Adaptive Dynamic Programming

John J. Murray, *Senior Member, IEEE*, Chadwick J. Cox, George G. Lendaris, *Life Fellow, IEEE*, and Richard Saeks, *Fellow, IEEE*

***Abstract*—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 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* (ADPA) which fuses *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.**

Specifically, the algorithm is initialized with a (stabilizing) cost functional and the system is run with the corresponding control law (defined by the Hamilton–Jacobi–Bellman equation), with the resultant state trajectories used to update the cost functional in a soft computing mode. Hard computing techniques are then used to show that this process is globally convergent with stepwise stability to the optimal cost functional/control law pair for an (unknown) input affine system with an input quadratic performance measure (modulo the appropriate technical conditions).

Three specific implementations of the ADPA are developed for 1) the linear case, 2) for the nonlinear case using a locally quadratic approximation to the cost functional, and 3) the nonlinear case using a radial basis function approximation of the cost functional; illustrated by applications to flight control.

***Index Terms*—Adaptive control, adaptive critic, dynamic programming, nonlinear control, optimal control.**

I. Introduction

THE PRESENT work has its roots in the approximate dynamic programming/adaptive critic concept [2], [30], [20], [32], [16], in which soft computing techniques are used to approximate the solution of a dynamic programming algorithm without the explicit imposition of a stability or convergence constraint, and the authors' stability criteria for these algorithms [6], [24]. Alternatively, a number of authors have combined hard and soft computing techniques to develop tracking controllers. These include Lyapunov synthesis techniques using both neural [25], [28], [18], [5], [21] and fuzzy learning laws [28], [29], [17], sliding mode techniques [31], and input–output techniques [9]. The objective of the present paper is to describe an *adaptive dynamic programming algorithm* (ADPA) 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 [3], [4], [19], which one solves for the *optimal cost functional*, $V^o(x_0, t_0)$. 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, $k^o(x, t_0)$, yielding an optimal cost functional/optimal control law pair, (V^o, k^o).

Although direct solution of the HJB equation is computationally intense (the so-called "curse of dimensionality"), the HJB equation and the relationship between V^o and the corresponding control law k^o, derived therefrom, serves as the basis of the ADPA developed in this 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^o, k^o) 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 (it took) to take the initial state x_0 at time t_0 to the final state, using control law k_i.
- Take k_{i+1} to be the corresponding control law derived from V_{i+1} via the HJB equation.
- Iterating the process until it converges.

Although this algorithmic process is similar to many of the soft computing algorithms which have been proposed for optimal control [16], [20], [30], [32], it is supported by a hard convergence and stability proof. Indeed, in Section II and in [24], it is shown that (with the technical assumptions defined in Section II) this process is

- *globally convergent* to
- the *optimal cost functional*, V^o, and the *optimal control law*, k^o, 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

Manuscript received March 20, 2001; revised June 11, 2002 This work was supported in part by NSF SBIR Grants DMI-9660604, DMI-9860370, and DMI-9983287, NASA Ames SBIR Contracts NAS2-98016 and NAS2-20008, and NSF Grant ECS-9904378.

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

C. J. Cox and R. Saeks are with Accurate Automation Corporation, Chattanooga, TN 37421 USA (e-mail: richard@saeks.org).

G. G. Lendaris is with Accurate Automation Corporation, Chattanooga, TN 37421 USA, on sabbatical from Portland State University, Portland, OR 97207 USA (e-mail: lendaris@ieee.org).

Digital Object Identifier 10.1109/TSMCC.2002.801727.

one is going to run the system for any length of time with control law k_i to generate data for the next iteration, 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. This is achieved by showing that V_i is a Lyapunov function for the feedback system with controller k_i, generalizing the classical result [19] that V^o is a Lyapunov function for the feedback system with controller k^o.

An analysis of the above algorithm (see Section II for additional details) will reveal that *a priori* knowledge of the state dynamics matrix is not required to implement the algorithm. Moreover, the requirement that the input matrix be known (to compute k_{i+1} from V_{i+1}), can be circumvented by the precompensator technique described in Appendix A. As such, the above described ADPA achieves one of the primary goals of soft control; applicability 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 ADPA, 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 III–V of this paper are devoted to the development of three approximate implementations of the ADPA 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;
- 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.

II. Adaptive Dynamic Programming Algorithm

In the formulation of the ADPA 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

$$\dot{x} = f(x, u) \equiv a(x) + b(x)u; \qquad x(t_0) = x_0 \tag{1}$$

with input quadratic utility function $l(x, u) \equiv [q(x) + u^T r(x) u]$ and performance measure

$$\begin{aligned} J &= \int_{t_0}^{\infty} l(x(x_0, \lambda), u(x_0, \lambda))\, d\lambda \\ &\equiv \int_{t_0}^{\infty} \left[q(x(x_0, \lambda)) + u^T(x_0, \lambda) r(x(x_0, \lambda)) u(x_0, \lambda)\right] d\lambda. \end{aligned} \tag{2}$$

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

1) $a(0) = 0$, producing a singularity at $(x, u) = (0, 0)$;
2) the eigenvalues of $da(0)/dx$ have negative real parts, i.e., the linearization of the uncontrolled plant at zero is exponentially stable;
3) $q(x) > 0$, $x \neq 0$; $q(0) = 0$;
4) $q(x)$ has a positive definite Hessian at $x = 0$, $d^2q(0)/dx^2 > 0$, i.e., any nonzero state is penalized independently of the direction from which it approaches 0;
5) $r(x) > 0$ for all x.

The goal of the ADPA is to use soft computing techniques 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 with hard convergence and stability criteria.

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 zero; 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 HJB 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 HJB equation then takes the form

$$\frac{dV^o(x)}{dx} F^o(x) = -l(x, k^o(x)) = -q(x) - k^{oT}(x) r(x) k^o(x) \tag{3}$$

in the time-invariant case [19].

Differentiating the HJB equation (3) with respect to $u^o = k^o(x)$ now yields

$$\frac{dV^o(x)}{dx} b(x) = -2k^{oT}(x) r(x) \tag{4}$$

or equivalently

$$u = k^o(x) = -\frac{1}{2} r^{-1}(x) b^T(x) \left[\frac{dV^o(x)}{dx}\right]^T \tag{5}$$

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 (5), though a similar implicit relationship can be derived in the general case. (See [24] for a derivation of this result.)

Given the above preparation, we may now formulate the desired adaptive dynamic programming learning 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^∞ function, $V_0(x) > 0,\ x \neq 0$; $V_0(0) = 0$, with a positive definite Hessian at $x = 0$, $d^2V_0(0)/dx^2 > 0$; and $k_0(x)$ is the C^∞ control law, $u = k_0(x) = -(1/2)r^{-1}(x)b^T(x)[dV_0(x)/dx]^T$.
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

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

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 precompensator at the cost of increasing its dimensionality, as shown in Appendix A. As such, the ADPA can be applied to plants with completely unknown dynamics, which is a primary goal of soft control. Unlike many soft control algorithms, however, it is fused with a rigorous convergence and stability theorem summarized below.

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 ADPA to plants with unknown dynamics is only meaningful in the context of the approximate implementations of Sections III–V, where only a local search or 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:

$$\begin{aligned}\dot{x} &= F_i(x) \equiv a(x) + b(x)k_i(x)\\ &= a(x) - \frac{1}{2}\, b(x)r^{-1}(x)b^T(x)\left[\frac{dV_i(x)}{dx}\right]^T.\end{aligned} \tag{6}$$

To initialize the ADPA 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 ε [though in practice we often take $k_0(x) = 0$]. Similarly, for a stabilizable plant, one can "prestabilize" the plant with any desired stabilizing control law such that $d^2V_0(0)/dx^2 > 0$ and the eigenvalues of $dF_0(0)/dx$ have negative real parts; and then initialize the ADPA 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, thereby reducing the scope of the required search.

The ADPA is characterized by the following Theorem.

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

1) $V_{i+1}(x)$ and $k_{i+1}(x)$ exist, where $V_{i+1}(x)$ and $k_{i+1}(x)$ are C^∞ functions with $V_{i+1}(x) > 0,\ x \neq 0$; $V_{i+1}(0) = 0$; $d^2V_{i+1}(0)/dx^2 > 0$; $i = 0, 1, 2, \ldots$.
2) The control law, k_{i+1}, stabilizes the plant [with Lyapunov function $V_{i+1}(x)$] for all $i = 0, 1, 2, \ldots$, and the eigenvalues of $dF_{i+1}(0)/dx$ have negative real parts.
3) The sequence of cost functionals, V_{i+1}, converge to the optimal cost functional, V^o.

