A Study on Architecture, Algorithms, and Applications of Approximate Dynamic Programming Based Approach to Optimal Control

A Thesis
Presented to
The Academic Faculty

by

Jong Min Lee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Chemical and Biomolecular Engineering
Georgia Institute of Technology
July 2004

A Study on Architecture, Algorithms, and Applications of Approximate Dynamic Programming Based Approach to Optimal Control

Approved by:

Dr. Jay H. Lee, Advisor

Dr. Martha A. Gallivan

Dr. F. Joseph Schork

Dr. Shabbir Ahmed

Dr. Matthew J. Realff

Date Approved: July 8. 2004

To my parents, Chang Sik Lee and Jung Hee Shin,

and

To my brother, Hee Dong Lee.

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest gratitude to my advisor Professor Jay H. Lee for all his guidance, advice, and financial support throughout the journey of my research at Georgia Tech. During the past five years, he has shown me "the" example of a good researcher possessing in-depth knowledge of the research field, a logical way of approaching problems, and superb technical writing and communication skills. Whenever I was frustrated by my lack of perspective and progress on the research, he helped me to find the right direction. His seriousness and perfectionism toward research and professional work present the model that I want to mould myself into for my future career. I also want to thank Prof. Lee and Mrs. Lee for kindly hosting many parties and dinners to encourage students who were far away from their homes.

I would like to acknowledge the helpful comments and advice I received from the professors who took the time to serve on my thesis committee: Professors Schork, Realff and Gallivan from the department, Professor Ahmed from the School of Industrial and Systems Engineering, and Professor Koenig from the College of Computing.

I want to thank my former advisor at Seoul National University, Professor En Sup Yoon, for having initially interested me in process systems engineering, for recommending me to Professor Lee as a Ph.D. student, and for his continued guidance.

When I first came to the States to begin graduate study, I would not have been able to get through all the difficulties of my first year without my fellow students, Bumsang Kim and Sungyong Mun. In every aspect they acted like big brothers to me. I hope and am confident that their academic careers as professors will be both prosperous and successful. I thank Dr. Seung-Jin Lee and Dr. Su Whan Sung for helping me to settle down at Purdue. I am also indebted to Dr. Jinhyun Kim and Dr. Hyun-Seob Song for the helpful advice and the many get-togethers that they hosted for us.

I would like to acknowledge all of the past and present members of the Lee group (ISSICL). Dr. Kangwook Lee for his kindness and for his taking care of me, especially when we moved down to Atlanta. Andrew Dorsey, Yangdong Pan, Jian Xiao, and Seshatre Natarajan for showing me the ropes as senior students. Niket S. Kaisare for being a dependable colleague, a nice friend (especially for our frequent lunch trips), and an intelligent coworker. Juan Camilo Zapata for being an energetic optimist. Jaein Choi for the concern he showed for me as a single guy, and for treating me like a family member, including countless invitations to family occasions as well as breakfasts, lunches, and dinners. He and his family (Mrs. Eunmyung Hong and his daughter Seoyoon Choi) made my life in Atlanta more stable and enjoyable. And Thidarat Tosukhowong, Manish Gupta, Swathy Ramaswamy, Anshul Dubey, and Nikolaos Pratikakis for being nice and responsible junior students to work with. Over the past years, I have had great opportunities to get to know and to work with many visiting scholars. I also would like to thank them for their friendship, Prof. Dae R. Yang, Jochen Till, Hyungjin Park, Catalina Peroni, Changkyu Yoo, Prof. Hyunkak Han, Dr. Jongku Lee, Dr. Kyung Joo Mo, and Heejin Lim.

In 2002 I had a great experience working for Owens Corning as an engineering intern. The experience was of great benefit to my research and it motivated me to look at research problems from a more practical viewpoint. Besides that, I was more than fortunate to work with nice people. I would like to thank Dr. James R. Beilstein for giving me exciting opportunities working for interesting projects in many different plant sites all over the U.S., and Dr. Wei Li for his guidance, practical discussions, and for taking me to nice restaurants. I would also like to thank the other Advanced Process Control team members (Sanjay Mansukhani and Elaina Carpino) and Aspentech people who were of great help to me when I was in Ohio.

Korean students and postdocs in the department have always been good friends to turn to, and a great diversion from work. They were always there when my fingers were itchy to play basketball or tennis, as well as I was thirsty. I thank them for their love and care for each other – Seongho Park, Se-Young Yoon, Young-Soo Kim, Yeu Chun Kim, Jeongwoo Lee, Ingu Song, Dr. Ketack Kim, Dr. Jaewon Lee, Dr. Weontae Oh, and Dr. Jeonghyun

Yeom.

I would like to thank my high school friends for being superb morale-boosters: Kangwook Yoon, Hyunsung Jeong, Jihyung Kim, Kiseong Jeong, and Taemin Eom. Friends from SNU helped me a lot too. Dr. Myungjune Park provided me with model parameters for the MMA polymerization process, and Geunsoo Chang has often offered encouragement. Dr. Sang Ok Song, Dong Eon Lee, and Chang Jun Lee have always been there cheering me up. And of course, I am also grateful to all the past and current PSLAB members, for our memories and friendship.

I thank my grandparents, uncle, and aunts for their love and care. And last but not least, I would like to thank my family for their support over the years. My brother, Hee Dong, has always shown a resolute attitude and an endurance toward everything, which has inspired me a lot. My parents tried their hardest to support me, and I would not be where I am today without their endless support and love.

I conclude by thanking all the people who have shown me their kindness, decency, and sense of humor. They were the ones that made it possible for me to continue my educational pursuits.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Further development of MPC for complex systems has been impeded by: difficult system identification, on-line computational burden, and fundamental limitations of handling uncertainties. Dynamic Programming (DP) has advantages over MPC in that it finds an optimal control policy *off-line* and provides *closed-loop* feedback solution for stochastic systems. However, the exorbitant computation of standard DP solution algorithms has been considered largely unrealistic for practical control problems. Meanwhile, Artificial Intelligence and Machine Learning communities have explored approximate DP algorithms, which are collectively known as Reinforcement Learning and Neuro-Dynamic Programming. Their main idea is to use simulation or operation data with a function approximator to solve DP in a computationally amenable manner. Owing to the significant disparity of problem formulations and objectives, the algorithms and techniques available from these fields are not directly applicable to process control problems.

This thesis develops approximate DP (ADP) strategies suitable for process control problems aimed at overcoming the limitations of MPC. The suggested approach identifies the relevant regions of state space by performing closed-loop simulations with judiciously chosen control policies. To deal with continuous variables, we employ a function approximation scheme and solve the Bellman equation in an iterative manner. The rationale is that only a very small fraction of the state space would be relevant for optimal control calculation. The advantages are that an improved control policy from starting ones is derived off-line, a multi-stage on-line optimal control problem is reduced to a single-stage one, and uncertainties can be handled in a more rigorous and convenient way.

A critical issue of the suggested method, however, is how to choose and design the function approximator properly. A comparative study on the choice of function approximators is provided to show that a family of local approximators is adequate for approximating the

'cost-to-go' function. Though the local approximators show "stable" off-line learning, undue extrapolations should be guarded against to have a guaranteed performance. A penalty function method is proposed to prevent the unreasonable predictions of cost-to-go values. The penalty function systematically adjusts "risky" estimates of cost-to-go by considering local density of training data.

The thesis also demonstrates versatility of the proposed ADP strategy with difficult process control problems. First, a stochastic adaptive control problem is presented. In this application an ADP-based control policy shows an "active" probing property to reduce uncertainties, leading to a better control performance. The second example is a dual-mode controller, which is a supervisory scheme that actively prevents the progression of abnormal situations under a local controller at their onset. Finally, two ADP strategies for controlling nonlinear processes based on input-output data are suggested. They are model-based and model-free approaches, and have the advantage of conveniently incorporating the knowledge of identification data distribution into the control calculation with performance improvement.

CHAPTER I

INTRODUCTION

The objective of this thesis is to develop approximate dynamic programming (ADP) strategies that are intended to overcome several outstanding problems associated with current control strategies based on mathematical/empirical modeling and on-line optimization, the most celebrated of which is model predictive control (MPC). The proposed approaches, in essence, use closed-loop simulation data and function approximation to circumvent the computational obstacle, so called 'curse-of-dimensionality' of the conventional algorithms for dynamic programming (DP). The main advantages of the new framework over MPC are: starting control policies can be improved in an evolutionary manner, on-line computation can be potentially reduced, and known uncertainties in a process model can be handled more rigorously. The work of this thesis is based on the ideas developed in the fields of Artificial Intelligence (AI) and Machine Learning (ML) – referred to by various names, such as Neuro-Dynamic Programming (NDP) and Reinforcement Learning (RL). Since the characteristics and requirements of their common applications considerably differ from those of the process control problems, these approximate methods should be understood and interpreted carefully from the viewpoint of process control before they can be considered for real process control problems. Hence, this thesis brings forward novel ADP methods designed for process control problems, addresses practical issues to make the suggested framework "reliable" to use, and presents important process control applications that can be tackled by the ADP methods.

1.1 Motivation

Model predictive control (MPC) is the currently accepted advanced control technique for the process industry owing to its ability to handle complex multivariable control problems with constraints. Typically, a dynamic model is used to build a prediction of future output

behavior, based on which an on-line optimization finds a sequence of input moves that minimize output deviation from a set-point trajectory. Since the first appearance of industrial MPC applications in the late 70s [97, 34], the last two decades of intensive research has brought sound theories and fundamental understandings of its behavior, and has spawned a myriad of design methods that guarantee stability and certain optimality properties [82, 76].

Despite this, there remain two important issues for MPC, which are both theoretical and practical in nature. The first is the potentially exorbitant on-line computation needed to calculate the optimal control moves at each sample time. This issue is particularly relevant when the underlying model of a system is large in dimension, demands the use of long prediction/control horizons, and is nonlinear or hybrid in nature [45, 17, 82]. The resulting optimization problem to be solved on-line is a large-scale nonlinear program or mixed integer program, which still presents nontrivial computational challenges despite all the advances made in computational hardware and numerical methods. The second issue is the MPC's inability to take into account the future interplay between uncertainty and estimation in the optimal control calculation [60, 31]. The problem the conventional MPC solves at each sample time is a *deterministic* open-loop optimal control problem, which thus ignores the uncertainty and feedback at future time points. MPC tries to address the issue of uncertainty by updating the prediction equation based on fresh measurements and re-solving the optimization on a moving window at each sample time, which is referred to as receding horizon control implementation. This approach, however, is inherently suboptimal for the problems that involve uncertainties and feedback.

Both issues can be addressed by the approach of (stochastic) dynamic programming (DP) in principle [15]. DP is a *closed-loop* formulation that derives an optimal control policy *off-line* for multi-stage dynamic optimization problems. The 'cost-to-go' function in DP can be used to reduce a multi-stage problem into an equivalent single stage problem, thus reducing the on-line computational load dramatically. Also, in stochastic DP, the cost-to-go function is calculated with respect to the information vector (sometimes called 'hyperstate') to reflect the effect of uncertainty on the future costs under the optimal *feedback* control

[8, 9, 62, 24]. The proper accounting of the uncertainty results in a control policy with several desirable properties, like cautiousness and active reduction of uncertainty (i.e. active probing) according to its importance for future control performance. Though successfully used to derive optimal control policies for simple problems such as linear quadratic (Gaussian) optimal control, the DP approach has been considered largely impractical because analytical solution is seldom possible and numerical solution via discretization and interpolation suffers from what is referred to as 'curse-of-dimensionality' [15].

1.2 Issues in Applying Existing ADP Methods

Conventional DP approaches necessitate solving the 'Bellman equation' for every possible state through discretization of entire state space. These strategies will find exponential growth in the computation with respect to the state dimension, resulting in excessive computational and storage requirements. Whereas the process control community concluded DP to be impractical early on, researchers in the fields of ML and AI began to explore the possibility of applying the theories of psychology and animal learning to solving DP in an approximate manner in the 1980s [119, 116]. The research areas related to the general concept of programming agents by "reward and punishment without specifying how the task is achieved" have been known as Reinforcement Learning (RL) [51, 120]. It has spawned a plethora of techniques to teach an agent to learn cost or utility of taking actions given a state of the system. The connection between these techniques and the classical dynamic programming was elucidated by Bertsekas and Tsitsiklis [25, 131], who coined the term Neuro-Dynamic Programming (NDP) because of the popular use of artificial neural networks (ANNs) as the function approximator.

Typical RL algorithms improve the cost-to-go function on-line by trial-and-error in a continual manner. For example, human controls a robot randomly to explore the state space in the early phase. They also assume that the environment does not change, which reduces the dimension of a concerned state space. On the other hand, NDP is more of an off-line based learning [25, 20], and its basic assumption is that large amounts of data can be collected from simulation trajectories obtained with "good" suboptimal policies.

Their common update rule, however, is based on the incremental 'temporal-difference' type learning, which is difficult to apply to continuous state variables. In addition, complex dynamics of most chemical processes would limit the amount of data, whereas the NDP or related algorithms require huge amounts of data [23]. Despite various approximate methods from RL/NDP communities, their applicability to process control problems is limited by the following disparities:

1. **Continuous state and action spaces**: Infinite number of state and action values are common in process control problems due to their continuous nature. Furthermore, the number of state variables is generally large. In this case, discretization and the common "incremental" update rule are not practical approaches. Function approximation should also be used with a caution, because approximation errors can grow quickly.

2. **Costly on-line learning**: Real-world-experience-based learning, which is the most prevalent approach in RL, is costly and risky for process control problems. For example, one cannot operate a chemical reactor in a random fashion without any suitable guidelines to explore state space and gather data. Consequently, off-line learning using simulation trajectories should be preferred to on-line learning. Furthermore, one should also exercise a caution in implementing on-line control by insuring against "unreliable" control actions calculated from only a partially learned cost-to-go function.

3. **Limited data quantity**: Though large amounts of simulation data can be collected for off-line learning, complex dynamics of most chemical processes still limit the state space that can be explored, leading to regions of sparse data. This limits the range over which the learned cost-to-go function is valid. Thus, learning and using of the cost-to-go function should be done cautiously by guarding against unreasonable extrapolations.

In summary, an adequate ADP approach for process control problems should be able to provide a reliable control policy given limited coverage of continuous state and action spaces by training data.

1.3 Outline of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we compare the conventional MPC formulation with that of DP to bring out the key advantages of the latter. A brief overview of DP formulation is provided along with traditional algorithms for solving it. Then, we review popular approximate approaches for solving DP, mainly found in the RL and NDP literature. Though the algorithms can be extended to any type of Markov processes in principle, typical setup of their concerned problems are discrete states and actions formulated with a probabilistic model, for which the algorithms have been developed accordingly. They are incrementally updating the cost-to-go function with the state trajectory obtained from experiments or simulations. The update scheme assumes that many trial-and-errors can be allowed for refinement of cost-to-go function and multiple realizations of control policies to visit a same state with different control actions. It turns out that these learning strategies are not adequate in the context of process control problems.

In Chapter 3, we suggest an ADP framework for process control applications. The proposed framework is based on 'value or policy iteration.' The suggested approach shares the commonality with the aforementioned techniques in that the simulation and function approximation are involved in solving the Bellman equation. Suboptimal control policies are judiciously chosen to identify a relevant envelope of the state space, and a function approximator is employed to generalize cost-to-go function over a continuous state space. A Van de Vusse reaction example illustrates the efficacy of the approach over the conventional MPC.

Most chemical process control problems have a high dimensional state space, sparse data distribution due to complex nonlinear dynamics, and constraints on the state variables, leading to a "stiff" cost-to-go structure. In these typical cases, proper choice of approximation structures is shown to be crucial for the proposed approach to be successful. In Chapter 4, we show that a family of local approximators is a preferred choice for approximating cost-to-go function by comparing a feedforward neural network (parametric global approximator) and a k-nearest neighborhood estimator (local "averagers") through several benchmark examples. It is found that global and parameterized approximators, such as a neural network,

can amplify approximation errors in an unpredictable manner during the off-line iteration step, whereas the local averagers show proper convergence behavior. Despite the advantageous properties of the local averagers, we demonstrate that the optimizer may push the solution to other regions of inadequate data density based on a cost-to-go value from the approximator that is not trustworthy. This result implies that quantifying the accuracy of a cost-to-go estimate based on the local data density and incorporating it into the control move optimization is essential for the success of the suggested ADP approach.

In Chapter 5, we propose a penalty function method to design a robust approximation structure for both off-line learning and on-line implementation. We quantify the confidence in the cost-to-go approximation based on the local data density, which is calculated from a nonparametric probability density estimator. The measure of confidence is used to define a 'risk' term, which is included in the objective function to discourage the controller from venturing into an unexplored envelope of the state space. This add-on feature is naturally compatible with the local approximation scheme.

Throughout Chapters 6 to 8, we present applications of the ADP strategy to the process control problems that have been considered difficult to tackle using formerly-existing control algorithms. In Chapter 6, a dual adaptive control problem is studied, where estimation quality and control performance are interlaced together. The optimal controller is known to be 'dual,' meaning it balances between control and "active" exploration. Chapter 7 discusses a dual-mode controller, which designs a nonlinear override controller to improve the performance of a local linear controller. The higher-level nonlinear controller monitors the dynamic state of the system under the local controller and sends an override control action whenever the system is predicted to move outside an acceptable operating regime under the local controller. In Chapter 8, we present input-output data-driven control schemes for nonlinear processes. A major difficulty in using an empirical model based on input-output identification data is the potential over-extrapolation of the model in the optimal control calculation step, leading to large mismatches between the actual closed-loop performance and that predicted by the model. Proposed ADP strategies handle the issue conveniently, as well as improve starting control policies dramatically.

Finally, Chapter 9 summarizes the contributions made by this thesis and discusses their relevance and possible directions for further work.

CHAPTER II

PRELIMINARIES

We identify the intrinsic disadvantages of a typical MPC formulation and then discuss dynamic programming (DP) as an alternative framework. We also give an overview of both conventional DP algorithms and some popular approximate techniques from Reinforcement Learning (RL) and Neuro-Dynamic Programming (NDP) literature.

2.1 MPC – Open-Loop Optimal Feedback Control

Typical MPC algorithms calculate the optimal control actions by minimizing an objective function that penalizes the deviations of the predicted input and output trajectories from their reference trajectories as illustrated in Figure 1. Feedback is updated after the first control action is implemented on the process. This procedure is repeated at each sample time by sliding forward the forecast window, referred to as receding horizon control implementation. At the heart of the MPC formulation is the model, which is used not only to predict the effects of future inputs, but also to estimate the current state of the process given the most recent measurements. A typical process model is described by a set of ordinary differential equations.

$$\begin{aligned} \frac{dx}{dt} &= f_c(x,u,t) \\ y &= g(x,t) \end{aligned} \tag{1}$$

where x is a process state vector, u is an input vector, and y is an output vector.

Since MPC algorithms are implemented on digital computers, a process model is generally represented in a discrete time form.

$$x(k+1) = f(x(k),u(k)) = x(k) + \int_{k\cdot t_s}^{(k+1)\cdot t_s} f_c(x(\tau),u(k),\tau)d\tau \tag{2}$$

where t_s is a sample time, and $u(k)$ is held constant for $k \cdot t_s \leq \tau \leq (k+1) \cdot t_s$. The relation between state variables and outputs remain same with that of the continuous time representation.

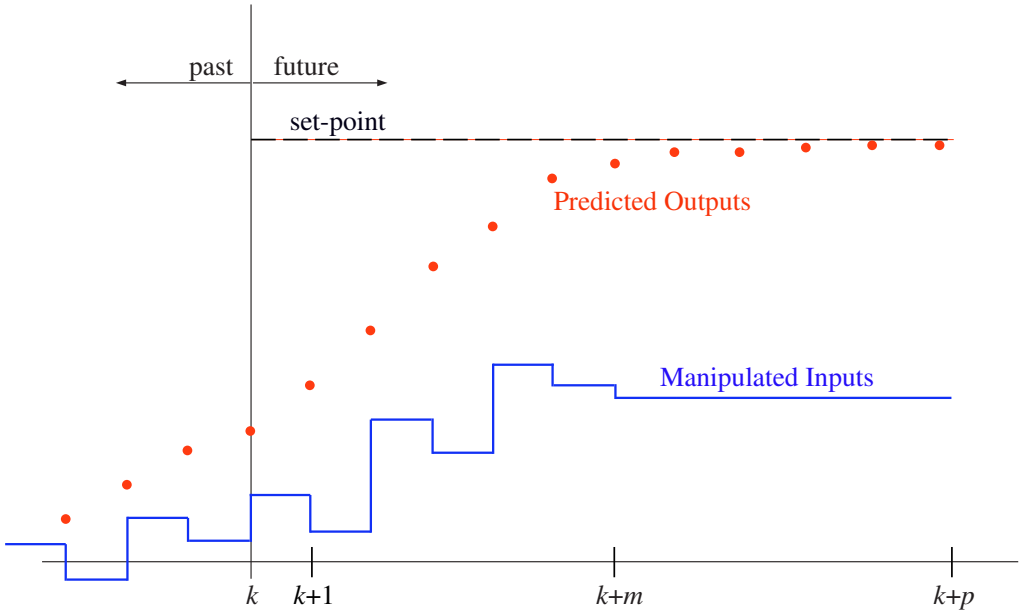

Figure 1: Optimization problem of model predictive control.

Given a process model, the on-line control action is calculated by solving the following open-loop optimal control problem at each sample time after a feedback update of the state:

$$\min_{u(0),\cdots,u(m-1)} \left\{ \left[\sum_{k=0}^{p-1} \phi(x(k), u(k)) \right] + \phi_t(x(p)) \right\} \tag{3}$$

where p is the prediction horizon, m is the control horizon, ϕ is the single-stage cost, and ϕ_t is the cost of the terminal state. A quadratic form is typically used for the single stage cost, and additional constraints such as input and output limits can be added to the above. The above open-loop optimal control problem (OLOCP) implicitly defines a relationship between initial state $x(0)$ and the optimal initial control adjustment $u(0)$, which can be denoted as $u(0) = \mu(x(0))$. μ is a policy that maps the state to the action. Thus $\mu(x(t))$ represents the feedback policy for MPC. As the MPC policy is only implicitly defined through the OLOCP, its implementation requires solving the optimization problem at each sample time on-line with $x(0) = x(t)$ (or $= \hat{x}(t)$, an estimate of $x(t)$) rather than determined off-line.

Because the optimization problem should be solved on-line within a sample time period, a linear model is a favorite choice to make it quadratic program, which can be easily solved with off-the-shelf softwares. The dynamic behavior for most chemical processes,

on the other hand, are rarely linear. Complex nonlinearities are easily introduced through thermodynamics of non-ideal mixtures, higher-order reactions with exponential terms in the Arrhenius equations, etc. Furthermore, logic rules may be incorporated to represent more realistic operations [17]. Though the linear controllers can be detuned to account for these complex dynamics of underlying process, the resulting control policies are often conservative and sometimes are not capable of maintaining a stable operation. For these reasons, MPC integrated with nonlinear/hybrid models is increasingly considered as a required technology. In the implementation of the nonlinear/hybrid MPC, we are faced with the nontrivial task of solving complex mathematical optimization problems like nonlinear programs and mixed integer programs on-line, which can become even more formidable to handle due to the multi-stage formulation of MPC [19, 45, 17]. Then challenge will be on how to shift some of the computational burden to off-line design and also how to devise a systematic way to tradeoff between optimality and reduction of the computational burden – without losing important properties like stability.

Another challenge is the question on how to ensure robustness against known uncertainties. Since a first-principle model is difficult to obtain in general, system identification is performed. This is a time-consuming and difficult task, which leads to modeling errors frequently. In this case, a state estimator is designed to compensate the plant-model mismatch, where the estimation problem and control problem are interlaced as the quality of estimation is affected by control and vice versa. Unfortunately, the formulation based on the OLOCP is fundamentally incapable of addressing this interplay. For example, since sampled data values for the input trajectories are directly optimized in the formulation in a deterministic manner, the ameliorating effect of exploratory input actions on future estimation through the generation of additional signals are not considered. This is true even when the uncertainty information is incorporated explicitly into the OLOCP – to minimize the average cost or the worst-case cost over the bounded parametric uncertainty regions. In summary, the MPC approach of solving the OLOCP repeatedly with feedback updates leads to only suboptimal control in the case of uncertain systems [62, 60].

2.2 Dynamic Programming – Closed-Loop Optimal Feedback Control

Dynamic Programming (DP) offers a unified approach for solving sequential multi-stage optimization problems with or without uncertainties. Bellman's Principle of Optimality [15] states that an optimal state trajectory has the property that no matter how an intermediate state is reached, the rest of the trajectory should coincide with an optimal trajectory calculated with the intermediate point as the starting point. This principle is used to define a so called 'cost-to-go' function, which can provide a *closed-loop* optimal control policy. The cost-to-go of a state is the sum of all single-stage costs that you can expect to incur under a given policy starting from the state, and hence expresses the quality of a state in terms of *future* performance. Given the optimal cost-to-go function, one can easily calculate the optimal action simply by minimizing the sum of the cost of the current state and the cost-to-go of the next state.

2.2.1 Deterministic System

For a deterministic system with a fixed starting state and a deterministic policy, the entire future sequence of states and actions is determined. For any given control policy μ, a finite horizon (N) optimal control problem can be defined as

$$J_N^{\mu}(x) = \left[\sum_{k=0}^{N-1} \phi(x(k), \mu(x(k))) + \phi_t(x(N)) \middle| x(0) = x \right] \quad \forall x \in \mathcal{X} \tag{4}$$

where $\mathcal{X}$ is the set of all possible states.

The optimal cost-to-go function, $J_N^* = J_N^{\mu^*}$, is the cost-to-go function under the optimal policy and is unique.

$$J_N^* = J_N^{\mu^*} = \inf_{\mu} J_N^{\mu} \tag{5}$$

Then the optimal cost-to-go function should satisfy the following recursive equation:

$$J_N^*(x(k)) = \min_{u(k) \in \mathcal{U}} \left[\phi(x(k), u(k)) + J_{N-1}^*(f(x(k), u(k))) \right] \tag{6}$$

where $\mathcal{U}$ is the set of all possible actions. To solve the above optimality equation, sequential calculation of J^* for all states is performed, usually in a backward manner starting from the terminal state with $J_0^*(x(k)) = \phi_t(x(k))$.

With the optimal cost-to-go function for the $N-1$ stage, $J^*_{N-1}(x)$, calculated off-line, one can solve the following *single-stage* optimal control problem, which is equivalent to N-stage problem defined earlier:

$$u(k) = \arg \min_{u(k)\in\mathcal{U}} \left[\phi(x(k), u(k)) + J^*_{N-1}\left(f(x(k), u(k))\right)\right] \tag{7}$$

Most of the important optimal control problems are concerned with minimizing the cost accumulating infinitely:

$$J^\mu_\infty(x) = \left[\sum_{k=0}^{\infty} \alpha^k \phi(x(k), \mu(x(k))) \middle| x(0) = x\right] \quad \forall x \in \mathcal{X} \tag{8}$$

$$J^*_\infty(x) = J^{\mu^*}_\infty = \inf_\mu J^\mu_\infty(x) \quad \forall x \in \mathcal{X} \tag{9}$$

where $\alpha \in [0, 1)$ is a discount factor that handles the tradeoff between immediate and delayed costs. This discount factor can also be introduced in the finite horizon problem. In general, one cannot just find the solution to the infinite horizon problem by taking the limit of the finite horizon solution as $N \to \infty$. This is because one cannot interchange the limit and the min operators. DP takes advantage of the problem's recursive nature and is formulated as the following *Bellman Equation*, which must be solved for all x to obtain the optimal cost-to-go function:

$$J^*_\infty(x) = \min_{u(k)\in\mathcal{U}} \left[\phi(x(k), u(k)) + \alpha J^*_\infty(f(x(k), u(k)))\middle| x(0) = x\right] \quad \forall x \in \mathcal{X} \tag{10}$$

In the rest of the thesis, we will denote J without the subscript of ∞ as a cost-to-go function for infinite horizon problems.

2.2.2 Stochastic System

One benefit of DP is that the method extends very naturally to stochastic problems. Whereas the DP (closed-loop) and the MPC (open-loop) formulations result in the same solution in the case of a deterministic system, the two approaches lead to very different results in the case of a stochastic system. Because the conventional MPC formulation treats the future inputs as deterministic, it represents only a suboptimal feedback strategy. For optimal feedback control, one should solve the following problem:

$$\min_\mu \mathbb{E}\left[\sum_{k=0}^{N-1} \phi(x(k), \mu(\mathcal{I}(k))) + \phi_t(x(N))\right] \tag{11}$$

where $\mathbb{E}$ is an expectation operator, $\mathcal{I}(k)$ is a vector summarizing the information available at the k^{th} time, which typically consists of the parameters defining the conditional probability distribution of the state, e.g. the state estimate and the error covariance matrix for a Gaussian system. It is assumed that there is an underlying equation for dynamic propagation of x and $\mathcal{I}$. Note that the inputs are no longer optimized as deterministic variables as $\mathcal{I}$ is stochastic. The consideration of feedback control gives rise to a stochastic dynamic program, which must be solved in a similarly sequential manner.

The Bellman equation is then defined as

$$J^*(\mathcal{I}(k)) = \min_{u(k)} \mathbb{E}\left[\phi(x(k), u(k)) + \alpha J^*\left(f_{\mathcal{I}}(I(k), u(k))\right) | \mathcal{I}(k)\right] \tag{12}$$

where $f_{\mathcal{I}}$ is the stochastic equation that relates the information vector from one sample time to the next. It is assumed that the equation for recursive calculation of $\mathcal{I}$ is available.

2.3 Conventional DP Algorithms

2.3.1 Dynamic Programming Operator

In this section, we first define the Dynamic Programming Operator (DP Operator) T and show its important properties, which are the fundamentals for the conventional DP algorithms introduced later. We consider a stochastic system defined by the transition function

$$x(k+1) = f(x(k), u(k), \omega(k)) \tag{13}$$

Note that (13) represents a Markov decision process (MDP), meaning the next state $x(k+1)$ depends only on the current state and input not on the states and actions of past times. The model form is quite general in that, if the next state did indeed depend on past states and actions, a new state vector can be defined by including those past variables. Whereas (13) is the typical model form used for process control problems, most operations research (OR) problems have a model described by a transition probability matrix, which describes how the probability distribution (over a finite set of discrete states) evolves from one time step to the next.

The DP operator T is then defined as

$$(TJ)(x(k)) = \min_{u(k)\in\mathcal{U}} \mathbb{E}\left[\phi(x(k), u(k)) + \alpha J(x(k+1))\right] \tag{14}$$

Equation (14) can be recast as

$$(TJ)(x) = \min_{u \in \mathcal{U}} \sum_{x' \in \mathcal{X}} p_{x,x'}(u) \left(\phi(x, u, x') + \alpha J(x') \right) \tag{15}$$

For notational simplicity, we set $x = x(k)$ and $x' = x(k+1)$. $p_{x,x'}(u)$ is the transition probability from x to x' under the control action u. This representation is convenient for proving some important properties of the T as will be shown later. We also define the operator T_μ with respect to a fixed policy μ as

$$(T_\mu J)(x) = \sum_{x' \in \mathcal{X}} p_{x,x'}(\mu(x)) \left(\phi(x, u, x') + \alpha J(x') \right) \tag{16}$$

Now we present some important properties of the DP operator for discounted dynamic programming.

