Adaptive Dynamic Programming

J. Murray
Department of Electrical Engineering
State University of New York at Stony Brook
Stony Brook, NY 11790

and

C. Cox, G. Lendaris\(^1\), J. Neidhoefer, and R. Saeks
Accurate Automation Corporation\(^2\)
7001 Shallowford Rd.
Chattanooga, TN 37421

February 1, 2001

Abstract

An Adaptive Dynamic Programming algorithm for nonlinear systems with unknown dynamics is developed. The algorithm is initialized with a positive definite cost functional / stabilizing control law pair \((V_0, k_0)\) (coupled via the Hamilton Jacobi Bellman Equation). Given \((V_i, k_i)\), one runs the system using control law \(k_i\) recording the state and control trajectories, with these trajectories used to define \(V_{i+1}\) as the cost to take the initial state \(x_0\) to the final state using control law, \(k_i\), while \(k_{i+1}\) is taken to be the control law derived from \(V_{i+1}\) via Hamilton Jacobi Bellman Equation.

In this paper we show that this process is globally convergent with step-wise stability to the optimal cost functional / control law pair, \((V^*, k^*)\), for an (unknown) input affine system with an input quadratic performance measure (modulo the appropriate technical conditions). Furthermore, three specific implementations of the Adaptive Dynamic Programming algorithm are developed; for i) the linear case, ii) for the nonlinear case using a locally quadratic approximation to the cost functional, and iii) the nonlinear case using a (potentially global) radial basis function approximation of the cost functional; illustrated by applications to flight control.

1. INTRODUCTION

Unlike the many soft computing applications where it suffices to achieve a “good approximation most of the time”, a control system must be stable all of the time. As such, if one desires to learn a

---

1. On Sabbatical Leave from Portland State University.
2. This research performed in part on NSF SBIR contracts DMI-9660604, DMI-9860370, and DMI-9983287, NASA Ames SBIR Contracts NAS2-98016 and NAS2-20008, and NSF Grant ECS-9904378.
control law in real-time, a fusion of soft computing techniques to learn the appropriate control law with hard computing techniques to maintain the stability constraint and guarantee convergence is required. The objective of the present paper is to describe an Adaptive Dynamic Programming Algorithm which uses soft computing techniques to learn the optimal cost (or return) functional for a stabilizable nonlinear system with unknown dynamics and hard computing techniques to verify the stability and convergence of the algorithm.


The centerpiece of Dynamic Programming is the Hamilton Jacobi Bellman (HJB) Equation [Bellman, R.E., (1957)][Bertsekas, D.P ., (1987)][Luenberger, D.G ., (1979)] which one solves for the optimal cost functional, $V^*(x_0, t_0)$. This equation characterizes the cost to drive the initial state, $x_0$, at time $t_0$ to a prescribed final state using the optimal control. Given the optimal cost functional, one may then solve a second partial differential equation (derived from the HJB Equation) for the corresponding optimal control law, $k^*(x, t_0)$, yielding an optimal cost functional / optimal control law pair, ($V^*$, $k^*$).

Although direct solution of the Hamilton Jacobi Bellman Equation is computationally untenable (the so-called “curse of dimensionality”), the HJB Equation and the relationship between $V^*$ and the corresponding control law, $k^*$, derived therefrom, serves as the basis of the Adaptive Dynamic Programming Algorithm developed in the present paper. In this algorithm we start with an initial cost functional / control law pair, ($V_0$, $k_0$), where $k_0$ is a stabilizing control law for the plant, and construct a sequence of cost functional / control law pairs, ($V_i$, $k_i$), in real-time, which converge to the optimal cost functional / control law pair, ($V^*$, $k^*$) as follows.

- Given $(V_i, k_i)$; $i = 0, 1, 2, \ldots$; we run the system using control law $k_i$ from an array of initial conditions, $x_0$, covering the entire state space (or that portion of the state space where one expects to operate the system);
- recording the state, $x_i(x_0, \cdot)$, and control trajectories, $u_i(x_0, \cdot)$, for each initial condition.
Given this data, we define $V_{i+1}$ to be the cost to take the initial state $x_0$ at time $t_0$, to the final state, using control law, $k_i$, and
take $k_{i+1}$ to be the corresponding control law derived from $V_{i+1}$ via HJB Equation;
iterating the process until it converges.

Indeed, in the following Section and in Appendix B it is shown that (with the appropriate technical assumptions) this process is:

- globally convergent to
- the optimal cost functional, $V^*$, and the optimal control law, $k^*$, and is
- stepwise stable; i.e., $k_i$ is a stabilizing controller at every iteration with Lyapunov function, $V_i$.

Since stability is an asymptotic property, technically it is sufficient that $k_i$ be stabilizing in the limit. In practice, however, if one is going to run the system for any length of time with control law, $k_i$, it is necessary that $k_i$ be a stabilizing controller at each step of the iterative process. As such, for this class of adaptive control problems we “raise the bar”, requiring stepwise stability; i.e., stability at each iteration of the adaptive process, rather than simply requiring stability in the limit. Moreover, a-priori knowledge of the state dynamics matrix is not required to implement the algorithm, while the requirement that the input matrix be known (to compute $k_{i+1}$ from $V_{i+1}$), can be circumvented by the pre-compensator technique described in Appendix A. As such the above described Adaptive Dynamic Programming Algorithm can be applied to plants with completely unknown dynamics.

While one must eventually explore the entire state space (probably repeatedly) in any (truly) nonlinear control problem with unknown dynamics, in the above described Adaptive Dynamic Programming Algorithm one must explore the entire state space at each iteration of the algorithm (by running the system from an array of initial states which cover the entire state space). Unfortunately, this is not feasible and is tantamount to fully identifying the plant dynamics at each iteration of the algorithm. As such, Sections 3. - 5. of the present paper are devoted to the development of three approximate implementations of the Adaptive Dynamic Programming Algorithm which do not require global exploration of the state space at each iteration. These include:

- the linear case, where one can evaluate $k_{i+1}$ and $V_{i+1}$ from $n$ local observations of the system state at each iteration;
- an approximation of the nonlinear control law at each point of the state space, derived using a quadratic approximation of the cost functional at that point, requiring $n(n+1)/2$ local observations of the system state at each iteration; and
A nonlinear control law, derived at each iteration of the algorithm from a *radial basis function approximation* of the cost functional, which is updated locally at each iteration using data obtained along a single state trajectory.

2. ADAPTIVE DYNAMIC PROGRAMMING ALGORITHM

In the formulation of the Adaptive Dynamic Programming Algorithm and Theorem, we use the following notation for the state and state trajectories associated with the plant. The variable “\(x\)” denotes a generic state while “\(x_0\)” denotes an initial state, “\(t\)” denotes a generic time and “\(t_0\)” denotes an initial time. We use the notation, \(x(x_0, \cdot)\), for the state trajectory produced by the plant (with an appropriate control) starting at initial state \(x_0\) (at some implied initial time), and the notation, \(u(x_0, \cdot)\), for the corresponding control. Finally, the state reached by a state trajectory at time “\(t\)” is denoted by \(x = x(x_0, t)\), while the value of the corresponding control at time “\(t\)” is denoted by \(u = u(x_0, t)\).

For the purposes of the present paper, we consider a stabilizable time-invariant input affine plant of the form

\[
\dot{x} = f(x, u) \equiv a(x) + b(x)u; \quad x(t_0) = x_0
\]

with input quadratic performance measure

\[
J = \int_{t_0}^{\infty} l(x(x_0, \lambda), u(x_0, \lambda)) d\lambda
\]

\[
\equiv \int_{t_0}^{\infty} [q(x(x_0, \lambda)) + u^T(x_0, \lambda) r(x(x_0, \lambda)) u(x_0, \lambda)] d\lambda
\]

Here \(a(x), b(x), q(x), \) and \(r(x)\) are \(C^\infty\) matrix valued functions of the state which satisfy:

1. \(a(0) = 0\), producing a singularity at \((x,u) = (0,0)\);
2. the eigenvalues of \(\frac{da(0)}{dx}\) have negative real parts, i.e., the linearization of the uncontrolled plant at zero is exponentially stable;
3. \(q(x) > 0, x \neq 0; \quad q(0) = 0\);
4. \(q(x)\) has a positive definite Hessian at \(x = 0, \frac{d^2 q(0)}{dx^2} > 0\), i.e., any non-zero state is penalized independently of the direction from which it approaches 0; and
5. \(r(x) > 0\) for all \(x\).
The goal of the Adaptive Dynamic Programming Algorithm is to adaptively construct an optimal control, \( u^o(x_0, \cdot) \), which takes an arbitrary initial state, \( x_0 \), at \( t_0 \) to the singularity at \((0,0)\), while minimizing the performance measure, \( J \).

Since the plant and performance measure are time invariant, the optimal cost functional and optimal control law are independent of the initial time, \( t_0 \) which we may, without loss of generality, take to be \( 0 \); i.e. \( V^o(x_0, t_0) = V^o(x_0) \) and \( k^o(x, t_0) \equiv k^o(x) \). Even though the optimal cost functional is defined in terms of the initial state, it is a generic function of the state, \( V^o(x) \), and is used in this form in the Hamilton Jacobi Bellman Equation and throughout the paper. Finally, we adopt the notation \( F^o(x) \equiv a(x) + b(x)k^o(x) \), for the optimal closed loop feedback system. Using this notation, the Hamilton Jacobi Bellman Equation then takes the form

\[
\frac{dV^o(x)}{dx} F^o(x) = -l(x, k^o(x)) = -q(x) - k^oT(x)r(x)k^o(x)
\]

in the time-invariant case [Luenberger, D.G., (1979)].

Differentiating the HBJ Equation (3) with respect to \( u^o = k^o(x) \) now yields

\[
\frac{dV^o(x)}{dx} b(x) = -2k^oT(x)r(x)
\]

or equivalently

\[
u = k^o(x) = -\frac{1}{2} r^{-1}(x)b^T(x) \left[ \frac{dV^o(x)}{dx} \right]^T
\]

which is the desired relationship between the optimal control law and the optimal cost functional. Note, that an input quadratic performance measure is required to obtain the explicit form for \( k^o \) in terms of \( V^o \) of Equation 5, though a similar implicit relationship can be derived in the general case. (See [Saeks, R., and C. Cox, (1998)] for a derivation of this result).

Given the above preparation, we may now formulate the desired Adaptive Dynamic Programming Algorithm as follows.

