11

REINFORCEMENT LEARNING AND OPTIMAL ADAPTIVE CONTROL

In this book we have presented a variety of methods for the analysis and design of optimal control systems. Design has generally been based on solving matrix design equations assuming full knowledge of the system dynamics. Optimal control is fundamentally a backward-in-time problem, as we have seen especially clearly in Chapter 6 Dynamic Programming. Optimal controllers are normally designed offline by solving Hamilton-Jacobi-Bellman (HJB) equations, for example, the Riccati equation, using complete knowledge of the system dynamics. The controller is stored and then implemented online in real time. In the linear quadratic case, this means the feedback gains are stored. Determining optimal control policies for nonlinear systems requires the offline solution of nonlinear HJB equations, which are often difficult or impossible to solve analytically.

In this book, we have developed many matrix design equations whose solutions yield various sorts of optimal controllers. This includes the Riccati equation, the HJB equation, and the design equations in Chapter 10 for differential games. These equations are normally solved offline. In this chapter we give practical methods for solving these equations online in real time using data measured along the system trajectories. Some of the methods do not require knowledge of the system dynamics.

In practical applications, it is often important to be able to design controllers online in real time without having complete knowledge of the plant dynamics. Modeling uncertainties may exist, including inaccurate parameters, unmodeled high-frequency dynamics, and disturbances. Moreover, both the system dynamics and the performance objectives may change with time. A class of controllers known as adaptive controllers learn online to control unknown systems using data measured in real time along the system trajectories. While learning the control solutions, adaptive controllers are able to guarantee stability and system performance. Adaptive control and optimal control represent different philosophies for designing feedback controllers. Adaptive controllers are not usually designed to be optimal in the sense of minimizing user-prescribed performance functions. Indirect adaptive controllers use system identification techniques to identify the system parameters, then use the obtained model to solve optimal design equations (Ioannou and Fidan 2006). Adaptive controllers may satisfy certain inverse optimality conditions, as shown in Li and Krstic (1997).

In this chapter we show how to design optimal controllers online in real time using data measured along the system trajectories. We present several adaptive control algorithms that converge to optimal control solutions. Several of these algorithms do not require full knowledge of the plant dynamics. In the LQR case, for instance, this amounts to using adaptive control techniques to learn the solution of the algebraic Riccati equation online without knowing the plant matrix A. In the nonlinear case, these algorithms allow the approximate solution of complicated HJ equations that cannot be exactly solved using analytic means. Design of optimal controllers online allows the performance objectives, such as the LQR weighting matrices, to change slowly in real time as control objectives change.

The framework we use in this chapter is the theory of Markov decision processes (MDP). It is shown here that MDP provide a natural framework that connects reinforcement learning, optimal control, adaptive control, and cooperative control (Tsitsiklis 1984, Jadbabaie et al. 2003, Olfati-Saber and Murray 2004). Some examples are given of cooperative decision and control of dynamical systems on communication graphs.

11.1 REINFORCEMENT LEARNING

Dynamic programming (Chapter 6) is a method for determining optimal control solutions using Bellman's principle (Bellman 1957) by working backward in time from some desired goal states. Designs based on dynamic programming yield offline solution algorithms, which are then stored and implemented online forward in time. In this chapter we show that techniques based on reinforcement learning allow the design of optimal decision systems that learn optimal solutions online and forward in time. This allows both the system dynamics and the performance objectives to vary slowly with time. The methods studied here depend on solving a certain equation, known as Bellman's equation (Sutton and Barto 1998), whose solution both evaluates the performance of current control policies and provides methods for improving those policies.

Reinforcement learning (RL) refers to a class of learning methods that allow the design of adaptive controllers that learn online, in real time, the solutions to user-prescribed optimal control problems. In machine learning, reinforcement learning (Mendel and MacLaren 1970, Werbos 1991, Werbos 1992, Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998, Powell 2007, Cao 2007, Busoniu et al. 2009) is a method for solving optimization problems that involves an actor or agent that interacts with its environment and modifies its actions, or control policies, based on stimuli received in response to its actions. Reinforcement

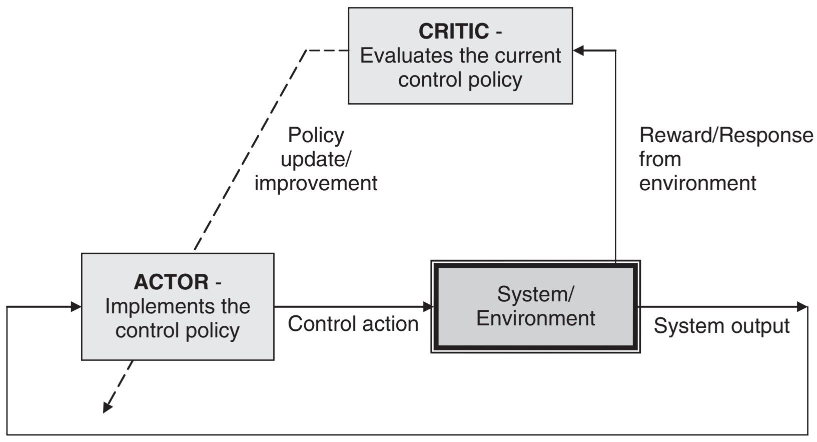
CRITIC - Evaluates the current control policy
Policy update/ improvement
Reward/Response from environment
ACTOR - Implements the control policy
Control action
System/ Environment
System output

FIGURE 11.1-1 Reinforcement learning with an actor–critic structure. This structure provides methods for learning optimal control solutions online based on data measured along the system trajectories.

learning is inspired by natural learning mechanisms, where animals adjust their actions based on reward and punishment stimuli received from the environment. Other reinforcement learning mechanisms operate in the human brain, where the dopamine neurotransmitter in the basal ganglia acts as a reinforcement informational signal that favors learning at the level of the neuron (Doya et al. 2001, Schultz 2004).

The actor–critic structures shown in Figure 11.1-1 (Barto et al. 1983) are one type of reinforcement learning algorithm. These structures give forward-in-time algorithms for computing optimal decisions that are implemented in real time where an actor component applies an action, or control policy, to the environment, and a critic component assesses the value of that action. The learning mechanism supported by the actor–critic structure has two steps, namely, policy evaluation, executed by the critic, followed by policy improvement, performed by the actor. The policy evaluation step is performed by observing from the environment the results of applying current control actions. These results are evaluated using a performance index that quantifies how close to optimal the current action is. Performance can be defined in terms of optimality objectives, such as minimum fuel, minimum energy, minimum risk, or maximum reward. Based on the assessment of the performance, one of several schemes can then be used to modify or improve the control policy in the sense that the new policy yields a performance value that is improved relative to the previous value. In this scheme, reinforcement learning is a means of learning optimal behaviors by observing the real-time responses from the environment to nonoptimal control policies.

Direct adaptive controllers tune the controller parameters to directly identify the controller. Indirect adaptive controllers identify the system, and the identified model is then used in design equations to compute a controller. Actor–critic schemes are a logical extension of this sequence in that they identify the performance value of the current control policy, and then use that information to update the controller.

This chapter presents the main ideas and algorithms of reinforcement learning and applies them to design adaptive feedback controllers that converge online to optimal control solutions relative to prescribed cost metrics. Using these techniques, we can solve online in real time the Riccati equation, the HJB equation, and the design equations in Chapter 10 for differential games. Some of the methods given here do not require knowledge of the system dynamics.

We start from a discussion of MDP and specifically focus on a family of techniques known as approximate or adaptive dynamic programming (ADP) (Werbos 1989, 1991, 1992) or neurodynamic programming (Bertsekas and Tsitsiklis 1996). We show that the use of reinforcement learning techniques provides optimal control solutions for linear or nonlinear systems using adaptive control techniques.

This chapter shows that reinforcement learning methods allow the solution of HJB design equations online, forward in time, and without knowing the full system dynamics. Specifically, the drift dynamics is not needed, but the input coupling function is needed. In the linear quadratic case, these methods determine the solution to the algebraic Riccati equation online, without solving the equation and without knowing the system A matrix. This chapter presents an expository development of ideas from reinforcement learning and ADP and their applications in feedback control systems. Surveys of ADP are given in Si et al. (2004), Lewis, Lendaris, and Liu (2008), Balakrishnan et al. (2008), Wang et al. (2009), and Lewis and Vrabie (2009).

11.2 MARKOV DECISION PROCESSES

A natural framework for studying RL is provided by Markov decision processes (MDP). Many dynamical decision problems can be cast into the framework of MDP. Included are feedback control systems for human engineered systems, feedback regulation mechanisms for population balance and survival of species (Darwin 1859, Luenberger 1979), decision-making in multiplayer games, and economic mechanisms for regulation of global financial markets. Therefore, we provide a development of MDP here. References for this material include (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998, Busoniu et al. 2009).

Consider the Markov decision process (MDP) (X, U, P, R), where X is a set of states and U is a set of actions or controls. The transition probabilities $P\colon X \times U \times X \to [0, 1]$ give for each state $x \in X$ and action $u \in U$ the conditional probability $P^u_{x,x'} = \Pr\{x'|x, u\}$ of transitioning to state $x' \in X$ given the MDP is in state x and takes action u. The cost function $R\colon X \times U \times X \to \mathbb{R}$ gives the expected immediate cost $R^u_{xx'}$ paid after transition to state $x' \in X$ given the MDP starts in state $x \in X$ and takes action $u \in U$. The Markov property refers to the fact that transition probabilities $P^u_{x,x'}$ depend only on the current state x and not on the history of how the MDP attained that state. An MDP is shown in Figure 11.2-1.

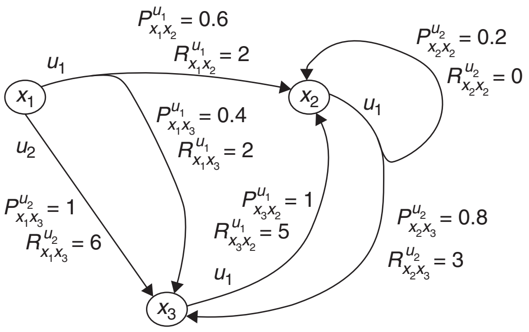

FIGURE 11.2-1 MDP shown as a finite state machine with controlled state transitions and costs associated with each transition.

The basic problem for MDP is to find a mapping $\pi\colon X \times U \to [0, 1]$ that gives for each state x and action u the conditional probability $\pi(x, u) = \Pr\{u|x\}$ of taking action u given the MDP is in state x. Such a mapping is termed a (closed-loop) control or action strategy or policy.

The strategy or policy $\pi(x, u) = \Pr\{u|x\}$ is called *stochastic* or mixed if there is a nonzero probability of selecting more than one control when in state x. We can view mixed strategies as probability distribution vectors having as component i the probability of selecting the ith control action while in state $x \in X$. If the mapping $\pi\colon X \times U \to [0, 1]$ admits only one control (with probability 1) when in every state x, it is called a *deterministic* policy. Then, $\pi(x, u) = \Pr\{u|x\}$ corresponds to a function mapping states into controls $\mu(x)\colon X \to U$.

Most work on reinforcement learning has been done for MDP that have finite state and action spaces. These are termed finite MDP.

Optimal Sequential Decision Problems

Dynamical systems evolve causally through time. Therefore, we consider sequential decision problems and impose a discrete stage index k such that the MDP takes an action and changes states at nonnegative integer stage values k. The stages may correspond to time or more generally to sequences of events. We refer to the stage value as the time. Denote state values and actions at time k by x_k, u_k. MDP traditionally evolve in discrete time.

Naturally occurring systems, including biological organisms and living species, have available a limited set of resources for survival and increase. Natural systems are therefore optimal in some sense. Likewise, human engineered systems should be optimal in terms of conserving resources such as cost, time, fuel, energy. Thus, it is important to capture the notion of optimality in selecting control policies for MDP.

Define, therefore, a *stage cost* at time k by $r_k = r_k(x_k, u_k, x_{k+1})$. Then $R^u_{xx'} = E\{r_k|x_k = x, u_k = u, x_{k+1} = x'\}$, with $E\{\cdot\}$ the expected value operator. Define a performance index as the sum of future costs over time interval $[k, k+T]$

$$J_{k,T} = \sum_{i=0}^{T} \gamma^i r_{k+i} = \sum_{i=k}^{k+T} \gamma^{i-k} r_i, \tag{11.2-1}$$

where $0 \le \gamma < 1$ is a discount factor that reduces the weight of costs incurred further in the future. T is a planning horizon over which decisions are to be made.

Traditional usage of MDP in the fields of computational intelligence and economics consider r_k as a reward incurred at time k, also known as *utility*, and $J_{k,T}$ as a discounted return, also known as strategic reward. We refer instead to stage costs and discounted future costs to be consistent with objectives in the control of dynamical systems. We may sometimes loosely call r_k the utility.

Consider that an agent selects a control policy $\pi_k(x_k, u_k)$ and uses it at each stage k of the MDP. We are primarily interested in stationary policies, where the conditional probabilities $\pi_k(x_k, u_k)$ are independent of k. Then $\pi_k(x, u) = \pi(x, u) = \Pr\{u|x\}$, for all k. Nonstationary deterministic policies have the form $\pi = \{\mu_0, \mu_1, \ldots\}$, where each entry is a function $\mu_k(x)$: $X \to U; k = 0, 1, \ldots$. Stationary deterministic policies are independent of time so that $\pi = \{\mu, \mu, \ldots\}$.

Select a fixed stationary policy $\pi(x, u) = \Pr\{u|x\}$. Then the ("closed-loop") MDP reduces to a Markov chain with state space X. That is, the transition probabilities between states are fixed with no further freedom of choice of actions. The transition probabilities of this Markov chain are given by

$$p_{x,x'} \equiv P^{\pi}_{x,x'} = \sum_u \Pr\{x'|x, u\} \Pr\{u|x\} = \sum_u \pi(x, u) P^u_{x,x'}, \tag{11.2-2}$$

where we have used the Chapman-Kolmogorov identity.

Under the assumption that the Markov chain corresponding to each policy (with transition probabilities given as in (11.2-2)) is ergodic, it can be shown that every MDP has a stationary deterministic optimal policy (Wheeler and Narendra 1986, Bertsekas and Tsitsiklis 1996). A Markov chain is *ergodic* if all states are positive recurrent and aperiodic (Luenberger 1979). Then, for a given policy there exists a stationary distribution $p_\pi(x)$ over X that gives the steady-state probability the Markov chain is in state x. We shall soon discuss more about the closed-loop Markov chain.

The *value* of a policy is defined as the conditional expected value of future cost when starting in state x at time k and following policy $\pi(x, u)$ thereafter,

$$V^{\pi}_k(x) = E_\pi\{J_{k,T}|x_k = x\} = E_\pi\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i | x_k = x\right\}. \tag{11.2-3}$$

Here, $E_\pi\{\cdot\}$ is the expected value given that the agent follows policy $\pi(x, u)$. $V^\pi(x)$ is known as the *value function* for policy $\pi(x, u)$. It tells the value of being in state x given that the policy is $\pi(x, u)$.

An important objective of MDP is to determine a policy $\pi(x, u)$ to minimize the expected future cost

$$\pi^*(x, u) = \arg\min_{\pi} V_k^{\pi}(s) = \arg\min_{\pi} E_{\pi}\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i | x_k = x\right\}. \tag{11.2-4}$$

This is termed the *optimal policy*, and the corresponding *optimal value* is given as

$$V_k^*(x) = \min_{\pi} V_k^{\pi}(x) = \min_{\pi} E_{\pi}\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i | x_k = x\right\}. \tag{11.2-5}$$

In computational intelligence and economics, when we talk about utilities and rewards, we are interested in maximizing the expected performance index.

A Backward Recursion for the Value

By using the Chapman-Kolmogorov identity and the Markov property we may write the value of policy $\pi(x, u)$ as

$$V_k^{\pi}(x) = E_{\pi}\{J_k | x_k = x\} = E_{\pi}\left\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i | x_k = x\right\} \tag{11.2-6}$$

$$V_k^{\pi}(x) = E_{\pi}\left\{r_k + \gamma \sum_{i=k+1}^{k+T} \gamma^{i-(k+1)} r_i | x_k = x\right\} \tag{11.2-7}$$

$$V_k^{\pi}(x) = \sum_u \pi(x, u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma E_{\pi}\left\{\sum_{i=k+1}^{k+T} \gamma^{i-(k+1)} r_i | x_{k+1} = x'\right\}\right]. \tag{11.2-8}$$

Therefore, the value function for policy $\pi(x, u)$ satisfies

$$V_k^{\pi}(x) = \sum_u \pi(x, u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^{\pi}(x')\right]. \tag{11.2-9}$$

This provides a backward recursion for the value at time k in terms of the value at time $k + 1$.

Dynamic Programming

The optimal cost can be written as

$$V_k^*(x) = \min_{\pi} V_k^{\pi}(x) = \min_{\pi} \sum_u \pi(x, u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^{\pi}(x')\right]. \tag{11.2-10}$$

Bellman's optimality principle (Bellman 1957) states that "an optimal policy has the property that no matter what the previous control actions have been, the remaining controls constitute an optimal policy with regard to the state resulting from those previous controls." Therefore, we may write

$$V_k^*(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x')\right]. \tag{11.2-11}$$

Suppose we now apply an arbitrary control u at time k and the optimal policy from time $k+1$ on. Then Bellman's optimality principle says that the optimal control at time k is given by

$$u_k^* = \arg\min_\pi \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x')\right]. \tag{11.2-12}$$

Under the assumption that the Markov chain corresponding to each policy (with transition probabilities given as in (11.2-2)) is ergodic, every MDP has a stationary deterministic optimal policy. Then we can equivalently minimize the conditional expectation over all actions u in state x. Therefore,

$$V_k^*(x) = \min_u \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x')\right], \tag{11.2-13}$$

$$u_k^* = \arg\min_u \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x')\right]. \tag{11.2-14}$$

The backward recursion (11.2-11), (11.2-13) forms the basis for dynamic programming (DP), which gives offline methods for working backward in time to determine optimal policies. DP was discussed in Chapter 6. It is an offline procedure for finding the optimal value and optimal policies that requires knowledge of the complete system dynamics in the form of transition probabilities $P_{x,x'}^u = \Pr\{x'|x,u\}$ and expected costs $R_{xx'}^u = E\{r_k | x_k = x, u_k = u, x_{k+1} = x'\}$. Once the optimal control has been found offline using DP, it is stored and implemented on the system online forward in time.

Bellman Equation and Bellman Optimality Equation (HJB)

Dynamic programming is a backward-in-time method for finding the optimal value and policy. By contrast, reinforcement learning is concerned with finding optimal policies based on causal experience by executing sequential decisions that improve control actions based on the observed results of using a current policy. This requires the derivation of methods for finding optimal values and optimal policies that can be executed forward in time. Here we develop the Bellman equation, which is the basis for such methods.