Theorem 1. $J^* = \lim_{N \to \infty} T^N J$

Proof. We consider a more general case where policy can change at each time, $\pi = \{\mu_0, \mu_1, \cdots\}$. Then any cost-to-go function starting with x can be written as

$$J^\pi(x) = \mathbb{E}\left[\sum_{k=0}^{N-1} \alpha^k \phi(x(k), \mu_k(x(k))) \middle| x(0) = x \right] + \mathbb{E}\left[\sum_{k=N}^{\infty} \alpha^k \phi(x(k), \mu_k(x(k))) \middle| x(0) = x \right] \tag{17}$$

The absolute value of the second term is less than $\frac{\alpha^N}{1-\alpha} M$, where M is a constant such that $|\phi(x(k), u(k))| < M$. With

$$(T^N J)(x(0)) = \min_\pi \mathbb{E}\left[\sum_{k=0}^{N-1} \alpha^k \phi(x(k), \mu_k(x(k))) + \alpha^N J(x(N)) \right] \tag{18}$$

We have the following inequalities:

$$\begin{aligned} J^\pi(x(0)) - \frac{\alpha^N}{1-\alpha} M - \alpha^N \|J\|_\infty &\leq \mathbb{E}\left[\sum_{k=0}^{N-1} \alpha^k \phi(x(k), \mu_k(x(k))) + \alpha^N J(x(N)) \right] \\ &\leq J^\pi(x(0)) + \frac{\alpha^N}{1-\alpha} M + \alpha^N \|J\|_\infty \end{aligned} \tag{19}$$

If we minimize each term w.r.t. π,

$$J^{\pi^*}(x(0)) - \frac{\alpha^N}{1-\alpha} M - \alpha^N \|J\|_\infty \leq T^N J \leq J^{\pi^*}(x(0)) + \frac{\alpha^N}{1-\alpha} M + \alpha^N \|J\|_\infty \tag{20}$$

As $N \to \infty$, $\alpha^N \to 0$. Hence, it follows that $J^* = \lim_{N \to \infty} T^N J$. □

Theorem 2. *T is an α-contraction mapping with respect to the infinity norm, i.e., $\|TJ - T\bar{J}\|_\infty \leq \alpha\|J - \bar{J}\|_\infty$ for all $J, \bar{J}$.*

Proof. We use the following property for arbitrary functions $g,\ h : \mathcal{A} \to \mathbb{R}$, where $\mathcal{A}$ is some arbitrary set.

$$\left|\min_a g(a) - \min_a h(a)\right| \leq \max_a |g(a) - h(a)| \tag{21}$$

Using the property we have

$$\begin{aligned} \left|(TJ)(x) - (T\bar{J})(x)\right| &= \left|\min_u \left(\sum_{x' \in \mathcal{X}} p_{x,x'}(u) \left(\phi(x,u,x') + \alpha J(x')\right)\right)\right. \\ &\quad \left. - \min_u \left(\sum_{x' \in \mathcal{X}} p_{x,x'}(u) \left(\phi(x,u,x') + \alpha \bar{J}(x')\right)\right)\right| \\ &\leq \max_u \alpha \sum_{x' \in \mathcal{X}} p_{x,x'}(u) \left|J(x') - \bar{J}(x')\right| \\ &\leq \alpha\|J - \bar{J}\|_\infty \end{aligned} \tag{22}$$

Because $\|TJ - T\bar{J}\|_\infty = \max_x \left|(TJ)(x) - (T\bar{J})(x)\right|$, the above inequality implies that $\|TJ - T\bar{J}\|_\infty \leq \alpha\|J - \bar{J}\|_\infty$. □

Theorem 3. *T has a monotonicity property, i.e., if $J \geq \bar{J}$, then $TJ \geq T\bar{J}$.*

Proof. Suppose $J \geq \bar{J}$, then

$$\sum_{x' \in \mathcal{X}} p_{x,x'}(u) J(x') \geq \sum_{x' \in \mathcal{X}} p_{x,x'}(u) \bar{J}(x') \qquad \forall x \in \mathcal{X},\ \forall u \in \mathcal{U} \tag{23}$$

By multiplying both sides by α and adding the expected single stage cost, we have

$$\sum_{x' \in \mathcal{X}} p_{x,x'}(u) \left(\phi(x,u,x') + \alpha J(x')\right) \geq \sum_{x' \in \mathcal{X}} p_{x,x'}(u) \left(\phi(x,u,x') + \alpha \bar{J}(x')\right) \tag{24}$$

for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$. The above inequality implies that $T_\mu J \geq T_\mu \bar{J}$ for any policy μ. Suppose μ^* satisfies $T_{\mu^*} J = TJ$, then $TJ \geq T_{\mu^*} \bar{J}$ as well as $T_{\mu^*} \bar{J} \geq T\bar{J}$. Therefore $TJ \geq T\bar{J}$. □

Theorem 4. *The sequence $\{T^N J\}$ converges for any J.*

Proof. We assume that $T : \mathbb{R}^n \to \mathbb{R}^n$. Then, it is sufficient to show that $\{T^N J\}$ is a Cauchy

sequence. For any K and $M, N \geq K$,

$$\begin{aligned} \|T^M J - T^N J\|_\infty &= \left\| \sum_{i=M}^{N-1} \left(T^i J - T^{i+1} J\right) \right\|_\infty \\ &\leq \sum_{i=M}^{N-1} \left\| \left(T^i J - T^{i+1} J\right) \right\|_\infty \\ &\leq \sum_{i=M}^{N-1} \alpha^i \|(J - TJ)\|_\infty \\ &\leq \frac{\alpha^K}{1-\alpha} \|(J - TJ)\|_\infty \end{aligned} \tag{25}$$

For any $\epsilon > 0$, one can find K such that

$$\frac{\alpha^K}{1-\alpha} \|(J - TJ)\|_\infty \leq \epsilon \tag{26}$$

Hence, $\{T^N J\}$ is a Cauchy sequence and converges. □

Theorem 5. *T has a unique fixed point.*

Proof. The sequence $\{T^N J\}$ converges to a fixed point of T, which means that there exists at least one fixed point. Suppose there are two fixed points, J_1 and J_2 for T. Then we have $TJ_1 = J_1$ and $TJ_2 = J_2$. Then we have,

$$\|TJ_1 - TJ_2\| = \|J_1 - J_2\| \tag{27}$$

This is contradictory to the α-contraction property of T. Therefore, the fixed point of T is unique. □

From the last two theorems, we can conclude that J^* is the unique solution to the equation

$$J^* = TJ^* \tag{28}$$

which is equivalent to the Bellman equation. It is also clear from the above theorems that T_μ is also a ∞-norm α-contraction mapping. This fact gives the following result.

Theorem 6. *A stationary policy $\pi = \{\mu, \mu, \cdots\}$ is optimal iff $TJ^* = T_\mu J^*$.*

Proof. Suppose that the stationary policy is optimal. Let J^μ be the cost-to-go function under this policy. Then we have $J^* = J^\mu$, and J^μ is the unique solution for $J = T_\mu J$. Consequently, $J^\mu = T_\mu J^\mu \Rightarrow J^* = T_\mu J^* \Rightarrow TJ^* = T_\mu J^*$.

Now suppose $TJ^* = T_\mu J^*$. This implies that $J^* = T_\mu J^*$. Since J^μ is the unique solution of the T_μ operator, $J^* = J^\mu$, so the stationary policy described by μ is optimal. □

This theorem implies that one can always find an optimal stationary policy for a discounted infinite horizon problem [94].

In the following sections, conventional algorithms to find the fixed solution of the Bellman equation are presented. They are either iteratively applying DP operators (T, T_μ) or employing a mathematical programming based approach. All the algorithms are demonstrated for discounted infinite horizon cases. Corresponding algorithms for finite horizon problems are also straightforward to derive but they are not of our interest in the thesis.

2.3.2 Value/Policy Iteration

Value iteration and policy iteration are two classical solution methods for DP using a given model. They form the basis for the various approximate methodologies introduced later.

- Value Iteration

  In value iteration, one starts with an initial guess for the cost-to-go for each state and iterates on the Bellman equation until convergence. This is equivalent to calculating the cost-to-go value for each state by assuming an action that minimizes the sum of the current stage cost and the cost-to-go for the next state according to the current estimate. Hence, each update assumes that the calculated action is optimal, which may not be true given that the cost-to-go estimate is inexact, especially in the early phase of iteration. The algorithm involves the following steps:

  1. Initialize $J^0(x)$ for all $x \in \mathcal{X}$.
  2. For each state x

     $$J^{i+1}(x) = \min_u \mathbb{E}\left[\phi(x,u) + \alpha J^i(\hat{x})\right] \tag{29}$$

     where $\hat{x} = f(x,u,\omega) \in \mathcal{X}$, and i is the iteration index.
  3. Perform the above iteration (step 2) until $J(x)$ converges.

  The update rule of (29) is called *full backup* because the cost-to-go values of the entire states are updated in every round of update.

- Policy Iteration

Policy iteration is a two-step approach composed of *policy evaluation* and *policy improvement*. Rather than solve for a cost-to-go function directly and then derive an optimal policy from it, the policy iteration method starts with a specific policy and the policy evaluation step computes the cost-to-go values under that policy. Then the policy improvement step tries to build an improved policy based on the previous policy. The policy evaluation and the policy improvement steps are repeated until the policy no longer changes. Hence, this method iterates on policy rather than the cost-to-go function.

The policy evaluation step iterates on the cost-to-go values but with the actions dictated by the given policy. Each evaluation step can be summarized as follows:

1. Given a policy μ, initialize $J^{\mu}(x)$ for all $x \in \mathcal{X}$.
2. For each state x, find the fixed point solution of T_{μ} according to

$$J^{\mu,\ell+1}(x) = \mathbb{E}\left[\phi(x, \mu(x)) + \alpha J^{\mu,\ell}(\hat{x})\right] \tag{30}$$

   where ℓ is the iteration index of policy evaluation step.
3. The above iteration (step 2) continues until $J^{\mu}(x)$ converges.

The overall policy iteration algorithm is given as follows:

1. Given an initial control policy μ^0, set $i = 0$.
2. Perform the policy evaluation step to evaluate the cost-to-go function for the current policy μ^i
3. The improved policy is represented by

$$\mu^{i+1}(x) = \arg\min_{u} \mathbb{E}\left[\phi(x, u) + \alpha J^{\mu^i}(\hat{x})\right] \tag{31}$$

   Calculate the action given by the improved policy for each state.
4. Iterate steps 2 and 3, and stop until $\mu(x)$ converges.

Figure 2 depicts the alternating feature of policy iteration algorithm.

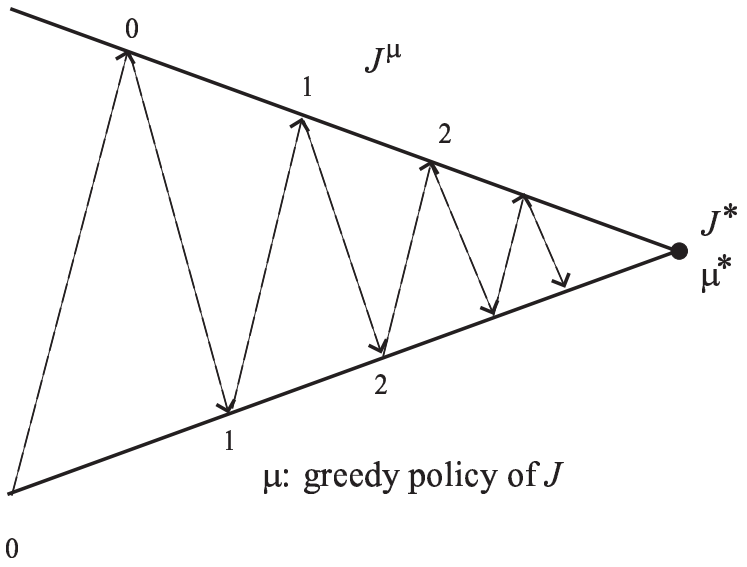

Figure 2: Graphical metaphor of policy iteration algorithm.

For systems with a finite number of states, both the value iteration and policy iteration algorithms converge to an optimal policy [15, 49, 24]. Whereas policy iteration requires complete policy evaluation between steps of policy improvement, each evaluation often converges in just few iterations because the cost-to-go function typically changes very little when the policy is only slightly improved. At the same time, policy iteration generally requires significantly fewer policy improvement steps than value iteration because each policy improvement is based on accurate cost-to-go information [94].

One difficulty associated with the value or policy iteration is that the update is performed after one "sweep" of an entire state set, making it prohibitively expensive for most problems. To avoid this difficulty, *asynchronous* iteration algorithms have been proposed [23, 22]. These algorithms do not back up the values of states in a strict order but use whatever updated values available. The values of some states may be backed up several times while the values of others are backed up once. However, obtaining optimal cost-to-go values requires infinite number of times update in general.

2.3.3 Linear Programming Based Approach

Another approach to solving DP is to use a linear programming (LP) formulation. The Bellman equation can be characterized by a set of linear constraints on the cost-to-go function. The optimal cost-to-go function can then be derived by solving the following LP [72, 38, 47, 26]:

$$\begin{aligned} \max \quad & \textstyle\sum_{x \in \mathcal{X}} J \\ \text{s.t.} \quad & TJ \geq J \end{aligned} \tag{32}$$

Since the standard LP formulation does not allow for the min operator of (32), it is translated into the following set of constraints:

$$\mathbb{E}\left[\phi(x,u) + \alpha J(\hat{x})\right] \geq J(x) \qquad \forall u \in \mathcal{U} \tag{33}$$

Note that the LP approach also leaves us with the same 'curse-of-dimensionality.' Moreover, the number of constraints in the above can be problematic as the number of possible actions is growing.

Theorem 7. *The above LP problem gives a unique solution of J^*.*

Proof. If J is feasible solution to the LP, then $TJ \geq J$. By the monotonicity property of T,

$$J \leq TJ \leq T^2 J \leq \cdots \leq J^* \tag{34}$$

Hence, any feasible J satisfies $J \leq J^*$. Because J^* is a fixed solution of T, maximizing the sum solves for J^* and it is unique. □

The LP approach is the only known algorithm that can solve DP in polynomial time, and recent years have seen substantial advances in algorithms for solving large-size linear programs. However, theoretically efficient algorithms have still been shown to be ineffective or even infeasible for practically-sized problems [51, 120].

2.4 Review of Approximate Methods for Dynamic Programming

In this section, we give an overview of popular approximation techniques for solving DP developed from the AI and ML fields. We first discuss the representation of state space,

and then review different approximate DP algorithms, which are categorized into model-based and model-free methods. The most striking feature shared by all the approximate DP techniques is the synergetic use of *simulations* (or interactive experiments) and *function approximation*. Instead of trying to build the cost-to-go function for an entire state space, they use sampled trajectories to identify parts of the state space relevant to optimal or "good" control where they want to build a solution and also obtain an initial estimate for the cost-to-go values.

2.4.1 State Space Representation

Typical MDPs have either a very large number of discrete states and actions or continuous state and action spaces. Computational obstacles arise from the large number of possible state/action vectors and the number of possible outcomes of the random variables. The 'curse-of-dimensionality' renders the conventional DP solution approach through exhaustive search infeasible. Hence, in addition to developing better learning algorithms, substantial efforts have been devoted to alleviating the curse-of-dimensionality through more compact state space representations. For example, state space quantization/discretization methods have been used popularly in the context of DP [12] and gradient descent technique [113]. The discretization/quantization methods have been commonly accepted because the standard RL/NDP algorithms were originally designed for systems with discrete states. The discretization method should be chosen carefully, however, because incorrect discretization could severely limit the performance of a learned control policy, for example, by omitting important regions of the state space and/or by affecting the original Markov property [84].

More sophisticated discretization methods have been developed based on adaptive resolutions such as the multi-level method [98], clustering-based method [57], triangularization method [84, 85], state aggregation [21], and divide-and-conquer method (Parti-Game algorithm) [80]. The parti-game algorithm, which is one of the most popular discretization strategies, attempts to search for a path from an initial state to a goal state (or region) in a multi-dimensional state space based on a coarse discretization. When the search fails, the resolution is iteratively increased for the regions of the state space where the path planner

is unsuccessful. Though some adaptive discretization methods can result in a better policy compared to the fixed versions [7], they can potentially suffer from their own 'curse-of-dimensionality' and become less reliable when the estimation of cost-to-go is necessary for the states lying in a smaller envelope than that of converged partitions.

Function approximation methods have also been employed to express the correlation between cost-to-go and system state, either by based on parametric structures (e.g. ANNs) or 'store-and-search' based nonparametric methods (e.g. nearest neighbor). The function approximation based approaches are more general because they are applicable to both finite and infinite number of states without modification of a given problem. The current status and the issues of incorporating function approximators into approximate strategies will be discussed separately in Chapters 4 and 5.

2.4.2 Model-Based Methods

If there exists a model that describes a concerned system, the main question becomes how to solve the Bellman equation efficiently. Given an exact model, a conceptually straightforward method is to use the value or policy iteration algorithm. A family of methods in which a model built from data is used to derive a control policy as if it were an exact representation of the system is called 'certainty-equivalence' approach, which is similar to the concept for process identication/control involving *learning phase* and *acting phase* [59]. We note that random exploration for gathering data to build such a model is much less efficient than using a policy-interlaced exploration [136, 53].

Independently from the researchers working on direct DP solution methods, Werbos proposed a family of adaptive critic designs (or actor-critic methods (AC) as named later in [14]) in the late 70s [134]. He extended the work and collectively called the approach *Heuristic Dynamic Programming* [135]. The purpose of the adaptive critic design is to learn optimal control laws by successively adapting two ANNs, namely, an action network and a critic network. These two ANNs indirectly approximate the Bellman equation. The action network calculates control actions using the performance index from the critic network. The critic network learns to approximate the cost-to-go function and uses the output of

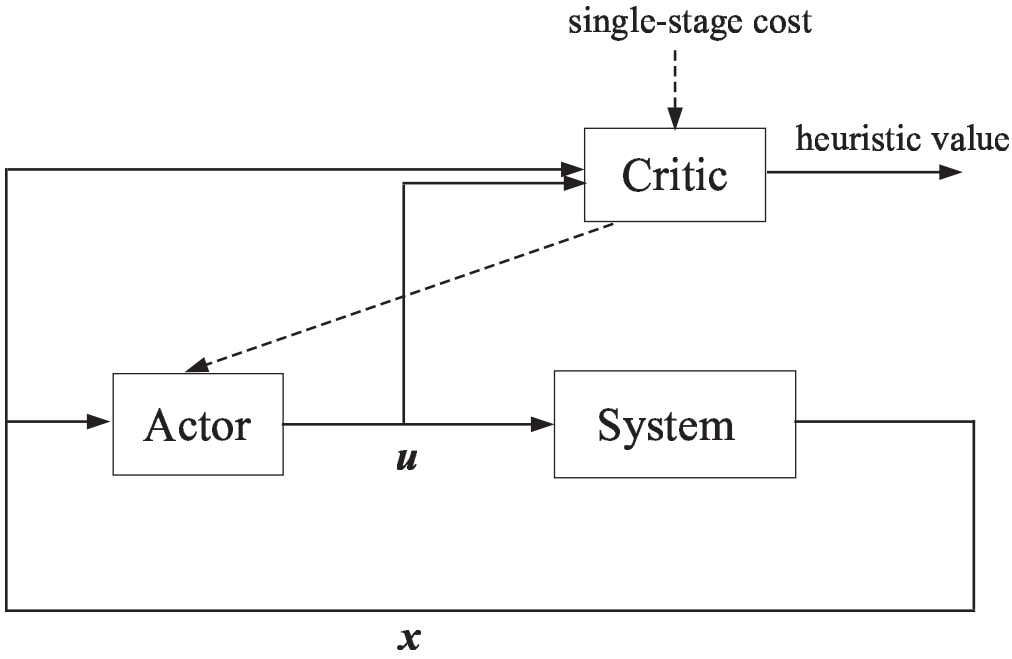

Figure 3: The actor-critic architecture.

the action network as one of its inputs, either directly or indirectly. This structure has been used as a "policy learner" in conjunction with many RL schemes in addition to being a popular structure for neuro-fuzzy-controllers [93]. The AC algorithms are less suited to cases where the data change frequently since the training of the networks is challenging and time-consuming. We note that the AC framework is not limited to the model-based learning scheme, and it has also been used as a framework for model-free learning. The general structure of AC is shown in Figure 3. RL literature has considered the model-based learning an alternative way to use gathered data efficiently during interactive learning with an environment, compared to a class of model-free learning schemes that will be introduced in the next section. They have been more interested in 'exploration through trial-and-error' to increase the search space, for example, in a robot-juggling problem [104]. Hence, most model-based approaches from the RL literature have been designed to learn an explicit model of a system simultaneously with a cost-to-go function and a policy [114, 115, 81, 92, 13]. The general algorithms iteratively 1) update the learned model, 2) calculate control actions that optimize the given cost-to-go function with the current learned model, 3) update

the corresponding cost-to-go function as in value iteration, and 4) execute the control policy and gather more data. Representative algorithms in this class are Dyna and RTDP (real-time dynamic programming) [114, 13]. These model-based interactive learning techniques have the advantage that they can usually find good control actions with fewer experiments since they can exploit the existing samples better by using the model [10].

2.4.3 Model-Free Methods

RL/NDP and other related research work have been mainly concerned with the question of how to obtain an optimal policy when a model is not available. This is mainly because the state transition rule of their concerned problems is described by a probability transition matrix, which is difficult to identify empirically. Many trial-and-errors, however, allow one to find the optimal policy eventually. These "on-line planning" methods have an agent interact with its environment directly to gather information (state and action vs. cost-to-go) from on-line experiment or in simulations. In this section, three important model-free learning frameworks are introduced – Temporal Difference (TD) learning, Q-learning, and SARSA, all of which learn the cost-to-go functions *incrementally* based on experiences with the environment.

2.4.3.1 Temporal Difference Learning

TD learning is a passive learner in that one calculates the cost-to-go values by operating an agent under a *fixed* policy. For example, we watch a robot wander around using its current policy μ to see what cost it incurs and which states it explores. This was suggested by Sutton and is known as the TD(0) algorithm [117]. The general algorithm is as follows:

1. Initialize $J(x)$.

2. Given a current policy ($\mu^i(x)$), let an agent interact with its environment, for example, let an agent (e.g. robot, controller, etc.) perform some relevant tasks.

3. Watch the agent's actions given from $\mu^i(x)$ and obtain a cost $\phi(x, \mu^i(x))$ and its successor state $\hat{x}$.

4. Update the cost-to-go using

$$J(x) \longleftarrow (1-\gamma)J(x) + \gamma\left\{\phi(x,\mu^i(x)) + \alpha J(\hat{x})\right\} \tag{35}$$

or equivalently,

$$J(x) \longleftarrow J(x) + \gamma\left\{\phi(x,\mu^i(x)) + \alpha J(\hat{x}) - J(x)\right\} \tag{36}$$

where γ is a learning rate from 0 to 1. The higher the γ, the more we emphasize our new estimates and forget the old estimates.

5. Set $x \longleftarrow \hat{x}$ and continue experiment.

6. If one sweep is completed, return to step 2 with $i \longleftarrow i+1$, and continue the procedure until convergence.

Whereas TD(0) update is based on the "current" difference only, a more general version called TD(λ) updates the cost-to-go values by including the temporal differences of the later states visited in a trajectory with exponentially decaying weights.

Suppose we generated a sample trajectory, $\{x(0),\ x(1),\ \cdots,\ x(t),\ \cdots\}$. The temporal difference term, $d(t)$, at time t is given by

$$d(t) = \phi(x(t),\mu(x(t))) + \alpha J(x(t+1)) - J(x(t)) \tag{37}$$

Then the policy evaluation step for a stochastic system is approximated by

$$J(x(t)) \longleftarrow J(x(t)) + \gamma\sum_{m=t}^{\infty}\lambda^{m-t}d(m) \tag{38}$$

where $0 \leq \lambda < 1$ is a decay parameter. Within this scheme, a single trajectory can include a state, say x, multiple times, for example, at times $t_1,\ t_2,\ \cdots, t_M$. In such a case, 'every-visit' rule updates the cost-to-go whenever the state is visited in the trajectory according to

$$J(x(t)) \longleftarrow J(x(t)) + \gamma\sum_{j=1}^{M}\sum_{m=t_j}^{\infty}\lambda^{m-t_j}d(m) \tag{39}$$

Graphical representation of the addition of update terms is shown in Figure 4. A cor-

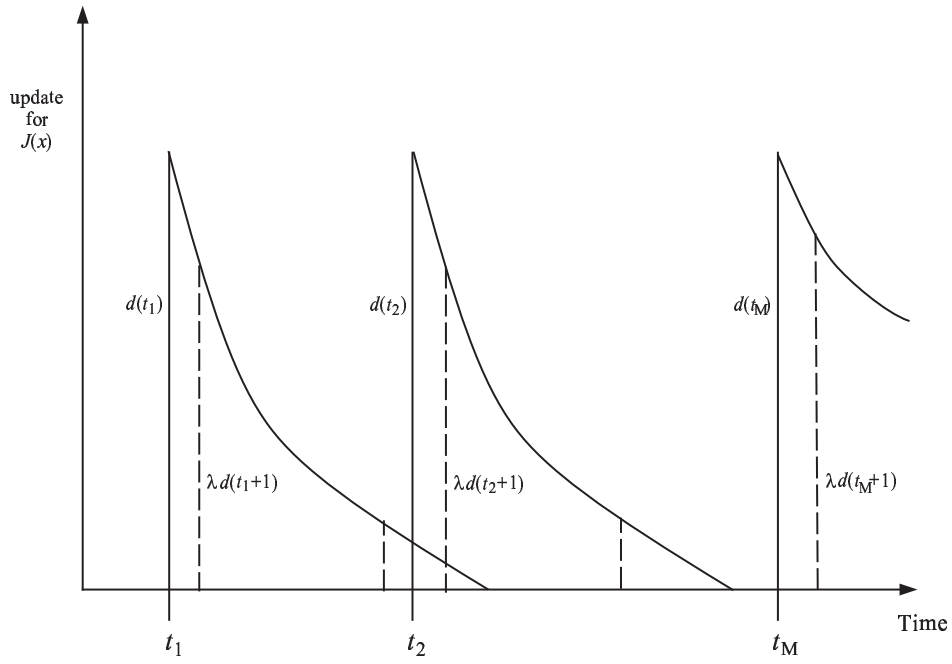

Figure 4: Cumulative addition of temporal difference terms in the every-visit method.

responding *on-line* update rule for the every-visit method is given by

$$J(x(t)) \longleftarrow J(x(t)) + \gamma \left\{ \phi(x(t), \mu^i(x(t))) + \alpha J(x(t+1)) - J(x(t)) \right\} e_t(x(t)) \tag{40}$$

where t is the time index in a single sample trajectory, and $e_t(x)$ is 'eligibility,' with which each state is updated. Note that the temporal difference term, $\{\cdot\}$ of (40), appears only if x has already been visited in a previous time of the trajectory. Hence, all eligibilities start out with zeros and are updated at each time t as follows:

$$e_t(x) = \begin{cases} \alpha\lambda e_{t-1}(x) & \text{if} \quad x \neq x(t) \\ \alpha\lambda e_{t-1}(x) + 1 & \text{if} \quad x = x(t) \end{cases} \tag{41}$$

where α is the discount factor for the cost-to-go. The eligibility thereby puts more emphasis on the temporal difference term in recent past. Singh and Sutton [108] proposed an alternative version of the eligibility assignment algorithm, where visited states in the most recent sample run always get an eligibility of unity rather than an increment of 1, which they called the 'first-visit' method.

Since the TD learning is based on a fixed policy, it can be combined with an AC-type policy-learner. The convergence properties of the AC-related algorithms were explored

[137]. The convergence property of TD(λ) learning was also studied by several researchers [35, 91, 129].

2.4.3.2 Q-learning

Q-learning [133, 132] is an active learner in that one modifies the 'greedy' policy as the agent learns. One can also tweak the policy to try different control actions from the calculated policy even when the agent is interacting with a real environment. For example, injection of random signals into actions is often carried out for exploration of the state space. Optimal Q-value is defined as the cost-to-go value of implementing a specific action u at state x, and then following the optimal policy from the next time step on. Hence, the optimal Q-function satisfies the following equation:

$$\begin{aligned} Q^*(x,u) &= \mathbb{E}\left[\phi(x,u) + \alpha \min_{\hat{u}} Q^*(\hat{x},\hat{u})\right] \\ &= \mathbb{E}\left[\phi(x,u) + \alpha J^*(\hat{x})\right] \end{aligned} \tag{42}$$

This also gives a recursive relation for the Q-function as does the Bellman equation for the 'J-function.' Once the optimal Q-function is known, the optimal policy $\mu^*(x)$ can be easily obtained by

$$\mu^*(x) = \arg\min_u Q^*(x,u) \tag{43}$$

The on-line incremental learning of the Q-function is similar to the TD-learning:

1. Initialize $Q(x,u)$.

2. Let an agent interact with its environment by solving (43) using the current approximation of Q instead of Q^*. If there are multiple actions giving a same level of performance, select an action randomly.

3. Update the Q values by

$$Q(x,u) \longleftarrow (1-\gamma)Q(x,u) + \gamma\left\{\phi(x,u) + \alpha \min_{\hat{u}} Q(\hat{x},\hat{u})\right\} \tag{44}$$

4. Set $x \longleftarrow \hat{x}$ and continue experiment.

5. Once a loop is completed, repeat from step 2 until convergence.

If one performs the experiment infinite times, the estimates of Q-function converge to $Q^*(x,u)$ with proper decaying of the learning rate γ [133, 50]. Greedy actions may confine the exploration space, especially in the early phase of learning, leading to failure in finding the optimal Q-function. To explore the state space thoroughly, random actions should be carried out on purpose. As the solution gets improved, the greedy actions are implemented. This randomization of control actions is similar to the simulated annealing technique used for global optimization.

2.4.3.3 SARSA

SARSA [99] also tries to learn the state-action value function (Q-function). It differs from Q-learning with respect to the incremental update rule. SARSA does not assume that the optimal policy is imposed after one time step. Instead of finding a greedy action, it assumes a fixed policy as does the TD learning. The update rule then becomes

$$Q(x,u) \longleftarrow Q(x,u) + \gamma \left\{ \phi(x,u) + \alpha Q(\hat{x}, \mu^i(\hat{x})) - Q(x,u) \right\} \tag{45}$$

A policy learning component like an AC scheme can also be combined with this strategy.

2.4.4 Applications of Approximate DP Methods

In this section, we briefly review some of the important applications of the approximate DP methods, mainly RL and NDP methods. Important OR applications are reviewed in [25, 120]. We classify the previous work by application areas.

2.4.4.1 Operations Research

The application area that benefited the most from the RL/NDP theory is game playing. Samuel's checker player was one of the first applications of DP in this field and used linear function approximation [101, 102]. One of the most notable successes is Tesauro's backgammon player, TD-Gammon [122, 123, 124]. It is based on TD methods with approximately 10^{20} states. To handle the large number of state variables, ANN with a simple feedforward structure was employed to approximate the cost-to-go function, which maps a board position to the probability of winning the game from the position. Two versions of learning

were performed for training the TD-Gammon. The first one used a very basic encoding of the state of the board. The advanced version improved the performance significantly by employing some additional human-designed features to describe the state of the board. The learning was done in an evolutionary manner over several months – playing against itself using greedy actions without exploration. TD-Gammon successfully learned to play competitively with world-champion-level human players. By providing large amounts of data frequently and realizing the state transitions in a sufficiently stochastic manner, TD-Gammon could learn a satisfactory policy without any explicit exploration scheme. No comparable successes to that of TD-Gammon have been reported in other games yet, and there are still open questions regarding how to design experiments and policy update in general [105, 125].

Another noteworthy application was in the problem of elevator dispatching. Crites and Barto [32, 33] used Q-learning for a complex simulated elevator scheduling task. The problem is to schedule four elevators operating in a building with ten floors. The objective is to minimize the discounted average waiting time of passengers. The formulated discrete Markov system has over 10^{22} states even in the most simplified version. They also used a neural network and the final performance was slightly better than the best known algorithm and twice as good as the policy most popular in real elevator systems. Other successful RL/NDP applications in this field include large-scale job-shop scheduling [142, 141, 143], cell-phone channel allocation [107], manufacturing [71], and finance applications [86].

2.4.4.2 Robot Learning

Robot learning is a difficult task in that it generally involves continuous state and action spaces, similar to process control problems. Barto *et al.* [14] proposed a learning structure for controlling a cart-pole system (inverted pendulum) that consisted of an associative search system and an adaptive critic system. Anderson [5] extended this work by training ANNs that learned to balance a pendulum given the actual state variables of the inverted pendulum as input with state space quantization for the evaluation network.