**Adaptive Dynamic Programming Algorithm:**

1. Initialize the algorithm with a stabilizing cost functional / control law pair \((V_0, k_0)\), where \( V_0(x) \) is a \( C^\infty \) function, \( V_0(x) > 0, x \neq 0 \); \( V_0(0) = 0 \), with a positive definite Hessian at \( x = 0 \), \( \frac{d^2V_0(0)}{dx^2} > 0 \); and \( k_0(x) \), is the \( C^\infty \) control law, \( u = k_0(x) = -\frac{1}{2} r^{-1}(x)b^T(x) \left[ \frac{dV_0(x)}{dx} \right]^T \).
Adaptive Dynamic Programming

2 For \( i = 0, 1, 2, \ldots \) run the system with control law, \( k_i \), from an array of initial conditions, \( x_0 \) at \( t_0 = 0 \), recording the resultant state trajectories, \( x_i(x_0, \cdot) \), and control inputs \( u_i(x_0, \cdot) = k_i(x_i(x_0, \cdot)) \).

3 For \( i = 0, 1, 2, \ldots \) let

\[
V_{i+1}(x_0) = \int_{0}^{\infty} l(x_i(x_0, \lambda), u_i(x_0, \lambda))d\lambda
\]

and

\[
u = k_{i+1}(x) = -\frac{1}{2} r^{-1}(x)b^T(x)\left[\frac{dV_{i+1}(x)}{dx}\right]^T
\]

where, as above, we have defined \( V_{i+1} \) in terms of initial states but use it generically.

4 Go to 2.

Since the state dynamics matrix, \( a(x) \), does not appear in the above algorithm one can implement the algorithm for a system with unknown \( a(x) \). Moreover, one can circumvent the requirement that \( b(x) \) be known in Step 3, by augmenting the plant with a known pre-compensator at the cost of increasing its dimensionality, as shown in Appendix A. As such, the Adaptive Dynamic Programming Algorithm can be applied to plants with completely unknown dynamics. As indicated in the introduction, however, the requirement that one fully explore the state space at each iteration of the algorithm is tantamount to identifying the plant dynamics. As such, the applicability of the Adaptive Dynamic Programming Algorithm to plants with unknown dynamics is only meaningful in the context of the approximate implementations of Sections 3 - 5., where only a local exploration of the state space is required.

In the following we adopt the notation \( F_i \) for the closed loop system defined by the plant and control law, \( k_i \),

\[
\dot{x} = F_i(x) = a(x) + b(x)k_i(x) = a(x) - \frac{1}{2} b(x)r^{-1}(x)b^T(x)\left[\frac{dV_i(x)}{dx}\right]^T
\]

To initialize the Adaptive Dynamic Programming Algorithm for a stable plant, one may take \( V_0(x) = \varepsilon x^T x \) and \( k_0(x) = -\varepsilon r^{-1}(x)b^T(x)x \) which will stabilize the plant for sufficiently small \( \varepsilon \) (though in practice we often take \( k_0(x) = 0 \)). Similarly, for a stabilizable plant, one can “pre-stabilize” the plant with any desired stabilizing control law such that \( \frac{d^2 V_0(0)}{dx^2} > 0 \) and the eigenvalues of \( \frac{dF_0(0)}{dx} \) have negative real parts; and then initialize the Adaptive Dynamic Programming Algorithm.
Programming Algorithm with the above cost functional / control law pair. Moreover, since the
state trajectory going through any point in state space is unique, and the plant and controller are
time-invariant, one can treat every point on a given state trajectory as a new initial state when
evaluating $V_{i+1}(x_0)$, by shifting the time scale analytically without rerunning the system.

The Adaptive Dynamic Programming Algorithm is characterized by the following Theorem.

**Adaptive Dynamic Programming Theorem**: Let the sequence of cost functional / control law pairs $(V_i, k_i)$; $i = 0, 1, 2, ...$; be defined by, and satisfy the conditions of the Adaptive Dynamic Programming Algorithm. Then,

i) $V_{i+1}(x)$ and $k_{i+1}(x)$ exist, where $V_{i+1}(x)$ and $k_{i+1}(x)$ are $C^\infty$ functions with

$$V_{i+1}(x) > 0, x \neq 0; \quad V_{i+1}(0) = 0; \quad \frac{d^2 V_{i+1}(0)}{dx^2} > 0; \quad i = 0, 1, 2, ... .$$

ii) The control law, $k_{i+1}$, stabilizes the plant (with Lyapunov function $V_{i+1}(x)$) for

all $i = 0, 1, 2, ...$, and that the eigenvalues of $\frac{dF_{i+1}(0)}{dx}$ have negative real parts.

iii) The sequence of cost functional / control law pairs, $(V_{i+1}, k_{i+1})$, converge to the

optimal cost functional / control law pair, $(V^*, k^*)$.

Note that in ii), the existence of the Lyapunov function $V_{i+1}(x)$ together with the eigenvalue
condition on $\frac{dF_{i+1}(0)}{dx}$ implies that the closed loop system, $F_{i+1}(x)$, is exponentially stable
[Halanay, A., and Rasvan, V., (1993)], rather than asymptotically stable, as implied by the
existence of the Lyapunov function alone.

In the following, we sketch the proof of the Adaptive Dynamic Programming Theorem, while the
details of the proof appear in Appendix B. The proof includes 4 steps as follows.

1 **Show that** $V_{i+1}(x)$ and $k_{i+1}(x)$ **exist and are** $C^\infty$ **functions, with**

$V_{i+1}(x) > 0, x \neq 0; \quad V_{i+1}(0) = 0; \quad i = 0, 1, 2, ... .$

The first step required to prove that $V_{i+1}(x)$ and $k_{i+1}(x)$ exist and are $C^\infty$ functions, is to
show that the state trajectories defined by the control law $k_i$ and their derivatives with respect
to the initial condition are integrable. Since $k_i$ is a stabilizing control law the state trajectories
$x_i(x_0, \cdot)$ are asymptotic to zero. Although this implies that they are bounded, it is not sufficient
for integrability. In combination with the condition that the eigenvalues of $\frac{dF_i(0)}{dx}$ have nega-
Adaptive Dynamic Programming

tive real values, however, asymptotic stability implies exponential stability [Halanay, A., and Rasvan, V., (1993)], which is sufficient to guarantee integrability. Intuitively, asymptotic stability guarantees that the state trajectories will eventually converge to a neighborhood of zero where the closed loop system defined by \( F_i \) may be approximated by the linear system defined by \( \frac{dF_i(0)}{dx} \), which is exponentially stable since the eigenvalues of \( \frac{dF_i(0)}{dx} \) have negative real values. See [Halanay, A., and Rasvan, V., (1993)] for the details of this theorem.

Similarly, one can show that the derivatives of the state trajectories with respect to the initial condition, \( \frac{d^n x_i(x_0, \cdot)}{dx^n} \), are exponentially stable by showing that they also satisfy a differential equation which may be approximated in the limit by the linear system defined by \( \frac{dF_i(0)}{dx} \).

Moreover, since the state trajectories and their derivatives with respect to the initial condition are exponentially stable, it follows from the defining properties for the plant and performance measure, that \( l(x_i(x_0, \cdot), u_i(x_0, \cdot)) \) and its derivatives with respect to the initial condition are also exponentially convergent to zero.

As such, \( l(x_i(x_0, \cdot), u_i(x_0, \cdot)) \) and its derivatives with respect to the initial condition are integrable, while they are \( C^\infty \) functions, since \( F_i \) is a \( C^\infty \) function [Dieudonne, J., (1960)]. As such,

\[
V_{i+1}(x_0) \equiv \int_{0}^{\infty} l(x_i(x_0, \lambda), u_i(x_0, \lambda)))d\lambda \tag{7}
\]

and

\[
k_{i+1}(x) = \frac{1}{2} r^{-1}(x)b^T(x) \left[ \frac{dV_{i+1}(x)}{dx} \right]^T \tag{8}
\]

exist and are \( C^\infty \) functions.

2. Show that the Iterative Hamilton Jacobi Bellman Equation

\[
\frac{dV_{i+1}(x)}{dx} F_i(x) = -l(x, k_i(x))
\]

is satisfied, and that \( \frac{d^2 V_{i+1}(0)}{dx^2} > 0; i = 0, 1, 2, \ldots \).
The iterative HJB Equation, which may be used as an alternative to Equation 7 for implementing the Adaptive Dynamic Programming algorithm, is derived by computing $\frac{dV_{i+1}(x_i(x_0, t))}{dt}$ via the chain rule to obtain the left side of the Iterative HJB Equation, and by directly differentiating Equation 7 to obtain the right side of the equation. Then if one takes the second derivative of both sides of the resultant equation, evaluates it at $x = 0$, and drops those terms which contain $\frac{dV_{i+1}(0)}{dx}$ or $F_i(0)$, both of which are zero; one obtains the Linear Lyapunov Equation

$$
\begin{align*}
\left[ \frac{dF_i(0)}{dx} \right]^T \frac{d^2V_{i+1}(0)}{dx^2} + \frac{d^2V_{i+1}(0)}{dx^2} \left[ \frac{dF_i(0)}{dx} \right] \\
= - \left[ \frac{d^2q(0)}{dx^2} + \frac{1}{2} \left( \frac{d^2V_i(0)}{dx^2} \right) (b(x)r^{-1}(x)b^T(x)) \left( \frac{d^2V_i(0)}{dx^2} \right)^T \right]
\end{align*}
$$

(9)

Now, since the eigenvalues of $\frac{dF_i(0)}{dx}$ have negative real parts, while the right side of Equation 9 is a negative definite symmetric matrix, the unique symmetric solution of the Linear Lyapunov Equation 9 is positive definite [Barnett, S., (1971)] and, as such, $\frac{d^2V_{i+1}(0)}{dx^2} > 0$, as required.

3 Show that $V_{i+1}(x)$ is a Lyapunov Function for the Closed Loop System, $F_{i+1}$, and that the eigenvalues of $\frac{dF_{i+1}(0)}{dx}$ have negative real parts; $i = 0, 1, 2, ...$.

This is achieved by directly computing $\frac{dV_{i+1}(x_i(x_0, t))}{dt}$, i.e., the derivative of $V_{i+1}(x)$ along the trajectories of the closed loop system, $F_{i+1}$, with the aid of the chain rule and the Iterative HJB equation, implying that $k_{i+1}$ is a stabilizing controller for the plant for all $i = 0, 1, 2, ...$.

