sensors

MDPI

Article

The Adaptive Dynamic Programming Toolbox

Xiaowei Xing and Dong Eui Chang *

School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, Korea; xwxing@kaist.ac.kr
* Correspondence: dechang@kaist.ac.kr

Abstract: The paper develops the adaptive dynamic programming toolbox (ADPT), which is a MATLAB-based software package and computationally solves optimal control problems for continuous-time control-affine systems. The ADPT produces approximate optimal feedback controls by employing the adaptive dynamic programming technique and solving the Hamilton–Jacobi–Bellman equation approximately. A novel implementation method is derived to optimize the memory consumption by the ADPT throughout its execution. The ADPT supports two working modes: model-based mode and model-free mode. In the former mode, the ADPT computes optimal feedback controls provided the system dynamics. In the latter mode, optimal feedback controls are generated from the measurements of system trajectories, without the requirement of knowledge of the system model. Multiple setting options are provided in the ADPT, such that various customized circumstances can be accommodated. Compared to other popular software toolboxes for optimal control, the ADPT features computational precision and time efficiency, which is illustrated with its applications to a highly non-linear satellite attitude control problem.

Keywords: adaptive dynamic programming; optimal control; software package

check for updates

Citation: Xing, X.; Chang, D.E. The Adaptive Dynamic Programming Toolbox. *Sensors* **2021**, *21*, 5609. https://doi.org/10.3390/s21165609

Academic Editor: Biswanath Samanta

Received: 16 July 2021
Accepted: 16 August 2021
Published: 20 August 2021

1. Introduction

Optimal control is an important branch in control engineering. For continuous-time dynamical systems, finding an optimal feedback control involves solving the so-called Hamilton–Jacobi–Bellman (HJB) equation [1]. For linear systems, however, the HJB equation simplifies to the well-known Riccati equation which results in the linear quadratic regulator [2]. For non-linear systems, solving the HJB equation is generally a formidable task due to its inherently non-linear nature. As a result, there has been a great deal of research devoted to approximately solving the HJB equation. Al'brekht proposed a power series method for smooth systems to solve the HJB equation [3]. Under the assumption that the optimal control and the optimal cost function can be represented in Taylor series, by plugging the series expansions of the dynamics, the cost integrand function, the optimal control and the optimal cost function into the HJB equation and collecting terms degree by degree, the Taylor expansions of the optimal control and the optimal cost function can be recursively obtained. Similar ideas can be found in [4,5]. A recursive algorithm is developed to sequentially improve the control law which converges to the optimal one by starting with an admissible control [6]. This recursive algorithm is commonly referred to as policy iteration (PI) and can be also found in [7–9]. The common limitation of these methods is that the complete knowledge of the system is required.

In the past few decades, reinforcement learning (RL) [10] has provided a means to design optimal controllers in an adaptive manner from the viewpoint of learning. Adaptive or approximate dynamic programming (ADP), which is an iterative RL-based adaptive optimal control design method, has been proposed in [11–15]. An approach that employs ADP is proposed in [11] for linear systems without requiring the priori knowledge of the system matrices. An ADP strategy is presented for non-linear systems with partially unknown dynamics in [12], and the necessity of the knowledge of system model is fully relaxed in [13–15].

Together with the growth of optimal control theory and methods, several software tools for optimal control have been developed. Notable examples are non-linear systems toolbox [16], control toolbox [17], ACADO [18], its successor ACADOS [19], and GPOPS-II [20]. A common feature of these packages is that system equations are used in them. In addition, optimal controls generated by [17–20] are open-loop, such that an optimal control is computed for each initial state. Therefore, if the initial state changes, optimal controls need to be computed again. In contrast, the non-linear systems toolbox [16] produces an optimal feedback control by solving the HJB equation.

The primary objective of this paper is to develop a MATLAB-based toolbox that solves optimal feedback control problems computationally for control-affine systems in the continuous-time domain. More specifically, employing the adaptive dynamic programming technique, we derive a computational methodology to compute approximate optimal feedback controls, based on which we develop the adaptive dynamic programming toolbox (ADPT). In the derivation, the Kronecker product used in [11,14] is replaced by Euclidean inner product for the purpose of memory saving during execution of the ADPT. The ADPT supports two working modes: the model-based mode and the model-free mode. The knowledge of system equations is required in the model-based mode. In the model-free mode, the ADPT produces the approximate optimal feedback control from measurements of system trajectories, removing the requirement of the knowledge of system equations. Moreover, multiple options are provided, such that the user can use the toolbox with much flexibility.

The remainder of the paper is organized as follows. Section 2 reviews the standard optimal control problem for a class of continuous-time non-linear systems and the model-free adaptive dynamic programming technique. Section 3 provides implementation details and software features of the ADPT. In Section 4, the ADPT is applied to a satellite attitude control problem in both the model-based mode and the model-free mode. Conclusions and potential future directions are given in Section 5. The codes of the ADPT are available at https://github.com/Everglow0214/The_Adaptive_Dynamic_Programming_Toolbox, accessed on 10 August 2021.

2. Review of Adaptive Dynamic Programming

We review the adaptive dynamic programming (ADP) technique to solve optimal control problems [13,14]. Consider a continuous-time control-affine system given by

$$\dot{x} = f(x) + g(x)u, \tag{1}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the control, $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are locally Lipschitz mappings with $f(0) = 0$. It is assumed that (1) is stabilizable at $x = 0$ in the sense that the system can be locally asymptotically stabilized by a continuous feedback control. To quantify the performance of a control, an integral cost associated with (1) is given by

$$J(x_0, u) = \int_0^\infty \left(q(x(t)) + u(t)^T R u(t)\right) \mathrm{d}t, \tag{2}$$

where $x_0 = x(0)$ is the initial state, $q : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is a positive definite function and $R \in \mathbb{R}^{m\times m}$ is a symmetric, positive definite matrix. A feedback control $u : \mathbb{R}^n \to \mathbb{R}^m$ is said to be admissible if it stabilizes (1) at the origin, and makes the cost $J(x_0, u)$ finite for all x_0 in a neighborhood of $x = 0$.

The objective is to find a control policy u that minimizes $J(x_0, u)$ given x_0. Define the optimal cost function $V^* : \mathbb{R}^n \to \mathbb{R}$ by

$$V^*(x) = \min_u J(x, u)$$

for $x \in \mathbb{R}^n$. Then, V^* satisfies the HJB equation

$$0 = \min_u \{\nabla V^*(x)^T(f(x) + g(x)u) + q(x) + u^T Ru\},$$

and the minimizer in the HJB equation is the optimal control which is expressed in terms of V^* as

$$u^*(x) = -\frac{1}{2}R^{-1}g(x)^T\nabla V^*(x).$$

Moreover, the state feedback u^* locally asymptotically stabilizes (1) at the origin and minimizes (2) over all admissible controls [2]. Solving the HJB equation analytically is extremely difficult in general except for linear cases. Hence, approximate or iterative methods are needed to solve the HJB, and the well-known policy iteration (PI) technique [6] is reviewed in Algorithm 1. Let $\{V_i(x)\}_{i\geq 0}$ and $\{u_{i+1}(x)\}_{i\geq 0}$ be the sequences of functions generated by PI in Algorithm 1. It is shown in [6] that $V_{i+1}(x) \leq V_i(x)$ for $i \geq 0$, and the limit functions $V(x) = \lim_{i\to\infty} V_i(x)$ and $u(x) = \lim_{i\to\infty} u_i(x)$ are equal to the optimal cost function V^* and the optimal control u^*.