Note that in 2), the existence of the Lyapunov function $V_{i+1}(x)$ together with the eigenvalue condition on $dF_{i+1}(0)/dx$ implies that the closed loop system, $F_{i+1}(x)$, is exponentially stable [12], 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 [24]. The proof includes four steps, as follows.

1) Show that $V_{i+1}(x)$ and $k_{i+1}(x)$ exist and are C^∞ functions, with $V_{i+1}(x) > 0,\ x \neq 0$; $V_{i+1}(0) = 0$; $i = 0, 1, 2, \ldots$: The first step required to prove that $V_{i+1}(x)$ and $k_{i+1}(x)$ exist and are C^∞ 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 $dF_i(0)/dx$ have negative real values, however, asymptotic stability implies exponential stability [12], 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 $dF_i(0)/dx$, which is exponentially stable since the eigenvalues of $dF_i(0)/dx$ have negative real values. See [12] for the details of this theorem.

Similarly, one can show that the derivatives of the state trajectories with respect to the initial condition, $\partial^n x_i(x_0, \cdot)/\partial x_0^n$, 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 $dF_i(0)/dx$. 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 $l(x_i(x_o, \cdot), u_i(x_o, \cdot))$, 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^∞ functions, since F_i is a C^∞ function [8]. 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^∞ 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 $d^2V_{i+1}(0)/dx^2 > 0$; $i = 0, 1, 2, \ldots$.

The iterative HJB equation, which may be used as an alternative to (7) for implementing the ADPA, is derived by computing $dV_{i+1}(x_i(x_0, t))/dt$ via the chain rule to obtain the left side of the iterative HJB equation, and by directly differentiating (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 $dV_{i+1}(0)/dx$ or $F_i(0)$, both of which are zero, one obtains the Linear Lyapunov equation

$$\begin{aligned}\left[\frac{dF_i(0)}{dx}\right]^T \frac{d^2V_{i+1}(0)}{dx^2} + \frac{d^2V_{i+1}(0)}{dx^2}\left[\frac{dF_i(0)}{dx}\right] \\ = -\left[\frac{d^2q(0)}{dx^2} + \frac{1}{2}\left[\frac{d^2V_{i+1}(0)}{dx^2}\right]\right. \\ \left.\cdot \left(b(x)r^{-1}(x)b^T(x)\right)\left[\frac{d^2V_{i+1}(0)}{dx^2}\right]^T\right]. \qquad (9)\end{aligned}$$

Now, since the eigenvalues of $dF_i(0)/dx$ have negative real parts, and the right side of (9) is a negative definite symmetric matrix, the unique symmetric solution of the Linear Lyapunov equation (9) is positive definite [1] and, as such, $d^2V_{i+1}(0)/dx^2 > 0$, 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 $dF_{i+1}(0)/dx$ have negative real parts; $i = 0, 1, 2, \ldots$: This is achieved by directly computing $dV_{i+1}(x_{i+1}(x_0, t))/dt$, 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 $i = 0, 1, 2, \ldots$.

To show that the eigenvalues of $dF_{i+1}(0)/dx$ have negative real parts, we use an argument similar to that used in 2, taking the second derivative of the expression derived for $(dV_{i+1}(x)/dx)F_{i+1}(x) = dV_{i+1}(x_{i+1}(x_0, t))/dt$ derived above.

4) Show that the sequence of cost functionals, V_{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, $d[V_{i+1}(x_i(x_0, t)) - V_i(x_i(x_0, t))]/dt > 0$, for $i = 1, 2, 3, \ldots$. 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 $d[V_{i+1}(x_i(x_0, t)) - V_i(x_i(x_0, t))]/dt > 0$ 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; $i = 1, 2, 3, \ldots$. 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; $i = 1, 2, 3, \ldots$. As such, V_{i+1} is a decreasing sequence of positive functions; $i = 1, 2, 3, \ldots$; and is therefore convergent (as is the sequence V_{i+1}; $i = 0, 1, 2, \ldots$; 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 III–V, 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 $i = 1, 2, 3, \ldots$.

III. Linear Case

The purpose of this section is to develop an implementation of the ADPA for the linear case, where local exploration of the state space at each iteration of the algorithm is sufficient, yielding a computationally tractable algorithm. As above, the linear algorithm preserves the fused soft computing/hard computing character of the general ADPA, combining soft computing techniques to iteratively solve the Matrix Riccati equation in real time for a plant with unknown dynamics, with hard computing techniques to guarantee convergence of the algorithm *and* stepwise stability of the controller.

For this purpose, we consider a linear time-invariant plant

$$\dot{x} = Ax + Bu; \qquad x(t_0) = x_0 \qquad (10)$$

with the quadratic performance measure

$$J = \int_{t_0}^{\infty} \left[x^T(x_0, \lambda)Qx(x_0, \lambda) + u^T(x_0, \lambda)Ru(x_0, \lambda)\right] d\lambda. \qquad (11)$$

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

To implement the ADPA in the linear case, we initialize the algorithm with a quadratic cost functional, $V_0(x) = x^T P_0 x$ and $K_0 = -R^{-1}B^T P_0$. Assuming that $V_i(x) = x^T P_i x$ is quadratic and $K_i = -R^{-1}B^T P_i$, $F_i(x) = [A - BK_i]x = [A - BR^{-1}B^T P_i]x \equiv F_i x$; 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 $x_i(x_0, t) = e^{F_i t}x_0$, while the corresponding control is $u_i(x_0, t) = K_i e^{F_i t}x_0$. As such

$$\begin{aligned} V_{i+1}(x_0) &= \int_0^{\infty} \left[x_i^T(x_0, \lambda)Qx_i(x_0, \lambda) + u_i^T(x_0, \lambda)Ru_i(x_0, \lambda)\right] d\lambda \\ &= \int_0^{\infty} \left[x_0^T e^{F_i^T \lambda}Qe^{F_i \lambda}x_0 + x_0^T e^{F_i^T \lambda}K_i^T QK_i e^{F_i \lambda}x_0\right] d\lambda \\ &= x_0^T \left[\int_0^{\infty} e^{F_i^T t}\left[Q + K_i^T RK_i\right]e^{F_i \lambda}\, d\lambda\right]x_0 \\ &= x_0^T \left[\int_0^{\infty} e^{F_i^T t}\left[Q + P_i BR^{-1}B^T P_i\right]e^{F_i \lambda}\, d\lambda\right]x_0 \\ &\equiv x_0^T P_{i+1}x_0. \end{aligned} \qquad (12)$$

Now, since F_i is asymptotically stable, the integral of (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 [1]

$$P_{i+1}F_i + F_i^T P_{i+1} = -\left[Q + P_i BR^{-1}B^T P_i\right]. \qquad (13)$$

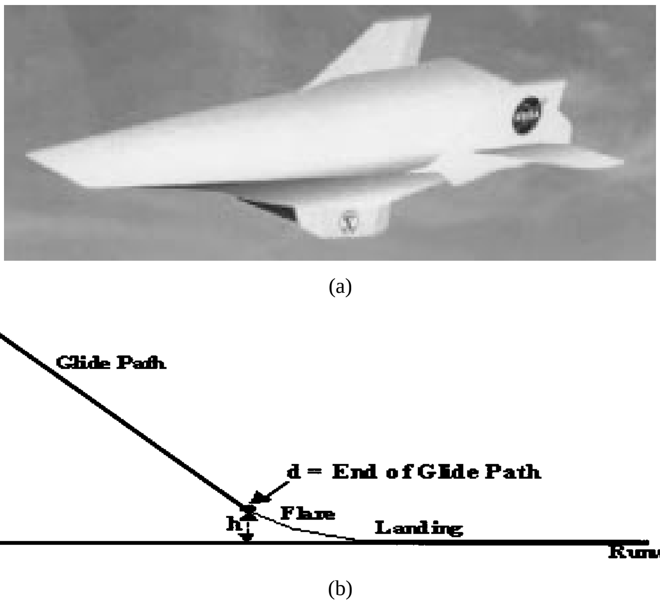


Fig. 1. (a) NASA X-43 (HyperX) and (b) its glide path.

As such, rather than directly evaluating the integral of (12), one can iteratively solve for P_{i+1} in terms of P_i by solving the Linear Lyapunov equation (13). Note, as an alternative to the above derivation one can obtain (13) by expressing $\frac{dV_{i+1}(x)}{dx}$ and K_{i+1} in the form $\frac{dV_{i+1}(x)}{dx} = 2x^T P_{i+1}$ 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_j; $j = 1, 2, \ldots n$; 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 $X_i = [x_1 x_2 \cdots x_n]$. 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 $F_i = \dot{X}_i X_i^{-1}$, yielding the alternative representation of the Linear Lyapunov equation

$$P_{i+1}\left[\dot{X}_i X_i^{-1}\right] + \left[\dot{X}_i X_i^{-1}\right]^T P_{i+1} = -\left[Q + P_i B R^{-1} B^T P_i\right] \tag{14}$$

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 precompensation technique of Appendix A.

