$ be the rate at which players discount future payoffs, we may now begin to describe how the behavior in such games can be analyzed.

We shall focus exclusively on what are known as stationary strategies. A stationary strategy for player $i$ is a (randomized) decision rule which is defined by specifying a probability $x^i_{sa}$ for $s \in S, a \in A^i(s)$, where $x^i_{sa}$ is the probability that player $i$ chooses action $a$ when the game is in state $s$. In other words, if players act according to such strategies, their decisions will depend only on the current states of the game. Of course, other kinds of non-stationary strategies can exist, which depend in some way on the past history of the game. One might ask, then, why it is reasonable to restrict attention to the class of stationary strategies.
One answer is that if the Markovian structure of the game is sufficiently rich, that is, if the states are defined sufficiently broadly, then knowing the current state (plus the other players’ strategies) can give a player sufficient information to make his own decisions. Moreover, it is not unreasonable to assume that players make decisions in the simplest way that suits their needs. Another aspect of this issue will be addressed below. Now, if one accepts that this is how players will behave, the question which naturally arises is, what will the ultimate outcome be to such a game?

Assuming a non-cooperative environment, the answer that is considered most sensible is similar to the one for standard one-shot games: a Nash Equilibrium (in stationary strategies). That is, an outcome in which simultaneously each player chooses the stationary strategy which maximizes his own payoff, given the stationary strategies of the other players. (The derivation that follows, in this section and the next, may be found in [5], among other places.) To define this concept more precisely, we first need to define the intertemporal value functions:

If the players’ (stationary) strategies are fixed to be \( \{ x_{sa}^i \} \), define the value function \( v^i \) for player \( i \) to be:

\[
(1) \quad v_s^i = \sum_{a^1 \in A^1(s)} \ldots \sum_{a^n \in A^n(s)} x_{sa^1}^1 \ldots x_{sa^n}^n [\Pi^i(s, a^1, \ldots a^n) + \delta \sum_{s' \in S} p(s, s', a^1, \ldots a^n)v_{s'}^i], \forall s.
\]

In other words, \( v_{s^*}^i \) is simply the present discounted value of the \( i \)’th player’s expected future payoffs when the game is in state \( s^* \), if the \( n \) players adopt the strategies \( \{ x_{sa}^i \} \) for all states \( s \) and actions \( a \). Then a Nash Equilibrium is simply a collection of strategies \( \{ x_{sa}^i \} \) that solve the Bellman equations (again, see [5] for a justification of this approach):

\[
(2) \quad v_s^i = \max_{a^1 \in A^1(s)} \sum_{a^1 \in A^1(s)} \ldots \sum_{a^{i-1} \in A^{i-1}(s)} \sum_{a^{i+1} \in A^{i+1}(s)} \ldots \sum_{a^n \in A^n(s)} x_{sa^1}^1 \ldots x_{sa^{i-1}}^{i-1} \cdot x_{sa^{i+1}}^{i+1} \ldots x_{sa^n}^n [\Pi^i(s, a^1, \ldots a^{i-1}, a, a^{i+1}, \ldots a^n) + \delta \sum_{s' \in S} p(s, s', a^1, \ldots a^{i-1}, a, a^{i+1}, \ldots a^n)v_{s'}^i], \forall s, \forall i.
\]

The first question that comes to mind is, of course, do such equilibria always exist? After all, Nash’s original, famous result only applied to finite, one-shot games. It turns out, however, that the result does
extend naturally to finite stochastic games. In particular, a stochastic game with finite state and action spaces does have at least one Nash equilibrium, in stationary strategies (see [5]). This result makes it feasible and reasonable to focus on stationary strategies only. That is not to say that past history has no bearing on the outcome of a game. We shall demonstrate that in subsequent sections when we deal with dynamics. The remainder of this paper will be concerned with how to compute Nash equilibria, and how to find "reasonable" equilibria when more than one exists.

3. NASH EQUILIBRIA AND NONLINEAR PROGRAMMING

As mentioned in the introduction, the problem of computing a Nash equilibrium by solving the dynamic programming equations as defined by (2) is nontrivial. Here we show why this is so.

3.1. Mathematical Programs with Equilibrium Constraints.
As we have described, the following relations are necessary and sufficient conditions to have a Nash equilibrium: a set of strategies \( \{ x^i_{sa} \} \) that satisfies

\[
\begin{align*}
\pi^i_{sa} &= \sum_{a^i \in A^i(s)} \sum_{a^{i-1} \in A^{i-1}(s)} \sum_{a^{i+1} \in A^{i+1}(s)} \sum_{a^n \in A^n(s)} x^1_{sa1} * \ldots * x^i_{sa1} * \ldots * x^{i-1}_{sa1} * \\
&* x^{i+1}_{sa} \ldots x^n_{sa} [\prod [s, a^1, \ldots a^{i-1}, a, a^{i+1}, \ldots a^n] v^i_{si}] & \forall s, a^i \in A^i(s), \forall i.
\end{align*}
\]

Now define the expected payoff \( \pi^i_{sa} \) from a particular action. to be:

\[
\begin{align*}
\pi^i_{sa} &= \sum_{a^i \in A^i(s)} \sum_{a^{i-1} \in A^{i-1}(s)} \sum_{a^{i+1} \in A^{i+1}(s)} \sum_{a^n \in A^n(s)} x^1_{sa1} * \ldots * x^{i-1}_{sa1} * \\
&* x^{i+1}_{sa} \ldots x^n_{sa} [\prod [s, a^1, \ldots a^{i-1}, a, a^{i+1}, \ldots a^n] v^i_{si}] & \forall s, a^i \in A^i(s).
\end{align*}
\]

That is, \( \pi^i_{s*} \) is the payoff to player \( i \) if with the game currently in state \( s^* \) he chooses \( a \) and in all subsequent states \( s \) plays according to \( \{ x^i_{sa} \} \), and each opponent \( j \) always plays \( \{ x^j_{sa} \} \) for all states \( s \) and actions \( a \). Then the conditions for a Nash equilibrium may be written as:

\[
\begin{align*}
\pi^i_{sa} &= \sum_{a^i \in A^i(s)} \sum_{a^{i-1} \in A^{i-1}(s)} \sum_{a^{i+1} \in A^{i+1}(s)} \sum_{a^n \in A^n(s)} x^1_{sa1} * \ldots * x^{i-1}_{sa1} * \\
&* x^{i+1}_{sa} \ldots x^n_{sa} [\prod [s, a^1, \ldots a^{i-1}, a, a^{i+1}, \ldots a^n] v^i_{si}] & \forall s, a^i \in A^i(s), \forall i.
\end{align*}
\]
(5) \[ v_s^i - \pi_s^{i \pi} \geq 0, x_{sa}^i \geq 0, \forall s, \forall a^i \in A^i(s), \]
\[ \sum_{a^i \in A^i(s)} x_{sa}^i = 1, \forall s \in S, \forall i, \]

with \( v^i(s) \) defined by (1).