To show that the eigenvalues of $\frac{dF_{i+1}(0)}{dx}$ have negative real parts, we use an argument similar to that used in 2, taking the second derivative of the expression derived for $\frac{dV_{i+1}(x)}{dx} F_{i+1}(x) = \frac{dV_{i+1}(x_i(x_0, t))}{dt}$ derived above.
4 Show that the sequence of cost functional / control law pairs, \((V_{i+1}, k_{i+1})\) is convergent.

This is achieved by showing that the derivative of \(V_{i+1}(x) - V_i(x)\) is positive along the trajectories of \(F_i\),
\[
\frac{d}{dt}[V_{i+1}(x_i(x_0, t)) - V_i(x_i(x_0, t))]
\]

is positive for \(i = 1, 2, 3, ...\). Moreover, since \(F_i\) is asymptotically stable, its state trajectories, \(x_i(x_0, t)\), converge to zero, and hence so does \(V_{i+1}(x_i(x_0, t)) - V_i(x_i(x_0, t))\). Since along these trajectories, however, this implies that \(V_{i+1}(x_i(x_0, t)) - V_i(x_i(x_0, t)) < 0\) on the trajectories of \(F_i\); \(i = 1, 2, 3, ...\). Since every point in the state space lies along some trajectory of \(F_i\) this implies that \(V_{i+1}(x) - V_i(x) < 0\), or equivalently, \(V_{i+1}(x) < V_i(x)\) for all \(x; i = 1, 2, 3, ...\). As such, \(V_{i+1}\) is a decreasing sequence of positive functions; \(i = 1, 2, 3, ...\); and is therefore convergent (as is the sequence \(V_{i+1}; i = 0, 1, 2, ...\); since the behavior of the first entry of a sequence does not affect its convergence).

Note, the requirement that \(i \geq 1\) in this step of the proof is a “physical fact” and not just a “mathematical anomaly”, as indicated by the examples of Sections 3.-5., where the “cost-to-go” from a given state typically jumps from its initial value for \(i = 0\) to a large value, and then monotonically decreases to the optimal cost as one runs the algorithm for \(i = 1, 2, 3, ...\).

3. THE LINEAR CASE

The purpose of this section is to develop an implementation of the Adaptive Dynamic Programming algorithm for the linear case, where local exploration of the state space at each iteration of the algorithm is sufficient, yielding a computationally tractable algorithm. For this purpose we consider a linear time-invariant plant
\[
\dot{x} = Ax + Bu; \quad x(t_0) = x_0
\]

with the quadratic performance measure
\[
J = \int_{t_0}^{\infty} [x^T(x_0, \lambda)Qx(x_0, \lambda) + u^T(x_0, \lambda)Ru(x_0, \lambda)] d\lambda
\]

Here \(Q\) is a positive matrix, while \(R\) is positive definite. For this case \(V^\rho(x) = x^T P^\rho x\) is a quadratic form, where \(P^\rho\) is a positive definite matrix. As such,
\[
\frac{dV^\rho}{dx}(x) = 2x^T P^\rho
\]
and
\[
u = K^\rho x = -R^{-1}B^T P^\rho x.
\]

To implement the Adaptive Dynamic Programming Algorithm in the linear case, we initialize the algorithm with a quadratic cost functional, \(V_0(x) = x^T P_0 x\) and \(K_0 = -R^{-1}B^T P_0\). Now, assuming
that $V_i(x) = x^T P_i x$ is quadratic and $K_i = -R^{-1} B^T P_i$, then

$$F_i(x) = [A - BK_i] x = [A - BR^{-1} B^T P_i] x \equiv F_i x;$$

where by abuse of notation we have used the symbol, $F_i$, for both the closed loop system and the matrix which represents it. As such, the state trajectories for the plant with control law $K_i$ can be expressed in the exponential form $x_i(x_0, t) = e^{F_i x_0}$, while the corresponding control is $u_i(x_0, t) = K_i e^{F_i x_0}$. As such,

$$V_{i+1}(x_0) = \int_{0}^{\infty} [x_i^T(x_0, \lambda) Q x_i(x_0, \lambda) + u_i^T(x_0, \lambda) R u_i(x_0, \lambda)] d\lambda$$

$$= \int_{0}^{\infty} \left[ x_0^T e^{F_i \lambda} Q e^{F_i \lambda} x_0 + x_0^T e^{F_i \lambda} K_i^T Q K_i e^{F_i \lambda} x_0 \right] d\lambda = x_0^T \left[ \int_{0}^{\infty} e^{F_i \lambda} \left[ Q + K_i^T R K_i \right] e^{F_i \lambda} d\lambda \right] x_0$$

(12)

Now, since $F_i$ is asymptotically stable the integral of Equation 12 exists, confirming that $V_{i+1}(x) = x^T P_{i+1} x$ is also quadratic. Moreover, the integral defining $P_{i+1}$ is the “well known” integral form of the solution of the Linear Lyapunov Equation [Barnett, S., (1971)]

$$P_{i+1} F_i + F_i^T P_{i+1} = -[Q + P_i B R^{-1} B^T P_i]$$

(13)

As such, in the linear case, rather directly evaluating the integral of Equation 12 one can iteratively solve for $P_{i+1}$ in terms of $P_i$ by solving the Linear Lyapunov Equation (13). Note, that as an alternative to the above derivation one can obtain Equation 13 by expressing $dV_{i+1}(x) / dx$ and $K_{i+1}$ in the form $dV_{i+1}(x) = 2x^T P_{i+1}$ and $K_{i+1} = -R^{-1} B^T P_{i+1}$, and substituting these expressions into the Iterative HJB Equation.

Although the $A$ matrix for the plant is implicit in $F_i (= [A - BR^{-1} B^T P_i])$, one can estimate $F_i$ directly from measured data without a priori knowledge of $A$. To this end, one runs the system using control law $K_i$ over some desired time interval, and observes the state at $n$ (the dimension of the state space) or more points, $x_i$: $j = 1, 2, \ldots, n$; while (numerically) estimating the time derivative of the state at the same set of points; $\dot{x}_j$: $j = 1, 2, \ldots, n$. Now, since $F_i$ is the closed loop system matrix for the plant with control law $K_i$, $\dot{x}_j = F_i x_j$: $j = 1, 2, \ldots$; or equivalently $\dot{X}_i = F_i X_i$ where $X_i = [x_1 \ x_2 \ \ldots \ x_n]$. Assuming that the points where one observes the state are linearly independent, one can then solve for $F_i$ from the observations via the equality
Adaptive Dynamic Programming

\[ F_i = \dot{X}_i X_i^{-1}, \] yielding the alternative representation of the Linear Lyapunov equation

\[
P_{i+1} [\dot{X}_i X_i^{-1}] + [\dot{X}_i X_i^{-1}]^T P_{i+1} = -[Q + P_i B R^{-1} B^T P_i] \]  

(14)

which can be solved for \( P_{i+1} \) in terms of \( P_i \) without a-priori knowledge of \( A \). Moreover, one can circumvent the requirement that \( B \) be known via the pre-compensation technique of Appendix A.

As such, Equation 14 can be used to implement the Adaptive Dynamic Programming Algorithm without a-priori knowledge of the plant. Moreover, since \( F_i = \dot{X}_i X_i^{-1} \) is asymptotically stable, Equation 14 always admits a well defined positive definite solution, \( P_{i+1} \), while there are numerous numerical solution techniques for solving this class of Linear Lyapunov Equations [Barnett, S., (1971)]. Moreover, unlike the full nonlinear algorithm, this implementation of the Adaptive Dynamic Programming Algorithm requires only local information at each iteration. Finally, if one implements the above algorithm off-line to construct the optimal controller for a system with known dynamics, using \( F_i \) at each iteration in lieu of \( \dot{X}_i X_i^{-1} \), then the algorithm reduces to the Newton-Raphson iteration for solving the matrix Riccati Equation [Holley, W.E., and S.Y. Wei, (1979)] [Kwakernaak, H., and R. Sivan, (1972)].

As an alternative to the above Linear Lyapunov Equation implementation, one can formulae an alternative implementation of linear Adaptive Dynamic Programming Algorithm using local information along a single state trajectory, \( x_i(x_0, \cdot) \), and the corresponding control, \( u_i(x_0, \cdot) = K_i x_i(x_0, \cdot) \), starting at initial state \( x_0 \) and converging to the singularity at \((0,0)\). Indeed, for this trajectory one may evaluate \( V_{i+1}(x_0) \) via

\[
V_{i+1}(x_0) = \int_0^\infty [x_i^T(x_0, \lambda) Q x_i(x_0, \lambda) + u_i^T(x_0, \lambda) R u_i(x_0, \lambda)] d\lambda = x_0^T P_{i+1} x_0 \]  

(15)

since the plant and control law are time-invariant. More generally, for any initial state, \( x_j = x_j(x_{0j}, t_j) \), along this trajectory

\[
V_{i+1}(x_j) = \int_{t_j}^\infty [x_i^T(x_0, \lambda) Q x_i(x_0, \lambda) + u_i^T(x_0, \lambda) R u_i(x_0, \lambda)] d\lambda = x_0^T P_{i+1} x_0 \]  

(16)

Now, since the positive definite matrix \( P_{i+1} \) has only \( q = n(n+1)/2 \) independent parameters, one can select \( q \) (or more) initial states along this trajectory; \( x_j; j = 1, 2, ..., q \); and solve the set of simultaneous equations

\[
x_j^T P_{i+1} x_j = V_{i+1}(x_j); \quad j = 1, 2, ..., q \]  

(17)
for $P_{i+1}$. Equivalently, applying the matrix Kronecker product formula, \[ \text{vec}(ABC) = [C^T \otimes A]\text{vec}(B) \] where the “vec” operator maps a matrix into a vector by stacking its columns on top of one another, one may transform Equation 17 into a $q \times n^2$ matrix equation

\[
\begin{bmatrix}
    x_1^T \otimes x_1^T \\
    x_2^T \otimes x_2^T \\
    \vdots \\
    x_q^T \otimes x_q^T 
\end{bmatrix} \text{vec}(P_{i+1}) = 
\begin{bmatrix}
    V_{i+1}(x_1) \\
    V_{i+1}(x_2) \\
    \vdots \\
    V_{i+1}(x_q) 
\end{bmatrix}
\tag{18}
\]

Now, let “vec$^+$” be the operator that maps an $n \times n$ matrix, $B$, to a $q = n(n+1)/2$ vector, vec$^+$(B), by stacking the upper triangular part of its columns, $b_{ij}$, $i \leq j$, on top of one another. Now, if $B$ is symmetric, vec$^+(B)$ fully characterizes $B$ and, as such, one may define an $n^2 \times q$ matrix, $S$, which maps vec$^+(B)$ to vec$(B)$ for any symmetric matrix, $B$. As such, one may express Equation 18 in the form of a $q \times q$ matrix equation in the unknown vec$^+(P_{i+1})$,

\[
\begin{bmatrix}
    x_1^T \\
    x_2^T \\
    \vdots \\
    x_q^T 
\end{bmatrix} S \text{vec}^+(P_{i+1}) = 
\begin{bmatrix}
    V_{i+1}(x_1) \\
    V_{i+1}(x_2) \\
    \vdots \\
    V_{i+1}(x_q) 
\end{bmatrix}
\tag{19}
\]

As such, assuming that the points where one observes the state are chosen to guarantee that Equation 19 has a unique solution, one can solve Equation 19 for a unique symmetric $P_{i+1}$. Moreover, since the general theory implies that Equation 19 has a positive definite solution, the unique symmetric solution of Equation 19 must, in fact, be positive definite. As such, one can implement the Adaptive Dynamic Programming Algorithm for a linear system by solving Equation 19 for $P_{i+1}$, instead of Equation 14.