As such, (14) can be used to implement the ADPA without *a-priori* knowledge of the plant, achieving one of the primary goals of soft control. Moreover, since $F_i = \dot{X}_i X_i^{-1}$ is asymptotically stable, (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 [1] providing the required hard convergence proof. Unlike the full nonlinear algorithm, the linear implementation of the ADPA 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 [13], [15].

As an alternative to the above Linear Lyapunov equation implementation, one can formulae an alternative implementation of the linear ADPA 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

$$\begin{aligned} V_{i+1}&(x_0) \\ &= \int_0^\infty \left[x_i^T(x_0, \lambda) Q x_i(x_0, \lambda) + u_i^T(x_0, \lambda) R u_i(x_0, \lambda)\right] d\lambda \\ &= x_0^T P_{i+1} x_0 \end{aligned} \tag{15}$$

since the plant and control law are time-invariant. More generally, for any initial state, $x_j = x_i(x_0, t_j)$, along this trajectory

$$\begin{aligned} V_{i+1}&(x_j) \\ &= \int_{t_j}^\infty \left[x_i^T(x_0, \lambda) Q x_i(x_0, \lambda) + u_i^T(x_0, \lambda) R u_i(x_0, \lambda)\right] d\lambda \\ &= x_0^T P_{i+1} x_0. \end{aligned} \tag{16}$$

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; $j = 1, 2, \ldots, q$; and solve the set of simultaneous equations

$$x_j^T P_{i+1} x_j = V_{i+1}(x_j); \qquad j = 1, 2, \ldots, q \tag{17}$$

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

TABLE I
STATES OF LINEARIZED 6 DoF X-43 MODEL

State	Symbol	Units	Trim Value	Initial Error
roll rate	p	rad/s	0.0	0.0850
yaw rate	r	rad/s	0.0	0.0850
pitch rate	q	rad/s	0.0	0.0850
roll	phi	rad	0.0	0.0850
yaw	psi	rad	-0.0778	0.0850
pitch	theta	rad	0.0	0.0850
vertical component of airspeed	w	ft/s (positive in down direction)	96.1442	-20.0000
horizontal/(forward) component of airspeed	u	ft/s	0.0	-20.0000
side component of airspeed	v	ft/s	30.6225	-20.0000
side tracking error (side deviation from desired glide-path)		ft		0.0850
altitude tracking error (vertical deviation from desired glide-path)		ft		0.0850

TABLE II
INPUTS (SYMBOL, UNITS) OF LINEARIZED 6 DoF X-43 MODEL

control	symbol	units	trim value	control	symbol	units	trim value
elevator deflection	de	deg	-15.86	aileron deflection	da	deg	0.0
rudder deflection	dr	deg	0.0	thrust	dt	pounds	-10.728

top of one another, one may transform (17) into a $q \times n^2$ matrix equation

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

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

$$\left(\begin{bmatrix} x_1^T \otimes x_1^T \\ x_2^T \otimes x_2^T \\ \vdots \\ x_q^T \otimes x_q^T \end{bmatrix} S\right) \operatorname{vec}^+(P_{i+1}) = \begin{bmatrix} V_{i+1}(x_1) \\ V_{i+1}(x_2) \\ \vdots \\ V_{i+1}(x_q) \end{bmatrix}. \tag{19}$$

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

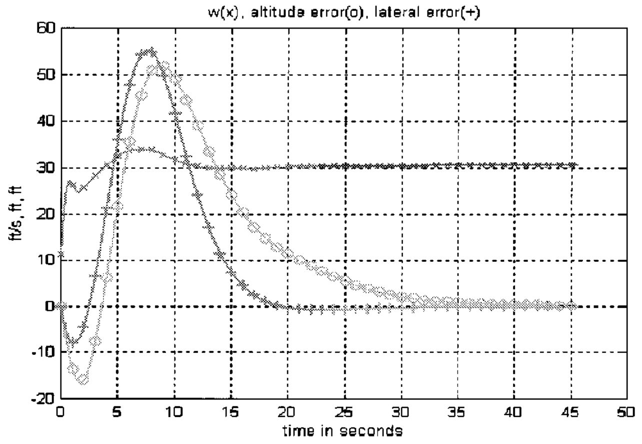


Fig. 2. X-43 autolander altitude error, lateral error, and sink rate.

Although both (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 (19), and since the $V_{i+1}(x_j)$ are computed by integrating along the entire state trajectory, measurement noise is filtered. On the other hand, the Linear Lyapunov formulation of (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

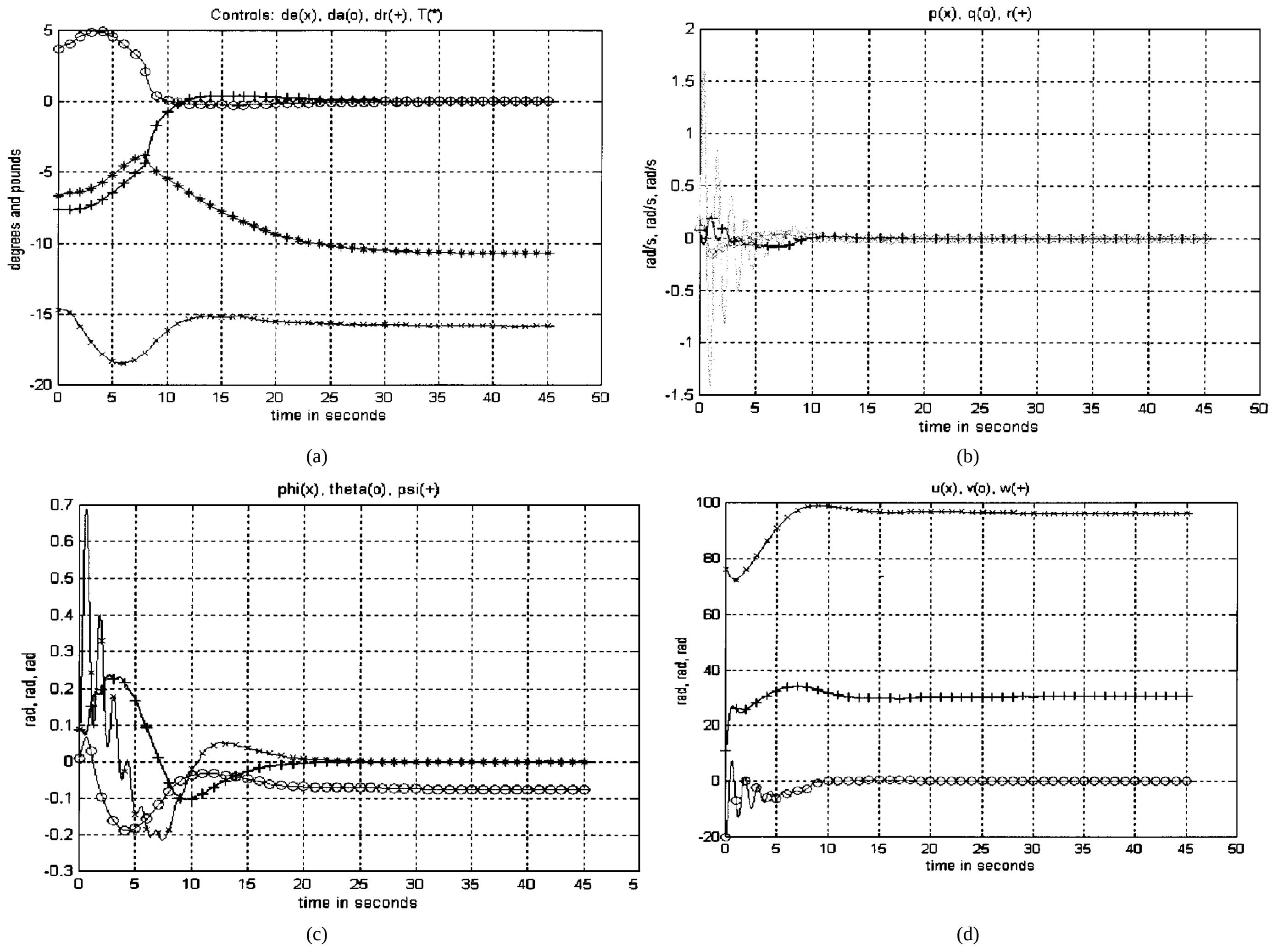


Fig. 3. (a) Aircraft controls (de, da, dr, and T); (b) Orientation rates (p, q, and r); (c) Orientation angles (phi, theta, and psi); and (d) Airspeed components (u, v, and w).

trajectory. In both implementations one must assume that the x_js 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 ADPA in the *linear case*, we developed an autolander for the NASA X-43A-LS (or HyperX) [27]. The X-43A, shown in Fig. 1(a), is an unmanned experimental aircraft for an advanced scramjet engine, operating in the Mach 7–10 range [14]. In its present configuration, the X-43A 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 below was to evaluate the feasibility of landing the planned X-43B. To this end, we designed, fabricated, and are in the process of flight testing the X-43A-LS; a full size subsonic version of the X43A with increased wingspan, designed to evaluate the low speed performance, landing, and takeoff characteristics of the X-43 design. The 12′ long X-43A-LS is powered by a 130 lb thrust AMT Phoenix turbojet engine, and is designed to fly at 250 kts [11].