Algorithm 1 Policy iteration

```
Input: An initial admissible control $u_0(x)$, and a threshold $\epsilon > 0$.
Output: The approximate optimal control $u_{i+1}(x)$ and the approximate optimal cost function $V_i(x)$.
1: Set $i \leftarrow 0$.
2: while $i \geq 0$ do
3:   Policy evaluation: solve for the continuously differentiable cost function $V_i(x)$ with $V_i(0) = 0$ using
       $$\nabla V_i(x)^T(f(x) + g(x)u_i(x)) + q(x) + u_i(x)^T Ru_i(x) = 0. \quad (3)$$
4:   Policy improvement: update the control policy by
       $$u_{i+1}(x) = -\frac{1}{2}R^{-1}g(x)^T\nabla V_i(x). \quad (4)$$
5:   if $\|u_{i+1}(x) - u_i(x)\| \leq \epsilon$ for all $x$ then
6:     break
7:   end if
8:   Set $i \leftarrow i + 1$.
9: end while
```

As proposed in [13,14], consider approximating the solutions to (3) and (4) by ADP instead of obtaining them exactly. For this purpose, choose an admissible feedback control $u_0 : \mathbb{R}^n \to \mathbb{R}^m$ for (1) and let $\{V_i(x)\}_{i\geq 0}$ and $\{u_{i+1}(x)\}_{i\geq 0}$ be the sequences of functions generated by PI in Algorithm 1 starting with the control $u_0(x)$. Following [13,14], choose a bounded time-varying exploration signal $\eta : \mathbb{R} \to \mathbb{R}^m$, and apply the sum $u_0(x) + \eta(t)$ to (1) as follows:

$$\dot{x} = f(x) + g(x)(u_0(x) + \eta(t)). \quad (5)$$

Assume that solutions to (5) are well defined for all positive time. Let $\mathcal{T}(x, u_0, \eta, [r,s]) = \{(x(t), u_0(x(t)), \eta(t)) \mid r \leq t \leq s\}$ denote the trajectory $x(t)$ of the system (5) with the input $u_0 + \eta$ over the time interval $[r,s]$ with $0 \leq r < s$. The system (5) can be rewritten as

$$\dot{x} = f(x) + g(x)u_i(x) + g(x)\nu_i(x,t), \quad (6)$$

where

$$\nu_i(x,t) = u_0(x) - u_i(x) + \eta(t).$$

Combined with (3) and (4), the time derivative of $V_i(x)$ along the trajectory $x(t)$ of (6) is obtained as

$$\dot{V}_i(x) = -q(x) - u_i(x)^T R u_i(x) - 2u_{i+1}(x)^T R v_i(x,t) \tag{7}$$

for $i \geq 0$. By integrating both sides of (7) over any time interval $[r,s]$ with $0 \leq r < s$, one gets

$$V_i(x(s)) - V_i(x(r)) = -\int_r^s (q(x) + u_i(x)^T R u_i(x) + 2u_{i+1}(x)^T R v_i(x,\tau))\, \mathrm{d}\tau. \tag{8}$$

Let $\phi_j : \mathbb{R}^n \to \mathbb{R}$ and $\varphi_j : \mathbb{R}^n \to \mathbb{R}^m$, with $j = 1,2,\ldots$, be two infinite sequences of continuous basis functions on a compact set in $\mathbb{R}^n$ containing the origin as an interior point that vanish at the origin [13,14]. Then, $V_i(x)$ and $u_{i+1}(x)$ for each $i \geq 0$ can be expressed as infinite series of the basis functions. For each $i \geq 0$ let $\hat{V}_i(x)$ and $\hat{u}_{i+1}(x)$ be approximations of $V_i(x)$ and $u_{i+1}(x)$ given by

$$\hat{V}_i(x) = \sum_{j=1}^{N_1} c_{i,j}\phi_j(x), \tag{9}$$

$$\hat{u}_{i+1}(x) = \sum_{j=1}^{N_2} w_{i,j}\varphi_j(x), \tag{10}$$

where $N_1 > 0$ and $N_2 > 0$ are integers and $c_{i,j}$, $w_{i,j} \in \mathbb{R}$ are coefficients to be found for each $i \geq 0$. Then, Equation (8) is approximated by $\hat{V}_i(x)$ and $\hat{u}_{i+1}(x)$ as follows:

$$\begin{aligned} &\sum_{j=1}^{N_1} c_{i,j}(\phi_j(x(s)) - \phi_j(x(r))) + \int_r^s (2\sum_{j=1}^{N_2} w_{i,j}\varphi_j(x)^T R\hat{v}_i)\, \mathrm{d}\tau \\ &\quad = -\int_r^s (q(x) + \hat{u}_i(x)^T R\hat{u}_i(x))\, \mathrm{d}\tau, \end{aligned} \tag{11}$$

where

$$\hat{u}_0 = u_0, \quad \hat{v}_i = u_0 - \hat{u}_i + \eta. \tag{12}$$

Suppose that we have K trajectories $\mathcal{T}(x, u_0, \eta, [r_k, s_k])$ available, $k = 1,\ldots,K$, where $x(t)$, $u_0(t)$, and $\eta(t)$ satisfy (6) over the K time intervals $[r_k, s_k]$, $k = 1,\ldots,K$. Then, we have K equations of the form (11) for each $i \geq 0$, which can be written as

$$e_{i,k} = 0, \quad k = 1,\ldots,K, \tag{13}$$

where

$$\begin{aligned} e_{i,k} := &\sum_{j=1}^{N_1} c_{i,j}(\phi_j(x(s_k)) - \phi_j(x(r_k))) + \int_{r_k}^{s_k} (2\sum_{j=1}^{N_2} w_{i,j}\varphi_j(x)^T R\hat{v}_i)\, \mathrm{d}\tau \\ &+ \int_{r_k}^{s_k} (q(x) + \hat{u}_i(x)^T R\hat{u}_i(x))\, \mathrm{d}\tau. \end{aligned}$$

Then, the coefficients $\{c_{i,j}\}_{j=1}^{N_1}$ and $\{w_{i,j}\}_{j=1}^{N_2}$ are obtained by minimizing

$$\sum_{k=1}^{K} \|e_{i,k}\|^2.$$

In other words, the K equations in (13) are solved in the least squares sense for the coefficients, $\{c_{i,j}\}_{j=1}^{N_1}$ and $\{w_{i,j}\}_{j=1}^{N_2}$. Thus two sequences $\{\hat{V}_i(x)\}_{i=0}^{\infty}$ and $\{\hat{u}_{i+1}(x)\}_{i=0}^{\infty}$ can be generated from (11). According to ([14], Cor. 3.2.4), for any arbitrary $\epsilon > 0$, there exist integers $i^* > 0$, $N_1^{**} > 0$ and $N_2^{**} > 0$, such that

$$\left\| \sum_{j=1}^{N_1} c_{i^*,j}\phi_j(x) - V^*(x) \right\| \leq \epsilon,$$

$$\left\| \sum_{j=1}^{N_2} w_{i^*,j}\varphi_j(x) - u^*(x) \right\| \leq \epsilon$$

for all x in a neighborhood of the origin, if $N_1 > N_1^{**}$ and $N_2 > N_2^{**}$.

