Differential Dynamic Programming for Nonlinear Dynamic Games

Bolei Di\textsuperscript{1} and Andrew Lamperski\textsuperscript{2}

Abstract—Dynamic games arise when multiple agents with differing objectives choose control inputs to a dynamic system. Dynamic games model a wide variety of applications in economics, defense, and energy systems. However, compared to single-agent control problems, the computational methods for dynamic games are relatively limited. As in the single-agent case, only very specialized dynamic games can be solved exactly, and so approximation algorithms are required. This paper extends the differential dynamic programming algorithm from single-agent control to the case of non-zero sum full-information dynamic games. The method works by computing quadratic approximations to the dynamic programming equations. The approximation results in static quadratic games which are solved recursively. Convergence is proved by showing that the algorithm iterates sufficiently close to iterates of Newton’s method to inherit its convergence properties. A numerical example is provided.

I. INTRODUCTION

Dynamic games arise when multiple agents with differing objectives act upon a dynamic system. In contrast, optimal control can be viewed as the specialization of dynamic games to the case of a single agent. Dynamic games have many applications including pursuit-evasion [1], active-defense [2], [3], economics [4] and the smart grid [5]. Despite a wide array of applications, the computational methods for dynamic games are considerably less developed than the single-agent case of optimal control.

This paper shows how the differential dynamic programming (DDP) method from optimal control [6] extends to discrete-time non-zero sum dynamic games. Closely related works from [7], [8] focus on the case of zero-sum dynamic games. Classical differential dynamic programming operates by iteratively solving quadratic approximations to the Bellman equation from optimal control. Our method applies similar methods to the generalization of the Bellman equation for dynamic games [9]. Here, at each stage, the algorithm solves a static game formed by taking quadratic approximations to the value function of each agent. We show that the algorithm converges quadratically in the neighborhood of a strict Nash equilibrium. To prove convergence, we extend arguments from [10], which relate DDP iterates to those of Newton’s method, to the case of dynamic games. In particular, we extend the recursive solution for Newton’s method [11] to dynamic games, and demonstrate that the solutions produced Newton’s method and the DDP method are close.

A. Related Work

A great deal of work on algorithmic solutions to dynamic games has been done. This subsection reviews related work which is a bit more removed from the closer references described above. As we will see, most works solve somewhat different problems compared to the current paper.

Methods for finding Nash equilibria via extremum seeking were presented in [12], [13], [14]. In particular, the controllers drive the states of a dynamic system to Nash equilibria of static games. A related method for linear quadratic games was presented in [15]. For these works, each agent only requires measurements of its own cost. However, it is limited to finding equilibria in steady state. Our method requires each agent to have explicit model information, but gives equilibria over finite horizons. This is particularly important for games in which trajectories from initial to final states are desired.

Several works focus on the solution to dynamic potential games. Potential games are more tractable than general dynamic games, as they can be solved using methods from single-agent optimal control [16], [17], [18], [19], [20], [21]. However, potential games satisfy restrictive symmetry conditions. In particular, the assumption precludes interesting applications with heterogeneous agents.

B. Paper Outline

The general problem is formulated in Section II. The algorithm is described in Section III and the convergence proof is sketched in Section IV. A numerical example is described in V. Conclusions and future directions are discussed in VI while the proof details are given in the appendix.

II. DETERMINISTIC NONLINEAR DYNAMIC GAME PROBLEM

In this section, we introduce deterministic finite-horizon nonlinear game problem, the notations for the paper, the solution concept and convergence criterion of our proposed method.

The main problem of interest is a deterministic full-information dynamic game of the form below.

Problem 1: Nonlinear dynamic game

Each player tries to minimize their own cost

\[
J_n(u) = \sum_{k=0}^{T} c_{n,k}(x_k, u_{-k})
\] (1)

Subject to constraints

\[
x_{k+1} = f_k(x_k, u_{-k})
\] (2a)

\[x_0 \text{ is fixed.}
\] (2b)
Here, the state of the system at time $k$ is denoted by $x_k \in \mathbb{R}^{n_s}$. Player $n$’s input at time $k$ is given by $u_{n,k} \in \mathbb{R}^{n_a}$. The vector of all player actions at time $k$ is denoted by: $u_{k} = [u_{1,k}, u_{2,k}, \ldots, u_{N,k}]^\top \in \mathbb{R}^{n_a}$. The cost for player $n$ at time $k$ is $c_{n,k}(x_k, u_{n,k})$. This encodes the fact that the costs for each player can depend on the actions of all the players.

In later analysis, some other notation will be helpful. The vector player $n$’s actions over all time is denoted by $u_{n,:} = [u_{n,0}, u_{n,1}, \ldots, u_{n,T}]^\top$. The vector of all actions other than those of player $n$ is denoted by $u_{-n,:} = [u_{1,:}, \ldots, u_{1,1}, u_{n,1}, u_{2,:}, \ldots, u_{N,1}]^\top$. The vector of all states is denoted by $x = [x_0^\top, \ldots, x_T^\top]$ while the vector of all inputs is given by $u = [u_{1,:}^\top, \ldots, u_{N,:}^\top]^\top$.

Note that since the initial state is fixed and the dynamics are deterministic, the costs for each player can be expressed as functions of the vector of actions, $J_n(u)$.

A local Nash equilibrium for problem \( \text{I} \) is a set of inputs $u^*$ such that

$$J_n(u_{n,:}, u_{n,:}^*) \geq J_n(u^*), \quad n = 1, 2, \ldots, N$$

for all $u_{n,:}$ in a neighborhood of $u_{n,:}^*$. In the context of dynamic games, this corresponds to an open-loop, local Nash equilibrium [9]. The equilibrium is called a strict local Nash equilibrium if the inequality in (3) is strict for all $u_{n,:} \neq u_{n,:}^*$ in a neighborhood of $u_{n,:}^*$.

In this paper, we focus on computing Nash equilibria by solving the following necessary conditions for local Nash equilibria:

**Problem 2: Necessary conditions**

$$\frac{\partial J_n}{\partial u_{n,:}} = 0$$

for $n = 1, \ldots, N$.

For convenient notation, we stack all of the gradient vectors from (4) into a single vector:

$$\mathcal{J}(u) = \left[ \frac{\partial J_1}{\partial u_{1,:}} \frac{\partial J_2}{\partial u_{2,:}} \ldots \frac{\partial J_N}{\partial u_{N,:}} \right]^\top.$$ (5)

Thus, the necessary condition is equivalent to $\mathcal{J}(u) = 0$. Such conditions arise in works such as [22], [23].

We will present a method for solving these necessary conditions for a local Nash equilibrium via differential dynamic programming (DDP). In principle, an input vector satisfying the necessary conditions $\mathcal{J}(u) = 0$ could be found via Newton’s method. Similar to the single-player case from [10], we analyze the convergence properties of DDP by proving that its solution is close to that computed by Newton’s method.

To guarantee convergence, we assume that $\mathcal{J}(u)$ satisfies the smoothness and non-degeneracy conditions required by Newton’s method [24]. For smoothness, we assume that $\mathcal{J}(u)$ is differentiable with locally Lipschitz derivatives. For non-degeneracy, we assume that $\frac{\partial \mathcal{J}(u)}{\partial u}$ is invertible. A sufficient condition for the smoothness assumptions is that the functions $f_k$ and $c_{n,k}$ are twice continuously differentiable with Lipschitz second derivatives. In our DDP solution, we will solve a sequence of stage-wise quadratic games. As we will see, a sufficient condition for invertibility of $\frac{\partial \mathcal{J}(u)}{\partial u}$ is the unique solvability of the stage-wise games near the equilibrium.

### III. Differential Dynamic Programming Algorithm

This section describes the differential dynamic programming algorithm for dynamic games of the form in Problem \( \text{I} \). Subsection III-A gives a high-level description of the algorithm, while Subsection III-B describes the explicit matrix calculations used in the algorithm.