The initial flight test of the X-43A-LS was performed in the Fall of 2001, while we are presently preparing for a flight test program which will include flight testing the X-43A-LS with two different adaptive flight control systems; one based on the ADPA described in the present paper and the other based on a Lyapunov Synthesis algorithm [21]. As a first step in this process we developed an autolander for the X-43A-LS based on the linear version of the ADPA [6]. That is, a special purpose flight control system designed to track a "glide path" from low altitude to the "flare" just above the end of the runway, as indicated in Fig. 1(b). The simulated performance of the X-43A-LS autolander, using a 6 degree-of-freedom *linearized* model of the X-43A-LS, is described as follows.

This linearized X-43A-LS model has eleven states (listed in Table I) and four inputs (listed in Table II). 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 *apriori* knowledge of either the A or B matrices for the X-43A-LS model was provided to the controller.

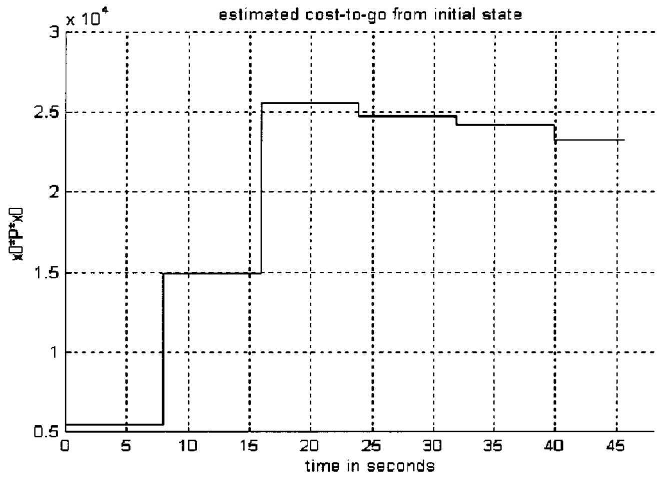


Fig. 4. Cost-to-go from initial state as a function of time.

A "trim routine" was 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 was calculated to put the aircraft on the specified glide slope. The performance of the X-43A-LS autolander is summarized in Fig. 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 Fig. 3(a), all of which are well within the dynamic range of the X-43A-LS's controls, while the remaining states of the aircraft during landing are shown in Fig. 3(b)–(d).

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 Fig. 4. As expected, the cost-to-go jumps from the low initial value associated with the initial guess, P_0, 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 in our implementation, which spreads the initial cost-to-go jump over several iterations.

IV. Quadratic Approximation of the Cost Functional

The purpose of this section is to develop an approximate implementation of the ADPA in which the actual cost functional is approximated by a quadratic at each point in state space. As above, the quadratic approximation preserves the fused soft computing/hard computing character of the general ADPA, combining soft computing techniques to iteratively solve for an approximate cost functional in real time for a plant with unknown dynamics, with hard computing techniques to guarantee convergence of the algorithm *and* stepwise stability of the controller.

To this end, we let $a(x), b(x), q(x)$, and $r(x)$ be C^∞ functions as defined in Section II, and we let $V_i(x) = x^T P_i x$ in which case $(dV_i/dx)(x) = 2x^T P_i$ and $k_i(x) = -r^{-1}(x)b^T(x)P_{i+1}x$ are also C^∞ functions. Substituting these expression into the Iterative HJB equation we obtain

$$x^T P_{i+1}\dot{x} = -q(x) - x^T P_i b(x) r^{-1}(x) b^T(x) P_i x. \qquad (20)$$

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

$$x_j^T P_{i+1}\dot{x}_j = -q(x_j) - x_j^T P_i b(x_j) r^{-1}(x_j) b^T(x_j) P_i x_j; \quad j = 1, 2 \ldots, q \qquad (21)$$

or, equivalently in matrix form

$$\begin{bmatrix} \dot{x}_1^T \otimes x_1^T \\ \dot{x}_2^T \otimes x_2^T \\ \vdots \\ \dot{x}_q^T \otimes x_q^T \end{bmatrix} \mathrm{vec}(P_{i+1}) = \begin{bmatrix} -q(x_1) - x_1^T P_i b(x_1) r^{-1}(x_1) b^T(x_1) P_i x_1 \\ -q(x_2) - x_2^T P_i b(x_2) r^{-1}(x_2) b^T(x_2) P_i x_2 \\ \vdots \\ -q(x_q) - x_2^T P_i b(x_q) r^{-1}(x_q) b^T(x_q) P_i x_q \end{bmatrix}. \qquad (22)$$

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 (19) is known to exist, in the nonlinear case one cannot guarantee that a hermitian solution to (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, $P_{i+1} = U_{i+1}^T U_{i+1}$, forcing P_{i+1} to be positive definite hermitian. Substituting this expression into (22) we then solve the quadratic equation

$$\begin{bmatrix} \dot{x}_1^T \otimes x_1^T \\ \dot{x}_2^T \otimes x_2^T \\ \vdots \\ \dot{x}_q^T \otimes x_q^T \end{bmatrix} \mathrm{vec}(U_{i+1}^T U_{i+1}) = \begin{bmatrix} -q(x_1) - x_1^T P_i b(x_1) r^{-1}(x_1) b^T(x_1) P_i x_1 \\ -q(x_2) - x_2^T P_i b(x_2) r^{-1}(x_2) b^T(x_2) P_i x_2 \\ \vdots \\ -q(x_q) - x_q^T P_i b(x_q) r^{-1}(x_q) b^T(x_q) P_i x_q \end{bmatrix} \qquad (23)$$

for U_{i+1}, yielding an approximation of the actual cost functional in the form $x^T U_{i+1}^T U_{i+1} x$.

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, $x_i(x_0, \cdot)$, 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

$$\begin{aligned} V_{i+1}(x_j) &= \int_{t_j}^{\infty} \left[q(x_i(x_0, \lambda)) + x_i^T(x_0, \lambda) P_i b(x_i(x_0, \lambda)) r^{-1} \right. \\ &\quad \left. \cdot\, (x_i(x_0, \lambda)) b^T(x_i(x_0, \lambda)) P_i x_i(x_0, \lambda) \right] d\lambda \\ &= x_j^T P_{i+1} x_j; \qquad j = 1, 2, \ldots, q \end{aligned} \qquad (24)$$

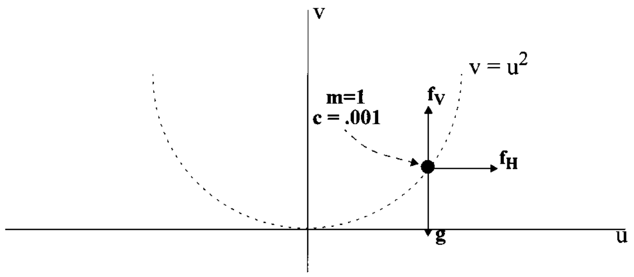


Fig. 5. Mass constrained to a parabolic track.

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

$$\begin{bmatrix} x_1^T \otimes x_1^T \\ x_2^T \otimes x_2^T \\ \vdots \\ x_q^T \otimes x_q^T \end{bmatrix} \text{vec}(P_{i+1}) = \begin{bmatrix} V_{i+1}(x_1) \\ V_{i+1}(x_2) \\ \vdots \\ V_{i+1}(x_q) \end{bmatrix} \tag{25}$$

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

$$\begin{bmatrix} x_1^T \otimes x_1^T \\ x_2^T \otimes x_2^T \\ \vdots \\ x_q^T \otimes x_q^T \end{bmatrix} \text{vec}(U_{i+1}^T U_{i+1}) = \begin{bmatrix} V_{i+1}(x_1) \\ V_{i+1}(x_2) \\ \vdots \\ V_{i+1}(x_q) \end{bmatrix} \tag{26}$$

which may be solved for U_{i+1}, yielding an approximation of the actual cost functional in the form $x^T U_{i+1}^T U_{i+1} x$.