Schaal and Atkeson [104] used a nonparametric learning technique to learn the dynamics

of a two-armed robot that juggles a device known as "devil-stick." They used task-specific knowledge to create an appropriate state space for learning. After 40 training runs, a policy capable of sustaining the juggling motion up to 100 hits was successfully obtained. A non-parametric approach was implemented to generalize the learning to unvisited states in the algorithm. This work was later extended to learn a pendulum swing-up task by using human demonstrations [11, 103]. In the work, however, neither parametric nor nonparametric approach could learn a task of balancing the pendulum reliably due to poor parametrization and insufficient information for important regions of the state-space, respectively.

We note that most robots used in assembly and manufacturing lines are trained in such a way that a human guides the robot through a sequence of motions that are memorized and simply replayed. Mahadevan and Connell [70] suggested a Q-learning algorithm with a clustering method for tabular approach to training a robot performing a box-pushing task. The robot learned to perform better than a human-programmed solution when a decomposition of sub-tasks was done carefully. Lin [67] used an ANN-based RL scheme to learn a simple navigation task. Asada *et al.* [7] designed a robot soccer control algorithm with a discretized state space based on some domain knowledge. Whereas most robot learning algorithms discretized the state space [121], Smart and Kaelbling [111] suggested an algorithm that deals with continuous state space in a more natural way. The main features are that approximated Q-values are used for training *neighboring* Q-values, and that a hyper-elliptic hull is designed to prevent extrapolation.

2.4.4.3 Process Control

After Bellman's publication, some efforts were made to use DP to solve various deterministic and stochastic optimal control problems. However, only a few important results could be achieved through analytical solution, the most celebrated being the LQ optimal controller [140]. This combined with the limited computing power available at the time caused most control researchers to abandon the approach. As the computing power grew rapidly in the 1980s, some researchers used DP to solve simple stochastic optimal control problems, e.g. the dual adaptive control problem for a linear integrating system with an unknown gain [8].

While the developments in the AI and OR communities went largely unnoticed by the process control community, there were a few attempts for using similar techniques on process control problems. Hoskins and Himmelblau [48] first applied the RL concept to develop a learning control algorithm for a nonlinear CSTR but without any quantitative control objective function. They employed an adaptive heuristic critic algorithm suggested by Anderson [5] to train a neural network that maps the current state of the process to a suitable control action through *on-line* learning by experience. This approach used qualitative subgoals for the controller and could closely approximate the behavior of the PID controller, but generalization of the method requires sufficient on-line experiments to cover the domain of interest at the cost of more trials for learning. Miller and Williams [79] used a temporal-difference learning scheme for control of a bio-reactor. They used a backpropagation network to estimate Q-values, and the internal state of a plant model was assumed to be known. The learning was based on trial-and-errors, and the search space was small (only 2 states).

Wilson and Martinez [138] studied batch process automation using fuzzy modeling and RL. To reduce the high dimensionality of the state and action spaces, they used a fuzzy look-up table for Q-values. Anderson *et al.* [6] suggested a RL method for tuning a PI controller of simulated heating coil. Their action space had only 9 discrete values, and therefore the look-up table method could be used. Martinez [74] suggested batch process optimization using RL, which was formulated as a two dimensional search space problem by shrinking the region of policy parameters. This work did not solve the DP, but only used a RL-based approach for exploring in the action space. Ahamed *et al.* [1] solved a power system control problem, which they represented it as a Markov chain with known transition probabilities so that the system dynamics would have finite candidate state and action sets for exploration and optimization.

To summarize, there has been no attempt to solve DP directly for deriving an optimal control policy of a process characterized by continuous state and action spaces with practically large dimensions. Some of them are unrealistic in that validation and improvement of a solution rely on on-line experiments, and some of the problem formulations are

impractically simplified.

2.4.5 Solution Property: Convergence and Optimality

Understanding the accuracy of a learned cost-to-go function and its corresponding policy is very important for successful implementation. Researchers have been interested in understanding the convergence property of a learning algorithm and its error bound (or bias from the "true" optimal cost-to-go function). Though exact value and policy iterations are shown to converge and their error bounds are presented in standard DP textbooks [94], most of the approximate DP algorithms employing function approximation are yet to be understood fully at a theoretical level. This is particularly true for problems with continuous state-action spaces.

Gordon's value iteration algorithm using a local averager with the nonexpansion property [44] is convergent but its error bound is only available for the 1-nearest neighborhood estimator. Tsitsiklis and Van Roy [128] provided a proof of convergence and its accuracy for linear function approximators when applied to finite MDPs with temporal difference learning under a particular on-line state sampling scheme. They concluded that the convergence properties of general nonlinear function approximators (e.g. neural network) were still unclear.

Ormoneit and Sen [89] suggested a kernel-based Q-learning for continuous state space using sample trajectories only. The algorithm is designed for discounted infinite horizon cost and employs a kernel averager like Gordon's to average a collection of sampled data where a *specific* action was applied. Hence, a separate training data set exists marked with each action to approximate the Q-function. This way they show that the kernel-based Q-learning can converge to a unique solution, and the optimal solution can be obtained as the number of samples increases to infinity, which they call *consistency*. They also conclude that all reinforcement learning using finite samples is subject to bias. Same results for average cost problems are provided in [88]. Though the theoretical argument on the convergence property could be established, error bounds for practical set-up of the algorithm are yet to be provided.

Sutton *et al.* [113] suggested an alternative approach that directly optimizes over the policy space. The algorithm uses a parametric representation of a policy, and gradient-based optimization is performed to update the parameter set. As the number of parameters increases, the learning converges to an optimal policy in a "local" sense due to the gradient-based search. Konda and Tsitsiklis [54] proposed a similar approach under an actor-critic framework, which guarantees convergence to a locally optimal policy. In both approaches, they consider a finite MDP with a randomized stationary policy that gives action selection probabilities.

De Farias and Van Roy [36] proposed an approximate LP approach to solving DP based on parameterized approximation. They derived an error bound that characterizes the quality of approximations compared to the "best possible" approximation of the optimal cost-to-go function with given basis functions. The approach is, however, difficult to generalize to continuous state problems, because the LP approach requires a description of the system's stochastic behavior as finite number of constraints, which is impossible without discretization of a probabilistic model.

2.5 Conclusions

In this chapter, we have pointed out the MPC's inherent drawbacks, which are exorbitant on-line computation for complex models and awkwardness of handling uncertainties. We show that DP can address the issues effectively because it reduces a multi-stage optimization problem to a single-stage one and rigorously handles the uncertainty in a closed-loop manner. We reviewed the DP formulation and its classical solution algorithms. Both analytical and numerical solutions to DP, however, are seldom possible to obtain for practical problems due to the curse-of-dimensionality.

We have covered some popular approximate DP techniques in the fields of AI and ML, which are designed for circumventing the curse-of-dimensionality. They are not adequate to use for process control problems because of their different assumptions such as discrete states and actions, on-line trial-and-error learning, or availability of huge amounts of data. In the following chapter, we propose an ADP framework suitable for process control problems and

identify its potential issues.

CHAPTER III

ADP ARCHITECTURE FOR PROCESS CONTROL PROBLEMS

The objective of this chapter is to present an approximate dynamic programming (ADP) strategy suited to process control problems. We describe why the proposed method is most appealing for process control applications and then present its algorithms for deterministic and stochastic systems. The method is intended to take advantage of a DP formulation with manageable computation. We use a nonlinear Van de Vusse reactor to illustrate the efficacy of the proposed ADP architecture and identify its potential issues.

3.1 Proposed ADP Architecture

Process control problems are characterized by continuous state and action variables with a high dimensional state space. Unpredictable operation of processes to explore the state space and improve a control policy is generally undesirable. In addition, a family of incremental update rules introduced in Chapter 2 is designed for discrete state and action variables, hence it is not directly applicable to a problem with continuous state and action spaces.

The most viable approach to solve DP accounting for the above disparities is an *off-line* iteration scheme using a function approximator. For the circumvention of the curse-of-dimensionality, the proposed ADP strategy solves the Bellman equation off-line using *closed-loop simulations* and *function approximation*. It gathers operation or simulation data with some judiciously chosen closed-loop policies subject to various possible disturbances/operation conditions that guarantee stable operations at least. This way one avoids random exploration of the state space, which may be detrimental to a process. Within limited regions of the state space visited by the data, one solves the DP off-line using value or policy iteration. The premise of the suggested approach is that although the state

space may be huge, only a very small fraction of it would be relevant for optimal or near-optimal control in practice. The rationale for using closed-loop simulations is that several suboptimal policies combined together should span an envelope in the state space, within which satisfactory controls can be found. Of course, this is not always true and one should consider adaptive adjustment of the envelope through cautious exploration and additional simulations from the resulting DP-based policies.

Function approximation should be employed to estimate cost-to-go values for the data that were not visited by the simulations. Estimation of cost-to-go values for the unvisited data points in solving the Bellman equation is necessary due to the continuous nature of the state variables. Hence, the proposed scheme is characterized by simulation and iteration between function approximation and improvement of the cost-to-go approximation through value/policy iteration. LP-based approaches are excluded because representation of the constraints is infeasible for continuous problems. Next, the off-line iteration and on-line implementation procedures are described for both deterministic and stochastic systems.

3.1.1 Deterministic Systems

The following are the steps for constructing and improving the cost-to-go function off-line:

1. Perform simulations of the closed-loop system with some suboptimal control policies (μ^0) under all representative operating conditions. The quality of suboptimal control policies matter: The closer they are to the optimal policy, the better. However, optimal or near optimal control policies may be unknown or computationally too expensive to simulate. It is recommended that several policies effective in different regions of the state space and conditions be simulated.

2. Using the simulation data, calculate the infinite (or finite) horizon cost-to-go (J^{μ^0}) for each state visited during the simulations.

3. Construct a function approximator for the data to approximate the cost-to-go as a function of the state, denoted as $\tilde{J}^{\mu^0}(x)$.

4. To improve the cost-to-go, which is suboptimal because it is with respect to the simulated suboptimal policies, use a value or policy iteration. The algorithms described below are for infinite horizon problems. The formula for the finite horizon case is equally straightforward to derive based on (6).

   The value iteration solves

$$J^{i+1}(x) = \min_u \left\{ \phi(x,u) + \alpha \tilde{J}^i(f(x,u)) \right\} \tag{46}$$

   where i denotes the i^{th} iteration step. Once the cost-to-go values are updated for all the states, then we fit another cost-to-go function approximation to the x vs. $J^{i+1}(x)$ data.

   The policy iteration has the following two steps:

$$\tilde{J}^{\mu^i,\ell+1}(x) = \phi(x,\mu^i(x)) + \alpha \tilde{J}^{\mu^i,\ell}(x) \qquad \text{policy evaluation} \tag{47}$$

$$\mu^{i+1}(x) = \arg\min_u \left\{ \phi(x,u) + \alpha \tilde{J}^{\mu^i}(f(x,u)) \right\} \qquad \text{policy improvement} \tag{48}$$

   where ℓ and i are the iteration indices of policy evaluation and improvement steps, respectively. The policy iteration continues until $\mu(x)$ converges.

Since function approximation based on limited data in a continuous state space is used in our iterations, approximation errors at each step can be significant and the above results do not hold. In addition, even convergence cannot be guaranteed. Also, the question of which algorithm performs better for various practical scenarios is an open question.

Compared to the classical solution approach for DP, the above approach reduces the computational burden significantly for two reasons: First, even for very high-dimensional systems, the operating regions the closed-loop system occupies may represent a low-dimensional manifold. Second, for the infinite horizon cost-to-go calculation, the iteration of the Bellman equation is started with a very good estimate $\tilde{J}^{\mu^0}$, which is obtained through simulation with a suboptimal (but good) control policy.

On-line implementation of the optimal control law is based on the converged cost-to-go function and involves solving at each sample time

$$\min_u \left\{ \phi(x,u) + \alpha \tilde{J}(f(x,u)) \right\} \tag{49}$$

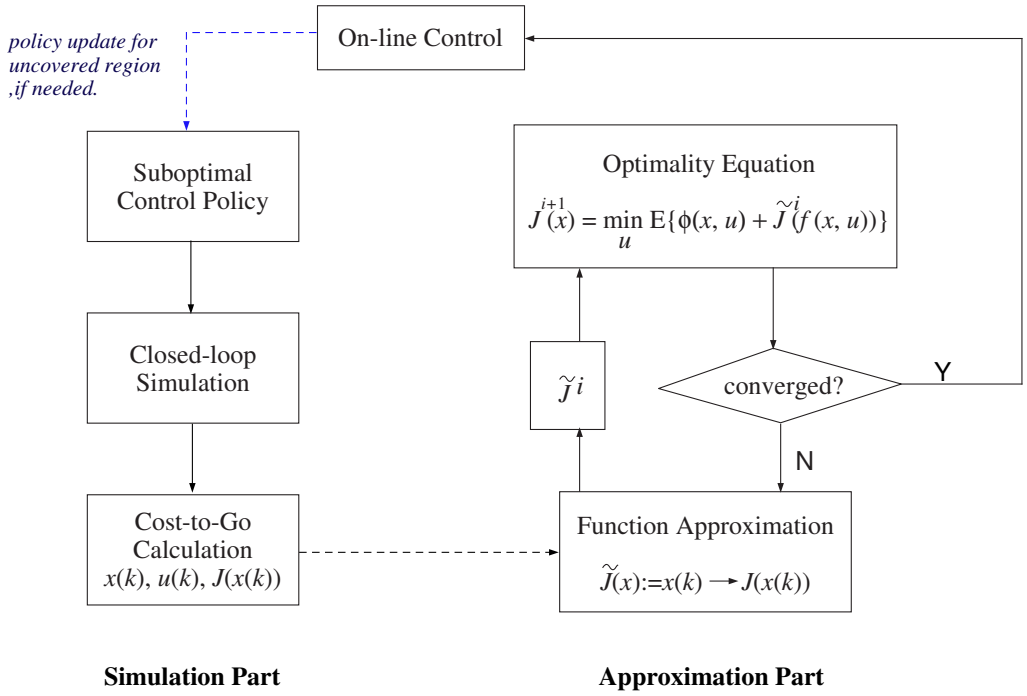

Figure 5: A schematic diagram of value iteration algorithm with simulation and function approximation.

which is relatively simple to solve computationally. Of course, one could iterate the closed-loop simulation with the new control policy defined above in order to add more state samples and obtain more accurate cost-to-go data for them. Figure 5 shows the value iteration scheme described above.

3.1.2 Stochastic Systems

In the stochastic case, the procedure is further complicated by the need to evaluate the expectation operator. However, the general idea remains the same as the following procedure shows:

1. Execute Monte Carlo simulations of the closed-loop system with some judiciously chosen suboptimal control law. For example, start with a control scheme that couples some state estimator like the extended Kalman Filter (EKF) and a deterministic control policy.

2. From the simulation data, calculate the discounted cost-to-go for all the visited states

and construct data for 'information vector' vs. cost-to-go.

3. Using a function approximation method, fit the data to obtain an initial approximation for the cost-to-go function, $\tilde{J}^{\mu^0}(\mathcal{I})$.

4. Perform value (or policy) iteration until convergence as follows.

   (a) With the current estimate $\tilde{J}^i$, calculate J^{i+1} for the given sample points of $\mathcal{I}$ by solving

   $$J^{i+1}(\mathcal{I}) = \min_u \mathbb{E}\left[\phi(\mathcal{I}, u) + \alpha \tilde{J}^i(f(\mathcal{I}, u))\right] \tag{50}$$

   This step is more demanding than before owing to the presence of the expectation operator (which arises from the fact that $f(\mathcal{I}, u)$ is a stochastic equation).

   (b) Fit an improved cost-to-go approximation to the $\mathcal{I}$ vs. $J^{i+1}(\mathcal{I})$ data.

5. One may also iterate the steps 1 – 4 with the updated suboptimal control policy for more improvement.

On-line implementation involves solving the following one-step optimization problem:

$$\min_u \mathbb{E}\left[\phi(\mathcal{I}, u) + \alpha \tilde{J}(f(\mathcal{I}, u))\right] \tag{51}$$

Note that, in calculating the expected cost-to-go from simulation data in the above iteration step, the expectation operator is not explicitly evaluated. Instead, it is thought that, by fitting an approximator to the data from various realizations of the stochastic system (i.e. Monte Carlo simulations), the fitted cost will represent the expected cost. This simulation-based approach also provides some additional flexibility such as the choice of disturbances. One does not have to limit oneself to some linear process driven by an i.i.d. Gaussian noise. One can work with non-normal distributions and (randomly occurring) deterministic disturbances mixed with stochastic noises.

Here we assumed a fixed estimator and it is not further optimized. This is not limiting as long as the model does not have any structural error. However, the information supplied to the estimator is influenced by the control decision and hence is optimized (for the particularly chosen estimator) directly for future control performance. Thus, one automatically gets some probing (i.e. active learning), designed for optimal control performance.

In the stochastic case, the exploration step (described in Step 5 of the aforementioned procedure) may be more important because the optimal control policy is known to behave very differently from the certainty equivalence control policy that ignores the uncertainty. Also, one may enhance the performance by performing closed-loop simulation subject to some dithering. In this way, the benefit of active exploration can be incorporated into the cost-to-go approximation.

This line of though also suggests a way to improve an already-implemented control policy in an evolutionary manner. One could collect the cost-to-go values from real operation and use it to improve the cost-to-go approximation and the corresponding control policy with the relatively simple framework described above.

3.2 Application to Van de Vusse Reactor

We consider a Van de Vusse reaction [130] in isothermal CSTR described by

$$\begin{aligned} \frac{dx_1}{dt} &= -k_1 x_1 - k_3 x_1^2 + (x_{1f} - x_1)u \\ \frac{dx_2}{dt} &= k_1 x_1 - k_2 x_2 - x_2 u \end{aligned} \tag{52}$$

where $k_1 = 50h^{-1}$, $k_2 = 100h^{-1}$, $k_3 = 10\frac{\ell}{mol \cdot h}$, and $x_{1f} = 10\frac{mol}{\ell}$. The state variables x_1 and x_2 represent the concentrations of the reactant and the intermediate, respectively. x_{1f} represents the concentration of the reactant in the feed, and u represents the dilution rate. The control objective is to regulate the output $y = x_2$, the concentration of the intermediate, at the set-point of 1.2 by manipulating the dilution rate with a sample time of $0.002h$. With the set-point, we have two steady state solutions: $\{x_1,\ x_2,\ u\} = \{5.5362,\ 1.2,\ 130.6758\}$ and $\{3.4960,\ 1.2,\ 45.6683\}$.

This reactor shows input multiplicities and process zero shifts from left-half-plane zero to right-half-plane zero at $u = 77.5$ when the output variable is x_2 [109]. A sufficiently long horizon is needed due to the nonminimum phase behavior of this system [77].

3.2.1 Deterministic Case

3.2.1.1 Suboptimal Nonlinear MPC

We consider the deterministic system and introduce step changes of various sizes in the parameters, k_1 and x_{1f} at $t = 0$. Hence, the cost-to-go function we construct is a function

of k_1 and x_{1f} as well as the two states.

As a starting suboptimal control policy, we use the nonlinear MPC algorithm based on successive linearization described in [61]. The method linearizes the nonlinear model at each current state and input values to calculate a prediction equation linear in terms of the future manipulated input moves. The control is computed by solving quadratic programming (QP), of which the Hessian matrix and the gradient vector are updated at each sample time. Here we assume that the full state variables are measured.

The one-stage cost $\phi(x, u)$ was chosen as:

$$\phi(x(k), u(k)) = Q(x_2(k+1) - 1.2)^2 + R\Delta u^2(k) \tag{53}$$

By formulating the single state cost to include a penalty on the error of the state at next sample time, we ensure that, in the on-line control calculation of (49), the one-step-ahead error is counted exactly (rather than through the approximated cost-to-go function), thus making the formulation more robust to approximation errors. The weighting factors we used are $Q = 1000$ and $R = 1$ in (53).

Assuming higher dilution rate is undesirable, we place an upper limit on the control input (70 h^{-1}) to prevent the input from drifting to the other side of the steady-state curve. The problem is the large inverse response, which can cause the MPC to drive the process away from the desired operating condition. To prevent this, we had to use a fairly large prediction horizon $p = 50$.

3.2.2 ADP Approach

We first employed the successively-linearized MPC (slMPC) scheme with $p = 50$ as a suboptimal control policy to generate closed-loop data for the initial cost-to-go calculation. Within $\pm 2\%$ of the nominal parameter values, we sampled 17 representative points from the disturbance space of k_1 and x_{1f} to cover the probable operating range. From the 17 simulation runs, 1360 data points for states vs. cost-to-go were obtained. The architecture for the cost-to-go function approximation we used in this work is a multilayer perceptron with 10 hidden nodes. The value iteration was used for off-line cost improvement and it

Table 1: Converged cost-to-go value in value iteration (deterministic case).

Infinite horizon cost	Average	Min.	Max.
Initial policy (slMPC)	1.01	0	125.27
Converged policy (ADP)	0.781	0	107.56

Table 2: On-line performance: closed-loop cost comparison of two control policies for 10 sample points (deterministic case).

Infinite horizon cost	Average	Min.	Max.	CPU time
Initial policy (slMPC)	58.95	5.92	113.25	110.17 s
ADP	24.39	2.57	49.69	52.62 s

converged after 2 runs with the following termination condition:

$$\frac{1}{N}\sum_{k=1}^{N}\left|\tilde{J}^{i+1}(x_k)-\tilde{J}^{i}(x_k)\right|<0.3 \tag{54}$$

where N is the number of data points, 1360. Table 1 compares the initial vs. converged cost-to-go for the 1360 data points.

In Table 2, we compare the on-line performance by calculating the infinite horizon costs from two different control policies, the NMPC control policy with $p = 50$ and the ADP-based control policy of (49). The actual infinite horizon costs were computed by performing closed-loop simulations with 10 fresh initial states that are different from those in the training set. The CPU time shown is the averaged value over the 10 simulations performed with MATLAB 6 on Pentium III (800 MHz). We can clearly see the superior performance of the control policy obtained from the ADP approach, both in terms of the optimality and computational time.

3.2.3 Stochastic Case with Full State Feedback

Consider the case where integrated white noises are introduced in k_1 and x_{1f} of (52).

$$\begin{aligned} k_1(k+1) &= x_3(k+1) = x_3(k)+e_1(k) \\ x_{1f}(k+1) &= x_4(k+1) = x_4(k)+e_2(k) \end{aligned} \tag{55}$$

where $e_1(k)\sim\mathcal{N}(0,0.2^2)$ and $e_2(k)\sim\mathcal{N}(0,0.04^2)$.

For reasonable sampling of the augmented state space, four representative realizations of the stochastic disturbances were selected: (1) monotonic increases in x_3 and x_4, (2)

Table 3: Comparison of closed-loop performance under two control policies with 10 fresh test data sets (stochastic case with full state feedback).

Averaged performance	slMPC	ADP
CPU time	326.4 s	120.4 s
Cost for 300 horizons	1082	888

monotonic increase in x_3 and monotonic decrease in x_4, (3) monotonic decrease in x_3 and monotonic increase in x_4, and (4) monotonic decreases in x_3 and x_4. It is important to note that we do not sample every possible state. The use of simulation to "judiciously" sample the space is the key idea. The total number of the state samples visited during the four stochastic simulations was 1200, and 800 data points for the cost-to-go approximation with the discount factor of 0.9.

In the simulation, the previously used state-feedback suboptimal nonlinear MPC with $p = 50$ gave an average discounted infinite horizon cost of 33.79 (averaged over all the states visited during the 4 simulations). We fitted to the 'state vs. cost-to-go' data from the four simulations using a multi-layer perceptron with 10 hidden nodes. This was followed by value iteration where the expectation was taken over 50 realizations. The termination condition of the iteration was

$$\frac{1}{N}\sum_{k=1}^{N} |\tilde{J}^{i+1}(x_k) - \tilde{J}^{i}(x_k)| < 1.0 \tag{56}$$

After 21 iterations, the average discounted cost associated with the converged approximator was 12.46, a significant reduction from the starting value, which means more optimal control policy was learned.

In order to compare control performances, we generated 10 fresh integrated noise disturbances (different from those in the training set). Table 3 shows the averaged result of the closed-loop simulation under the slMPC with $p = 50$ and the ADP-based approach with the test data set. The cost in the table is calculated for 300 time steps without a discount factor.

3.3 Conclusions

An ADP framework suitable for process control applications was proposed to combat two important deficiencies of MPC. By solving DP for the important regions of state space

with a function approximation scheme, the new framework was shown to offer enhanced computation time for on-line optimization and more improved control policy from a starting one. Simulation results from a nonlinear Van de Vusse reactor indicate that the suggested approach provides a promising framework to generalize MPC to handle nonlinear and/or hybrid system models as well as stochastic system models in a computationally amenable way. Since it is based on a function approximation scheme based on the data distributed over a limited envelope of state space, cautious utilization of simulation data and design of approximators are requisite for the success of the proposed scheme. We will discuss this issue in the next two chapters in detail.

CHAPTER IV

A COMPARATIVE STUDY ON THE CHOICE OF APPROXIMATOR

This chapter investigates the choice of function approximator within the strategy of approximate dynamic programming (ADP). A proper choice and design of the function approximator turns out to be critical for stability of the iteration and the quality of a learned control policy. This is because an approximation error can grow quickly through the iteration of optimization and function approximation. Typically, there are two classes of approximators for the approach: parameterized global approximators (e.g. artificial neural networks) and nonparametric local averagers (e.g. k-nearest neighbor). In this chapter, we assert based on some case studies and existing theories that the local-averager type approximators should be preferred over the global approximators as the former ensures stability of the off-line iteration, an important requirement for bringing the ADP strategy to practice. We also conclude that simple use of local averagers, while assuring convergence of the off-line iteration, does not necessarily lead to a stable learned control policy due to the problem of over-extrapolation. We hint that a local-averager again is better suited to handle this issue.

4.1 Introduction

In Chapter 3, we have shown the efficacy of the ADP approach using a nonlinear control problem. The main issue that arises in the practical implementation of the method, however, is that stability of learning and quality of a learned control policy are critically dependent on the structure and reliability of function approximator. The typical approach in the NDP and RL literature is to fit a global approximator like a neural network to cost-to-go data. While this approach has seen notable successes in some applications including a backgammon player at world champion level [122], it has also met with failures in many other applications [28, 105]. In certain instances, the off-line iteration would even fail to

converge, with the cost-to-go approximation showing extreme non-monotonic behavior or even instability with iteration.

The failure with a general function approximator was first explained by Thrun and Schwartz [126] with what they called an "overestimation" effect. They assumed uniformly distributed and independent error in the approximation and derived bounds on the necessary accuracy of the function approximator and discount factor. Sabes [100] showed that bias in optimality can be large when a basis function approximator is employed. Boyan and Moore [27] listed several simple simulation examples where popular approximators fail miserably in off-line learning. Sutton [118] modified the experimental setup for the same examples and adopted a model-free on-line learning scheme to make them work. In summary, experiments with different function approximation schemes have produced mixed results, probably because of the different learning schemes and problem setups.

Gordon [44] presented a 'stable' cost-to-go learning scheme with off-line iteration for a fixed set of states. A class of function approximators with a 'nonexpansion' property (e.g. k-nearest neighbor) was shown to guarantee off-line convergence of cost-to-gos to some values upon iteration. Gordon also provided a result for the accuracy of converged cost-to-go values in the case of 1-nearest neighbor, which has a fixed point property in approximation. Tsitsiklis and Van Roy [128] provided a proof of convergence and its accuracy for linear function approximators when applied to finite MDPs under temporal difference learning with a particular on-line state sampling scheme. They commented that the convergence properties with general nonlinear function approximators (e.g. neural network) remained unclear.

For systems with a continuous state space (infinite MDP), there are no proofs for convergence bound in the learning of cost-to-go function with *general* function approximators. For linear quadratic regularization problems, there exist proofs of convergence and its accuracy bounds for continuous state-action space problems with specific approximator structures [135]. Recently, a kernel-based local averaging structure was shown to have a property of convergence to the optimal cost-to-go with increasing number of samples and decreasing the kernel bandwidth for a certain model-free learning scheme [89, 88].

We argue for the use of local averagers for the function approximation in the approach. Our experience with the suggested approaches confirmed the great difficulty associated with selecting the structure and training algorithm of a global neural network in order to ensure stability of value iteration [64]. Hence, the use of local averagers appears more promising for process control problems. Even for continuous problems, we can show that the nonexpansion property guarantees stable learning in the off-line value iteration step. We also back up this assertion with several case studies. We believe that this is an important step towards making the approach more user-friendly and bringing it to practice. It still remains to make sure that the converged cost-to-go indeed leads to optimal or at least improved closed-loop performance. In this regard, we show that simple use of local averagers does not necessarily lead to a converged cost-to-go function guaranteeing closed-loop stability and performance, especially for a system with high state/input dimension. Analysis of the failed cases will show that the reason for poor closed-loop performance is attributed to 'over-extrapolation' of the cost-to-go approximation, both during the off-line iteration and on-line control, and a safeguard against over-extrapolation is needed for the approach to be successful.

The rest of the chapter is organized as follows: In Section 4.2, we discuss a class of local approximators with the nonexpansion property. In Section 4.3, our criteria for evaluating different types of approximators are first provided, and then several case studies are presented to compare the performances of global approximators vs. local approximators. Section 4.4 offers some conclusions.

4.2 Function Approximators with Nonexpansion Property

In this section, we discuss a family of local approximators with the nonexpansion property described in [44] and show how off-line convergence is guaranteed for discounted problems.

We first define a *contraction mapping* h with a chosen norm as

$$\forall x, y \in \mathcal{X}, \qquad \|h(x) - h(y)\| \leq \gamma \|x - y\| \tag{57}$$

where $\mathcal{X}$ is a closed vector space with a norm $\|\cdot\|$ on $\mathcal{X}$ and $\gamma \in [0, 1)$. We call h a *nonexpansion mapping* if $\gamma \in [0, 1]$.

In Chapter 2, the DP operator T was shown to be a contraction mapping with respect to the infinity norm and have a fixed point solution for discounted problems. Furthermore, with every possible state in the state space involved in a look-up table form, infinite number of applications of T will converge to the optimal cost-to-go values and a convergence bound can be provided for finite discounted MDP problems [94].

The suggested ADP scheme samples the state space using simulations and hence does not compute and store the cost-to-go value of every state in the state space. Let us denote the set of sampled states as X_{sam}. We also have the set of states, X_{est}, for which the cost-to-go should be estimated using the function approximator in solving (28). Finally, let $\tilde{T}$ represent the approximate Bellman operator with a chosen function approximation scheme.

Theorem 8. *Suppose $\tilde{T}$ is based on a family of function approximators with a nonexpansion property in the sense of the infinity norm. That is, the function approximator used in the approximate Bellman operator is a local averager with the following property:*

$$\tilde{J}(x_0) = \beta_0 k_0 + \sum_{i=1}^{n} \beta_i J(x_i) \tag{58}$$

with

$$\sum_{i=0}^{n} \beta_i = 1 \qquad \beta_i \geq 0 \quad (i = 0, \ \cdots, \ n) \tag{59}$$

where x_0 is a query point in the set X_{est} and n is the number of neighboring points in the set X_{sam} for approximation.

Then, the iteration of $\tilde{T}$ on the cost-to-go values for the sampled data points converges in the following sense:

$$\|J^{i+1}(x_j) - J^i(x_j)\|_\infty \leq \alpha^i \|\tilde{T}J^0(x_j) - J^0(x_j)\|_\infty \tag{60}$$

where $x_j (j = 1, \cdots, N) \in X_{sam}$ and i is the iteration index.

Proof. Suppose that there are $N(\geq n)$ sampled points in X_{sam} and $x_k \in X_{est}$. In each iteration, we compute $\tilde{T}J$ for $\forall x \in X_{sam}$, which requires evaluation of cost-to-go for several x_k's $\in X_{est}$ in the evaluation of the minimization operator. First, the approximation error

for all $x_k \in X_{est}$ over each iteration can be expressed as

$$\begin{aligned} \left|\tilde{J}^{i+1}(x_k) - \tilde{J}^{i}(x_k)\right| &= \left|\textstyle\sum_j \beta_{kj} \left(J^{i+1}(x_j) - J^{i}(x_j)\right)\right| \\ &\leq \max_j \left(J^{i+1}(x_j) - J^{i}(x_j)\right) = \|J^{i+1}(x_j) - J^{i}(x_j)\|_\infty \end{aligned} \tag{61}$$

where i is the iteration index and $x_j \in X_{sam}$. Here $\tilde{J}$ is used to emphasize the fact that its cost-to-go value is approximated based on its neighboring points in X_{sam} via the local averager.