To derive forward-in-time methods for finding optimal values and optimal policies, set now the time horizon T to infinity and define the infinite-horizon cost

$$J_k = \sum_{i=0}^{\infty} \gamma^i r_{k+i} = \sum_{i=k}^{\infty} \gamma^{i-k} r_i. \tag{11.2-15}$$

The associated (infinite-horizon) value function for policy $\pi(x, u)$ is

$$V^{\pi}(x) = E_{\pi}\{J_k | x_k = x\} = E_{\pi}\left\{\sum_{i=k}^{\infty} \gamma^{i-k} r_i | x_k = x\right\}. \tag{11.2-16}$$

By using (11.2-8) with $T = \infty$ we see that the value function for policy $\pi(x, u)$ satisfies the *Bellman equation*

$$V^{\pi}(x) = \sum_u \pi(x, u) \sum_{x'} P^u_{xx'}\left[R^u_{xx'} + \gamma V^{\pi}(x')\right]. \tag{11.2-17}$$

This equation is of extreme importance in reinforcement learning. It is important that the same value function appears on both sides. This is due to the fact that the infinite-horizon cost was used. Therefore, (11.2-17) can be interpreted as a consistency equation that must be satisfied by the value function at each time stage. The Bellman equation expresses a relation between the current value of being in state x and the value(s) of being in the next state x' given that policy $\pi(x, u)$ is used.

The Bellman equation forms the basis for a family of reinforcement learning algorithms for finding optimal policies by using causal experiences received stagewise forward in time. In this context, the meaning of the Bellman equation is shown in Figure 11.2-2, where $V^{\pi}(x)$ may be considered as a predicted performance, $\sum\limits_u \pi(x, u) \sum\limits_{x'} P^u_{xx'} R^u_{xx'}$ the observed one-step reward, and $V^{\pi}(x')$

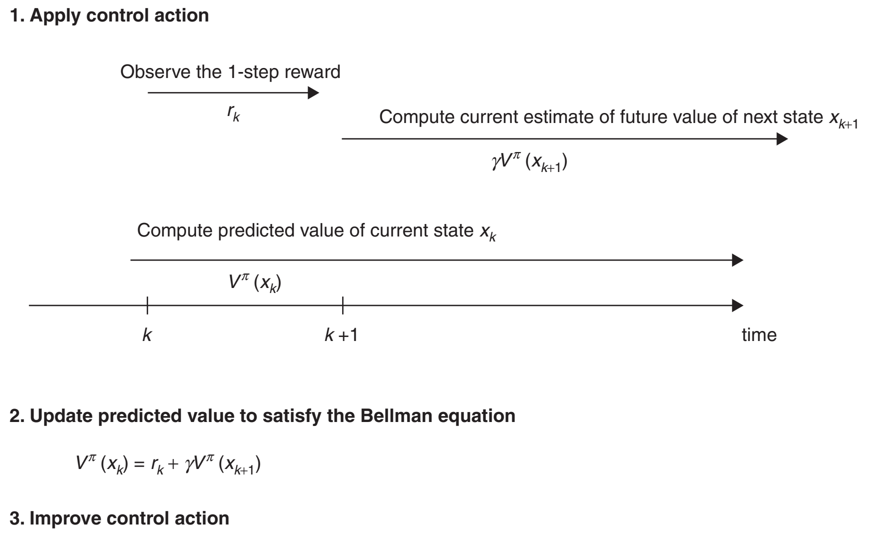
1. Apply control action
Observe the 1-step reward
r_k
Compute current estimate of future value of next state x_{k+1}
$\gamma V^{\pi}(x_{k+1})$
Compute predicted value of current state x_k
$V^{\pi}(x_k)$
k
$k+1$
time
2. Update predicted value to satisfy the Bellman equation
$V^{\pi}(x_k) = r_k + \gamma V^{\pi}(x_{k+1})$
3. Improve control action

FIGURE 11.2-2 Temporal difference interpretation of Bellman equation, showing how the Bellman equation captures the action, observation, evaluation, and improvement mechanisms of reinforcement learning.

a current estimate of future behavior. These notions are capitalized on in the subsequent discussion of temporal difference learning, which uses them to develop adaptive control algorithms that can learn optimal behavior online in real-time applications.

If the MDP is finite and has N states, then the Bellman equation (11.2-17) is a system of N simultaneous linear equations for the value $V^{\pi}(x)$ of being in each state x given the current policy $\pi(x,u)$. The optimal value satisfies

$$V^*(x) = \min_{\pi} V^{\pi}(x) = \min_{\pi} \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^{\pi}(x')\right]. \quad (11.2\text{-}18)$$

Bellman's optimality principle then yields the *Bellman optimality equation*

$$V^*(x) = \min_{\pi} V^{\pi}(x) = \min_{\pi} \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^{*}(x')\right]. \quad (11.2\text{-}19)$$

Equivalently, under the ergodicity assumption on the Markov chains corresponding to each policy, we have

$$V^*(x) = \min_{u} \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^{*}(x')\right]. \quad (11.2\text{-}20)$$

If the MDP is finite and has N states, then the Bellman optimality equation is a system of N nonlinear equations for the optimal value $V^*(x)$ of being in each state. The optimal control is given by

$$u^* = \arg\min_{u} \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^{*}(x')\right]. \quad (11.2\text{-}21)$$

Though the ideas just introduced may not seem familiar to the control engineer, it is shown in the next examples that they correspond to some familiar notions in feedback control system theory.

Example 11.2-1. Bellman Equation for Discrete-time Linear Quadratic Regulator (DT LQR)

This example studies the Bellman equation for the discrete-time LQR and shows that it is closely related to ideas developed in Chapter 2.

a. MDP Dynamics for Deterministic DT Systems

Consider the discrete-time (DT) linear quadratic regulator (LQR) problem where the MDP is deterministic and satisfies the state transition equation

$$x_{k+1} = Ax_k + Bu_k, \quad (11.2\text{-}22)$$

with k the discrete time index. The associated infinite horizon performance index has deterministic stage costs and is

$$J_k = \frac{1}{2}\sum_{i=k}^{\infty} r_i = \frac{1}{2}\sum_{i=k}^{\infty} \left(x_i^{\mathrm{T}} Q x_i + u_i^{\mathrm{T}} R u_i\right). \quad (11.2\text{-}23)$$

In this example, the state space $X = R^n$ and action space $U = R^m$ are infinite and continuous.

b. Bellman Equation for DT LQR: The Lyapunov Equation

The performance index J_k depends on the current state x_k and all future control inputs $u_k, u_{k+1}, \ldots$. Select a fixed stabilizing policy $u_k = \mu(x_k)$ and write the associated value function as

$$V(x_k) = \frac{1}{2}\sum_{i=k}^{\infty} r_i = \frac{1}{2}\sum_{i=k}^{\infty}\left(x_i^{\mathrm{T}} Q x_i + u_i^{\mathrm{T}} R u_i\right). \tag{11.2-24}$$

The value function for a fixed policy depends only on the initial state x_k. A difference equation equivalent to this infinite sum is given by

$$\begin{aligned} V(x_k) &= \frac{1}{2}\left(x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k\right) + \frac{1}{2}\sum_{i=k+1}^{\infty}\left(x_i^{\mathrm{T}} Q x_i + u_i^{\mathrm{T}} R u_i\right) \\ &= \frac{1}{2}\left(x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k\right) + V(x_{k+1}). \end{aligned} \tag{11.2-25}$$

That is, the positive definite solution $V(x_k)$ to this equation that satisfies $V(0) = 0$ is the value given by (11.2-24). Equation (11.2-25) is exactly the Bellman equation (11.2-17) for the LQR.

Assuming the value is quadratic in the state so that

$$V_k(x_k) = \tfrac{1}{2} x_k^{\mathrm{T}} P x_k, \tag{11.2-26}$$

for some kernel matrix P yields the Bellman equation form

$$2V(x_k) = x_k^{\mathrm{T}} P x_k = x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k + x_{k+1}^{\mathrm{T}} P x_{k+1}, \tag{11.2-27}$$

which, using the state equation, can be written as

$$2V(x_k) = x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k + (A x_k + B u_k)^{\mathrm{T}} P (A x_k + B u_k). \tag{11.2-28}$$

Assuming a constant (that is, stationary) state feedback policy $u_k = \mu(x_k) = -K x_k$ for some stabilizing gain K, we write

$$2V(x_k) = x_k^{\mathrm{T}} P x_k = x_k^{\mathrm{T}} Q x_k + x_k^{\mathrm{T}} K^{\mathrm{T}} R K x_k + x_k^{\mathrm{T}} (A - BK)^{\mathrm{T}} P (A - BK) x_k. \tag{11.2-29}$$

Since this holds for all state trajectories, we have

$$(A - BK)^{\mathrm{T}} P (A - BK) - P + Q + K^{\mathrm{T}} R K = 0. \tag{11.2-30}$$

This is a Lyapunov equation. That is, the Bellman equation (11.2-17) for the DT LQR is equivalent to a Lyapunov equation. Since the performance index is undiscounted ($\gamma = 1$), we must select a stabilizing gain K, that is, a stabilizing policy.

The formulations (11.2-25), (11.2-27), (11.2-29), (11.2-30) for the Bellman equation are all equivalent. Note that forms (11.2-25) and (11.2-27) do not involve the system dynamics (A, B). On the other hand, the Lyapunov equation (11.2-30) can be used only if we know the state dynamics (A, B). Optimal controls design using matrix equations is the standard procedure in control systems theory. Unfortunately, by assuming that (11.2-29) holds for all trajectories and going to (11.2-30), we lose all possibility of applying any sort of reinforcement learning algorithms to solve for the optimal control and value online by observing data along the system trajectories. By contrast, it will be shown that by employing the form (11.2-25) or (11.2-27) for the Bellman equation, we can devise RL algorithms for learning optimal solutions online by using temporal difference methods. That is, RL allows us to solve the Lyapunov equation online without knowing A or B.

c. Bellman Optimality Equation for DT LQR: The Algebraic Riccati Equation

The DT LQR Hamiltonian function is

$$H(x_k, u_k) = x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k + (Ax_k + Bu_k)^{\mathrm{T}} P (Ax_k + Bu_k) - x_k^{\mathrm{T}} P x_k. \qquad (11.2\text{-}31)$$

This is known as the temporal difference error in MDP. A necessary condition for optimality is the stationarity condition $\partial H(x_k, u_k)/\partial u_k = 0$, which is equivalent to (11.2-21). Solving this yields the optimal control

$$u_k = -Kx_k = -(B^{\mathrm{T}} P B + R)^{-1} B^{\mathrm{T}} P A x_k.$$

Putting this into (11.2-31) yields the DT algebraic Riccati equation (ARE)

$$A^{\mathrm{T}} P A - P + Q - A^{\mathrm{T}} P B (B^{\mathrm{T}} P B + R)^{-1} B^{\mathrm{T}} P A = 0. \qquad (11.2\text{-}32)$$

This is exactly the Bellman optimality equation (11.2-19) for the DT LQR. ■

The Closed-loop Markov Chain

References for this section include (Wheeler and Narendra 1986, Bertsekas and Tsitsiklis 1996, Luenberger 1979). Consider a finite MDP with N states. Select a fixed stationary policy $\pi(x, u) = \Pr\{u|x\}$ Then the "closed-loop" MDP reduces to a Markov chain with state space X. The transition probabilities of this Markov chain are given by (11.2-2). Enumerate the states using index $i = 1, \ldots, N$ and denote the transition probabilities (11.2-2) by $p_{ij} = p_{x=i,x'=j}$. Define the transition matrix $P = [p_{ij}] \in R^{N\times N}$. Denote the expected costs $r_{ij} = R_{ij}^{\pi}$ and define the cost matrix $R = [r_{ij}] \in R^{N\times N}$. Array the scalar values of the states $V(i)$ into a vector $V = [V(i)] \in R^N$.

With this notation, we may write the Bellman equation (11.2-17) as

$$V = \gamma P V + (P \odot R)\underline{1}, \qquad (11.2\text{-}33)$$

where $\odot$ is the Hadamard (element-by-element) matrix product, $P \odot R = [p_{ij} r_{ij}] \in R^{N\times N}$ is the cost-transition matrix, and $\underline{1}$ is the N-vector of 1's. This is a system of N linear equations for the value vector V of using the selected policy.

Transition matrix P is stochastic, that is, all row sums are equal to one. If the Markov chain is irreducible, that is, all states communicate with each other with nonzero probability, then P has a single eigenvalue at 1 and all other eigenvalues inside the unit circle (Gershgorin theorem). Then, if the discount factor is less than one, $I - \gamma P$ is nonsingular and there is a unique solution to (11.2-33) for the value, namely $V = (I - \gamma P)^{-1}(P \odot R)\underline{1}$.

Define vector $p \in R^N$ as a probability distribution vector (that is, its elements sum to 1) with element $p(i)$ being the probability that the Markov chain is in state i. Then, the evolution of p is given by

$$p_{k+1}^{\mathrm{T}} = p_k^{\mathrm{T}} P, \tag{11.2-34}$$

with k the time index and starting at some initial distribution p_0. The solution to this equation is

$$p_k^{\mathrm{T}} = p_0^{\mathrm{T}} P^k. \tag{11.2-35}$$

The limiting value of this recursion is the invariant or steady-state distribution, given by solving

$$p_\infty^{\mathrm{T}} = p_\infty^{\mathrm{T}} P, \quad p_\infty^{\mathrm{T}}(I - P) = 0. \tag{11.2-36}$$

Thus, p_∞ is the left eigenvector of the eigenvalue $\lambda = 1$ of P. Since P has all row sums equal one, the right eigenvector of $\lambda = 1$ is given by $\underline{1}$. If the Markov chain is irreducible, the eigenvalue $\lambda = 1$ is simple, and vector p_∞ has all elements positive. That is, all states have a nonzero probability of being occupied in steady state.

Define the average cost per stage and its value under the selected policy as

$$\overline{V}(i) = \lim_{T \to \infty} \frac{1}{T} E\left(\sum_{k=0}^{T-1} r_k | x_0 = i\right). \tag{11.2-37}$$

Define the vector $\overline{V} = [\overline{V}(i)] \in R^N$. Then we have

$$\overline{V} = (P \odot R)\underline{1}. \tag{11.2-38}$$

Define the expected value over all the states as $\overline{\overline{V}} = E_i\{\overline{V}(i)\}$. Then

$$\overline{\overline{V}} = p_\infty^{\mathrm{T}}(P \odot R)\underline{1}. \tag{11.2-39}$$

The left eigenvector $p_\infty = [p(i)]$ for $\lambda = 1$ has received a great deal of attention in the literature about cooperative control (Jadbabaie et al. 2003, Olfati-Saber and Murray 2004). The meaning of its elements $p(i)$ is interesting. Element $p(i)$ is the probability that the Markov chain is in state i at steady state, and also the percentage of time the Markov chain is in state i at steady state. Starting in state i, the time at which the Markov chain first returns to state i is a random variable known as the *hitting time* or first return time. Its expected value T_i is known as the *expected return time* to state i. If the Markov chain is irreducible, $T_i > 0, \forall i$; then $p(i) = 1/T_i$.

11.3 POLICY EVALUATION AND POLICY IMPROVEMENT

Given a current policy $\pi(x, u)$, we can determine its value (11.2-16) by solving the Bellman equation (11.2-17). As will be discussed, there are several methods of doing this, several of which can be implemented online in real time. This procedure is known as *policy evaluation*.

Moreover, given the value function for any policy $\pi(x, u)$, we can always use it to find another policy that is better, or at least no worse. This step is known as *policy improvement*. Specifically, suppose $V^{\pi}(x)$ satisfies (11.2-17). Then define a new policy $\pi'(x, u)$ by

$$\pi'(x, u) = \arg\min_{u} \sum_{x'} P^{u}_{xx'} \left[R^{u}_{xx'} + \gamma V^{\pi}(x') \right]. \tag{11.3-1}$$

Then it is easy to show that $V^{\pi'}(x) \leq V^{\pi}(x)$ (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998). The policy determined as in (11.3-1) is said to be *greedy* with respect to value function $V^{\pi}(x)$.

In the special case that $V^{\pi'}(x) = V^{\pi}(x)$ in (11.3-1), then $V^{\pi'}(x)$, $\pi'(x, u)$ satisfy (11.2-20) and (11.2-21) so that $\pi'(x, u) = \pi(x, u)$ is the optimal policy and $V^{\pi'}(x) = V^{\pi}(x)$ the optimal value. That is, (only) an optimal policy is greedy with respect to its own value. In computational intelligence, *greedy* refers to quantities determined by optimizing over short or 1-step horizons, without regard to potential impacts far into the future.

Now let us consider algorithms that repeatedly interleave the two procedures.

Policy evaluation by Bellman equation:

$$V^{\pi}(x) = \sum_{u} \pi(x, u) \sum_{x'} P^{u}_{xx'} \left[R^{u}_{xx'} + \gamma V^{\pi}(x') \right], \quad \text{for all } x \in S \subseteq X. \tag{11.3-2}$$

Policy improvement:

$$\pi'(x, u) = \arg\min_{u} \sum_{x'} P^{u}_{xx'} \left[R^{u}_{xx'} + \gamma V^{\pi}(x') \right], \quad \text{for all } x \in S \subseteq X. \tag{11.3-3}$$

S is a suitably selected subspace of the state space, to be discussed later. We call an application of (11.3-2) followed by an application of (11.3-3) one *step*. This is in contrast to the decision time stage k defined above.

At each step of such algorithms, one obtains a policy that is no worse than the previous policy. Therefore, it is not difficult to prove convergence under fairly mild conditions to the optimal value and optimal policy. Most such proofs are based on the Banach fixed-point theorem. Note that (11.2-20) is a fixed-point equation for $V^{*}(\cdot)$. Then the two equations (11.3-2), (11.3-3) define an associated map that can be shown under mild conditions to be a contraction map (Bertsekas and Tsitsiklis 1996 and Powell 2007). Then, it converges to the solution of (11.2-20) for any initial policy.

There is a large family of algorithms that implement the policy evaluation and policy improvement procedures in various ways, or interleave them differently, or select subspace $S \subseteq X$ in different ways, to determine the optimal value and optimal policy. We shall soon outline some of them.

The importance for feedback control systems of this discussion is that these two procedures can be implemented for dynamical systems online in real time by observing data measured along the system trajectories. This yields a family of adaptive control algorithms that converge to optimal control solutions. These algorithms are of the *actor–critic* class of reinforcement learning systems, shown in Figure 11.1-1. There, a critic agent evaluates the current control policy using methods based on (11.3-2). After this has been completed, the action is updated by an actor agent based on (11.3-3).

Policy Iteration

One method of reinforcement learning for using (11-3.2), (11-3.3) to find the optimal value and optimal policy is *policy iteration (PI).*

POLICY ITERATION (PI) ALGORITHM

Initialize.

Select an initial policy $\pi_0(x, u)$. Do for $j = 0$ until convergence:

Policy evaluation (value update):

$$V_j(x) = \sum_u \pi_j(x, u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right], \quad \text{for all } x \in X. \tag{11.3-4}$$

Policy improvement (policy update):

$$\pi_{j+1}(x, u) = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right], \quad \text{for all } x \in X. \tag{11.3-5}$$

At each step j the PI algorithm determines the solution of the Bellman equation (11.3-4) to compute the value $V_j(x)$ of using the current policy $\pi_j(x, u)$. This value corresponds to the infinite sum (11.2-16) for the current policy. Then the policy is improved using (11.3-5). The steps are continued until there is no change in the value or the policy.