Although both Equations 14 and 19 require that one solve a linear equation in $q = n(n+1)/2$ unknowns, the derivatives of the state are not required by the Kronecker product formulation of Equation 19, while the $V_{i+1}(x_j)$ are computed by integrating along the entire state trajectory, thereby filtering any measurement noise. On the other hand, the Linear Lyapunov formulation of Equation 14 requires that one observe the state at only $n$ points per iteration, and allows one to adapt the control multiple times along a given state trajectory. In both implementations one must assume that the $x_j$s are chosen to guarantee that the appropriate matrix will be invertible. Although this is generically the case, this assumption may fail when one reaches the “tail” of the state trajectory. As such, in our implementation of the algorithm, we dither the state in the “tail” of a trajectory and cease to update the control law when the state is near the singularity at $(0,0)$. 
To illustrate the implementation of the Adaptive Dynamic Programming Algorithm in the linear case, we developed an autolander for the NASA X-43 (or HyperX) [Singh, S.N., Yim, W., and W.R. Wells, (1995)]. The X-43, shown in Figure 1a, is an experimental testbed for an advanced scramjet engine operating in the Mach 7-10 range. In its present configuration, the X-43 is an expendable test vehicle, which will be launched from a Pegasus missile, perform a flight test program using its scramjet engine, after which it will crash into the ocean. The purpose of the simulation described here was to evaluate the feasibility of landing a follow-on series of X-43s. To this end, we developed an autolander for the X-43 designed to follow the glide path illustrated in Figure 1b, using the Adaptive Dynamic Programming Algorithm, and simulated its performance using a 6 degree-of-freedom linearized model of the X-43.

This model has eleven states indicated in Table 1 and four inputs indicated in Table 2. To stress the adaptive controller, the simulation used an extremely steep glide path angle. Indeed, so steep that the drag of the aircraft was initially insufficient to cause the aircraft to fall fast enough, requiring negative thrust. Of course, in practice one would never use such a steep glide slope, alleviating the requirement for thrust reversers in the aircraft. To illustrate the adaptivity of the controller, no a-priori knowledge of either the $A$ or $B$ matrices for the X-43 model was provided to the controller.

### Table 1: States of Linearized 6 DoF X-43 Model

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>Units</th>
<th>Trim Value</th>
<th>Initial Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>roll rate</td>
<td>p</td>
<td>rad/s</td>
<td>0.0</td>
<td>0.0850</td>
</tr>
<tr>
<td>yaw rate</td>
<td>r</td>
<td>rad/s</td>
<td>0.0</td>
<td>0.0850</td>
</tr>
<tr>
<td>pitch rate</td>
<td>q</td>
<td>rad/s</td>
<td>0.0</td>
<td>0.0850</td>
</tr>
<tr>
<td>roll</td>
<td>phi</td>
<td>rad</td>
<td>0.0</td>
<td>0.0850</td>
</tr>
<tr>
<td>yaw</td>
<td>psi</td>
<td>rad</td>
<td>-0.0778</td>
<td>0.0850</td>
</tr>
<tr>
<td>pitch</td>
<td>theta</td>
<td>rad</td>
<td>0.0</td>
<td>0.0850</td>
</tr>
<tr>
<td>vertical component of airspeed</td>
<td>w</td>
<td>ft/s (positive in down direction)</td>
<td>96.1442</td>
<td>-20.0000</td>
</tr>
<tr>
<td>forward component of airspeed</td>
<td>u</td>
<td>ft/s</td>
<td>0.0</td>
<td>-20.0000</td>
</tr>
<tr>
<td>side component of airspeed</td>
<td>v</td>
<td>ft/s</td>
<td>30.6225</td>
<td>-20.0000</td>
</tr>
</tbody>
</table>

Figure 1: a) NASA X-43 (HyperX) and b) its Glide Path
A “trim routine” is used to calculate the steady state settings of the aircraft control surfaces required to achieve the desired flight conditions, with the state variables and controlled inputs for the flight control system taken to be the deviations from the trim point. In the present example the trim control was calculated to maintain the aircraft on the specified glide slope. The performance of the X-43 autolander is summarized in Figure 2 where the altitude and lateral errors from the glide path and the vertical component of the aircraft velocity (sink rate) along the glide path are plotted. After correcting for the initial deviation from trim, the autolander brings the aircraft to, and maintains it on, the glide path. The control values employed by the autolander to achieve this level of performance are shown in Figure 3a, all of which are well within the dynamic range of the X-43’s controls, while the remaining states of the aircraft during landing are shown in Figures 3b, 3c, and 3d.

To evaluate the adaption rate of the autolander, the “cost-to-go” from the initial state is plotted as a function of time as the controller adapts in Figure 4. As expected, the cost-to-go jumps from the
low initial value associated with the initial guess, $P_o$, to a relatively high value, and then decays monotonically to the optimal value as the controller adapts. Although the theory predicts that the cost-to-go jump should occur in a single iteration, a filter was used to smooth the adaptive process.

Figure 3: a) Aircraft Controls (de, da, dr, and T); b) Orientation Rates (p, q, and r); c) Orientation Angles (phi, theta, and psi); and Airspeed Components (u, v, and w).

Figure 4: Cost-to-Go from Initial State as a Function of Time
in our implementation, which spreads the initial cost-to-go jump over several iterations

4. QUADRATIC APPROXIMATION OF THE COST FUNCTIONAL

The purpose of this section is to develop an approximate implementation of the Adaptive Dynamic Programming Algorithm in which the actual cost functional is approximated by a quadratic at each point in state space. To this end, we let \( a(x), b(x), q(x), \) and \( r(x) \) be \( C^\infty \) functions as defined in Section 2., and we let \( V_i(x) = x^TP_ix \) in which case \( \frac{dV_i}{dx}(x) = 2x^TP_i \) and \( k_i(x) = -r^{-1}(x)b^T(x)P_{i+1}x \) are also \( C^\infty \) functions. Substituting these expression into the Iterative HJB Equation we obtain

\[
 x^TP_{i+1}\dot{x} = -q(x) - x^TP_ib(x)r^{-1}(x)b^T(x)P_ix \tag{20}
\]

Following the model developed for the Kronecker Product formulation of the linear algorithm in Section 3., we observe the state at \( q = n(n+1)/2 \) points; \( x_j; j = 1, 2, ..., q \); and solve the set of simultaneous equations

\[
 x_j^TP_{i+1}\dot{x}_j = -q(x_j) - x_j^TP_ib(x_j)r^{-1}(x_j)b^T(x_j)P_ix_j; \quad j = 1, 2, ..., q \tag{21}
\]

or, equivalently in matrix form

\[
 \begin{bmatrix}
 x_1^T \otimes x_1 \\
 x_2^T \otimes x_2 \\
 \vdots \\
 x_q^T \otimes x_q 
\end{bmatrix} \vec{P}_{i+1} = \\
 -q(x_1) - x_1^TP_ib(x_1)r^{-1}(x_1)b^T(x_1)P_ix_1 \\
 -q(x_2) - x_2^TP_ib(x_2)r^{-1}(x_2)b^T(x_2)P_ix_2 \\
 \vdots \\
 -q(x_q) - x_q^TP_ib(x_q)r^{-1}(x_q)b^T(x_q)P_ix_q 
\]

Unlike the linear case, however, where one could reduce the number of degrees of freedom of \( P_{i+1} \) to \( q \) by requiring it to be hermitian, with positivity following from the fact that a positive definite solution of Equation 19 is known to exist, in the nonlinear case one cannot guarantee that a hermitian solution to Equation 22 will be positive definite. As such, we reduce the number of degrees of freedom of \( P_{i+1} \) to \( q \) by expressing it as the product of an upper triangular matrix with positive diagonal entries, \( U_{i+1} \), and its transpose, \( P_{i+1} = U_{i+1}^TU_{i+1} \), forcing \( P_{i+1} \) to be positive definite hermitian. Substituting this expression into Equation 22 we then solve the quadratic equation

---

Accurate Automation Corporation Proprietary - Nothing on This Page is Classified.
for $U_{i+1}$, yielding an approximation of the actual cost functional in the form $x^T U_{i+1} U_{i+1} x$.

To circumvent the differentiation of the observed state trajectory, one can formulate an alternative implementation of the above algorithm using observations obtained along a state trajectory, $x_j(x_0, \cdot)$, starting at initial state $x_0$ and converging to the singularity at $(0,0)$. As before, we approximate $V_i(x)$ by a quadratic, $x^T P_i x$, but work with the integral expression for $V_{i+1}(x)$ rather than the Iterative HJB equation, obtaining the set of equations

$$V_{i+1}(x_j) = \int_t^{\infty} [q(x_j(x_0, \lambda)) + x_j^T (x_0, \lambda) P_i b(x_j(x_0, \lambda)) r^{-1} (x_j(x_0, \lambda)) b^T (x_j(x_0, \lambda)) P_i x_j(x_0, \lambda)] d\lambda,$$

for a sequence of initial states $x_j = x_j(x_0, t_j); j = 1, 2, ..., q$; along $x_j(x_0, \cdot)$. Converting Equation 24 to Kronecker Product form now yields the $q x n^2$ matrix equation

$$\begin{bmatrix} x_1^T \otimes x_1^T \\ x_2^T \otimes x_2^T \\ \vdots \\ x_q^T \otimes x_q^T \end{bmatrix} \text{vec} (P_{i+1}) = \begin{bmatrix} V_{i+1}(x_1) \\ V_{i+1}(x_2) \\ \vdots \\ V_{i+1}(x_q) \end{bmatrix}$$

or equivalently, letting $P_{i+1} = U_{i+1}^T U_{i+1}$

$$\begin{bmatrix} x_1^T \otimes x_1^T \\ x_2^T \otimes x_2^T \\ \vdots \\ x_q^T \otimes x_q^T \end{bmatrix} \text{vec} (U_{i+1}^T U_{i+1}) = \begin{bmatrix} V_{i+1}(x_1) \\ V_{i+1}(x_2) \\ \vdots \\ V_{i+1}(x_q) \end{bmatrix}$$

which may be solved for $U_{i+1}$, yielding an approximation of the actual cost functional in the form
Although both Equations 22 and 26 require that one solve a quadratic equation in \( q = n(n+1)/2 \) unknowns, the derivatives of the state are not required by the formulation of Equation 26, while the \( V_{i+1}(x_j) \) are computed by integrating along the entire state trajectory, \( x_j(x_0,:) \), from \( x_0 \) to the singularity at \((0,0)\) thereby filtering out “most” of the measurement noise. On the other hand, the formulation of Equation 22 allows one to adapt the control multiple times along a given state trajectory. In both implementations one must assume that the \( x_j \)s are chosen to guarantee that the appropriate matrix will be invertible. As in the linear case, this assumption may fail when one reaches the “tail” of the state trajectory.