We also note that the following property holds for arbitrary functions $g,\ h : \mathcal{A} \to \mathbb{R}$, where $\mathcal{A}$ is some arbitrary set:

$$\left|\min_a g(a) - \min_a h(a)\right| \leq \max_a |g(a) - h(a)| \tag{62}$$

Using the property the iteration error over the finite samples $x_j \in X_{sam}$ becomes

$$\begin{aligned} \left|J^{i+1}(x_j) - J^{i}(x_j)\right| &= \Big|\min_u \Big\{\textstyle\sum_{x_k \in X_{est}} p_{x_j,x_k}(u) \left(\phi(x_j,u) + \alpha \tilde{J}^{i}(x_k)\right)\Big\} \\ &\quad - \min_u \Big\{\textstyle\sum_{x_k \in X_{est}} p_{x_j,x_k}(u) \left(\phi(x_j,u) + \alpha \tilde{J}^{i-1}(x_k)\right)\Big\}\Big| \\ &\leq \max_u \alpha \textstyle\sum_{x_k \in X_{est}} p_{x_j,x_k}(u) \left|\tilde{J}^{i}(x_k) - \tilde{J}^{i-1}(x_k)\right| \\ &\leq \alpha \|\tilde{J}^{i}(x_k) - \tilde{J}^{i-1}(x_k)\|_\infty \end{aligned} \tag{63}$$

where p_{x_j,x_k} is a transition probability from x_j to x_k for a given choice of u. For a continuous state space, the transition probability is replaced by a transition probability density function and the sum by an integral over the entire state space, which can subsequently be replaced by a sum using a quadrature approximation.

From (61), Equation (63) becomes

$$\begin{aligned} \left\|J^{i+1}(x_j) - J^{i}(x_j)\right\|_\infty &\leq \alpha \left\|J^{i}(x_j) - J^{i-1}(x_j)\right\|_\infty \\ &= \alpha \|\tilde{T}^{i} J(x_j) - \tilde{T}^{i-1} J(x_j)\|_\infty \leq \alpha^i \|\tilde{T} J^0(x_j) - J^0(x_j)\|_\infty \end{aligned} \tag{64}$$

Since $\alpha < 1$, the right side converges to 0 as $i \to \infty$, and we can conclude that under the approximate value iteration scheme, the cost-to-go converges to some value for every $x_j \in X_{sam}$. □

The above theorem implies that the local averagers preserve the contraction mapping of the DP operator for the sampled data in X_{sam}, and therefore the vector $J(x)$ $(x \in X_{sam})$

converges upon the iteration for the discounted problems. We note, however, that accuracy bounds for the converged solution cannot be computed in general because such an averager may not necessarily have a same fixed point solution with the DP operator. Limited analysis on the converged solution's accuracy can be found for a finite MDP with a 1-nearest neighbor approximation in [44].

4.3 Case Studies

In this section, we present some case studies comparing different choices of function approximators. We provide the criteria for evaluating their performance in terms of the ease of off-line learning and on-line performance.

4.3.1 Choice of Function Approximators

4.3.1.1 Neural Network

Global and parametric approximators such as an artificial neural network (ANN) have been the popular choice in the NDP and RL literature [122, 142, 25, 107]. ANNs can learn arbitrarily complex functions and make predictions efficiently once trained. New training data are incorporated into the model and then discarded. This parametric nature loses information on the distribution of the data used for the training. For this study, we test a feedforward neural network of the following structure:

$$\tilde{J}(x) = \sum_{m=1}^{M} g_m(w_m^T x) \tag{65}$$

where g_m is a nonlinear function (e.g. sigmoid), and w_m is a vector of weighting factor, which is generally learned through a gradient-descent approach with error back-propagation.

4.3.1.2 Instance-Based Algorithms

Instance-based algorithms are nonparametric representations in that they simply store the training data. They use the stored points "close" to a query point to make a prediction when they are asked to do so. The closeness is typically defined according to some distance metric (e.g. Euclidean distance). K-nearest neighbor (kNN) and kernel-based predictors are instance-based algorithms. The computational load of these algorithms is mostly borne

during prediction not during learning, because the prediction involves calculation of distance and averaging whereas the learning amounts to simply storing the data. This is why they are often called *lazy learning* algorithms.

In this paper, we use kNN as the representative of the instance-based algorithms. We also note that the kNN has the nonexpansion property. Predictions of the kNN are given by

$$\tilde{J}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} J(x_i) \tag{66}$$

where $N_k(x)$ is the data set composed of the k-nearest neighbors of x defined by the training samples.

As a variant of the kNN, a distance weighted kNN can also be used to make the contribution of each data point to the prediction inversely proportional to the distance.

$$\tilde{J}(x) = \sum_{x_i \in N_k(x)} w_i J(x_i) \tag{67}$$

where

$$w_i = \frac{1/d_i}{\sum_i 1/d_i} \tag{68}$$

For the distance metric, we use the Euclidean distance defined as

$$d_i = \sqrt{(x - x_i)^T W (x - x_i)} \tag{69}$$

where W is a feature weighting matrix with zero off-diagonal elements assigning a weight to each dimension. This flexibility allows for emphasizing dimensions of the state variables that are more important than others.

4.3.2 Evaluation Criteria

4.3.2.1 Convergence Behavior in the Off-line Iteration Step

In the off-line iteration step, the approximate cost-to-go function is obtained through the iteration of the Bellman equation based on some function approximation scheme for the sampled points in the state space until convergence. We define the following properties as the criteria for evaluating the suitability of different types of function approximators.

- Learning stability: Does the cost-to-go converge within a given error tolerance (ε) upon the iteration?
- Monotonicity: Do the iteration errors (i.e. some norm of the differences between the cost-to-go values of the two consecutive iteration runs) decrease monotonically?
- Rate: How fast is the convergence?

We also adopt the following measure of the iteration error:

$$e_{abs}(i) = \max_{k=1,\cdots,N} \left| J^i(x_k) - J^{i-1}(x_k) \right| \tag{70}$$

where i denotes the iteration index and N is the number of sampled data points in the state space.

Alternatively, one can use a relative error in case that the magnitude of cost-to-go is very large.

$$e_{rel}(i) = \max_{k=1,\cdots,N} \left| \frac{J^i(x_k) - J^{i-1}(x_k)}{J^{i-1}(x_k)} \right| \tag{71}$$

4.3.2.2 Accuracy of the Converged Approximator

Of course, the resulting cost-to-go function should converge to a "reasonable" approximation of the optimal cost-to-go values. As a measure of proper learning, we use the following measure of the closed-loop performance obtained by implementing a control policy μ, which is a greedy policy based on the converged cost-to-go $\tilde{J}$:

$$J^{\mu}(x(0)) = \sum_{k=0}^{\infty} \phi(x(k), \mu(x(k))) \tag{72}$$

In the rest of the section, we present the results of applying the two function approximation schemes to three process control problems that feature different types of cost-to-go function structures: a smooth cost-to-go structure that is relatively easy to learn, a stiff cost-to-go structure that occurs often in the state constrained systems, and finally a system with a large number of states leading to sparse high dimensional data.

4.3.3 Case 1: Smooth Cost-to-Go Structure – Van de Vusse Reactor

We consider a simple nonlinear system, Van de Vusse reactor introduced in Chapter 3. The system is given by (52) with the same nominal parameter values. The control objective is to drive the system from any initial state to the equilibrium point $\{x_1,\ x_2,\ u\} = \{3.496,\ 1.2,\ 45.67\}$. The same one-stage cost ϕ of (53) is used.

4.3.3.1 Convergence Behavior in the Off-line Learning

For this example, 11 initial points were sampled by gridding the state space in an equally-spaced manner within the range of $0 \leq x_1 \leq 6.9920$ and $0 \leq x_2 \leq 2.4$. 121 sampled states were then obtained from closed-loop simulations of the system under a successive linearization based MPC (slMPC) controller [61], starting from the various initial states. The prediction and control horizons were 10 and 5, respectively. The cost-to-go values for the sampled states were initialized with the cost-to-go values calculated from the closed-loop simulation data.

Two approximators were tested: a feedforward neural network with three hidden nodes in the middle layer and a k-nearest neighbor averager with $k = 4$. The weight vector of the neural network was identified using the MATLAB Neural Network Toolbox [37]. Absolute convergence criterion of (70) was used with $\varepsilon = 0.1$. Figure 6 shows that the off-line learning behavior of kNN is stable and monotonic while that of NN is not. The cost-to-go structure is very smooth as depicted in Figure 7. The fluctuation in the iteration error for the NN approximation was shown to occur at the edge of sampled state values. The iteration behaviors are summarized in Table 4.

4.3.3.2 Accuracy of the Cost-to-Go

To demonstrate the effectiveness of the learned cost-to-go from each method, we compare the on-line performance based on (72). Of course, for the neural network, it is difficult to decide where to terminate the iteration. We present the result with the cost-to-go function after 100 iterations and note that results with other iteration runs produced similar solutions.

For comparison, we sampled 169 fresh data points different from the training data

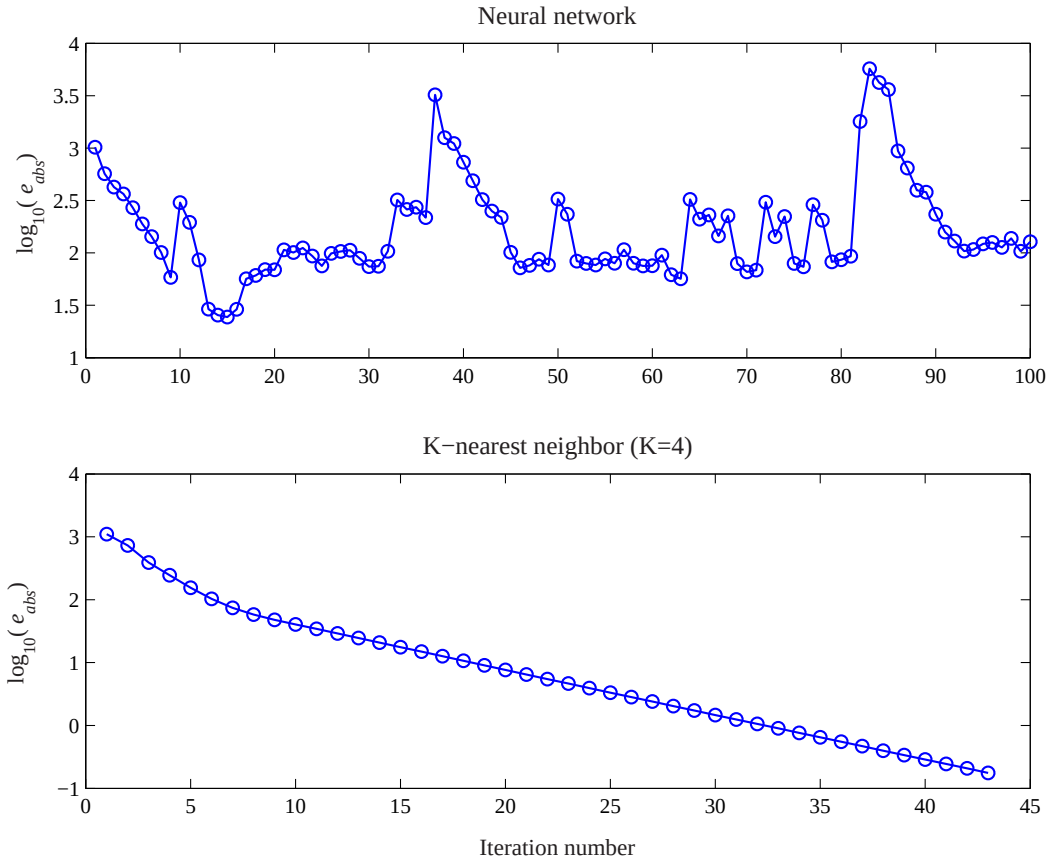

Figure 6: Van de Vusse reactor: comparison of off-line iteration trends.

Table 4: Van de Vusse reactor: convergence behavior of the off-line learning.

	Neural network	K-nearest neighbor
Stability	Not stable: e_{abs} shows sudden increases and fluctuates around 100 - 200, which is around 2-3% of maximum of cost-to-go.	Stable: the convergence tolerance was met after 43 iterations ($e_{abs} = 0.1$).
Monotonicity	No.	Yes.
Rate	Does not converge.	Converged and faster than neural network.

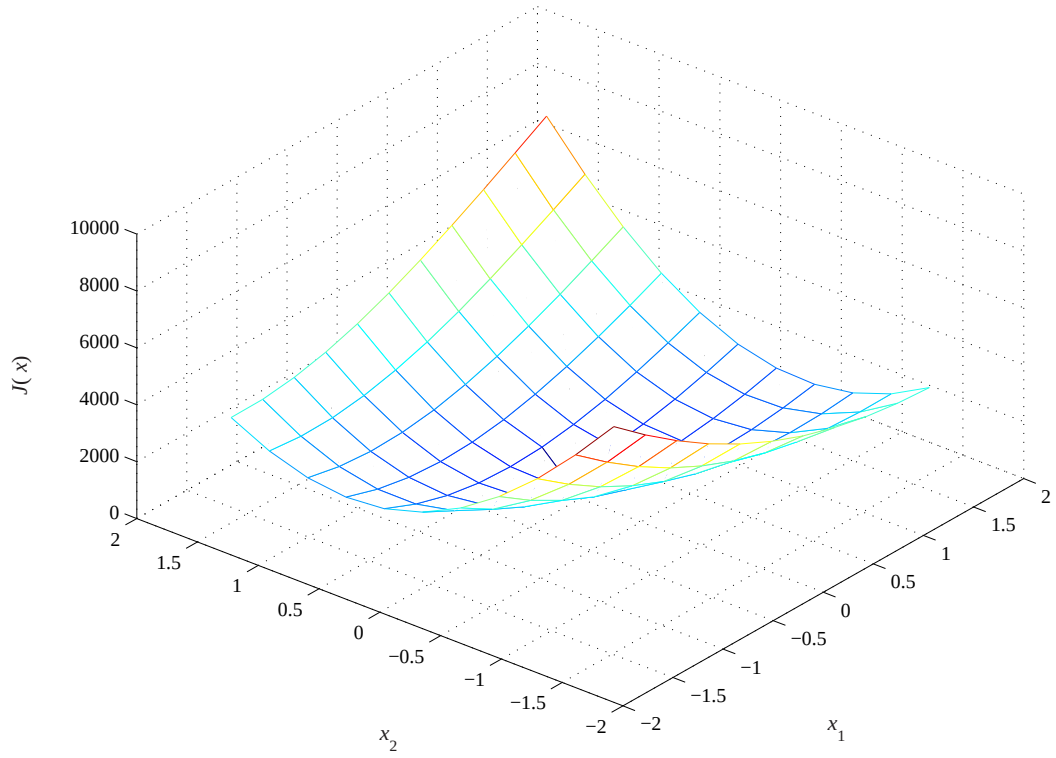

Figure 7: Van de Vusse reactor: cost-to-go function with 100 iterations using a neural network.

points. We also tested the trained data points. Out of 169 fresh data points, 129 data points showed better performances with the neural network and the rest with the k-nearest neighbor averager. In the case of the training data set, similar trends were observed. For 121 data points, 90 points showed better on-line performances with the neural network. The on-line performances are compared in Table 5. We can see that significant improved control performances resulted compared to the starting control policy in both cases. Despite the oscillatory behavior of the neural network in the off-line iteration, the cost-to-go approximation from the neural network at certain iterations was reasonable enough to yield a good on-line performance. However, this may be limited to the current case of smooth cost-to-go structure with a relatively "small" state space.

4.3.4 Case 2: Stiff Cost-to-Go Structure – State-Constrained System

Many process control problems have output or state constraints. Here we consider an example involving soft constraints, where a large penalty is assigned to the states violating

Table 5: Van de Vusse reactor: comparison of on-line performances.

$\frac{1}{N}\sum_{k=1}^{N} J^{\tilde{\mu}}(x_k)$	Neural network	K-nearest neighbor	slMPC
Trained points (N = 121)	1.2036e3	1.349e3	1.507e3
Fresh points (N = 169)	994.51	1.126e3	1.241e3

the constraints. This generally gives a very "stiff" cost-to-go shape, which is difficult for function approximators to learn. We consider the problem of disturbance rejection for a linear system with state and input constraints.[1] The model originally refers to the control of engine rpm (y) using a bypass valve (u) in the presence of step disturbance in torque load (d) [83]. With a sample time of 0.2 (sec), a discrete state space model is obtained as

$$\begin{aligned} x(k+1) &= \begin{bmatrix} -0.04223 & -0.3932 & -0.08433 \\ 0.04299 & 1.035 & -0.3984 \\ 0.08795 & 0.4352 & 0.4681 \end{bmatrix} x(k) + \begin{bmatrix} 0.06815 \\ 0.2912 \\ 0.6062 \end{bmatrix} u(k) \\ &\quad + \begin{bmatrix} -0.2026 \\ 1.68 \\ -0.468 \end{bmatrix} d(k) \\ y(k) &= \begin{bmatrix} 0.2999 & -2.021 & 0.9494 \end{bmatrix} x(k) \end{aligned} \tag{73}$$

The pertinent constraints imposed are $-5 \leq u \leq 5$ and $-5 \leq y \leq 5$. The disturbances are assumed to be measured and can take values between 0 and 1.

The constraints impose some interesting limits on the achievable performance. For $d > 0.84$, the system cannot be controlled to the set-point $y = 0$ with the given constraints. Also, for $d = 0.8$ and starting at the origin, no sequence of control actions $u(k)$ can be found to satisfy the state constraint, which necessitates constraint relaxation or softening. The cost-to-go function is defined as

$$J(x(k)) = \sum_{i=0}^{\infty} \alpha^i \phi(x(k+i), u(k+i)), \qquad \alpha = 0.99 \tag{74}$$

$$\phi(x(k), u(k)) = Qy^2(k+1) + R\Delta u^2(k) \tag{75}$$

[1] This simulation work was done by Niket S. Kaisare.

4.3.4.1 Convergence Behavior in the Off-line Learning

Two different PI controllers ($K_c = 0.25$, $\tau_I = 0.15$; $K_c = 0.5$, $\tau_I = 0.08$) were used as initial suboptimal policies, with the input moves truncated to satisfy the constraint $|u| \leq 5$. Simulations were performed for two different starting points and four different d values. 75 data points were obtained from each of the 16 scenarios (2 controllers, 2 initial conditions, 4 disturbances).

The weighting matrices for one-stage cost were chosen to be $Q = 1$ and $R = 0.04$. For the states with constraint violations, the stage-wise cost was multiplied 100 times (i.e., $Q = 100$, $R = 4$ if $|y(k)| > 5$). This results in a cost-to-go function having a very stiff structure. This is because the states that violate (or lead to immediate future violations of) state constraints have high cost-to-go values, while those that do not have significantly lower cost-to-go values.

The augmented state is composed of the system state, the disturbance, and the deviation from the set-point, i.e., $x = [x_1\ x_2\ x_3\ d\ (r-y)]^T$. Two different approximators were used to obtain cost-to-go as a function of system state:

- A feedforward neural network with 7 hidden nodes, and
- distance-weighted k-nearest neighbor ($k = 4$). As the disturbance (d) and error ($r-y$) were judged to be more critical than the three system states, feature weighting matrix $W = \text{diag}[1\ 1\ 1\ 6\ 10]$ was used in computing the distances for the normalized data.

The neural network was unable to provide a good approximation of the stiff cost-to-go function. The cost-to-go function did not converge even after 150 iterations. As shown in Figure 8, the learning with ANN is unstable. For the kNN, the relative iteration error decreased monotonically with increasing number of value iterations. The value iteration converged in 27 iterations with the tolerance level of $\varepsilon = 0.005$. The iteration behavior is summarized in Table 6.

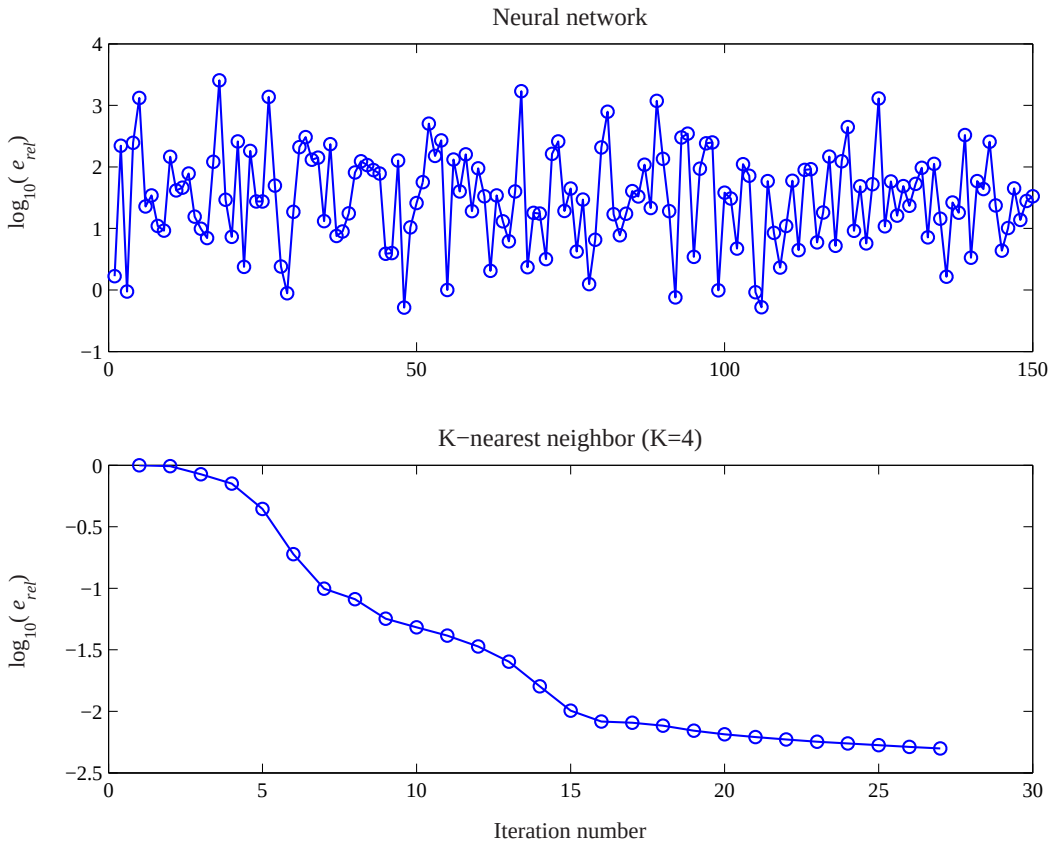

Figure 8: State-constrained case: comparison of off-line iteration trends.

Table 6: State-constrained case: convergence behavior of the off-line learning.

	Neural network	K-nearest neighbor
Stability	Not stable: e_{rel} shows fluctuations around the order of 10^2.	Stable: the convergence tolerance was met after 27 iterations ($e_{rel} = 0.005$).
Monotonicity	No.	Yes.
Rate	Does not converge.	Converged and faster than neural network.

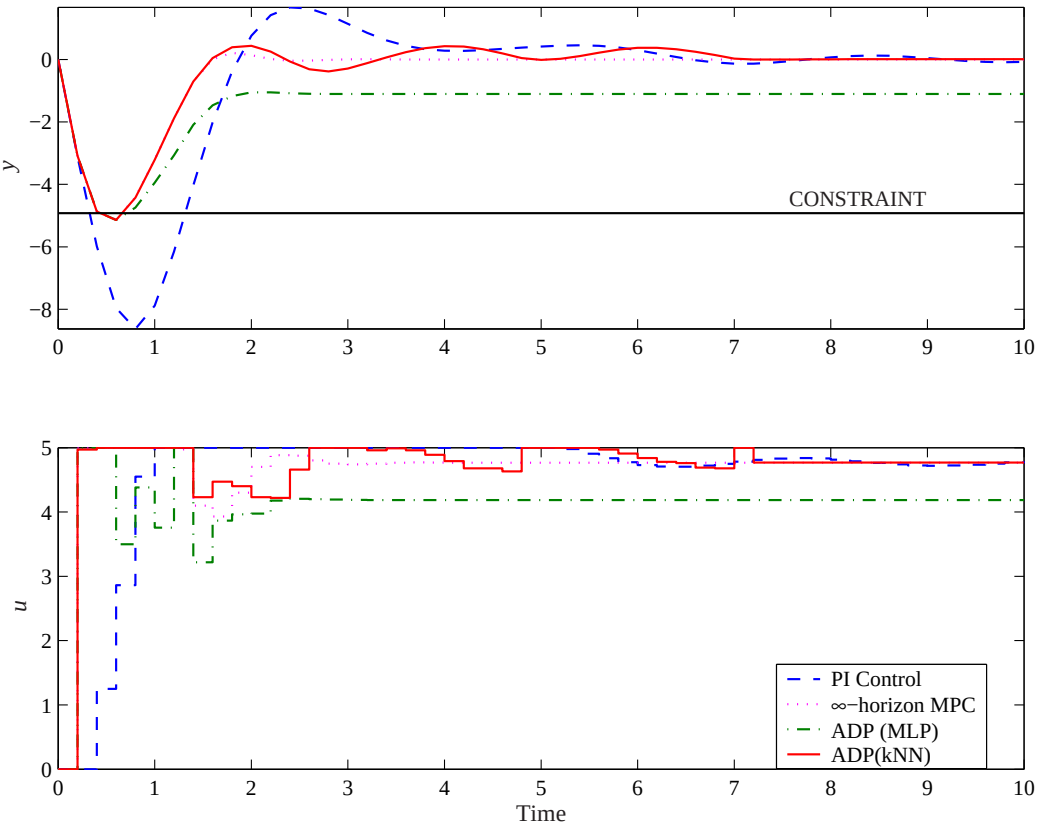

Figure 9: State-constrained case: On-line performance of the two approximators is compared with the optimal ∞-horizon control and the original suboptimal PI control.

4.3.4.2 Accuracy of the Cost-to-Go

None of the 150 trained neural networks were able to control the system. The dash-dot line in Figure 9 shows the performance of the best ANN cost approximator. On the other hand, the on-line performance using the k-nearest neighbor averager based on the converged cost-to-go data (solid line in Figure 9) is comparable to that of the truly optimal $\infty-$horizon MPC. The specific plots are for $d = 0.8$, wherein constraint softening is required as at least one point violates the constraint. Note that $d = 0.8$ is a new point, i.e., this condition was not used in learning the cost function.

4.3.5 Case 3: High Dimensional State Space with Sparse Data – MMA Polymerization Reactor

In this section, we consider the control of a free radical polymerization of methyl methacrylate (MMA) in the solution phase of a jacketed continuous stirred tank reactor (CSTR).

Table 7: Elementary reactions for free radical polymerization of MMA.

Initiation
$I \xrightarrow{k_d} 2R$
$M + R \xrightarrow{k_i} P_1$
Propagation
$P_n + M \xrightarrow{k_p} P_{n+1} \qquad (n \geqq 1)$
Chain transfer to monomer
$P_n + M \xrightarrow{k_{trm}} M_n + P_1 \qquad (n \geqq 1)$
Chain transfer to solvent
$P_n + S \xrightarrow{k_{trs}} M_n + P_1 \qquad (n \geqq 1)$
Termination
$P_n + P_m \xrightarrow{k_{td}} M_n + M_m \qquad$ (disproportionation)
$P_n + P_m \xrightarrow{k_{tc}} M_{n+m} \qquad$ (combination)

Note that P_n and M_n denote living and dead polymers, respectively.

The complete plant model has 8 state variables and is highly nonlinear [2]. This system presents challenges in that the state space is high dimensional but the amount of sampled data is limited, resulting in sparse occupation of the state space by the training data. This would be the typical scenario for practical process control problems.

4.3.5.1 Model Description and Problem Statement

The reaction kinetics of the free radical polymerization mechanism including chain transfer reactions to both solvent and monomer are shown in Table 7. The k^{th} moments of living and dead polymer concentrations, G_k and F_k, respectively are defined as

$$G_k = \sum_{n=1}^{\infty} n^k P_n(t) \tag{76}$$

$$F_k = \sum_{n=2}^{\infty} n^k M_n(t) \tag{77}$$

The details of the complete plant model can be found in [2] and are omitted due to space limitation. The state vector consists of variables: the concentrations of initiator, monomer, and solvent, the moments of dead polymer concentrations, and the temperatures of reactor and jacket, respectively.

$$x = \left[\frac{I}{I_f} \ \frac{M}{M_f} \ \frac{S}{S_f} \ \frac{F_0}{F_{00}} \ \frac{F_1}{F_{10}} \ \frac{F_2}{F_{20}} \ \frac{T_r}{T_f} \ \frac{T_j}{T_f} \right]^T \tag{78}$$

Jacket inlet temperature and flow rate are the manipulated inputs.

$$u = \begin{bmatrix} \frac{T_j^{in}}{T_f} & \frac{q_f}{q_{f0}} \end{bmatrix}^T \tag{79}$$

The controlled variables are the conversion and weight-average molecular weight defined as

$$X = \frac{V_p \rho_p}{V \rho_r} = \frac{F_1 W_m}{M W_m + S W_s + F_1 W_m} \tag{80}$$

$$M_w = W_m \frac{G_2 + F_2}{G_1 + F_1} \tag{81}$$

where V, ρ, W denote total volume of the reaction mixture, density, and molecular weight. The subscripts p, m, and r represent polymer, monomer, and reaction mixture, respectively. The controlled output vector y is represented by

$$y = \begin{bmatrix} X & \frac{M_w}{10000} \end{bmatrix}^T \tag{82}$$

Further simplification is possible by making the assumption of quasi-steady state in the concentrations of living polymers, constant V, and $q = q_f$ with dimensionless time τ.

$$\theta = \frac{V_0}{q_{f0}} = \frac{0.9l}{0.01l/60s} = 5400s \tag{83}$$

$$\tau = t/\theta \tag{84}$$

$$\frac{dx_1}{d\tau} = u_2 - u_2 x_1 - \theta k_d x_1 \tag{85}$$

$$\frac{dx_2}{d\tau} = u_2 - u_2 x_2 + \theta \left\{ -2 f k_d x_1 \frac{I_f}{M_f} - (k_p + k_{trm}) x_2 G_0 \right\} \tag{86}$$

$$\frac{dx_3}{d\tau} = u_2 - u_2 x_3 + \theta \left\{ -k_{trs} x_3 G_0 \right\} \tag{87}$$

$$\frac{dx_4}{d\tau} = -u_2 x_4 + \frac{\theta}{F_{00}} \left\{ \frac{1}{2}(k_t + k_{td}) G_0^2 + (k_{trm} M_f x_2 + k_{trs} S_f x_3) G_0 \right\} \tag{88}$$

$$\frac{dx_5}{d\tau} = -u_2 x_5 + \theta G_1 (k_t G_0 + k_{trm} M_f x_2 + k_{trs} S_f x_3) \tag{89}$$

$$\frac{dx_6}{d\tau} = -u_2 x_6 + \frac{\theta}{F_{20}} \left\{ (k_{td} G_0 + k_{trm} M_f x_2 + k_{trs} S_f x_3) G_2 + k_{tc} (G_0 G_2 + G_1^2) \right\} \tag{90}$$

$$\frac{dx_7}{d\tau} = u_2 (1 - x_7) + \frac{\theta}{T_f \overline{\rho C_p}} (-\Delta H) k_p M_f x_2 G_0 + \frac{UA}{\overline{\rho C_p} q_{f0}} (x_8 - x_7) \tag{91}$$

$$\frac{dx_8}{d\tau} = \theta \left\{ q_c \frac{(u_1 - x_7)}{V_j} + \frac{UA}{\rho_w C_{pw} V_j} (x_7 - x_8) + \frac{U_a A_a}{\rho_w C_{pw} V_j} \left(\frac{T_a}{T_f} - x_8 \right) \right\} \tag{92}$$

where

$$G_0 = \sqrt{\frac{2 f k_d I}{k_t}} \tag{93}$$

Table 8: Model parameters of MMA polymerization reactor.

k_{d0}	1.25×10^{18}	E_d	35473
k_{p0}	2.94×10^{6}	E_p	5656
k_{t0}	5.2×10^{8}	E_t	1394
k_{td0}	1.83×10^{27}	E_{td}	44467
k_{trm0}	$9.32k_{p0} \times 10^{3}$	E_{trm}	$E_p + 13971$
k_{trs0}	$8.75k_{p0} \times 10^{-5}$	E_{trs}	$E_p + 42.6$
k_{tc}	$k_t - k_{td}$	R	1.987 $[\frac{cal}{mol \cdot K}]$
W_m	100.12 $[\frac{g}{gmol}]$	W_s	88.12 $[\frac{g}{gmol}]$
W_i	242.23 $[\frac{g}{gmol}]$	q_{f0}	1.67×10^{-4} $[\frac{l}{s}]$
T_f	20 $[^\circ C]$	T_a	20 $[^\circ C]$
$-\Delta H$	13800	q_c	0.0833
I_f	0.0206	M_f	4.7104
S_f	5.1085		
F_{00}	0.001	F_{20}	1000
f	0.4	UA	3.2
$U_a A_a$	3.2	V_j	0.8
V	0.9 $[l]$	θ	5400 $[s]$

$$G_1 = \frac{2fk_dI + (k_pM + k_{trm}M + k_{trs}S)\,G_0}{k_tG_0 + k_{trm}M + k_{trs}S} \tag{94}$$

$$G_2 = G_1 + \frac{2k_pMG_1}{k_tG_0 + k_{trm}M + k_{trs}S} \tag{95}$$

Since the concentration of living polymers is much smaller than that of dead polymers, the contribution of living polymer moments to the overall molecular weight is negligibly small. Thus, Equation (81) is further reduced to

$$M_w = W_m \frac{F_2}{F_1} \tag{96}$$

The control objective is to drive the outputs to the set-points of $[0.2,\ 15]^T$ from an initial state. The model parameters and the feed condition are found in Table 8. Step changes of various sizes in the activation energies of dissociation and propagation are introduced at the initial time as depicted in Figure 10. In the sequel, the simplified model will be used as a plant and no model/plant mismatch is assumed to exist.