Remark 1. *The ADP algorithm relies only on the measurements of states, the initial control policy and the exploration signal, lifting the requirement of knowing the precise system model, while the conventional policy iteration algorithm in Algorithm 1 requires the knowledge of the exact system model. Hence, the ADP algorithm is 100% data-based and model-free.*

Remark 2. *Equation (11) depends on the initial control u_0, the exploration signal η, the time interval $[r,s]$ as well as the index i, where the first three u_0, η, and $[r,s]$ are together equivalent to the trajectory $\mathcal{T}(x, u_0, \eta, [r,s])$ if the initial state $x(r)$ at $t = r$ is given. Hence, we can generate more diverse trajectories by changing η and $[r,s]$, as well as the initial state, and enrich the ADP algorithm accordingly, as follows. Suppose that we have available K trajectories $\mathcal{T}(x^k, u_0, \eta^k, [r_k, s_k])$, $1 \leq k \leq K$, where x^k, u_0 and η^k satisfy (6), i.e.,*

$$\dot{x}^k(t) = f(x^k(t)) + g(x^k(t))(u_0(x^k(t)) + \eta^k(t))$$

for $r_k \leq t \leq s_k$. Then, we have K equations of the form (11) for each $i \geq 0$, which can be written as $e_{i,k} = 0$, $k = 1, \ldots, K$, where

$$e_{i,k} := \sum_{j=1}^{N_1} c_{i,j}(\phi_j(x^k(s_k)) - \phi_j(x^k(r_k))) + \int_{r_k}^{s_k} (2\sum_{j=1}^{N_2} w_{i,j}\varphi_j(x^k)^T R\hat{v}_i^k)\, \mathrm{d}\tau$$
$$+ \int_{r_k}^{s_k} (q(x^k) + \hat{u}_i(x^k)^T R\hat{u}_i(x^k))\, \mathrm{d}\tau$$

with $\hat{u}_0 = u_0$ and $\hat{v}_i^k = u_0 + \eta^k - \hat{u}_i$. Then, the coefficients $\{c_{i,j}\}_{j=1}^{N_1}$ and $\{w_{i,j}\}_{j=1}^{N_2}$ are obtained by minimizing $\sum_{k=1}^{K} \|e_{i,k}\|^2$. For the sake of simplicity of presentation, however, in this paper we will fix η and the initial states and vary only the time intervals to generate trajectory data.

3. Implementation Details and Software Features

We now discuss implementation details and features of the adaptive dynamic programming toolbox (ADPT). We provide two modes to generate approximate optimal feedback controls; one mode requires the knowledge of system model, but the other eliminates this requirement, giving rise to the ADPT's unique capability of handling model-free cases.

3.1. Implementation of Computational Adaptive Dynamic Programming

To approximate $V_i(x)$ and $u_{i+1}(x)$ in (3) and (4), monomials composed of state variables are selected as basis functions. For a pre-fixed number $d \geq 1$, define a column vector $\Phi_d(x)$ by ordering monomials in graded reverse lexicographic order [21] as

$$\Phi_d(x) = (x_1, \ldots, x_n, x_1^2, x_1x_2, \ldots, x_n^2, \ldots, x_n^d) \in \mathbb{R}^{N \times 1},$$

where $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ is the state, $d \geq 1$ is the highest degree of the monomials, and N is given by

$$N = \sum_{i=1}^{d} \binom{i+n-1}{n-1}.$$

For example, if $n = 3$ and $d = 3$, the corresponding ordered monomials are

$$\begin{aligned}
&x_1, x_2, x_3;\\
&x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2;\\
&x_1^3, x_1^2x_2, x_1^2x_3, x_1x_2^2, x_1x_2x_3, x_1x_3^2, x_2^3, x_2^2x_3, x_2x_3^2, x_3^3.
\end{aligned}$$

According to (9) and (10), the cost function $V_i(x)$ and the control $u_{i+1}(x)$ are approximated by $\hat{V}_i(x)$ and $\hat{u}_{i+1}(x)$, which are defined as

$$\hat{V}_i(x) = c_i\Phi_{d+1}(x), \tag{14}$$
$$\hat{u}_{i+1}(x) = W_i\Phi_d(x), \tag{15}$$

where $d \geq 1$ is the approximation degree, and $c_i \in \mathbb{R}^{1\times N_1}$ and $W_i \in \mathbb{R}^{m\times N_2}$ are composed of coefficients corresponding to the monomials in $\Phi_{d+1}(x)$ and $\Phi_d(x)$ with

$$N_1 = \sum_{i=1}^{d+1}\binom{i+n-1}{n-1}, \quad N_2 = \sum_{i=1}^{d}\binom{i+n-1}{n-1}.$$

We take the highest degree of monomials to approximate V_i greater by one than the approximation degree since u_{i+1} is obtained by taking the gradient of V_i in (4) and $g(x)$ is constant in most cases.

Theorem 1. *Let a set of trajectories be defined as $\mathcal{S}_\mathcal{T} = \{\mathcal{T}(x, u_0, \eta, [r_k, s_k]), k = 1, 2, \ldots, K\}$ with $K \geq 1$, and let*

$$\begin{aligned}
\alpha(x) &= R\eta\Phi_d(x)^T,\\
\beta(x) &= R(u_0(x) + \eta)\Phi_d(x)^T,\\
\gamma(x) &= \Phi_d(x)\Phi_d(x)^T.
\end{aligned}$$

Then the coefficients c_i and W_i satisfy

$$A_i\begin{bmatrix} c_i^T \\ \mathrm{vec}(W_i)\end{bmatrix} = b_i, \tag{16}$$

where

$$A_0 = \left[\begin{array}{c:c} \Phi_{d+1}^{[r_1,s_1]}(x) & 2\mathrm{vec}(\int_{r_1}^{s_1}\alpha(x)\,\mathrm{d}t)^T \\ \vdots & \vdots \\ \Phi_{d+1}^{[r_K,s_K]}(x) & 2\mathrm{vec}(\int_{r_K}^{s_K}\alpha(x)\,\mathrm{d}t)^T \end{array}\right] \in \mathbb{R}^{K\times(N_1+mN_2)},$$

$$b_0 = \begin{bmatrix} -\int_{r_1}^{s_1}(q(x) + u_0(x)^TRu_0(x))\,\mathrm{d}t \\ \vdots \\ -\int_{r_K}^{s_K}(q(x) + u_0(x)^TRu_0(x))\,\mathrm{d}t \end{bmatrix} \in \mathbb{R}^{K\times 1},$$