**A. Algorithm Overview**

The equilibrium solution to the general dynamic game can be characterized by the Bellman recursion:

$$V_{n,T+1}(x_{T+1}) = 0$$

$$Q_{n,k}(x_k, u_{n,k}) = c_{n,k}(x_k, u_{n,k}) + V_{n,k+1}(f_k(x_k, u_{n,k}))$$

$$V_{n,k}(x_k) = \min_{u_{n,k}} Q_{n,k}(x_k, u_{n,k}).$$

In particular, if a solution to the Bellman recursion is found, the corresponding optimal strategy for player $n$ at time $k$ would be the $u_{n,k}$ which minimizes $Q_{n,k}(x_k, u_{n,k})$. Note that (6c) defines a static game with respect to the $u_{1,k}$ variable at step $k$.

The idea of the differentiable dynamic programming (DDP) is to maintain quadratic approximations of $V_{n,k}$ and $Q_{n,k}$ denoted by $\hat{V}_{n,k}$ and $\hat{Q}_{n,k}$, respectively.

We need some notation for our approximations. For a scalar-valued function, $h(z)$, we denote the quadratic approximation near $\bar{z}$ by:

$$\text{quad}(h(z))\bar{z} = \frac{1}{2} \left[ \begin{array}{c} 1 \\ \delta z \end{array} \right]^\top \left[ \begin{array}{cc} h(\bar{z}) & \partial h(\bar{z})^\top \\ \partial h(\bar{z}) & \partial^2 h(\bar{z})^\top \end{array} \right] \left[ \begin{array}{c} 1 \\ \delta z \end{array} \right]$$

(7a)

$$\delta z = z - \bar{z}.\quad (7b)$$

If $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$ we form the quadratic approximation by stacking all of the quadratic approximations of the entries:

$$\text{quad}(h(z))\bar{z} = [\text{quad}(h_1(z))\bar{z}, \ldots, \text{quad}(h_m(z))\bar{z}]^\top$$

(8)

Let

$$z_k = \left[ \begin{array}{c} x_k \\ u_{n,k} \end{array} \right]$$

(9)

and let $\bar{z}_k$ and $\bar{u}_{n,k}$ be a trajectory of states and actions satisfying the dynamic equations from (3). The approximate Bellman recursion around this trajectory is given by:

$$\hat{V}_{n,T+1}(x_{T+1}) = 0$$

$$\hat{Q}_{n,k}(z_k) = \text{quad}(c_{n,k}(z_k) + \hat{V}_{n,k+1}(f_{n,k}(z_k)))z_k$$

$$\hat{V}_{n,k}(x_k) = \min_{u_{n,k}} \hat{Q}_{n,k}(x_k, u_{n,k}).$$

(10a)

(10b)

(10c)

Note that (10c) is now a quadratic game in the $u_{n,k}$ variables which has unique and ready solution [9]. Recall the $\mathcal{J}(u)$ function defined in (5). A sufficient condition for solvability of these games is given in terms of $\mathcal{J}(u)$ is given in the following lemma. Its proof is in Appendix I-D.
Lemma 1: If \( \frac{\partial f(x)}{\partial u} \) is invertible, the game defined by (10c) has a unique solution of the form:

\[
  u_{t,k} = \bar{u}_{t,k} + \tilde{K}_k \delta x_k + \tilde{s}_k. 
\] (11)

In the notation defined above, we have that \( \delta x_k = x_k - \bar{x}_k \).

Note that if \( \frac{\partial f(x)}{\partial u} \) is invertible, then \( \frac{\partial f(x)}{\partial u} \) is invertible for all \( \bar{u} \) in a neighborhood of \( u^* \).

Here we provide the DDP algorithm for applying DDP game solution in pseudo code.

**Algorithm 1** Differential Dynamic Programming for Nonlinear Dynamic Games

Generate an initial trajectory \( \bar{x}, \bar{u} \)

**Backward Pass:**

Perform the approximate Bellman recursion from (10)

Compute \( \tilde{K}_k \) and \( \tilde{s}_k \) from (11).

**Forward Pass:**

Generate a new trajectory using the affine policy defined by \( \tilde{K}_k, \tilde{s}_k \)

end loop

**B. Implementation Details**

All of the operations in the backwards pass of the DDP algorithm, Algorithm 1, can be expressed more explicitly in terms of matrices.

To construct the required matrices, we define the following approximation terms:

\[
A_k = \left[ \frac{\partial f_k(x_k, u_{-k})}{\partial x_k} \right] \bigg|_{\bar{u}} \quad B_k = \left[ \frac{\partial f_k(x_k, u_{-k})}{\partial u_{-k}} \right] \bigg|_{\bar{u}} \] (12a)

\[
G_k = \left[ \frac{\partial^2 f_k(x_k, u_{-k})}{\partial x_l \partial u_{-k}} \right] \bigg|_{\bar{u}} \quad \tilde{G}_k = \left[ \frac{\partial^2 f_k(x_k, u_{-k})}{\partial x_l \partial x_k} \right] \bigg|_{\bar{u}} \quad \tilde{G}_k^\top = 
\] (12b)

\[
R_k(\delta x_k, \delta u_{-k}) = \begin{bmatrix}
\delta x_k^\top & \delta x_k^\top & \cdots & \delta x_k^\top \\
\delta x_k^\top & \delta x_k^\top & \cdots & \delta x_k^\top \\
\vdots & \vdots & \ddots & \vdots \\
\delta x_k^\top & \delta x_k^\top & \cdots & \delta x_k^\top \\
\delta u_{-k}^\top & \delta u_{-k}^\top & \cdots & \delta u_{-k}^\top \\
\tilde{G}_k^\top & \tilde{G}_k^\top & \cdots & \tilde{G}_k^\top \\
\tdots & \tdots & \ddots & \tdots \\
\tilde{G}_k^\top & \tilde{G}_k^\top & \cdots & \tilde{G}_k^\top \\
\tilde{G}_k & \tilde{G}_k & \cdots & \tilde{G}_k \\
\end{bmatrix} 
\] (12c)

\[
M_{n,k} = \left[ \begin{array}{c}
\frac{\partial c_{n,k}}{\partial x_k} \\
\frac{\partial c_{n,k}}{\partial x_{-k}} \\
\frac{\partial c_{n,k}}{\partial u_{-k}} \\
\frac{\partial c_{n,k}}{\partial u_{-k}} \\
\frac{\partial c_{n,k}}{\partial u_{-k}} \\
\frac{\partial c_{n,k}}{\partial x_{-k}} \\
\frac{\partial c_{n,k}}{\partial x_{-k}} \\
\frac{\partial c_{n,k}}{\partial x_{-k}} \\
\frac{\partial c_{n,k}}{\partial x_{-k}} \\
\frac{\partial c_{n,k}}{\partial x_{-k}} \\
\frac{\partial c_{n,k}}{\partial u_{-k}} \\
\frac{\partial c_{n,k}}{\partial u_{-k}} \\
\frac{\partial c_{n,k}}{\partial u_{-k}} \\
\end{array} \right] \bigg|_{\bar{u}}
\] (12d)

Using the notation from (7), (8) and (9), the second-order approximations of the dynamics and cost are given by:

\[
\text{quad}(f_k(z_k))_{z_k} = f_k(z_k) + \Delta_k \delta x_k + B_{k \delta u_{-k}} + R_k(\delta z_k) \] (13a)

\[
\text{quad}(c_{n,k}(z_k))_{z_k} = \left[ \begin{array}{c}
1 \\
\delta z_k \\
\end{array} \right] \begin{bmatrix}
M_{n,k} & 1 \\
\delta Z_k & 1 \\
\end{bmatrix}. \] (13b)