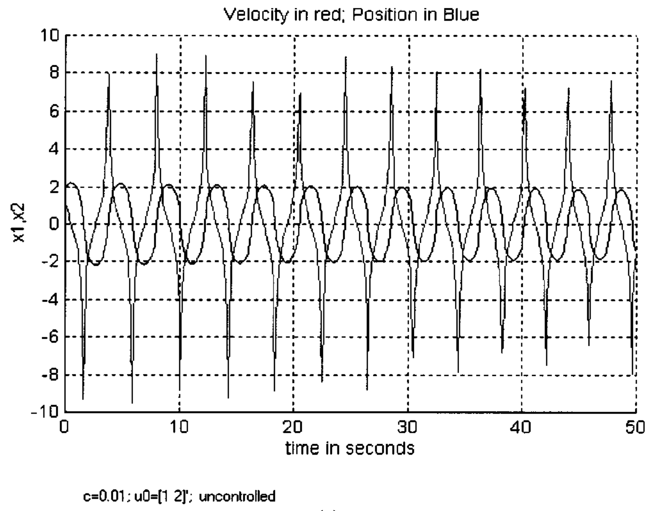
Although both (22) and (26) require that one solve a quadratic equation in $q = n(n+1)/2$ unknowns, the derivatives of the state are not required by the formulation of (26) and, as in the linear case, the $V_{i+1}(x_j)$ are filtered by integrating along the entire state trajectory, $x_i(x_0, \cdot)$, from x_0 to the singularity at (0, 0). On the other hand, the formulation of (22) allows one to adapt the control multiple times along a given state trajectory. In both implementations one must assume that the x_js 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 ADPA of (22), we selected the system illustrated in Fig. 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 2nd 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

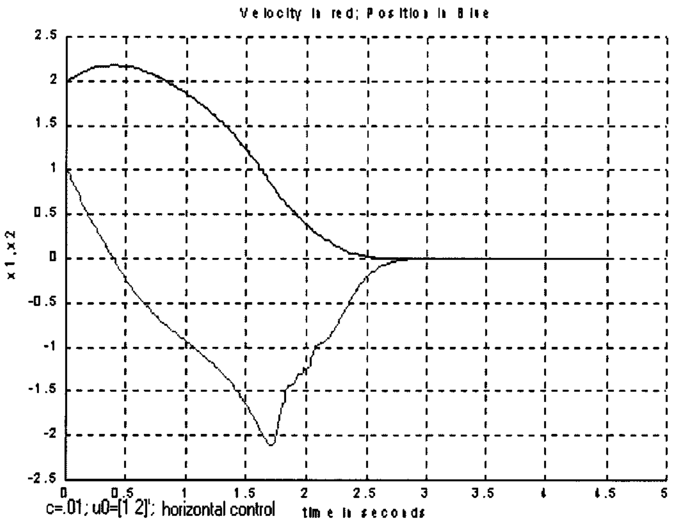
$$\dot{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}} = \begin{bmatrix} \dfrac{-4x_1^2 x_2 - 2g x_2}{1 + 4x_2^2} & \dfrac{-c x_1}{m} \\ x_1 & 0 \end{bmatrix} + \begin{bmatrix} \dfrac{2x_2}{m(1 + 4x_2^2)} & \dfrac{1}{m(1 + 4x_2^2)} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} f_V \\ f_H \end{bmatrix}. \tag{27}$$

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

$$E = \frac{m}{2}\left(1 + 4x_2^2\right) x_1^2 + m g x_2^2 \tag{28}$$



(a)



(b)

Fig. 6. (a) Uncontrolled and (b) controlled response of parabolically constrained mass.

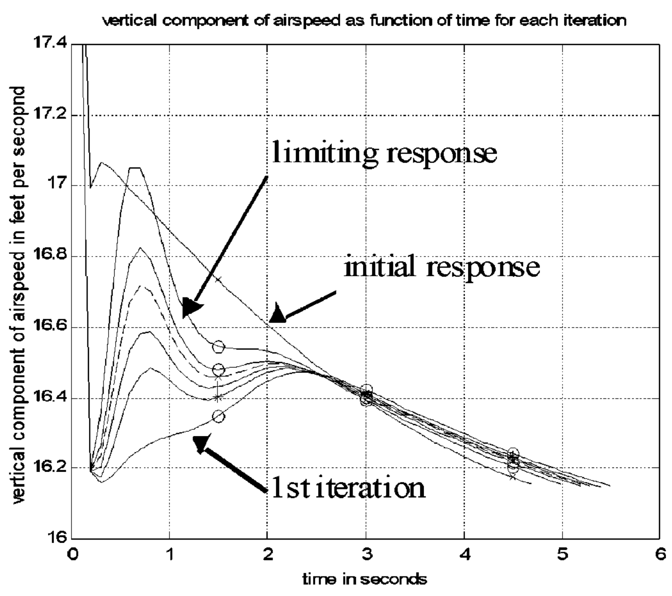
Fig. 7. LoFLYTE UAV at Edwards AFB.

while the derivative of E along the trajectories of the system takes the form

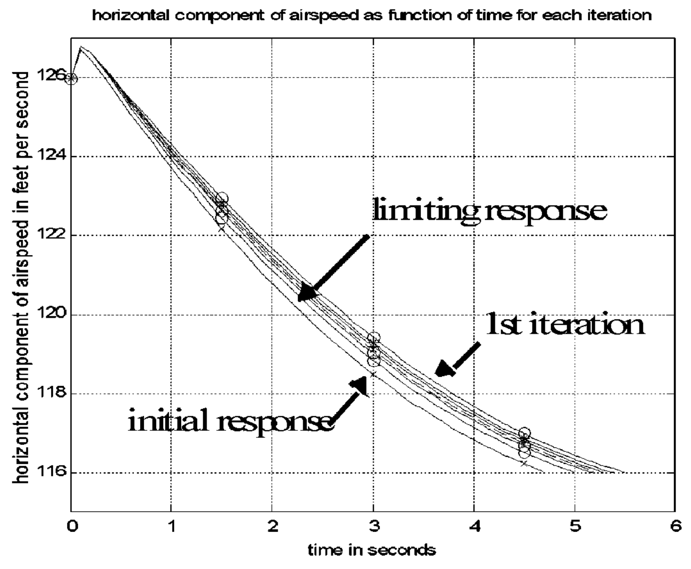
$$\dot{E} = -c\left(1 + 4x_2^2\right) x_1^2. \tag{29}$$

TABLE III
STATES OF NONLINEAR LONGITUDINAL LoFLYTE MODEL

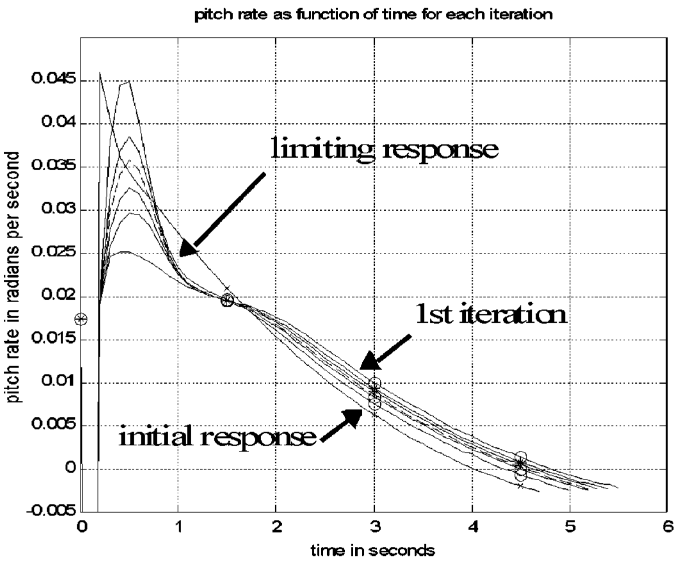
State	Symbol	Units	Min	Trim Point	Max
pitch rate	**q**	**rad/s**	**-0.3491**	**0.0000**	**0.3491**
pitch	**theta**	**rad**	**-0.6850**	**-0.3359**	**0.0132**
vertical component of airspeed	**w**	**ft/s (positive in down direction)**	**-4.116**	**16.12**	**36.12**
horizontal/(forward) component of airspeed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