Note that j is not the time or stage index k, but a PI step iteration index. It will be seen how to implement PI for dynamical systems online in real time by observing data measured along the system trajectories. Generally, data for multiple times k is needed to solve the Bellman equation (11.3-4) at each step j.

If the MDP is finite and has N states, then the policy evaluation equation (11.3-4) is a system of N simultaneous linear equations, one for each state. The PI algorithm must be suitably initialized to converge. The initial policy $\pi_0(x, u)$ and value V_0 must be selected so that $V_1 \leq V_0$. Then, for finite Markov chains with N states, PI converges in a finite number of steps (less than or equal to N) because there are only a finite number of policies.

The Bellman equation (11.3-4) is a system of simultaneous equations. Instead of directly solving the Bellman equation, we can solve it by an iterative policy evaluation procedure. Note that (11.3-4) is a fixed-point equation for $V_j(\cdot)$. It defines the iterative policy evaluation map

$$V_j^{i+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_j^i(x') \right], \quad i = 1, 2, \ldots \tag{11.3-6}$$

which can be shown to be a contraction map under rather mild conditions. By the Banach fixed-point theorem the iteration can be initialized at any non-negative value of $V_j^1(\cdot)$ and it will converge to the solution of (11.3-4). Under certain conditions, this solution is unique. A good initial value choice is the value function $V_{j-1}(\cdot)$ from the previous step $j-1$. On (close enough) convergence, we set $V_j(\cdot) = V_j^i(\cdot)$ and proceed to apply (11.3-5).

Index j in (11.3-6) refers to the step number of the PI algorithm. By contrast i is an iteration index. It is interesting to compare iterative policy evaluation (11.3-6) to the backward-in-time recursion (11.2-9) for the finite-horizon value. In (11.2-9), k is the time index. By contrast, in (11.3-6), i is an iteration index. Dynamic programming is based on (11.2-9) and proceeds backward in time. The methods for online optimal adaptive control described in this chapter proceed forward in time and are based on PI and similar algorithms.

The usefulness of these concepts is shown in the next example, where we use the theory of MDP and iterative policy evaluation to derive the relaxation algorithm for solution of Poisson's equation.

Example 11.3-1. Solution of Partial Differential Equations: Relaxation Algorithms

Consider Poisson's equation in two dimensions

$$\Delta V(x,y) = \nabla^2 V(x,y) = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) V(x,y) = f(x,y), \tag{11.3-7}$$

where $\Delta = \nabla^2$ is the Laplacian operator and ∇ the gradient. Function $f(x,y)$ is a forcing function, often specified on the boundary of a region. Discretizing the equation on a uniform mesh as shown in Figure 11.3-1 with grid size h in the (x,y) plane we write in terms of the forward difference

$$\begin{aligned}
\frac{\partial V(x,y)}{\partial x} &= \frac{1}{h}\left(V(x+h,y) - V(x,y)\right) + O(h), \\
\frac{\partial^2 V(x,y)}{\partial x^2} &= \frac{1}{h^2}\left(V(x+h,y) + V(x-h,y) - 2V(x,y)\right) + O(h^2), \\
\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial x^2} \right) V(x,y) &= \frac{1}{h^2}(V(x+h,y) + V(x-h,y) + V(x,y+h) \\
&\quad + V(x,y-h) - 4V(x,y)) \approx f(x,y).
\end{aligned}$$

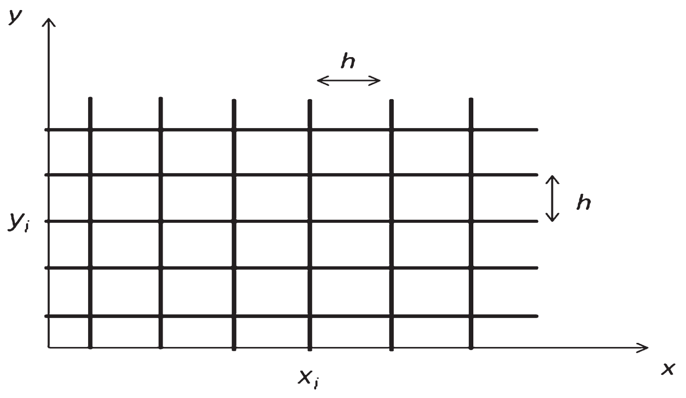

FIGURE 11.3-1 Sampling in the (x, y) plane with a uniform mesh of size h.

Indexing the (x, y) positions with an index i and denoting $X_i = (x_i, y_i)$ we have approximately to order h^2

$$\frac{1}{h^2}\left(V(X_{i,R}) + V(X_{i,L}) + V(X_{i,u}) + V(X_{i,D}) - 4V(X_i)\right) = f(X_i), \tag{11.3-8}$$

for states not on the boundary and a similar equation for boundary states. $X_{i,R}$ denotes the (x, y) location of the state to the right of X_i, and similarly for states to the left, up, and down relative to X_i. This is a set of N simultaneous equations.

We can interpret (11.3-8) as a Bellman equation (11.2-17) for a properly defined underlying MDP. Specifically, define an MDP with stage costs equal to zero and $\gamma = 1$. Then, we may interpret the state transitions as deterministic and the control as the equi-probable control with probabilities of 1/4 (for nonboundary nodes) for moving up, right, down, and left. Alternatively, we may interpret the control as deterministic and the state transition probabilities as equi-probable with probabilities of 1/4 for moving up, right, down, and left. In either case, the Bellman equation for the MDP is (11.3-8). Functions $f(X_i)$ may be interpreted as stage costs.

Now the iterative policy evaluation method of solution (11.3-6) performs the iterations

$$V^{m+1}(X_i) = \tfrac{1}{4}V^m(X_{i,U}) + \tfrac{1}{4}V^m(X_{i,R}) + \tfrac{1}{4}V^m(X_{i,D}) + \tfrac{1}{4}V^m(X_{i,L}) - \tfrac{h^2}{4}f(X_i). \tag{11.3-9}$$

The theory of MDP guarantees that this algorithm will converge to the solution of (11.3-8). These updates may be done for all nodes or states simultaneously. Then, this is nothing but the relaxation method for numerical solution of Poisson's equation. The relaxation algorithm converges, but may do so slowly. Variants have been developed to speed it up. ■

Value Iteration

A second method for using (11.3-2), (11.3-3) in reinforcement learning is *value iteration (VI)*, which is easier to implement than policy iteration.

VALUE ITERATION (VI) ALGORITHM

Initialize.

Select an initial policy $\pi_0(x,u)$. Do for $j = 0$ until convergence—

Value update:

$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right], \quad \text{for all } x \in S_j \subseteq X. \tag{11.3-10}$$

Policy improvement:

$$\pi_{j+1}(x,u) = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_{j+1}(x') \right], \quad \text{for all } x \in S_j \subseteq X. \tag{11.3-11}$$

We may combine the value update and policy improvement into one equation to obtain the equivalent form for VI

$$V_{j+1}(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right], \quad \text{for all } x \in S_j \subseteq X. \tag{11.3-12}$$

or, equivalently under the ergodicity assumption, in terms of deterministic policies

$$V_{j+1}(x) = \min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right], \quad \text{for all } x \in S_j \subseteq X. \tag{11.3-13}$$

Note that, now, (11.3-10) is a simple one-step recursion, not a system of linear equations as is (11.3-4) in the PI algorithm. In fact, VI uses simply one iteration of (11.3-6) in its value update step. It does not find the value corresponding to the current policy, but takes only one iteration toward that value. Again, j is not the time index, but the VI step index.

It will be seen later how to implement VI for dynamical systems online in real time by observing data measured along the system trajectories. Generally, data for multiple times k is needed to solve the update (11.3-10) for each step j.

Asynchronous Value Iteration. Standard VI takes the update set as $S_j = X$, for all j. That is, the value and policy are updated for all states simultaneously. *Asynchronous* VI methods perform the updates on only a subset of the states at each step. In the extreme case, one may perform the updates on only one state at each step.

It is shown in Bertsekas and Tsitsiklis (1996) that standard VI ($S_j = X$, for all j) converges for finite MDP for any initial conditions when the discount factor satisfies $0 < \gamma < 1$. When $S_j = X$, for all j and $\gamma = 1$ an absorbing state is added and a "properness" assumption is needed to guarantee convergence to the optimal value. When a single state is selected for value and policy updates at each step, the algorithm converges, for any choice of initial value, to the optimal cost and policy if each state is selected for update infinitely often. More general algorithms result if value update (11.3-10) is performed multiple times for various choices of S_j prior to a policy improvement. Then updates (11.3-10) and (11.3-11) must be performed infinitely often for each state, and a monotonicity assumption must be satisfied by the initial starting value.

Considering (11.2-20) as a fixed-point equation, VI is based on the associated iterative map (11.3-10), (11.3-11), which can be shown under certain conditions to be a contraction map. In contrast to PI, which converges under certain conditions in a finite number of steps, VI generally takes an infinite number of steps to converge (Bertsekas and Tsitsiklis 1996). Consider finite MDP and consider the transition probability graph having probabilities (11.2-2) for the Markov chain corresponding to an optimal policy $\pi^*(x, u)$. If this graph is acyclic for some $\pi^*(x, u)$, then VI converges in at most N steps when initialized with a large value.

Having in mind the dynamic programming equation (11.2-9) and examining the VI value update (11.3-10), we can interpret $V_j(x')$ as an approximation or estimate for the future stage cost-to-go from the future state x'. See Figure 11.2-2. Those algorithms wherein the future cost estimate are themselves costs or values for some policy are called *rollout algorithms* in Bertsekas and Tsitsiklis (1996). These policies are forward looking and self-correcting. They are closely related to receding horizon control, as shown in Zhang et al. (2009).

MDP, policy iteration, and value iteration provide connections between optimal control, decisions on finite graphs, and cooperative control of networked systems, as amplified in the next examples. The first example shows how to use VI to derive the Bellman Ford algorithm for finding the shortest path in a graph to a destination node. The second example uses MDP and iterative policy evaluation to develop control protocols familiar in cooperative control of distributed dynamical systems. The third example shows that for the discrete-time LQR, policy iteration and value iteration can be used to derive algorithms for solution of the optimal control problem that are quite familiar in the feedback control systems community, including Hewer's algorithm.

Example 11.3-2. Deterministic Shortest Path Problems: The Bellman Ford Algorithm

A special case of finite MDP is the *shortest path problems* (Bertsekas and Tsitsiklis 1996), which are undiscounted $\gamma = 1$, and have a cost-free termination or absorbing state. These effectively have the time horizon T finite since the number of states is finite. The setup is shown in Figure 11.3-2.

Consider a directed graph $G = (V, E)$ with a nonempty finite set of N nodes $V = \{v_1, \dots, v_N\}$ and a set of edges or arcs $E \subseteq V \times V$. We assume the graph is simple, that is, it has no repeated edges and $(v_i, v_i) \notin E, \forall i$ no self-loops. Let the edges have weights $e_{ij} \geq 0$, with $e_{ij} > 0$ if $(v_i, v_j) \in E$ and $e_{ij} = 0$ otherwise. Note $e_{ii} = 0$. The set of neighbors of a node v_i is $N_i = \{v_j : (v_i, v_j) \in E\}$, that is, the set of nodes with arcs coming out from v_i.

Consider an agent moving through the graph along edges between nodes. Interpret the control at each node as the decision on which edge to follow leading out of that

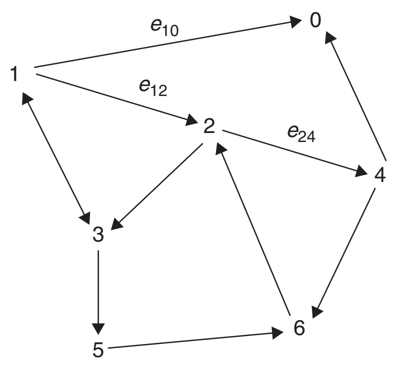

FIGURE 11.3-2 Sample graph for shortest path routing problem. The absorbing state 0 has only incoming edges. The objective is to find the shortest path from all nodes to node 0 using only local neighborhood computations.

node. Assume deterministic controls and state transitions. Interpret the edge weights e_{ij} as deterministic costs incurred by moving along that link. Then, the value iteration algorithm (11.3-13), (11.3-11) in this general digraph is

$$V_i(k+1) = \min_{j \in N_i}(e_{ij} + V_j(k)), \tag{11.3-14}$$

$$u_i(k+1) = \arg\min_{j \in N_i}(e_{ij} + V_j(k)), \tag{11.3-15}$$

with k an iteration index. We have changed the notation to conform to existing practice in cooperative control theory. Endow the graph with one node, v_0, which uses as its update law $V_0(k+1) = V_0(k) = 0$. This corresponds to an absorbing state in MDP parlance. It is interpreted as a node having only incoming edges, none outgoing.

The VI algorithm in this scenario is nothing but the Bellman-Ford algorithm, which finds the shortest path from any node in the graph to the absorbing node v_0. In the terminology of cooperative systems, each node v_i is endowed with two state variables. The node state variable $V_i(k)$ keeps track of the value, that is, the shortest path length to node v_0, while node state variable $u_i(k)$ keeps track of which direction to follow while leaving the ith node in order to follow the shortest path.

The VI update iterations may be performed on all nodes simultaneously. Alternatively, updates may be performed on one state at a time, as in asynchronous VI. With simultaneous updates at all nodes, according to results about VI (Bertsekas and Tsitsiklis 1996) it is known that this algorithm converges to the solution to the shortest path problem in a finite number of steps (less than or equal to N) if there is a path from every node to node v_0 and the graph is acyclic. It is known further that the algorithm converges, possibly in an infinite number of iterations, if there are no cycles with net negative gain (that is, the product of gains around the cycle is not negative). With asynchronous updates, the algorithm converges under these connectivity conditions if each node is selected for update infinitely often.

What Is a State?

Note that in the terminology of MDP, the nodes are termed states, so that the state space is finite. On the other hand, in the terminology of feedback control systems theory, the state (variable) of each node is $V_i(m)$, which is a real number so that the state space is continuous. ■

Example 11.3-3. Cooperative Control Systems

Consider the directed graph $G = (V, E)$ with N nodes $V = \{v_1, \ldots, v_N\}$ and edge weights $e_{ij} \geq 0$, with $e_{ij} > 0$ if $(v_i, v_j) \in E$ and $e_{ij} = 0$ otherwise. The set of neighbors of a node v_i is $N_i = \{v_j \colon (v_i, v_j) \in E\}$, that is, the set of nodes with arcs coming out from v_i. Define the out-degree of node i in graph G as $d_i = \sum_{j \in N_i} e_{ij}$. Figure 11.3-3 shows a representative graph topology.

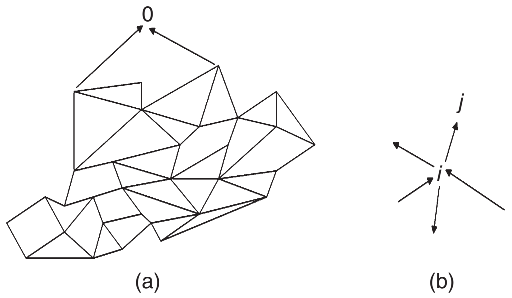

FIGURE 11.3-3 Cooperative control of multiple systems linked by a communication graph structure. The objective is for all nodes to reach consensus to the value of control node 0 by using only local neighborhood communication. (a) Representative graph structure. (b) Local neighborhood of node i. Each node has edges incoming and outgoing. The outgoing edges show the states reached from node i in one step.

On the graph G, define a MDP that has a stochastic policy $\pi(v_i, u_{ij}) = \Pr\{u_{ij} | v_i\}$, where u_{ij} means the control action that takes the MDP from node v_i to node v_j. Let the transition probabilities be deterministic. Endow the graph with one absorbing node v_0 that is connected to a few of the existing nodes. If there is an edge from node v_i to node v_0, define $b_i \equiv e_{i0}$ as its edge weight. Let actions u_{ij} have probabilities $u_{ii} = 1/(1 + d_i + b_i)$, $u_{ij} = e_{ij}/(1 + d_i + b_i)$, that is, the MDP may return to the same state in one step. Let the stage costs all be zero. Then, the Bellman equation (11.2-17) is

$$V_i = \frac{1}{1 + d_i + b_i} \left[V_i + \sum_{j \in N_i} e_{ij} V_j + b_i V_0 \right], \tag{11.3-16}$$

where V_i is the value of node v_i. This is a set of N simultaneous equations in the values $V_i, i = 1, N$ of the nodes.

Iterative policy evaluation (11.3-6) can be used to solve this set of equations. The iterative policy evaluation algorithm is written here as

$$V_i(k+1) = \frac{1}{1 + d_i + b_i} \left[V_i(k) + \sum_{j \in N_i} e_{ij} V_j(k) + b_i V_0 \right], \tag{11.3-17}$$

with k the iteration index, which can be thought of here as a time index. Node v_0 keeps its value constant at V_0.

The updates in (11.3-17) may be done simultaneously at all nodes. Then the theory of MDP shows that it is guaranteed to converge to the solution of the set of equations (11.3-16). Alternatively, one may use the theory of asynchronous VI or generalized PI to motivate other update schemes. For instance, if only one node is updated at each value of k, the theory of MDP shows that the algorithm still converges as long as each node is selected for update infinitely often.

Algorithm (11.3-17) can be written as the local control protocol

$$V_i(k+1) = V_i(k) + \frac{1}{1+d_i+b_i}\left[\sum_{j\in N_i} e_{ij}\left(V_j(k) - V_i(k)\right) + b_i\left(V_0 - V_i(k)\right)\right]. \tag{11.3-18}$$

On convergence $V_i(k+1) = V_i(k)$ so that the term in square brackets converges to zero. Assuming all nodes have a path to the absorbing node v_0, it is easy to show that this guarantees that $\left\| V_j(k) - V_i(k)\right\| \to 0,\ \|V_0 - V_i(k)\| \to 0,$ for all i, j; that is, all nodes reach the same consensus value, namely the value of the absorbing node (Jadbabaie et al. 2003, Olfati-Saber and Murray 2004).

The term in square brackets in (11.3-18) is known as the *temporal difference error* for this MDP.

Routing Graph vs. Control Graph

The graphs used in routing problems have edges from node i to node j if the transition probabilities (11.2-2) in the MDP using a fixed policy $\pi(x_i, u)$, namely $p_{ij} \equiv P^{\pi}_{x_i,x_j} = \sum_u \pi(x_i, u) P^u_{x_i,x_j}$, are nonzero. The edge weights are taken as $e_{ij} = p_{ij}$. Then, e_{ij} is nonzero if there is an edge coming *out* of node i to node j. By contrast, the edge weights a_{ij} for graphs in cooperative control problems are generally taken as nonzero if there is an edge coming *in* from node j to node i, that is, $a_{ij} > 0$ iff $(v_j, v_i) \in E$ This is interpreted to mean that information from node j is available to node i for its decision process in computing its control input.

It is convenient to think of the former as motion or routing graphs, and the latter as information flow or decision and control graphs. In fact, the routing graph is the reverse of the decision graph. The reverse graph of a given graph $G = (V, E)$ is the graph with the same node set, but all edges reversed. Note that the Bellman-Ford protocols (11.3-14), (11.3-15) assume that node i gets information from its neighbor node j, but that the shortest path problem is, in fact, solved for motion in the reverse graph having edges e_{ij}.