Since the \( \{ x_{sa}^i \} \) are probabilities, (1) may be rewritten as follows:

\[ \sum_{a^i \in A^i(s)} x_{sa}^i v_s^i = \sum_{a^i \in A^i(s)} ... \sum_{a^n \in A^n(s)} x_{sa1}^1 \ast ... \ast x_{san}^n \Pi_i^i(s, a^1, ... a^n) \]
\[ + \delta \sum_{s' \in S} p(s, s', a^1, ... a^n) v_{s'}^i, \]

which then may be written as

\[ \sum_{a^i \in A^i(s)} x_{sa}^i v_s^i = \sum_{a^i \in A^i(s)} x_{sa}^i \pi_{sa}^i, \]

or equivalently

\[ \sum_{a^i \in A^i(s)} x_{sa}^i (v_s^i - \pi_{sa}^i) = 0, \forall s, \forall i. \]

Given (5), we may now write the conditions for a Nash equilibrium as:

(6) \[ x_{sa}^i (v_s^i - \pi_{sa}^i) = 0, \]
\[ v_s^i - \pi_{sa}^i \geq 0, x_{sa}^i \geq 0, \forall s, \forall a^i \in A^i(s), \]
\[ \sum_{a^i \in A^i(s)} x_{sa}^i = 1, \forall s \in S, \forall i. \]

An intuitive justification for why this system constitutes an equilibrium is that each player only plays with positive probability those actions in any state whose payoff achieves the maximum; all other actions are never played at all. (6) may be viewed as a system of nonlinear equations in \( \{ x_{sa}^i \} \) and \( \{ v_s^i \} \) (note, \( \pi_{sa}^i \) is only being used for compactness of notation, it is really just a function of the variables, as given by (4)). All the functions are polynomials, and the degree of the polynomials is equal to \( n + 1 \), one more than the number of players. That is why even a two-player stochastic game gives rise to a nonlinear problem, which in general will be nonconvex. One might thus imagine that direct methods for solving nonlinear equations numerically, in other words, the techniques of nonlinear programming, could be applied to solve this problem. And indeed, that is the case. But the nonlinear
program constituted by (6) (with an objective function of 0) is a very
difficult problem, as it falls in a particular class of nonlinear programs,
called mathematical programs with equilibrium constraints (MPEC’s),
which are known to cause difficulties for standard mathematical pro-
gramming techniques.

**Definition 1.** A Mathematical Program with Equilibrium Constraints
(MPEC) is a nonlinear program of the form

\[
\begin{align*}
\min & \quad f(x) \\
\text{s.t.} & \quad F(x)^T G(z) = 0 \\
& \quad F(x) \geq 0, G(z) \geq 0, \\
& \quad H(x, z) = 0.
\end{align*}
\]

It is shown in, among other places [14], that MPEC’s post theoretical
problems for nonlinear programming algorithms. In [1] it is shown
how to resolve this theoretical difficulty by reformulating the nonlinear
program in particular way. We show in a previous paper, [2] , that this
theoretical fix does indeed make a difference in practice, and makes it
practically feasible to solve MPEC’s. As our problem (6) is an MPEC,
with a constant objective function, the results from [2] are directly
applicable (indeed, the most important example that we present in
that paper is a dynamic game, a model which we shall deal with more
thoroughly later in this paper). These results show that a good way to
solve (6) is to solve the nonlinear program

\[
\begin{align*}
\min & \quad \zeta \\
\text{s.t.} & \quad x_{sai}^i (v_s^i - \pi_{sai}^i) \leq \zeta, \\
& \quad v_s^i - \pi_{s}^i \geq 0, x_{sai}^i \geq 0, \forall s, \forall a^i \in A^i(s), \\
& \quad \sum_{a^i \in A^i(s)} x_{sai}^i = 1, \forall s \in S, \forall i,
\end{align*}
\]

with solutions of 0 corresponding to Nash equilibria. This approach has
one deficiency, however. This nonlinear program, while well-behaved,
remains highly nonconvex. As a result, only local convergence is guar-
anteed for any standard nonlinear solver applied to such a problem.
That is, from an arbitrary starting point, a nonlinear solver may very
well produce a solution with \( \zeta > 0 \), which is not a Nash equilibrium.
We now propose a simple extension to this approach which guarantees
getting a solution of 0.
3.2. A Global Random-Start Algorithm. Having to find a solution of 0 to (7) is a global optimization problem. There is generally no guaranteed way to find the global optimum to a nonconvex problem, but in this case we have an advantage: we know that at the global optimum, the value of the objective function is 0, since at least one Nash equilibrium must exist for a finite game. The results from [1] and [2] indicate that choosing a starting point sufficiently close to a solution of (7) guarantees convergence to that solution for a standard nonlinear solver. Thus an obvious process suggests itself:

Randomly select starting points from the space \( \{x^t_{sa}, v^t\} \) for a nonlinear solver applied to (7), until a solution of 0 is found. Of course searching all real numbers for starting points would be extremely inefficient. Fortunately, that is not necessary. One knows that at a solution the variables must satisfy (1). Thus it is sufficient to search the space of probability vectors to choose the starting \( \{x^t_{sa}\} \), and then solve the linear system (1) to determine the starting \( \{v^t\} \).

This procedure is guaranteed with probability 1 to eventually produce a Nash equilibrium. The question is, how long will it take? How efficient will it be in practice? In a subsequent section we shall provide a numerical example that shows that, for a broad class of stochastic games, this method can be quite efficient for a reasonably large number of variables.

4. Nash Equilibria and Dynamics

In the previous section we described a method to compute a Nash equilibrium. But we did not address a major unresolved issue in game theory: The question of how to select amongst equilibria, when a game has more than one, has attracted much attention for decades. Without a satisfactory answer to this question, it is not even clear how meaningful a solution concept Nash equilibrium really is. A great amount of research has been done on this topic for bimatrix games (i.e. two player, single-shot games), though without any conclusive result. Work on equilibrium selection for stochastic games is very much in its infancy (see [5]).

