Approximate dynamic programming-based control of distributed parameter systems

Article *in* Asia-Pacific Journal of Chemical Engineering · May 2011

DOI: 10.1002/apj.568

2 authors:

Midhun Joy
Lehigh University
6 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Niket S. Kaisare
Indian Institute of Technology Madras
84 PUBLICATIONS **1,853** CITATIONS
SEE PROFILE

ASIA-PACIFIC JOURNAL OF CHEMICAL ENGINEERING
Asia-Pac. J. Chem. Eng. 2011; **6**: 452–459
Published online 15 March 2011 in Wiley Online Library
(wileyonlinelibrary.com) DOI:10.1002/apj.568

Special theme research article

Approximate dynamic programming-based control of distributed parameter systems

*Midhun Joy and Niket S. Kaisare**

Department of Chemical Engineering, Indian Institute of Technology – Madras, Chennai 600036, India

Received 29 September 2010; Revised 21 January 2011; Accepted 31 January 2011

ABSTRACT: The objective of this work is to extend the approximate dynamic programming (ADP) framework to online control of distributed parameter systems. The ADP framework involves using suboptimal control policies to identify the relevant regions of the state space and to generate a *cost-to-go* function approximation applicable in this region. We present model-based *value iteration* and model-free *Q-learning* approaches for feedback control of an adiabatic plug flow reactor. The state dimension is reduced using appropriate model reduction and sensor placement techniques. We show that both the approaches provide better performance than the initial model predictive control and Proportional-Integral-Derivative (PID) controllers. Finally, an extension of ADP to the stochastic case with full state feedback is presented. © 2011 Curtin University of Technology and John Wiley & Sons, Ltd.

KEYWORDS: approximate dynamic programming; value iteration; Q-learning; hyperbolic PDEs; k-nearest neighbor approximator

INTRODUCTION

Distributed parameter systems (DPS) represent a wide class of dynamic processes, wherein the system variables and parameters vary at different locations within the system, in addition to variation with time. The dynamic behavior of such processes is modeled as a system of partial differential equations (PDEs) comprising of mass, momentum and energy balances. The task of synthesizing suitable control strategy is challenging because of the 'infinite dimensional' nature of DPS, while the system has only a finite number of control degrees of freedom.[1]

Control of DPS has a long and rich history, with some of the earlier work based on calculus of variations for optimal control of DPS. The origin of this field is often attributed to Butkovskiy and Lerner,[2] who derived maximum principle for a class of DPS. A lot of initial work in this field followed the seminal contributions of Wang and coworkers (see[3] for a review of their work). Balas[4] developed a feedback controller for controlling *finite number of modes* of a system governed by generalized wave equation. This work is perhaps a starting point for an important class of approaches to DPS control, where finite dimensional approximations like Galerkin's method are employed in order to develop a finite-dimensional controller.[1] Several applications of Lyapunov stability analysis of PDEs have been published since then. Another approach for robust controller synthesis is based on obtaining ordinary differential equations (ODEs) that capture the dominant dynamics of the PDE system through techniques such as approximate inertial manifold,[5,6] method of characteristics,[7] etc. This reduced-order model is then used for the controller synthesis based on the Lyapunov's direct methods. Arguably, the most popular method of model reduction for low order controller synthesis is proper orthogonal decomposition, also known as Karhunen–Loeve expansion.

After the initial surfeit of interest, application of optimal control for DPS receded in the last two decades. Besides variational and gradient-based approaches, dynamic programming (DP) method[8] was also popular in 1960s.[9] DP, which was introduced by Bellman,[8] provides a powerful method to solve any arbitrary stage-wise optimization problem. Analytical solutions to the DP problem were obtained for a small range of distributed parameter examples using procedures similar to the lumped parameter systems. Numerical solutions to DP are also possible. However, the numerical solution suffers from 'curse of dimensionality', referring to the fact that number of states grows exponentially with the number of state variables.

**Correspondence to*: Niket S. Kaisare, Department of Chemical Engineering, Indian Institute of Technology – Madras, Chennai 600036, India. E-mail: nkaisare@iitm.ac.in

The curse of dimensionality can be handled by limiting the control law calculations to only the relevant regions of the state space, which is the crux of approximate dynamic programming (ADP) approaches.[10] ADP identifies the *relevant* regions of the state space (instead of the entire state space, as in DP) through closed-loop simulations from suboptimal control laws; the control policy is then iteratively improved using procedures similar to those used in DP.[10,11] A suitable function approximator[12] can be used to interpolate the so-called 'cost-to-go' values that represent the 'desirability' of any state.