4.3.5.2 State Space Sampling using Suboptimal Control Policy: slMPC

To cover the pertinent operating ranges, we sampled 19 points in the disturbance space as shown in Figure 10 and performed closed-loop simulations using slMPC. 3549 data points

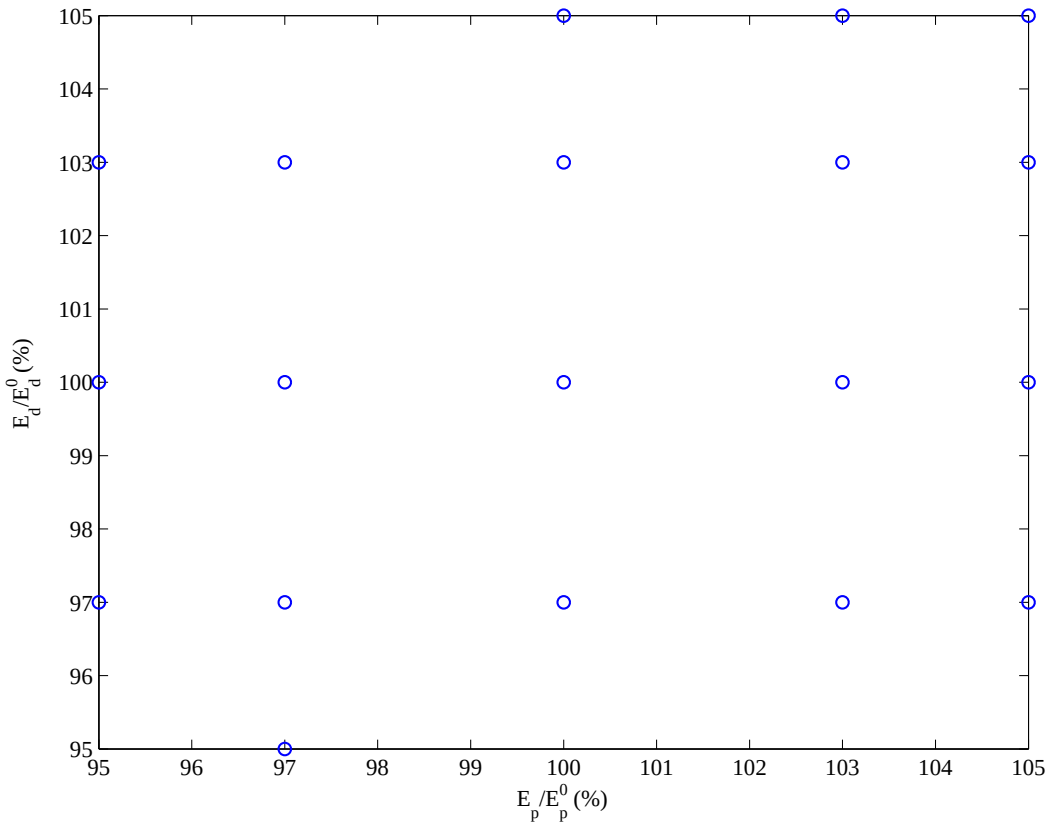

Figure 10: Possible step disturbances in the parameter space of MMA polymerization reactor.

Table 9: MMA reactor: input constraints and parameters for slMPC.

Sample time	90 sec
Prediction horizon	15
Control horizon	5
Output weighting	$\begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$
Input weighting	$\begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}$
Input magnitude constraints	$50°C \leq T_j^{in} \leq 90°C$, $0.1 \leq q_f \leq 2q_{f0}$
Input rate constraints	$\lvert\Delta T_j^{in}\rvert \leq 5°C$, $\lvert\Delta q_f\rvert \leq 5ml/min$

were obtained from the simulations. The tuning parameters of slMPC and the input constraints are given in Table 9. We note that it was very difficult to find other control policies and initial conditions that operate the system stably. The cost-to-go is defined as

$$J(x(k)) = \sum_{t=0}^{\infty} \alpha^t \phi(x(k+t), u(k+t)), \qquad \alpha = 0.98 \tag{97}$$

$$\phi(x(k), u(k)) = (r - y(k+1))^T \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} (r - y(k+1))^T + \Delta u^T(k) \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix} \Delta u(k) \tag{98}$$

The state vector is augmented as follows for encoding the cost-to-go information:

$$x^{aug} = [x_1 \; x_2 \; \cdots \; x_8 \; E_d \; E_p \; y_1 \; y_2]^T \tag{99}$$

Note that the outputs are also included in the augmented state in order to ensure integral actions. This helps because the output variables are nonlinear functions of the state variables. Figure 11 shows the visited states under the closed-loop simulations using the slMPC policy.

4.3.5.3 Convergence Behavior in the Off-line Learning

For the 3549 data points, two different function approximators relating the cost-to-go with augmented state were tested: a feedforward neural network and a distance-weighted k-nearest neighbor (K=4) averager. Based on the approximators, value iteration was performed. In using the neural network, the number of hidden nodes was adjusted at each iteration to respect the fitting criterion of MSE < 0.001. $\varepsilon = 0.03$ was used for the relative error convergence criterion of (71). Figure 12 shows that the kNN approximator learns

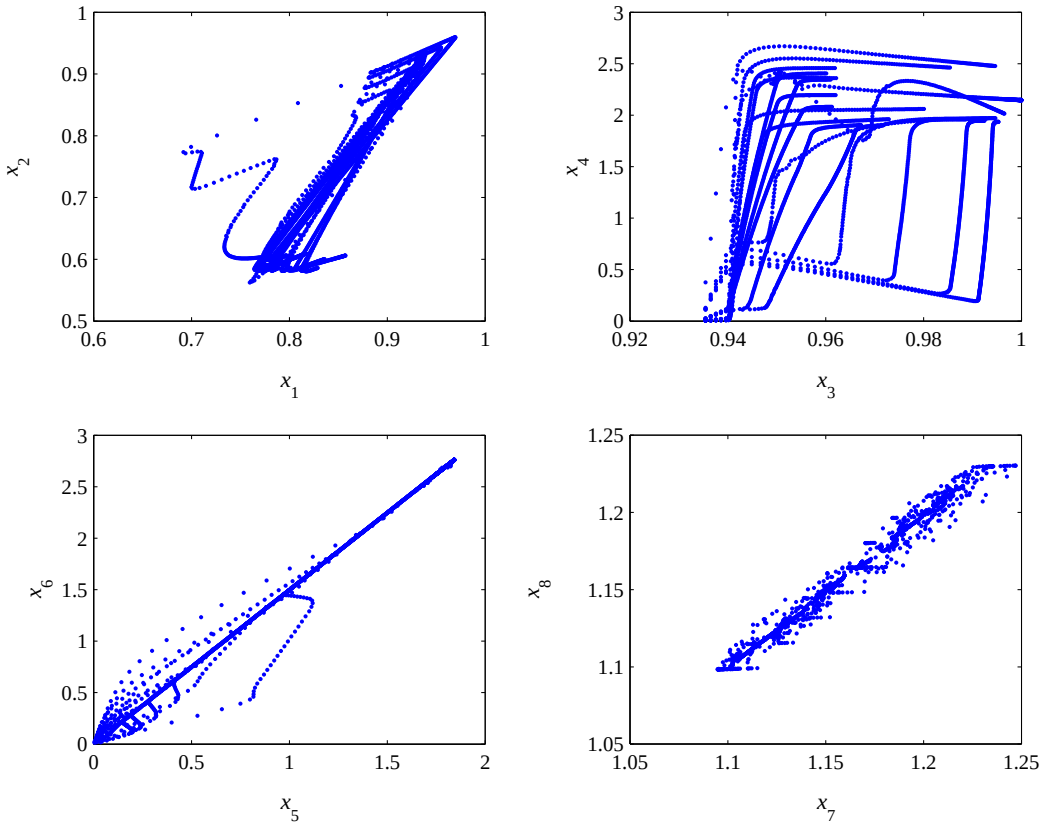

Figure 11: MMA reactor: state space plot of states visited during suboptimal (slMPC) simulations.

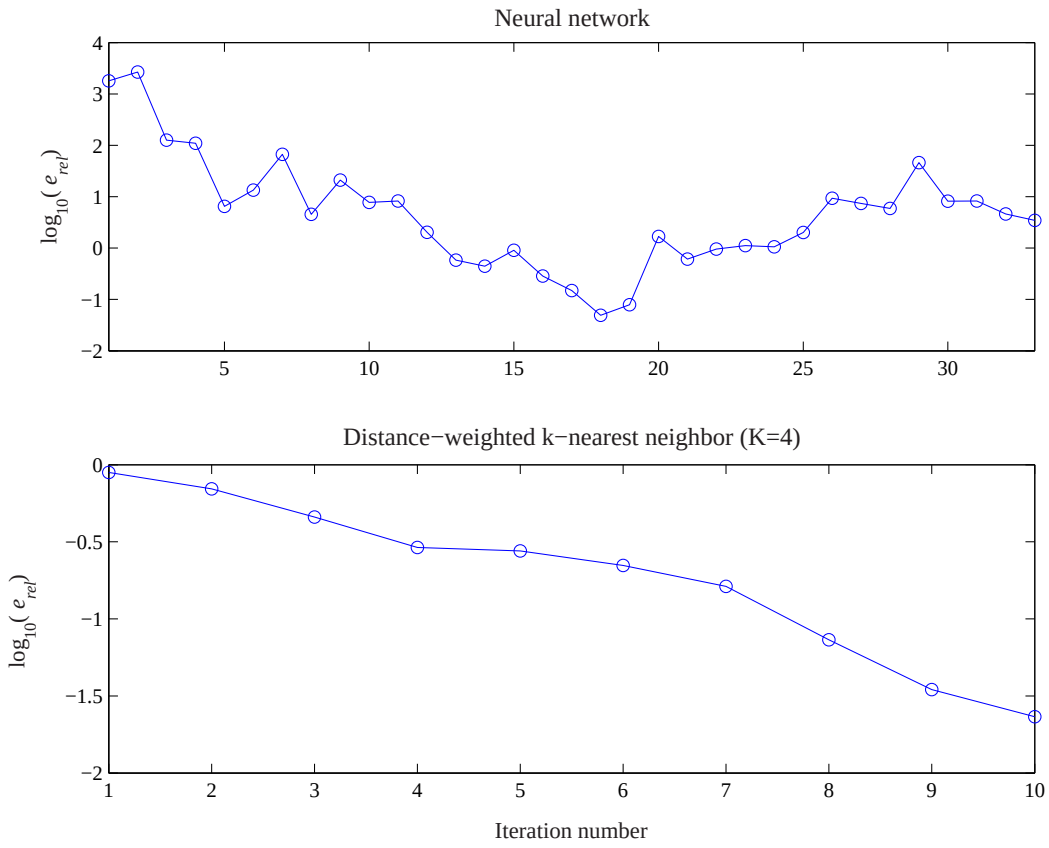

Figure 12: MMA reactor: comparison of off-line iteration trends.

stably with convergence achieved after 10 iterations, while the NN does not. Table 10 summarizes the observation of off-line learning behavior.

4.3.5.4 Accuracy of the Cost-to-Go

We tested the kNN averager based on the converged cost-to-go data and the neural network obtained at the 18^{th} iteration which showed the minimum iteration error. A representative result for nominal values of E_d and E_p is shown as solid lines in Figure 13. On-line performance was tested for the various values in the parameter space. Though the cost-to-go converged stably in the kNN case, both approximators do not control the system successfully. An investigation of the state space plot in Figure 14 suggests that extrapolations to previously unvisited regions of the state space are responsible for the undesirable closed-loop behavior. Hence, this case study indicates that the case of sparsely sampled high dimensional state space should be approached with a more robust algorithm for off-line learning

Table 10: MMA reactor: convergence behavior of the off-line learning.

	Neural network	K-nearest neighbor
Stability	Not stable: e_{rel} shows fluctuations and e_{abs} increases exponentially.	Stable: the convergence tolerance was met after 10 iterations ($e_{rel} = 0.03$).
Monotonicity	No.	Yes.
Rate	Does not converge.	Converged and faster than neural network.

as well as on-line optimization. This is a topic of the Chapter 5. We also note that other cost-to-go functions from different iteration runs for the neural network case show the same trends.

4.4 Conclusions

In this chapter, we investigated the choice of approximators for the approximate dynamic programming strategy for process control. Specifically, we compared a parameterized global approximator and a nonparameterized local averager with a 'nonexpansion' property in terms of their off-line convergence behavior and eventual on-line performance. Some theoretical analysis as well as the simulation results of three process control problems let us conclude that the use of local averagers gives more predictable off-line convergence behavior and often better on-line performance. Though global and parametric representation such as artificial neural networks has been used successfully as a cost-to-go function approximator in many applications, training data should be large and densely occupy the state space, which is rare for process control problems. This result has an important practical implication for the future of the ADP strategy in process industries as unpredictable off-line training results and on-line performance will undoubtedly frustrate the users and have them give up entirely.

Despite the nice behavior of the local averagers in the off-line learning phase, it is shown that its use does not automatically guarantee an acceptable on-line performance, especially in the case that the learning data is sparse in the state space. This is because uncontrolled extrapolations can deteriorate the quality of cost-to-go approximations since the learned cost-to-go information is valid only in the regions of the state space where learning data

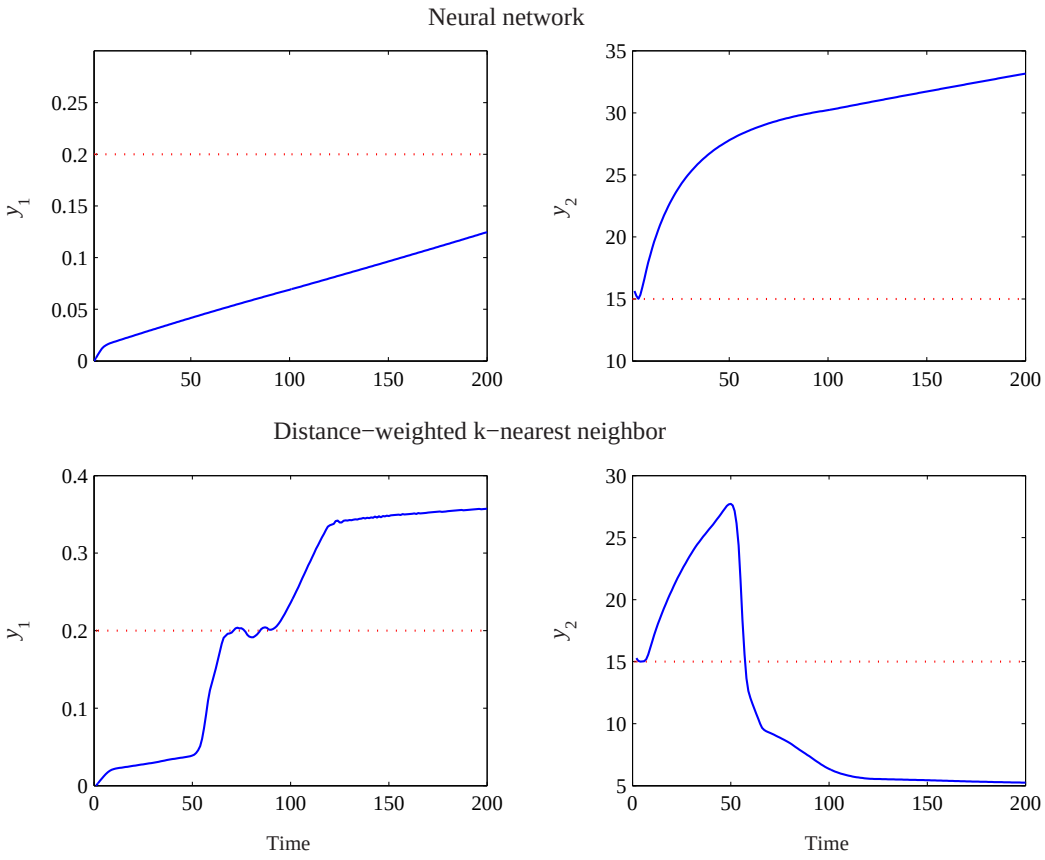

Figure 13: MMA reactor: comparison of on-line performances: $E_d/E_{d0} = 1$, $E_p/E_{p0} = 1$. The dotted lines are set-points.

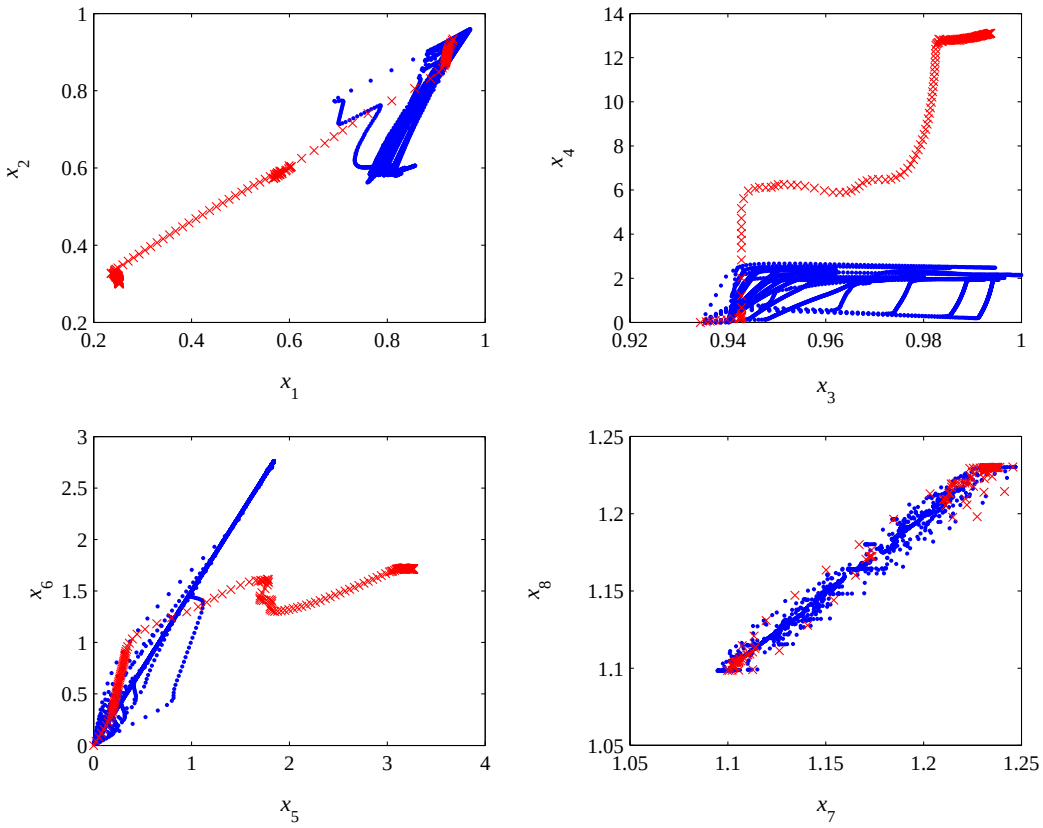

Figure 14: MMA reactor: state space plot of states visited during on-line implementation with distance-weighted kNN (X). The dots are data used for cost-to-go approximation.

were available – a fact that is obvious but not always given attention to in applications. This observation suggests that an additional strategy for balancing between exploration and exploitation (cautious use of the learned cost-to-go information) in a user-controlled manner should be devised for the ADP strategy to be applicable to practical process control problems.

CHAPTER V

DESIGN OF PENALTY FUNCTION FOR ROBUSTNESS

This chapter addresses the problem of 'excessive extrapolation' during learning and use of a cost-to-go function in implementing the approximate dynamic programming strategy. We propose a penalty function term to be used in conjunction with a local averager for more robust estimation and use of cost-to-go information. Though the use of certain local averagers guarantees convergence in the off-line value iteration step as shown in Chapter 4, cost-to-go predictions provided by the local averagers may not be accurate, if the data density around a query point is inadequate. This is especially a problem in cases of high dimensional state space with sparse training data. To cope with this difficulty, we propose that a penalty term be included in the objective function in each minimization to discourage the optimizer from finding a solution in the regions of state space for which confidence in the approximation is low. Confidence in this context can be measured by an estimate of local data density around a query point. The Parzen density estimator, which can be naturally combined with a local averager, is suggested for this purpose and a quadratic penalty term is designed based on the local data density estimate. The suggested use of the penalty function endows the user with the ability to use given cost-to-go information in a more controlled manner and avoid excessive extrapolations, which can lead to a significant bias in the learned cost-to-go and poor closed-loop performance. We illustrate the potential robustness advantage that the suggested modification allows for with the MMA polymerization reactor example used in the previous chapter.

5.1 Introduction

In Chapter 4, we compared a neural network and a local averager for function approximation and concluded that the latter gives better off-line convergence behavior and oftentimes results in better closed-loop performances within the overall ADP strategy.

Since the approximation of a cost-to-go function is based on the data generated by closed-loop simulations or experiments, the domains of the state space wherein the approximation is valid may be limited. Sparseness of training data is expected to be a rule rather than an exception in process control problems where high dimensional state space, complex nonlinear dynamics, and limited opportunities for experimentation are typical. Though the learning based on data sampled from closed-loop simulations/experiments helps us deal with the curse of dimensionality, caution must be exercised in generalizing the learned cost-to-go information since it may not be globally valid. This problem is fundamental and the use of a local averager does not prevent it. However, it does facilitate the design of a strategy to deal with it, as we will see in this chapter.

Though the problem of cost-to-go function approximation has been studied extensively in dynamic programming literature, the issue of *extrapolation* has not been dealt with explicitly. For example, in Neuro-Dynamic Programming, it is normally assumed that large amounts of training data are available [25]. In addition, random exploration of state space is not considered detrimental due to the nature of studied applications. In fact, learning is designed to be evolutionary (e.g., game playing and robot-learning) in their common algorithms in that they are intended to explore through randomized control policies and learn from "bad" on-line trajectories as well as good trajectories.

One noteworthy work dealing with the issue of excessive extrapolation in the context of cost-to-go function approximation is found in [111]. They construct an approximate convex hull, which is called *independent variable hull (IVH)* taking an elliptic form as depicted in Figure 15. It finds a fixed number of training data that lie closer than some threshold value from a query point and builds the IVH. Whenever they have to estimate the cost-to-go for a query point, the IVH is calculated and the query point is checked whether it lies inside the hull or not. Any queries within the convex hull are considered to be reliable and those outside are deemed unreliable. A query point, q, is categorized as a reliable point, if

$$q^T(X^TX)^{-1}q \leq \max_i v_{ii} \tag{100}$$

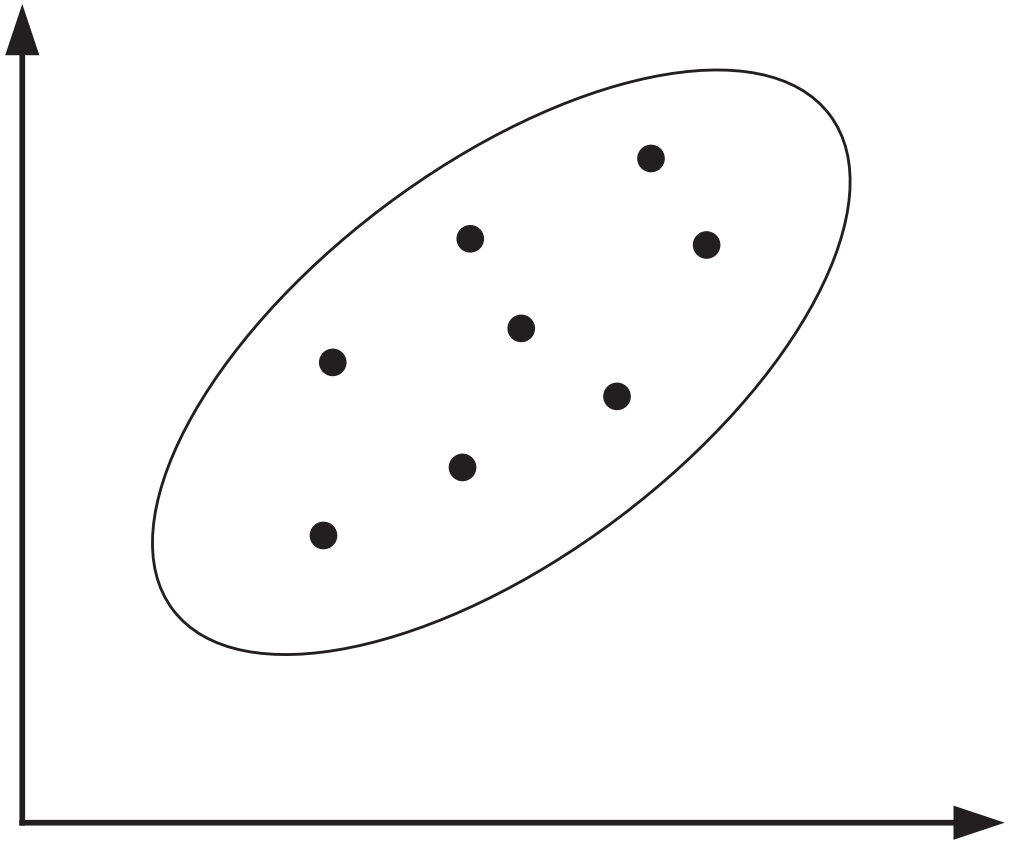

Figure 15: Two-dimensional independent variable hull (IVH).

where v_{ii} are the diagonal elements of matrix V, which is calculated according to

$$V = X(X^T X)^{-1} X^T \tag{101}$$

where X is a matrix with its rows corresponding to the training data. This approach is not computationally attractive. It also gives up making a prediction for the point outside the hull and requires more random exploration. In addition, the design of a convex hull may be misleading if the elliptic hull contains a significant empty region around the query point.

In chemical process control, it is of paramount importance for control to ensure stable and safe operations, even if it means the economic optimality is sacrificed a bit. This calls for a modification in the typical ADP approach, which requires explorations through randomized policies and encourages 'learning by mistakes.' In process control, large control actions with unpredictable outcomes should be avoided at all cost. This implies that an

ADP algorithm for process control should avoid excessive stretch of given or learned cost-to-go information beyond the domain of training data and try to produce a solution within the boundaries of given information, even though such a solution will generally be suboptimal.

One plausible approach to deal with this problem is to calculate and use information on the local data density in solving the minimization appearing in the Bellman equation. To this end, we propose to use a probability density estimator proposed by Parzen [90] for estimation of local data density. The density estimate is then translated into a quadratic penalty term added to the cost-to-go estimate to discourage the optimizer from finding a solution (i.e. a next state) in regions where the training data density is inadequately low. The penalty term systematically biases upward the cost-to-go function in a manner inversely proportional to the local data density. This way we can make the off-line iteration steps and on-line control calculations more robust against potential approximation errors from insufficient sampling.

For demonstrating the effectiveness of the proposed method, the continuous methyl methacrylate (MMA) polymerization reactor example studied Chapter 4 is revisited where simple application of a local averager was shown to be inadequate. The rest of the chapter is organized as follows: Section 5.2 presents the design of a penalty function based on a multi-dimensional Parzen density estimator. In Section 5.3, the modified ADP strategy that makes use of the penalty function is discussed. In Section 5.4, the performance of the controller resulting from the modified ADP approach is compared with that of a starting control policy (successive linearization based MPC by Lee and Ricker [61]) as well as a nonlinear MPC controller. Section 5.5 offers some conclusions.

5.2 Penalty Function Based on Local Data Density

5.2.1 Local Data Density Estimator

There are two principal approaches to (probability) density estimation, the parametric and the nonparametric design. In the parametric estimation, the distribution of data is assumed to follow a certain form with a few adjustable parameters. An example is the Gaussian distribution, which is parameterized by a mean vector and a covariance matrix. Such an

approach is statistically and computationally efficient but can lead to poor results if the presumed form is not close to an underlying density distribution. Alternatively, a nonparametric approach can be taken where the shape of the density estimate is also determined by the data. In principle, given enough data, arbitrary densities can be estimated to a desired accuracy, which is called *consistency* [106]. One of the most popular nonparametric methods is the Parzen estimator, which is based on local smoothing of the data with a kernel function. A disadvantage of the approach is the intensive computational requirement. However, we will discuss later that this is not the case for our application since the purpose of our adoption is not to estimate the global density. Instead, we employ the multi-dimensional Parzen probability density function [29] to arrive at a measure for confidence in a cost-to-go approximation.

Suppose that we have a training data set Ω and a new query point x_0. The Parzen density estimate, $f_\Omega(x_0)$, is obtained as a sum of kernel functions placed at each sample in Ω according to

$$f_\Omega(x_0) = \frac{1}{N\sigma^{m_0}} \sum_{i=1}^{N} K\left(\frac{x_0 - x_i}{\sigma}\right) \tag{102}$$

where $x_0,\ x_i \in \mathbb{R}^{m_0}$, $x_i \in \Omega$, K is a selected kernel function, N is the number of data in Ω, and σ is the smoothing parameter (or the bandwidth). The kernel function, K, is often chosen in a way such that it has mathematically tractable properties such as continuity or differentiability. A well-known and widely used kernel is the following multivariate Gaussian distribution function:

$$K\left(\frac{x_0 - x_i}{\sigma}\right) = \frac{1}{(2\pi\sigma^2)^{(m_0/2)}} \exp\left(-\frac{\|x_0 - x_i\|_2^2}{2\sigma^2}\right) \tag{103}$$

Hence, the multi-dimensional Parzen density function is based on Euclidean distance between the query point x_0 and neighboring points x_i in the training data set, Ω, through the kernel function. The kernel assigns high values to the training points close to x_0 and low values to those far.