Much of the work on equilibrium selection, not unreasonably, has centered on dynamics. The very word ”equilibrium” suggests that Nash equilibrium should be the steady state of some sort of dynamic process. Various kinds of models have been proposed over the years as a dynamic framework for games, such as gradient dynamics and replicator dynamics (for details about these, see for instance [22]). But these models have drawbacks, especially in the context of stochastic games.
The concern with gradient dynamics ("gradient" refers to each player moving in the direction of steepest improvement of his payoffs) is how to define the payoff gradient in a stochastic game, which has multiple states. Replicator dynamics have the undesirable property that 0 is always a fixed point. This might be reasonable in evolutionary biology (an extinct species cannot rise from the dead), where the idea of course originates, but is problematic in games with individual agents. A player might very well start taking a particular action that he was avoiding before. A further source of dissatisfaction with all dynamics proposed thus far is that convergence to a Nash equilibrium cannot be guaranteed, and is often quite rare. Moreover, from our point of view, a major shortcoming with the work done so far is that it has been by and large theoretical in nature, often not dealing with games with more than two players and two actions per player (let alone stochastic games). Thus there is nothing which can be systematically applied to analyze practical models of any size. We shall now make an attempt to tackle some of these issues here.

4.1. Variational Inequalities and Projected Dynamical Systems. We are going to propose a simple, but we think reasonable, dynamic model for stochastic games. But before doing so, we need to define some concepts.

Definition 2. Given a closed convex set $K \subseteq \mathbb{R}^n$ and a mapping $F : K \to \mathbb{R}^n$, the finite-dimensional variational inequality (VI) problem is to find $x^* \in K$ such that $F(x^*)^T (x - x^*) \geq 0 \ \forall x \in K$.

It is not hard to see that our Nash equilibrium problem, (6), is a variational inequality. The variables are the $\{x_{sa}^i\}$, $K$ is the simplex, $\sum_{a^i \in A^i(s)} x_{sa}^i = 1$, $x_{sa}^i \geq 0, \forall s \in S, \forall a^i \in A^i(s), \forall i$, and $F(x_{sa}^i) = v^i_s - \pi_{sa}^i$. The $v^i$ are simply functions of the $\{x_{sa}^i\}$, uniquely defined by (1) (note that for the nonlinear programming approach the $v^i$ were treated as variables), as of course are the $\{\pi_{sa}^i\}$ (defined by (4)).

Every variational inequality may be viewed in a canonical way as the fixed point of a particular dynamical system. To describe this system, we first need the following:

Definition 3. For $x \in K \subseteq \mathbb{R}^n$, $K$ closed and convex, $v \in \mathbb{R}^n$, the projection of $v$ at $x$ with respect to $K$, $\Lambda_K(x, v)$, is given by:

$$\Lambda_K(x, v) = \lim_{\delta \to 0} \frac{P_K(x + \delta v) - x}{\delta},$$

where $P_K(v)$ is the minimum (L2) norm projection of the vector $v$ onto $K$. 
We may now define the *projected dynamical system* associated with every VI:

**Definition 4.** The projected dynamical system (PDS) associated with every VI is given by the following ordinary differential equation (ODE):

\[ \dot{x} = \Lambda_K(x, -F(x)), \]

with a given initial condition \( x(0) = x_0 \).

The important thing to note about the PDS is that its fixed points (i.e. where \( \dot{x} = 0 \)) correspond precisely with the solutions of the associated VI. This makes it a natural, dynamic extension to the VI framework. Moreover, we submit that applied to (6), it is a model which makes good economic sense. It is in a way not that different from the replicator dynamics: the (relative) frequency of a particular action, \( x_{sa}^i \), in a given state, will tend to increase (decrease), if the expected payoff from that particular action, \( \pi_{sa}^i \), is greater (less) than the current average payoff in that state, \( v_s^i \). The crucial difference is in the presence of the projection operator, which ensures that the trajectory will stay in the required space, without having to artificially make \( 0 \) a fixed point.

The basic results concerning the PDS, such as existence, uniqueness, and continuity are given in [11]. These results, which describe the conditions required for the PDS to be well-behaved, only depend on \( F \) being Lipschitz continuous, which it certainly will be in our application. In [11] the PDS is proposed as a dynamic model for network and spatial equilibrium problems, but we believe we are the first to consider it for stochastic games. Furthermore, merely proposing it as a model is not enough, of course. As we shall see, actually making use of it is nontrivial. The idea is that if the PDS converges to a (Nash) equilibrium, then that equilibrium must be stable, and hence a "reasonable" outcome. Unstable equilibria may discarded as unlikely to occur. One way to use the PDS, then, is to employ the method described in the previous section to compute a Nash equilibrium, and then check to see if it is stable with respect to the PDS. The hope is that stable equilibria are common enough to make this a worthwhile selection criterion. The drawback, though, with the approach just outlined, is that one might check many equilibria without finding a stable one, and not have any idea if a stable one exists. It would be far better to track the trajectory the PDS takes from a given starting point. If the trajectory fails to converge from enough starting points, then by definition one can conclude that there is no stable equilibrium. This idea might sound
simple, but we shall now discuss the complications which arise when trying to implement it.

4.2. Integrating the Projected Dynamical System. Simulating the PDS of course means integrating a system of ODE’s. We must design our own scheme for doing this, as no existing software is applicable to this kind of projected system. Standard methods exist for numerically integrating classical ODE’s (i.e. equations of the form \( \dot{x} = f(x) \)), and fortunately, these can be readily extended to the projected case.

**Euler Method:** The Euler method is the simplest way to integrate a classical ODE. It simply says that given the current point \( x_t \), the next point is produced according to the following relation:

\[
x_{t+1} = x_t + \alpha_t f(x_t),
\]

where \( \alpha_t \) is an appropriately chosen steplength.

The extension to the PDS is simple enough. The relation becomes:

\[
x_{t+1} = P_K[x_t + \alpha_t F(x_t)].
\]

The Euler method is not very accurate, however, as it is only a first-order method (meaning that the error in the integration is of order 2 with respect to the steplength—see [15] for details).