Both model-based and model-free ADP approaches have been proposed in the literature. The model-free learning control (MFLC) algorithms are particularly significant for complex systems where physics-based modeling approach cannot be applied. This class of methods is generalized as temporal-difference learning.[13] These methods learn directly from interactions of the controller (agent) with the process system (environment) without a model of the system dynamics. An example of MFLC for DPS is the use of actor-critic methods based on neural networks for optimal policy formulation.[14] A major limitation in the implementation of ADP to DPS is, however, that of limited data quality. Most of the distributed parameter systems have complex dynamics, which limits the state space that can be explored, leading to regions of sparse data. This limits the range over which the learned cost-to-go approximation can be applied without significant errors.

The objective of this paper is to extend the model-based and model-free ADP methods to control a distributed parameter system described by a set of hyperbolic PDEs. The rest of the paper is organized as follows: the next section introduces the system model and mathematical formulation of optimal control. The subsequent section provides a brief description on ADP; specifically, value iteration (model-based) and Q-learning (model-free) algorithms. Simulation results for implementation on a non-isothermal tubular reactor are discussed thereafter: equivalence between full-dimensional and reduced-state controllers is demonstrated, followed by application of the reduced-state controllers for model-based and model-free ADP. Application of value iteration and Q-learning to stochastic systems is also demonstrated briefly. Conclusions of this work are presented in the final section.

CONTROL FORMULATION FOR DPS

A mathematical model for a distributed parameter system obtained from conservation equations for a process system may be written in the following general form: ...

$$\frac{\partial \psi}{\partial t} = g\left(\psi, \frac{\partial \psi}{\partial z}, \frac{\partial^2 \psi}{\partial z^2}, u, \ldots\right) \tag{1}$$

where, $\psi(z,t)$ represents state variable(s) of interest and u represents manipulated input(s). The above PDE is classified as hyperbolic or parabolic PDE if it is first- or second-order in space, respectively. As the variable ψ varies over real-valued spatial coordinate, z, the overall model is said to be infinite dimensional.

The above PDE is converted into a set of ODEs using 'Method of Lines', where the entire spatial dimension is discretized at N discrete locations using finite difference approximation. The resulting ODE can be integrated over one sampling interval, from $k \cdot t_s$ to $(k+1)t_s$, to yield the following discrete-time model:

$$x_{k+1} = f(x_k, u_k) \tag{2}$$

In the above expression, $x = [\psi_1, \psi_2, \ldots, \psi_N]^T$ is the state vector consisting of the values of ψ at the N discrete locations (at time k), u_k is the manipulated input and $f(\cdot)$ is the transition function. The output is represented as the following nonlinear relation:

$$y_k = h(x_k, u_k) \tag{3}$$

The objective function for control implementation is written as minimization of certain cost function over a control horizon (p):

$$\min_{u(.)} \sum_{i=0}^{p} \gamma^i \phi(x_{k+i}, u_{k+i}) \tag{4}$$

The term $\phi(x,u)$ is the *single-stage cost* function and $\gamma \in (0,1)$ is discount factor that handles trade-off between the immediate and delayed costs. In a feedback control system, the future actions $\{u_k, \ldots, u_{k+p}\}$ are generated from a certain control policy $\mu(x)$. We define V_p^μ as the p-step performance function for the control policy $\mu(x)$:

$$V_p^\mu = \sum_{i=0}^{p} \gamma^i \phi(x_{k+i}, \mu(x_{k+i})) \tag{5}$$

Model predictive control (MPC) is an example of one such control policy, where one aims to find a sequence of control moves $\{u_k, u_{k+1}, \ldots, u_{k+p}\}$ that minimizes $V_p^\mu(x_k)$ over a fixed horizon (p) at each time k and implement the current input move u_k in a receding horizon manner.[15,16] However, the computational cost in solving a multi-stage optimization problem online at each sampling instance is significant, especially for nonlinear systems. Linear MPC, on the other hand, only

DOI: 10.1002/apj

provides a suboptimal solution to the online control problem.