Define the matrix of transition probabilities $P = [p_{ij}]$ and the adjacency matrix $A = [a_{ij}]$. Then $A = P^{\mathrm{T}}$. ■

Example 11.3-4. Policy Iteration and Value Iteration for the DT LQR

The Bellman equation (11.2-17) is equivalent for the DT LQR to all the formulations (11.2-25), (11.2-27), (11.2-29), (11.2-30) in Example 11.2-1. We may use any of these to implement policy iteration and value iteration.

a. Policy Iteration: Hewer's Algorithm

With step index j, and using superscripts to denote algorithm steps and subscripts to denote the time k, the PI policy evaluation step (11.3-4) applied on (11.2-25) in

Example 11.2-1 yields

$$V^{j+1}(x_k) = \tfrac{1}{2}\left(x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k\right) + V^{j+1}(x_{k+1}). \tag{11.3-19}$$

PI applied on (11.2-27) yields

$$x_k^{\mathrm{T}} P^{j+1} x_k = x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k + x_{k+1}^{\mathrm{T}} P^{j+1} x_{k+1}, \tag{11.3-20}$$

and PI on (11.2-30) yields the Lyapunov equation

$$0 = (A - BK^j)^{\mathrm{T}} P^{j+1} (A - BK^j) - P^{j+1} + Q + (K^j)^{\mathrm{T}} R K^j. \tag{11.3-21}$$

In all cases the PI policy improvement step is

$$\mu^{j+1}(x_k) = K^{j+1} x_k = \arg\min(x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k + x_{k+1}^{\mathrm{T}} P^{j+1} x_{k+1}), \tag{11.3-22}$$

which can be written explicitly as

$$K^{j+1} = -(B^{\mathrm{T}} P^{j+1} B + R)^{-1} B^{\mathrm{T}} P^{j+1} A. \tag{11.3-23}$$

PI algorithm format (11.3-21), (11.3-23) relies on repeated solutions of Lyapunov equations at each step, and is nothing but Hewer's algorithm, well known in control systems theory. It was proven in Hewer (1971) to converge to the solution of the Riccati equation (11.2-32) in Example 11.2-1 if (A, B) is reachable and $(A, \sqrt{Q})$ is observable. It is an offline algorithm that requires complete knowledge of the system dynamics (A, B) to find the optimal value and control. It requires that the initial gain K^0 be stabilizing.

b. Value Iteration: Lyapunov Recursions

Applying VI (11.3-10) to Bellman equation format (11.2-27) in Example 11.2-1 yields

$$x_k^{\mathrm{T}} P^{j+1} x_k = x_k^{\mathrm{T}} Q x_k + u_k^{\mathrm{T}} R u_k + x_{k+1}^{\mathrm{T}} P^{j} x_{k+1}, \tag{11.3-24}$$

and on format (11.2-30) in Example 11.2-1 yields the Lyapunov recursion

$$P^{j+1} = (A - BK^j)^{\mathrm{T}} P^{j} (A - BK^j) + Q + (K^j)^{\mathrm{T}} R K^j. \tag{11.3-25}$$

In both cases the policy improvement step is still given by (11.3-22), (11.3-23).

VI algorithm format (11.3-25), (11.3-23) is simply a Lyapunov recursion, which is easy to implement and does not, in contrast to PI, require Lyapunov equation solutions. This algorithm was shown in Lancaster and Rodman (1995) to converge to the solution of the Riccati equation (11.2-32) in Example 11.2-1. It is an offline algorithm that requires complete knowledge of the system dynamics (A, B) to find the optimal value and control. It does not require that the initial gain K^0 be stabilizing, and can be initialized with any feedback gain.

c. Online Solution of the Riccati Equation without Knowing Plant Matrix A

Hewer's algorithm and the Lyapunov recursion algorithm are both offline methods for solving the algebraic Riccati equation (11.2-32) in Example 11.2-1. Full knowledge of the plant dynamics (A, B) is needed to implement these algorithms. By contrast, it will be seen that both PI Algorithm format (11.3-20), (11.3-22) and VI Algorithm format (11.3-24), (11.3-22) can be implemented online to determine the optimal value and control *in real time* using data measured along the system trajectories, and without knowing the system A matrix. This is accomplished through the temporal difference methods to be presented. That is, RL allows the solution of the algebraic Riccati equation online without knowing the system A matrix.

d. Iterative Policy Evaluation

Given a fixed policy K, the iterative policy evaluation procedure (11.3-6) becomes

$$P^{j+1} = (A - BK)^{\mathrm{T}} P^j (A - BK) + Q + K^{\mathrm{T}} RK. \tag{11.3-26}$$

This recursion converges to the solution to the Lyapunov equation $P = (A - BK)^{\mathrm{T}} P(A - BK) + Q + K^{\mathrm{T}} RK$ if $(A - BK)$ is stable, for any choice of initial value P^0. ■

Generalized Policy Iteration

In PI one fully solves the system of linear equations (11.3-4) at each step to compute the value (11.2-16) of using the current policy $\pi_j(x, u)$. This can be accomplished by running iterations (11.3-6) until convergence at each step j. By contrast, in VI one takes only one iteration of (11.3-6) in the value update step (11.3-10). Generalized policy iteration (GPI) algorithms make several iterations (11.3-6) in their value update step.

Generally, PI converges to the optimal value in fewer steps j, since it does more work in solving equations at each step. On the other hand, VI is the easiest to implement as it takes only one iteration of a recursion as per (11.3-10). GPI provides a suitable compromise between computational complexity and convergence speed. GPI is a special case of the VI algorithm given above, where we select $S_j = X$, for all j and perform value update (11.3-10) multiple times before each policy update (11.3-11).

Q Function

The conditional expected value in (11.2-13)

$$Q_k^*(x, u) = \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x') \right] = E_\pi \{ r_k + \gamma V_{k+1}^*(x') | x_k = x, u_k = u \}. \tag{11.3-27}$$

is known as the *optimal Q (quality) function* (Watkins 1989, Watkins and Dayan 1992). This has also been called the action-value function (Sutton and Barto 1998). It is equal to the expected return for taking an arbitrary action u at time k in state x and thereafter following an optimal policy. It is a function of the current state x and the action u.

In terms of the Q function, the Bellman optimality equation has the particularly simple form

$$V_k^*(x) = \min_u Q_k^*(x, u), \tag{11.3-28}$$

$$u_k^* = \arg\min_u Q_k^*(x, u). \tag{11.3-29}$$

Given any fixed policy $\pi(x, u)$, define the Q function for that policy as

$$Q_k^\pi(x, u) = E_\pi\{r_k + \gamma V_{k+1}^\pi(x') | x_k = x, u_k = u\} = \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^\pi(x')\right], \tag{11.3-30}$$

where we have used (11.2-9). This is equal to the expected return for taking an arbitrary action u at time k in state x and thereafter following the existing policy $\pi(x, u)$. The meaning of the Q function is elucidated by the next example.

Example 11.3-5. Q Function for the DT LQR

The Q function following a given policy $u_k = \mu(x_k)$ is defined in (11.3-30). For the DT LQR in Example 11.2-1 the Q function is

$$Q(x_k, u_k) = \tfrac{1}{2}\left(x_k^{\mathrm T} Q x_k + u_k^{\mathrm T} R u_k\right) + V(x_{k+1}), \tag{11.3-31}$$

where the control u_k is arbitrary and the policy $u_k = \mu(x_k)$ is followed for $k + 1$ and subsequent times. Writing

$$Q(x_k, u_k) = x_k^{\mathrm T} Q x_k + u_k^{\mathrm T} R u_k + (Ax_k + Bu_k)^{\mathrm T} P (Ax_k + Bu_k), \tag{11.3-32}$$

with P the Riccati solution yields the Q function for the DT LQR:

$$Q(x_k, u_k) = \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^{\mathrm T} \begin{bmatrix} A^{\mathrm T}PA + Q & B^{\mathrm T}PA \\ A^{\mathrm T}PB & B^{\mathrm T}PB + R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}. \tag{11.3-33}$$

Define

$$Q(x_k, u_k) \equiv \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^{\mathrm T} S \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^{\mathrm T} \begin{bmatrix} S_{xx} & S_{xu} \\ S_{ux} & S_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \tag{11.3-34}$$

for kernel matrix S.

Applying $\partial Q(x_k, u_k)/\partial u_k = 0$ to (11.3-34) yields

$$u_k = -S_{uu}^{-1} S_{ux} x_k, \tag{11.3-35}$$

and to (11.3-33) yields

$$u_k = -(B^{\mathrm T}PB + R)^{-1} B^{\mathrm T}PAx_k. \tag{11.3-36}$$

The latter equation requires knowledge of the system dynamics (A, B) to perform the policy improvement step of either PI or VI. On the other hand, the former equation requires knowledge only of the Q-function matrix kernel S. We shall subsequently show how to use RL temporal difference methods to determine the kernel matrix S online in real time without knowing the system dynamics (A, B) using data measured along the system trajectories. This provides a family of Q learning algorithms that can solve the algebraic Riccati equation online without knowing the system dynamics (A, B). ■

Note that $V_k^\pi(x) = Q_k^\pi(x, \pi(x, u))$ so that (11.3-30) may be written as the backward recursion in the Q function:

$$Q_k^\pi(x, u) = \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma Q_{k+1}^\pi(x', \pi(x', u')) \right]. \tag{11.3-37}$$

The Q function is a 2-dimensional function of both the current state x and the action u. By contrast, the value function is a 1-dimensional function of the state. For finite MDP, the Q function can be stored as a 2-D lookup table at each state/action pair. Note that direct minimization in (11.2-11), (11.2-12) requires knowledge of the state transition probabilities $P_{xx'}^u$ (system dynamics) and costs $R_{xx'}^u$. By contrast, the minimization in (11.3-28), (11.3-29) requires knowledge only of the Q function and not the system dynamics.

The importance of the Q function is twofold. First, it contains information about control actions in every state. As such, the best control in each state can be selected using (11.3-29) by knowing only the Q function. Second, the Q function can be estimated online in real time directly from data observed along the system trajectories, without knowing the system dynamics information (that is, the transition probabilities). We shall see how this is accomplished later.

The infinite horizon Q function for a prescribed fixed policy is given by

$$Q^\pi(x, u) = \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V^\pi(x') \right]. \tag{11.3-38}$$

The Q function also satisfies a Bellman equation. Note that, given a fixed policy $\pi(x, u)$,

$$V^\pi(x) = Q^\pi(x, \pi(x, u)), \tag{11.3-39}$$

whence according to (11.3-38) the Q function satisfies the Bellman equation

$$Q^\pi(x, u) = \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma Q^\pi(x', \pi(x', u')) \right]. \tag{11.3-40}$$

It is important that the same Q function Q^π appears on both sides of this equation. The Bellman optimality equation for the Q function is

$$Q^*(x, u) = \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma Q^*(x', \pi^*(x', u')) \right], \tag{11.3-41}$$

$$Q^*(x,u) = \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma \min_{u'} Q^*(x',u') \right]. \tag{11.3-42}$$

Compare (11.2-20) and (11.3-42), where the minimum operator and the expected value operator occurrences are reversed.

Based on Bellman equation (11.3-40), PI and VI are especially easy to implement in terms of the Q function, as follows.

PI USING Q FUNCTION

Policy evaluation (value update):

$$Q_j(x,u) = \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma Q_j(x', \pi_j(x',u')) \right], \quad \text{for all } x \in X. \tag{11.3-43}$$

Policy improvement:

$$\pi_{j+1}(x,u) = \arg\min_u Q_j(x,u), \quad \text{for all } x \in X. \tag{11.3-44}$$

VI USING Q FUNCTION

Value update:

$$Q_{j+1}(x,u) = \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma Q_j(x', \pi_j(x',u')) \right], \quad \text{for all } x \in S_j \subseteq X. \tag{11.3-45}$$

Policy improvement:

$$\pi_{j+1}(x,u) = \arg\min_u Q_{j+1}(x,u), \quad \text{for all } x \in S_j \subseteq X. \tag{11.3-46}$$

Combining both steps of VI yields the form

$$Q_{j+1}(x,u) = \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma \min_{u'} Q_j(x',u') \right], \quad \text{for all } x \in S_j \subseteq X, \tag{11.3-47}$$

which should be compared to (11.3-13)

As shall be seen, the importance of the Q function is that these algorithms can be implemented online in real time, without knowing the system dynamics, by measuring data along the system trajectories. They yield optimal adaptive control algorithms—that is, adaptive control algorithms that converge online to optimal control solutions.

Methods for Implementing PI and VI

There are different methods for performing the value and policy updates for PI and VI (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998, Powell 2007). The main three are exact computation, Monte Carlo methods, and temporal difference (TD) learning. The last two methods can be implemented without knowledge of the system dynamics. TD learning is the means by which optimal adaptive control algorithms may be derived for dynamical systems. Therefore, TD is covered in the next section.

Exact computation. PI requires solution at each step of Bellman equation (11.3-4) for the value update. For a finite MDP with N states, this is a set of linear equations in N unknowns (the values of each state). VI requires performing the one-step recursive update (11.3-10) at each step for the value update. Both of these can be accomplished exactly if we know the transition probabilities $P^u_{x,x'} = \Pr\{x'|x, u\}$ and costs $R^u_{xx'}$ of the MDP. This corresponds to knowing full system dynamics information. Likewise, the policy improvements (11.3-5), (11.3-11) can be explicitly computed if the dynamics are known. It was shown in Example 11.2-1 that, for the DT LQR, the exact computation method for computing the optimal control yields the Riccati equation solution approach. As shown in Example 11.3-4, PI and VI boil down to repetitive solutions of Lyapunov equations or Lyapunov recursions. In fact, PI becomes nothing but Hewer's method (Hewer 1971), and VI becomes a well-known Lyapunov recursion scheme that was shown to converge in Lancaster and Rodman (1995). These are offline methods relying on matrix equation solutions and requiring complete knowledge of the system dynamics.

Monte Carlo learning is based on the definition (11.2-16) for the value function, and uses repeated measurements of data to approximate the expected value. The expected values are approximated by averaging repeated results along sample paths. An assumption on the ergodicity of the Markov chain with transition probabilities (11.2-2) for the given policy being evaluated is implicit. This is suitable for episodic tasks, with experience divided into episodes (Sutton and Barto 1998)—namely, processes that start in an initial state and run until termination, and are then restarted at a new initial state. For finite MDP, Monte Carlo methods converge to the true value function if all states are visited infinitely often. Therefore, to ensure good approximations of value functions, the episode sample paths must go through all the states $x \in X$ many times. This is called the problem of maintaining exploration. There are several ways of ensuring this, one of which is to use *exploring starts*, in which every state has nonzero probability of being selected as the initial state of an episode.

Monte Carlo techniques are useful for dynamic systems control because the episode sample paths can be interpreted as system trajectories beginning in a prescribed initial state. However, no updates to the value function estimate or the control policy are made until after an episode terminates. In fact, Monte Carlo learning methods are closely related to repetitive or *iterative learning control (ILC)* (Moore 1993). They do not learn in real time along a trajectory, but learn as trajectories are repeated.

11.4 TEMPORAL DIFFERENCE LEARNING AND OPTIMAL ADAPTIVE CONTROL

The temporal difference (TD) method (Sutton and Barto 1998) for solving Bellman equations leads to a family of optimal adaptive controllers—that is, adaptive controllers that learn online the solutions to optimal control problems without knowing the full system dynamics. TD learning is true online reinforcement learning, wherein control actions are improved in real time based on estimating their value functions by observing data measured along the system trajectories. In the context of TD learning, the interpretation of the Bellman equation is shown in Figure 11.2-2, where $V^\pi(x)$ may be considered a predicted performance, $\sum_u \pi(x,u) \sum_{x'} P^u_{xx'} R^u_{xx'}$, the observed one-step reward, and $V^\pi(x')$ a current estimate of future behavior.

Temporal Difference Learning along State Trajectories

PI requires solution at each step of N linear equations (11.3-4). VI requires performing the recursion (11.3-10) at each step. Temporal difference RL methods are based on the Bellman equation, and solve equations such as (11.3-4) and (11.3-10) without using systems dynamics knowledge, but using and data observed along a single trajectory of the system. This makes them extremely applicable for feedback control applications. TD updates the value at each time step as observations of data are made along a trajectory. Periodically, the new value is used to update the policy. TD methods are related to adaptive control in that they adjust values and actions online in real time along system trajectories.

TD methods can be considered to be stochastic approximation techniques, whereby the Bellman equation (11.2-17), or its variants (11.3-4), (11.3-10), is replaced by its evaluation along a single sample path of the MDP. This turns the Bellman equation, into a deterministic equation, which allows the definition of a so-called *temporal difference error*.

Equation (11.2-9) was used to write the Bellman equation (11.2-17) for the infinite-horizon value (11.2-16). According to (11.2-7)–(11.2-9), an alternative form for the Bellman equation is

$$V^\pi(x_k) = E_\pi\{r_k | x_k\} + \gamma E_\pi\{V^\pi(x_{k+1}) | x_k\}. \qquad (11.4\text{-}1)$$

This equation forms the basis for TD learning.

Temporal difference RL uses one sample path, namely the current system trajectory, to update the value. That is (11.4-1) is replaced by the deterministic Bellman equation

$$V^\pi(x_k) = r_k + \gamma V^\pi(x_{k+1}), \qquad (11.4\text{-}2)$$

which holds for each observed data experience set (x_k, x_{k+1}, r_k) at each time stage k. This set consists of the current state x_k, the observed cost incurred r_k, and the next state x_{k+1}. The temporal difference error is defined as

$$e_k = -V^{\pi}(x_k) + r_k + \gamma V^{\pi}(x_{k+1}), \tag{11.4-3}$$

and the value estimate is updated to make the temporal difference error small.

The Bellman equation can be interpreted as a consistency equation, which holds if the current estimate for the value $V^{\pi}(x_k)$ is correct. Therefore, TD methods update the value estimate $\hat{V}^{\pi}(x_k)$ to make the TD error small. The idea is that if the deterministic version of Bellman's equation is used repeatedly, then on average one will converge toward the solution of the stochastic Bellman equation.

11.5 OPTIMAL ADAPTIVE CONTROL FOR DISCRETE-TIME SYSTEMS

A family of optimal adaptive control algorithms can now be developed for dynamical systems. Physical analysis of dynamical systems using Lagrangian mechanics, Hamiltonian mechanics, etc. produces system descriptions in terms of nonlinear ordinary differential equations. Discretization yields nonlinear difference equations. The bulk of research in RL has been conducted for systems that operate in discrete time (DT). Therefore, we cover DT dynamical systems first, then continuous-time systems.

TD learning is a stochastic approximation technique based on the deterministic Bellman's equation (11.4-2). Therefore, we lose little by considering deterministic systems here. Consider a class of discrete-time systems described by deterministic nonlinear dynamics in the affine state space difference equation form

$$x_{k+1} = f(x_k) + g(x_k)u_k, \tag{11.5-1}$$

with state $x_k \in R^n$ and control input $u_k \in R^m$. We use this form because its analysis is convenient. The following development can be generalized to the general sampled-data form $x_{k+1} = F(x_k, u_k)$.