**Predictor-Corrector:** The predictor-corrector method, as we shall use it, is as follows for a classical system:

\[
x_{t+1} = x_t + \frac{\alpha_t}{2}(f(x_t) + f(x_t + \alpha_t f(x_t))).
\]

Here we use the Euler method as the predictor step, and the trapezoid rule for the corrector step (again, see [15]). Applied to the PDS, this becomes:

\[
x_{t+1} = P_K[x_t + \frac{\alpha_t}{2}(F(x_t) + F(P_K[x_t + \alpha_t F(x_t)]))].
\]

The predictor-corrector step defined in this way is a second-order method ([15]). The use of these classical methods to integrate the projected system is suggested in [11], but only the Euler method is actually implemented in that book. As the authors themselves acknowledge, a first-order method is not likely to be adequate for practical purposes, though until now no one has tried to implement a higher-order method for projected systems. This is what we shall endeavour to do, but first we must discuss what this entails.

To implement the numerical scheme defined by (9), three steps are necessary:
• Compute $F(x_t)$
• Select an appropriate $\alpha_t$
• Perform the projection onto the simplex $K$

Computing $F(x_t)$ requires that the linear system (1) be solved for the $v^i$. Selecting the appropriate $\alpha_t$ is the most sensitive part of the exercise. First, a starting value is required. Then, we must have a scheme to change $\alpha_t$ as required during the course of the integration, to keep the error of the approximation to the true trajectory within acceptable bounds. Finally, the projection onto $K$ requires the solution of a (convex) quadratic program. We shall discuss the details of our implementation in a later section when we present our numerical examples.

4.3. A Hybrid Method to Compute Stable Nash Equilibria. Two further complications need to be considered. The first is that even if the PDS converges to a Nash equilibrium, it will do so quite slowly. This is due to the fact that a system of the form $\dot{x} = f(x)$ converges only at a first-order rate to its stable fixed points (or more precisely, any discrete approximation to $\dot{x} = f(x)$, no matter how accurate, will only converge to a fixed point at a first-order rate). As we do not believe this observation has been made in the context of the dynamics of games, we shall quickly prove it here. For this purpose, we first need the following definition of order (which is taken from [20]).

(For the following definition and theorem, we will use superscripts to refer to iteration number, and subscripts to refer to the component number in a vector.)

**Definition 5.** Given a sequence of points $\{x^i\}$ in $R^n$ where $x^{i+1} = \phi(x^i)$, $\phi$ an n-dimensional mapping with p continuous partial derivatives, if $\{x^i\} \rightarrow x^*$ (so that $\phi(x^*) = x^*$), the convergence is of order $p$ if

$$\frac{\partial \phi_i(x^*)}{\partial x_{j_1}...\partial x_{j_k}} = 0, \forall 1 \leq k \leq p - 1, \forall 1 \leq i, j_1, ..., j_k \leq n, \text{ and } \frac{\partial \phi_p(x^*)}{\partial x_{j_1}...\partial x_{j_p}} \neq 0$$

for at least one set of values $1 \leq i, j_1, ..., j_p \leq n$.

In other words, a sequence generated by a mapping $\phi$ and converging to $x^*$ is of order $p$ if all partial derivatives of $\phi$ of order less than $p$ vanish at $x^*$, and at least one $p$'th order partial derivative is nonzero. We then have the following simple result.

**Theorem 1.** Given a sequence of points $\{x^i\}$ in $R^n$ where $x^{i+1} = \phi(x^i) = x^i + \alpha f_i(x^i)$ and $f$ has continuous first order partial derivatives, if $\{x^i\}$ converges to $x^*$ (so that $f(x^*) = 0$) for sufficiently small $\alpha$, the convergence is of order one.

**proof.** We simply compute one of the first order partials of $\phi$ at $x^*$:
\[
\frac{\partial \phi_1(x^*)}{\partial x_1} = \lim_{h \to 0} \frac{\phi_1(x_1^* + h, x_2^*, \ldots, x_n^*) - \phi_1(x^*)}{h} \\
= \lim_{h \to 0} \frac{x_1^* + h + \alpha f_1(x_1^* + h, x_2^*, \ldots, x_n^*) - x_1^*}{h} \\
= \lim_{h \to 0} \frac{h + \alpha f_1(x_1^* + h, x_2^*, \ldots, x_n^*)}{h} \\
= \lim_{h \to 0} 1 + \alpha \frac{\partial f_1(x_1^* + h, x_2^*, \ldots, x_n^*)}{\partial x_1} \quad \text{(applying L’Hopital’s Rule)}.
\]

Since \( f \) has continuous first order partials, it follows that this final limit will be positive for sufficiently small \( \alpha \). \( \square \)

The second possibility of which we should be cognizant, which we have mentioned as a common worry concerning Nash equilibria and dynamics, is that convergence to an equilibrium might be quite rare. This is due to the fact that convergence to a fixed point \( x^* \) requires that \( x^* \) be a strictly monotone attractor.

**Definition 6.** An equilibrium point \( x^* \) of a dynamical system is a strictly monotone attractor if \( \exists \epsilon > 0 \) such that

\[
\forall x_t \in B(x^*, \epsilon), \quad d(x_t, x^*) < 0,
\]

where \( B(a, r) \) is an open ball of radius \( r \) centered at \( a \), and \( d(x, y) \) is the Euclidean distance between \( x \) and \( y \).

The point is that not all stable equilibria need be strict attractors. An equilibrium \( x^* \) is a monotone attractor if all trajectories in some neighborhood of \( x^* \) remain in that neighborhood. In other words, the system can approach the vicinity of \( x^* \) without ever actually reaching it. This kind of stability can obviously be frustrating if we are hoping to find a nicely convergent system. In order to deal with these concerns, we propose the following hybrid method:

Simulate the PDS. If it appears to get close to an equilibrium (according to some reasonable definition), but progress either slows, or stalls completely, use the current point as the starting point for an appropriate nonlinear programming algorithm applied to the nonlinear program (7).

As we have discussed, if the starting point is sufficiently close to the equilibrium, the nonlinear solver is guaranteed to converge to it. This addresses both the concerns raised: most nonlinear solvers locally converge to a solution at a superlinear (usually quadratic) rate. That is, they are locally much faster than a first-order method. For instance,
interior-point methods are based on Newton’s method for solving equations, which has a known quadratic rate (see [20]) and SQP methods are also superlinear (see e.g. [6]). Furthermore, this hybrid method will find equilibria which are sufficiently stable (i.e. which pull the system in sufficiently close), even if they are not strict attractors. As far as we know, this has not been possible before. In the next section, we shall discuss the details of our implementation, and provide some promising numerical results.