To evaluate the performance of the Adaptive Dynamic Programming Algorithm of Equation 22, we selected the system illustrated in Figure 5, in which a unit mass \((m = 1)\) is constrained to follow a parabolic track \((u = v^2)\) under the influence of horizontal \((f_H)\) and vertical \((f_V)\) forces, gravity \((g)\), and a small amount of viscous damping \((c = 0.001)\). This 2\textsuperscript{nd} order system, though somewhat academic, is highly nonlinear yet sufficiently well understood to allow us to evaluate the performance of the adaptive controller. Taking the state variables to be \( x_2 = u \) and \( x_1 = \dot{x}_2 \), this system has the input affine state model

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} =
\begin{bmatrix}
-4x_1^2x_2 - 2gx_2 - cx_1 \\
1 + 4x_2^2
\end{bmatrix}
+ \begin{bmatrix}
\frac{2x_2}{m(1 + 4x_2^2)} & \frac{1}{m(1 + 4x_2^2)} \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
f_V \\
f_H
\end{bmatrix}
\tag{27}
\]

Moreover, it is stable with a Lyapunov function taken to be the total (kinetic + potential) energy

\[
E = \frac{m}{2} (1 + 4x_2^2)x_1^2 + mgx_2^2
\tag{28}
\]
while the derivative of $E$ along the trajectories of the system takes the form

$$
\dot{E} = -c(1 + 4x_2^2)x_1^2
$$

(29)

To evaluate the performance of the Adaptive Dynamic Programming Algorithm without a-priori knowledge of either $a(x)$ or $b(x)$, a 1st order pre-compensator was used (increasing the order of the system to 3 as per Appendix A). The state response of the system starting from initial state $x_0 = [1, 2]^T$ at $t_0 = 0$ without control is shown in Figure 6a while the response of the controlled system is shown in Figure 6b. Here, the controlled response converges to the singularity at $(0,0)$ in less than 3 seconds with a reasonably smooth response, while the minimally damped uncontrolled system oscillates for several minutes before settling down.

5. RADIAL BASIS APPROXIMATION OF THE COST FUNCTIONAL

Unlike the nonlinear implementation of the Adaptive Dynamic Programming algorithm of Section 4., where one approximates $V_{i+1}(x)$ locally by a quadratic function of the state, the purpose of this section is to develop an implementation of the algorithm in which $V_{i+1}(x)$ is approximated nonparametrically by a linear combination of radial basis functions. Since radial basis functions are “local approximators,” however, one can update the approximation locally in a neighborhood of each trajectory, $x_i(x_0 \cdot)$, without waiting to explore the entire state space. As such, an approximation of $V_{i+1}(x)$, updated on the basis of a local exploration of the state space at each iteration, which is “potentially” globally convergent is obtained.

To demonstrate the radial basis function implementation of the Adaptive Dynamic Programming algorithm we chose a 4th order longitudinal model of the LoFLYTE® UAV (Unmanned Autonomous Vehicle) with a nonlinear pitching moment coefficient, illustrated in 7. The states of the model are indicated in Table 3, with the zero point in the state space shifted to correspond to a

![Figure 6: a). Uncontrolled and b). Controlled Response of Parabolically Constrained Mass](image)
selected trim point for the aircraft. The input for this model was the elevator deflection, with \( \delta_e = 0 \) in the model corresponding to a downward elevator deflection of \(-2.784^\circ\).

Table 3: States of Nonlinear Longitudinal LoFLYTE® Model

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>Units</th>
<th>Min</th>
<th>Trim Point</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>pitch rate</td>
<td>q</td>
<td>rad/s</td>
<td>-0.3491</td>
<td>0.0000</td>
<td>0.3491</td>
</tr>
<tr>
<td>pitch</td>
<td>theta</td>
<td>rad</td>
<td>-0.6850</td>
<td>-0.3359</td>
<td>0.0132</td>
</tr>
<tr>
<td>vertical component of airspeed</td>
<td>w</td>
<td>ft/s (positive in down direction)</td>
<td>-4.116</td>
<td>16.12</td>
<td>36.12</td>
</tr>
<tr>
<td>forward component of airspeed</td>
<td>u</td>
<td>ft/s</td>
<td>95.97</td>
<td>115.97</td>
<td>135.97</td>
</tr>
</tbody>
</table>

For our radial basis function implementation of the Adaptive Dynamic Programming algorithm, each axis of the state space is covered by 21 radial-basis-functions, from a predetermined minimum to a predetermined maximum value indicated in Table 3. As such, that part of state space where the UAV operates is covered by \(21 \times 21 \times 21 \times 21 = 194,481\) radial-basis-functions. Given the local nature of the radial basis functions, however, at any point in the state space \(V_{i+1}(x)\) is computed by summing the values a \(5 \times 5 \times 5 \times 5 = 625\) block of radial basis functions in a neighborhood of \(x\), corresponding to a 4-cube in state space centered at \(x\) with \(\Delta u = +/-4.76\) ft/s, \(\Delta v = +/-4.76\) ft/s, \(\Delta q = +/-0.083\) rad/s, and \(\Delta \theta = +/-0.083\) rad.

The following figures illustrate the performance of the radial basis function implementation of the Adaptive Dynamic Programming algorithm, learning an “optimal” control strategy from a given initial point in the state space, using the quadratic performance measure

\[
J = \int_0^\infty [x^T Q x + u^T R u] d\lambda
\]

with \(Q = \text{diag}(.0015, .0015, .0015, .0015)\) and \(R = [.005]\). The algorithm was initiated on the 0th iteration with \(k_0(x) = 0\). After the state converged to the trim point, the iteration count was incriminated, a radial basis function approximation of \(V_{i+1}(x)\) was computed, the new control law, \(k_{i+1}(x)\), was constructed, and the system was restarted at the same
initial state. In these simulations, the aircraft state was updated 100 times per second while the elevator deflection angle was updated 10 times per second. The performance of the radial basis function implementation of the Adaptive Dynamic Programming algorithm is illustrated in Figures 8 through 11, where we have plotted each of the key system variables on the 0th, 1st, 2nd, 3rd, 4th, 5th, iterations of the algorithm and the limiting value of these plots (at the 60th iteration).

The state variables of the aircraft are plotted in Figure 8. For each state variable the initial (0th)
constant at the trim point of -2.784° (indicated by “x”s). The elevator deflection then jumps to a high values on the 1st iteration (indicated by “o”s), and then converges toward the limiting value. All variables are well within a reasonable dynamic range for the LoFLYTE® UAV except for the initial drop of the aircraft (indicated by the initial positive spike in the vertical velocity curve of Figure 8a), due to the use of a “null” controller on the first iteration (which would not be the case for the actual aircraft where \( k_0(x) \) would be selected on the basis of prior simulation).

The performance of the Adaptive Dynamic Programming algorithm is illustrated in Figure 10 where the computed (Figure 10a) and radial basis function approximation (Figure 10b) of the optimal cost functional are plotted as a function of time along the state trajectory, on the 0th, 1st,
2\textsuperscript{nd}, 3\textsuperscript{rd}, 4\textsuperscript{th}, 5\textsuperscript{th}, and 60\textsuperscript{th} (limiting) iteration of the algorithm. In both cases, the initial estimate (indicated by “x”s) is low and converges upward to the limiting value, with the RBF approximation error decreasing in parallel with the adaption process. Finally, the cost-to-go based on the computed (“x”s) and radial basis function approximation (“o”s) of the optimal cost functional is plotted as a function of the iteration number in Figure 11. As predicted by the theory,

![Graph showing cost-to-go based on computed and RBF approximation](image)

Figure 11: Cost-to-Go based on the Computed (“x”s) and RBF Approximation (“o”s) of the Optimal cost functional vs. Iteration Number

the cost-to-go has an initial spike and then declines monotonically to the limiting value.

6. CONCLUSIONS

Our goal in the preceding has been to provide the framework for a family of Asymptotic Dynamic Programming algorithms by developing the general, if not directly applicable, theory and the three implementations of Sections 3.-5. Indeed, several alternative implementations come to mind. First, by taking advantage of the intrinsic adaptivity of the algorithm, one could potentially use a linear adaptive controller on a nonlinear system, letting it adapt to a different linearization of the plant at each point in state space, effectively implementing an “adaptive gain scheduler.” Secondly, since the control law is based on \( \frac{dV_i(x)}{dx} \), not \( V_i(x) \), any approximation of the cost functional should consider the gradient error as well as the direct approximation error. Therefore, in Section 5, one might replace the radial basis function approximation, which produces a “bumpy” Tchebychev-like approximation of \( V_i(x) \), with a “smoother” cubic spline approximation, or an alternative local approximator. Finally, by requiring the plant and performance measure matrices to be “real analytic” rather than \( C^\infty \) (and extending the proof of the theorem to guarantee that the matrices generated by the iterative process are also “real analytic”) one might consider the possibility of using analytic continuation to extrapolate local observations of the state space to the entire (or a larger region in the) state space, implementing the process with one of the modern symbolic mathematics codes.
7. REFERENCES

APPENDIX A. PRE-COMPENSATION PROCEDURE

The purpose of this appendix is to derive the precompensation technique originally described in [Saeks, R., and C. Cox, (1998)], which embeds the $b(x)$ matrix of an input affine plant into the $a(x)$ matrix of the combined pre-compensator / plant model, thereby allowing one to apply the Adaptive Dynamic Programming techniques developed in the present paper for a plant with an unknown $a(x)$ matrix, to a plant with both $a(x)$ and $b(x)$ unknown. This technique is illustrated in Figure 12, where, the pre-compensator is defined by any desired (controllable) input affine differential equation, $\dot{u} = \alpha(u) + \beta(u)v$, whose state vector is of the same dimension as the input vector for the given plant, with a singularity at $(u=0, v=0)$.