and for $i = 1, 2, \ldots,$

$$A_i = \left[\begin{array}{c:c} \Phi_{d+1}^{[r_1,s_1]}(x) & 2\mathrm{vec}(\int_{r_1}^{s_1}(\beta(x) - RW_{i-1}\gamma(x))\,\mathrm{d}t)^T \\ \vdots & \vdots \\ \Phi_{d+1}^{[r_K,s_K]}(x) & 2\mathrm{vec}(\int_{r_K}^{s_K}(\beta(x) - RW_{i-1}\gamma(x))\,\mathrm{d}t)^T \end{array}\right] \in \mathbb{R}^{K\times(N_1+mN_2)},$$

$$b_i = \begin{bmatrix} -\int_{r_1}^{s_1}q(x)\,\mathrm{d}t - \langle W_{i-1}^TRW_{i-1}, \int_{r_1}^{s_1}\gamma(x)\,\mathrm{d}t\rangle \\ \vdots \\ -\int_{r_K}^{s_K}q(x)\,\mathrm{d}t - \langle W_{i-1}^TRW_{i-1}, \int_{r_K}^{s_K}\gamma(x)\,\mathrm{d}t\rangle \end{bmatrix} \in \mathbb{R}^{K\times 1},$$

where

$$\Phi_{d+1}^{[r_k,s_k]}(x) = \Phi_{d+1}(x(s_k))^T - \Phi_{d+1}(x(r_k))^T$$

for $k = 1, 2, \ldots, K$, the operator $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product with $\langle E, F \rangle = \sum_{ij} E_{ij} F_{ij}$ for matrices $E = [E_{ij}]$ and $F = [F_{ij}]$ of equal size, and the operator $\mathrm{vec}(\cdot)$ *is defined as*

$$\mathrm{vec}(Z) = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \in \mathbb{R}^{mn \times 1}$$

with $z_j \in \mathbb{R}^{m\times 1}$ being the jth column of a matrix $Z \in \mathbb{R}^{m \times n}$ for $j = 1, \ldots, n$.

Proof. Combining (11), (14) and (15), one has

$$\begin{aligned} & c_0(\Phi_{d+1}(x(s_k)) - \Phi_{d+1}(x(r_k))) + 2\int_{r_k}^{s_k} \Phi_d(x)^T W_0^T R\eta \, \mathrm{d}t \\ & = -\int_{r_k}^{s_k} (q(x) + u_0(x)^T R u_0(x)) \, \mathrm{d}t, \end{aligned} \tag{17}$$

and for $i = 1, 2, \ldots,$

$$\begin{aligned} & c_i(\Phi_{d+1}(x(s_k)) - \Phi_{d+1}(x(r_k))) + 2\int_{r_k}^{s_k} \Phi_d(x)^T W_i^T R(u_0(x) + \eta) \, \mathrm{d}t \\ & - 2\int_{r_k}^{s_k} \Phi_d(x)^T W_i^T R W_{i-1} \Phi_d(x) \, \mathrm{d}t \\ & = -\int_{r_k}^{s_k} (q(x) + \Phi_d(x)^T W_{i-1}^T R W_{i-1} \Phi_d(x)) \, \mathrm{d}t. \end{aligned} \tag{18}$$

By applying the property

$$\langle A, BC \rangle = \langle AC^T, B \rangle = \langle B^T A, C \rangle$$

of the Euclidean inner product, one may rewrite (17) and (18) as

$$\begin{aligned} & c_0(\Phi_{d+1}(x(s_k)) - \Phi_{d+1}(x(r_k))) + 2\Big\langle W_0, \int_{r_k}^{s_k} R\eta \Phi_d(x)^T \, \mathrm{d}t \Big\rangle \\ & = -\int_{r_k}^{s_k} (q(x) + u_0(x)^T R u_0(x)) \, \mathrm{d}t, \end{aligned} \tag{19}$$

and for $i = 1, 2, \ldots,$

$$\begin{aligned} & c_i(\Phi_{d+1}(x(s_k)) - \Phi_{d+1}(x(r_k))) + 2\Big\langle W_i, \int_{r_k}^{s_k} R(u_0(x) + \eta)\Phi_d(x)^T \, \mathrm{d}t \Big\rangle \\ & - 2\Big\langle W_i, R W_{i-1} \int_{r_k}^{s_k} \Phi_d(x)\Phi_d(x)^T \, \mathrm{d}t \Big\rangle \\ & = -\Big\langle W_{i-1}^T R W_{i-1}, \int_{r_k}^{s_k} \Phi_d(x)\Phi_d(x)^T \, \mathrm{d}t \Big\rangle - \int_{r_k}^{s_k} q(x) \, \mathrm{d}t. \end{aligned} \tag{20}$$

Then, the system of linear equations in (16) readily follows from (19) and (20). □

We now give the computational adaptive dynamic programming algorithm in Algorithm 2 for practical implementation. To solve the least squares problem in line 5 in the algorithm, we need to have a sufficiently large number K of trajectories, such that the minimization problem can be solved well numerically. Then the approximate optimal feedback control is generated by the algorithm as $\hat{u}_{i+1} = W_i \Phi_d(x)$.

Algorithm 2 Computational adaptive dynamic programming

```
Input: An approximation degree $d \geq 1$, an initial admissible control $u_0(x)$, an exploration
    signal $\eta(t)$, and a threshold $\epsilon > 0$.
Output: The approximate optimal control $\hat{u}_{i+1}(x)$ and the approximate optimal cost func-
    tion $\hat{V}_i(x)$.
 1: Apply $u = u_0 + \eta$ as the input during a sufficiently long period and collect necessary
    data.
 2: Set $i \leftarrow 0$.
 3: while $i \geq 0$ do
 4:    Generate $A_i$ and $b_i$.
 5:    Obtain $c_i$ and $W_i$ by solving the minimization problem
           $$\min_{c_i, W_i} \left\| A_i \begin{bmatrix} c_i^T \\ \mathrm{vec}(W_i) \end{bmatrix} - b_i \right\|^2.$$
 6:    if $i \geq 1$ and $\|c_i - c_{i-1}\|^2 + \|W_i - W_{i-1}\|^2 \leq \epsilon^2$ then
 7:       break
 8:    end if
 9:    Set $i \leftarrow i + 1$.
10: end while
11: return $\hat{u}_{i+1}(x) = W_i \Phi_d(x)$ and $\hat{V}_i(x) = c_i \Phi_{d+1}(x)$
```

Remark 3. *As in the statement of Theorem 1, several integral terms are included in A_i and b_i for $i \geq 0$. As in (12), u_0 does not get approximated by the basis functions, so the matrices A_0 and b_0 in Theorem 1 are obtained with $x(r_k)$, $x(s_k)$, $\int_{r_k}^{s_k} q(x)\,\mathrm{d}t$, $\int_{r_k}^{s_k} u_0(x)^T R u_0(x)\,\mathrm{d}t$ and $\int_{r_k}^{s_k} \alpha(x)\,\mathrm{d}t$, $1 \leq k \leq K$. For $i \geq 1$, the matrices A_i and b_i in Theorem 1 need, in addition, $\int_{r_k}^{s_k} \beta(x)\,\mathrm{d}t$ and $\int_{r_k}^{s_k} \gamma(x)\,\mathrm{d}t$, $1 \leq k \leq K$, as well as W_{i-1}.*