5. Implementation and Numerical Examples

5.1. Implementation of Global Random-Start Algorithm. The local nonlinear solver we used to implement the random-start algorithm was LOQO, an interior-point method (see [21] for details concerning LOQO) which was the solver we used successfully to solve MPEC’s in our previous paper ([2]). The coding for the random-start routine was done entirely in the AMPL programming language (see [7] for details about AMPL). We used AMPL’s uniform random-number generator to produce the random-numbers used to select the starting points \( \{x_{sa}\} \), and then used LOQO to solve the linear system (1) to find the starting \( v^i \). The termination condition was that the objective function of the nonlinear program (7), \( \zeta \), be less than .00001. We should emphasize that we sampled starting points from the space of probability vectors according to a uniform distribution, with each point sampled independently of all previous points. This is certainly not the most efficient procedure, and we plan to investigate more elaborate sampling methods in the future, but for now it is adequate for the purpose of testing the potential of the random-start approach.

5.2. Details of PDS Integration. As discussed before, there are three steps to the integration: computing \( F \), selecting the steplength \( \alpha_t \), and performing the projection onto \( \bar{K} \). The main work in computing \( F \) is to solve the linear system (1), which as for the random-start algorithm, we did using LOQO. We also used LOQO to compute the projection. The reason we used LOQO for both these functions is that interior-point methods are provably polynomial-time algorithms, when applied to both linear programs and convex quadratic programs (see [23]).

This leaves the issue of choosing the steplength. It is first necessary to choose a starting value, \( \alpha_0 \). To do this, we noted that the trajectory of our system must lie on the simplex \( K \), and set \( \alpha_0 \) to be the maximum value such that
\[ x^i_{sa} + \alpha_i F(x^i_{sa}) < 2, \forall s, a \in A^i(s), \forall i, \]

2 being a number which seemed to work in practice, based on our experiments (which we shall shortly discuss). Then we need a rule to control the step length during the integration, to keep the error within acceptable bounds. The error in question is usually referred to as the “local error”, referring to the error in integration that occurs over one step, as opposed to the cumulative error. For the predictor-corrector method that we employed, if we let \( \tilde{x}_{t+1} = P_K[x_t + \alpha_t F(x_t)] \), then an asymptotically correct estimate of the error is \( x_t - \tilde{x}_t \), where by “asymptotic” we mean that as \( \alpha_t \to 0 \), the estimate approaches the true error (see [15] for details). Then \( \alpha_{t+1} \) is kept unchanged if

\[ x_t - \tilde{x}_t < \tau, \]

where \( \tau \) is the error tolerance. We set \( \tau \) to be \( .01/\sqrt{t} \), where \( t \) is the iteration number. The dependence on \( t \) was due to the desire to increase the accuracy as the simulation neared an equilibrium (if indeed it approached one). The precise number, again, we arrived at by trial and error. This now leads to the remaining question of how to actually change the step length, when it needs changing. We set \( \alpha_t \) to be \( \frac{m}{n} \), i.e. the ratio of two integers. We initially set \( n \) to 1 (thus setting \( m = \alpha_o \)). After that we never increased the step length, and decreased it when it failed the error test, by simply increasing \( n \) by 1. This way, only small changes were made to the step length each time, making the simulation more stable.

Finally, we must mention the not totally trivial matter of how to measure progress during the simulation. As the system (6) is of the form

\[ x^i F(x, v) = 0, \]
\[ x \geq 0, F(x, v) \geq 0, \]
\[ \sum_i x_i = 1, \]

we decided to use the quantity \( \sqrt{\frac{\sum (x_i F(x))^2}{\|x\|^2 \|F(x)\|^2} + \frac{\sum x_i^2}{\|x\|^2} + \frac{\sum F(x)^2}{\|F(x)\|^2}} \) to monitor progress. Clearly this goes to 0 if we approach an equilibrium and is normalized to deal with numbers of different orders of magnitude. In nonlinear programming parlance this is known as a measure of ”infeasibility”. We found this preferable to monitoring the distance between the successive iterates of the simulation (which is often how progress
towards a fixed point is measured, e.g. in [11]), as it is independent of the step length.

5.3. **Implementation of Hybrid Method.** The main issue with implementing the hybrid method was what nonlinear solver to use, if we in fact decided to switch to the nonlinear programming approach. The idea behind the hybrid method is dependent on a point in the space \( \{ x^i_{sa}, v^i \} \) being close enough to an equilibrium to give the nonlinear solver a good "warm start". This assumes that the solver will not add any extra variables of its own. But interior-point methods such as LOQO typically have to add slack variables to a problem before solving it (see [21]). Initially, these slacks should not be close to 0, or the method will proceed very slowly, even if they must be 0 at the solution; this is the classic warm-start problem with interior-point methods (see [10]). We thus decided to use FILTER, an SQP method, as the local solver for the hybrid approach (see [6] for details on FILTER), as SQP methods do not add any extra variables to a model formulation.

5.4. **A Numerical Example.** We consider a model that is proposed in [18] and [19] as an extension of the model discussed extensively in those papers. The model is of a game involving two (identical) firms which compete on the basis of price to sell an identical product. The game takes place over an infinite time-horizon at discrete intervals. In each round, a firm may or may not be free to move. If it is, it has the option to choose a price from a discrete set of prices or sit out of the market for a round. If it does choose a price, it is committed to that price for the following round as well. The payoff matrix for an individual round is determined by a standard duopoly structure (see [18] and [19]). Each firm tries to maximize its future (discounted) payoff. Note that nature has no role in this game, that is, the actions of the players (and the current state) entirely determine what the next state of the game will be. The strategy variables of this problem can be defined as follows:

Let the price space be \((-1, 0, 1,...n - 1)\), where \( n \) is the number of prices, and \(-1\) corresponds to the action of waiting (i.e. not setting a price). Then we have

i) A reaction matrix \( R \). The \((i, k)\) entry of \( R \) gives the probability that the firm will take action \( k \) when its rival is committed to price \( i \) from the previous round. The action space that \( k \) belongs to consists of the prices \( 0,...n - 1 \) and the option of waiting.
ii) A strategy vector \( s \). The coordinates of \( s \) correspond to the prices \( 0, \ldots n - 1 \) and the option of waiting. The entry \( k \) of \( s \) gives the probability that the firm takes action \( k \) when it does not know its rival’s price for the current period.

iii) Value functions. These are:

\( v_{Ri} \): the expected discounted payoff for a firm which is free to choose a new price and its rival is committed to price \( i \) from the previous period.