5.2.2 Quadratic Penalty Function

Since a local averager estimates a cost-to-go value of a query point based on those of the nearest neighboring points, which can be very far, states in regions with little data can still

be chosen as a solution to the minimization. However, this is undesirable as the cost-to-go estimate for such a state is likely to have a significant error. If we know the structure of the true cost-to-go function, accurate extrapolations to such a region could be done, but this is not a general case.

To assure a predictable outcome, we should select a control action such that the solution state has adequate data density around it. To this end, a penalty function based on the local data density is designed and added to the cost-to-go estimate in solving the minimization. Penalty function method has been an important element of constrained optimization for decades [87]. In optimization, a penalty function is mainly used to force the search to stay inside or close to a feasible region. A similar approach can be taken in our problem's context: A penalty term can be included in the objective function to bias the search to those regions of adequate data densities. This way, excessive extrapolation can be avoided.

In this work, we use a quadratic penalty function that adjusts the objective function as follows:

$$\tilde{J}_{\text{aug}}(x_0) \Leftarrow \tilde{J}(x_0) + J_{\text{bias}}(x_0) \tag{104}$$

$$J_{\text{bias}}(x_0) = A \cdot H\left(\frac{1}{f_\Omega(x_0)} - \rho\right) \cdot \left[\frac{\frac{1}{f_\Omega(x_0)} - \rho}{\rho}\right]^2 \tag{105}$$

where x_0 is a query point, ρ is a user-given threshold value, A is a scaling parameter, and H is a heavy-side step function defined as

$$H(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{106}$$

Figure 16 depicts how the penalty function is designed.

Parameters A and ρ can be determined by the following procedures:

1. Determine the bandwidth parameter σ in (102). σ is determined by considering the distance range to be considered. For example, in the MMA example case, we use $\sigma = 0.3118$, which is 1.5% of normalized distance range ($6\sqrt{m_0}$).

2. The buffer zone, inside which no penalty term is assigned, is determined by setting $\|x_0 - x_i\|_2^2 = \sigma^2$ in (103) calculating the corresponding ρ from (102).

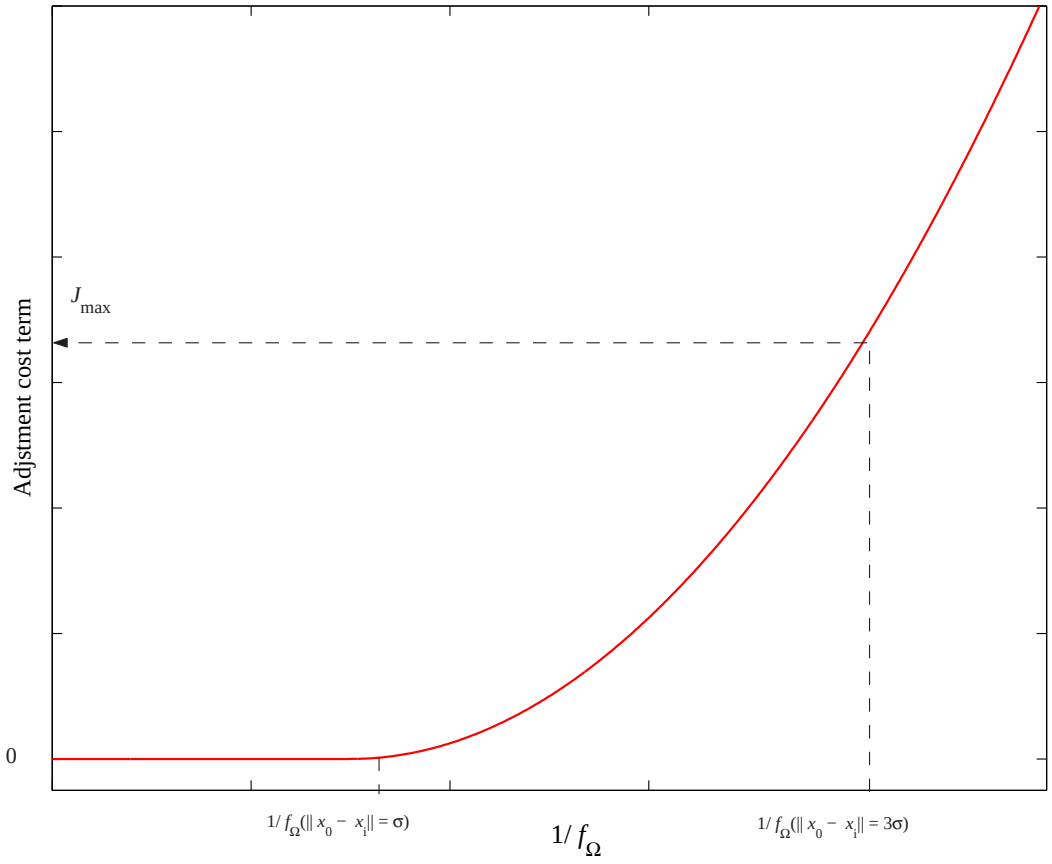

Figure 16: Quadratic penalty adjustment term.

3. A, which controls the rate of increasing penalty term, is determined by

$$J_{\max} = A \cdot H\left(\frac{1}{f_\Omega(x_0)} - \rho\right) \cdot \left[\frac{\frac{1}{f_\Omega(x_0)} - \rho}{\rho}\right]^2 \tag{107}$$

where $J_{\max}$ is some large cost-to-go value, and $f_\Omega(x_0)$ is determined by setting $\|x_0 - x_i\|_2^2 = (3\sigma)^2$.

5.3 Modified ADP Strategy

In this section, we describe how the penalty function is incorporated into the aforementioned ADP strategy. In the off-line value iteration, one solves the Bellman equation in the following recursive manner:

$$J^{i+1}(x) = \min_u \mathbb{E}\left[\phi(x,u) + \alpha \tilde{J}^i\left(f(x,u)\right)\right] \tag{108}$$

Each iteration step in the minimization involves estimating the cost-to-go $\tilde{J}^i\left(f(x,u)\right)$ for a candidate u using a local averager based on the stored cost-to-go data. Without a penalty term, the minimization is done with no regard to the accuracy of the cost-to-go estimate and the solution may lie in a region with little data present. In Chapter 4, we saw that this is especially a problem for a process with a high dimensional state space like the MMA polymerization reactor example.

It is important to note that, even though we solve the minimization with the bias term (Equations (104) and (105)) included, the cost-to-go value we record for each state in the training set after the minimization is solved should not include the extra penalty term. This is because the large penalty term can accumulate with the iteration and can start biasing the cost-to-go values for the entire state space.

The modified ADP algorithm with the penalty function can be summarized as follows:

1. Perform closed-loop simulations and sample the resulting state trajectories to form a training data set, X_{sam}.

2. Evaluate the initial cost-to-go value for each state in X_{sam} based on the data.

3. Improve the cost-to-go for each state in X_{sam} by evaluating (108). In solving the

minimization, adjust the objective function for each candidate u according to (104) and (105). $\tilde{J}$ is estimated using a local averager like the k-nearest neighbor averager.

4. Record the cost-to-go value $\tilde{J}$ (not $\tilde{J}_{\text{aug}}$) corresponding to the solution for each state in X_{sam}. Note that the penalty term is not to be included in the recorded cost-to-go value.

5. Repeat steps 3–4 until a convergence criterion is met.

6. With the converged cost-to-go, real-time calculation of a control action is done by solving

$$\min_u \mathbb{E}\left[\phi(x,u) + \alpha \tilde{J}^*_{\text{aug}}(f(x,u))\right] \tag{109}$$

with adjustment of the objective function according to (104) and (105).

We also note that the local data density is evaluated using (102) with neighboring points ($N = k$) only. This is a small number compared to the entire data set and thus obviates the heavy computational requirement associated with a nonparametric data density estimator.

5.4 Control of Continuous MMA Polymerization Reactor

In this section, we revisit the example of MMA polymerization reactor presented in the previous chapter. Without any guard against excessive extrapolation, the ADP controller based on the converged cost-to-go values could not regulate the process. We also compare the performance of the ADP controller with those of the slMPC and the nonlinear programming based MPC (NMPC) [61]. Model equations and parameter values are found in the previous chapter. The control objective is to drive the conversion and weight-average molecular weight to the set-point of $[0.2\ 15]^T$. The activation energies for dissociation and propagation are assumed to change in a deterministic manner at time 0 so that they are also included in the state vector. In reality, they would have to be estimated from measurements.

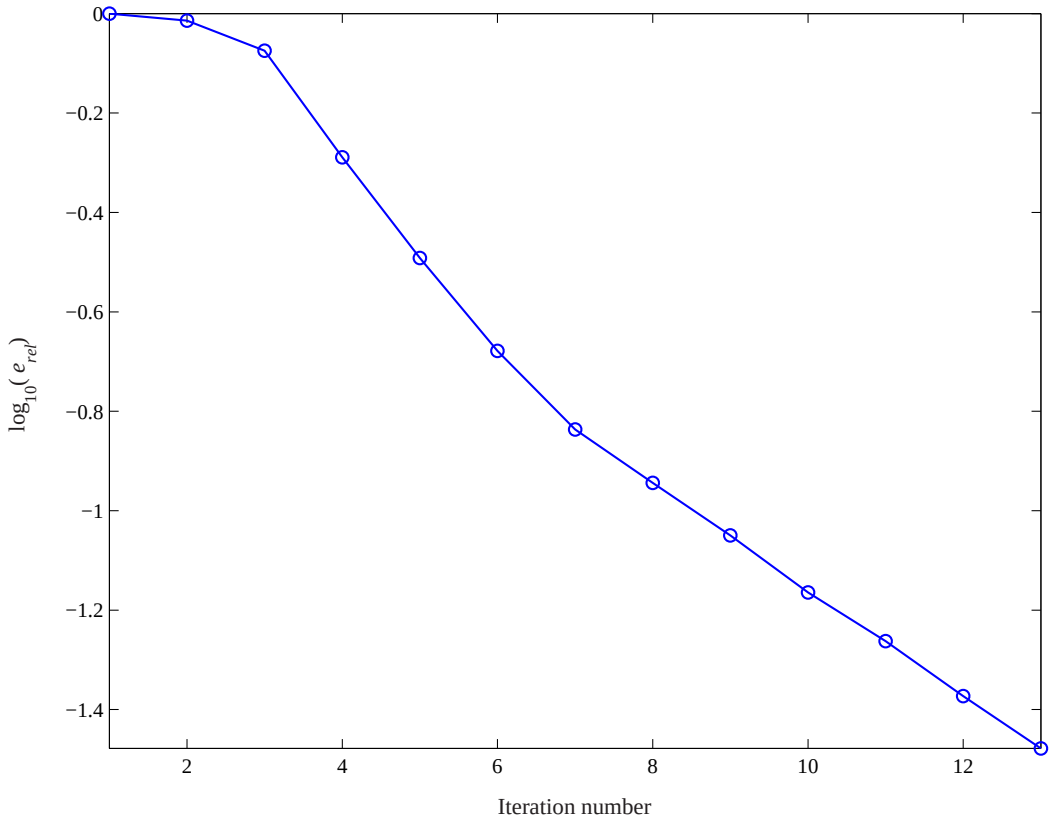

Figure 17: MMA reactor: off-line iteration trends using a penalty function.

5.4.1 The Modified ADP Approach

The same data set (composed of 3594 points) obtained from the closed-loop simulations under the slMPC policy was used for estimation of the initial cost-to-go values. A distance-weighted k-nearest neighbor method with $k = 4$ was employed for the cost-to-go evaluation in the minimization. For the Parzen density estimator, the bandwidth parameter σ was chosen as 0.3118, which was 1.5% of the range of normalized distance. Given $J_{\max} = 10^5$, the corresponding threshold value ρ and the parameter A were calculated as 7.1765×10^{-8} and 34.8, respectively. Value iteration was performed with the penalty function suggested in the previous section, and the cost-to-go function converged after 13 iterations with $e_{rel} < 0.03$ as shown in Figure 17.

Table 11: Cost incurred during on-line operation under different policies.

Case	Ed/Ed_0	Ep/Ep_0	slMPC	NMPC	ADP
1	1.05	1.00	12878	11888	11318
2	1.05	1.03	7080	6491	6265
3	1.05	1.05	4516	4124	3869
4	1.03	0.95	9424	8601	8406
5	1.03	0.97	6310	5744	5620
6	1.03	1.00	3124	2853	2775
7	1.03	1.03	1317	1213	1184
8	1.03	1.05	641	599	567
9	1.00	0.95	594	559	592
10	1.00	0.97	239	232	238
11	1.00	1.00	28	26	28
12	1.00	1.03	39	38	39
13	1.00	1.05	133	130	132
14	0.97	0.95	176	164	175
15	0.97	0.97	321	315	321
16	0.97	1.00	713	683	712
17	0.97	1.03	1362	1295	1338
18	0.97	1.05	2001	1915	2214
19	0.95	0.97	1532	1456	1505

5.4.2 Comparison of On-line Performance

The converged cost-to-go was implemented as $\tilde{J}^*$ for on-line optimal control using (109). Note that we are dealing with a deterministic system, which does not require evaluation of any expectation operator. The performance is compared using slMPC, NMPC, and the modified ADP controller. Table 11 shows on-line cost incurred under the three controllers for 19 disturbance scenarios. The on-line cost was calculated using

$$\sum_{k=0}^{\infty} \phi(x(k), \mu(x(k))) \tag{110}$$

$$\phi(x(k), u(k)) = (r - y(k+1))^T \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} (r - y(k+1))^T + \Delta u^T(k) \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix} \Delta u(k) \tag{111}$$

The NMPC was formulated with prediction horizon of 5 and control horizon of 1, which was found to be a tractable nonlinear optimization problem using the subroutine `fmincon` of MATLAB. The results show that in most cases the modified ADP approach can improve the performance from the starting control policy significantly and for some cases the performance was even better than those of the particular NMPC policy, especially for the cases

where the calculated trajectories were in regions where data is rich as shown in Figures 18 and 19. However, in the case 18, the ADP controller performs worse than the starting control policy (slMPC). This was because the policy learned to explore aggressively while there is only one trajectory generated by the slMPC in the nearby region. This phenomenon is clearly shown in the state space plot of x_3 and x_4 in Figure 20. However, the penalty function helped stabilize the on-line control eventually. By reducing the ρ to the density of 0.5% of normalized distance range from a query point, we could trap the state trajectory very close to that of slMPC, leading to the same on-line performance with that of slMPC. Hence, the parameters of penalty function can be designed to control the level of exploration by user's choice. The observation suggests that the ADP strategy may benefit from simulation data covered by several different control policies or input dithering schemes, which can allow the ADP strategy to derive a more improved control policy within a larger subset of the state space.

We also tested for the "fresh" step disturbance cases that were not included in training data set, and similar trends of performance improvement with closed-loop stability were observed with those of the 19 cases.

5.5 Conclusions

Use of a penalty function was proposed to control extrapolation of the learned cost-to-go data within the approximate dynamic programming strategy. In the minimization of off-line value iteration and on-line control, the penalty function based on an estimate of local data density systematically biases the search to those regions of adequate data density, thereby "trapping" the solution within them. The continuous MMA polymerization reactor was used to illustrate the efficacy of the modified approach: The modification provided a control policy that regulated the system successfully and significantly improved on-line performances were obtained compared to the starting control policy in most cases. We also noted that simulations with different scenarios and control policies will enlarge the domain of search and help improve the solution significantly.

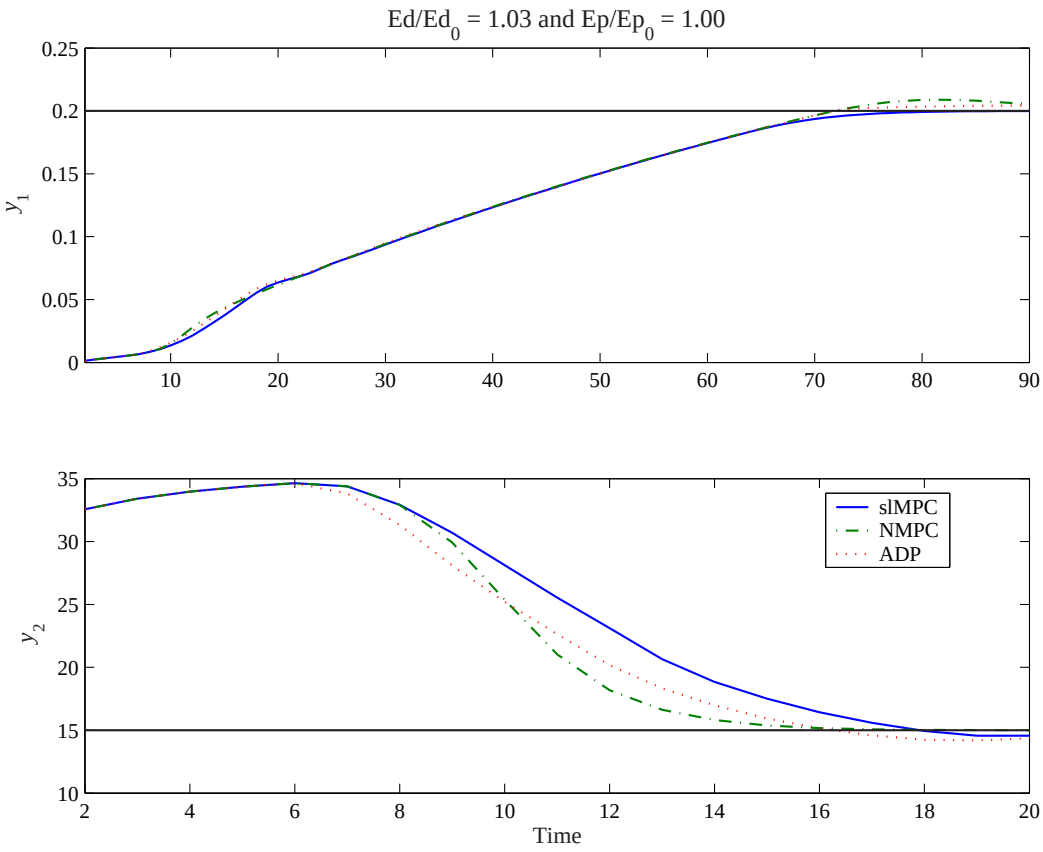

Figure 18: Output trajectories of different control policies for case 6.

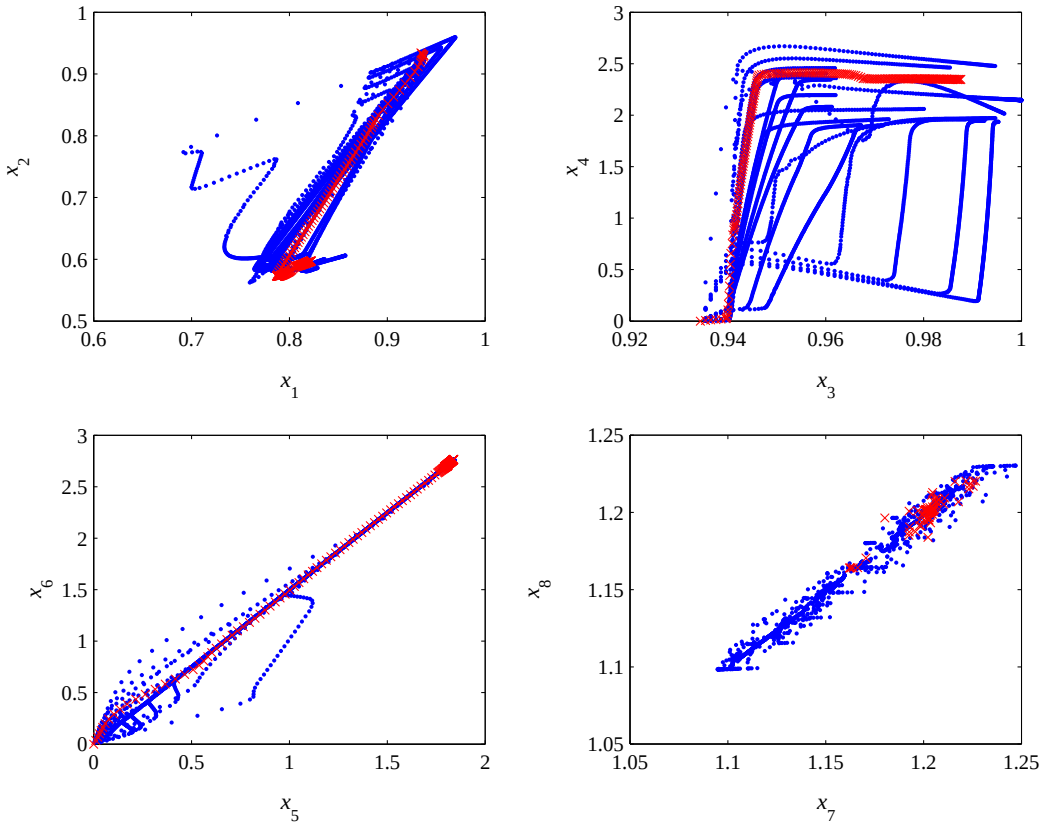

Figure 19: State trajectories of ADP for case 6. X's denote the on-line state trajectory and dots denote simulation data.

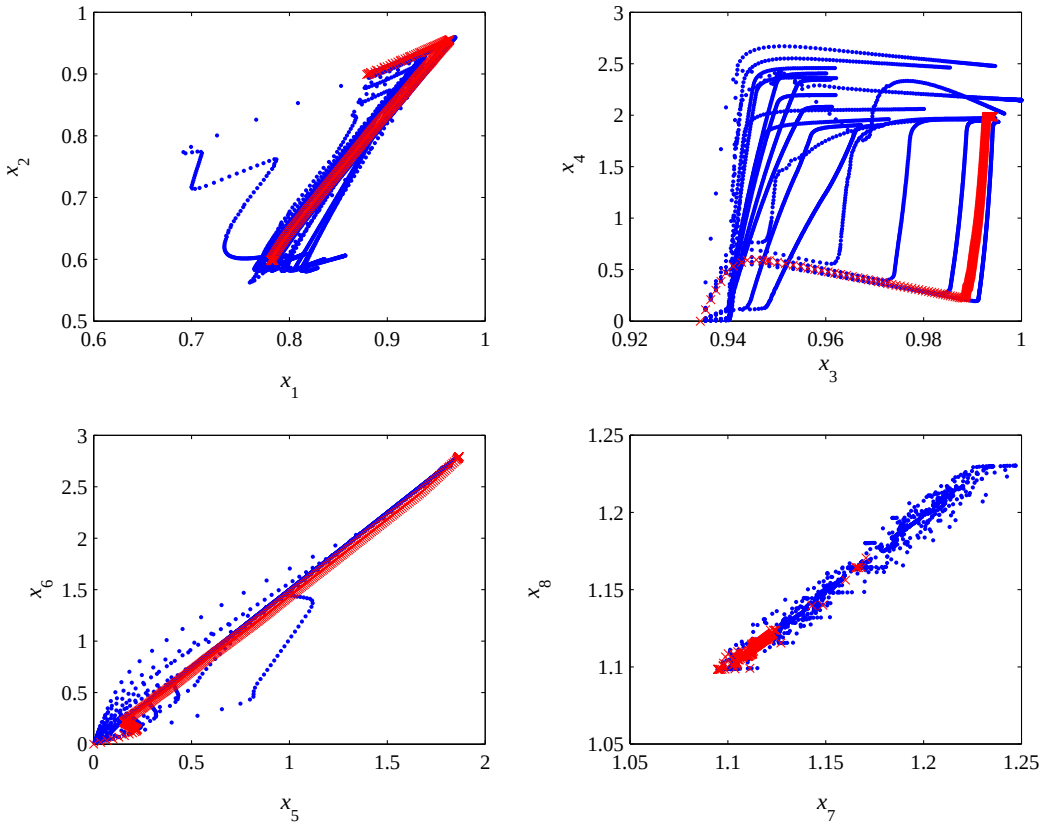

Figure 20: State trajectories of ADP for case 18. X's denote the on-line state trajectory and dots denote simulation data.

CHAPTER VI

STOCHASTIC OPTIMAL CONTROL: DUAL ADAPTIVE CONTROL

In this chapter we solve a stochastic dual adaptive control problem using the ADP strategy. The optimal dual control law can be obtained via dynamic programming, but solving the Bellman equation is analytically and computationally intractable using conventional solution methods that involve sampling of a complete hyperstate space. We show that the ADP method, when judiciously applied, generates a dual control policy that takes into account accuracy of current and future parameter estimates, yet is computationally amenable. An integrating process with an unknown gain is used for illustration.

6.1 Introduction

Practical control problems are characterized by mismatches between model and plant, which can be caused by structural/parametric uncertainties and unknown exogenous disturbances. MPC solves an open-loop (oftentimes deterministic) optimal control problem on a future time horizon, with a feedback update occurring at each time step. However, this approach can lead to highly suboptimal results for systems with uncertainties. Uncertainties may be modeled using either deterministic bounds or as stochastic processes. In the case of deterministic bounds, min-max control formulations have been proposed. While several algorithms have been proposed that guarantee robust stability, most of them are based on repeating open-loop optimal control calculations and therefore the result can be highly conservative. Recently, some closed-loop formulations have been put forward [56, 62, 16], but these algorithms are either computationally intractable or based on very limited assumptions on how uncertainties enter the model. For example, most algorithms assume that the uncertainties can vary with time in an arbitrary manner within assigned bounds. This may be conservative as most uncertainties show strong correlations in time.

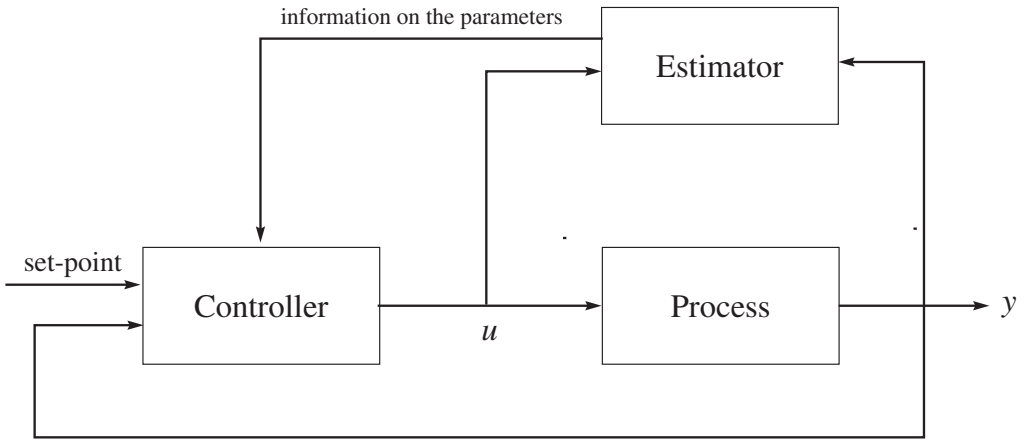

Figure 21: Adaptive control system.

In the case of stochastic parameters, the usual approach is to combine parameter estimation and control into an adaptive control strategy as shown in Figure 21. The estimator block delivers information about the unknown parameters, such as their mean values and covariances. Different classes of adaptive controllers are obtained depending on how the information is utilized. The most popular approach is to perform a control calculation by assuming that the estimated parameters are true values, which is referred to as the 'certainty equivalence' approach. This approach, however, disregards uncertainties in the parameter estimates and can lead to severe robustness problems like the "bursting" phenomenon. In addition, the disregard of the coupling between the estimation and control makes the learning "passive," meaning the controller does not make exploratory moves to actively generate information about important parametric uncertainties.

To obtain useful information about the process dynamics, it is necessary to perturb the process in general. On the other hand, such a perturbation may not be favorable from a viewpoint of closed-loop performance. Thus, there is a conflict between information gathering and present control quality. This problem was first introduced and discussed by Fel'dbaum in his series of papers published in the early 60s [39, 40, 41, 42]. The optimal controller has *dual* goals, meaning it should balance between control and exploration. By gaining more process information when needed, better control performance can be achieved in the future. Fel'dbaum also showed that Dynamic Programming (DP) should be solved to

obtain the optimal solution to the dual control problem. It has been thought that the DP solution for the problem is intractable, and only a few simple examples have been solved this way after reducing the problem size through some analytical insights into the specific problem [8]. Because of the computational complexity, most dual control problems have been approached by introducing cautious and active probing features to simpler suboptimal controllers in a somewhat ad hoc manner [31, 68].

We employ the ADP approach to solve the dual optimal control problem. The approach enables us to combine the merits of the different starting policies systematically through interpolation and improvement of cost-to-go values in the state space. If successful, the derived ADP-based controller should show a well-balanced dual feature. It will be shown that a nonparametric local averager is a good choice for function approximation for a highly complex and nonlinear cost-to-go structure in this problem's context. This chapter is organized as follows: In Section 6.2 we present a dual adaptive control problem. Section 6.3 discusses a general procedure for applying the ADP approach to the dual optimal control problem. An example of integrator with an unknown gain is presented in Section 6.4. Section 6.5 concludes the chapter.

6.2 Stochastic Adaptive Control

6.2.1 Problem Formulation

We consider a discrete time model described as

$$x(k+1) = f(x(k), u(k), \theta(k), \zeta(k)) \tag{112}$$

where $x(k)$ is a state vector, which is assumed to be measured, $u(k)$ is a manipulated input vector, $\theta(k)$ is a vector containing unknown parameters of the model, and $\zeta(k)$ is an exogenous noise, which we assume here to be independently identically distributed (i.i.d.) Gaussian. We also assume that the structure f is known and the unknown parameter θ is also described by a Gaussian process.

The control objective is to minimize an infinite horizon cost:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \alpha^t \phi(x(k+t), u(k+t)) \middle| \xi(k)\right] \tag{113}$$

where α is a discount factor, and expectation operator $\mathbb{E}$ is taken over the distribution of ζ and θ. $\xi(k)$ is an information state (or *hyperstate*) at time k, which includes the process state $x(k)$ and the first two moments of the a posteriori probability density function of the Gaussian parameter vector.

$$\xi(k) = \left[x(k),\ \hat{\theta}(k),\ P(k)\right]^T \tag{114}$$

where $\hat{\theta}$ and P are the conditional mean and the covariance matrix of θ (conditioned by the measurements), respectively. A feasible control policy is the one that determines $u(k)$ based on the information available at time k (i.e. $\xi(k)$).

A closed-loop optimal solution to (113) assumes that the future inputs are determined in a feedback-optimal sense, which means they are dependent on the future hyperstates (which themselves are stochastic variables). The optimal control policy can be derived by solving the following stochastic dynamic programming:

$$J^*(\xi(k)) = \min_{u(k)} \mathbb{E}\left[\phi(x(k), u(k)) + \alpha J^*(\xi(k+1))|\, \xi(k)\right] \tag{115}$$

Note that $\xi(k+1)$ is a stochastic variable and is affected by the choice of control action $u(k)$. The difficulty in solving the above is that the minimization, which requires expectation calculation for each evaluation of a candidate u, must be solved for all the points in a densely gridded hyperstate space. The control action influences the immediate cost ϕ, quality of future estimation (reflected through future hyperstate ξ), and future control performance. Even though the optimal controller will have the desired dual feature, the DP formulation is intractable in all but simplest cases if a conventional solution approach (e.g., value iteration, policy iteration) is taken.

6.2.2 Passive Learning Policies

This section introduces popular "passive" control policies, the certainty equivalence (CE) control policy and the cautious control policy. The CE policy calculates a control action at each sample time as if the estimate $\hat{\theta}(k)$ were exact:

$$u_{CE}(k) = \mu_{CE}(x(k),\ \hat{\theta(k)}) \tag{116}$$

The inputs are designed without any regard for their effects on future estimation quality, which can make the achieved performance substantially suboptimal and cause intermittent instability phenomenon known as 'bursting' [4].

A simple design that takes into account the uncertainty is to minimize the cost function of (113) only for a single step. Note that, for a single step problem, u can be optimized as a deterministic variable. The resulting controller is called a *cautious controller*. It adds a measure of caution to account for uncertainty in that the gain in the controller is decreased as the uncertainty increases. It does not, however, take into account the effects of a control action on future estimation quality, and can lead to *turn off* of the controller if the uncertainty gets too large. The cautious policy is also a passive learning controller because there is no active probing signal generated to improve the identification.