Remark 4. *In Theorem 1, the Kronecker product that is used in [11,14] for practical implementation is replaced by Euclidean inner product. Notice that $\int_{r_k}^{s_k} \gamma(x)\,\mathrm{d}t \in \mathbb{R}^{N_2 \times N_2}$ is symmetric, $k = 1, \ldots, K$. Thus, only upper triangular elements of these matrices are required to be stored. On the other hand, by using Kronecker product, one has to save all the elements of these matrices. As a result, less memory space of the processor is occupied by Theorem 1 especially when the number of basis functions to represent the approximate optimal control is large.*

Remark 5. *In the situation where the system dynamic equations are known, the ADPT uses the Runge–Kutta method to simultaneously compute the trajectory points $x(r_k)$ and $x(s_k)$ and the integral terms that appear in A_i and b_i. In the case when system equations are not known but trajectory data are available, the ADPT applies the trapezoidal method to evaluate these integrals numerically. In this case, each trajectory $\mathcal{T}(x, u_0, \eta, [r_k, s_k])$ is represented by a set of its sample points $\{x(t_{k,\ell}), u_0(t_{k,\ell}), \eta(t_{k,\ell})\}_{\ell=1}^{L_k}$, where $\{t_{k,\ell}\}_{\ell=1}^{L_k}$ is a finite sequence that satisfies $r_k = t_{k,1} < t_{k,2} < \ldots < t_{k,L_k-1} < t_{k,L_k} = s_k$, and then the trapezoidal method is applied on these sample points to numerically evaluate the integrals over the time interval $[r_k, s_k]$. If intermediate points in the interval $[r_k, s_k]$ are not available so that partitioning the interval $[r_k, s_k]$ is impossible, then we use the two end points r_k and s_k to evaluate the integral by the trapezoidal method as*

$$\int_{r_k}^{s_k} h(t)\,\mathrm{d}t \approx \frac{(s_k - r_k)(h(s_k) + h(r_k))}{2} \tag{21}$$

for a function $h(t)$.

3.2. Software Features

The codes of the ADPT are available at https://github.com/Everglow0214/The_Adaptive_Dynamic_Programming_Toolbox, accessed on 10 August 2021.

3.2.1. Symbolic Expressions

It is of great importance for an optimal control package that the user can describe functions, such as system equations, cost functions, etc., in a convenient manner. The idea of the ADPT is to use symbolic expressions. Consider an optimal control problem, where the system model is in the form (1) with

$$f(x) = \begin{bmatrix} x_2 \\ \frac{-k_1 x_1 - k_2 x_1^3 - k_3 x_2}{k_4} \end{bmatrix}, \quad g(x) = \begin{bmatrix} 0 \\ \frac{1}{k_4} \end{bmatrix}, \tag{22}$$

where $x = (x_1, x_2) \in \mathbb{R}^2$ is the state, $u \in \mathbb{R}$ is the control, and $k_1, k_2, k_3, k_4 \in \mathbb{R}$ are system parameters. The cost function is in the form (2) with

$$q(x) = 5x_1^2 + 3x_2^2, \quad R = 2. \tag{23}$$

Then in the ADPT the system dynamics and the cost function can be defined in lines 1–17 in Listing A1 provided in the Appendix A.

3.2.2. Working Modes

Two working modes are provided in the ADPT; the model-based mode and the model-free mode. The model-based mode deals with the situation where the system model is given, while the model-free mode addresses the situation where the system model is not known but only trajectory data are available. An example of the model-based mode is given in Listing A1, where after defining the system model (22), the cost function (23) and the approximation degree d in lines 1–20, the function, `adpModelBased`, returns the coefficients W_i and c_i for the control $\hat{u}_{i+1}$ and the cost function $\hat{V}_i$, respectively, in line 21.

An example of the model-free mode is shown in Listing A2 in the Appendix A, where the system model (22) is assumed to be unknown. The initial control u_0 is in the form of $u_0(x) = -Fx$ with the feedback control gain F defined in line 18. The exploration signal η is composed of four sinusoidal signals, as shown in lines 21–22. A list of two initial states $x(0) = (-3, 2)$ and $x(0) = (2.2, 3)$ is given in lines 28–29, and a list of the corresponding total time span for simulation is given in lines 30–31, where the time interval $[0, 6]$ is divided into sub-intervals of size 0.002 so that trajectory data are recorded every 0.002 second in lines 36–41. The time stamps are saved in the column vector `t_save` in line 39, and the values of states are saved in the matrix `x_save` in line 40, with each row in `x_save` corresponding to the same row in `t_save`. Similarly, the values of the initial control u_0 and the exploration signal η are saved in vectors `u0_save` and `eta_save` in lines 43–44. These measurements are passed to the function, `adpModelFree`, in lines 48–49 to compute the optimal control and the optimal cost function approximately.

In both the model-based and model-free modes the approximate control is saved in the file, uAdp.m, that is generated automatically and can be applied by calling `u=uAdp(x)` without dependence on other files. Similarly, the user may also check the approximate cost through the file, VAdp.m.

3.2.3. Options

Multiple options are provided such that the user may customize optimal control problems in a convenient way. We here illustrate usage of some of the options, referring the reader for the other options to the user manual available at https://github.com/Everglow0214/The_Adaptive_Dynamic_Programming_Toolbox, accessed on 10 August 2021.

In the model-based mode, the user may set option values through the function, `adpSetModelBased`, in a name-value manner before calling `adpModelBased`. That is, the specified values may be assigned to the named options. An example is shown in Listing A3 in the Appendix A, where two sets of initial states, time intervals and exploration signals are specified in lines 1–9. Then, in line 15 the output of `adpSetModelBased` should be passed to `adpModelBased` for the options to take effect. Otherwise, the default values would be used for the options as in line 21 in Listing A1.

For the command, `adpModelFree`, option values can be modified with the function, `adpSetModelFree`, in the name-value manner. Among the options, 'stride' enables the user to record values of states, initial controls and exploration signals in a high frequency for a long time, while using only a portion of them in the iteration process inside `adpModelFree`. To illustrate it, let each trajectory in the set $\mathcal{S_T}$ of trajectories in the statement of Theorem 1 be represented by two sample points at time r_k and s_k, that is, the trapezoidal method evaluates integrals over $[r_k, s_k]$ by taking values at r_k and s_k as in (21). Suppose that trajectories in $\mathcal{S_T}$ are consecutive, that is, $s_k = r_{k+1}$ for $k = 1, 2, \dots, K-1$. By setting 'stride' to a positive integer δ, the data used to generate A_i and b_i in Algorithm 2 become $\{\mathcal{T}(x, u_0, \eta, [r_{1+i\delta}, s_{(i+1)\delta}]), i \in \mathbb{N}, (i+1)\delta \leq K\}$. For example, consider 3 consecutive trajectories $\mathcal{T}(x, u_0, \eta, [r_k, r_{k+1}])$ with $k = 1, 2, 3$. If 'stride' is set to 1, one will have three equations from (11) as follows:

$$\begin{aligned}\sum_{j=1}^{N_1} c_{i,j}(\phi_j(x(r_{k+1})) - \phi_j(x(r_k))) + \int_{r_k}^{r_{k+1}} (2\sum_{j=1}^{N_2} w_{i,j}\varphi_j(x)^T R\hat{v}_i)\, \mathrm{d}\tau \\ = -\int_{r_k}^{r_{k+1}} (q(x) + \hat{u}_i(x)^T R\hat{u}_i(x))\, \mathrm{d}\tau\end{aligned}$$