By construction \( \tilde{V}_{n,k}(x_k) \) and \( \tilde{Q}_{n,k}(x_k, u_{-k}) \) are quadratic, and so there must be matrices \( S_{n,k} \) and \( \Gamma_{n,k} \) such that:

\[
\tilde{V}_{n,k}(x_k) = \frac{1}{2} \left[ \begin{array}{c}
\delta x_k \\
\delta x_k \\
\end{array} \right] \begin{bmatrix}
\tilde{S}_{n,k} & \tilde{S}_{n,k} \\
\tilde{S}_{n,k} & \tilde{S}_{n,k} \\
\end{bmatrix} \begin{bmatrix}
\delta x_k \\
\delta x_k \\
\end{bmatrix}. \] (14a)

\[
\tilde{Q}_{n,k}(x_k, u_{-k}) = \frac{1}{2} \left[ \begin{array}{c}
\delta x_k \\
\delta x_k \\
\end{array} \right] \begin{bmatrix}
\Gamma_{n,k} & \Gamma_{n,k} \\
\Gamma_{n,k} & \Gamma_{n,k} \\
\end{bmatrix} \begin{bmatrix}
\delta x_k \\
\delta x_k \\
\end{bmatrix}. \] (14b)

**Lemma 2:** The matrices in (14) are defined recursively by:

\[
D_{n,k} = \sum_{l=1}^{n_l} S_{n,k+l} G^l_{n,k} \] (15a)

\[
\tilde{S}_{n,k} = M_{n,k} \] (15b)

\[
\tilde{S}_{n,k} = \begin{bmatrix}
\tilde{G}_{n,k}^1 & \tilde{G}_{n,k}^1 & \cdots & \tilde{G}_{n,k}^1 \\
\tilde{G}_{n,k}^2 & \tilde{G}_{n,k}^2 & \cdots & \tilde{G}_{n,k}^2 \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{G}_{n,k}^{n_l} & \tilde{G}_{n,k}^{n_l} & \cdots & \tilde{G}_{n,k}^{n_l} \\
\end{bmatrix} \] (15c)

\[
\tilde{F}_k = \begin{bmatrix}
\tilde{R}_{1,k} & \tilde{R}_{1,k} & \cdots & \tilde{R}_{1,k} \\
\tilde{F}_{1,k} & \tilde{F}_{1,k} & \cdots & \tilde{F}_{1,k} \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{F}_{n_l,k} & \tilde{F}_{n_l,k} & \cdots & \tilde{F}_{n_l,k} \\
\end{bmatrix} \] (15d)

\[
\tilde{P}_k = \begin{bmatrix}
\tilde{F}_{1,k} & \tilde{F}_{1,k} & \cdots & \tilde{F}_{1,k} \\
\tilde{F}_{1,k} & \tilde{F}_{1,k} & \cdots & \tilde{F}_{1,k} \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{F}_{n_l,k} & \tilde{F}_{n_l,k} & \cdots & \tilde{F}_{n_l,k} \\
\end{bmatrix} \] (15e)

\[
\tilde{S}_{n,k} = -\tilde{F}_{k}^{-1} \tilde{P}_k, \quad \tilde{K}_k = -\tilde{F}_{k}^{-1} \tilde{P}_k \] (15f)

\[
\tilde{S}_{n,k} = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 \\
\end{bmatrix} \] (15g)

for \( k = T, T-1, \ldots, 0 \).

**Proof:** By construction we must have \( \tilde{S}_{n,T+1} = 0 \).

Plugging (13a) into (15b) and dropping all cubic and higher terms gives (10b). Since \( u_{-k} = \bar{u}_{-k} + \delta u_{-k} \) and \( \bar{u}_{-k} \) is constant, the static game defined in (10c) can be solved in the \( \delta u_{-k} \) variables. Differentiating (14b) by \( \delta u_{-k} \), collecting the derivatives for all players and setting them to zero leads to...
Additionally, we will assume that
\[ \dot{F}_k \delta u_{t,k} + \dot{F}_k \delta x_k + \dot{H}_k = 0. \] (16)

Thus, the matrices for the equilibrium strategy are given in [15]. Plugging (11) into (14b) leads to (15g).

**Remark 1:** The next section will describe how the algorithm converges to strict Nash equilibria if it begins sufficiently close. To ensure that the algorithm converges regardless of initial condition, a Levenberg-Marquardt style regularization can be employed. Such regularization has been used in centralized DDP algorithms, [25], [26], to ensure that the required inverses exist and that the solution improves. In the current setting, such regularization would correspond to using a regularization of the form \( \dot{F}_k + \lambda I \) where \( \lambda \geq 0 \) is chosen sufficiently large to ensure that the matrix is positive definite.

### IV. CONVERGENCE

This section outlines the convergence behavior of the DDP algorithm for dynamic games. The main result is Theorem 1 which demonstrates quadratic convergence to local Nash equilibria:

**Theorem 1:** If \( u^* \) is a strict local equilibrium such that \( \frac{\partial J(u^*)}{\partial u} \) is invertible, then the DDP algorithms converges locally to \( u^* \) at a quadratic rate.

The proof depends on several intermediate results. Subsection IV-A reformulates Newton’s method for the necessary conditions, [4], as the solution to a dynamic game. Subsection IV-B demonstrates that the solutions of the dynamic games solved by Newton’s method and DDP close. Then, Subsection IV-C finishes the convergence proof by demonstrating that the DDP solution is sufficiently close to the Newton solution to inherit its convergence property.

Throughout this section we will assume that both Newton’s method and DDP are starting from the same initial action trajectory, \( \tilde{u} \). Let \( u^N \) and \( u^D \) be the updated action trajectories of Newton’s method and DDP, respectively. Define update steps, \( \delta u^N \) and \( \delta u^D \), by:

\[ u^N = \tilde{u} + \delta u^N \quad u^D = \tilde{u} + \delta u^D. \] (17)

Additionally, we will assume that \( u^* \) is a strict local equilibrium with \( \frac{\partial J(u^*)}{\partial u} \) invertible.

#### A. Dynamic Programming Solution for the Newton Step

The proof of Theorem 1 proceeds by demonstrating that the solutions from DDP and Newton’s method are sufficiently close that DDP inherits the quadratic convergence of Newton’s method. To show closeness, we demonstrate that the Newton step can be interpreted as the solution to a dynamic game. This dynamic game has a recursive solution that is structurally similar to the recursions from DDP. This subsection derives the corresponding game and solution.

The Newton step for solving (4) is given by:

\[ \frac{\partial J(\tilde{u})}{\partial u} \delta u^N = -J(\tilde{u}). \] (18)

This rule leads to a quadratic convergence to a root in (4) whenever \( V_{\tilde{u}}J(\tilde{u}) \) is locally Lipschitz and invertible [24]. The next two lemmas give game-theoretic interpretations of the Newton step.

**Lemma 3:** Solving (18) is equivalent to solving the quadratic game defined by:

\[ \min_{\delta u_{t}} \frac{1}{2} \sum_{k=0}^{T} \left( \frac{1}{\delta x_k \partial u_{t,k}} \right)^T M_{n,k} \left( \frac{1}{\delta x_k \partial u_{t,k}} \right) + M_{n,k}^{\delta} \Delta x_k \] (20a)

subject to

\[ \delta x_0 = 0 \] (20b)
\[ \Delta x_0 = 0 \] (20c)
\[ \delta x_{t+1} = A_t \delta x_t + B_t \delta u_{t,k} \] (20d)
\[ \Delta x_{t+1} = A_t \Delta x_t + R_t (\delta x_t, \delta u_{t,k}) \] (20e)
\[ k = 0, 1, \ldots, T \] (20f)

Note that the states of the dynamic game are given by \( \delta x_k \) and \( \Delta x_k \) as

\[ \delta x_k = \sum_{l=0}^{T} \frac{\partial \delta x_l}{\partial u_{t,l}} \bigg|_{\tilde{u},\tilde{u}} \delta u_{t,l} \] (21a)
\[ \Delta x_k = \sum_{l=0}^{T} \sum_{j=0}^{l} \frac{\partial^2 \delta x_l}{\partial u_{t,j} \partial u_{t,l}} \bigg|_{\tilde{u},\tilde{u}} \delta u_{t,j}, l = 1, 2, \ldots, n \] (21b)

It follows that the equilibrium solution of this dynamic game is characterized by the following Bellman recursion:

\[ V_{n,T+1}(\delta x_{T+1}, \Delta x_{T+1}) = 0 \] (22a)
\[ Q_{n,k}(\delta x_k, \Delta x_k, \delta u_{t,k}) = \frac{1}{2} \left( \frac{\partial \delta x_k}{\partial u_{t,k}} \right)^T M_{n,k} \left( \frac{\partial \delta x_k}{\partial u_{t,k}} \right) + M_{n,k}^{\delta} \Delta x_k \] (22b)
\[ + V_{n,k+1}(A_t \delta x_t + B_t \delta u_{t,k}, A_t \Delta x_t, R_t (\delta x_t, \delta u_{t,k})) \] (22c)
\[ V_{n,k}(\delta x_k, \Delta x_k) = \min_{\delta u_{t,k}} Q_{n,k}(\delta x_k, \Delta x_k, \delta u_{t,k}) \] (22d)

Note that (22d) defines a static quadratic game and \( V_{n,k}(\delta x_k, \Delta x_k) \) is found by solving the game and substituting the solution back to \( Q_{n,k}(\delta x_k, \Delta x_k, \delta u_{t,k}) \).