One common objective function in control calculations is to regulate the output $y(k)$ at the desired set-point y_{sp}. The cost function for output regulation problem takes the typical quadratic form, with

$$\phi(x_k, u_k) = Q(y_{sp} - y_{k+1})^2 + R\Delta u_k^2 \quad (6)$$

Q and R are the error weightings which affects the robustness of the control policy. Other forms of the objective function are also used, e.g. maintaining a suitable state trajectory throughout the system, maximizing the yield of the desired product, or minimizing energy consumption. Any such control problem can be realized in the general framework described here. Finally, in case of infinite horizon control, we choose $p = \infty$.

APPROXIMATE DYNAMIC PROGRAMMING

DP provides a better theoretical approach to solving the optimal control problem, as it takes into consideration the cost accumulated over a infinite horizon of time rather than a fixed horizon. DP aims at finding optimal '*cost-to-go function*', $J^*(x_k)$, which is nothing but the minimum value of the performance index V_∞^μ achievable at any state $x_k \in \chi$:

$$J^*(x_k) = \min_\mu V_\infty^\mu(x_k) \quad (7)$$

Bellman's principle of optimality[8] states that the optimal cost-to-go function satisfies the following equation:

$$J^*(x_k) = \min_{u_k}[\phi(x_k, u_k) + \gamma J^*(f(x_k, u_k))] \ \forall x \in \chi \quad (8)$$

The optimal cost-to-go is obtained for all states in the state space, either analytically or numerically (in most cases). However, the computational requirement increases exponentially with state dimension because $J^*(x_k)$ needs to be computed for the entire state space. This is referred to as the *curse of dimensionality*.

ADP is a family of methods aimed at *learning* an approximation of optimal cost-to-go or value function. ADP circumvents the curse of dimensionality associated with DP by (1) performing iterations of the Bellman equation only over *relevant regions* of the state space and (2) using a function approximator to extend the cost-to-go values to other states in the state space. Simulations of suboptimal control policies are used to define these relevant regions of the state space. Iterations of the Bellman equation are used to characterize an optimal solution as a function of state or of state-action pair. Some algorithms are aimed at learning the cost-state mapping, whereas other algorithms try to obtain control policy directly as a function of the current state. Two different formulations of ADP, namely state-value function-based 'value iteration' algorithm and a model-free 'Q-learning' algorithm are described in this section.

Choice of approximator

The choice of a proper function approximator is essential for the convergence of the Bellman iteration and determines the quality of the final control policy. Two major classes of function approximator used include the parametric global approximators (e.g. neural network approximators) and non-parametric local approximators (e.g. k-nearest neighbors, kNN).

Following the recommendations of Lee *et al*.[12] kNN averaging is used to interpolate the cost-to-go values. kNN are memory-based approximators as they store the states and their corresponding cost-to-go values. Let x be any 'query point', where the cost-to-go value is desired; the interpolated cost-to-go value using the kNN approximator is given by:

$$\tilde{J}(x) = \sum_{x_j \in N_k(x)} w_j . J(x_j) \quad \text{where } w_j = \frac{1/d_j}{\sum_m 1/d_m} \quad (9)$$

Here, w_j denote the weights and $N_k(x)$ is the data set composed of the kNN of x based on the Euclidian distance between x and x_j. Distance weighting (rather than simple averaging) ranks the effect of neighboring points inversely as the distance from the query point, x.

Approximate value iteration

Value iteration method uses the Bellman equation (8) as a recursive update equation to obtain improved cost-to-go values. The value function approximations are continuously updated by sweeping through the state space until they converge to the optimal values. In case of value iteration-based ADP, the cost-to-go function is updated only for the states visited during the suboptimal policy simulation. The update equation for value iteration can be obtained by converting the Bellman equation into an update rule and is given by

$$J^i(x_k) = \min_{u_k \in \upsilon}[\phi(x_k, u_k) + \gamma \tilde{J}^{i-1}(f(x_k, u_k))] \ \forall x_k \in \chi^{\text{visited}} \quad (10)$$