for $k = 1, 2, 3$. These three equations contribute to three rows of A_i and three rows of b_i as in Theorem 1. If 'stride' is set to 3, then one will have only one equation from (11) as follows:

$$\begin{aligned}\sum_{j=1}^{N_1} c_{i,j}(\phi_j(x(r_4)) - \phi_j(x(r_1))) + \int_{r_1}^{r_4} (2\sum_{j=1}^{N_2} w_{i,j}\varphi_j(x)^T R\hat{v}_i)\, \mathrm{d}\tau \\ = -\int_{r_1}^{r_4} (q(x) + \hat{u}_i(x)^T R\hat{u}_i(x))\, \mathrm{d}\tau,\end{aligned} \tag{24}$$

where the integrals over $[r_1, r_4]$ are evaluated by the trapezoidal method with the interval $[r_1, r_4]$ partitioned into the three sub-intervals $[r_1, r_2] \cup [r_2, r_3] \cup [r_3, r_4]$, i.e, with the points at r_1, r_2, r_3, and r_4. Equation (24) will contribute to one row of A_i and one row of b_i as in Theorem 1. With the assumption that A_i has full rank with 'stride' set to 3, by setting 'stride' to 3, the number of equations in the minimization problem in Algorithm 2 is two thirds less than that with 'stride' set to 1, and as a result, the computation load is reduced in the numerical minimization. It is remarked that with 'stride' equal to 3, all the four points at $r_1, \dots, r_4$ are used by the trapezoidal method to evaluate the integrals over the interval $[r_1, r_4]$ in (24), producing a more precise value of integral than the one that would be obtained with the two end points at r_1 and r_4 only. An example of calling `adpSetModelFree` is shown in Listing A4 in the Appendix A. Similarly, `adpModelFree` takes the output of `adpSetModelFree` as an argument to validate the options specified.

4. Applications to the Satellite Attitude Stabilizing Problem

In this section, we apply the ADPT to the satellite attitude stabilizing problem because a stabilization problem can be formulated as an optimal control problem. In the first example, the system model is known and the controller is computed by the function `adpModelBased`. The same problem is solved again in the second example by the function `adpModelFree` when the system dynamics is unknown. The source codes for these two examples are available at https://github.com/Everglow0214/The_Adaptive_Dynamic_Programming_Toolbox (accessed on 10 August 2021), where more applications of the toolbox can be found.

4.1. Model-Based Case

Let $\mathbb{H}$ denote the set of quaternions and $\mathrm{S}^3 = \{q \in \mathbb{H} \mid \|q\| = 1\}$. The equations of motion of the continuous-time fully-actuated satellite system are given by

$$\dot{q} = \frac{1}{2}q\Omega, \tag{25}$$
$$\dot{\Omega} = \mathbb{I}^{-1}((\mathbb{I}\Omega) \times \Omega) + \mathbb{I}^{-1}u, \tag{26}$$

where $q \in \mathrm{S}^3$ represents the attitude of the satellite, $\Omega \in \mathbb{R}^3$ is the body angular velocity vector, $\mathbb{I} \in \mathbb{R}^{3\times3}$ is the moment of inertia matrix and $u \in \mathbb{R}^3$ is the control input. The quaternion multiplication is carried out for $q\Omega$ on the right-hand side of (25) where Ω is treated as a pure quaternion. By the stable embedding technique [22], the system (25) and (26) defined on $\mathrm{S}^3 \times \mathbb{R}^3$ is extended to the Euclidean space $\mathbb{H} \times \mathbb{R}^3$ [23,24] as

$$\dot{q} = \frac{1}{2}q\Omega - \alpha(|q|^2 - 1)q, \tag{27}$$
$$\dot{\Omega} = \mathbb{I}^{-1}((\mathbb{I}\Omega) \times \Omega) + \mathbb{I}^{-1}u, \tag{28}$$

where $q \in \mathbb{H}, \Omega \in \mathbb{R}^3$ and $\alpha > 0$.

Consider the problem of stabilizing the system (27) and (28) at the equilibrium point $(q_e, \Omega_e) = ((1, 0, 0, 0), (0, 0, 0))$. The error dynamics is given by

$$\dot{e}_q = \frac{1}{2}(e_q + q_e)e_\Omega - \alpha(|e_q + q_e|^2 - 1)(e_q + q_e),$$
$$\dot{e}_\Omega = \mathbb{I}^{-1}((\mathbb{I}e_\Omega) \times e_\Omega) + \mathbb{I}^{-1}u,$$

where $e_q = q - q_e$ and $e_\Omega = \Omega - \Omega_e$ are state errors. Since the problem of designing a stabilizing controller can be solved by designing an optimal controller, we pose an optimal control problem with the cost integral (2) with $q(x) = x^T Qx$, where $x = (e_q, e_\Omega) \in \mathbb{R}^7$ and $Q = 2I_{7\times7}$, and $R = I_{3\times3}$. The inertia matrix $\mathbb{I}$ is set to $\mathbb{I} = \mathtt{diag}(0.1029, 0.1263, 0.0292)$. The parameter α that appears in the above error dynamics is set to $\alpha = 1$.

We set the option 'xInit' with three different initial states. For each initial state, the option 'tSpan' is set to $[0, 15]$. We use the option 'explSymb' to set exploration signals; refer, for the usage of the option 'explSysb', to the user manual available at https://github.com/Everglow0214/The_Adaptive_Dynamic_Programming_Toolbox (accessed on 10 August 2021). For the initial control u_0, the default initial control is used, which is an LQR controller computed for the linearization of the error dynamics around the origin with the weight matrices $Q = 2I_{7\times7}$ and $R = I_{3\times3}$. We then call the function, `adpModelBased`, to generate controllers of degree $d = 1, 2, 3$. The computation time taken by the function, `adpModelBased`, to produce the controllers are recorded in Table 1. For the purpose of comparison, we also apply Al'brekht's method with the non-linear systems toolbox (NST) [16] to produce controllers of degree $d = 1, 2, 3$ for the same optimal control problem, and record their respective computation time in Table 1. For comparison in terms of optimality, we apply the controllers to the system (27) and (28) for the initial error state $x_0 = ((\cos(\theta/2) - 1, \sin(\theta/2), 0, 0), (0, 0, 0))$ with $\theta = 1.99999\pi$ and compute their corresponding values of the cost integral in Table 1. Since we do not know the exact optimal value of the cost integral $J(x_0, u)$ for this initial state, we employ the software package called ACADO [18] to numerically produce the optimal control for this optimal control problem with the given initial state. We note that both NST and ACADO are model-based.

We can see in Table 1 that ADPT in the model-based mode is superior to NST in terms of optimality, and ADPT (model-based) for $d = 2, 3$ is on par with ACADO in terms of optimality. Notice however that ACADO produces an *open-loop* optimal control for each given initial state, which is a drawback of ACADO, while ADPT produces a *feedback* optimal control that is independent of initial states. Moreover, even for the given initial state ACADO takes a tremendous amount of time to compute the open-loop optimal controller. From these observations, we can say that ADPT in the model-based mode is superior to NST and ACADO in terms of optimality, speed, and usefulness all taken into account.