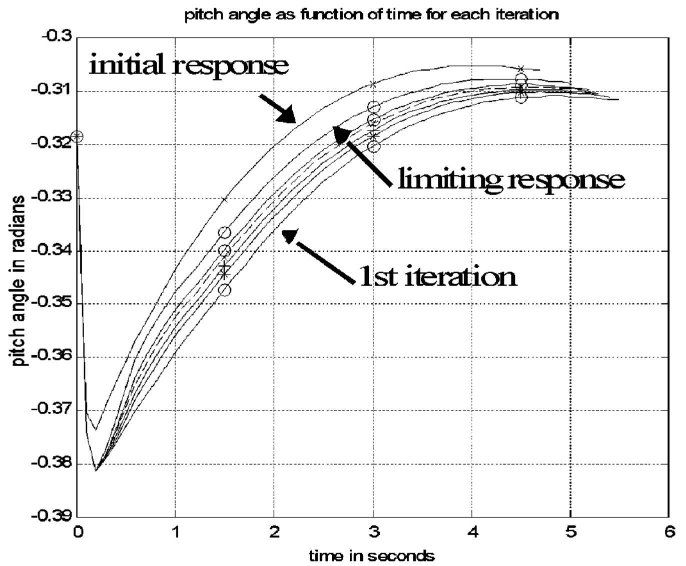
(a)



(b)



(c)



(d)

Fig. 8. Aircraft state variables on the 0th, first, second, third, fourth, fifth, and sixtieth iteration: (a) vertical velocity; (b) horizontal velocity; (c) pitch rate; and (d) pitch angle.

To evaluate the performance of the ADPA without *apriori* knowledge of either $a(x)$ or $b(x)$, a 1st order precompensator 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 Fig. 6(a) while the response of the controlled system is shown in

Fig. 6(b). Here, the controlled response converges to the singularity at (0, 0) in less than 3 s with a reasonably smooth response, while the minimally damped uncontrolled system oscillates for several minutes before settling down.

V. Radial Basis Approximation of the Cost Functional

Unlike the nonlinear implementation of the ADPA of Section IV, 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. As above the radial basis approximation preserves the fused soft computing/hard computing character of the general ADPA, combining soft computing and radial basis function techniques to iteratively solve for an approximate cost functional in real time for a plant with unknown dynamics, with hard computing techniques used to guarantee convergence of the algorithm *and* stepwise stability of the controller.

To demonstrate the radial basis function implementation of the ADPA we chose a fourth-order longitudinal model of the LoFLYTE[1] aircraft [10], illustrated in Fig. 7, with a nonlinear pitching moment coefficient. LoFLYTE is an unmanned subsonic testbed for a Mach 5 waverider which was built to evaluate the low speed, landing, and takeoff performance of the waverider design. It is eight feet long, powered by a 40–lb thrust AMT Olympus turbojet, and designed to fly at 150 knots. LoFLYTE was extensively flight tested in the late 1990s. An upgraded version of LoFLYTE, which is presently being prepared for flight testing, will be used as an adaptive flight control testbed; for the ADPA, a Lyapunov Synthesis algorithm [21], [22] and system ID based algorithm developed at NASA.

The states of the nonlinear longitudinal LoFLYTE model are indicated in Table III, with the zero point in the state space shifted to correspond to a 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 $-2.784°$.

For our radial basis function implementation of the ADPA, 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 III. 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 $5\times\ 5\times\ 5\times\ 5 = 625$ block of radial basis functions in a neighborhood of x, corresponding to a 4-cube in state space centered at x with $\Delta u = \pm 4.76$ ft/s, $\Delta w = \pm 4.76$ ft/s $\Delta q = \pm 0.083$ rad/s, and $\Delta\theta = \pm 0.083$ rad.

[1]LoFLYTE is a registered trademark of Accurate Automation Corporation, Chattanooga, TN 37421 USA.

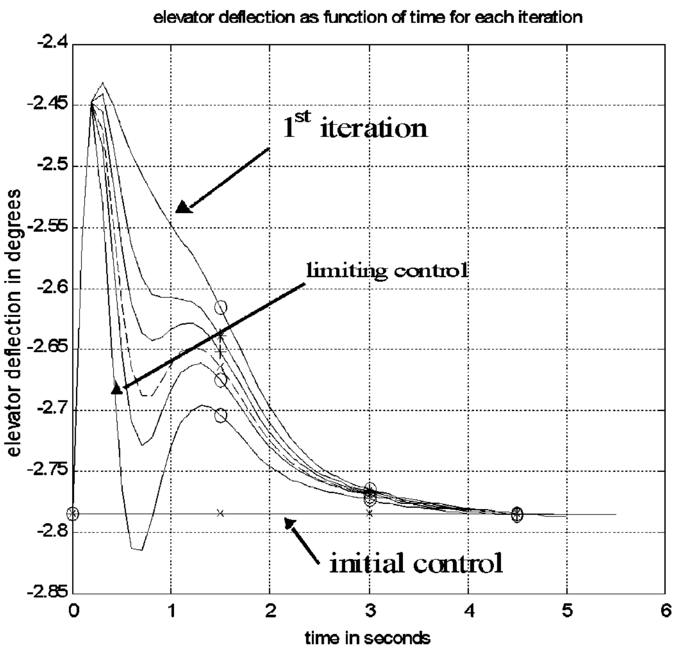


Fig. 9. Elevator deflection on the 0th, first, second, third, fourth, and fifth and sixtieth iteration.

Figs. 8–11 illustrate the performance of the radial basis function implementation of the ADPA, learning an "optimal" control strategy from a given initial point in the state space, using the quadratic performance measure $J = \int_0^\infty [x^T Qx + u^T Ru] d\lambda$ with $Q = \mathrm{diag}[0.0015, 0.0015, 0.0015, 0.0015]$ and $R = [0.005]$. 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 incremented, 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/s while the elevator deflection angle was updated ten times/s. The performance of the radial basis function implementation of the ADPA is illustrated in Figs. 8–11, where we have plotted each of the key system variables on the 0th, first, second, third, fourth, and fifth iterations of the algorithm and the limiting value of these plots (at the sixtieth iteration).

The state variables of the aircraft are plotted in Fig. 8. For each state variable the initial (0th) response (indicated by "x"s) is at one extreme (high for the vertical velocity, pitch, and pitch rate; and low for the horizontal velocity), with the response jumping to the opposite extreme on the 1st iteration (indicated by "o"s) and then converging toward the limiting value, with the adaption process effectively convergent after 10 iterations. The elevator deflection required to achieve these responses is show in Fig. 9. Since $k_0(x) = 0$ the initial (0th) elevator deflection remains constant at the trim point of $-2.784°$ (indicated by "x"s). The elevator deflection then jumps to a high value on the first 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 Fig. 8(a)], due to the use of a "null" controller on the first iteration [which would not be the case for the ac-

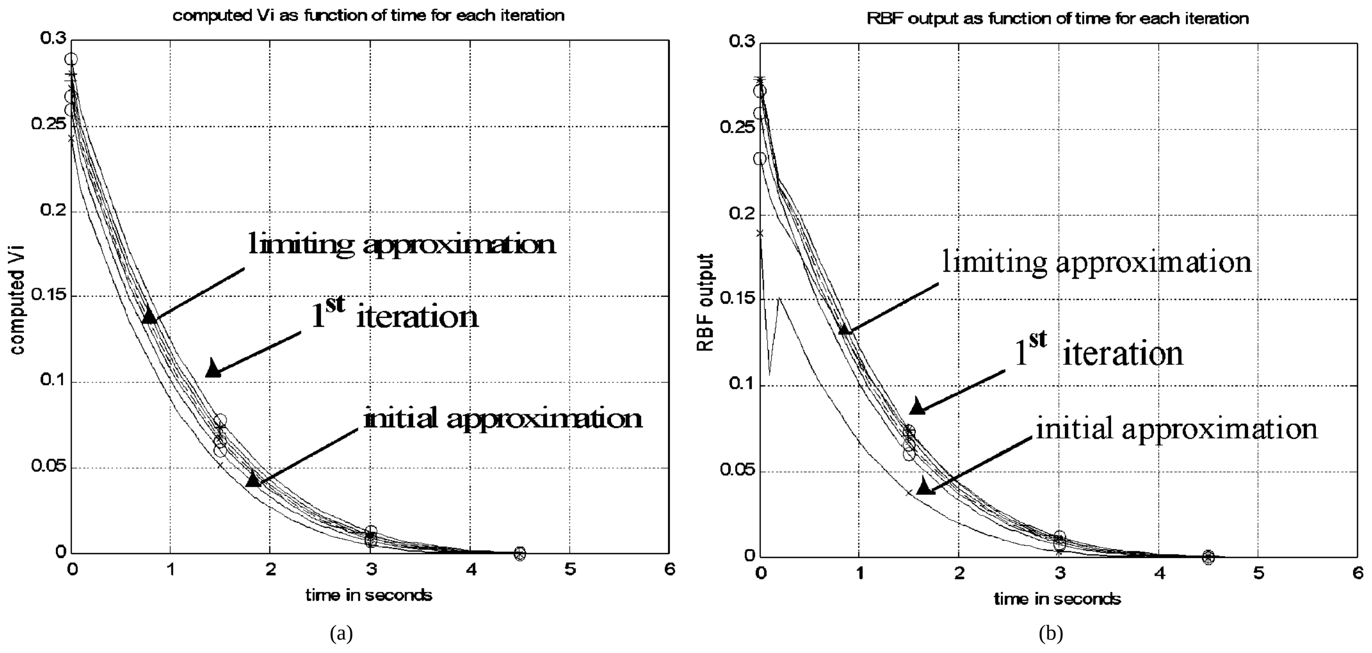


Fig. 10. (a) Computed and (b) RBF approximation of the optimal cost functional on the 0th, first, second, third, fourth, and fifth and sixtieth iteration.