Now, the dynamics of the augmented plant, obtained by combining the pre-compensator with the original plant take the form

$$\begin{bmatrix} \dot{x} \\ \dot{u} \end{bmatrix} = \begin{bmatrix} a(x) + b(x)u \\ \alpha(u) + \beta(u)v \end{bmatrix} = \begin{bmatrix} a(x) + b(x)u \\ 0 \end{bmatrix} v + \begin{bmatrix} \alpha(u) \\ \beta(u) \end{bmatrix} v$$

which is also input affine, with the augmented state vector, $\tilde{x} = \begin{bmatrix} x \\ u \end{bmatrix}^T$ and a singularity at $(\tilde{x}=0, v=0)$. Moreover, all of the dynamics of the original plant are now embedded in the $\tilde{a}(\tilde{x})$ matrix of the augmented plant with $\tilde{b}(\tilde{x})$ known (since the dynamics of the precompensator are specified by the system designer). Furthermore, we may define an augmented performance measure by

$$\tilde{J} = \int_0^\infty [\tilde{l}(\tilde{x}(x_0, \lambda), v(x_0, \lambda))] d\lambda$$

$$= \int_0^\infty [l(x(x_0, \lambda), u(x_0, \lambda)) + l_v(v(x_0, \lambda))] d\lambda = J + \int_{t_0}^\infty [l_v(v(x_0, \lambda))] d\lambda$$

Figure 12: Control System with Pre-Compensator
where \( l_i(v(x_0, t)) \geq 0 \) with equality if and only if \( v = 0 \).

As such, one can apply the above described Adaptive Dynamic Programming Algorithm to a plant in which both \( a(x) \) and \( b(x) \) are unknown by applying the algorithm to the augmented system of Equation 30 with the augmented performance measure of Equation 31, yielding a control law of the form \( \tilde{k}^o(\tilde{x}) = \tilde{k}^o(x, u) \). This is, however, achieved at the cost of using a modified performance measure and increasing the dimension of the state space.

**APPENDIX B. PROOF OF THE ADAPTIVE DYNAMIC PROGRAMMING THEOREM**

The proof of the Adaptive Dynamic Programming Theorem follows the 4 steps indicated in Section 2.

1. **Show that** \( V_{i+1}(x) \) **and** \( k_{i+1}(x) \) **exist and are** \( C^\infty \) **functions with**

\[
V_{i+1}(x) > 0, \ x \neq 0; \ V_{i+1}(0) = 0; \ i = 0, 1, 2, ... .
\]

By construction \( V_{i+1}(x) > 0, \ x \neq 0; \ V_{i+1}(0) = 0 \), while the existence and smoothness of \( k_{i+1}(x) \) follows from that of \( V_{i+1}(x) \) since \( b(x) \) and \( r(x) \) are \( C^\infty \) functions and \( r^{-1}(x) \) exists.

As such, it suffices to show that \( V_{i+1}(x) \) exists and is a \( C^\infty \) function. Since \( V_{i+1}(x) \) is defined by the state trajectories generated by the \( i \)th control law, \( k_i(x) \), we begin by characterizing the properties of the state trajectories \( x_i(x_0, \cdot) \). In particular, since the control law and the plant are defined by \( C^\infty \) functions, the state trajectories are also \( C^\infty \) functions of both \( x_0 \) and \( t \) [Dieudonne, J., (1960)]. Furthermore, since \( k_i(x) \) is a stabilizing controller and the eigenvalues of \( \frac{dF_i(0)}{dx} \) have negative real parts, the state trajectories, \( x_i(x_0, \cdot) \), converge to zero exponentially [Halanay, A., and Rasvan, V., (1993)].

In addition to showing that the state trajectories, \( x_i(x_0, \cdot) \), are exponentially stable, we would also like to show that the partial derivatives of the state trajectories with respect to the initial condition, \( \frac{\partial^n x_i(x_0, \cdot)}{\partial x_0^n} \), are also exponentially stable. To this end we observe that \( \frac{\partial x_i(x_0, \cdot)}{\partial x_0} \) satisfies the differential equation

\[
\frac{\partial}{\partial t} \left[ \frac{\partial x_i(x_0, \cdot)}{\partial x_0} \right] = \frac{\partial F_i(x_i(x_0, \cdot))}{\partial x_0} = \frac{dF_i(x_i(x_0, \cdot))}{dx} \left[ \frac{\partial x_i(x_0, \cdot)}{\partial x_0} \right], \quad \frac{\partial x_i(x_0, 0)}{\partial x_0} = 1
\]
Since \( x_i(x_0, \cdot) \) is asymptotic to zero, Equation 32 reduces to the linear time-invariant differential equation

\[
\frac{\partial}{\partial t} \left[ \frac{\partial x_i(x_0, \cdot)}{\partial x_0} \right] = \left[ \frac{dF_i(0)}{dx} \right] \left[ \frac{\partial x_i(x_0, \cdot)}{\partial x_0} \right]; \quad \frac{\partial x_i(x_0, 0)}{\partial x_0} = 1
\]

(33)

for large \( t \). As such, the partial derivative of the state trajectory \( \frac{\partial x_i(x_0, \cdot)}{\partial x_0} \) with respect to the initial condition is exponentially stable since the eigenvalues of \( \frac{dF_i(0)}{dx} \) have negative real parts.

Applying the above argument inductively, we assume that \( x_i(x_0, \cdot) \) and \( \frac{\partial^j x_i(x_0, \cdot)}{\partial x_0^j} ; j = 1, 2, \ldots n-1 \); are exponentially stable and observe that \( \frac{\partial^n x_i(x_0, \cdot)}{\partial x_0^n} \) satisfies a differential equation of the form

\[
\frac{\partial}{\partial t} \left[ \frac{\partial^n x_i(x_0, \cdot)}{\partial x_0^n} \right] = \left[ \frac{dF_i(x_i(x_0, \cdot))}{dx} \right] \left[ \frac{\partial^n x_i(x_0, \cdot)}{\partial x_0^n} \right] + D(t); \quad \frac{\partial^n x_i(x_0, 0)}{\partial x_0^n} = 0
\]

(34)

where \( D(t) \) is a polynomial in \( x_i(x_0, \cdot) \) and the trajectories of the lower derivatives, \( \frac{\partial^j x_i(x_0, \cdot)}{\partial x_0^j} ; j = 1, 2, \ldots n-1 \). By the inductive hypothesis \( x_i(x_0, \cdot) \) and \( \frac{\partial^j x_i(x_0, \cdot)}{\partial x_0^j} ; j = 1, 2, \ldots n-1 \); are all exponentially convergent to zero and, therefore, so is \( D(t) \). As such, Equation 34 reduces to the linear time-invariant differential equation

\[
\frac{\partial}{\partial t} \left[ \frac{\partial^n x_i(x_0, \cdot)}{\partial x_0^n} \right] = \left[ \frac{dF_i(0)}{dx} \right] \left[ \frac{\partial^n x_i(x_0, \cdot)}{\partial x_0^n} \right]; \quad \frac{\partial^n x_i(x_0, 0)}{\partial x_0^n} = 0
\]

(35)

for large \( t \), implying that the \( n \)th partial derivative of the state trajectory \( \frac{\partial x_i(x_0, \cdot)}{\partial x_0} \) with respect to the initial condition is exponentially stable, since the eigenvalues of \( \frac{dF_i(0)}{dx} \) have negative
real parts. As such, $x_i(x_0, \cdot)$ and $\frac{\partial^n x_i(x_0, \cdot)}{\partial x_0^n} ; n = 1, 2, \ldots$ are exponentially convergent to zero.

See [Devinatz, A., and Kaplan, J.L., (1972)] for an alternative proof that the derivatives of the state trajectories with respect to the initial condition are exponentially convergent to zero, directly in terms of Equations 32 and 34.

To verify the existence of $V_{i+1}(x)$, we express $l(x_i(x_0, \cdot), u_i(x_0, \cdot))$ in the form

$$l(x_i(x_0, \cdot), u_i(x_0, \cdot)) = q(x) + k_i^T(x)r(x)k_i(x)$$

$$= q(x) + \frac{1}{4}\left[\frac{dV_i(x)}{dx}\right]b(x)r^{-1}(x)b^T(x)\left[\frac{dV_i(x)}{dx}\right]^T \equiv l_i(x_i(x_0, \cdot))$$

(36)

where the notation, $l_i(x_i(x_0, \cdot))$, is used to simplify the expression and emphasize that $l(x_i(x_0, \cdot), u_i(x_0, \cdot))$ is a function of the state trajectory. Now, expanding $q(x)$ as a power series around $x = 0$ and recognizing that $q(0) = 0$ and $\frac{dq(0)}{dx} = 0$, since $x = 0$ is a minimum of the positive definite function, $q(x)$, we obtain

$$q(x) = q(0) + \frac{dq(0)}{dx}x + \frac{d^2q(0)}{dx^2}x + o(\|x\|^3) = x\frac{d^2q(0)}{dx^2}x + o(\|x\|^3)$$

(37)

As such, there exists $K_1$ such that $q(x) < K_1\|x\|^2$ for small $x$. Similarly, upon expanding $\frac{dV_i(x)}{dx}$ in a power series around $x = 0$, and recognizing that $\frac{dV_i(0)}{dx} = 0$ since $x = 0$ is a minimum of $V_i$, we obtain

$$\frac{dV_i(x)}{dx} = \frac{dV_i(0)}{dx} + \frac{d^2V_i(x)}{dx^2}x + o(\|x\|^2) = \frac{d^2V_i(x)}{dx^2}x + o(\|x\|^2)$$

(38)

As such, there exists $K_2$ such that $\frac{dV_i(x)}{dx} < K_2\|x\|$ for small $x$. Finally, since $b(x)r(x)^{-1}b^T(x)$ is continuous at zero, there exists $K_3$ such that $b(x)r^{-1}(x)b^T(x) < K_3$ for small $x$. Substituting the inequalities $q(x) < K_1\|x\|^2$, $\frac{dV_i(x)}{dx} < K_2\|x\|$, and $b(x)r^{-1}(x)b^T(x) < K_3$ into Equation 36 therefore yields
As such,

\[ V_{i+1}(x) \equiv \int_0^\infty l(x_i(x_0, \lambda), u(u_i(x_0, \lambda)))d\lambda \]  

exists and is continuous in \( x_0 \), since the state trajectory, \( x_i(x_0, \cdot) \), is exponentially convergent to zero.