Table 1. Costs at x_0 and computation time by ADPT, NST, and ACADO. $J(x_0, u)$ denotes the integral cost of the corresponding control u for initial state x_0. 'Time [s]' denotes the computation time taken by the method to obtain the controller.

		$J(x_0, u)$	**Time [s]**
ADPT (model-based)	$d = 1$	37.8259	1.5994
	$d = 2$	33.6035	3.2586
	$d = 3$	33.4986	13.1021
ADPT (model-free)	$d = 1$	43.8308	0.9707
	$d = 2$	36.8319	3.3120
	$d = 3$	37.4111	64.8562
NST	$d = 1$	208.9259	0.2702
	$d = 2$	94.6868	0.6211
	$d = 3$	64.0721	3.6201
ACADO	-	32.6000	2359.67

4.2. Model-Free Case

Consider solving the same optimal problem as in Section 4.1, but the system dynamics in (25) and (26), or equivalently the error dynamics are not available. Since we do not have real trajectory data available, for the purpose of demonstration we generate some trajectories with four initial states for the error dynamics, where the same initial control u_0 and exploration signals η are used as the model-based case in Section 4.1. The simulation for data collection is run over the time interval $[0, 20]$ with the recording period being 0.002 s, producing 10,000 = $20/0.002$ sampled points for each run. For the function `adpModelFree`, the option of 'stride' is set to 4. Then, the function, `adpModelFree`, is called to generate controllers of degree $d = 1, 2, 3$, the computation time taken for each of which is recorded in Table 1. For the purpose of comparison in terms of optimality, we apply the controllers generated by `adpModelFree` to the system (27) and (28) with the initial error state $x_0 = ((\cos(\theta/2) - 1, \sin(\theta/2), 0, 0), (0, 0, 0))$ with $\theta = 1.99999\pi$ and compute the corresponding values of the cost integral; see Table 1 for the values.

From Table 1, we can see that ADPT in the model-free mode takes more computation time than ADPT in the model-based mode, and the cost integrals by ADPT in the model-free working mode is slightly higher than those in the model-based working mode, since the integrals in the iteration process are evaluated less accurately. However, ADPT in the model-free mode is superior to NST in terms of optimality and to ACADO in terms of computation time. More importantly, it is noticeable that the result by model-free ADPT is comparable to model-based ADPT, which shows the power of data-based adaptive dynamic programming and the ADP toolbox.

To see how the computed optimal controller works in terms of stabilization, the norm of the state error under the control with $d = 3$ generated by ADPT in the model-free mode is plotted in Figure 1 together with the norm of state error by the NST controller with degree 3. We can see that the convergence to the origin is faster with the model-free ADP controller than with the controller by NST that is model-based. This comparison result is consistent with the comparison of the two in terms of optimality.

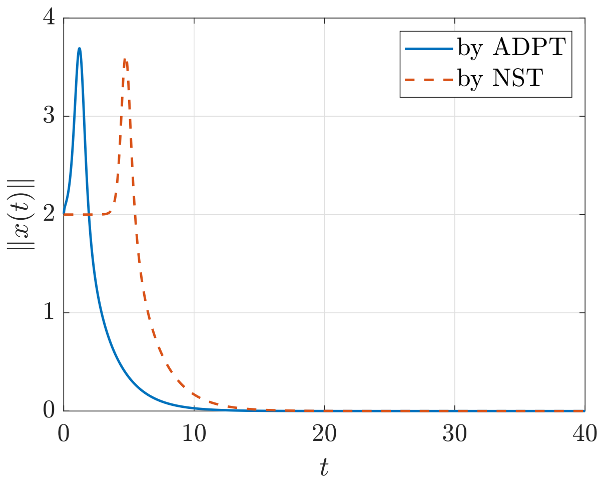

Figure 1. The state errors $\|x(t)\|$ with the controllers of degree 3 generated by ADPT in the model-free working mode and by NST.

4.3. Discussion

To compare with other toolboxes on ADP or RL, we investigate MATLAB reinforcement learning toolbox with the same control problem. Equations (27) and (28) are discretized using the 4th order Runge–Kutta method to construct the environment in reinforcement learning toolbox. The integrand in (2) is taken as the reward function. The deep deterministic policy gradient (DDPG) algorithm [25] is selected to train the RL agent since the control input in (26) is continuous. However, it is found in simulations that the parameters of the agent generally diverge even after a long training time and the system cannot be stabilized. The reason probably is that by setting only parameters of the exploration signal of standard normal distribution such as mean and deviation rather than choosing an exploration signal of a specific form, the system states may go to infinity in some episodes. Although one may stop the episode before all steps run out in such a situation, the experiences saved in the replay buffer may be detrimental to the training. On the other hand, the options provided by ADPT allow the user to determine what kind of trajectories to be used so that the optimal feedback control may be found quickly.

5. Conclusions and Future Work

The adaptive dynamic programming toolbox, a MATLAB-based package for optimal control for continuous-time control-affine systems, has been presented. By employing the adaptive dynamic programming technique, we propose a computational methodology to approximately produce the optimal control and the optimal cost function, where the Kronecker product used in previous literature is replaced by Euclidean inner product for less memory consumption at runtime. The ADPT can work in the model-based mode or in the model-free mode. The model-based mode deals with the situation where the system model is given while the model-free mode handles the situation where the system dynamics are unknown but only system trajectory data are available. Multiple options are provided, such that the ADPT can be easily customized. The optimality, the running speed, and the utility of the ADPT are illustrated with a satellite attitude stabilizing problem.

Currently control policies and cost functions are approximated by polynomials in the ADPT. As mathematical principles of neural networks are being revealed [26,27], we plan to use deep neural networks in addition to polynomials in the ADPT to approximately represent optimal controls and optimal cost functions to provide users of the ADPT more options.

Author Contributions: Conceptualization, X.X. and D.E.C.; methodology, X.X. and D.E.C.; software, X.X.; validation, X.X.; formal analysis, X.X.; investigation, X.X. and D.E.C.; writing—original draft preparation, X.X.; writing—review and editing, X.X. and D.E. C.; visualization, X.X.; supervision,

D.E.C.; project administration, D.E.C.; funding acquisition, D.E.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Center for Applied Research in Artificial Intelligence (CARAI) grant funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD) (UD190031RD), and by the NRF grant funded by the Korea government (MSIT) (2021R1A2C2010585).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Listing A1. An example of the model-based mode.

```
n = 2; % state dimension
m = 1; % control dimension
%% Symbolic variables.
syms x [n,1] real
syms u [m,1] real
syms t real

%% Define the system.
k1 = 3; k2 = 2; k3 = 2; k4 = 5;
f = [x2;
     (-k1*x1-k2*x1^3-k3*x2)/k4];
g = [0;
     1/k4];

%% Define the cost function.
q = 5*x1^2 + 3*x2^2;
R = 2;

%% Execute ADP iterations.
d = 3; % approximation degree
[w,c] = adpModelBased(f,g,x,n,u,m,q,R,t,d);
```