The next lemma describes an explicit solution to the backward recursion (22). The key step in the convergence proof is showing that the matrices used in this recursion are appropriately close to the matrices used in DDP.
Lemma 5: The functions $V_{n,k}$ and $Q_{n,k}$ can be expressed as

$$V_{n,k}(\delta x_k, \Delta x_k) = \frac{1}{2} \left( \begin{pmatrix} 1 \\ \delta x_k \end{pmatrix}^T S_{n,k} \begin{pmatrix} 1 \\ \delta x_k \end{pmatrix} + \Omega_{n,k} \Delta x_k \right)$$  \hfill (23a)

$$Q_{n,k}(\delta x_k, \Delta x_k, \delta u_k) = \frac{1}{2} \left( \begin{pmatrix} 1 \\ \delta x_k \end{pmatrix}^T \Gamma_{n,k} \begin{pmatrix} 1 \\ \delta u_k \end{pmatrix} + \Omega_{n,k} \Delta x_k \right)$$  \hfill (23b)

where the matrices $S_{n,k}$, $\Gamma_{n,k}$, and $\Omega_{n,k}$ are defined recursively by $S_{n,T+1} = 0$, $\Omega_{n,T+1} = 0$, and

$$\Omega_{n,k} = M_{n,k}^T + \Omega_{n,k+1} A_k$$  \hfill (24a)

$$D_{n,k} = \sum_{i=k}^{T} \Omega_{i,k+1} G_i$$  \hfill (24b)

$$\Gamma_{n,k} = M_{n,k}$$

$$\begin{bmatrix}
S_{n,k+1}^{11} \\
S_{n,k+1}^{1u} \\
S_{n,k}^{1u} \\
\vdots \\
S_{n,k}^{u,u}
\end{bmatrix} =
\begin{bmatrix}
\Gamma_{n,k}^{11} & \Gamma_{n,k}^{1u} & \Gamma_{n,k}^{12} & \cdots & \Gamma_{n,k}^{1u} \\
\Gamma_{n,k}^{1u} & \Gamma_{n,k}^{uu} & \Gamma_{n,k}^{1u} & \cdots & \Gamma_{n,k}^{1u} \\
\Gamma_{n,k}^{1u} & \Gamma_{n,k}^{1u} & \Gamma_{n,k}^{uu} & \cdots & \Gamma_{n,k}^{1u} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\Gamma_{n,k}^{u,u} & \Gamma_{n,k}^{u,u} & \Gamma_{n,k}^{u,u} & \cdots & \Gamma_{n,k}^{uu}
\end{bmatrix}
$$  \hfill (24c)

$$F_k =
\begin{bmatrix}
\Gamma_{n,k}^{1u} & \Gamma_{n,k}^{1u} & \cdots & \Gamma_{n,k}^{1u} \\
\Gamma_{n,k}^{1u} & \Gamma_{n,k}^{uu} & \cdots & \Gamma_{n,k}^{1u} \\
\vdots & \vdots & \ddots & \vdots \\
\Gamma_{n,k}^{u,u} & \cdots & \Gamma_{n,k}^{uu}
\end{bmatrix},
H_k =
\begin{bmatrix}
\Gamma_{n,k}^{1u} \\
\vdots \\
\Gamma_{n,k}^{u,u}
\end{bmatrix}
$$  \hfill (24d)

$$s_k = -F_k^{-1} H_k, \quad K_k = -F_k^{-1} P_k$$  \hfill (24g)

$$S_{n,k} =
\begin{bmatrix}
1 & 0 \\
0 & K_k^T
\end{bmatrix} \Gamma_{n,k}
\begin{bmatrix}
1 & 0 \\
0 & K_k
\end{bmatrix}$$  \hfill (24h)

for $k = T, T-1, \ldots, 0$.

From this lemma, we can see that the matrices used in the recursions for both DDP and Newton’s method are very similar in structure. Indeed, the iterations are identical aside from the definitions of the $D_{n,k}$ and $\Gamma_{n,k}$ matrices.

Proof: The proof is very similar to the proof for Lemma 2. Solving the equilibrium strategy and $V_{n,k}(\delta x_k, \Delta x_k)$ based on $Q_{n,k}(\delta x_k, \Delta x_k, \delta u_k)$ is the same as how we arrived at (10) and (15), since the extra terms of $\Delta x_k$ are not coupled with $\delta u_k$ and other terms are of the exact same form. The stepping back in time of $Q_{n,k}(\delta x_k, \Delta x_k, \delta u_k)$ is achieved by substituting (20d) and (20k) into (23a), which is slightly different because of the extra terms related to $\Delta x_k$.

$$Q_{n,k}(\delta x_k, \Delta x_k, \delta u_k) = \frac{1}{2} \left( \begin{pmatrix} 1 \\ \delta x_k \end{pmatrix}^T M_{n,k} \begin{pmatrix} 1 \\ \delta u_k \end{pmatrix} + M_{n,k}^T \Delta x_k \right)$$  \hfill (25a)

$$+ V_{n,k+1}(A_k \delta x_k + B_k \delta u_k, A_k \Delta x_k + B_k \Delta u_k) (25b)$$

$$Q_{n,k}(\delta x_k, \Delta x_k, \delta u_k) = \frac{1}{2} \left( \begin{pmatrix} 1 \\ \delta x_k \end{pmatrix}^T M_{n,k} \begin{pmatrix} 1 \\ \delta u_k \end{pmatrix} \right) +$$

$$\frac{1}{2} \left( \begin{pmatrix} 1 \\ \delta x_k \end{pmatrix}^T M_{n,k} \begin{pmatrix} 1 \\ \delta u_k \end{pmatrix} \right) +$$

$$\frac{1}{2} \left( \left( M_{n,k}^T + \Omega_{n,k+1} A_k \right) \Delta x_k \right)$$  \hfill (25c)

So (24a), (24b) and (24c) are true.

B. Closeness Lemmas

This subsection gives a few lemmas which imply that the Newton step, $\delta u^N$, and the DDP step, $\delta u^D$, are close. For the rest of the section, we set $\| \tilde{u} - u^* \| = \epsilon$.

The following lemma shows that the matrices used in the backwards recursion are close. It is proved in Appendix I-B.