DOI: 10.1002/apj

In the above equation, i denotes the update index and X^{visited} represents the covered regions of the state space. Value iteration is a model-based algorithm as it uses the state transition $f(x_k, u_k)$ to update the cost-to-go values. As cost-to-go is obtained over a finite number of states, kNN averaging is used to interpolate cost-to-go values within this region; $\tilde{J}(x_k)$ represents this interpolated value. In each iteration, the cost-to-go values are updated for each of the visited states. Value iteration requires infinite number of updates to converge to optimal value functions.[13] However, in actual practice, updating of the state values is terminated once the changes in the cost-to-go values in successive iterations become negligible.[13]

Q-learning

The Q-learning algorithm is a model-free learning control (MFLC) method which finds its roots in the reinforcement learning literature. In value iteration (also referred to as J-learning), the cost-to-go $J(x_k)$ is a function of the state x_k and expresses 'desirability' of x_k in context of the control problem. The cost-to-go function $Q(x_k, u_k)$ in Q-learning maps the cost function to the state-action pair (x_k, u_k). As the Q-values capture the effect of current state and input pair, Q-learning does not use an explicit model of the system.[13]

The traditional Q-learning approach is based on scalar reward function associated with each transition of the system. The state and action spaces in process control problems are continuous in nature. We tried the approach of Syafiie *et al*.[17] where the state and action spaces are quantized and a scalar reward function is defined for each transition. However, the simulations revealed that this quantization approach was ill-suited to the example due to the large state dimension. Moreover, the accuracy of the learned Q-values was dependent on the number of visits to each state-action pair. Multiple visits to the same state-action pair cannot be guaranteed due to the large system dimension, resulting in erroneous Q-values, and extremely slow and non-monotonic learning behavior.

Similar arguments were made by Lee[11] who implemented a modified Q-learning approach for a lumped parameter reactor model, adopting a procedure similar to that of the J-learning. In the modified algorithm, the Q-value is defined as:

$$Q(x_k, u_k) = \sum_{i=0}^{\infty} \gamma^i \phi(x_{k+i}, u_{k+i}) \tag{11}$$

In the modified Q-learning approach, $\phi(x_k, u_k)$ is a continuous function of the state and input (instead of a scalar reward associated with the state transition). The definition of Q-value differs from that of the J-value in one way: the J-value involves minimization over the action u_k. The Q-value is a function of the current state and the current action. The control actions are limited to the previously implemented region of the action space to ensure a stable plant performance.

As before, simulations of the suboptimal policies (PID, linear MPC, etc.) are used to initialize the state, action and Q-values required to initiate Q-learning. The kNN averaging interpolates the Q-values for unvisited state-action pairs. Unlike value iteration, an *episodic* learning procedure is used to improve the Q-values as follows:

1. Choose an initial state x_0. Implement the Q-learning-based control policy until *terminal state* is reached:
   a. Select an input action set (U_{x_j}) for state x_j by considering the range of input moves implemented for its nearest k neighbors.
   b. For each $u \in (U_{x_j})$ define the augmented state-action vector, $z = [x_j^T \; u^T]^T$. The Q-value for this z is computed from the kNN approximator as

$$\tilde{Q}(z) = \sum_{z_m \in N_k(z)} w_m . Q(z_m) \tag{12}$$

   c. Implement the control action computed by solving

$$u_j = \arg\min_{u \in U_{x_j}} \tilde{Q}^i(x_j, u_j) \tag{13}$$

2. Calculate Q-value (Q^{on}) for each visited state-action pair according to Eqn (11). The Q-values are thus computed for each x_j visited starting with x_0 in step 1 above.
3. Repeat steps 1 and 2 for a few initial conditions x_0, each time computing x_j, u_j, Q^{on}.
4. Update the Q-values for the state-action pairs actually visited during the control policy implementation (in steps 1 to 3 above) and those of the neighboring states using the Q^{on} values:

$$Q^{i+1}(z) = Q^i(z) + \alpha \left[Q^{i,\text{on}}(z) - Q^i(z) \right] \tag{14}$$

Here, i is the update index, α is the learning rate parameter and Q^{on} is Q-value for each state-action pair visited during the simulation run.

As with value iteration to obtain optimal $J(x)$, iterations of Eqn (14) are performed until $Q^i(z)$ converges to the optimal value. The converged values can then be used online to determine the control actions as per Eqn (13). Note that the online implementation is similar to step 1 in the Q-learning procedure above.