\( v_S \): the expected discounted payoff for a firm when both it and its rival are free to choose new prices for the current period.

\( w_{Ri} \): the expected discounted payoff for a firm when it is committed to price \( i \) from the previous period, but its rival is free to choose a new price.

\( w_{Si} \): the expected discounted payoff for a firm when it is committed to price \( i \), and its rival to price \( j \) from the previous period.

The value functions must satisfy the relations:

\[
v_S = \sum_k \sum_l s_k s_l (\Pi_{kl} + \delta \begin{cases} v_S & k = -1, l = -1 \\ v_{Rk} & k = -1, l \geq 0 \\ w_{Rk} & k \geq 0, l = -1 \\ w_{kl} & k \geq 0, l \geq 0 \end{cases}) = 0,
\]

\[
v_{Ri} = \sum_k r_{ik} (\Pi_{ki} + \delta \begin{cases} w_{Rk} & k \geq 0 \\ v_S & k = -1 \end{cases}), \forall i \geq 0,
\]

\[
w_{Ri} = \sum_l r_{il} (\Pi_{il} + \delta \begin{cases} v_{Rl} & l \geq 0 \\ v_S & l = -1 \end{cases}) \forall i \geq 0,
\]

\[
w_{Si} = \Pi_{ij} + \delta v_S \forall i \geq 0, j \geq 0.
\]

Then the nonlinear system (6) takes the form
\[
\begin{align*}
\ r_{ik} & = (\Pi_{ki} + \delta \left\{ \begin{array}{l}
  w_{R_k} k \geq 0 \\
  v_s k = -1 \\
\end{array} \right\}) = 0 \forall i \geq 0, k, \\
\ s_k & = \sum_l s_l (\Pi_{kl} + \delta \left\{ \begin{array}{l}
  v_s k = -1, l = -1 \\
  v_{R_k} k = -1, l \geq 0 \\
  w_{R_k} k \geq 0, l = -1 \\
  w_{S_{kl}} k \geq 0, l \geq 0 \\
\end{array} \right\}) = 0 \forall k, \\
\ v_{R_k} & = (\Pi_{ki} + \delta \left\{ \begin{array}{l}
  w_{R_k} k \geq 0 \\
  v_s k = -1 \\
\end{array} \right\}) \geq 0 \forall i \geq 0, k, \\
\ v_s & = \sum_l s_l (\Pi_{kl} + \delta \left\{ \begin{array}{l}
  v_s k = -1, l = -1 \\
  v_{R_k} k = -1, l \geq 0 \\
  w_{R_k} k \geq 0, l = -1 \\
  w_{S_{kl}} k \geq 0, l \geq 0 \\
\end{array} \right\}) \geq 0 \forall k, \\
\ r_{ik}, s_k & \geq 0 \forall i \geq 0, k, \\
\sum_k r_{ik} & = 1 \forall i \geq 0 \text{ and } \sum_k s_k = 1, \\
\ w_{R_k} & = \sum_l r_{il} (\Pi_{il} + \delta \left\{ \begin{array}{l}
  v_{R_l} l \geq 0 \\
  v_s l = -1 \\
\end{array} \right\}) \forall i \geq 0, \\
\ w_{S_{ij}} & = \Pi_{ij} + \delta v_s \forall i \geq 0, j \geq 0.
\end{align*}
\]

We applied our methods to this problem, for 7, 10, 15, 20, and 30 prices (which result in 64, 121, 256, 441, and 961 decision variables respectively). For the random-start algorithm, the following table lists the number of random starts needed to get a solution (i.e. a solution of 0) for each model instance.

**Global Method for Duopoly Problem (LOQO)**

<table>
<thead>
<tr>
<th>Number of prices/variables</th>
<th>Number of random starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/64</td>
<td>1</td>
</tr>
<tr>
<td>10/121</td>
<td>1</td>
</tr>
<tr>
<td>15/256</td>
<td>6</td>
</tr>
<tr>
<td>20/441</td>
<td>90</td>
</tr>
<tr>
<td>30/961</td>
<td>5</td>
</tr>
</tbody>
</table>

We should note that even for the largest of these problems, LOQO took less than a minute (about 20 seconds) to find a local solution from each starting point. We also tried using FILTER for the random-start method, and got the following results:
Global Method for Duopoly Problem (FILTER)

<table>
<thead>
<tr>
<th>Number of prices/variables</th>
<th>Number of random starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/64</td>
<td>3</td>
</tr>
<tr>
<td>10/121</td>
<td>1</td>
</tr>
<tr>
<td>15/256</td>
<td>28</td>
</tr>
<tr>
<td>20/441</td>
<td>101</td>
</tr>
<tr>
<td>30/961</td>
<td>1000+ (no solution)</td>
</tr>
</tbody>
</table>

It quickly became apparent that FILTER takes many more random starts, though why this is the case we have not yet determined. We note that FILTER found a solution (to the nonlinear program (7)) from every starting point, but the objective function value was invariably non-zero.

The following table gives the results for the hybrid method. For each (random) starting point, we tried to switch to the nonlinear solver after the first 25 iterations, and then after every 50 subsequent iterations. If we failed to find a solution after 526 iterations, we changed the starting point. The table indicates how many starting points had to be tried, how many iterations it took before we were able to successfully switch to the local nonlinear solver, and for the trajectories from which we successfully switched, what the infeasibility was at the starting point, and at the final point before we switched.