Lemma 6: The matrices from the backwards recursions of DDP and Newton’s method are close in the following sense:

$$\hat{\Gamma}_{n,k} - \Gamma_{n,k} = \begin{bmatrix}
\Gamma_{n,k}^{11} - \hat{\Gamma}_{n,k}^{11} \\
\Gamma_{n,k}^{1u} - \hat{\Gamma}_{n,k}^{1u} \\
\vdots \\
\Gamma_{n,k}^{u,u} - \hat{\Gamma}_{n,k}^{u,u}
\end{bmatrix} = \begin{bmatrix}
O(\epsilon) & O(\epsilon^2) & O(\epsilon^3) \\
O(\epsilon^2) & O(\epsilon) & O(\epsilon) \\
O(\epsilon^3) & O(\epsilon) & O(\epsilon)
\end{bmatrix}$$  \hfill (26a)

$$\hat{S}_{n,k} - S_{n,k} = \begin{bmatrix}
\hat{S}_{n,k}^{11} - S_{n,k}^{11} \\
\hat{S}_{n,k}^{1u} - S_{n,k}^{1u} \\
\hat{S}_{n,k}^{u,u} - S_{n,k}^{u,u}
\end{bmatrix} = \begin{bmatrix}
O(\epsilon) & O(\epsilon^2) & O(\epsilon^3) \\
O(\epsilon^2) & O(\epsilon) & O(\epsilon) \\
O(\epsilon^3) & O(\epsilon) & O(\epsilon)
\end{bmatrix}.$$  \hfill (26b)

Furthermore, the following matrices are small:

$$\hat{\Gamma}_{n,k}^{1u} = \hat{\Gamma}_{n,k}^{1u} = O(\epsilon), \quad \hat{\Gamma}_{n,k}^{1u} = \hat{\Gamma}_{n,k}^{1u} = O(\epsilon)$$  \hfill (26c)

$$s_{n,k} = O(\epsilon), \quad \hat{s}_{n,k} = O(\epsilon).$$  \hfill (26d)

After showing that the matrices are close, it can be shown that the states and actions computed in the update steps are close. It is proved in Appendix I-C.
Lemma 7: The states and actions computed by DDP and Newton’s method are close:

\[ \delta u_k^N - \delta u_k^D = O(\epsilon^2) \]  
[27a]

\[ \delta x_k^N - \delta x_k^D = O(\epsilon^2) \]  
[27b]

Furthermore, the updates are small:

\[ \delta u_k^D = o(\epsilon), \quad \delta u_k^N = o(\epsilon) \]  
[27c]

\[ \delta x_k^D = o(\epsilon), \quad \delta x_k^N = o(\epsilon). \]  
[27d]

C. Proof of Theorem 7

Lemma 7 implies that \( \|\delta u^N - \delta u^D\| = O(\epsilon^2) \). Furthermore, the Newton step satisfies:

\[ \|\bar{u} + \delta u^N - u^*\| = O(\epsilon^2). \]  
[28]

See [24]. The proof of quadratic convergence is completed by the following steps:

\[ \|\bar{u} + \delta u^D - u^*\| = \|\bar{u} + \delta u^N - u^* + \delta u^D - \delta u^N\| \]  
[29a]

\[ \leq \|\bar{u} + \delta u^N - u^*\| + \|\delta u^D - \delta u^N\| \]  
[29b]

\[ = O(\epsilon^2). \]  
[29c]

V. Numerical Example

We apply the proposed DDP algorithm for deterministic nonlinear dynamic games to a toy examples in this section. The example is implemented in Python and all derivatives of nonlinear functions are computed via Tensorflow [27].

We consider a simple 1-D owner-dog problem, with horizon \( T = 11 \) and initial state \( x_{0,0} = [-1, 2] \) where the dynamics of both the owner and the dog are given respectively by

\[ x_{k+1}^0 = x_k^0 + \tanh u_k^0 \]  
[30a]

\[ x_{k+1}^1 = x_k^1 + \tanh u_k^1 \]  
[30b]

The owner cares about going to \( x^0 = 1 \) and that the dog can stay at \( x^1 = 2 \). The dog, however, only tries to catch up with the owner. Each player also concerns itself with the energy consumption, therefore has a cost term related to the magnitude of its input. Their cost functions are formulated as

\[ c_{0,k}(x, u) = 10 \text{ sigmoid}(x_k^0 - 1)^2 + 40(x_k^1 - 2)^2 + (u_k^0)^2 \]  
[31a]

\[ c_{1,k}(x, u) = \tanh(x_k^0 - x_k^1)^2 + (u_k^1)^2 \]  
[31b]

Nonlinear functions are added to the dynamics and costs to create a nonlinear game rather than for explicit physical meaning. We initialize a trajectory with zero input and initial state, i.e. \( \bar{u} = [0, 0, \ldots, 0] \) and \( \bar{x} = [-1, 2, -1, 2, -1, 2, \ldots, -1, 2] \). We used an identity regularization matrix with a magnitude of 400.

Fig. 1 shows the solution via DDP to this problem over iterations, where the more transparent the trajectories, the earlier in the iterations they are. The starred trajectory is the final equilibrium solution. We simulated 100 iterations after the initial trajectory and picked 10 uniformly spaced ones to show in the figures.

VI. Conclusion

In this paper we have shown how differential dynamic programming extends to dynamic games. The key steps were involved finding explicit forms for both DDP and Newton iterations that enable clean comparison of their solutions. We demonstrated the performance of the algorithm on a simple nonlinear dynamic game.

Many extensions are possible. We will examine larger examples and work on numerical scaling. Also of interest are stochastic dynamic games and problems in which agents have differing, imperfect information sets. Additionally, handling scenarios in which agents have imperfect model information will of great practical importance.

References

[1] I. Rusnak, “The lady, the bandits, and the bodyguards—a two team dynamic game,” in Proceedings of the 16th world IFAC congress, 2005, pp. 934–939.

[2] O. Prokopov and T. Shima, “Linear quadratic optimal cooperative strategies for active aircraft protection,” Journal of Guidance, Control, and Dynamics, vol. 36, no. 3, pp. 753–764, 2013.
We first derive a few useful results which we’ll use later in this chapter.

First, we bound the spectral radius of $F_k^{-1}$ from above

$$\rho(F_k^{-1}) \leq \hat{F}$$

where $\hat{F}$ is a constant. Consider inverting $\nabla_u J(u)$ in Newton’s method by successively eliminating $\delta u_{i,k}$ for $k = T, T - 1, \ldots, 0$. The $F_k$ matrices are exactly the matrices which would be inverted when eliminating $\delta u_{i,k}$. Since $\nabla_u J(u)$ is Lipschitz continuous, its eigenvalues are bounded away from zero in a neighborhood of $u^*$. It follows that the eigenvalues of $F_k$ must also be bounded away from zero and $F_k^{-1}$ is bounded above.

Secondly, we assert that

$$\Omega_{n,k} = \frac{\partial}{\partial x_k} \sum_{i=k}^{T} c_{n,i}(x_i, u_{i,j})$$

which means $\Omega_{n,k}$ captures the first-order effect of $x_k$ on the sum of all later costs for each player. $\Omega_{n,k}$ is constructed according to (24a). Equation (35) is true for $k = T$ by construction. We prove by induction and assume that (35) holds for $k + 1$, i.e. $\Omega_{n,k+1} = \frac{\partial}{\partial x_k} \sum_{i=k+1}^{T} c_{n,i}(x_i, u_{i,j})$, then

$$\Omega_{n,k} = M_{n,k} + \Omega_{n,k+1}A_k$$

where $M_{n,k}$ is the matrix of first-order coefficients and $\Omega_{n,k+1}$ is the matrix of second-order coefficients.

We use these results to prove that $J_n(u)$ is twice differentiable hence Lipschitz, i.e.

$$\left| J_n(u) \frac{\partial}{\partial u_{i,k}} - J_n(u) \frac{\partial}{\partial u_{j,k}} \right|^2 \leq \text{constant} \cdot |u_i - u_j|^2 = O(\varepsilon)$$
The first equality holds because

\[
\frac{\partial J(u)}{\partial u_{i,k}}\bigg|_{i,\bar{u}} = \frac{\partial}{\partial u_{i,k}} \sum_{i=0}^{T} c_{n,i}(x_i, u_{i,\bar{u}}) = \frac{\partial}{\partial u_{i,k}} \sum_{i=0}^{T} c_{n,i}(x_i, u_{i,\bar{u}})
\]

(37a)

\[
= \frac{\partial}{\partial u_{i,k}} \frac{\partial}{\partial x_i} c_{n,i}(x_i, u_{i,\bar{u}}) + \frac{\partial}{\partial u_{i,k}} \sum_{i=k+1}^{T} c_{n,i}(x_i, u_{i,\bar{u}})
\]

(37b)

\[
= M_{n,k}^{l} + \frac{\partial}{\partial x_i} c_{n,i}(x_i, u_{i,\bar{u}}) \bigg|_{i,\bar{u}} \sum_{i=0}^{T} c_{n,i}(x_i, u_{i,\bar{u}}) \frac{\partial x_{k+1}}{\partial u_{i,k}} \bigg|_{i,\bar{u}}
\]

(37c)

\[
= M_{n,k}^{l} + \Omega_{n,k+1} B_k
\]

(37d)

These results are used implicitly in the later proofs of lemmas.