DOI: 10.1002/apj

CASE STUDY: CONTROL OF A NON-ISOTHERMAL TUBULAR REACTOR

Consider a non-isothermal tubular reactor that is used to carry out an irreversible, first order exothermic reaction. The model is the same as the one used in Ref. [7]. An incompressible flow is assumed inside the reactor with negligible radial gradients and axial diffusion.[7] The model comprises of the following hyperbolic PDEs:

$$\frac{\partial c}{\partial t} + (u+1)\frac{\partial c}{\partial x} + R(c,T) = 0 \tag{15}$$

$$\frac{\partial T}{\partial t} + (u+1)\frac{\partial T}{\partial x} - R(c,T) = 0 \tag{16}$$

$$R(c,T) = k_0 \exp\left(-\frac{E_0}{T+T_r}\right) c \tag{17}$$

In the above model, c and T are dimensionless deviation variables expressing concentration and temperature, respectively; $R(c,T)$ is the dimensionless reaction rate; $k_0 = 200$; $E_0 = 10$; $T_r = 0.5$. The dimensionless inlet concentration and temperature are $c_0 = 1$ and $T_0 = 1$.

Numerical solutions of the PDEs[15,16] are obtained using the method of lines. First order backward difference approximation is used to discretize the spatial derivatives in $N = 20$ equispaced grid points. The resulting system of ODEs describes the concentration and temperature variation at each of the 20 node points. Thus, the state vector is 40 dimensional, comprising of the concentration and temperature values at the 20 internal nodes along the reactor length. The reactor is operating at steady state conditions for input $u_{\text{initial}} = 1.1$. This corresponds to reactant exit concentration $\overline{c}_{(l=1)} = 0.85$. A step change in the set-point is given at $t = 0$. The control problem is to regulate the exit concentration (controlled variable) at $c = 0.4$. Feed flow rate is the manipulated variable.

Implementation of value iteration

The single stage cost in ADP implementation of the set-point tracking problem assumes the quadratic form (Eqn (6)), with the set-point for the outlet concentration at $y_{\text{sp}} = 0.4$. The output and input weights in this example are taken to be $Q = 1000$ and $R = 1$.

Linear MPC and PID controllers are used as suboptimal control laws for identification of relevant regions of the state space and to initialize cost-to-go approximation. Controller tuning parameters, given in Table 1, are chosen to ensure a reasonable control performance and an adequate coverage of the relevant regions of the state space. The reactor is initially kept at steady state corresponding to input value mentioned in the last column of the table. As noted earlier, the initial cost-to-go values are not vital as the Bellman iterations will improve the cost function; the controllers are primarily chosen to cover the relevant regions of the state space.

Table 1. Simulation parameters for the suboptimal control of the tubular reactor.

	Controller settings	Initial condition
PID_1	$K_c = 3.2$, $\tau_I = 0.3$, $\tau_D = 0.08$	$u_{SS} = 1.1$
PID_2	$K_c = 3$, $\tau_I = 0.7$, $\tau_D = 0.12$	$u_{SS} = 2.0$
MPC_1	$m = 15$; $p = 20$; $Q = 1$; $R = 10$; $\lvert\Delta u\rvert \le 0.5$	$u_{SS} = 1.1$
MPC_2	$m = 3$; $p = 8$; $Q = 1$; $R = 10$	$u_{SS} = 1.1$
MPC^*	$m = 10$; $p = 100$; $Q = 1000$; $R = 1$	$u_{SS} = 1.1$

MPC^* represents linear MPC with the parameters similar to the ADP controller. MPC^* was not used as an initial suboptimal controller.

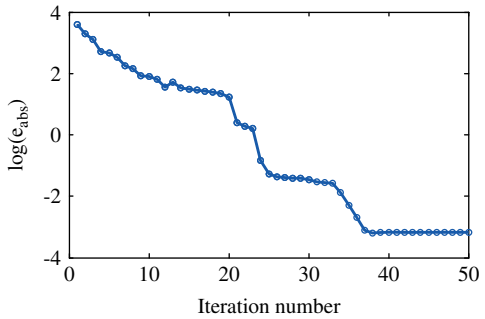

Figure 1. Convergence behavior of approximate value iteration algorithm. This figure is available in colour online at www.apjChemEng.com.