6.3 ADP Implementation

In this section, we describe the ADP procedure for solving the previously described stochastic control problem. Due to the difficulties in computing the exact solution to the DP formulation, several approximate solutions have been proposed [139]. One of them is to find an approximate solution for two-step ahead cost-to-go function [68, 69]. It is, however, still very complex and is restricted to simple problems. The ADP approach instead solves for a discounted infinite horizon formulation for a reasonable number of points in the hyper-state space, which are sampled from closed-loop simulations of different controller types, and hence should be more generally applicable. Based on the basic algorithm presented in Section 5.3, the practical implementation steps are as follows:

1. Perform closed-loop simulations.

   Since the ADP algorithm derives an improved control policy from the data visited by starting policies, it is preferable to simulate with different control policies having the characteristics of cautiousness and active exploration. For example, passive controllers with dither signals can be used for the simulation.

   Input dithering is a randomized policy, where a white noise signal is typically added to the control action. With a high noise level, uncertainty can be reduced significantly.

Despite its simple implementation, the systematic design of a good noise level is still difficult, and the blind randomization with respect to the current uncertainty information is not the optimal way to explore.

2. Approximation of the initial cost-to-go values.

   We use a local averager introduced in Chapter 4. Note that the expectation operator is not explicitly evaluated in this step, but the function approximator should smoothen the stochastic nature giving a good estimate of the expected cost-to-go value. However, this is not so critical because the off-line iteration step will refine the cost-to-go with explicit evaluation of the expectation operator.

   We also note that it is generally difficult to fit an artificial neural network to the data generated from a stochastic system. In our previous study involving neural networks [63], use of prior knowledge on the cost-to-go function was necessary to obtain an acceptable approximation. In contrast, the local approximator provides much more robust results, as will be shown in the example.

3. Improvement of cost-to-go estimates using value iteration.

   This step is complicated by the expectation operator coupled with the minimization. The expectation operator is evaluated by sampling the *innovation* term, which is also affected by the control action (See the example for more details). We not only sample the control actions used in the suboptimal control policies but discretize the actions with a reasonable grid size. Each candidate action gives probability distribution of the corresponding innovation, according to which the possible outcomes of hyperstate are sampled using Monte Carlo simulations. A penalty function is also designed for guarding against over-extrapolations of the available cost-to-go data.

   We also note that it is desirable to perform sufficient number of simulations under dithered policies as well as to perform the sampling and averaging cautiously. This is because a few "outliers" could significantly bias the average cost-to-go due to the penalty term. In the example studied in the next section, we left out the sampled outliers in averaging, if the total number of the outliers is less than 10% of the entire

sample set. This will not be necessary if the simulations covered all the possible realizations.

4. With the converged cost-to-go values, real-time calculation of control action is done by solving the minimization on-line, where the expectation operator is evaluated in the same manner as described in the value iteration step.

6.4 Example: An Integrator with Unknown Gain

6.4.1 Problem Statement

Consider an integrator process [8] described by

$$y(k+1) = y(k) + bu(k) + e(k+1) \tag{117}$$

where $y(k)$ is the output, $u(k)$ is the manipulated input, $e(k)$ is a white noise, and b is an unknown parameter. e and b follow the normal distributions.

$$e \sim \mathcal{N}(0, \sigma^2) \tag{118}$$

$$b \sim \mathcal{N}\left(\hat{b}(0), P(0)\right) \tag{119}$$

Furthermore, the unknown parameter b can vary in time and its behavior is modeled as

$$b(k+1) = b(k) + w(k) \tag{120}$$

where $w(k)$ is also a Gaussian white noise.

The control objective is to minimize the following discounted infinite horizon objective function:

$$\mathbb{E}\left[\sum_{t=k+1}^{\infty} \alpha^{t-(k+1)} \left[y(t)\right]^2 \middle| \mathcal{Y}_k\right] \tag{121}$$

where $\mathcal{Y}_k$ denotes the sequence of observed outputs and inputs available at time k. Given the measurements $\mathcal{Y}_k$, the estimator generates the conditional probability distribution of the parameter b. Given that b is a Gaussian distribution, the conditional distribution is represented by its mean and covariance defined as follows:

$$\hat{b}(k) = \mathbb{E}\left\{ b(k) \middle| \mathcal{Y}_k \right\} \tag{122}$$

$$P(k) = \mathbb{E}\left\{ \left[\hat{b}(k) - b(k)\right]^2 \middle| \mathcal{Y}_k \right\} \tag{123}$$

They can be calculated recursively according to

$$\hat{b}(k+1) = \hat{b}(k) + K(k)\left[y(k+1) - y(k) - \hat{b}(k)u(k)\right] \tag{124}$$

$$K(k) = \frac{P(k)u(k)}{\sigma^2 + P(k)u^2(k)} \tag{125}$$

$$P(k+1) = \left[1 - K(k)u(k)\right]^2 P(k) + \sigma^2 K^2(k) + R_w \tag{126}$$

where R_w is the variance of w. Then, the hyperstate of the process, $\xi(k)$, is defined as

$$\xi(k) = \left[y(k),\ \hat{b}(k),\ P(k)\right]^T \tag{127}$$

6.4.2 Simulation Scenarios

In most cases, the following CE controller is nearly-optimal for the given problem:

$$u(k) = -\frac{y(k)}{\hat{b}(k)} \tag{128}$$

Let us first consider a simple but somewhat idealistic case where the gain b can jump from the initial value of 0.5 to a value between -15 and 15 (except 0) and the initial parameter value is assumed to be known exactly so that the estimator is initiated with a covariance of $P(0) = 0$. The covariance of the exogenous noise term is set as 1 ($\sigma = 1.0$) but in the particular realization we simulate it is kept to as a zero signal up to some time period ($t = 100$), during which a parameter jump occurs at a certain time period ($t = 10$). We also assume (somewhat unrealistically but for the sake of simplicity in this first scenario) that the parameter change can be detected and the covariance in the estimator is reset to 200 at that point.

As shown in Figures 22 and 23, the CE controller can suffer from temporary instability (bursting) when the parameter uncertainty is large. The following cautious controller is derived by minimizing the one-step ahead cost-to-go function.

$$u(k) = -\frac{\hat{b}(k)}{\hat{b}^2(k) + P(k)} y(k) \tag{129}$$

Though the cautious controller includes the uncertainty parameter P, the controller can *turn off* itself when the uncertainty becomes large. The phenomenon is displayed in Figure 24.

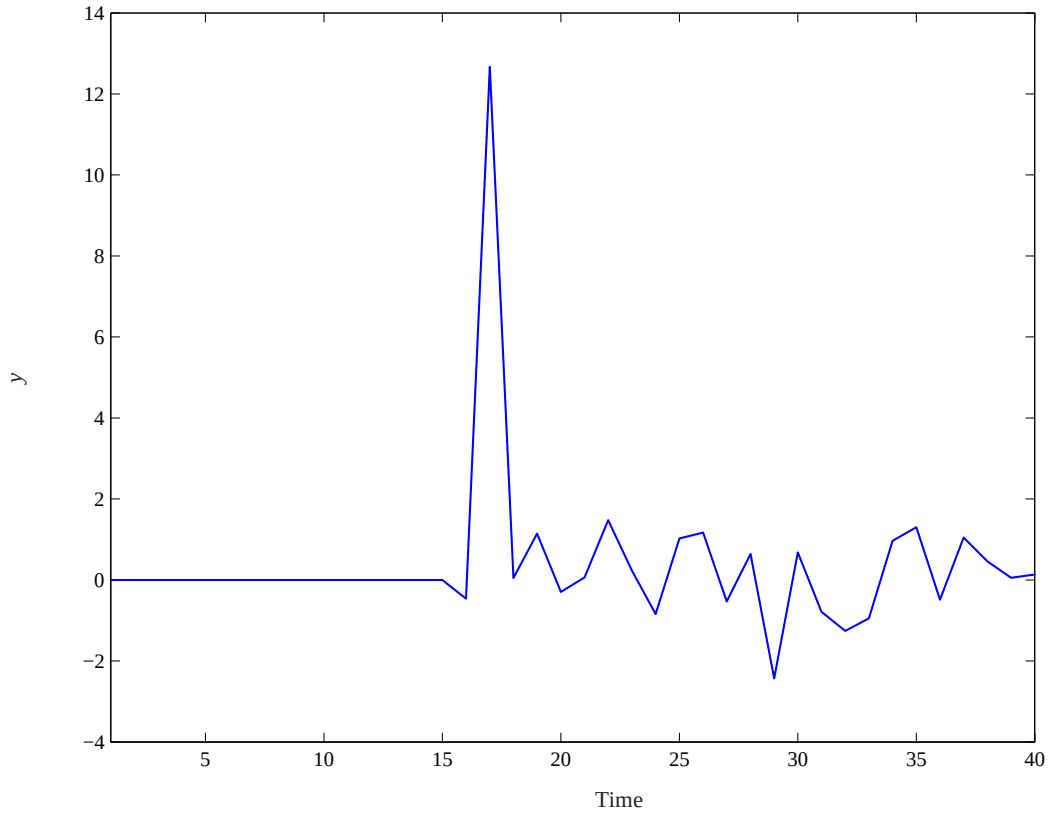

Figure 22: Output of the CE controller when b jumps from 0.5 to 15 at time 10 and the measurement noise enters at time 15.

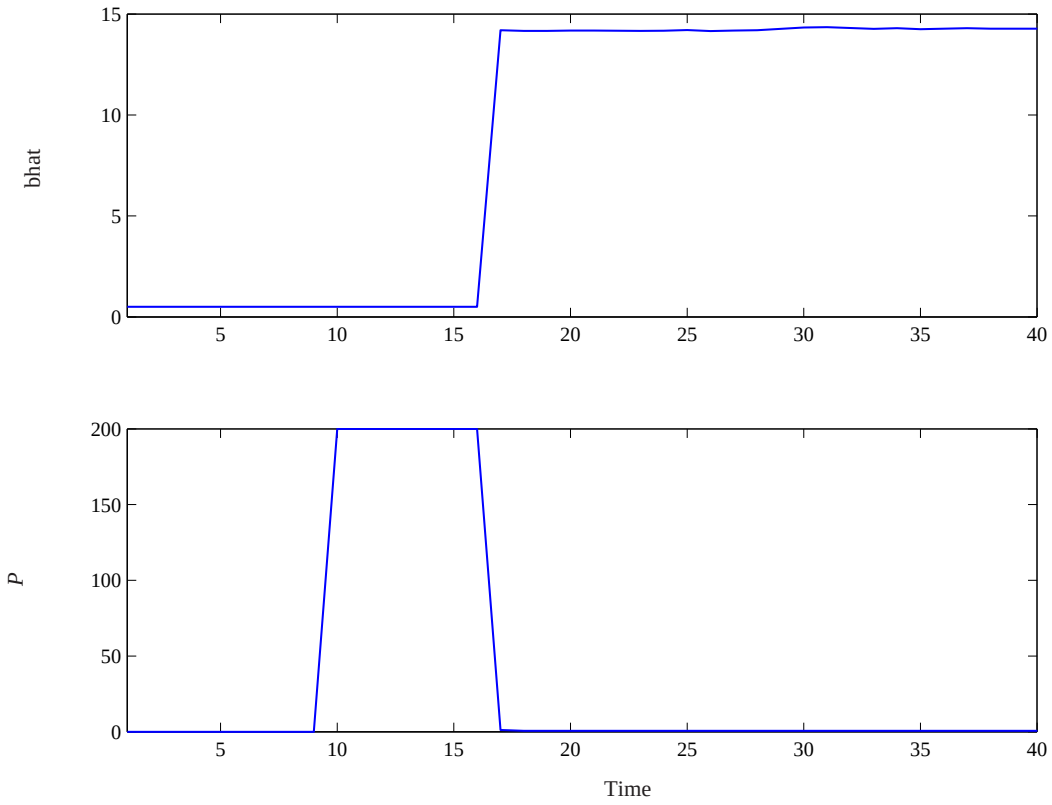

Figure 23: $\hat{b}$ and P of the CE controller when b jumps from 0.5 to 15 at time 10 and the measurement noise enters at time 15.

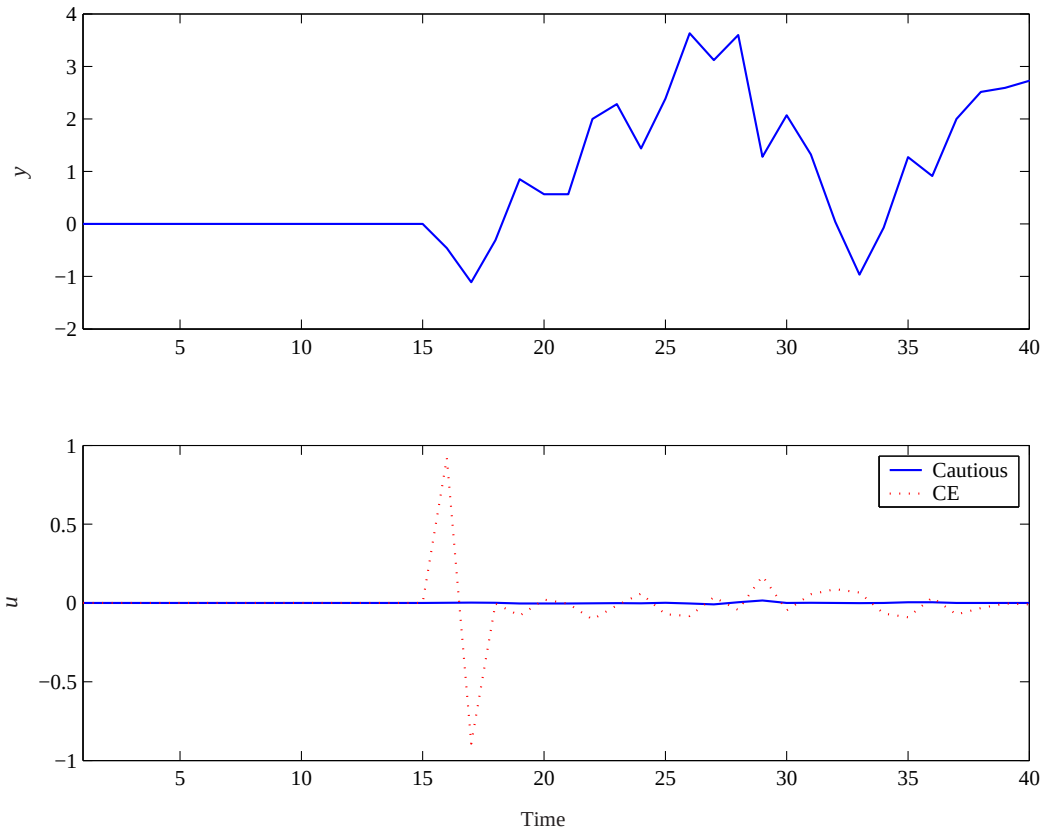

Figure 24: Input and output of the cautious controller when b jumps from 0.5 to 15 at time 10 and the measurement noise enters at time 15.

Because of the turn off of the control signals after the covariance is reset, output deviates significantly due to the integration of exogenous noises.

6.4.3 ADP-based Controller

6.4.3.1 Data Generation

For generation of training data (hyperstate vs. cost-to-go), closed-loop simulations were performed using the following control policies: (1) The CE controller, (2) the cautious controller (3) the CE and cautious controllers with dithered inputs. We simulated parameter jumps from the nominal value to $b = \pm 5, \pm 10, \pm 15$. The dither signals were randomly generated from the uniform distribution of $[-0.1\ 0.1]$. Three sets of the dither signals were injected at regular intervals during quiet periods. Each set of dither signals lasted for 4 sample times. Three and five realizations of e were simulated for non-dithered policies and dithered policies, respectively. Three separate realizations of the input dithering were also

simulated for each realization of the dithered policies. The parameter was modeled as a constant for this scenario, which means we set $R_w = 0$, and the variance of σ was set as 1. The total number of simulation data obtained was 3849.

6.4.3.2 Value Iteration

In the value iteration step, we solve

$$J^{i+1}(\xi(k)) = \min_{u(k)} \mathbb{E}\left[y^2(k+1) + \alpha \tilde{J}(\xi(k+1))\right] \tag{130}$$

where $\alpha = 0.98$ was used. The expectation operator was evaluated by sampling 50 innovation values ($\epsilon(k)$) for each action candidate $u(k)$ used for the optimization.

$$\epsilon(k) = y(k+1) - y(k) - \hat{b}(k)u(k) \tag{131}$$

has the following distribution:

$$\epsilon(k) \sim \mathcal{N}\left(0,\ 1 + u(k)^2 P(k)\right) \tag{132}$$

As shown in Figure 25, the value iteration step converged after 24 runs with

$$e_{rel} < 0.03 \tag{133}$$

A distance weighted k-nearest neighbor estimator was used for the cost-to-go approximation with $k = 4$, and the quadratic penalty function of (105) was designed with the parameter choices of $A = 0.87,\ \rho = 0.047,\ \sigma = 0.35,\ J_{\max} = 2500$.

6.4.3.3 On-line Performance

Different parameter jump cases were simulated to compare the ADP policy with the suboptimal control policies. The parameter jump cases that had not been simulated for generating the training data set were also tested. For each case, the total cost over 50 sample times ($\sum_{t=1}^{50} y^2(t)$) were calculated. The total cost averaged over 10 realizations are compared in Table 12. Whereas the average performance of the ADP controller does not vary much with different parameters, the other control policies suffer from bursting or turn off phenomena, leading to poor average performances.

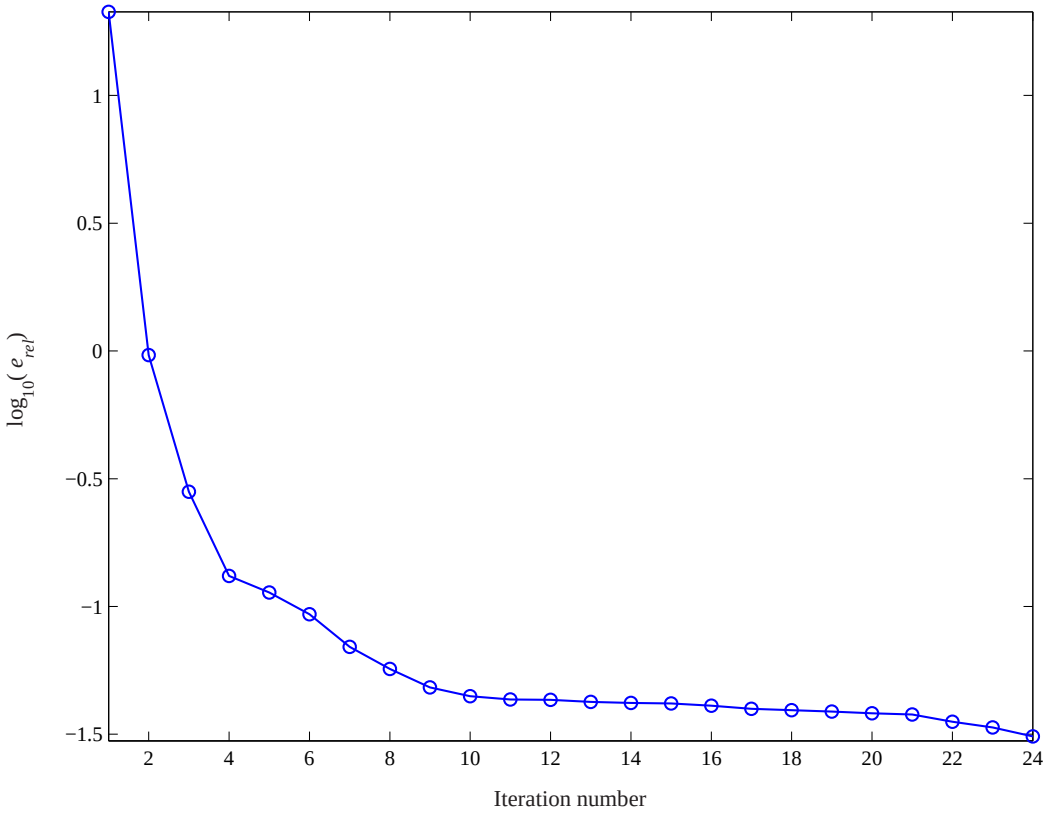

Figure 25: Convergence behavior of the value iteration.

Table 12: Averaged cost over 50 sample times with 10 realizations of e.

b	CE	Cautious	Dithered CE	Dithered Cautious	ADP
15	630.5	152.3	63.1	79.8	52.9
-15	936.7	179.8	116.0	93.3	50.7
10	194.6	169.6	99.7	85.5	66.3
-10	184.1	163.9	156.0	64.3	56.7
5	68.4	142.9	41.2	113.7	56.8
-5	72.0	130.5	83.4	64.7	48.0
12	630.1	109.3	60.0	52.3	51.2
-12	401.5	85.8	875.0	51.7	46.6
7	125.9	126.8	60.0	83.8	46.6
-7	345.1	167.4	84.1	65.2	60.7

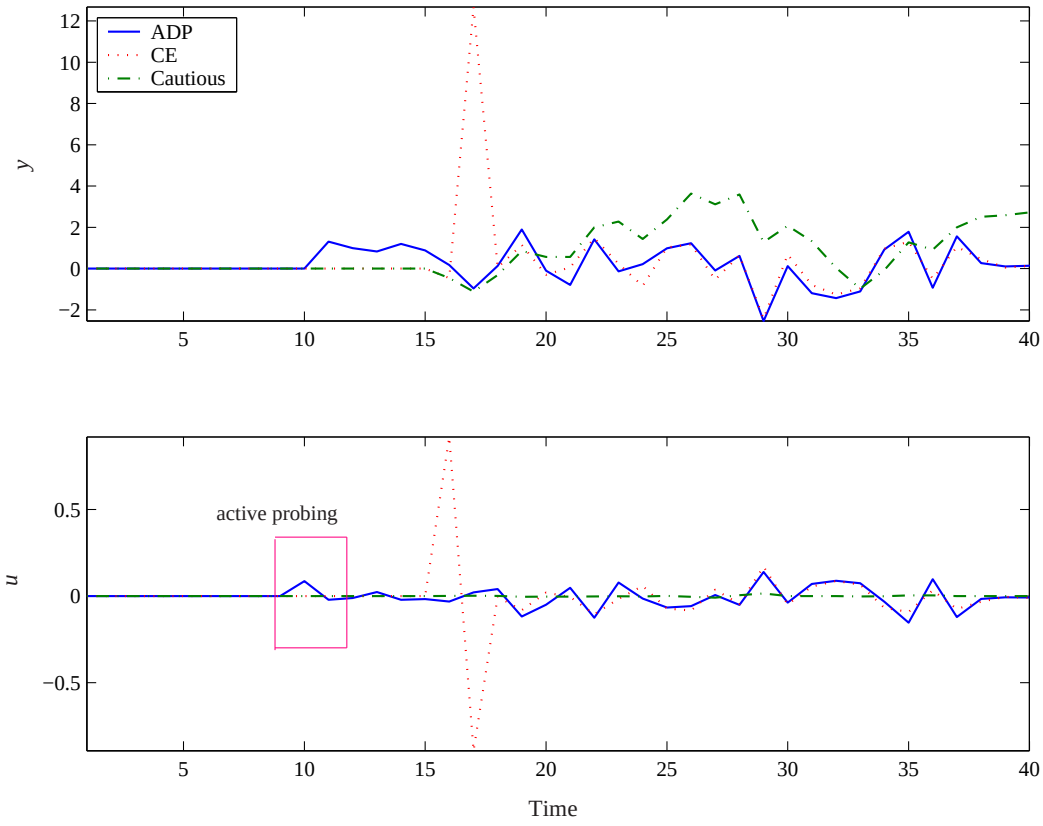

Figure 26: A sample run of the parameter jump case ($b = 15$): y and u.

The performance disparities were observed during the transient period when the parameter jump occurred and exogenous noises entered the system. Figures 26 and 27 show sample results of the output regulation and estimation under the three policies (CE, Cautious, and ADP). At time 10, b jumps from 0.5 to 15 and the covariance is reset to 200. White noise e enters the system at time 15. Figure 26 shows that the ADP controller injects the probing signal at time 10 and achieves the best overall performance of regulation, whereas the passive policies do not move the control actions until time 15, and the performances are degraded either by bursting of the output or by turn off of the control signals.

6.5 Conclusions

In this chapter, we solved a stochastic optimal control problem that has a dual objective of identification and control using the ADP approach. Starting from different control policies, including several passive and randomized policies, the ADP approach could derive a

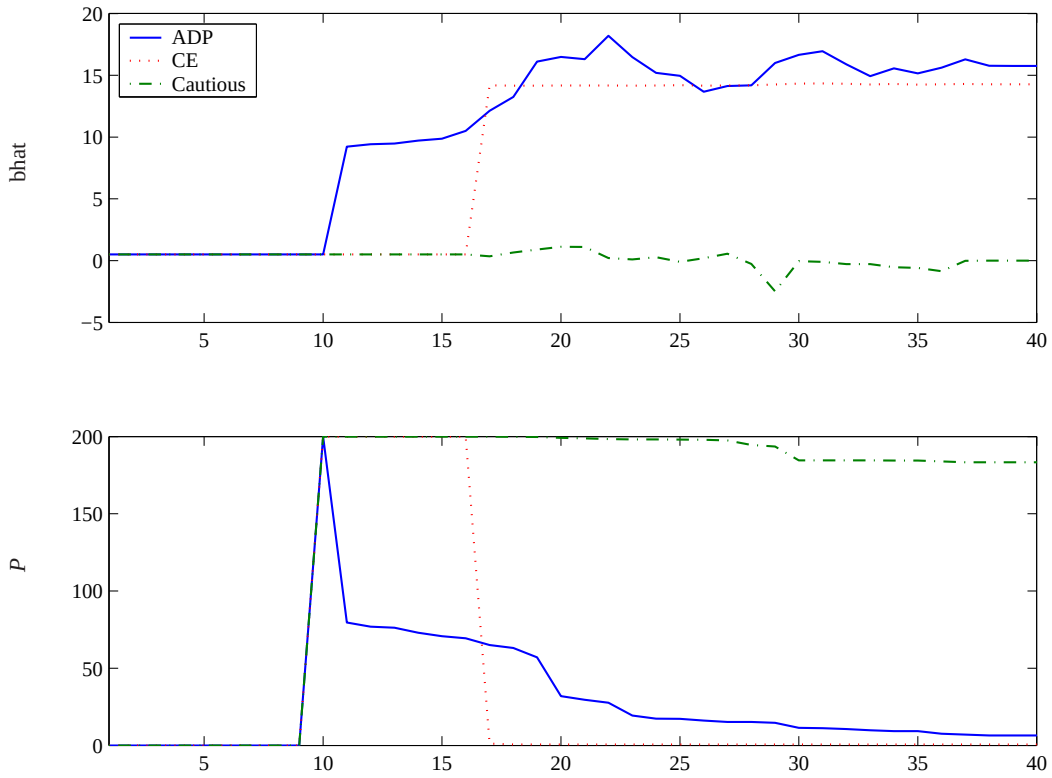

Figure 27: A sample run of the parameter jump case ($b = 15$): $\hat{b}$ and P.

superior control policy that actively reduces the parameter uncertainty, leading to a robust performance. Our experience also indicates that sufficient number of simulations with dithered signals during the transient period must be performed in order for the approach to learn a policy with the desired dual feature.

CHAPTER VII

SIMULATION-BASED DUAL MODE CONTROLLER FOR NONLINEAR PROCESSES

This chapter presents a simulation-based strategy for designing a nonlinear override control scheme to improve the performance of a local linear controller. The higher-level nonlinear controller monitors the dynamic state of the system under the local controller and sends an override control action whenever the system is predicted to move outside an acceptable operating regime under the local controller. For this purpose, a cost-to-go function is defined, an approximation of which is constructed by using simulation or historic operation data. The cost-to-go function delineates the "admissible" region of state space within which the local controller is effective, thereby yielding a switching rule. The same cost-to-go function can also be used to calculate override control actions designed to bring the system state back into the admissible region as quickly as possible. The proposed scheme is demonstrated and discussed with nonlinear examples.

7.1 Introduction

MPC is being widely used in the process industry because of its ability to control multivariable processes with hard constraints. Most of the current commercial MPC solutions are based on linear dynamic models, which are easier in terms of identification and on-line computation [95]. On the other hand, many chemical processes exhibit strong nonlinearities. This disparity has prompted several studies on MPC formulations with nonlinear system models [60]. Since most nonlinear MPC (NMPC) formulations require on-line solution of a nonlinear program (NLP), issues related to computational efficiency and stability of a control algorithm have received much attention.

The initial focus was on formulating a computationally tractable NMPC method with guaranteed stability. Mayne and Michalska [75] showed that stability can be guaranteed by

introducing a terminal state equality constraint at the end of prediction horizon. In this case, the value function for the NMPC can be shown to be a Lyapunov function under some mild assumptions. Because the equality constraint is difficult to handle numerically, Michalska and Mayne [78] extended their work to suggest a dual-mode MPC scheme with a local linear state feedback controller inside an elliptical invariant region. This effectively relaxed the terminal equality constraint to an inequality constraint for the NMPC calculation. The dual-mode control scheme was designed to switch between the NMPC and the linear feedback controller depending on the location of the state. Chen and Allgöwer [30] proposed a quasi-infinite horizon NMPC, which solves a finite horizon problem with a terminal cost and a terminal state inequality constraint. The main difference from the Michalska and Mayne's method is that a fictitious local linear state feedback controller is used only to determine the terminal penalty matrix and the terminal region off-line, and switching between controllers is not required.

These NMPC schemes have theoretical rigor but have some practical drawbacks. First, these methods still require solving a multi-stage nonlinear program at each sample time. Assurance of a globally optimal solution or even a feasible solution is difficult to guarantee. Second, the optimization problem for determining the invariant region for a local linear controller and the corresponding terminal weight are both conservative and computationally demanding.

Motivated by the drawbacks and the industry's reluctance to adopt full-blown NMPC, we propose an override (or supervisory) control strategy for monitoring and improving the performance of a local controller. Our method is similar to the dual-mode MPC suggested by Michalska and Mayne in that the switch between two different control policies depends on current location of the state. However, we employ a cost-to-go function based approach instead of NMPC. First, a cost-to-go function under the local controller is defined, which serves to delineate the admissible region within which the local controller can effectively keep the system inside acceptable operating limits. The same cost-to-go function is also shown to facilitate the calculation of override control actions that will bring the system outside the admissible region back into the region as quickly as possible. We propose to

use simulation or historic data to construct an approximation to the cost-to-go function. With the cost-to-go function, an override control action can be calculated by solving a single stage nonlinear optimization problem, which is considerably simpler than the multi-stage nonlinear program solved in the NMPC.

7.2 Simulation-Based Construction of an Override Controller

The proposed scheme uses either simulation or actual plant data to identify the region of the state space, in which the local controller can effectively keep the system inside an acceptable operating regime (defined by some inequalities in the state space). We do this by assigning to each state a cost-to-go value defined as

$$J^{\mu}(x_0) = \sum_{i=0}^{\infty} \alpha^i \phi(x_i) \tag{134}$$

where $J^{\mu}(x_0)$ is the cost-to-go for state x_0 under the local control policy $\mu(x)$, α is a discount factor, and $\phi(x_i)$ is a stage-wise cost that takes the value of 0 if the state at time i is inside the acceptable operating limit and 1 if outside when x_0 is the state at time 0. This way, if a particular state x_0 under the control policy evolves into a state outside the limit *in some near future* under the policy μ, the cost-to-go value will reflect it. On the other hand, those states that are not precursors of future violation of the operating limit will have negligible cost-to-go values. The latter states comprise the "admissible" region.

The cost-to-go function is approximated by first simulating the closed-loop behavior of the nonlinear model under the local linear controller for various possible operating conditions and disturbances. This generates x vs. $J^{\mu}(x)$ data for all the visited states during the simulation. Then the generated data can be interpolated to give an estimate of $J^{\mu}(x)$, $\tilde{J}^{\mu}(x)$, for any given x in the state space.