Hybrid Method Applied to Duopoly Problem

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7/64</td>
<td>1</td>
<td>25</td>
<td>.5013</td>
<td>.001</td>
</tr>
<tr>
<td>10/121</td>
<td>1</td>
<td>226</td>
<td>.6089</td>
<td>.0036</td>
</tr>
<tr>
<td>15/256</td>
<td>1</td>
<td>526</td>
<td>.7376</td>
<td>.0056</td>
</tr>
<tr>
<td>20/441</td>
<td>1</td>
<td>126</td>
<td>.71</td>
<td>.011</td>
</tr>
<tr>
<td>30/961</td>
<td>5</td>
<td>No sol.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the problem with 30 prices we were unable to find a solution in 5 tries, indicating that no particularly stable equilibrium exists, which is an acceptable answer: there is no guarantee that a given game will have a stable solution. The models with 7 and 10 prices have strict attractors, meaning that if we let the PDS run, it converges to the equilibrium eventually on its own. But the models with 15 and 20 prices are instances of games which have stable equilibria which are not strict attractors (the final model may be of the same type, as the PDS reduced its infeasibility significantly, as it did for the others, but the degree of stability is less- it is a matter of judgement how much
stability one desires, and hence how many starting points one wants to try). This shows that our hybrid method does indeed have the ability to find stable equilibria which could not be found before. Even for the problems with strict attractors, switching to the nonlinear solver speeds up convergence, and enhances accuracy. We take the model with 7 prices as an example. For this model, after 176 iterations, from a typical starting point (the same one as in the table), the PDS reaches the following point:

\[
 x_s = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},
\]

\[
 x_r = \begin{bmatrix}
 0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\
 0 & .353909 & 0 & 0 & .000314 & .089672 & .556104 & 0 \\
 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 6 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}.
\]

At this point, the infeasibility is .001. Switching to the nonlinear solver instantly yields the solution

\[
 x_s = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},
\]

\[
 x_r = \begin{bmatrix}
 0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\
 0 & .360441 & 0 & 0 & 0 & .089672 & .549887 & 0 \\
 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 6 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}.
\]

This has an infeasibility of less than \(10^{-6}\). The PDS takes 376 iterations to reduce the infeasibility to \(1.35\times10^{-6}\), beyond which it is unable to proceed, even if we cut the steplength at every iteration. Thus the hybrid method is clearly better than the PDS by itself, even for a small model.

This last example also serves to highlight the efficacy of our integration scheme for the PDS. After a few iterations the steplength stayed constant, with \(m = 6.4157, n = 10\), and this constant step was sufficient to get us to the equilibrium. As mentioned, increasing the steplength at this point by increasing \(n\) did not improve the accuracy.
of the solution, suggesting that the original step was already fairly accurate. Furthermore, this step-control policy is clearly faster than a policy of always cutting the step (by increasing \( n \) at each iteration, the method proposed in [11]). This shows that even if the purpose of integration is to reach a fixed point, which is usually not the goal when people simulate dynamical systems, a constant steplength can still be effective.

We can also use this example to show that our integration scheme is reasonably scale invariant. Scale invariance says that a classical system  
\[
\dot{x} = af(x)
\]
should have the same trajectory (in the \( \{x\} \) space), for any positive number \( a \). The same should hold for the PDS. We tested our integration for this property by multiplying the payoff matrix by a factor of 36 (which transforms all the payoffs into integers). For this model our simulation behaved in an almost identical fashion, but this time it settled down to a constant steplength with \( m = .1782 \) (with \( n = 10 \)), which is what one would expect, since increasing the payoffs (and hence \( F \)) by a constant factor effectively increases the speed at which the system moves (but not the direction), so a smaller steplength is needed to compensate. Thus our strategy for choosing an initial steplength seems to work, which makes the PDS quite different from classical systems, for which estimating the initial steplength has always been difficult ([15]).

5.5. **Economic Interpretation of Results.** A word should be said about the economic implications of the results just described. As we have noted, the whole purpose of modeling the dynamics of games is to aid with equilibrium selection. Consider again the duopoly model with 7 prices. The following is an equilibrium that we found using the random-start algorithm.

\[
x_s = \begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\
0 & .8389 & 0 & 0 & 0 & 0 & 0 & .1611
\end{bmatrix},
\]

\[
x_r = \begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}.
\]

But this equilibrium is not very stable. It is not unstable, in that a trajectory which starts nearby remains so. However, its region of attraction is not large: in choosing 10 starting points at random, the
PDS was not drawn to this point even once. Instead, every time the system converged to an equilibrium of the form

\[ x_s = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \]

\[ x_r = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & -1 \\ 0 & \beta_1 & 0 & 0 & \beta_2 & \beta_3 & \beta_4 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \]

with \( \beta_1, \beta_3, \beta_4 > 0 \).

Thus an equilibrium of this form is truly stable and might constitute a reasonable "solution" to this problem. Of course, not all problems will have equilibria which are as stable. Indeed, looking at the table of results, it is apparent that as the price grid becomes denser and denser, the equilibria tend to become less stable. This is not surprising since as the number of variables goes up, the function \( F \) which drives the PDS becomes more nonlinear. But this does raise a modeling issue. As many economic models involve discretizing a quantity such as price, a consequence of what we have discussed is that the level of discretization can fundamentally affect the behavior of the game. This will have to be the subject of future research. Future research will also be able to build on the positive results we have presented here. The duopoly model is a problem with cubic equations, so we have demonstrated that we should be comfortable handling this degree of nonlinearity. This covers a large class of games- not only all two-player stochastic games, but any game in which no more than two players move simultaneously (since the number of players moving simultaneously at any time is what determines the degree of the polynomials in (6)). Many markets have one competitor (or perhaps two) set the trend, with others following, and our results should be applicable to all these markets.

5.6. A Further Example. In order to push the envelope further, we attempted to solve a game with three players (involving quartic equations). We built a model of a dynamic auction with three identical (or symmetric) bidders. In each round the individual auction is a standard, first-price (low-price wins) sealed-bid auction. Two features distinguish the model from the standard kind. One is that the winner of an auction is prohibited by capacity constraints from competing in the following auction. The other is that submitting a bid (even a losing bid) entails a transaction cost. A firm can therefore elect not to submit a bid, and
avoid this cost. Firms do not know if rivals are submitting a bid or not, until after all bids have been submitted. The firms’ individual cost structure is of the usual type—each firm knows its own cost, but only knows that the other firms’ costs are drawn from an identical uniform distribution. Costs in one time-period are independent of costs in another. Note that this is a model with incomplete-information (as firms don’t know each other’s costs), while the stochastic game model described in section 2 assumed complete information. But the same results and ideas go through straightforwardly to the more general case.

In order to solve this problem numerically (as far as we know, no analytic solutions exist, which is how auctions are usually analyzed), we must of course discretize the state and action space. That is, firms’ costs must be derived from a discrete probability space, and they must choose their bids from a finite set of prices. One issue might be justifying the imposition of a maximum price, but this is not an unreasonable assumption, as the bid-taker will presumably not want to accept a bid that is above some threshold, and in a repeated-auction environment, the bidders should get to know what that threshold is. Discrete spaces imply the possibility of a tie, so all ties will be broken at random. A tie in our model may be thought of as akin to a situation in the real world in which two bids are very close to each other, in which case who actually wins is essentially a random result. This analogy, we believe, makes the random tiebreaking rule reasonable. We are now ready to formally describe the model.