Finally, to verify that \( V_{i+1}(x) \) is a \( C^\infty \) function it suffices to show that trajectories \( \frac{d^n l_i(x_i(x_0, \cdot))}{dx_0^n} \) are integrable, in which case one can interchange the derivative and integral operators obtaining

\[ \frac{d^n V_{i+1}(x_0)}{dx_0^n} = \int_0^\infty \frac{d^n l_i(x_i(x_0, \cdot))}{dx_0^n} d\lambda \]  

Now,

\[ \frac{d l_i(x_i(x_0, \cdot))}{dx_0} = \frac{d}{dx} \frac{dl_i(x_i(x_0, \cdot))}{dx_0} \]  

while \( \frac{d^n l_i(x_i(x_0, \cdot))}{dx_0^n} \) is a sum of products composed of factors of the form \( \frac{dl_i(x_i(x_0, \cdot))}{dx} \) and \( \frac{d^k x_i(x_0, \cdot)}{dx_k} \), where every term has at least one factor of the latter type. Since the \( i^{th} \) closed loop system is stable each state trajectory, \( x_i(x_0, \cdot) \), is contained in a compact set and since \( l_i(x_i(x_0, \cdot)) \) is a \( C^\infty \) function the derivatives, \( \frac{dl_i(x_i(x_0, \cdot))}{dx} \), are bounded on the state trajectory, \( x_i(x_0, \cdot) \), while we have already shown that the derivatives of the state trajectories with respect to the initial conditions, \( \frac{d^k x_i(x_0, \cdot)}{dx_k} \), converge to zero exponentially. As such,
converges to zero exponentially and is therefore integrable, validating Equation 41 and verifying that \( V_{i+1}(x) \) is a \( C^\infty \) function.

2 Show that the Iterative Hamilton Jacobi Bellman Equation

\[
\frac{dV_{i+1}(x)}{dx} F_i(x) = -l(x, k_i(x))
\]

is satisfied, and that \( \frac{d^2V_{i+1}(0)}{dx^2} > 0 \); \( i = 0, 1, 2, \ldots \).

To verify the Iterative HJB equation we compute \( \frac{dV_{i+1}(x_i(x_0, t))}{dt} \) via the chain rule, obtaining

\[
\frac{dV_{i+1}(x_i(x_0, t))}{dt} = \frac{dV_{i+1}(x_i(x_0, t))}{dx} \frac{dx_i(x_0, t)}{dt} = \frac{dV_{i+1}(x_i(x_0, t))}{dx} F_i(x_i(x_0, t))
\]

and by directly differentiating the integral

\[
V_{i+1}(x_i(x_0, t)) = \int_0^\infty [l(x_i(x_i(x_0, t), \lambda), u_i(x_i(x_0, t), \lambda))] d\lambda
\]

Since there is a unique state trajectory passing through the state, \( x_i(x_0, t) \), the trajectory \( x_i(x_i(x_0, t), \cdot) \) must coincide with the tail, after time \( t \), of the trajectory \( x_i(x_0, \cdot) \) starting at \( x_0 \) at \( t_0 = 0 \). Translating this trajectory in time to start at \( t_0 = 0 \), then yields the relationship

\[
x_i((x_i(x_0, t), \lambda)) = x_i(x_0, \lambda + t); \quad \lambda \geq 0
\]

and similarly for the corresponding control. Substituting this expression into 44 and invoking the change of variable, \( \gamma = \lambda + t \), now yields

\[
V_{i+1}(x_i(x_0, t)) = \int_0^\infty l(x_i(x_0, \lambda + t), u_i(x_0, \lambda + t)) d\lambda = \int_t^\infty l(x_i(x_0, \gamma), u_i(x_0, \gamma)) d\gamma
\]
Adaptive Dynamic Programming

\[ \frac{dV_{i+1}(x_i(x_0, t))}{dt} = \frac{d}{dt} \int_{t}^{\infty} l(x_i(x_0, \gamma), u(u_i(x_0, \gamma))) d\gamma \]

(47)

\[ l(x_i(x_0, \gamma), u(u_i(x_0, \gamma))) \big|_{t}^{\infty} = -l(x_i(x_0, t), u(u_i(x_0, t))) \]

since, \( l(x_i(x_0, \cdot), u(u_i(x_0, \cdot))) \equiv l_i(x_i(x_0, \cdot)) \) is asymptotic to zero (See 1 above).

Finally, the Iterative HJB equations follows by equating the two expressions for \( \frac{dV_{i+1}(x_i(x_0, t))}{dt} \) of Equations 43 and 47.

To show that \( \frac{d^2V_{i+1}(0)}{dx^2} > 0 \), we note that \( \frac{dV_i(0)}{dx} = 0 \) since zero it is a minimum of \( V_i(x) \), and similarly for \( \frac{dV_{i+1}(0)}{dx} = 0 \), while \( F_i(0) = a(0) - \frac{1}{2}b(0)r^{-1}(0)b^T(0)\left[\frac{dV_i(0)}{dx}\right]^T = 0 \) since \( a(0) = 0 \). As such, taking the second derivative on both sides of the Iterative HJB Equation, evaluating it at \( x = 0 \), and deleting those terms which contain \( \frac{dV_i(0)}{dx}, \frac{dV_{i+1}(0)}{dx} \), or \( F_i(0) \) as a factor, yields

\[ 2\frac{d^2V_{i+1}(0)}{dx^2} \frac{dF_i(0)}{dx} = -\left[\frac{d^2q(0)}{dx^2} + \frac{1}{2}\left[\frac{d^2V_i(0)}{dx^2}\right]^T (b(x)r^{-1}(x)b^T(x)) \left[\frac{d^2V_i(0)}{dx^2}\right]^T \right] \]

(48)

Since the right side of Equation 48 is symmetric so is the left side. As such, one can replace one of the two \( \frac{d^2V_{i+1}(0)}{dx^2} \frac{dF_i(0)}{dx} \) terms on the left side of Equation 48 by its transpose yielding the Linear Lyapunov Equation [Barnett, S., (1971)]

\[ \left[\frac{dF_i(0)}{dx}\right]^T \frac{d^2V_{i+1}(0)}{dx^2} + \frac{d^2V_{i+1}(0)}{dx^2} \left[\frac{dF_i(0)}{dx}\right] = -\left[\frac{d^2q(0)}{dx^2} + \frac{1}{2}\left[\frac{d^2V_i(0)}{dx^2}\right]^T (b(x)r^{-1}(x)b^T(x)) \left[\frac{d^2V_i(0)}{dx^2}\right]^T \right] \]

(49)
where we have used the fact that \( \frac{d^2 V_{i+1}(0)}{dx^2} \) is symmetric in deriving Equation 49. Moreover, since the eigenvalues of \( \frac{dF_i(0)}{dx} \) have negative real parts, while \( \frac{d^2 q(0)}{dx^2} > 0 \) and

\[
\left[ \frac{d^2 V_i(0)}{dx^2} \right] (b(x)r^{-1}(x)b^T(x)) \left[ \frac{d^2 V_i(0)}{dx^2} \right]^T \geq 0,
\]

the unique symmetric solution of Equation 49 is positive definite [Barnett, S., (1971)]. As such, \( \frac{d^2 V_{i+1}(0)}{dx^2} > 0 \), as required.

3 Show that \( V_{i+1}(x) \) is a Lyapunov Function for the Closed Loop System, \( F_{i+1} \), and that the eigenvalues of \( \frac{dF_{i+1}(0)}{dx} \) have negative real parts; \( i = 0, 1, 2, \ldots \).

To show that \( k_{i+1} \) is a stabilizing control law for the plant, we show that \( V_{i+1}(x) \) is a Lyapunov Function for the closed loop system, \( F_{i+1} \); \( i = 0, 1, 2, \ldots \). Since \( V_{i+1}(x) \) is positive definite it suffices to show that the derivative of \( V_{i+1}(x) \) along the state trajectories defined by the control law \( k_{i+1} \),

\[
\frac{dV_{i+1}(x_{i+1}(x_0, t))}{dt}
\]

is negative definite. To this end we use the chain rule to compute

\[
\frac{dV_{i+1}(x_{i+1}(x_0, t))}{dt} = \frac{d[V_{i+1}(x_{i+1}(x_0, t))]}{dx} \frac{dx_{i+1}(x_0, t)}{dt}
\]

\[
= \frac{d[V_{i+1}(x_{i+1}(x_0, t))]}{dx} F_{i+1}(x_{i+1}(x_0, t))
\]  \hspace{1cm} (50)

Now, upon substituting

\[
F_{i+1}(x_{i+1}) = a(x_{i+1}) - \frac{1}{2} b(x_{i+1})r^{-1}(x_{i+1})b^T(x_{i+1}) \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right]^T
\]  \hspace{1cm} (51)

(where we have used \( x_{i+1} \) as a shorthand notation for \( x_{i+1}(x_0, t) \)) into Equation 50 we obtain

\[
\frac{dV_{i+1}(x_{i+1}(x_0, t))}{dt} = \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right] a(x_{i+1}) - \frac{1}{2} \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right] b(x_{i+1})r^{-1}(x_{i+1})b^T(x_{i+1}) \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right]^T
\]  \hspace{1cm} (52)

Similarly, we may substitute the equality
Adaptive Dynamic Programming

\[ F_i(x_{i+1}) = a(x_{i+1}) - \frac{1}{2} b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[ \frac{dV_i(x_{i+1})}{dx} \right]^T \]  

(53)

into the Iterative HJB Equation obtaining

\[
\left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right] a(x_{i+1})
\]

(54)

\[
= \frac{1}{2} \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[ \frac{dV_i(x_{i+1})}{dx} \right]^T - l(x_{i+1}, k_i(x_{i+1}))
\]

Substituting Equation 54 into Equation 52 now yields

\[
\frac{dV_{i+1}(x_{i+1}(x_0, t))}{dt}
\]

(55)

\[
= \frac{1}{2} \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[ \frac{dV_i(x_{i+1})}{dx} \right]^T - l(x_{i+1}, k_i(x_{i+1}))
\]

while expressing \( l(x_{i+1}, k_i(x_{i+1})) \) in the form

\[
l(x_{i+1}, k_i(x_{i+1})) = q(x_{i+1}) + \frac{1}{4} \left[ \frac{dV_i(x_{i+1})}{dx} \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[ \frac{dV_i(x_{i+1})}{dx} \right]^T
\]

(56)

and substituting this expression into Equation 55 yields

\[
\frac{dV_{i+1}(x_{i+1}(x_0, t))}{dt}
\]

(57)

\[
= \frac{1}{2} \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[ \frac{dV_i(x_{i+1})}{dx} \right]^T - q(x_{i+1})
\]

\[
- \frac{1}{4} \left[ \frac{dV_i(x_{i+1})}{dx} \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[ \frac{dV_i(x_{i+1})}{dx} \right]^T
\]

\[
- \frac{1}{2} \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[ \frac{dV_{i+1}(x_{i+1})}{dx} \right]^T
\]

Finally, upon completing the square, Equation 57 reduces to
As such, is negative definite, verifying that $V_{i+1}(x)$ is a Lyapunov function for $F_{i+1}$ and that $k_{i+1}$ is a stabilizing controller for the plant, as required.