A control policy is defined as a function from state space to control space $h(\cdot)$: $R^n \to R^m$. That is, for every state x_k, the policy defines a control action

$$u_k = h(x_k). \tag{11.5-2}$$

That is, a policy is simply a feedback controller.

Define a cost function that yields the value function

$$V^h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i) = \sum_{i=k}^{\infty} \gamma^{i-k} \left(Q(x_i) + u_i^{\mathrm{T}} R u_i \right), \tag{11.5-3}$$

with $0 < \gamma \leq 1$ a discount factor, $Q(x_k) > 0$, $R > 0$, and $u_k = h(x_k)$ a prescribed feedback control policy. The stage cost

$$r(x_k, u_k) = Q(x_k) + u_k^{\mathrm{T}} R u_k. \tag{11.5-4}$$

is taken to be quadratic in u_k to simplify developments, but can be any positive definite function of the control. We assume the system is stabilizable on some set $\Omega \in R^n$; that is, there exists a control policy $u_k = h(x_k)$ such that the closed-loop system $x_{k+1} = f(x_k) + g(x_k)h(x_k)$ is asymptotically stable on Ω. A control policy $u_k = h(x_k)$ is said to be *admissible* if it is stabilizing and yields a finite cost $V^h(x_k)$ for trajectories in Ω.

For the deterministic DT system, the optimal value is given by Bellman's optimality equation

$$V^*(x_k) = \min_{h(\cdot)} \left(r(x_k, h(x_k)) + \gamma V^*(x_{k+1}) \right). \tag{11.5-5}$$

This is just the discrete-time Hamilton-Jacobi-Bellman (HJB) equation. One then has the optimal policy as

$$h^*(x_k) = \arg\min_{h(\cdot)} \left(r(x_k, h(x_k)) + \gamma V^*(x_{k+1}) \right). \tag{11.5-6}$$

In this setup, the deterministic Bellman's equation (11.4-2) is

$$V^h(x_k) = r(x_k, u_k) + \gamma V^h(x_{k+1}) = Q(x_k) + u_k^{\mathrm{T}} R u_k + \gamma V^h(x_{k+1}), \; V^h(0) = 0. \tag{11.5-7}$$

This is nothing but a difference equation equivalent of the value (11.5-3). That is, instead of evaluating the infinite sum (11.5-3), one can solve the difference equation (11.5-7), with boundary condition $V(0) = 0$, to obtain the value of using a current policy $u_k = h(x_k)$.

The DT Hamiltonian function is

$$H(x_k, h(x_K), \Delta V_k) = r(x_k, h(x_k)) + \gamma V^h(x_{k+1}) - V^h(x_k), \tag{11.5-8}$$

where $\Delta V_k = \gamma V_h(x_{k+1}) - V_h(x_k)$ is the forward difference operator. The Hamiltonian function captures the energy content along the trajectories of a system. In fact, the Hamiltonian is the temporal difference error (11.4-3). The Bellman equation requires that the Hamiltonian be equal to zero for the value associated with a prescribed policy.

For the DT linear quadratic regulator (DT LQR) case,

$$x_{k+1} = A x_k + B u_k, \tag{11.5-9}$$

$$V^h(x_k) = \frac{1}{2} \sum_{i=k}^{\infty} \gamma^{i-k} \left(x_i^{\mathrm{T}} Q x_i + u_i^{\mathrm{T}} R u_i \right), \tag{11.5-10}$$

and the Bellman equation is written in several ways, as seen in Example 11.2-1.

Policy Iteration and Value Iteration Using Temporal Difference Learning

It has been seen that two forms of reinforcement learning can be based on policy iteration and value iteration. For TD learning, PI is written as follows in terms of the deterministic Bellman equation.

POLICY ITERATION USING TD LEARNING

Initialize.

Select any admissible control policy $h_0(x_k)$. Do for $j = 0$ until convergence—

Policy evaluation:

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1}). \tag{11.5-11}$$

Policy improvement:

$$h_{j+1}(x_k) = \arg\min_{h(.)} \left(r(x_k, h(x_k)) + \gamma V_{j+1}(x_{k+1}) \right), \tag{11.5-12}$$

or equivalently

$$h_{j+1}(x_k) = -\frac{\gamma}{2} R^{-1} g^{\mathrm{T}}(x_k) \nabla V_{j+1}(x_{k+1}), \tag{11.5-13}$$

where $\nabla V(x) = \partial V(x)/\partial x$ is the gradient of the value function, interpreted here as a column vector.

VI is similar, but the policy evaluation procedure is performed as follows.

VALUE ITERATION USING TD LEARNING

Value update step:

Update the value using

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_j(x_{k+1}). \tag{11.5-14}$$

Also, in VI we may select any initial control policy $h_0(x_k)$, not necessarily admissible or stabilizing.

It has been shown in Example 11.2-1 that for DT LQR the Bellman equation (11.5-7) is nothing but a linear Lyapunov equation and that (11.5-5) is the DT algebraic Riccati equation (ARE). In Example 11.3-4 it was seen that for the DT LQR the policy evaluation step (11.5-11) in PI is a Lyapunov equation and PI exactly corresponds to Hewer's algorithm (Hewer 1971) for solving the DT ARE. Hewer proved that it converges under stabilizability and detectability assumptions. For DT LQR, VI is a Lyapunov recursion, which has been shown to converge to the solution to the DT ARE under the stated assumptions by (Lancaster and Rodman 1995).

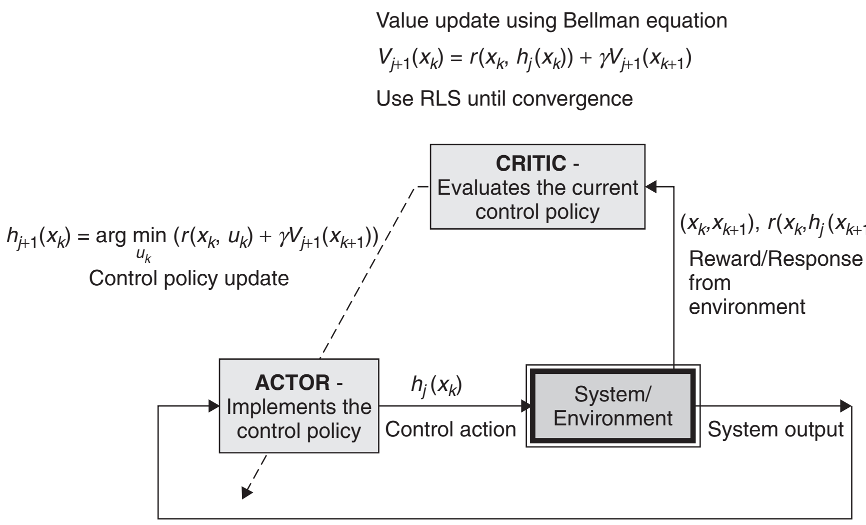
Value update using Bellman equation
$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$
Use RLS until convergence
CRITIC - Evaluates the current control policy
$h_{j+1}(x_k) = \arg\min_{u_k} (r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$
Control policy update
$(x_k, x_{k+1}), r(x_k, h_j(x_{k+1})))$
Reward/Response from environment
ACTOR - Implements the control policy
$h_j(x_k)$
Control action
System/ Environment
System output

FIGURE 11.5-1 Temporal difference learning using policy iteration. At each time one observes the current state, the next state, and the cost incurred. This is used to update the value estimate. Based on the new value, the action is updated.

The online implementation of PI using temporal difference learning is shown in Figure 11.5-1. PI and VI using TD learning have an actor–critic structure, as shown in Figure 11.1-1. The critic evaluates the current policy by solving the Bellman equation (11.5-11). Then, the actor updates the policy using (11.5-12).

Value Function Approximation

PI and VI can be implemented for finite MDP by storing and updating look-up tables. The key to practical schemes for implementing PI and VI online for dynamical systems with infinite state and action spaces is to approximate the value function by a suitable approximator structure in terms of unknown parameters. Then, the unknown parameters are tuned online exactly as in system identification. This idea of *value function approximation (VFA)* was used by Werbos (1989, 1991, 1992) and called approximate dynamic programming (ADP) or adaptive dynamic programming. It was used by Bertsekas and Tsitsiklis (1996) and called neuro-dynamic programming (NDP). See Powell (2007) and Busoniu et al. (2009).

In the LQR case it is known that the value is quadratic in the state so that

$$V(x_k) = \tfrac{1}{2}x_k^{\mathrm{T}}Px_k = \tfrac{1}{2}(\mathrm{vec}(P))^{\mathrm{T}}(x_k \otimes x_k) \equiv \overline{p}^{\mathrm{T}}\overline{x}_k \equiv \overline{p}^{\mathrm{T}}\phi(x_k), \qquad (11.5\text{-}15)$$

for some kernel matrix P. The Kronecker product $\otimes$ (Brewer 1978) allows us to write this quadratic form as linear in the parameter vector $\overline{p} = \mathrm{vec}(P)$, which is formed by stacking the columns of the P matrix. The vector $\phi(x_k) = \overline{x}_k = x_k \otimes$

x_k is the quadratic polynomial vector containing all possible pairwise products of the n components of x_k. Noting that P is symmetric and has only $n(n+1)/2$ independent elements, we remove the redundant terms in $x_k \otimes x_k$ to define a quadratic basis set $\phi(x_k)$ with $n(n+1)/2$ independent elements.

For general nonlinear systems (11.5-1) the value function contains higher-order nonlinearities. Then, we assume the Bellman equation (11.5-7) has a local smooth solution (van der Schaft 1992). According to the Weierstrass higher-order approximation theorem, there exists a dense basis set $\{\varphi_i(x)\}$ such that

$$V(x) = \sum_{i=1}^{\infty} w_i \varphi_i(x) = \sum_{i=1}^{L} w_i \varphi_i(x) + \sum_{i=L+1}^{\infty} w_i \varphi_i(x) \equiv W^{\mathrm{T}} \phi(x) + \varepsilon_L(x), \tag{11.5-16}$$

where basis vector $\phi(x) = \begin{bmatrix} \varphi_1(x) & \varphi_2(x) & \cdots & \varphi_L(x) \end{bmatrix} : R^n \to R^L$, and $\varepsilon_L(x)$ converges uniformly to zero as the number of terms retained $L \to \infty$. In the Weierstrass Theorem, standard usage takes a polynomial basis set. In the neural network (NN) community, approximation results have been shown for various other basis sets including sigmoid, hyperbolic tangent, Gaussian radial basis functions, etc. There, standard results show that the NN approximation error $\varepsilon_L(x)$ is bounded by a constant on a compact set. L is referred to as the number of hidden-layer neurons, $\varphi_i(x)$ as the NN activation functions, and w_i as the NN weights.

Optimal Adaptive Control Algorithms for DT Systems

We are now in a position to present several adaptive control algorithms based on TD RL that converge online to the optimal control solution.

The parameters in $\overline{p}$ or W are unknown. Substituting the value function approximation into the value update (11.5-11) in PI we obtain the following algorithm.

OPTIMAL ADAPTIVE CONTROL USING A POLICY ITERATION ALGORITHM

Initialize.

Select any admissible control policy $h_0(x_k)$. Do for $j = 0$ until convergence—

Policy evaluation step:

Determine the least-squares solution W_{j+1} to

$$W_{j+1}^{\mathrm{T}} \left(\phi(x_k) - \gamma \phi(x_{k+1})\right) = r(x_k, h_j(x_k)) = Q(x_k) + h_j^{\mathrm{T}}(x_k) R h_j(x_k). \tag{11.5-17}$$

Policy improvement step:

Determine an improved policy using

$$h_{j+1}(x_k) = -\frac{\gamma}{2} R^{-1} g^{\mathrm{T}}(x_k) \nabla \phi^{\mathrm{T}}(x_{k+1}) W_{j+1}. \tag{11.5-18}$$

This algorithm is easily implemented online by standard system identification techniques (Ljung 1999). In fact, note that (11.5-17) is a scalar equation, whereas the unknown parameter vector $W_{j+1} \in R^L$ has L elements. Therefore, data from multiple time steps is needed for its solution. At time $k+1$ one measures the previous state x_k, the control $u_k = h_j(x_k)$, the next state x_{k+1}, and computes the resulting utility $r(x_k, h_j(x_k))$. This gives one scalar equation. This is repeated for subsequent times using the same policy $h_j(\cdot)$ until we have at least L equations, at which point we may determine the LS solution W_{j+1}. We may use batch LS for this.

Alternatively, note that equations of the form (11.5-17) are exactly those solved by recursive least-squares (RLS) techniques (Ljung 1999) Therefore, we may run RLS online until convergence. Write (11.5-17) as

$$W_{j+1}^{\mathrm{T}}\Phi(k) \equiv W_{j+1}^{\mathrm{T}}\left(\phi(x_k) - \gamma\phi(x_{k+1})\right) = r(x_k, h_j(x_k)), \qquad (11.5\text{-}19)$$

with $\Phi(k) \equiv (\phi(x_k) - \gamma\phi(x_{k+1}))$ a regression vector. At step j of the PI algorithm, one fixes the control policy at $u = h_j(x)$. Then, at each time k one measures the data set $\left(x_k, x_{k+1}, r(x_k, h_j(x_k))\right)$. One step of RLS is then performed. This is repeated for subsequent times until convergence to the parameters corresponding to the value $V_{j+1}(x) = W_{j+1}^{\mathrm{T}}\phi(x)$.

Note that for RLS to converge, the regression vector $\Phi(k) \equiv (\phi(x_k) - \gamma\phi(x_{k+1}))$ must be persistently exciting.

As an alternative to RLS, we could use a gradient descent tuning method such as

$$W_{j+1}^{i+1} = W_{j+1}^{i} - \alpha\Phi(k)\left(\left(W_{j+1}^{i}\right)^{\mathrm{T}}\Phi(k) - r(x_k, h_j(x_k))\right), \qquad (11.5\text{-}20)$$

with $\alpha > 0$ a tuning parameter. The step index j is held fixed, and index i is incremented at each increment of the time index k. Note that the quantity inside the large brackets is just the temporal difference error.

Once the value parameters have converged, the control policy is updated according to (11.5-18). Then, the procedure is repeated for step $j+1$. This entire procedure is repeated until convergence to the optimal control solution.

This provides an online reinforcement learning algorithm for solving the optimal control problem using policy iteration by measuring data along the system trajectories. Likewise, an online reinforcement learning algorithm can be given based on value iteration. Substituting the value function approximation into the value update (11.5-14) in VI we obtain the following algorithm.

OPTIMAL ADAPTIVE CONTROL USING A VALUE ITERATION ALGORITHM

Initialize

Select any control policy $h_0(x_k)$, not necessarily admissible or stabilizing. Do for $j = 0$ until convergence—

Value update step:

Determine the least-squares solution W_{j+1} to

$$W_{j+1}^{\mathrm{T}}\phi(x_k) = r(x_k, h_j(x_k)) + W_j^{\mathrm{T}}\gamma\phi(x_{k+1}). \tag{11.5-21}$$

Policy improvement step:

Determine an improved policy using (11.5-18).

To solve (11.5-21) in real-time we can use batch LS, RLS, or gradient-based methods based on data $\left(x_k, x_{k+1}, r(x_k, h_j(x_k))\right)$ measured at each time along the system trajectories. Then the policy is improved using (11.5-18).

Note that the old weight parameters are on the right-hand side of (11.5-21). Thus, the regression vector is now $\phi(x_k)$, which must be persistently exciting for convergence of RLS.

Online Solution of Lyapunov and Riccati Equations

It is important to note that the PI and VI adaptive optimal control algorithms just given actually solve the Bellman equation (11.5-7) and the HJB equation (11.5-5) online in real time by using data measured along the system trajectories. The system drift function $f(x_k)$ (or the A matrix in the LQR case) is not needed in these algorithms. That is, these algorithms solve the Riccati equation online in real time without knowledge of the A matrix. The online implementation of PI is shown in Figure 11.5-1.

According to Example 11.3-4, for DT LQR policy iteration, this means that the Lyapunov equation

$$0 = (A - BK^j)^{\mathrm{T}} P^{j+1}(A - BK^j) - P^{j+1} + Q + (K^j)^{\mathrm{T}} RK^j, \tag{11.5-22}$$

has been replaced by (11.5-17) or

$$\overline{p}_{j+1}^{\mathrm{T}}\,(\overline{x}_{k+1} - \overline{x}_k) = r(x_k, h_j(k)) = x_k^{\mathrm{T}}(Q + K_j^{\mathrm{T}} RK_j)x_k, \tag{11.5-23}$$

which is solved for the parameters $\overline{p}_{j+1} = \mathrm{vec}(P^{j+1})$ using, for instance, RLS by measuring the data set $\left(x_k, x_{k+1}, r(x_k, h_j(x_k))\right)$ at each time. For this step the dynamics (A, B) can be unknown, as they are not needed. For DT LQR value iteration, the Lyapunov recursion

$$P^{j+1} = (A - BK^j)^{\mathrm{T}} P^j (A - BK^j) + Q + (K^j)^{\mathrm{T}} RK^j \tag{11.5-24}$$

has been replaced by (11.5-21) or

$$\overline{p}_{j+1}^{\mathrm{T}}\overline{x}_{k+1} = r(x_k, h_j(k)) + \overline{p}_j^{\mathrm{T}}\overline{x}_k = x_k^{\mathrm{T}}(Q + K_j^{\mathrm{T}} RK_j)x_k + \overline{p}_j^{\mathrm{T}}\overline{x}_k, \tag{11.5-25}$$

which may be solved for the parameters $\overline{p}_{j+1} = \mathrm{vec}(P_{j+1})$ using RLS without knowing *A,B*.

Introduction of a Second "Actor" Neural Network

Using value function approximation allows standard system identification techniques to be used to find the value function parameters that approximately solve the Bellman equation. The approximator structure just described that is used for approximation of the value function is known as the critic NN (neural network), as it determines the value of using the current policy. Using VFA, the PI and VI reinforcement learning algorithm solve the Bellman equation during the value update portion of each iteration step j by observing only the data set $\left(x_k, x_{k+1}, r(x_k, h_j(x_k))\right)$ at each time along the system trajectory and solving (11.5-17) or (11.5-21).

However, according to Example 11.3-4, in the LQR case, the policy update (11.5-18) is given by

$$K^{j+1} = -(B^{\mathrm{T}} P^{j+1} B + R)^{-1} B^{\mathrm{T}} P^{j+1} A, \tag{11.5-26}$$

which requires full knowledge of the dynamics (A, B). Note further that the embodiment (11.5-18) cannot easily be implemented in the nonlinear case because it is implicit in the control, since x_{k+1} depends on u_k and is the argument of a nonlinear activation function.