Listing A2. An example of the model-free mode.

```
n = 2; % state dimension
m = 1; % control dimension

%% Define the cost function.
q = @(x) 5*x(1)^2 + 3*x(2)^2;
R = 2;

%% Generate data.
syms x [n,1] real
syms t real
k1 = 3; k2 = 2; k3 = 2; k4 = 5;
% System dynamics.
f = [x2;
     (-k1*x1-k2*x1^3-k3*x2)/k4];
g = [0;
      1/k4];

F = [1, 1] % feedback gain

% Exploration signal.
eta = 0.8*(sin(7*t)+sin(1.1*t)+sin(sqrt(3)*t)+...
    sin(sqrt(6)*t));
e = matlabFunction(eta,'Vars',t);

% To be used in the function ode45.
dx = matlabFunction(f+g*(-F*x+eta),'Vars',{t,x});

xInit = [-3, 2;
          2.2, 3];
tSpan = [0:0.002:6;
          0:0.002:6];
odeOpts = odeSet('RelTol',1e-6,'AbsTol',1e-6);

t_save = [];
x_save = [];
for i = 1:size(xInit,1)
    [time, states] = ode45(@(t,x)dx(t,x),tSpan(i,:),...
        xInit(i,:),odeOpts);
    t_save = [t_save; time];
    x_save = [x_save; states];
end

u0_save = -x_save * F;
eta_save = e(t_save);

%% Execute ADP iterations.
d = 3; % approximation degree
[w,c] = adpModelFree(t_save,x_save,n,u0_save,m,...
    eta_save,d,q,R);
```

Listing A3. A demonstration of calling the function `adpSetModelBased`.

```
%% The user may specify settings.
xInit = [-3, 2;
          2.2, 3];
tSpan = [0, 10;
          0, 8];

syms t real
eta = [0.8*sin(7*t)+sin(3*t);
       sin(1.1*t)+sin(pi*t)];

adpOpt = adpSetModelBased('xInit',xInit,'tSpan',tSpan,...
    'explSymb',eta);

%% Execute ADP iterations.
[w,c] = adpModelBased(f,g,x,n,u,m,q,R,t,d,adpOpt);
```

Listing A4. A demonstration of calling the function `adpSetModelFree`.

```
%% The user may specify settings.
adpOpt = adpSetModelFree('stride',2);

%% Execute ADP iterations.
[w,c] = adpModelFree(t_save,x_save,n,u0_save,m,...
   eta_save,d,q,R,adpOpt);
```

References

1. Kirk, D.E. *Optimal Control Theory: An Introduction*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1970.
2. Lewis, F.L.; Vrabie, D.L.; Syrmos, V.L. *Optimal Control*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2012.
3. Al'brekht, E.G. On the optimal stabilization of nonlinear systems. *J. Appl. Math. Mech.* **1961**, *25*, 1254–1266. [CrossRef]
4. Garrard, W.L.; Jordan, J.M. Design of nonlinear automatic flight control systems. *Automatica* **1977**, *13*, 497–505. [CrossRef]
5. Nishikawa, Y.; Sannomiya, N.; Itakura, H. A method for suboptimal design of nonlinear feedback systems. *Automatica* **1971**, *7*, 703–712. [CrossRef]
6. Saridis, G.N.; Lee, C.-S.G. An approximation theory of optimal control for trainable manipulators. *IEEE Trans. Syst. Man Cybern.* **1979**, *SMC-9*, 152–159. [CrossRef]
7. Beard, R.W.; Saridis, G.N.; Wen, J.T. Galerkin approximations of the generalized Hamilton-Jacobi-Bellman equation. *Automatica* **1997**, *33*, 2159–2177. [CrossRef]
8. Beard, R.W.; Saridis, G.N.; Wen, J.T. Approximate solutions to the time-invariant Hamilton-Jacobi-Bellman equation. *J. Optim. Theory Appl.* **1998**, *96*, 589–626. [CrossRef]
9. Abu-Khalaf, M.; Lewis, F.L. Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach. *Automatica* **2005**, *41*, 779–791. [CrossRef]
10. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998.
11. Jiang, Y.; Jiang, Z.-P. Computational adaptive optimal control for continuous-time linear systems with completely unknown dynamics. *Automatica* **2012**, *48*, 2699–2704. [CrossRef]
12. Vrabie, D.L.; Lewis, F.L. Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems. *Neural Netw.* **2009**, *22*, 237–246. [CrossRef] [PubMed]
13. Jiang, Y.; Jiang, Z.-P. Robust adaptive dynamic programming and feedback stabilization of nonlinear systems. *IEEE Trans. Neural Netw. Learn. Syst.* **2014**, *25*, 882–893. [CrossRef] [PubMed]
14. Jiang, Y.; Jiang, Z.-P. *Robust Adaptive Dynamic Programming*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2014.
15. Lee, J.Y.; Park, J.B.; Choi, Y.H. Integral reinforcement learning for continuous-time input-affine nonlinear systems with simultaneous invariant explorations. *IEEE Trans. Neural Netw. Learn. Syst.* **2015**, *26*, 916–932. [PubMed]
16. Krener, A.J. Nonlinear Systems Toolbox. MATLAB Toolbox Available upon Request from ajkrener@ucdavis.edu.
17. Giftthaler, M.; Neunert, M.; Stäuble, M.; Buchli, J. The Control Toolbox—An open-source C++ library for robotics, optimal and model predictive control. In Proceedings of the IEEE 2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), Brisbane, Australia, 16–19 May 2018; pp. 123–129.

18. Houska, B.; Ferreau, H.J.; Diehl, M. ACADO Toolkit—An open source framework for automatic control and dynamic optimization. *Optim. Control Appl. Meth.* **2011**, *32*, 298–312. [CrossRef]
19. Verschueren, R.; Frison, G.; Kouzoupis, D.; Frey, J.; van Duijkeren, N.; Zanelli, A.; Novoselnik, B.; Albin, T.; Quirynen, R.; Diehl, M. ACADOS: A modular open-source framework for fast embedded optimal control. *arXiv* **2019**, arXiv:1910.13753.
20. Patterson, M.A.; Rao, A.V. GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Trans. Math. Softw.* **2014**, *41*, 1–37. [CrossRef]
21. Cox, D.A.; Little, J.; O'Shea, D. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*; Springer: New York, NY, USA, 2015.
22. Chang, D.E. On controller design for systems on manifolds in Euclidean space. *Int. J. Robust Nonlinear Control* **2018**, *28*, 4981–4998. [CrossRef]
23. Ko, W. A Stable Embedding Technique for Control of Satellite Attitude Represented in Unit Quaternions. Master's Thesis, Korea Advanced Institute of Science & Technology, Daejeon, Korea, 2020.
24. Ko, W.; Phogat, K.S.; Petit, N.; Chang, D.E. Tracking controller design for satellite attitude under unknown constant disturbance using stable embedding. *J. Electr. Eng. Technol.* **2021**, *16*, 1089–1097. [CrossRef]
25. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:509.02971.
26. Gurney, K. *An Introduction to Neural Networks*; UCL Press: London, UK, 1997.
27. Caterini, A.L.; Chang, D.E. *Deep Neural Networks in a Mathematical Framework*; Springer: New York, NY, USA, 2018.