A total of 942 states (forming χ^{visited}) were sampled during the implementation of the suboptimal controllers. Value iteration is performed for the system, using Eqn (10), based on kNN approximation with $k = 4$. In other words, an interpolated cost-to-go value is used, which is a weighted average of the cost-to-go values of the four nearest neighbors (based on the Euclidian distance). The objective function of Eqn (6) is chosen to ensure integral action of the ADP controller. With this choice, the state vector has to be augmented with the previous control action, u_{k-1}. To ensure good ADP performance, the weights w_j in the kNN for the original states are chosen as in Eqn (9), while the weight for the input is chosen to be 0.

The change in cost-to-go values from successive iterations are shown in Fig. 1. The maximum deviation $\left(\max\left|\tilde{J}^{i+1} - \tilde{J}^{i}\right|\right)$ in the cost-to-go values in subsequent iterations is plotted *vs* the iteration index. The error in subsequent iterations decreases monotonically.

DOI: 10.1002/apj

The converged cost-to-go (after 38th iteration) is then used to compute control actions online.

Results from online implementation of ADP controller are shown as the solid line in Fig. 2. The suboptimal PID and MPC schemes are also shown for comparison. Clearly, the ADP-based controller shows a faster response compared to the original suboptimal policies. While the controller provides an excellent online performance, the computational time required is significant. Most of the computational effort goes in obtaining the kNN approximation because the kNN is a memory-based averaging. When the state dimension is large (state dimension is 40 in this example), indentifying the N_k nearest points based on the Euclidian distance from the query point becomes computationally demanding.

Implementation of ADP with lower dimensional state representation

Several approaches are possible for mitigating the computational complexity described above. These include reducing the state dimension, efficient method to identify the nearest neighbors, or using an alternate function approximator. In this work, the strategy of employing a low-dimensional state representation is implemented for the tubular reactor.

There are several algorithms outlined in the literature to obtain a reduced order state representation. The algorithm chosen in this section involves identifying the optimal set of sensor locations based on multivariate regression approach, and thus obtaining a lower dimensional representation of the state space. Using the standard methods available in the literature, the optimal set of sensor locations for the temperature and the concentration measurements were identified as node numbers 20 (end of the reactor), 11, 10, 4 and 1, each corresponding to the node location along the reactor length for the current discretization scheme. This simplified state representation, which takes into account only the measurements at these selected locations, is then used with the kNN approximator.

The value iteration procedure is repeated with the reduced-order representation, until the new cost-to-go function approximation converges. The converged cost-to-go is then used online for regulating the reactor at the desired set-point of $y_{sp} = 0.4$. The performance of the ADP controller based on the reduced-state dimension is shown as a dashed line in Fig. 3. The previous full-order ADP (solid line) as well as a linear MPC (dash–dot line) with the same output/input weights are shown for comparison. The parameters used for this linear MPC are given in the last row of Table 1. Clearly, the ADP controller performs better than the *linear* MPC with same controller tuning parameters. The reduced-order controller reveals no apparent drop in the performance. However, there is a significant reduction in computational load: The value iteration algorithm converges faster during offline learning and the time required for online computation of the control actions is significantly lower.

Implementation of *Q*-learning

The controller obtained using value iteration algorithm in the previous subsections requires an explicit model of the system or a means to obtain state transition $x_{k+1} = f(x_k, u_k)$. On the other hand, the Q-learning algorithm provides a model-free approach for control.

The procedure for Q-learning outlined in the previous section is implemented for the tubular reactor example.

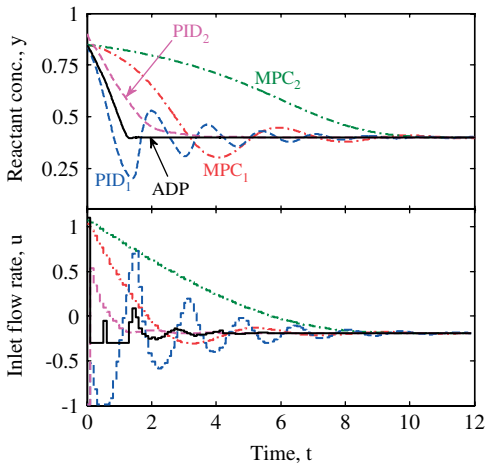

Figure 2. Comparison of ADP control (solid lines) with MPC (dash–dot lines) and PID (dashed lines) control schemes used as the initial suboptimal control policies. This figure is available in colour online at www.apjChemEng.com.