These problems are both solved by introducing a *second neural network* for the control policy, known as the actor NN (Werbos, 1989, 1991, 1992). Introduce a parametric approximator structure for the control action

$$u_k = h(x_k) = U^{\mathrm{T}} \sigma(x_k), \tag{11.5-27}$$

with $\sigma(x)$: $R^n \rightarrow R^M$ a vector of M activation or basis functions and $U \in R^{M \times m}$ a matrix of weights or unknown parameters. After convergence of the critic NN parameters to W_{j+1} in PI or VI, it is required to perform the policy update (11.5-18). To achieve this we may use a gradient descent method for tuning the actor weights U such as

$$U_{j+1}^{i+1} = U_{j+1}^{i} - \beta\sigma(x_k)\left(2R\left(U_{j+1}^{i}\right)^{\mathrm{T}} \sigma(x_k) + \gamma g(x_k)^{\mathrm{T}} \nabla\phi^{\mathrm{T}}(x_{k+1}) W_{j+1}\right)^{\mathrm{T}}, \tag{11.5-28}$$

with $\beta > 0$ a tuning parameter. The tuning index i may be incremented with the time index k. On convergence, set the updated policy to $h_{j+1}(x_k) = (U_{j+1}^{i})^{\mathrm{T}} \sigma(x_k)$.

Several items are worthy of note at this point. First, the tuning of the actor NN requires observations at each time k of the data set (x_k, x_{k+1}), that is, the current state and the next state. However, as per the formulation (11.5-27), the actor NN yields the control u_k at time k in terms of the state x_k at time k. The next state x_{k+1} is not needed in (11.5-27). Thus, after (11.5-28) has converged, (11.5-27) is a legitimate feedback controller. Second, in the LQR case, the actor NN (11.5-27) embodies the feedback gain computation (11.5-26). This is highly intriguing, for the latter contains the state internal dynamics A, but the former does not. This means that the A matrix is not needed to compute the feedback control. The reason is that the actor NN has learned information about A in its weights, since (x_k, x_{k+1}) are used in its tuning.

Finally, note that only the input function $g(\cdot)$ (in the LQR case, the B matrix) is needed in (11.5-28) to tune the actor NN. Thus, introducing a second actor NN has completely avoided the need for knowledge of the state drift dynamics $f(\cdot)$ (or A in the LQR case).

Example 11.5-1. Discrete-time Optimal Adaptive Control of Power System Using Value Iteration

In this simulation we show how to use DT value iteration to solve the DT ARE online without knowing the system matrix A. We simulated the online VI algorithm (11.5-21), (11.5-28) for load frequency control of an electric power system. Power systems are complex nonlinear systems. However, during normal operation the system load, which gives the nonlinearity, has only small variations. As such, a linear model can be used to represent the system dynamics around an operating point specified by a constant load value. A problem rises from the fact that in an actual plant the parameter values are not precisely known, as reflected in an unknown system A matrix, yet an optimal control solution is sought.

The model of the system that is considered here is $\dot{x} = Ax + Bu$, where

$$A = \begin{bmatrix} -1/T_p & K_p/T_p & 0 & 0 \\ 0 & -1/T_T & 1/T_T & 0 \\ -1/RT_G & 0 & -1/T_G & -1/T_G \\ K_E & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1/T_G \\ 0 \end{bmatrix}.$$

The system state is $x(t) = \begin{bmatrix} \Delta f(t) & \Delta P_g(t) & \Delta X_g(t) & \Delta E(t) \end{bmatrix}^T$, where $\Delta f(t)$ is the incremental frequency deviation (Hz), $\Delta P_g(t)$ is the incremental change in generator output (p.u. MW), $\Delta X_g(t)$ is the incremental change in governor position (p.u. MW), and $\Delta E(t)$ is the incremental change in integral control. The system parameters are T_G, the governor time constant; T_T, turbine time constant; T_P, plant model time constant; K_P, planet model gain; R, speed regulation due to governor action; and K_E, integral control gain.

The values of the CT system parameters were randomly picked within specified operating ranges so that

$$A = \begin{bmatrix} -0.0665 & 8 & 0 & 0 \\ 0 & -3.663 & 3.663 & 0 \\ -6.86 & 0 & -13.736 & -13.736 \\ 0.6 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 13.7355 & 0 \end{bmatrix}.$$

The discrete-time dynamics was obtained using the zero-order hold method with a sampling period of $T = 0.01$ sec. The solution to the DT ARE with cost function weights $Q = I$, $R = I$, and $\gamma = 1$ is

$$P_{\text{DARE}} = \begin{bmatrix} 0.4750 & 0.4766 & 0.0601 & 0.4751 \\ 0.4766 & 0.7831 & 0.1237 & 0.3829 \\ 0.0601 & 0.1237 & 0.0513 & 0.0298 \\ 0.4751 & 0.3829 & 0.0298 & 2.3370 \end{bmatrix}.$$

In this simulation, only the time constant T_G of the governor, which appears in the B matrix, is considered to be known, while the values for all the other parameters appearing in the system A matrix are not known. That is, the A matrix is needed only to simulate the system and obtain the data and is not needed by the control algorithm.

For the DT LQR, the value is quadratic in the states $V(x) = \frac{1}{2}x^{\mathrm{T}}Px$, as in (11.5-15). Therefore, the basis functions for the critic NN in (11.5-16) are selected as the quadratic polynomial vector in the state components. Since there are $n = 4$ states, this vector has $n(n+1)/2 = 10$ components. The control is linear in the states $u = -Kx$, so the basis functions for the actor NN (11.5-27) are taken as the state components.

The online implementation of VI may be done by setting up a batch least-squares problem to solve for the 10 critic NN parameters, that is the Riccati solution entries $\overline{p}_{j+1} \equiv W_{j+1}$ in (11.5-21), for each step j. In this simulation the matrix P^{j+1} is determined after collecting 15 points of data $(x_k, x_{k+1}, r(x_k, u_k))$ for each least-squares problem. Therefore, a least-squares problem for the critic weights is solved each 0.15 sec. Then the actor NN parameters (that is, the feedback gain matrix entries) are updated using (11.5-28). The simulations were performed over a time interval of 60 sec.

The system states trajectories are given in Figure 11.5-2, which shows that they are regulated to zero as desired. The convergence of the Riccati matrix parameters is shown in Figure 11.5-3. The final values of the critic NN parameter estimates are

$$P_{\text{critic NN}} = \begin{bmatrix} 0.4802 & 0.4768 & 0.0603 & 0.4754 \\ 0.4768 & 0.7887 & 0.1239 & 0.3834 \\ 0.0603 & 0.1239 & 0.0567 & 0.0300 \\ 0.4754 & 0.3843 & 0.0300 & 2.3433 \end{bmatrix}.$$

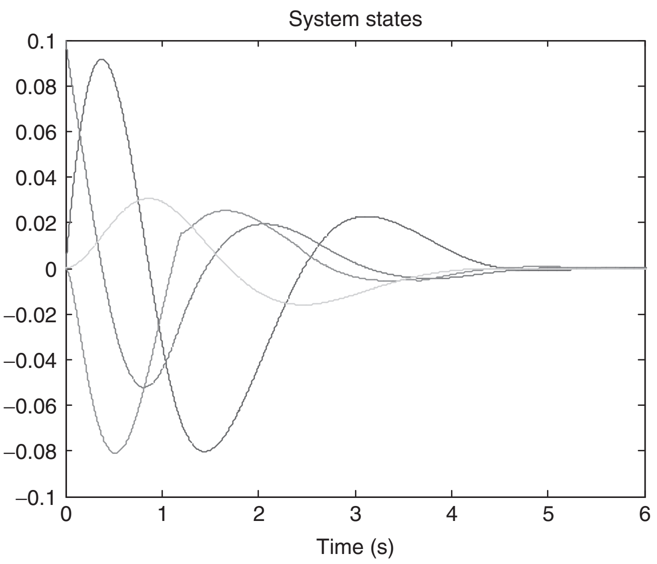

FIGURE 11.5-2 System states during the first 6 sec. This figure shows that even though the A matrix of the power system is unknown, the adaptive controller based on value iteration keeps the states stable and regulates them to zero.

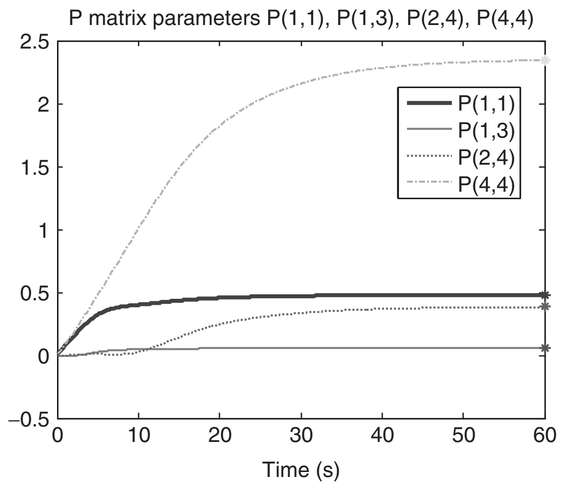

FIGURE 11.5-3 Convergence of selected ARE solution parameters. This figure shows that the adaptive controller based on VI converges to the Riccati-equation solution in real time without knowing the system A matrix.

The optimal adaptive control VI algorithm has converged to the optimal control solution, as given by the ARE solution. This has been accomplished in real time without knowing the system A matrix. ■

Actor–Critic Implementation of DT Optimal Adaptive Control

Two algorithms for optimal adaptive control of DT systems based on RL have been given. A PI algorithm is implemented by solving (11.5-17) using RLS or batch LS to determine the value of the current policy, then the policy is updated by running (11.5-28). A VI algorithm is implemented by solving (11.5-21) using RLS or batch LS to determine the value of the current policy, then the policy is updated by running (11.5-28).

The implementation of reinforcement learning using two NNs, one as a critic and one as an actor, yields the actor–critic RL structure shown in Figure 11.1-1. In this control system, the critic and the actor are tuned online using the observed data $\left(x_k, x_{k+1}, r(x_k, h_j(x_k))\right)$ along the system trajectory. The critic and actor are tuned sequentially in both the PI and the VI algorithms. That is, the weights of one NN are held constant while the weights of the other are tuned until convergence. This procedure is repeated until both NN have converged. Then, the controller has learned the optimal controller online. Thus, this is an online adaptive optimal control system wherein the value function parameters are tuned online and the convergence is to the optimal value and control. The convergence of value iteration using two NN for the DT nonlinear system (11.5-1) was proven in Al-Tamimi et al. (2008).

Q Learning for Optimal Adaptive Control

The Q learning RL method gives an adaptive control algorithm that converges online to the optimal control solution for completely unknown systems. That is, it solves the Bellman equation (11.5-7) and the HJB equation (11.5-5) online in real time by using data measured along the system trajectories, without any knowledge of the dynamics $f(x_k)$, $g(x_k)$.

Q learning is a simple method for reinforcement learning that works for unknown MDP, that is, for systems with completely unknown dynamics. It was developed by Watkins (1989) and Watkins and Dayan (1992) and Werbos (1989, 1991, 1992), who called it action-dependent heuristic dynamic programming (ADHDP), since the Q function depends on the control input. Q learning learns the Q function (11.3-38) using TD methods by performing an action u_k and measuring at each time stage the resulting data experience set (x_k, x_{k+1}, r_k) consisting of the current state, the next state, and the resulting stage cost. Writing the Q function Bellman equation (11.3-40) along a sample path gives

$$Q^\pi(x_k, u_k) = r(x_k, u_k) + \gamma Q^\pi(x_{k+1}, h(x_{k+1})), \tag{11.5-29}$$

which defines a TD error

$$e_k = -Q^\pi(x_k, u_k) + r(x_k, u_k) + \gamma Q^\pi(x_{k+1}, h(x_{k+1})). \tag{11.5-30}$$

The VI algorithm for Q function is given as (11.3-47). Based on this, the Q function is updated using the algorithm

$$Q_k(x_k, u_k) = Q_{k-1}(x_k, u_k) + \alpha_k\Big[r(x_k, u_k) + \gamma \min_u Q_{k-1}(x_{k+1}, u) - Q_{k-1}(x_k, u_k)\Big]. \tag{11.5-31}$$

This Q learning algorithm is similar to stochastic approximation methods of adaptive control or parameter estimation used in control systems. It was developed for finite MDP and the convergence proven by Watkins (1989) using stochastic approximation methods. It was shown that the algorithm converges for finite MDP provided that all state–action pairs are visited infinitely often and

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty, \tag{11.5-32}$$

that is, standard stochastic approximation conditions. On convergence, the TD error is (approximately) equal to zero. For finite MDP, Q learning requires storing a 2-D lookup table in terms of all the states x and actions u.

The requirement that all state–action pairs are visited infinitely often translates to the problem of maintaining sufficient exploration during learning. Let us now derive methods for Q learning for dynamical systems that yield adaptive control algorithms that converge to optimal control solutions. PI and VI algorithms have been given using the Q function in (11.3-43)–(11.3-47). A Q learning algorithm is easily developed for DT dynamical systems using Q function approximation (Werbos 1989, 1991, 1992; Bradtke et al. 1994). It was shown in Example 11.3-5 that for DT LQR the Q function is a quadratic form in terms of $z_k \equiv [x_k^{\mathrm{T}} \quad u_k^{\mathrm{T}}]^{\mathrm{T}} \in R^{n+m}$. Assume, therefore, that for nonlinear systems the Q function is parameterized as

$$Q(x,u) = W^{\mathrm{T}}\phi(z),$$

for some unknown parameter vector W and basis set vector $\phi(z)$. For the DT LQR, $\phi(z)$ is the quadratic basis set formed from the state and input components. Substituting the Q function approximation into the TD error (11.5-30) yields

$$e_k = -W^{\mathrm{T}}\phi(z_k) + r(x_k,u_k) + \gamma W^{\mathrm{T}}\phi(z_{k+1}), \tag{11.5-33}$$

upon which either PI or VI algorithms can be based. Considering the PI algorithm (11.3-43), (11.3-44) yields the Q function evaluation step

$$W_{j+1}^{\mathrm{T}}\left(\phi(z_k) - \gamma\phi(z_{k+1})\right) = r(x_k, h_j(x_k)) \tag{11.5-34}$$

and the policy improvement step

$$h_{j+1}(x_k) = \arg\min_u \left(W_{j+1}^{\mathrm{T}}\phi(x_k,u)\right), \quad \text{for all } x \in X. \tag{11.5-35}$$

Q learning using VI (11.3-45) is given by

$$W_{j+1}^{\mathrm{T}}\phi(z_k) = r(x_k, h_j(x_k)) + \gamma W_j^{\mathrm{T}}\phi(z_{k+1}), \tag{11.5-36}$$

and (11.5-35). These equations do not require knowledge of the dynamics $f(\cdot), g(\cdot)$.

For online implementation, batch LS or RLS can be used to solve (11.5-34) for the parameter vector W_{j+1} given the regression vector $(\phi(z_k) - \gamma\phi(z_{k+1}))$, or (11.5-36) using regression vector $\phi(z_k)$. The observed data at each time instant is $(z_k, z_{k+1}, r(x_k,u_k))$ with $z_k \equiv [x_k^{\mathrm{T}} \; u_k^{\mathrm{T}}]^{\mathrm{T}}$. We take $z_{k+1} \equiv [x_{k+1}^{\mathrm{T}} \; u_{k+1}^{\mathrm{T}}]^{\mathrm{T}}$, with $u_{k+1} = h_j(x_{k+1})$ and $h_j(\cdot)$ the current policy. Probing noise must be added to the control input to obtain persistence of excitation.

After convergence of the Q function parameters, the action update (11.5-35) is performed. This is easily accomplished without knowing the system dynamics due to the fact that the Q function contains u_k as an argument so that $\partial\left(W_{j+1}^{\mathrm{T}}\phi(x_k,u)\right)/\partial u$ can be explicitly computed. In fact,

$$\frac{\partial Q(x,u)}{\partial u} = \left(\frac{\partial z}{\partial u}\right)^{\mathrm{T}}\left(\frac{\partial\phi(z)}{\partial z}\right)^{\mathrm{T}} W = \begin{bmatrix}0_{m,n} & I_m\end{bmatrix}\nabla\phi^{\mathrm{T}}W, \tag{11.5-37}$$

where $0_{m,n} \in R^{m\times n}$ is a matrix of zeros. For the specific case of the DT LQR, for instance, from Example 11.3-5 we have

$$Q(x,u) \equiv \frac{1}{2}\begin{bmatrix} x \\ u \end{bmatrix}^{\mathrm{T}} S \begin{bmatrix} x \\ u \end{bmatrix} = \frac{1}{2}z^{\mathrm{T}}Sz = \frac{1}{2}\mathrm{vec}^{\mathrm{T}}(S)(z\otimes z) \equiv W^{\mathrm{T}}\phi(z), \quad (11.5\text{-}38)$$

with $\otimes$ the Kronecker product (Brewer 1978) and $\mathrm{vec}(S) \in R^{(n+m)^2}$ the vector formed by stacking the columns of the S matrix. The basis vector $\phi(z) = z \otimes z \in R^{(n+m)^2}$ is the quadratic polynomial vector containing all possible pairwise products of the $n+m$ components of z. Define $N = n+m$. Then,

$$\nabla\phi^{\mathrm{T}} = \frac{\partial\phi^{\mathrm{T}}}{\partial z} = (I_N \otimes z + z \otimes I_N)^{\mathrm{T}} \in R^{N\times N^2} \quad (11.5\text{-}39)$$

Using these equations we obtain a form equivalent to the control update (11.5-35) given in Example 11.3-5, which can be done knowing the Q function parameters S without knowing system matrices A,B.

If the control can be computed explicitly from the action update (11.5-35), as in the DT LQR case, an actor NN is not needed for Q learning and it can be implemented using only one critic NN for Q function approximation.

11.6 INTEGRAL REINFORCEMENT LEARNING FOR OPTIMAL ADAPTIVE CONTROL OF CONTINUOUS-TIME SYSTEMS

Reinforcement learning is considerably more difficult for continuous-time (CT) systems than for discrete-time systems, and its development has lagged. See (Abu-Khalaf et al. 2006) for the development of a PI method for CT systems. Using a method known as i*ntegral reinforcement learning* (IRL) (Vrabie et al. Vrabie and Lewis 2009) allows the application of RL to formulate online optimal adaptive control methods for CT systems.

Consider the continuous-time nonlinear dynamical system

$$\dot{x} = f(x) + g(x)u, \quad (11.6\text{-}1)$$

with state $x(t) \in R^n$, control input $u(t) \in R^m$, and the usual assumptions required for existence of unique solutions and an equilibrium point at $x = 0$, e.g., $f(0) = 0$ and $f(x) + g(x)u$ Lipschitz on a set $\Omega \subseteq R^n$ that contains the origin. We assume the system is stabilizable on Ω; that is, there exists a continuous control function $u(t)$ such that the closed-loop system is asymptotically stable on Ω.