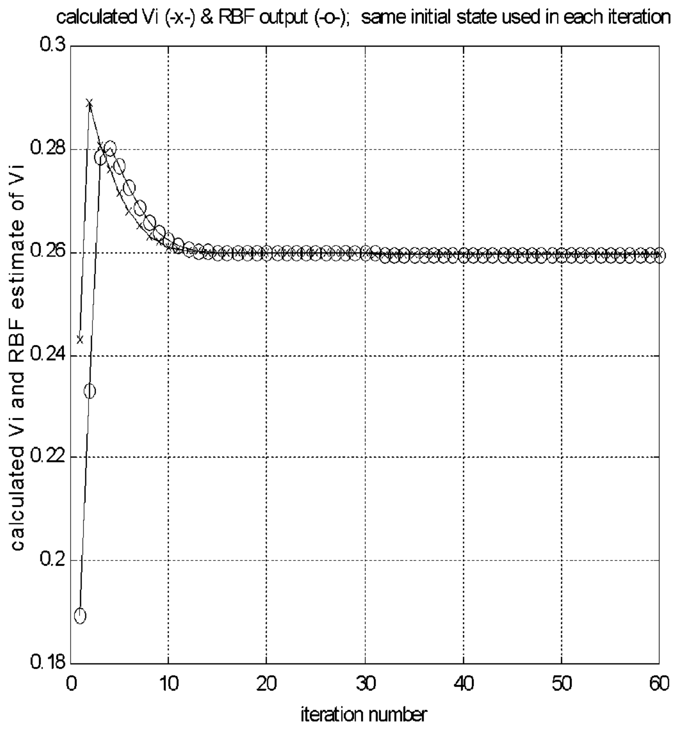


Fig. 11. Cost-to-go based on the computed ("x"s) and RBF approximation ("o"s) of the optimal cost functional versus iteration number.

tual aircraft where $k_0(x)$ would be selected on the basis of prior simulation].

The performance of the ADPA is illustrated in Fig. 10 where the computed [Fig. 10(a)] and radial basis function approximation [Fig. 10(b)] of the optimal cost functional are plotted as a function of time along the state trajectory, on the 0th, first, second, third, fourth, and fifth and sixtieth (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 Fig. 11. As predicted by the theory, the cost-to-go has an initial spike and then declines monotonically to the limiting value.

VI. Conclusions

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 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. Our goal in the preceding has been to provide the framework for a family of ADPAs, fusing hard and soft computing techniques, by developing a general theory and the three implementations of Sections III–V.

Indeed, several alternative implementations are possible. 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 $dV_i(x)/dx$, 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 V one might replace the radial basis function approximation, which produces a "bumpy" Tchebychev-like approximation of $V_i(x)$, with a "smoother" neural network approximation, or an alternative local approximator (cf., a cubic spline). Finally, by requiring the plant and performance measure matrices to be "real analytic" rather than C^∞ (and extending the proof of the theorem to guarantee that the matrices generated by the

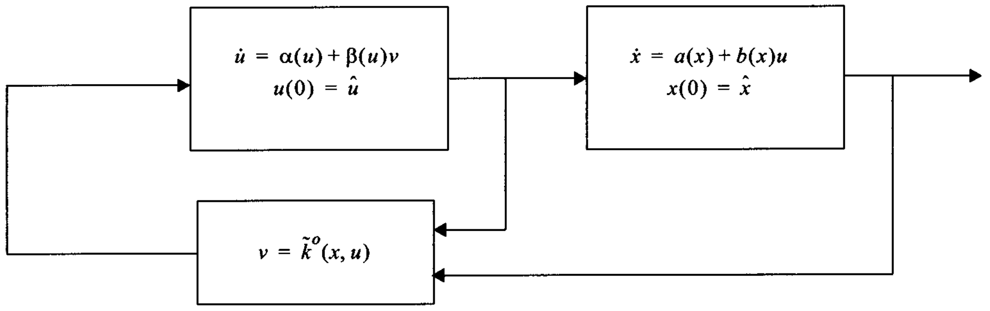

$\dot{u} = \alpha(u) + \beta(u)\nu$
$u(0) = \hat{u}$
$\dot{x} = a(x) + b(x)u$
$x(0) = \hat{x}$
$\nu = \tilde{k}^o(x, u)$

Fig. 12. Control system with precompensator.

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.

Appendix A
Precompensation Procedure

The purpose of this appendix is to derive the precompensation technique originally described in [23], which embeds the $b(x)$ matrix of an input affine plant into the $a(x)$ matrix of the combined precompensator/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 Fig. 12, where the precompensator is defined by any desired (controllable) input affine differential equation, $\dot{u} = \alpha(u) + \beta(u)\nu$, whose state vector is of the same dimension as the input vector for the given plant, with a singularity at $(u = 0, \nu = 0)$.

Now, the dynamics of the augmented plant, obtained by combining the precompensator with the original plant, take the form

$$\dot{\tilde{x}} = \begin{bmatrix} x \\ u \end{bmatrix}^{\cdot} = \begin{bmatrix} a(x) + b(x)u \\ \alpha(u) + \beta(u)\nu \end{bmatrix} = \begin{bmatrix} a(x) + b(x)u \\ \alpha(u) \end{bmatrix} + \begin{bmatrix} 0 \\ \beta(u) \end{bmatrix}\nu,$$
$$= \tilde{a}(x, u) + \tilde{b}(x, u)\nu = \tilde{a}(\tilde{x}) + \tilde{b}(\tilde{x})\nu \qquad (30)$$

which is also input affine, with the augmented state vector, $\tilde{x} = [x \quad u]^T$ and a singularity at $(\tilde{x} = 0, \nu = 0)$. 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

$$\tilde{J} = \int_0^\infty \left[\tilde{l}(\tilde{x}(x_0, \lambda), \nu(x_0, \lambda))\right] d\lambda$$
$$= \int_0^\infty [l(x(x_0, \lambda), u(x_0, \lambda)) + l_\nu(\nu(x_0, \lambda))]\, d\lambda$$
$$= J + \int_{t_0}^\infty [l_\nu(\nu(x_0, \lambda))]\, d\lambda \qquad (31)$$

where $l_\nu(\nu(x_0, t)) \geq 0$ with equality if and only if $\nu = 0$.

As such, one can apply the above described ADPA to a plant in which both $a(x)$ and $b(x)$ are unknown by applying the algorithm to the augmented system of (30) with the augmented performance measure of (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.

References

[1] S. Barnett, *The Matrices of Control Theory*. New York: Van Nostrand Reinhold, 1971.

[2] A. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC–13, no. 5, pp. 834–846, 1983.

[3] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[4] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[5] C. J. Cox, M. Lothers, R. Pap, and C. Thomas, "A neurocontroller for robotics applications," in *Proc. Syst., Man, Cybernetics Conf.*, Chicago, IL, 1992, pp. 712–716.

[6] C. J. Cox, S. W. Stepniewski, C. C. Jorgensen, and R. Saeks, "On the design of a neural network autolander," *Int. J. Robust Nonlinear Contr.*, vol. 9, pp. 1071–1096, 1999.

[7] A. Devinatz and J. L. Kaplan, "Asymptotic estimates for solutions of linear systems of ordinary differential equations having multiple characteristics roots," *Indiana Univ. Math J.*, vol. 22, p. 335, 1972.

[8] J. Dieudonne, *Foundations of Mathematical Analysis*. New York: Academic, 1960.