To show that the eigenvalues of have negative real parts, we note that since it is a minimum of $V_{i+1}(x)$, and similarly for $V_i(x)$, while $a(0) = 0$. Now, substituting Equation 50 for the left side of Equation 58, taking the second derivative on both sides of the resultant equation, evaluating it at $x=0$, and deleting those terms which contain $dV_i(x)$ or $F_i(0)$ as a factor, yields

$$\frac{dV_{i+1}(x_{i+1}(x_0, t))}{dt} = -q(x_{i+1})$$

$$-\frac{1}{4}\left[d\left[V_{i+1}(x_{i+1}) - V_i(x_{i+1})\right] \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[\frac{d[V_{i+1}(x_{i+1}) - V_i(x_{i+1})]}{dx}\right]^T$$

$$-\frac{1}{4}\left[dV_{i+1}(x_{i+1}) \right] b(x_{i+1}) r^{-1}(x_{i+1}) b^T(x_{i+1}) \left[\frac{dV_{i+1}(x_{i+1})}{dx}\right]^T$$

As such, $\frac{dV_{i+1}(x_{i+1}(x_0, t))}{dt} < 0$; $x_{i+1}(x_0, t) \neq 0$ is negative definite, verifying that $V_{i+1}(x)$ is a Lyapunov function for $F_{i+1}$ and that $k_{i+1}$ is a stabilizing controller for the plant, as required.

To show that the eigenvalues of $\frac{dF_{i+1}(0)}{dx}$ have negative real parts, we note that $\frac{dV_{i+1}(0)}{dx} = 0$ since it is a minimum of $V_{i+1}(x)$, and similarly for $\frac{dV_i(0)}{dx} = 0$, while $F_{i+1}(0) = a(0) - \frac{1}{2} b(0) r^{-1}(0) b^T(0) \left[\frac{dV_{i+1}(0)}{dx}\right]^T = 0$ since $a(0) = 0$. Now, substituting Equation 50 for the left side of Equation 58, taking the second derivative on both sides of the resultant equation, evaluating it at $x = 0$, and deleting those terms which contain $\frac{dV_{i+1}(0)}{dx}$, $\frac{dV_i(0)}{dx}$, or $F_{i+1}(0)$ as a factor, yields

$$\frac{2 d^2 V_{i+1}(0) dF_{i+1}(0)}{dx^2}$$

$$= -\frac{d^2 q(0)}{dx^2} - \frac{1}{2} \left[\frac{d^2 V_{i+1}(0)}{dx^2}\right] b(0) r^{-1}(0) b^T(0) \left[\frac{d^2 V_{i+1}(0)}{dx^2}\right]^T$$

$$-\frac{1}{4}\left[d^2[V_{i+1}(0) - V_i(0)] \right] b(0) r^{-1}(0) b^T(0) \left[\frac{d^2[V_{i+1}(0) - V_i(0)]}{dx^2}\right]^T$$

Now, since the right side of Equation 59 is symmetric so is the left side and, as such, we may equate the left side of Equation 59 to its hermitian part. Moreover, since $-\frac{d^2 q(0)}{dx^2} < 0$ while the second and third terms on the right side of 59 are negative symmetric, the right side of
Equation 59 reduces to a negative definite symmetric matrix, -Q. As such, Equation 59 may be expressed in the form

$$\begin{bmatrix} \frac{dF_{i+1}(0)}{dx} \\ \frac{d^2V_{i+1}(0)}{dx^2} \end{bmatrix}^T \frac{d^2V_{i+1}(0)}{dx^2} + \frac{d^2V_{i+1}(0)}{dx^2} \begin{bmatrix} \frac{dF_{i+1}(0)}{dx} \\ \frac{d^2V_{i+1}(0)}{dx^2} \end{bmatrix} = -Q \quad (60)$$

Finally, to verify that the eigenvalues of $\frac{dF_{i+1}(0)}{dx}$ have negative real parts we let $\lambda$ be an arbitrary eigenvalue of $\frac{dF_{i+1}(0)}{dx}$ with eigenvector $v$. As such, $\frac{dF_{i+1}(0)}{dx}v = \lambda v$, while post-multiplying this relationship by $v^T \frac{d^2V_{i+1}(0)}{dx^2}$ yields

$$v^T \frac{d^2V_{i+1}(0)}{dx^2} \frac{dF_{i+1}(0)}{dx}v = \lambda v^T \frac{d^2V_{i+1}(0)}{dx^2}v \quad (61)$$

Now, upon taking the complex conjugate of Equation 61 and adding it to Equation 61, we obtain

$$v^T \left[ \begin{bmatrix} \frac{dF_{i+1}(0)}{dx} \\ \frac{d^2V_{i+1}(0)}{dx^2} \end{bmatrix}^T \frac{d^2V_{i+1}(0)}{dx^2} \right] \frac{dF_{i+1}(0)}{dx}v + \frac{d^2V_{i+1}(0)}{dx^2} \left[ \begin{bmatrix} \frac{dF_{i+1}(0)}{dx} \\ \frac{d^2V_{i+1}(0)}{dx^2} \end{bmatrix} \right]v = 2 \text{Re}(\lambda) v^T \frac{d^2V_{i+1}(0)}{dx^2}v \quad (62)$$

Finally, substituting Equation 60 in Equation 62 yields

$$-v^T Qv = 2 \text{Re}(\lambda) v^T \frac{d^2V_{i+1}(0)}{dx^2}v \quad (63)$$

from which it follows that $\text{Re}(\lambda) < 0$, since $\frac{d^2V_{i+1}(0)}{dx^2} > 0$ (see part 2 of the proof), and

$$-v^T Qv < 0.$$

Show that the sequence of cost functional / control law pairs, $(V_{i+1}, k_{i+1})$, is convergent.

The key step in our convergence proof is to show that

$$\frac{d[V_{i+1}(x_i(x_0, t)) - V_i(x_i(x_0, t))]}{dt} = \frac{d[V_{i+1}(x_i(x_0, t))]}{dt} - \frac{d[V_i(x_i(x_0, t))]}{dt} \quad (64)$$

is positive along the trajectories defined by the control law, $k_i$. Substituting Equation 36 into Equation 47 yields
Adaptive Dynamic Programming

\[
\frac{d[V_{i+1}(x_i(x_0, t))]}{dt} = -l(x_i, u_i) = -q(x_i) - \frac{1}{4}\left[\frac{dV_i(x_i)}{dx}\right]b(x_i)r^{-1}(x_i)b^T(x_i)\left[\frac{dV_i(x_i)}{dx}\right]^T \tag{65}
\]

(where we have used \(x_i\) as a shorthand notation for \(x_i(x_0, t)\) and similarly for \(u_i\)) while one can obtain an expression for \(\frac{d[V_i(x, t)]}{dt}\) from Equation 58 by replacing the index “\(i+1\)” by the index “\(i\)”

\[
\frac{dV_i(x_i(x_0, t))}{dt} = -q(x_i) - \frac{1}{4}\left[\frac{dV_i(x_i)}{dx}\right]b(x_i)r^{-1}(x_i)b^T(x_i)\left[\frac{dV_i(x_i)}{dx}\right]^T \tag{66}
\]

which is valid for \(i = 1, 2, 3, ...\) after reindexing. Finally, substituting Equations 65 and 66 into Equation 64 yields

\[
\frac{d[V_{i+1}(x_i(x_0, t)) - V_i(x_i(x_0, t))]}{dt} = -q(x_i) + \frac{1}{4}\left[\frac{dV_i(x_i)}{dx}\right]b(x_i)r^{-1}(x_i)b^T(x_i)\left[\frac{dV_i(x_i)}{dx}\right]^T
\]

\[
\quad \quad \quad + \frac{1}{4}\left[\frac{dV_i(x_i)}{dx}\right]b(x_i)r^{-1}(x_i)b^T(x_i)\left[\frac{dV_{i+1}(x_i)}{dx}\right]^T \tag{67}
\]

\[
\quad \quad \quad = \frac{1}{4}\left[\frac{dV_i(x_i)}{dx}\right]b(x_i)r^{-1}(x_i)b^T(x_i)\left[\frac{dV_i(x_i)}{dx}\right]^T > 0
\]

for \(i = 1, 2, 3, ...\).

Since \(F_i\) is asymptotically stable, its state trajectories, \(x_i(x, \cdot)\), converge to zero, and hence so does \(V_{i+1}(x_i(x_0, \cdot)) - V_i(x_i(x_0, \cdot))\). Since \(\frac{d[V_{i+1}(x) - V_i(x)]}{dt} > 0\) on these trajectories, however, this implies that \(V_{i+1}(x_i(x_0, \cdot)) - V_i(x_i(x_0, \cdot)) < 0\) on the trajectories of \(F_i; i = 1, 2, 3, ...\). Since every point, \(x\), in the state space lies along some trajectory of \(F_i; x = x_i(x_0, t)\), however, this implies that \(V_{i+1}(x) - V_i(x) < 0\) for all \(x\) in the state space, or equivalently, \(V_{i+1}(x) < V_i(x)\) for all \(x; i = 1, 2, 3, ...\). As such, \(V_{i+1}(x); i = 1, 2, 3, ...\) is a decreasing sequence of positive numbers; \(i = 1, 2, 3, ...\);
and is therefore convergent (as is the sequence, $V_{i+1}(x); i = 0, 1, 2, ...$; since the behavior of the first entry of a sequence does not affect its convergence), completing the proof of the Adaptive Dynamic Programming Theorem.