A. Proof of Lemma 4

First, we prove that the dynamics constraints (20d) and (20e) are inductive definitions of the following approximation terms:

\[
\delta x_k = \sum_{i=0}^{T} \frac{\partial x_k}{\partial u_{i,j}} \bigg|_{i,\bar{u}} \delta u_{i,j}
\]

(38a)

\[
\Delta^l_k = \sum_{i=0}^{T} \sum_{j=0}^{T} \delta u_{i,j} \frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} \bigg|_{i,\bar{u}} \delta u_{j,i}, \ l = 1, 2, \ldots, n_k
\]

(38b)

Note that \(x_0\) is fixed so that (38) holds at \(k = 0\). Now we handle each of the terms inductively.

For \(\delta x_{k+1}\), we have

\[
\delta x_{k+1} = \sum_{i=0}^{T} \frac{\partial x_{k+1}}{\partial u_{i,j}} \bigg|_{i,\bar{u}} \delta u_{i,j}
\]

\[
= \sum_{i=0}^{T} \frac{\partial f_k(x_k, u_{i,k})}{\partial u_{i,j}} \bigg|_{i,\bar{u}} \delta u_{i,j}
\]

\[
= \frac{\partial f_k(x_k, u_{i,k})}{\partial x_k} \bigg|_{i,\bar{u}} \sum_{i=0}^{T} \frac{\partial x_k}{\partial u_{i,k}} \delta u_{i,k} + \frac{\partial f_k(x_k, u_{i,k})}{\partial u_{i,k}} \bigg|_{i,\bar{u}} \delta u_{i,k}
\]

\[
= A_k \delta x_k + B_k \delta u_{i,k}
\]

(39)

We used the fact that \(\frac{\partial f_k(x_k, u_{i,k})}{\partial u_{i,k}}\) is zero unless \(i = k\).

For \(\Delta_{k+1}^l\), row \(l\) is given by:

\[
\Delta_{k+1}^l = \sum_{i=0}^{T} \sum_{j=0}^{T} \delta u_{i,j} \frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} \bigg|_{i,\bar{u}} \delta u_{j,i}
\]

(40a)

\[
= \sum_{i=0}^{T} \sum_{j=0}^{T} \delta u_{i,j} \frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} + \frac{\partial x_k}{\partial u_{i,j}} \frac{\partial^2 f_l}{\partial x_k \partial u_{j,i}} \bigg|_{i,\bar{u}} \delta u_{j,i}
\]

(40b)

\[
\delta u_{i,j} \frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} \bigg|_{i,\bar{u}} + \delta x_k \frac{\partial^2 f_l}{\partial x_k \partial u_{j,i}} \bigg|_{i,\bar{u}} \delta u_{j,i}
\]

(40c)

\[
= \delta u_{i,j} \frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} + \delta u_{i,j} \frac{\partial^2 f_l}{\partial x_k \partial u_{j,i}} \delta x_k + \delta u_{i,j} \frac{\partial^2 f_l}{\partial x_k \partial u_{j,i}} \delta u_{j,i}
\]

(40d)

To get to each term in (40c), we used the fact that

\[
\frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} = 0, \text{ for } i \neq k \text{ or } j \neq k
\]

(41a)

\[
\delta x_k = \sum_{i=0}^{T} \frac{\partial x_k}{\partial u_{i,j}} \delta u_{i,j} = \sum_{i=0}^{T} \frac{\partial x_k}{\partial u_{i,j}} \delta u_{i,j} = \delta x_k
\]

(41b)

To get to (40d), we used the fact

\[
\frac{\partial^2 f_l}{\partial x_k \partial u_{j,i}} = A_k^l
\]

(42a)

\[
= \sum_{i=0}^{T} \sum_{j=0}^{T} \delta u_{i,j} \frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} \bigg|_{i,\bar{u}} \delta u_{j,i} = \Delta^l_k
\]

(42b)

\[
\delta u_{i,j} \frac{\partial^2 f_l}{\partial u_{i,j} \partial u_{j,i}} \bigg|_{i,\bar{u}} \delta u_{j,i} = \Delta^l_k
\]

(42c)

Both \(l\) and \(p\) are used to pick out the corresponding element for a vector or matrix. \(A_k^l\) means the \(l\)th row and \(p\)th column of matrix \(A_k\). Equation (40d) actually describes each element in (38b), so we’ve proven that both are true.

Next we prove (20a) is the quadratic approximation of \(J_n(u)\), i.e.

\[
\text{quad}(J_n(u))_{\bar{u}} = J_n(\bar{u}) + \frac{1}{2} \frac{\partial J_n}{\partial u} \bigg|_{\bar{u}} \delta u + \frac{1}{2} \frac{\partial^2 J_n}{\partial u^2} \bigg|_{\bar{u}} \delta u
\]

(43)

We’ll need the explicit expressions for the associated
We break down each term in (43). First the second order term.

\[
\frac{\partial^2 J_n(\bar{u})}{\partial u^2} \delta u = \left. \frac{\partial^2 J_n(u)}{\partial u_i \partial u_j} \right|_{\bar{u}} \delta u_{ij} = \sum_{i,j=0}^{T} \sum_{k=0}^{n} \frac{\partial^2 c_{n,k}(x_k,u_k)}{\partial u_i \partial x_k} \frac{\partial^2 c_{n,k}(x_k,u_k)}{\partial x_k \partial u_j} \delta u_{ij} + \sum_{i,j=0}^{T} \sum_{k=0}^{n} \frac{\partial^2 c_{n,k}(x_k,u_k)}{\partial x_k \partial u_i} \frac{\partial^2 c_{n,k}(x_k,u_k)}{\partial u_j \partial x_k} \delta u_{ij} + \sum_{i,j=0}^{T} \sum_{k=0}^{n} \frac{\partial^2 c_{n,k}(x_k,u_k)}{\partial u_i \partial x_k} \frac{\partial^2 c_{n,k}(x_k,u_k)}{\partial u_j \partial x_k} \delta u_{ij}
\]

(45b)

The first order term

\[
\frac{\partial J_n(\bar{u})}{\partial u} = \sum_{i=0}^{T} \frac{\partial J_n(u)}{\partial u_i} \delta u_i
\]

(46a)

\[
= \sum_{i=0}^{T} \left( \frac{\partial c_{n,k}(x_k,u_k)}{\partial u_i} \right) \frac{\partial c_{n,k}(x_k,u_k)}{\partial x_k} \delta u_i + \sum_{i=0}^{T} \left( \frac{\partial c_{n,k}(x_k,u_k)}{\partial x_k} \right) \frac{\partial c_{n,k}(x_k,u_k)}{\partial u_i} \delta u_i
\]

(46b)

\[
= \sum_{i=0}^{T} \left( \frac{\partial c_{n,k}(x_k,u_k)}{\partial u_i} \right) \delta u_i + \sum_{i=0}^{T} \left( \frac{\partial c_{n,k}(x_k,u_k)}{\partial x_k} \right) \frac{\partial c_{n,k}(x_k,u_k)}{\partial u_i} \delta u_i \sum_{i=0}^{T} \left( \frac{\partial c_{n,k}(x_k,u_k)}{\partial u_i} \right) \delta u_i
\]