As usual, let $S$ be the state (i.e. cost) space for each firm. Let action (price space) be $0,...,n$. with $0$ representing no bid. As some auctions may have no winner (if all three firms elect not to bid), some auctions will have two potential bidders, and some three. The payoff function for a firm in a two-bidder auction is

$$
\Pi_2(s,i,j) = \begin{cases} 
0 & i = 0 \\
-c & i > 0, j > 0, i > j \\
-i - s - c & i > 0, i < j \text{ or } j = 0 \\
i & i > 0, i = j
\end{cases},
$$

where $s$ is its cost, $i$ is its bid, and $j$ is its rival’s bid, and $c$ is the transaction cost (which is the same for all firms in all states).

The payoff function for a firm in a three-bidder auction is
\[
\Pi_3(s, i, j, k) = \begin{cases} 
0 & i = 0 \\
-c & i > 0, i > j, j > 0 \\
& \text{or } i > k, k > 0 \\
i - s - c & i > 0, i < j \text{ if } j > 0, \\
& \text{and } i < k \text{ if } k > 0 \\
i s - c & i > 0, i = j, i = k \\
i s - c & \text{otherwise}
\end{cases}
\]

where again \( s \) is its cost, \( i \) is its bid, and \( j \) and \( k \) its rivals’ bids.

Let \( x_{2si} \) and \( x_{3si} \) be the decision variables (i.e. the probability that a bidder in state \( s \) chooses action \( i \) in a two-bidder and three-bidder auction, respectively). Then the value functions, \( v_2(s) \), \( v_3(s) \) (for a bidder in state \( s \) in a two-bidder and three-bidder auction, respectively) satisfy the relations

\[
v_2(s) = \sum_{i \in A} x_{2si} \sum_{t \in S} p_t \sum_{j \in A} x_{2tj} \cdot (\Pi_2(s, i, j)) + \\ \left\{ \begin{array}{l}
\delta \sum_{r \in S} p_r v_2(r) \\
\delta \sum_{r \in S} p_r v_3(r) \\
\delta^2 (\beta \sum_{r \in S} p_r v_3(r) + (1 - \beta) \sum_{r \in S} p_r v_2(r)) \\
\frac{1}{2} [\delta (\sum_{r \in S} p_r v_2(r) + \delta^2 (\beta \sum_{r \in S} p_r v_3(r)) + (1 - \beta) \sum_{r \in S} p_r v_2(r))] \\
\end{array} \right. \\
\]

\[
v_3(s) = \sum_{i \in A} x_{3si} \sum_{t \in S} \sum_{r \in S} p_t p_r \sum_{j \in A} \sum_{k \in A} x_{3tjk} x_{3rk} \cdot (\Pi_3(s, i, j, k)) + \\ \left\{ \begin{array}{l}
\delta \sum_{u \in S} p_u v_2(u) \\
\delta \sum_{u \in S} p_u v_3(u) \\
\delta^2 (\beta \sum_{u \in S} p_u v_3(u) + (1 - \beta) \sum_{u \in S} p_u v_2(u)) \\
\frac{2}{\delta} \delta^2 (\sum_{u \in S} p_u v_2(u) + \frac{1}{2} \delta^2 (\beta \sum_{u \in S} p_u v_3(u) + (1 - \beta) \sum_{u \in S} p_u v_2(u)) \\
\frac{1}{2} \delta (\sum_{u \in S} p_u v_2(u) + \delta^2 (\beta \sum_{u \in S} p_u v_3(u) + (1 - \beta) \sum_{u \in S} p_u v_2(u))] \\
\end{array} \right. \\
\]

\( p_s = \frac{1}{n} \) is the probability of state \( s \) occurring (recall that costs are drawn from a uniform distribution), \( \beta = (\sum_{s \in S} p_s v_2(s))^2 \), i.e. the probability that no one will bid in a two-bidder auction, and \( \delta \) is of course the discount rate. Now the variational inequality (6) can be derived as
usual. Note that the quartic equations come into the picture because all three bidders can sometimes bid simultaneously, but in fact, the way we have modeled it, since $\beta$ is a variable, there are actually fifth-degree or quintic equations here. They could be reduced to quartic by writing the model differently, but we decided to leave the model this way, to provide a real test for our methods.

We attempted to solve the problem for the cost space $\{5, 10, 15\}$, and the price space of integers 1...20, resulting in 60 variables. Applying the random-start algorithm, we got a solution in 4 starts. The hybrid method also worked quite nicely, producing a solution from the first starting point tried, after 25 iterations. But clearly the presence of quintic equations makes the problem much more nonlinear and makes even small models a challenge.

5.7. Comparison to Other Solvers. For comparison’s sake, we also ran the PATH solver on the models presented above. PATH, of course, exclusively solves pure complementarity problems. But the Nash Equilibrium problem can always be written in such a form, as is done in [17]. For all the models tried, PATH failed to converge from any of ten different starting points, reaching its iteration limit each time. This includes the simpler version of the duopoly model, which is analyzed in [18] and [19]. This model only has quadratic equations (the full model we have analyzed has cubic equations) and we restricted the model’s size to 7 prices. Nevertheless, PATH again reached its iteration limit each time. This confirms past experience that PATH is not effective at solving such highly nonlinear complementarity problems.

6. Conclusion

In this paper we have proposed two new methods for computing Nash equilibria of stochastic games, one a random-start algorithm based on nonlinear programming, the other a hybrid method combining a dynamical system with nonlinear programming to find stable Nash equilibria. The random-start method is, we believe, the first globally convergent method proposed for finding Nash equilibria which is also practical to implement. The hybrid method takes a step towards addressing the major concern with most dynamics suggested for games in the past, namely that convergence to equilibria is often rare. Promising numerical results have been presented which show the potential of the two approaches. Future work will involve developing more efficient sampling techniques for the random-start method, while studying the effect of model size on the behavior of the dynamical system which drives the hybrid method. But also, we plan to take advantage of these tools to
build new, realistic models for applications such as auctions and real options, of which the auction model presented in the last section is an example.

7. ACKNOWLEDGEMENT

Research of the first author was supported by NSF grant DMS-9870317 and ONR grant N00014-98-1-0036. Research of the second author was supported by NSF grant DMS-0107450.

REFERENCES