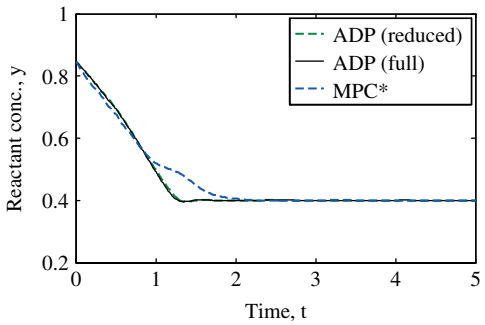

Figure 3. Performance of the ADP algorithm with a lower dimensional state representation. Results of the ADP algorithm with full state representation and linear MPC with the same weights are shown for comparison. This figure is available in colour online at www.apjChemEng.com.

DOI: 10.1002/apj

The suboptimal control policies of Table 1 are used to generate the initial data (x_k, u_k, Q). The Q-values for any new state-action pair are approximated using the kNN averaging, with the weights chosen as before. The available choice of actions at each state is limited to the range of input moves previously implemented during the course of training (i.e. initial suboptimal policies and previous iterations) as per Eqn (13). After the simulation of each greedy control policy, the set of Q-values is updated. This procedure is repeated until the Q-values converge, which required a total of 150 updates for our example.

The performance of the controller with the final Q-value estimates (i.e. after 150 iterations) is shown in Fig. 4. Simulations are performed for various different initial conditions, each corresponding to the steady state temperature and concentration profile for a particular inlet feed rate [initial $u_{ss} = (1.2, 0.3, 0, -0.2, -0.5, -0.8)$]. In each case, the controller attempts to take the system from an initial state (corresponding to the chosen u_{ss} value) to the set-point of 0.4. The results show that the Q-learning algorithm is able to provide good control performance even for the initial conditions not previously seen during the learning process (i.e. good interpolation capabilities).

Extension to stochastic systems

The effect of unmeasured disturbances, model uncertainty and sensor errors gives rise to stochasticity in the system model. The discrete-time state space model is modified to account for uncertainties using additive state and measurement noise:

$$x_{k+1} = f(x_k, u_k) + \varepsilon_k \tag{18}$$

$$y_k = h(x_k) + v_k \tag{19}$$

ε_k and v_k are normally distributed random error signals with zero mean. Their variances are chosen so as to maintain signal to noise ratio (SNR) of 10. Such a formulation for the error signals ensures that a suitable SNR is maintained for all the measurements, in spite of each measurement having different variabilities.

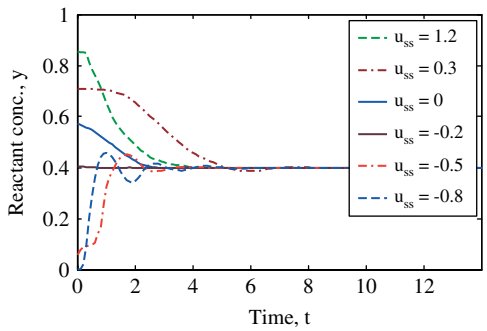

Figure 4. Online performance of the Q-learning-based control policy for different initial conditions. This figure is available in colour online at www.apjChemEng.com.

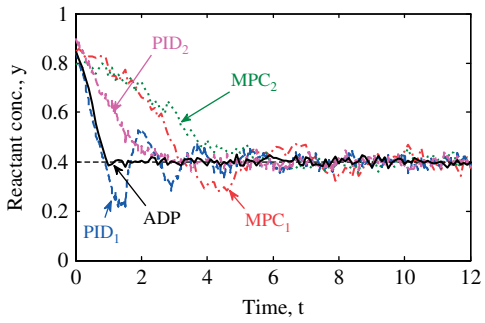

Figure 5. Comparison of the system responses under various control policies (PID, MPC and value iteration-based ADP) for a stochastic case. This figure is available in colour online at www.apjChemEng.com.