[9] S. S. Ge, C. C. Hang, and T. Zhang, "Adaptive neural network control of nonlinear systems by stable output feedback," *IEEE Trans. Syst., Man, Cybern. B*, pp. 818–828, Dec. 1999.

[10] C. S. Gibson, J. K. Buckner, L. A. Carlton, C. J. Cox, M. P. Kocher, and C. E. Lewis, "The LoFLYTE Program," in *Proc. AIAA 9th Int. Space Planes and Hypersonic Systems and Technologies Conf.*, 1999.

[11] C. S. Gibson, J. C. Neidhoefer, S. M. Cooper, and L. A. Carlton, "Development and flight test of the X-43A-LS hypersonic configuration UAV," in *Proc. 1st AIAA Unmanned Aerospace Vehicles, Systems, Technologies, and Operations Conf. and Workshop*, 2002.

[12] A. Halanay and V. Rasvan, *Applications of Lyapunov Methods in Stability*. Dordrecht, The Netherlands: Kluwer, 1993.

[13] W. E. Holley and S. Y. Wei, "An improvement in the MacFarlane-Potter method for solving the algebraic Riccati equation," in *Proc. Joint Auto. Cont. Conf.*, 1979, pp. 921–923.

[14] "Jour. of Spacecraft and Rockets, Special Section on Hyper-X,", (eight papers), vol. 38, 2001.

[15] H. Kwakernaak and R. Sivan, *Linear Optimal Control Systems*. New York: Wiley, 1972.

[16] G. G. Lendaris, L. Schultz, and T. Shannon, "Adaptive critic design for intelligent steering and speed control of a 2-axle vehicle," in *Proc. Int. Joint Conf. on Neural Networks*, Como, 2000, Paper 28-03.

[17] G. Lingari and M. Tomizuko, "Stability of fuzzy linguistic control systems," in *Proc. IEEE Decision and Control*, 1990, pp. 2185–2190.

[18] G. P. Liu, V. Kadkivkamanthan, and S. A. Billings, "Variable neural network for adaptive control of nonlinear systems," *IEEE Trans. Syst., Man, Cybern. C*, vol. 29, pp. 34–43, Feb. 1999.

[19] D. G. Luenberger, *Introduction to Dynamic Systems: Theory, Models, and Applications*. New York: Wiley, 1979.

[20] D. Prokhorov and L. Feldkamp, "Primitive adaptive critics," in *Proc. 1997 Int. Conf. Neural Networks*, vol. 4, 1997, pp. 2263–2267.

[21] R. E. Saeks and C. J. Cox, "LoFLYTE: A neurocontrols testbed," in *35th AIAA Aerospace Sciences Meeting*, 1997, AIAA Paper 97-0085.

[22] R. E. Saeks, C. J. Cox, J. C. Neidhoefer, and G. G. Lendaris, “Neural adaptive control of LoFLYTE,” in *Proc. Amer. Control Conf.*, 2001, pp. 2913–2917.

[23] R. E. Saeks and C. J. Cox, “Adaptive critic control and functional link networks,” in *Proc. 1998 IEEE Conf. Systems, Man, Cybernetics*, 1998, pp. 1652–1657.

[24] R. E. Saeks and J. Murray, “Proof of the adaptive dynamic programming theorem,”, Chattanooga, TN, Internal Tech. Rep., Accurate Automation Corp., 2001.

[25] R. M. Sanner and J. E. Slotine, “Gaussian networks for direct adaptive control,” in *Proc. Amer. Control Conf.*, 1991, pp. 2153–2159.

[26] S. N. Singh, W. Yim, and W. R. Wells, “Direct adaptive and neural control of wing-rock motion of slender delta wings,” *J. Guid., Contr. Dynam.*, vol. 18, pp. 25–30, 1995.

[27] J. Sitz, “HYPER-X: Hypersonic experimental research vehicle,”, NASA Fact Sheet, FS-1994-11-030, 1998.

[28] L.-X. Wang, “Stable adaptive fuzzy control with applications to inverted pendulum tracking,” *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 146–155, 1993.

[29] ——, “Stable adaptive fuzzy control of nonlinear systems,” *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, pp. 677–691, Oct. 1996.

[30] P. J. Werbos, “Approximate dynamic programming for real time control and neural modeling,” in *Handbook of Intelligent Control*, P. J. White and P. J. Sofge, Eds. New York: Van Nostrand, 1994, pp. 493–525.

[31] J. C. Wu and T. S. Liu, “Fuzzy control stabilization with applications to motorcycle control,” *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, pp. 836–847, Dec. 1996.

[32] R. Zaman, D. Prokhorov, and D. Wunsch, “Adaptive critic design in learning to play game of go,” in *Proc. 1997 Int. Conf. Neural Networks*, vol. I, 1997, pp. 1–4.

John J. Murray (S’78–SM’88) received the Ph.D. degree from the University of Notre Dame, Notre Dame, IN.

He is currently an Associate Professor, Electrical Engineering Department, State University of New York at Stony Brook. He was previously on the faculty at Texas Tech University, Lubbock, and has taught courses at the undergraduate and graduate levels in digital signal processing, circuit theory, electronics, and control systems.

Chadwick J. Cox received the B.S. degree in mathematics and computer science from Jacksonville State University, Jacksonville, AL, in 1990.

He then joined Accurate Automation Corporation and has since worked on a wide variety of projects, some involving trajectory optimization, fault diagnosis, robotic manipulation, aircraft engine control, neural networks, flight control, aircraft safety, robotic surface finishing, and ground vehicle control. He is currently with Accurate Automation, Chattanooga, TN. His primary interests are nonlinear control, adaptive control, learning control, autonomous systems, and image processing.

George G. Lendaris (M’58–SM’74–F’83–LF’97) received the B.S., M.S., and Ph.D. degrees from the University of California, Berkeley.

He then joined the GM Defense Research Laboratories, where he did extensive work in control systems, neural networks, and pattern recognition. In 1969, he went to academia, first joining Oregon Graduate Institute, where he was Chair of the Faculty, and, two years later, moved to Portland State University (PSU), Portland, OR, where he became one of the founders and developers of their Systems Science Ph.D. Program, where he has served for the past 30–plus years. In this context, he expanded his academic and research activities into general system theory and practice, and, more recently, to computational intelligence. He has served a number of capacities at PSU, including Director of the SySc Ph.D. Program, and President of the Faculty Senate. He is currently Director, Systems Science Ph.D. Program and the NW Computational Intelligence Laboratory. He has been active in neural network research since the early 1960s, and, in particular, the past 15 years. During recent years, his research has focused on application of adaptive critic and dynamic programming methods to control system design.

Dr. Lendaris developed the optical Diffraction Pattern Sampling methods of pattern recognition and was declared “Father of Diffraction Pattern Sampling” by the SPIE in 1977, and was elevated to Fellow of the IEEE in 1982 in recognition of this seminal work. He is past Vice President of the SMC Society, and, more recently, on its AdCom. He has been active in the neural network professional arena, serving a number of capacities, such as General Chair of the 1993 IJCNN, up to his just-completed term as President of the International Neural Network Society.

Richard Saeks (M’65–SM’74–F’77) received the B.S. degree from Northwestern University, Evanston, IL, the M.S. degree from Colorado State University, Fort Collins, and the Ph.D. degree from Cornell University, Ithaca, NY, all in electrical engineering.

He is Chief Technical Officer of the Accurate Automation Corporation, Chattanooga, TN, where he is responsible for the activities of a team of electrical and aeronautical engineers doing research in advanced aeronautics and adaptive systems. Ongoing programs include the development and flight test of Accurate Automation’s LoFLYTE, HYFlyte, X-43A-LS, and GLOV demonstrator aircraft; and research and development in weakly ionized plasma shock wave modification, adaptive flight control, and neural network based fault diagnosis. He has led Accurate Automation’s research programs in adaptive control and plasma shock wave modification, he is the Principal Investigator for the NASA/Air Force LoFLYTE program and R&D programs in neurocontrol and hypersonics, and the architect of Accurate Automation’s MIMD parallel Neural Network Processor and its special purpose programming language. Prior to joining Accurate Automation, he was Dean of Engineering, Illinois Institute of Technology, Chicago, Chairman of the Electrical Engineering Department at Arizona State University, Tempe, and a faculty member with joint appointments in electrical engineering, computer science, and mathematics at Texas Tech University (TTU), Lubbock. He held the Paul Whitfield Horn Professorship at TTU, where his research activities spanned the areas of mathematical systems theory control, large-scale systems, and fault diagnosis.

Dr. Saeks is Past-President of the IEEE Systems, Man, and Cybernetics Society and an Associate Fellow of the AIAA.