Define a performance measure or cost function that has the value associated with the feedback control policy $u = \mu(x)$ given by

$$V^{\mu}(x(t)) = \int_t^{\infty} r(x(\tau), u(\tau))\, d\tau, \quad (11.6\text{-}2)$$

with utility $r(x, u) = Q(x) + u^{\mathrm{T}} Ru$, $Q(x)$ positive definite, that is, $Q(x) > 0$ for all x and $x = 0 \Rightarrow Q(x) = 0$, and $R > 0$ a positive definite matrix. For the CT linear quadratic regulator (LQR) we have

$$\dot{x} = Ax + Bu, \tag{11.6-3}$$

$$V^{\mu}(x(t)) = \frac{1}{2} \int_{t}^{\infty} (x^{\mathrm{T}} Qx + u^{\mathrm{T}} Ru)\, d\tau. \tag{11.6-4}$$

A policy is called admissible if it is continuous, stabilizes the system, and has a finite associated cost. If the cost is smooth, then an infinitesimal equivalent to (11.6-2) can be found by differentiation to be the nonlinear equation

$$0 = r(x, \mu(x)) + (\nabla V^{\mu})^{\mathrm{T}} (f(x) + g(x)\mu(x)), \quad V^{\mu}(0) = 0, \tag{11.6-5}$$

where ∇V^{μ} (a column vector) denotes the gradient of the cost function V^{μ} with respect to x. This is the CT Bellman equation. It is defined based on the CT Hamiltonian function

$$H(x, \mu(x), \nabla V^{\mu}) = r(x, \mu(x)) + (\nabla V^{\mu})^{\mathrm{T}} (f(x) + g(x)\mu(x)). \tag{11.6-6}$$

The optimal value satisfies the CT Hamilton-Jacobi-Bellman (HJB) equation (Chapter 6),

$$0 = \min_{\mu} H(x, \mu(x), \nabla V^{*}), \tag{11.6-7}$$

and the optimal control satisfies

$$\mu^{*} = \arg\min_{\mu} H(x, \mu(x), \nabla V^{*}). \tag{11.6-8}$$

We now see the problem with CT systems immediately. Compare the CT Bellman Hamiltonian (11.6-6) to the DT Hamiltonian (11.5-8). The former contains the full system dynamics $f(x) + g(x)u$, while the DT Hamiltonian does not. This means the CT Bellman equation (11.6-5) cannot be used as a basis for reinforcement learning unless the full dynamics are known.

Reinforcement learning methods based on (11.6-5) can be developed (Baird 1994, Doya 2000, Murray et al. 2001, Mehta and Meyn 2009, Hanselmann et al. 2007). These have limited use for adaptive control purposes because the system dynamics must be known. In another approach, one can use Euler's method to discretize the CT Bellman equation (Baird 1994). Noting that

$$0 = r(x, \mu(x)) + (\nabla V^{\mu})^{\mathrm{T}} (f(x) + g(x)\mu(x)) = r(x, \mu(x)) + \dot{V}^{\mu}, \tag{11.6-9}$$

We use Euler's method to discretize this to obtain

$$0 = r(x_k, u_k) + \frac{V^{\mu}(x_{k+1}) - V^{\mu}(x_k)}{T} \equiv \frac{r_S(x_k, u_k)}{T} + \frac{V^{\mu}(x_{k+1}) - V^{\mu}(x_k)}{T}, \tag{11.6-10}$$

with sample period T so that $t = kT$. The discrete sampled utility is $r_S(x_k, u_k) = r(x_k, u_k)T$, where it is important to multiply the CT utility by the sample period.

Now note that the discretized CT Bellman equation (11.6-10) has the same form as the DT Bellman equation (11.5-7). Therefore, all the reinforcement learning methods just described for DT systems can be applied.

However, this is an approximation only. An alternative exact method for CT reinforcement learning was given by Vrabie et al. (2009) and Vrabie and Lewis (2009). This is termed integral reinforcement learning (IRL). Note that we may write the cost (11.6-2) in the integral reinforcement form

$$V^{\mu}(x(t)) = \int_{t}^{t+T} r(x(\tau), u(\tau))\, d\tau + V^{\mu}(x(t+T)), \tag{11.6-11}$$

for any $T > 0$. This is exactly in the form of the DT Bellman equation (11.5-7). According to Bellman's principle, the optimal value is given in terms of this construction as (Chapter 6)

$$V^*(x(t)) = \min_{\overline{u}(t:t+T)} \left(\int_{t}^{t+T} r(x(\tau), u(\tau))\, d\tau + V^*(x(t+T)) \right),$$

where $\overline{u}(t\colon t+T) = \{u(\tau)\colon t \le \tau < t+T\}$. The optimal control is

$$\mu^*(x(t)) = \arg\min_{\overline{u}(t:t+T)} \left(\int_{t}^{t+T} r(x(\tau), u(\tau))\, d\tau + V^*(x(t+T)) \right).$$

It is shown in (Vrabie et al. 2009) that the nonlinear equation (11.6-5) is exactly equivalent to the integral reinforcement form (11.6-11). That is, the positive definite solution of both that satisfies $V(0) = 0$ is the value (11.6-2) of the policy $u = \mu(x)$. Therefore, integral reinforcement form (11.6-11) also serves as a Bellman equation for CT systems and serves is a fixed-point equation. Thus, we can define the temporal difference error for CT systems as

$$e(t\colon t+T) = \int_{t}^{t+T} r(x(\tau), u(\tau))\, d\tau + V^{\mu}(x(t+T)) - V^{\mu}(x(t)). \tag{11.6-12}$$

This does not involve the system dynamics.

Now, policy iteration and value iteration can be directly formula for CT systems. The following algorithms are termed integral reinforcement learning for CT systems (Vrabie et al. 2009). They both give optimal adaptive controllers for CT systems, that is, adaptive control algorithms that converge to optimal control solutions.

IRL OPTIMAL ADAPTIVE CONTROL USING POLICY ITERATION (PI)

Initialize

Select any admissible control policy $\mu_0(x)$. Do for $j = 0$ until convergence—

Policy evaluation step:

Solve for $V_{j+1}(x(t))$ using

$$V_{j+1}(x(t)) = \int_t^{t+T} r(x(s), \mu_j(x(s)))\, ds + V_{j+1}(x(t+T)), \quad \text{with } V_{j+1}(0) = 0. \tag{11.6-13}$$

Policy improvement step:

Determine an improved policy using

$$\mu_{j+1} = \arg\min_u [H(x, u, \nabla V_{j+1})], \tag{11.6-14}$$

which explicitly is

$$\mu_{j+1}(x) = -\tfrac{1}{2} R^{-1} g^{\mathrm{T}}(x) \nabla V_{j+1}. \tag{11.6-15}$$

IRL OPTIMAL ADAPTIVE CONTROL USING VALUE ITERATION (VI)

Initialize

Select any control policy $\mu_0(x)$, not necessarily stabilizing. Do for $j = 0$ until convergence—

Policy evaluation step:

Solve for $V_{j+1}(x(t))$ using

$$V_{j+1}(x(t)) = \int_t^{t+T} r(x(s), \mu_j(x(s)))\, ds + V_j(x(t+T)). \tag{11.6-16}$$

Policy improvement step:

Determine an improved policy using (11.6-15).

Note that neither algorithm requires knowledge about the system drift dynamics function $f(x)$. That is, they work for partially unknown systems. Convergence of PI is proved in Vrabie et al. (2009).

Online Implementation of IRL—A Hybrid Optimal Adaptive Controller

Both of these IRL algorithms may be implemented online by reinforcement learning techniques using value function approximation $V(x) = W^{\mathrm{T}}\phi(x)$ in a critic approximator network. Using VFA in the PI algorithm (11.6-13) yields

$$W_{j+1}^{\mathrm{T}}\left[\phi(x(t)) - \phi(x(t+T))\right] = \int_{t}^{t+T} r(x(s), \mu_j(x(s)))\, ds. \qquad (11.6\text{-}17)$$

Using VFA in the VI algorithm (11.6-16) yields

$$W_{j+1}^{\mathrm{T}}\phi(x(t)) = \int_{t}^{t+T} r(x(s), \mu_j(x(s)))\, ds + W_j^{\mathrm{T}}\phi(x(t+T)). \qquad (11.6\text{-}18)$$

RLS or batch LS can be used to update the value function parameters in these equations. On convergence of the value parameters, the action is updated using (11.6-15). The implementation is shown in Figure 11.6-1. This is an optimal adaptive controller, that is, an adaptive controller that measures data along the system trajectories and converges to optimal control solutions. Note that only the system input coupling dynamics $g(x)$ are needed to implement these algorithms, since it appears in action update (11.6-15). The drift dynamics $f(x)$ are not needed.

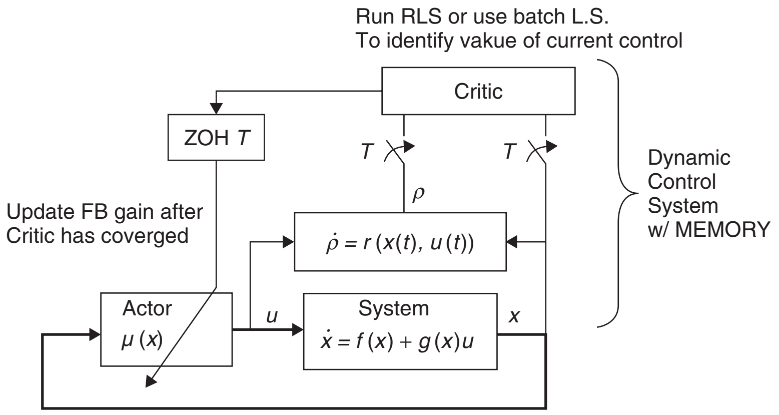

FIGURE 11.6-1 Hybrid optimal adaptive controller based on integral reinforcement learning (IRL), showing the two-time scale hybrid nature of the IRL controller. The integral reinforcement signal is added as an extra state and functions as the memory of the controller. The critic runs on a slow time scale and learns the value of using the current control policy. When the critic has converged, the actor control policy is updated to obtain an improved value.

The time is incremented at each iteration by the reinforcement learning time interval T. This time interval need not be the same at each iteration. T can be changed depending on how long it takes to get meaningful information from the observations. T is not a sample period in the standard meaning.

The measured data at each time increment is $(x(t), x(t+T), \rho(t\colon t+T))$, where

$$\rho(t\colon t+T) = \int_t^{t+T} r(x(\tau), u(\tau))\, d\tau, \tag{11.6-19}$$

is the integral reinforcement measured on each time interval. This can be implemented by introducing an integrator $\dot{\rho} = r(x(t), u(t))$, as shown in Figure 11.6-1. That is, the integral reinforcement $\rho(t)$ is added as an extra continuous-time state. It functions as the memory or controller dynamics. The remainder of the controller is a sampled data controller.

Note that the control policy $\mu(x)$ is updated periodically after the critic weights have converged to the solution of (11.6-17) or (11.6-18). Therefore, the policy is piecewise constant in time. On the other hand, the control varies continuously with the state between each policy update. IRL for CT systems is, in fact, a hybrid CT/DT adaptive controller that converges to the optimal control solution in real time without knowing the drift dynamics $f(x)$. Due to the fact that the policy update (11.6-15) for CT systems does not involve the drift dynamics $f(x)$, no actor NN is needed in IRL. Only a critic NN is needed for VFA.

Online Solution of the Algebraic Riccati Equation without Full Plant Dynamics

It can be shown that the integral reinforcement form (11.6-11) is equivalent to the nonlinear Lyapunov (11.6-5) (Vrabie et al. 2009). Thus, the IRL controller solves the Lyapunov equation online without knowing the drift dynamics $f(x)$. Moreover, it converges to the optimal control so that it solves the HJB equation (11.6-7).

In the CT LQR case (11.6-3), (11.6-4) we have linear state feedback control policies $u = -Kx$. Then, equation (11.6-5) is

$$(A - BK)^{\mathrm{T}}P + P(A - BK) + Q + K^{\mathrm{T}}RK = 0, \tag{11.6-20}$$

which is a Lyapunov equation. The HJB equation (11.6-7) becomes the CT ARE

$$A^{\mathrm{T}}P + PA + Q - PBR^{-1}B^{\mathrm{T}}P = 0. \tag{11.6-21}$$

Thus, IRL solves both the Lyapunov equation and the ARE online in real time, using data measured along the system trajectories, without knowing the A matrix.

For the CT LQR, (11.6-13) is equivalent to a Lyapunov equation at each step, so that policy iteration is exactly the same as Kleinman's algorithm (Kleinman 1968) for solving the CT Riccati equation. This is a Newton method for finding the optimal value. CT value iteration, on the other hand, is a new algorithm that solves the CT ARE based on iterations on certain discrete-time Lyapunov equations that are equivalent to (11.6-16).

Example 11.6-1. Continuous-time Optimal Adaptive Control Using IRL

This example shows the hybrid control nature of the IRL optimal adaptive controller. Consider the DC motor model

$$\dot{x} = Ax + Bu = \begin{bmatrix} -10 & 1 \\ -0.002 & -2 \end{bmatrix} x + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u,$$

with cost weight matrices $Q = I$, $R = I$. The solution to the CT ARE is computed to be

$$P = \begin{bmatrix} 0.05 & 0.0039 \\ 0.0039 & 0.2085 \end{bmatrix}.$$

In this simulation we used the IRL-based CT VI algorithm. This algorithm does not require knowledge of the system A matrix. For the CT LQR, the value is quadratic in the states. Therefore, the basis functions for the critic NN are selected as the quadratic polynomial vector in the state components, $\phi(x) = \begin{bmatrix} x_1^2 & x_1x_2 & x_2^2 \end{bmatrix}$. The IRL time interval was selected as $T = 0.04$ sec. To update the 3 critic weights $\overline{p}_{j+1} \equiv W_{j+1}$ (that is, the ARE solution elements) using (11.6-18), a batch LS solution can be obtained. Measurements of the data set $(x(t), x(t+T), \rho(t\colon t+T))$ are taken over 3 time intervals of $T = 0.04$ sec. Then, provided that there is enough excitation in the system, after each 0.12 sec enough data are collected from the system to solve for the value of the matrix P. Then a greedy policy update is performed using (11.6-15), that is, $u = -R^{-1}B^{\mathrm{T}}Px \equiv -Kx$.

The states are shown in Figure 11.6-2, which shows the good regulation achieved. The control input and feedback gains are shown in Figure 11.6-3. Note that the control

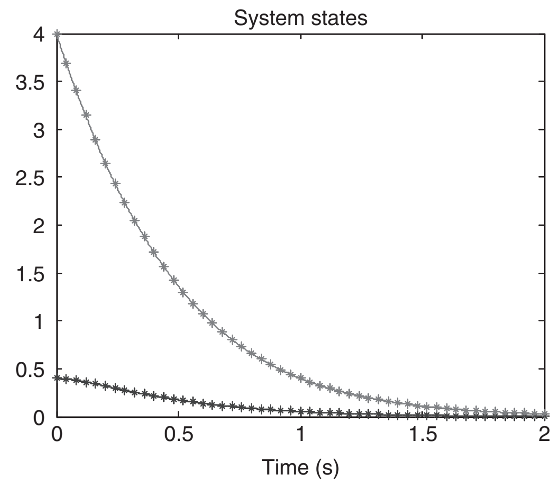

FIGURE 11.6-2 System states during the first 2 sec, showing that the continuous time IRL adaptive controller regulates the states to zero without knowing the system A matrix.

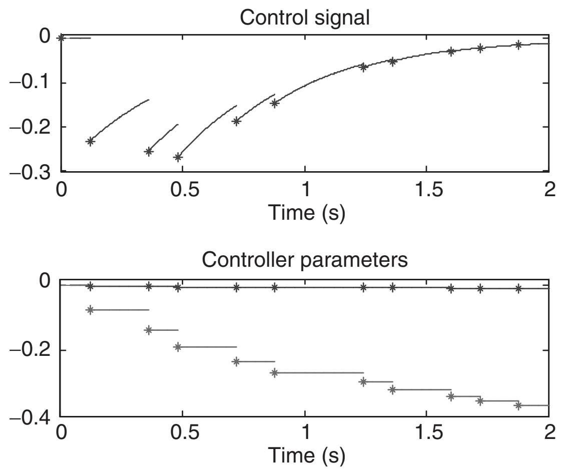

FIGURE 11.6-3 Control input and feedback gains, showing the hybrid nature of the IRL optimal adaptive controller. The controller gain parameters are discontinuous and piecewise constant, while the control signal itself is continuous between the gain parameter updates.

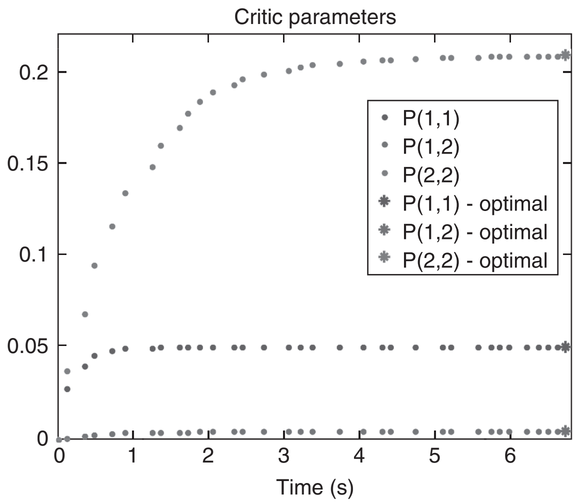

FIGURE 11.6-4 P matrix parameter estimates, showing that the IRL adaptive controller converges online to the optimal control Riccati equation solution without knowing the system A matrix.

gains are piecewise constant, while the control input is a continuous function of the state between policy updates. The critic NN parameter estimates are shown in Figure 11.6-4. They converge almost exactly to the entries in the Riccati solution matrix P. Thus, the ARE has been solved online without knowing the system A matrix. ■

Example 11.6-2. Continuous-time Optimal Adaptive Control for Power System Using IRL

In this example we simulate the CT IRL optimal adaptive control based on VI for the electric power system in Example 11.5-1. The same system matrices and performance index were used. The solution to the CT ARE is computed to be

$$P_{\mathrm{ARE}} = \begin{bmatrix} 0.4750 & 0.4766 & 0.0601 & 0.4751 \\ 0.4766 & 0.7831 & 0.1237 & 0.3829 \\ 0.0601 & 0.1237 & 0.0513 & 0.0298 \\ 0.4751 & 0.3829 & 0.0298 & 2.3370 \end{bmatrix}.$$

This is close to the DT ARE solution presented in Example 11.5-1, since the sample period used there is small.

The VI IRL algorithm was simulated, which does not require knowledge of the system A matrix. The IRL time interval was taken as $T = 0.1$ sec. (Note the IRL interval is not related at all to the sample period used to discretize the system in Example 11.5-1.) Fifteen data points $(x(t), x(t+T), \rho(t\colon t+T))$ were taken to compute each batch LS update for the critic parameters $\overline{p}_{j+1} \equiv W_{j+1}$ (e.g., the elements of the ARE solution P) using (11.6-18). Hence, the value estimate was updated every 1.5 sec. Then the policy was computed using (11.6-15), that is, $u = -R^{-1}B^{\mathrm{T}}Px \equiv -Kx$.

The state trajectories are similar to those presented in Example 11.5-1. The critic parameter estimates for the P matrix entries are shown in Figure 11.6-5. They converge to the true solution to the CT ARE. Thus, the ARE has been solved online without knowing the system A matrix.