The presence of the stochastic factor in the system response introduces further complexities in the cost-to-go calculations for a particular control policy. In this case, each control policy simulation becomes essentially a continuing task rather than an episodic one (an episodic task is the one which has a terminal state associated with it, whereas a continuing task has no such terminal state). A discounted formulation is used with $\gamma = 0.90$ to avoid cost-to-go values blowing up to infinity. The update equation for value iteration is suitably modified as

$$J^i(x) = \min_u [E\{\phi(x,u) + \gamma \tilde{J}^{i-1}(f(x,u))\}] \tag{20}$$

The cost-to-go values for the different states in the state space are estimated by averaging the values obtained over 50 Monte-Carlo runs (i.e. 50 possible successor states are sampled to evaluate the expectation operator above). Once the cost values associated with the optimal policy are obtained, the optimal policy can be found out using a single step optimization. Full state feedback is assumed and an observer is therefore not required. Figure 5 compares the performance of the ADP policy with other control policies when the system response is stochastic in nature. By comparing with results of Fig. 2, we find that the ADP-based method is able to provide good control performance in presence of state and measurement noise.

CONCLUSIONS

In this paper, we have analyzed the suitability of the ADP-based control for systems modeled using partial

DOI: 10.1002/apj

differential equations. Both the model-based value iteration (J-learning) and the model-free Q-learning algorithms showed optimal control performance at a reasonable computational cost for the high-dimensional model resulting from the distributed parameter nature of the system. The Q-learning approach can therefore be used for an optimal control policy formulation even in the absence of a perfect knowledge of the system dynamics. As both the ADP approaches rely on obtaining a state-value mapping, extension to large dimensional DPS is nontrivial. We showed that a reduced-state representation of the system with sensors placed at five locations along the reactor were able to provide excellent control performance with reduced computational requirements. In summary, we identify several avenues for research on ADP for distributed parameter systems, such as improving the state-value mapping, development of efficient function approximators, techniques to identify low-dimensional manifolds in the system (model) or data (approximator), convergence of the learning algorithm, and incorporating risk factor in learning.

REFERENCES

[1] M.J. Balas. *J. Math. Anal. Appl.,* **1983;** *91*, 527–546.

[2] A.G. Butkovskiy, A.Y. Lerner. *Automat. Rem. Contr.,* **1960;** *21*, 472–477.

[3] P.K.C. Wang. *Int. J. Contr.,* **1968;** *17*, 101–116.

[4] M.J. Balas. *IEEE Trans. Automat. Contr.,* **1978;** *AC-23*, 673–679.

[5] P.D. Christofides, P. Daoutidis. *Comput. Chem. Eng.,* **1996;** *20*, S1071–S1076.

[6] P.D. Christofides, P. Daoutidis. *Chem. Eng. Sci.,* **1998;** *53*, 85–105.

[7] E.M. Hanczyc, A. Palazoglu. *Ind. Eng. Chem. Res.,* **1995;** *34*, 557–566.

[8] R. Bellman. *Dynamic Programming*, Princeton University Press: Princeton, **1957**.

[9] W.H. Ray. *Automatica,* **1978;** *14*, 281–287.

[10] J.H. Lee, J.M. Lee. *Comput. Chem. Eng.,* **2006;** *30*, 1603–1618.

[11] J.M. Lee. A study on architecture, algorithms and applications of approximate DP based approach to optimal control, *Ph.D. Thesis*, Georgia Institute of Technology, **2004**.

[12] J.M. Lee, N.S. Kaisare, J.H. Lee. *J. Process Contr.,* **2006;** *16*, 135–156.

[13] R.S. Sutton, A.G. Barto. *Reinforcement Learning: An Introduction*, The MIT Press: Cambridge, **1998**.

[14] R. Padhi, S.N. Balakrishnan. *Neural Netw.,* **2003;** *16*, 719–728.

[15] M. Morari, J.H. Lee. *Comput. Chem. Eng.,* **1999;** *23*, 667–682.

[16] J.H. Lee, N.L. Ricker. *Ind. Eng. Chem. Res.,* **1994;** *33*, 1530–1541.

[17] S. Syafiie, F. Tadeo, E. Martinez. *Eng. Appl. Artif. Int.,* **2007;** *20*, 767–782.

DOI: 10.1002/apj