(46c)

\[
= \sum_{i=0}^{T} \left( \frac{\partial c_{n,k}(x_k,u_k)}{\partial u_i} \right) \delta u_i + \sum_{i=0}^{T} \left( \frac{\partial c_{n,k}(x_k,u_k)}{\partial x_k} \right) \frac{\partial c_{n,k}(x_k,u_k)}{\partial u_i} \delta u_i
\]

(46d)

And constant term

\[
J_n(\bar{u}) = \sum_{k=0}^{n} c_{n,k}(\bar{x}_k,\bar{u}_k)
\]

(47)

From (45), (46), and (47) it follows that (43) is true.

\[\square\]

B. Proof of Lemma 6

A more complete version of Lemma 6 is

\[
D_{n,k} = D_{n,k} + O(\varepsilon)
\]

(48a)

\[
S_{n,k}^{1} = S_{n,k} + O(\varepsilon)
\]

(48b)

\[
\tilde{S}_{n,k}^{1} = \tilde{S}_{n,k}^{1} + O(\varepsilon)
\]

(48c)

\[
S_{n,k}^{1} = S_{n,k}^{1} + O(\varepsilon)
\]

(48d)

\[
\tilde{S}_{n,k} = \tilde{S}_{n,k} + O(\varepsilon)
\]

(48e)

\[
\Gamma_{n,k} = [\Gamma_{n,k}^{1} \Gamma_{n,k}^{2}] + O(\varepsilon^{2})
\]

(48f)

\[
\Gamma_{n,k} = \Gamma_{n,k} + O(\varepsilon)
\]

(48g)

\[
\Gamma_{n,k}^{1} = O(\varepsilon)
\]

(48h)

\[
\Gamma_{n,k}^{2} = O(\varepsilon)
\]

(48i)

\[
\tilde{F}_{k} = F_{k} + O(\varepsilon)
\]

(48j)

\[
\tilde{P}_{k} = P_{k} + O(\varepsilon)
\]

(48k)

\[
\tilde{H}_{k} = H_{k} + O(\varepsilon^{2})
\]

(48l)

\[
\tilde{H}_{k} = H_{k} + O(\varepsilon^{2})
\]

(48m)

\[
\tilde{H}_{k} = H_{k} + O(\varepsilon)
\]

(48n)

\[
\tilde{S}_{k} = \tilde{S}_{k} + O(\varepsilon^{2})
\]

(48o)

\[
\tilde{S}_{k} = S_{k} + O(\varepsilon)
\]

(48p)

\[
\tilde{S}_{k} = S_{k} + O(\varepsilon)
\]

(48q)

\[
\tilde{K}_{k} = K_{k} + O(\varepsilon)
\]

(48r)

of which we give a proof by induction for in this section.

For \(k = T\), because of the way these variables are constructed, they are identical, i.e.

\[
\Gamma_{n,T} = \tilde{\Gamma}_{n,T}
\]

(49a)

\[
F_{T} = \tilde{F}_{T}
\]

(49b)

\[
P_{T} = \tilde{P}_{T}
\]

(49c)

\[
H_{T} = \tilde{H}_{T}
\]

(49d)

\[
s_{T} = \tilde{s}_{T}
\]

(49e)

\[
K_{T} = \tilde{K}_{T}
\]

(49f)

\[
S_{n,T} = \tilde{S}_{n,T}
\]

(49g)
So we have (48d)(48e)(48g)(48k)(48l)(48o)(48r) hold for $k = T$.

We also know that

$$M^n_{u,T} = \frac{\partial c_{n,T}}{\partial u_T} = \frac{\partial J_p(u)}{\partial u_T} = O(\varepsilon) \quad (50)$$

where the first equality is by construction, the second is true because $u_T$ only appears in $J_p(u)$ in $c_{n,T}$. By construction, $\Gamma^{u}_{n,T} = \Gamma^{u}_{n,T} = M^{u}_{n,T} = O(\varepsilon)$, so (48h)(48i) are true for $k = T$. Similarly, $H_T$ and $H_T$ are constructed from $\Gamma^{u}_{n,T}$ and $\Gamma^{u}_{n,T}$, so (48m)(48n) are true for $k = T$.

Because $F^{-1}_k$ is bounded above, $s_T = -F^{-1}_T H_T = -F^{-1}_T O(\varepsilon) = O(\varepsilon)$. Similarly, $\tilde{s}_T = O(\varepsilon)$. Equations (48p)(48q) are true for $k = T$.

From (24b) and $\Gamma_{n,T} = M_{n,T}$, we can get

$$S^{1k}_{n,T} = M^{1k}_{n,T} + M^{1u}_{n,T} K_T + s^{T}_{n,T} (M^{1k}_{n,T} + M^{1u}_{n,T} K_T) = \Omega_{n,T} + O(\varepsilon) \quad (51a)$$

$$ = \Omega_{n,T} + O(\varepsilon) \quad (51b)$$

$$= \Omega_{n,T} + O(\varepsilon) \quad (51c)$$

because $M_{n,T}$ is bounded. Hence (48b) is true. Further, (48c) is also true.

The time indices for $D_{n,T-1}$ and $\tilde{D}_{n,T-1}$ go to a maximum of $T - 1$, so to prove things inductively, we need (48a) to hold for $k = T - 1$. The difference between constructions of $D_{n,T-1}$ and $\tilde{D}_{n,T-1}$ is that the former uses $\Omega_{n,T}$ and the later uses $\tilde{S}^{1k}_{n,T}$. But since we’ve proven $\Omega_{n,T} = \tilde{S}^{1k}_{n,T} + O(\varepsilon)$, and $G_{T-1}$ is bounded, we can also conclude $D_{n,T-1} = \tilde{D}_{n,T-1} + O(\varepsilon)$. Therefore (48a) is true for $k = T - 1$.

So far, we’ve proven that for the last step, either $k = T$ or $k = T - 1$, (48) is true. Assuming except for (48a), (48) is true for $k + 1$ and (48a) is true for $k$. If we can prove all equations hold one step back, our proof by induction would be done.

Assume (48a) holds for $k$ and other equations in (48) hold for $k + 1$. Readers be aware that we’ll use these assumptions implicitly in the derivations following.

From (24c) we can get

$$\Gamma^{1u}_{n,k} = M^{1u}_{n,k} + S^{1k}_{n,k + 1} B_k$$

$$= M^{1u}_{n,k} + \Omega_{n,k + 1} B_k + O(\varepsilon) B_k$$

$$= O(\varepsilon) \quad (52a)$$

Here we used (35). Similarly, we can prove $\tilde{\Gamma}^{1u}_{n,k} = O(\varepsilon)$. So (48h) and (48i) hold for $k$.

From (24c) and (15b) we can compute the difference between $\tilde{\Gamma}_{n,k}$ and $\Gamma_{n,k}$ as

$$\tilde{\Gamma}_{n,k} - \Gamma_{n,k} =$$

$$= \begin{bmatrix}
\tilde{S}^{11}_{n,k+1} - S^{11}_{n,k+1} & 0 & 0 \\
A^{\top}_k (\tilde{S}^{11}_{n,k+1} - S^{11}_{n,k+1}) A_k & 0 & 0 \\
B^{\top}_k (\tilde{S}^{11}_{n,k+1} - S^{11}_{n,k+1}) B_k & 0 & 0 \\
\end{bmatrix}$$

$$= \begin{bmatrix}
0 & 0 & 0 \\
A^{\top}_k (\tilde{S}^{11}_{n,k+1} - S^{11}_{n,k+1}) A_k + (\tilde{D}^{xx}_k - D^{xx}_k) & 0 & 0 \\
B^{\top}_k (\tilde{S}^{11}_{n,k+1} - S^{11}_{n,k+1}) B_k + (\tilde{D}^{uu}_k - D^{uu}_k) & 0 & 0 \\
\end{bmatrix}$$