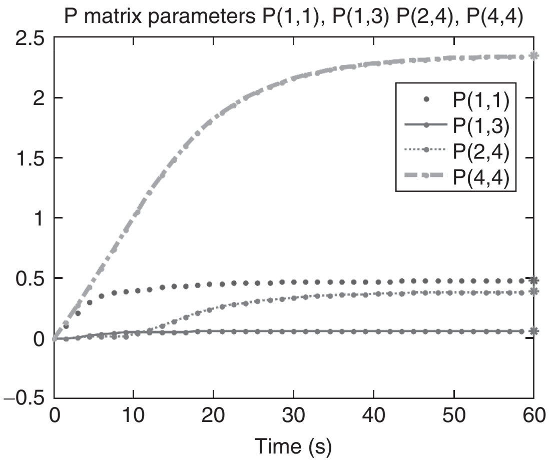

FIGURE 11.6-5 P matrix parameter estimates, showing that the IRL adaptive controller converges online to the optimal control Riccati equation solution without knowing the system A matrix.

Note that far less computation is needed using this IRL algorithm on the CT dynamics than was used in Example 11.5-1 for the DT optimal adaptive control algorithm. There, the critic parameter estimates were updated every 0.15 sec. Yet, the parameter estimates for the P matrix entries almost overlay each other. ■

Example 11.6-3. Continuous-time IRL Optimal Adaptive Control for Nonlinear System

In this example we show that IRL can solve the HJB equation for nonlinear CT systems by using data measured along the trajectories in real time. This example was developed using the converse HJB approach (Nevistic and Primbs 1996), which allows construction of nonlinear systems starting from the known optimal cost function.

Consider the nonlinear system given by the equations

$$\begin{cases} \dot{x}_1 = -x_1 + x_2 + 2x_2^3 \\ \dot{x}_2 = f(x) + g(x)u \end{cases}, \tag{11.6-22}$$

with $f(x) = -\frac{1}{2}(x_1 + x_2) + \frac{1}{2}x_2(1 + 2x_2^2)\sin^2(x_1)$, $g(x) = \sin(x_1)$. If we define $Q(x) = x_1^2 + x_2^2 + 2x_2^4$, $\mathrm{R} = 1$, then the optimal cost function for this system is $V^*(x) = \frac{1}{2}x_1^2 + x_2^2 + x_2^4$ and the optimal controller is $u^*(x) = -\sin(x_1)(x_2 + 2x_2^3)$. It can be verified that for these choices the HJB equation (11.6-7) and the Bellman equation (11.6-5) are both satisfied.

The cost function was approximated by the smooth function $V_j(x(t)) = W_j^{\mathrm{T}}\phi(x(t))$ with $L = 8$ neurons and $\phi(x) = \begin{bmatrix} x_1^2 & x_1x_2 & x_2^2 & x_1^4 & x_1^3x_2 & x_1^2x_2^2 & x_1x_2^3 & x_2^4 \end{bmatrix}^{\mathrm{T}}$. The PI IRL algorithm (11.6-17), (11.6-15) was used. This does not require knowledge of the drift dynamics $f(x)$.

To ensure exploration so that the HJB solution is found over a suitable region, data were taken along five trajectories defined by five different initial conditions chosen randomly in the region $\Omega = \{-1 \le x_i \le 1; i = 1, 2\}$. The IRL time period was taken as $T = 0.1$ sec. At each iteration step we set up a batch least-squares problem to solve for the eight NN weights using 40 data points measured on each of the 5 trajectories in Ω. Each data point consists of $(x(t), x(t+T), \rho(t\colon t+T))$, with $\rho(t\colon t+T)$ the measured integral

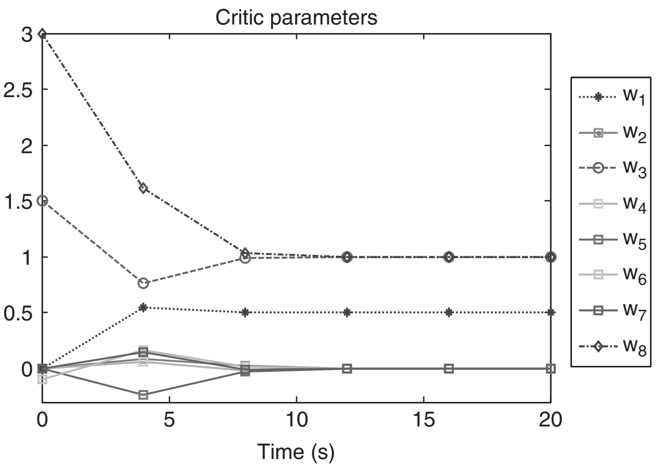

FIGURE 11.6-6 Convergence of critic NN parameters, showing that the IRL adaptive controller converges online to the optimal control Riccati equation solution without knowing the system A matrix.

reinforcement cost. In this way, at every 4 sec, the value was solved for and then a policy update was performed.

The result of applying the algorithm is presented in Figure 11.6-6, which shows that the parameters of the critic neural network converged to the coefficients of the optimal cost function $V^*(x) = \frac{1}{2}x_1^2 + x_2^2 + x_2^4$, that is, $W = \begin{bmatrix}0.5 & 0 & 1 & 0 & 0 & 0 & 0 & 1\end{bmatrix}^{\mathrm{T}}$. We observe that after 3 iteration steps (that is, after 12 sec) the critic NN parameters have effectively converged. Then the controller is close to the optimal controller $u^*(x) = -\sin(x_1)(x_2 + 2x_2^3)$. The approximate solution to the HJB equation has been determined online and the optimal control has been found without knowing the system drift dynamics $f(x)$. Note that analytic solution of the HJB equation in this example would be intractable. ■

11.7 SYNCHRONOUS OPTIMAL ADAPTIVE CONTROL FOR CONTINUOUS-TIME SYSTEMS

The integral reinforcement learning controller just given tunes the critic NN to determine the value while holding the control policy fixed, then a policy update is performed. Now we develop an adaptive controller that has two NN, one for value function approximation and one to approximate the control. We could call these the critic NN and actor NN. These two NN are tuned simultaneously, that is, synchronously in time (Vamvoudakis and Lewis 2010a and b). This is more nearly in line with accepted practice in adaptive control. Though this synchronous controller does require knowledge of the dynamics, it converges to the approximate local solutions to the HJB equation and the Bellman equation online, yet does not require explicitly solving either one. The HJB is generally impossible to solve for nonlinear systems.

Based on the CT Hamiltonian (11.6-6) and the stationarity condition $0 = \partial H(x, u, \nabla V^{\mu})/\partial u$, we could write a PI algorithm for CT systems based on the policy evaluation step

$$0 = H(x, \mu_j(x), \nabla V_{j+1}) = r(x, \mu_j(x)) + (\nabla V_{j+1})^{\mathrm{T}}(f(x) + g(x)\mu_j(x)),$$
$$V_{j+1}(0) = 0 \tag{11.7-1}$$

and the policy improvement step

$$\mu_{j+1} = \arg\min_{\mu} H(x, \mu, \nabla V_{j+1}). \tag{11.7-2}$$

Unfortunately, the full dynamics $f(x)$, $g(x)$ are needed to implement this algorithm. Moreover, (11.7-1) is a nonlinear equation and cannot generally be solved.

However, this algorithm provides the structure needed to develop another adaptive control algorithm that can be implemented online using measured data along the trajectories and converges to the optimal control. Specifically, select a value function approximation (VFA), or critic NN, structure as

$$V(x) = W_1^{\mathrm{T}}\phi(x) \tag{11.7-3}$$

and a control action approximation structure or actor NN as

$$u(x) = -\tfrac{1}{2}R^{-1}g^{\mathrm{T}}(x)\nabla\phi^{\mathrm{T}}W_2, \tag{11.7-4}$$

which could be, for instance, two neural networks with unknown parameters (weights) W_1, W_2, and $\phi(x)$ the basis set (activation functions) of the first NN. The structure of the second action NN comes from (11.6-15). Then, it can be shown that tuning the NN weights as

$$\dot{W}_1 = -\alpha_1 \frac{\sigma}{(\sigma^{\mathrm{T}}\sigma + 1)^2}[\sigma^{\mathrm{T}}W_1 + Q(x) + u^{\mathrm{T}}Ru], \tag{11.7-5}$$

$$\dot{W}_2 = -\alpha_2\{(F_2W_2 - F_1\overline{\sigma}^{\mathrm{T}}W_1) - \tfrac{1}{4}D(x)W_2m^{\mathrm{T}}(x)W_1\}, \tag{11.7-6}$$

guarantees system stability as well as convergence to the optimal value and control (Vamvoudakis and Lewis, 2010a,b).

In these parameter estimation algorithms, $\alpha_1, \alpha_2, F_1, F_2$ are algorithm tuning parameters, $D(x) = \nabla\phi(x)g(x)R^{-1}g^{\mathrm{T}}(x)\nabla\phi^{\mathrm{T}}(x)$, $\sigma = \nabla\phi(f + gu)$, $\overline{\sigma} = \sigma/(\sigma^{\mathrm{T}}\sigma + 1)$, and $m(x) = \sigma/(\sigma^{\mathrm{T}}\sigma + 1)^2$. A PE condition on $\overline{\sigma}(t)$ is needed to get convergence to the optimal value.

This is an adaptive control algorithm that requires full knowledge of the system dynamics $f(x), g(x)$, yet converges to the optimal control solution. That is, it solves (locally approximately) the HJB equation, which is generally intractable for general nonlinear systems. In the CT LQR case, it solves the ARE using data measured along the trajectories (and knowledge of A,B). The importance of this algorithm is that it can approximately solve the HJB equation for nonlinear systems using data measured along the system trajectories in real time. The HJB is generally impossible to solve for nonlinear systems.

The VFA tuning algorithm for W_1 is based on gradient descent, while the control action tuning algorithm is a form of backpropagation (Werbos 1989), which is, however, also tuned by the VFA weights W_1. The similarity to the actor–critic RL structure in Figure 11.1-1 is clear. However, in contrast to IRL, this algorithm is a CT optimal adaptive controller with two parameter estimators tuned simultaneously, that is, synchronously and continuously in time.

Example 11.7-1. Continuous-time Synchronous Optimal Adaptive Control

In this example we show that the synchronous optimal adaptive control algorithm can approximately solve the HJB equation for nonlinear CT systems by using data measured along the trajectories in real time. This example was developed using the method of Nevistic and Primbs (1996).

Consider the affine in the control input nonlinear system $\dot{x} = f(x) + g(x)u, x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^{\mathrm{T}} \in R^2$, where

$$f(x) = \begin{bmatrix} -x_1 + x_2 \\ -x_1^3 - x_2 - \frac{x_1^2}{x_2} + 0.25x_2(\cos(2x_1 + x_1^3) + 2)^2 \end{bmatrix}$$

$$g(x) = \begin{bmatrix} 0 \\ \cos(2x_1 + x_1^3) + 2 \end{bmatrix}.$$

We select $Q = I$, $R = 1$. Then the optimal value function that solves the HJB equation is $V^*(x) = \frac{1}{4}x_1^4 + \frac{1}{2}x_2^2$ and the optimal control policy is $u^*(x) = -\frac{1}{2}(\cos(2x_1 + x_1^3) + 2)x_2$.

We select the critic NN vector activation function as $\phi(x) = \begin{bmatrix} x_1^2 & x_2^2 & x_1^4 & x_2^4 \end{bmatrix}$. The tuning algorithms (11.7-5), (11.7-6) were run for the critic NN and control actor NN, respectively, simultaneously in time. A probing noise was added to the control to guarantee persistence of excitation. This noise was decayed exponentially during the simulation. The evolution of the states is given in Figure 11.7-1. They are stable and approach zero as the probing noise decays to zero.

Figure 11.7-2 shows the critic parameters, denoted by $W_1 = [W_{c1}\ \ W_{c2}\ \ W_{c3}\ \ W_{c4}]^{\mathrm{T}}$ After 80 sec the critic NN parameters converged to $W_1(t_f) = [0.0033\ \ 0.4967\ \ 0.2405\ \ 0.0153]^{\mathrm{T}}$, which is close to the true weights corresponding to the optimal value $V^*(x)$ that solves the HJB equation. The actor NN parameters converge to $W_2(t_f) = [0.0033\ \ 0.4967\ \ 0.2405\ \ 0.0153]^{\mathrm{T}}$. Thus, the control policy converges to

$$\hat{u}_2(x) = -\frac{1}{2}\begin{bmatrix} 0 \\ \cos(2x_1 + x_1^3) + 2 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 2x_1 & 0 & 4x_1^3 & 0 \\ 0 & 2x_2 & 0 & 4x_2^3 \end{bmatrix} \hat{W}_2(t_f).$$

This is the optimal control.

Figure 11.7-3 shows the 3-D plot of the difference between the approximated value function, by using the online synchronous adaptive algorithm, and the optimal value. The errors are small relative to the magnitude of the optimal value. Figure 11.7-4 shows the

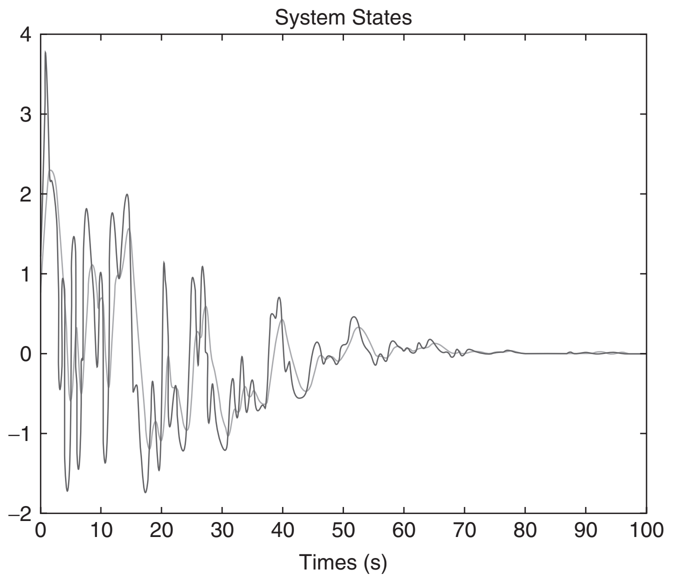

FIGURE 11.7-1 Evolution of the states, showing that the synchronous optimal adaptive controller ensures stability and regulates the states to zero.

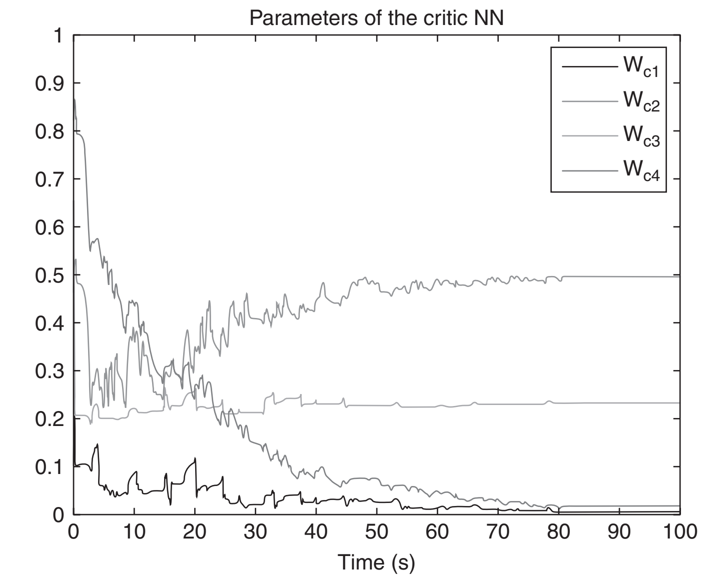

FIGURE 11.7-2 Convergence of critic NN parameters, showing that the optimal adaptive controller converges to the approximate solution of the nonlinear HJB equation.

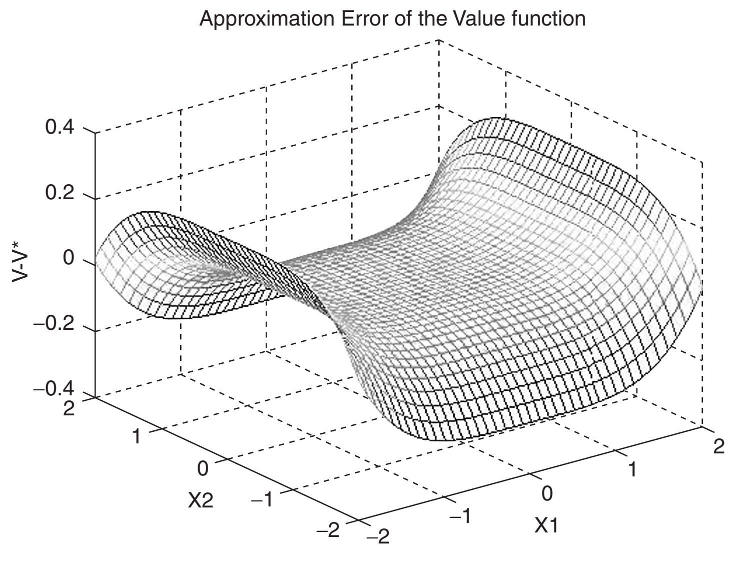

FIGURE 11.7-3 Error between optimal and approximated value function. This 3-D plot of the value function error shows that the synchronous optimal adaptive controller converges to a value function that is very close to the true solution of the HJB equation.

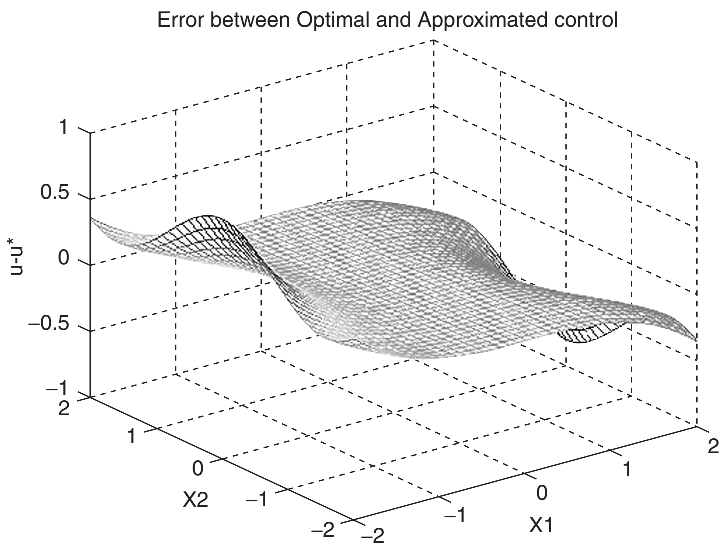

FIGURE 11.7-4 Error between optimal and approximated control input. This 3-D plot of the feedback control policy error shows that the synchronous optimal adaptive controller converges very close to the true optimal control policy.

3-D plot of the difference between the approximated feedback control policy found by using the online algorithm and the optimal control.

This example demonstrates that the synchronous optimal adaptive controller is capable of approximately solving the HJB equation online by using data measured along the system trajectories. The HJB equation for this example is intractable to solve analytically. ■

Queries in Chapter 11

Q1. We have shorten the running head since it exceeds the text width. Please confirm if this is fine with you.

Q2. We have shorten the running head since it exceeds the text width. Please confirm if this is fine with you.

Q3. We have placed Figure 11.6-6 before its callout due to fit this figure within the example. Please confirm if this is fine with you.