$$= \begin{bmatrix}
0 & 0 & 0 \\
A^{\top}_k (\tilde{S}^{11}_{n,k+1} - S^{11}_{n,k+1}) B_k + (\tilde{D}^{uu}_k - D^{uu}_k) & 0 & 0 \\
B^{\top}_k (\tilde{S}^{11}_{n,k+1} - S^{11}_{n,k+1}) B_k + (\tilde{D}^{uu}_k - D^{uu}_k) & 0 & 0 \\
\end{bmatrix}$$

$$= \begin{bmatrix}
\tilde{O}(\varepsilon) & A^{\top}_k O(\varepsilon^2) & B^{\top}_k O(\varepsilon^2) \\
A^{\top}_k O(\varepsilon^2) & A^{\top}_k O(\varepsilon^2) A_k + O(\varepsilon) & A^{\top}_k O(\varepsilon^2) B_k + O(\varepsilon) \\
B^{\top}_k O(\varepsilon^2) & B^{\top}_k O(\varepsilon^2) A_k + O(\varepsilon) & B^{\top}_k O(\varepsilon^2) B_k + O(\varepsilon) \\
\end{bmatrix}$$

$$= \begin{bmatrix}
\tilde{O}(\varepsilon) & O(\varepsilon^2) & O(\varepsilon^2) \\
O(\varepsilon^2) & O(\varepsilon^2) & O(\varepsilon) \\
O(\varepsilon^2) & O(\varepsilon^2) & O(\varepsilon) \\
\end{bmatrix}$$

from which we can see that (48h) and (48g) are true. Once we proved the closeness between $\tilde{\Gamma}_{n,k}$ and $\Gamma_{n,k}$ and the specific terms are $O(\varepsilon)$, i.e. (48a) to (48b), because of the way they are constructed from $\Gamma_{n,k}$ and $\tilde{\Gamma}_{n,k}$, we are safe to say

$$\tilde{F}_k = F_k + O(\varepsilon) \quad (54a)$$

$$\tilde{H}_k = H_k + O(\varepsilon^2) \quad (54c)$$

$$H_k = O(\varepsilon) \quad (54d)$$

Therefore, (48), (48k), (48h), (48m) and (48n) are true for $k$.

Now that we have the results with $F_k$, $\tilde{F}_k$, $H_k$, $\tilde{H}_k$, $P_k$ and $\tilde{H}_k$, we can move to what are immediately following, i.e. $s_k$, $\tilde{s}_k$, $K_k$, and $\tilde{K}_k$.

$$s_k = -F^{-1}_k H_k = -F^{-1}_k O(\varepsilon) = O(\varepsilon) \quad (55)$$

which is true because $F^{-1}_k$ is bounded above. Similarly, we have $\tilde{s}_k = O(\varepsilon)$. Equations (48p) and (48q) are true.

$$\tilde{s}_k = -\tilde{F}^{-1}_k \tilde{H}_k = -(F_k + O(\varepsilon))^{-1} (H_k + O(\varepsilon^2))$$

$$= -(F^{-1}_k + O(\varepsilon^2)) (H_k + O(\varepsilon^2))$$

$$= -F^{-1}_k H_k + F^{-1}_k O(\varepsilon^2) + H_k O(\varepsilon) + O(\varepsilon^2)$$

$$= s_k + O(\varepsilon^2) \quad (56d)$$

$$\tilde{K}_k = -\tilde{F}^{-1}_k \tilde{P}_k = -(F_k + O(\varepsilon))^{-1} (P_k + O(\varepsilon))$$

$$= -(F^{-1}_k + O(\varepsilon^2)) (P_k + O(\varepsilon))$$

$$= -F^{-1}_k P_k + (F^{-1}_k + P_k) O(\varepsilon) + O(\varepsilon^2)$$

$$= K_k + O(\varepsilon) \quad (56h)$$
Now we are equipped to get closeness/small results for $S_{n,k}$ and $\tilde{S}_{n,k}$.

\[
\tilde{S}_{n,k} - S_{n,k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & I & \tilde{K}_k \end{bmatrix} \Gamma_{n,k} \begin{bmatrix} 1 & 0 \\ 0 & I \\ 0 & \tilde{K}_k \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & I \\ 0 & K_k \end{bmatrix} \Gamma_{n,k} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\]

\[
\begin{aligned}
\tilde{S}_{n,k} & \left( \Gamma_{n,k} + \Gamma_{n,k} \tilde{K}_k \right) s_k - \left( \Gamma_{n,k} + \Gamma_{n,k} K_k \right) s_k \\
\tilde{S}_{n,k} & \left( \Gamma_{n,k} + \Gamma_{n,k} K_k \right) s_k - \left( \Gamma_{n,k} + \Gamma_{n,k} K_k \right) s_k
\end{aligned}
\]

So that (48d) and (48e) are true for $S_{n,k}$ and $\tilde{S}_{n,k}$.

\[
\begin{aligned}
\mathcal{L}_{n,k} & = M_{n,k}^{1x} + \sum_{i=1}^{n_c} \mathcal{L}_{n,k} G_{k}^{i} \\
\mathcal{L}_{n,k} & = \mathcal{L}_{n,k}^{1x} + \mathcal{L}_{n,k}^{1x} A_k + \Gamma_{n,k} \mathcal{L}_{n,k} K_k + s_k \left( \Gamma_{n,k} + \Gamma_{n,k} K_k \right) O(\varepsilon)
\end{aligned}
\]

Therefore, (48b) holds and then naturally (48c) holds.

C. Proof of Lemma 7

A more complete version of Lemma 7 is

\[
\delta x_{n,k}^{x} = A_k \delta x_{k}^{x} + B_k \delta u_{k}^{x} + O(\varepsilon^2)
\]

\[
\delta x_{n,k}^{N} = A_k \delta x_{k}^{N} + B_k \delta u_{k}^{N} + O(\varepsilon^2)
\]

Equation (60a) comes directly from the Taylor series expansion of (2) and (60b) from (60d).

We prove (60c) to (60h) by induction. For $k=0$, $\delta x_{n,k}^{N} = \delta x_{0}^{N} = 0$ and $\delta u_{n,k}^{N} = 0$. We know from the proof of Lemma 6 that $s_n = s_0 + O(\varepsilon^2)$, $s_0 = O(\varepsilon)$ and $s_1 = O(\varepsilon)$, so (60c) to (60h) hold for $k=0$. Assume (60c) to (60h) hold for $k$, then

\[
\begin{aligned}
\delta x_{n,k+1}^{x} & = K_{k+1} \delta x_{k}^{x} + s_{k+1}^{x} = O(\varepsilon) \\
\delta x_{n,k+1}^{N} & = K_{k+1} \delta x_{k}^{N} + s_{k+1}^{N} = O(\varepsilon) \\
\delta u_{n,k+1}^{x} & = A_k \delta x_{k}^{x} + B_k \delta u_{k}^{x} + O(\varepsilon^2) = O(\varepsilon) \\
\delta u_{n,k+1}^{N} & = A_k \delta x_{k}^{N} + B_k \delta u_{k}^{N} + O(\varepsilon^2) = O(\varepsilon)
\end{aligned}
\]

So (60c) to (60h) hold for $k+1$ and the proof by induction is done. Equation (60a) comes directly as a result. Equation (60b) is classic convergence analysis for Newton’s method [24]. Equation (60K) follows directly from (60b) and (60h).

D. Proof of Lemma 7

As discussed in the proof of Lemma 2, a necessary condition for the solution of (10C) is given by (16). Thus, a sufficient condition for a unique solution is that $F_k$ be invertible. At the beginning of the appendix, we showed that $F_k$ exists near $u^*$ and that its spectral radius is bounded.

Now we show that $\tilde{F}_k$ exists and $\rho(\tilde{F}_k^{-1})$ is bounded. Lemma 6 implies that $F_k = F_k + O(\varepsilon)$. It follows that

\[
F_k^{-1} = (F_k + O(\varepsilon))^{-1} = F_k^{-1} - F_k^{-1} O(\varepsilon) F_k^{-1} = F_k^{-1} + O(\varepsilon)
\]

It follows that $\tilde{F}_k$ exists and is bounded in a neighborhood of $u^*$.

So (48a) is true.