In the real-time application, whenever the process reaches a state with a significant cost-to-go value, it is considered to be a warning sign that the local controller's action will not be adequate. When this happens, an override control action is calculated and implemented to bring the process back to the "admissible" region where the cost-to-go is insignificant.

One can calculate such an action by implementing the override policy of

$$\text{if} \quad \tilde{J}^{\mu}(x_{t+1}(x_t, \mu(x_t))) \geq \eta, \quad u_t = \arg\min_{u'_t} \tilde{J}^{\mu}(x_{t+1}(x_t, u'_t)) \tag{135}$$

where η is a user-given threshold value for triggering the override control scheme. If no u'_t can be found such that $\tilde{J}^{\mu}(x_{t+1}(x_t, u'_t)) < \tilde{J}^{\mu}(x_{t+1}(x_t, \mu(x_t)))$, then $u_t = \mu(x_t)$ is used for the current sample time.

7.3 A Kernel-Based Approximator of Cost-to-Go Function

We use a kernel-based local averager to approximate the cost-to-go values, as empirical studies in Chapter 4 show that global approximators (e.g. neural network) are not good choices in general. In addition, it was shown that the local averager with the nonexpansion property is compatible with the DP operator and effective for representing local characteristics of state spaces.

Another reason for adopting the local averaging approach is our concern for grossly incorrect cost-to-go estimates that can arise from extrapolating to a region not accounted for in the simulation step. In implementing a risk-averse cost-to-go based controller, Kaisare *et al.* [52] used a feedforward neural network but gridded the state space in order to separate regions visited by simulation from those not. For those cells with little or no data, a high cost-to-go value was assigned to prevent the controller from driving the state trajectory into these uncertain regions. However, this is difficult to implement for cases with high dimensional state spaces.

For a convenient implementation of the risk-averse or risk-sensitive scheme, we propose to use a variation of Gaussian-kernel-based approximators.[1] This structure decides whether a reliable estimate can be given to a query point based on the available data. For a "reliable" query point it gives local weights calculated from a Gaussian kernel to give more influence on the regression to those training points closer to the query point than those farther away.

[1] This is a rudimentary version of the penalty function method in Chapter 5, which was developed later than this work.

Table 13: "Risk-averse" prediction using Gaussian-kernel-based approximator.

Prediction Algorithm
1. Is the query point x_0 in the memory?
a. Yes: Use the value in the memory.
b. No: Go to step 2.
2. Enumerate the data points inside r around the x_0.
Is the number of data points greater than k_{min}?
a. Yes: Average with the kernel.
b. No: $\tilde{J}(x_0)$ cannot be estimated. Assign $J_{\max}$ to x_0.

The suggested structure of kernel-based prediction is

$$\tilde{J}(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) J(x_i)}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)} \tag{136}$$

where

$$K_\lambda(x_0, x_i) = \exp\left(-\frac{\|x_0 - x_i\|_2^2}{\lambda^2}\right) \tag{137}$$

The number of neighbor points N is the number of data points inside a hypersphere, the radius of which is a user-given value r. In addition to r, there are other parameters that user should provide. These are the Gaussian kernel width λ, minimum number of data points inside the hypersphere k_{min}, and the high cost-to-go value $J_{\max}$ to be assigned to an "unreliable" query point. Table 13 describes how the estimate of cost-to-go value for a query point is calculated.

7.4 Illustrative Examples

7.4.1 Simple Nonlinear Example

7.4.1.1 Problem Description

We consider a system with two states, one output, and one manipulated input described by

$$\begin{aligned} x_1(k+1) &= x_1^2(k) - x_2(k) + u(k) \\ x_2(k+1) &= 0.8\exp\{x_1(k)\} - x_2(k)u(k) \\ y(k) &= x_1(k) \end{aligned} \tag{138}$$

with an equilibrium point of $x_{eq} = (-0.3898,\ 0.5418)$, $u_{eq} = 0$.

We also define the *acceptable* operating regime by

$$W(x) = \left\{(x_1 - x_{1eq}) + \sqrt{3}(x_2 - x_{2eq})\right\}^2 + \left\{\frac{(x_2 - x_{2eq}) - \sqrt{3}(x_1 - x_{1eq})}{0.3}\right\}^2 - 4 \leq 0 \quad (139)$$

A linear MPC controller was designed based on a linearized model around the equilibrium point. The control objective is to regulate y to y_{eq}. The linear MPC is used as the local controller with the following design:

$$\min_{\Delta u} \sum_{i=1}^{p} 5\bar{y}^2(k+i) + \sum_{l=0}^{m-1} \Delta\bar{u}^2(k+l) \quad (140)$$

with $p = 2$ and $m = 1$.

$$\begin{aligned} -3 \leq \bar{u} \leq 3 \\ \Delta\bar{u} \leq 0.2 \end{aligned} \quad (141)$$

The closed-loop behavior under the local controller starting at $x_0 = x_{eq} + [0.3\ 0.6] = [-0.0898\ 1.1418]$ is shown as dotted lines in Figure 28. Though the initial point is inside the operating limit, the system under the local linear controller violates the limit several times until the system is regulated to the equilibrium point.

7.4.1.2 Simulation-Based Design

To design the proposed override controller, closed-loop simulations under the local controller were performed using 347 initial points inside the operating limit. The simulations generated 17006 data points and cost-to-go values for each state in the trajectory were calculated using (134) with a value of $\alpha = 1$ and

$$\phi(x_t) = \begin{cases} 1 & \text{if} \quad W(x_{1t}, x_{2t}) \leq 0 \\ 0 & \text{if} \quad W(x_{1t}, x_{2t}) > 0 \end{cases} \quad (142)$$

Next step is to design a Gaussian-kernel approximator. Considering the coverage of state space, following parameters were chosen: $r = 0.05$, $k_{min} = 3$, $\lambda = 0.03$, $J_{\max} = 30$.

The actual value of cost-to-go is zero for the states inside the admissible region of a linear controller and outside the region the cost-to-go will be over unity. This makes the structure of cost-to-go function very stiff. However, the approximator will smoothen out

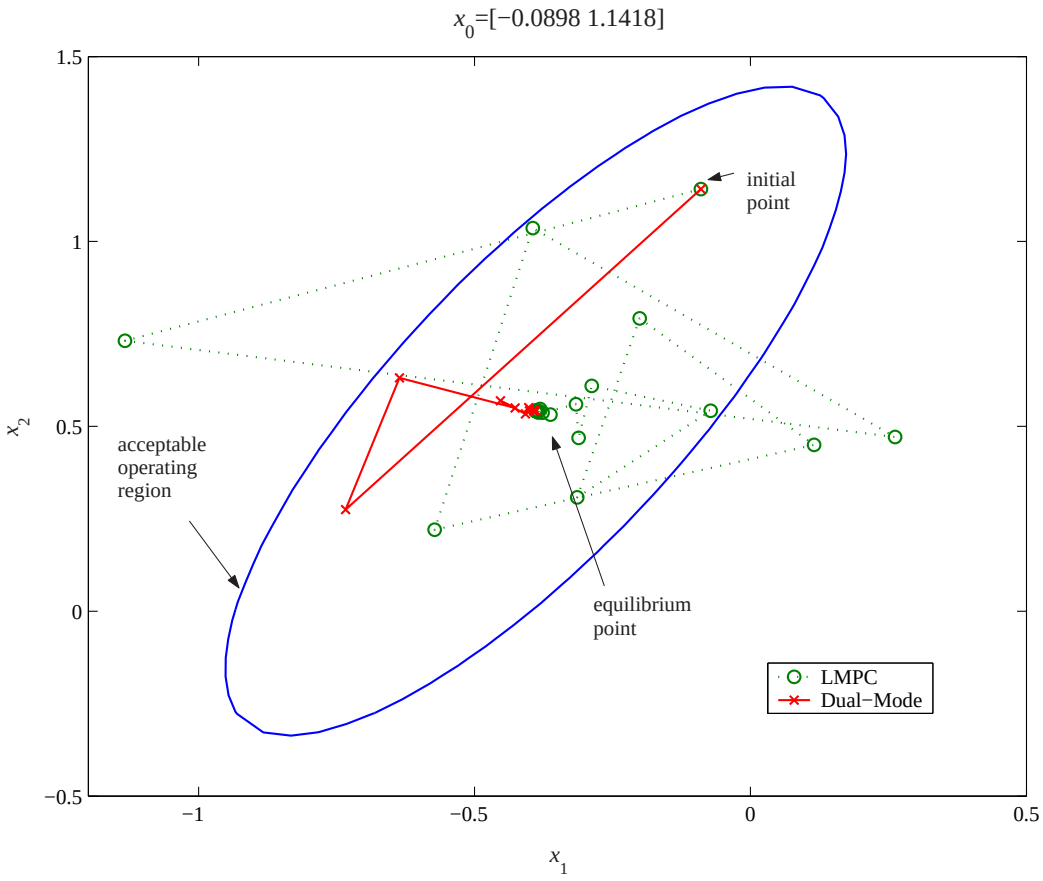

Figure 28: State trajectories under local MPC and dual-mode controller.

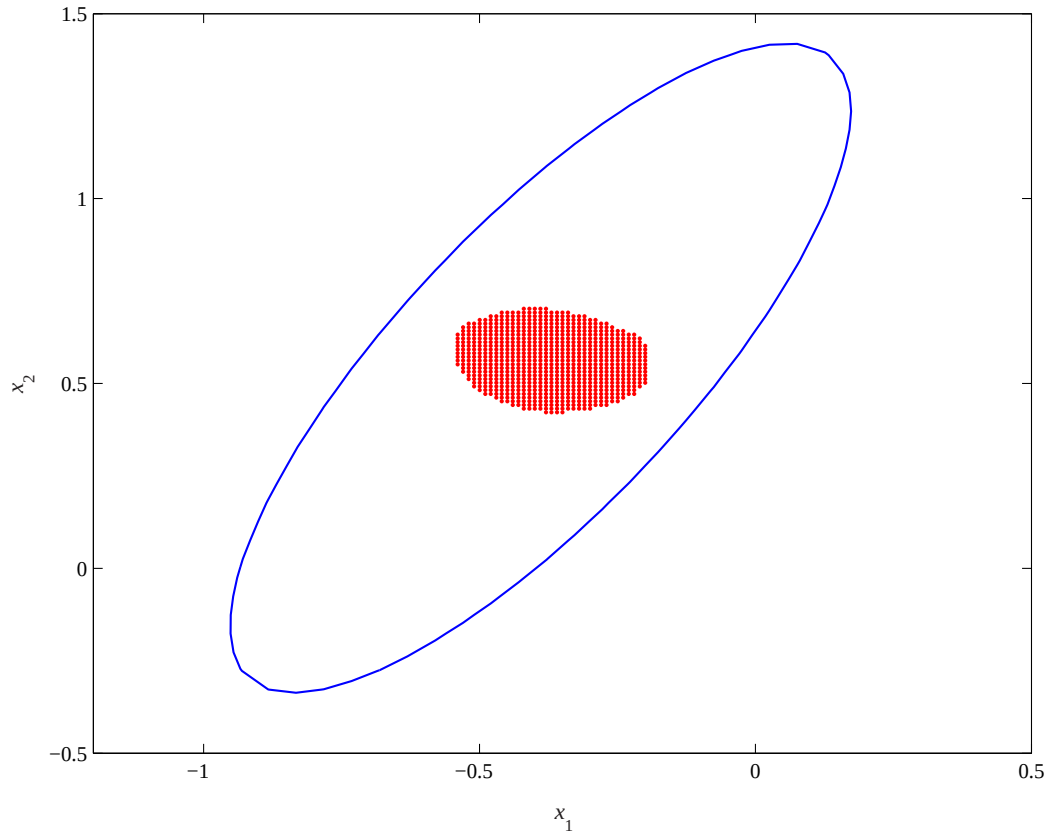

Figure 29: Regions under local controller with $\tilde{J}(x) < 0.02$.

the stiff structure a bit by averaging. Therefore small tolerance value ($\eta = 0.02$) was chosen to illustrate a possible shape of the admissible region under the local controller, which is illustrated in Figure 29.

7.4.1.3 Real-Time Application

To compare on-line performances of the local controller alone and the dual mode controller (i.e. the local controller combined with the proposed override controller), eight initial points different from the training set were sampled. We also compare the proposed dual-mode controller with the successive linearization based MPC (slMPC) scheme suggested in [61]. Finally, we also simulated the LMPC and the slMPC with the state constraints of $-0.95 \leq x_1 \leq 0.2$ and $-0.35 \leq x_2 \leq 0.45$ (denoted by scLMPC and scslMPC). The prediction and control horizons of slMPC are the same as those of the LMPC.

The solid lines in Figure 28 is the state trajectory with the same initial point under

Table 14: Comparison of performances (total # of limit violations).

Test pt	LMPC	slMPC	scLMPC	scslMPC	Override
1	diverged	5	diverged	diverged	0
2	3	3	diverged	diverged	0
3	2	0	0	diverged	0
4	2	0	0	diverged	0
5	0	0	diverged	diverged	0
6	0	0	0	diverged	0
7	7	15	1	diverged	0
8	diverged	diverged	diverged	diverged	0

Table 15: Model parameters: bioreactor example.

μ_{max}	0.53 hr^{-1}	k_m	0.12 g/l
k_1	0.4545 l/g	Y(yield)	0.4
D_s, x_{2fs}	0.3 hr^{-1}, 4.0 g/l	x_s	[0.9951 1.5123]

the dual-mode controller. For the first three points, the override control actions were used instead of those of LMPC's. The proposed scheme successfully steers the state back to the region with lower cost-to-go values. Table 14 shows the sum of stage-wise cost (the total number of violation of operating limit) and the suggested control design outperforms for all the test points. We can also see that imposing state constraints did not work here as many infeasible solutions were returned, eventually causing divergence.

7.4.2 Bioreactor Example

In this section, we consider a bioreactor example with two states: biomass and substrate [18]. With a substrate inhibition for growth rate expression of biomass, the system shows multiple steady states. To operate at the unstable equilibrium, closed-loop control must be used. The system equation is

$$\begin{aligned} \frac{dx_1}{dt} &= (\mu - D)x_1 \\ \frac{dx_2}{dt} &= D(x_{2f} - x_2) - \frac{\mu x_1}{Y} \\ \mu &= \frac{\mu_{max} x_2}{k_m + x_2 + k_1 x_2^2} \end{aligned} \tag{143}$$

where x_1 is biomass concentration and x_2 is substrate concentration. Table 15 shows the parameters for the model at the unstable steady state.

7.4.2.1 Local Linear Controller

A linear MPC was designed based on a linearized model around the unstable equilibrium point with sample time of $0.1h$. The control objective is to regulate x to x_s at the equilibrium values and the manipulated variables are the substrate concentration in the feed x_{2f} and the dilution rate D. The parameters of the LMPC controller are $Q = 100I,\ R = 10I,\ p = 10$, and $m = 5$, where I is a 2 by 2 identity matrix, Q is a state weighting matrix, and R is an input weighting matrix.

We also define an acceptable operating region as

$$\begin{aligned} W(x) &= \left\{\frac{0.52(x_1 - x_{1eq}) + 0.85(x_2 - x_{2eq})}{7}\right\}^2 \\ &\quad + \left\{\frac{-0.85(x_1 - x_{1eq}) + 0.52(x_2 - x_{2eq})}{0.5}\right\}^2 - 1 \leq 0 \end{aligned} \tag{144}$$

which is shown in Figure 30. The input constraints for MPC is

$$\begin{aligned} 0 \leq D \leq 0.5 \qquad & |\Delta D| \leq 0.2 \\ 0 \leq x_{2f} \leq 8 \qquad & |\Delta x_{2f}| \leq 2 \end{aligned} \tag{145}$$

The closed-loop behavior under the LMPC for different initial points are shown in Figure 30. As in the previous example, the LMPC cannot drive the state back into the equilibrium point without violating the operating limit.

7.4.2.2 Simulation-Based Dual Mode Controller

With the same definition of one-stage cost as in (142), a cost-to-go-based override controller was designed. For the simulation, 109 initial points were sampled inside the operating limit and closed-loop simulations under the LMPC yielded 21909 points. Parameters for a kernel-based approximator were chosen as: $r = 0.1$, $k_{min} = 5$, $\lambda = 0.05$, $J_{\max} = 50$, $\eta = 0.02$.

As in the previous example, the dual mode controller successfully navigated the state to the equilibrium point without violating the operating limit by searching for the path with lowest cost-to-go values. One of the sample trajectories tested is shown in Figure 31.

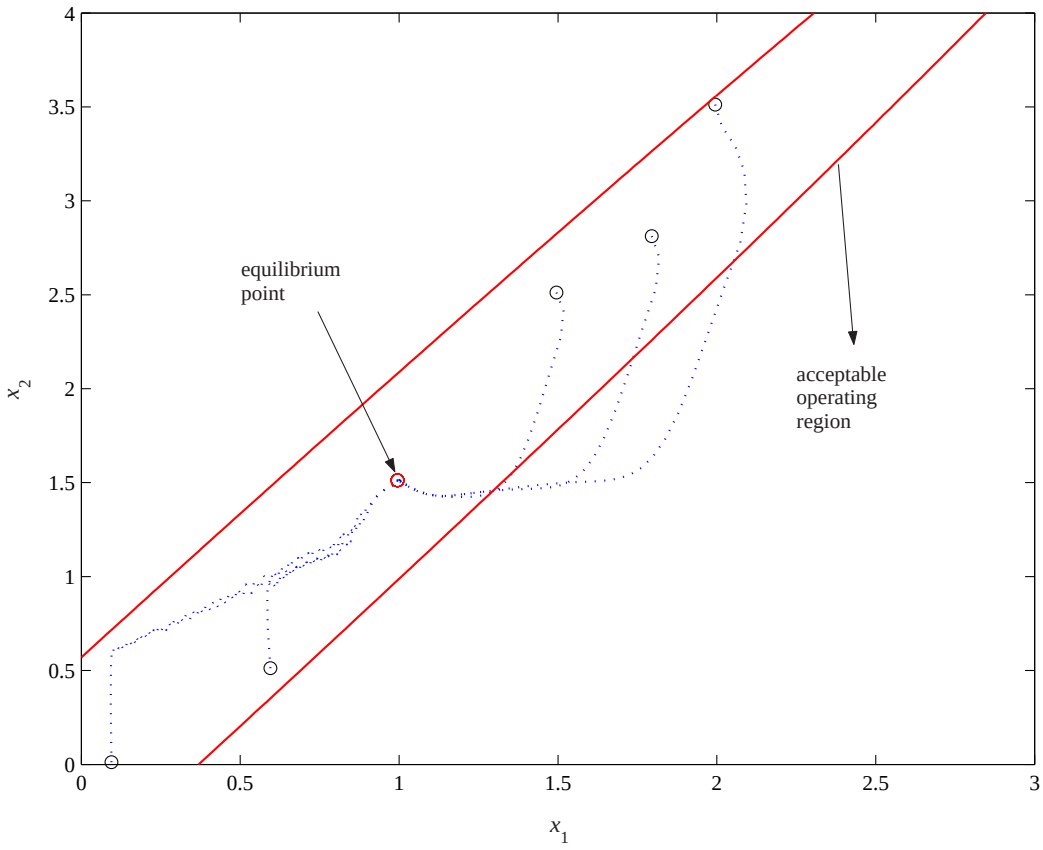

Figure 30: State trajectories under local MPC.

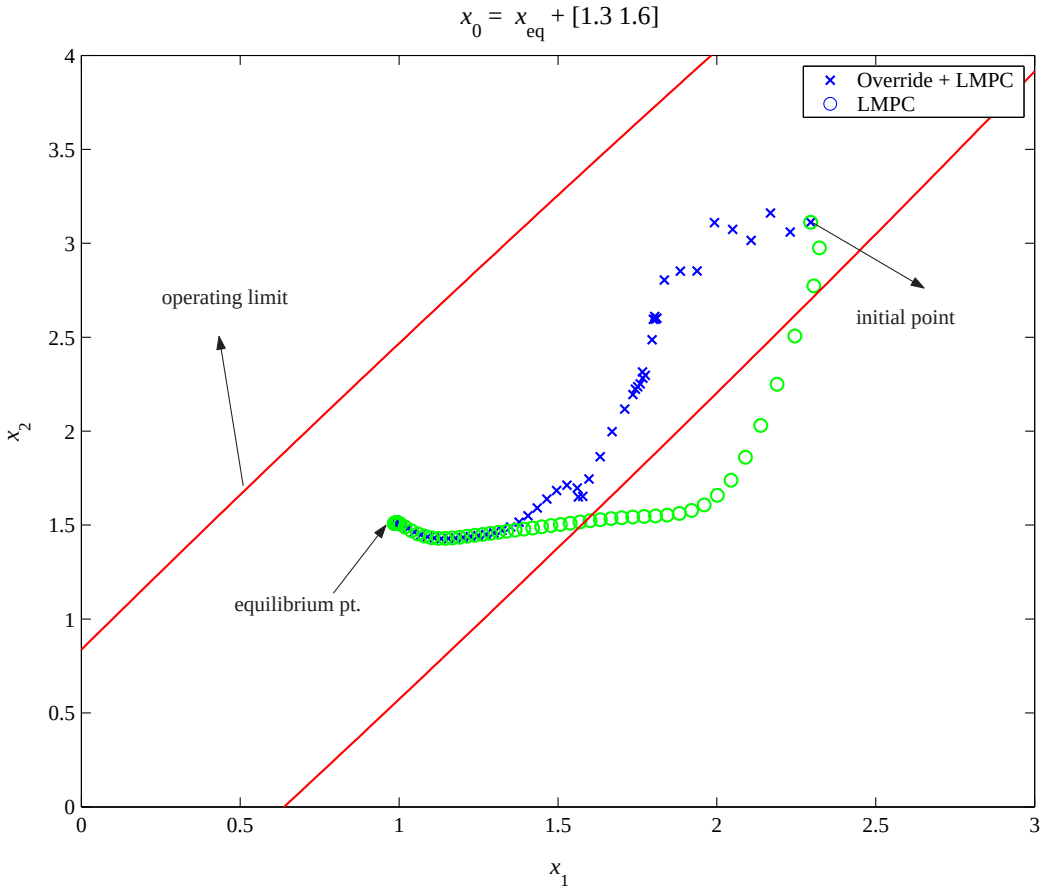

Figure 31: State trajectory under dual-mode controller.

7.5 Evolutionary Improvement of Cost-to-Go

Because the approximator employed in the calculation of override control action is based on the cost-to-go value of the local linear controller, it is not the optimal cost-to-go. The resulting override controller from the suboptimal cost-to-go approximation is also suboptimal. Hence, further improvement of the override control policy to steer the system back into the admissible region of the linear controller is possible by iteratively solving the following optimality equation (as in *value-iteration*) until $\tilde{J}$ converges.

$$J^{i+1}(x) = \min_u \left[\phi^i(x) + \tilde{J}^i(f(x,u)) \right] \tag{146}$$

where f is a state transition equation and i denotes iteration index.

For this purpose, the one-stage cost is re-defined differently as

$$\phi^i(x) = \begin{cases} 1 & \tilde{J}^i(x) \geq \eta \\ 0 & \tilde{J}^i(x) < \eta \end{cases} \tag{147}$$

With this change, the aim of the optimal control is to bring the system state back into the "admissible" region as quickly as possible. The value iteration was performed for the first illustrative example and the iteration converged after 5 steps with the following convergence criterion.

$$\|\tilde{J}^{i+1}(x) - \tilde{J}^i(x)\|_\infty < 0.1 \tag{148}$$

Figure 32 shows one of the state trajectory with the initial point of $x_0 = x_{eq} + [0.3\ 0.75]$ when the improved cost-to-go function is used in the override control calculation. As shown in the figure, the improved override controller brings the state back into the admissible region more efficiently than that based on the cost-to-go approximation under the LMPC.

7.6 Conclusions

A simulation-based override control scheme was shown to improve the performance and stability of a given local controller. The ease of design and implementation makes it a potentially appealing addition to an existing controller in industrial applications. The suggested framework can give operators indications on the future performance of the local controller and also suggest override control actions, if needed.

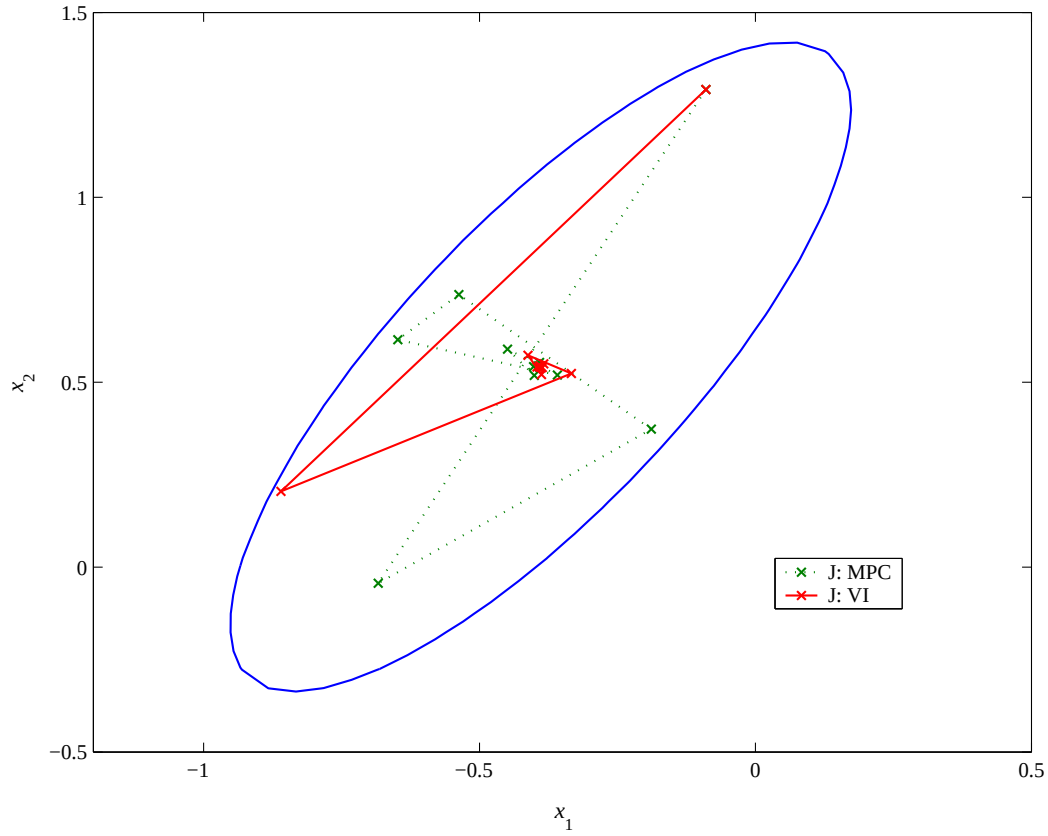

Figure 32: State trajectory with the dual-mode controller using improved cost-to-go.

CHAPTER VIII

INPUT-OUTPUT DATA-DRIVEN CONTROL OF NONLINEAR PROCESSES

In this chapter we propose two ADP based strategies for controlling nonlinear processes using input-output data. The first strategy, which we term 'J-learning,' builds an empirical nonlinear model using closed-loop test data and then uses it to derive an improved control policy by performing dynamic programming. The second strategy, called 'Q-learning,' tries to learn an improved control policy in a model-less manner. Compared to the conventional approach of building an input-output model and then designing and implementing a predictive controller based on it, the new approach brings some practical advantages besides the potential reduction in the on-line computational burden. Though many nonlinear input-output model structures have been proposed and used for control in the literature, models built in practice tend to be valid only within limited regions of the state space, mainly due to the lack of guidelines on test signal designs and presence of various limitations on plant testing. These prevent the full excitation of dynamic operating space, especially if the model order is chosen high to avoid significant bias. This difficulty in the identification step can translate into potential over-extrapolation of the model in the optimal control calculation step, leading to large mismatches between the actual closed-loop performance and that predicted by the model. For robust control, one must prevent such abusage of the model in the control calculation step, for example, by forcing the optimizer to restrict its search to regions of the state space with sufficient identification data. However, this is very difficult to implement within the multi-step predictive control setting. In the proposed ADP-based strategies, this issue can be handled conveniently by imposing the penalty term based on local data distribution. A diabatic CSTR example is provided to illustrate the proposed approaches.

8.1 Introduction

As discussed in Chapter 2, typical MPC formulations find a sequence of control actions by solving on-line minimization problems. Since the optimization is based on the predictions of future output behavior, an accurate model is essential for a successful outcome.

While the dynamic behavior of most chemical processes is nonlinear, linear models have been used predominantly in practice because of the difficulty associated with building an accurate nonlinear model. For a more widespread use of nonlinear model-based control, a control method tightly integrated with a *nonlinear system identification* strategy needs to emerge. Currently, popular nonlinear model structures studied in the literature include Volterra series models, block-oriented models such as Hammerstein and Wiener models, bilinear models, and NARX (Nonlinear AutoRegressive with eXogenous input) models [110].

To perform a nonlinear system identification, one must first decide on the system order. For example, for the NARX model of the form

$$\hat{y}(k+1) = f\left(y(k), \cdots, y(k-n_y), u(k), \cdots, u(k-n_u)\right) \tag{149}$$

one has to choose the order parameters n_y and n_u. This can be done by using methods like the False Nearest Neighborhood [96] and Akaike Information Criterion [3]. This is followed by choosing a parameterized functional structure of f, which can be a series expansion or a neural network. Finally, the parameters of f are determined through least squares estimation or similar regression methods. All three steps can contribute significantly to final model error. Given little prior knowledge one typically has in the beginning, the model order may have to be chosen high and the parameterization quite general to avoid bias, leading to an ill-conditioned estimation problem. This is particularly serious given the difficulty associated with designing and implementing persistently exciting signals for many nonlinear model structures. Lin and Jang [66] suggested an information theory based approach for designing data set to construct a reliable empirical model. However, implementation of such a design may be very expensive for practical problems. For example, most industrial processes can only be perturbed mildly around a few operating points (and transition trajectories), and this can result in insufficient information about many parts of the dynamic

state space [112]. Control actions and performance predictions calculated from the resulting model may not be reliable.

Since an empirical nonlinear model may be valid only in the regions of the state space where sufficient amounts of identification data were available, it is essential to exercise a caution in using it for model-based control. One option in predictive control is to restrict the search for optimal control moves to those keeping the system within the parts of the state space covered by the identification data. The question is how to do this in a systematic and computationally amenable manner. Leonard *et al.* [65] proposed a radial basis function network (RBFN) that computes the prediction reliability in terms of extrapolation and interpolation. The *validity index* was computed using clustered points with Parzen density estimator [90] for detecting over-extrapolation. Though developed for the RBFN-type structures, the idea of using a probability density estimator to define "reliable" search space seems attractive. Nevertheless, accommodation of such a measure of confidence within predictive control should be difficult and computationally demanding, as such measures must be applied to all time steps in the prediction window. Tsai *et al.* [127] proposed a coordinated architecture between a conventional MPC and an additional neural adaptive controller (NAC), which uses NAC for unexplored regions as detected by a density estimator. This approach can be inefficient and unreliable for highly nonlinear systems in that the structure requires additional training for the NAC design and simply combines control actions of MPC and NAC in a linear manner.

Another option is to continually update both the parameters and the structure of a given model on the basis of incoming data. Hernández and Arkun [46] used a recursive prediction error method to update a model. Chikkula and Lee [31] derived an input weighting function for a second-order Volterra model with stochastic parameters based on the calculation of an expected quadratic cost value. It is worth noting that most formulations for robust MPC, even those for linear systems, are based on *open-loop* performance objective, which can lead to very conservative control actions [62]. Furthermore, the on-line parameter update option can take a long transient time before enough data are collected to give the controller the needed robustness.

The idea of constructing cost-to-go function within the limited regions of state space and using it *cautiously* with a penalty function method seems a promising approach to addressing the problems of data-based nonlinear control. In Chapter 5, we showed that unreasonable extrapolations arising from the limited validity of the cost-to-go approximation can be curbed effectively in the single-stage optimization. In this work, building upon the suggested ADP framework, we present two different ADP based strategies for nonlinear control using input-output data alone. The first strategy builds and uses an empirical nonlinear model just like in the conventional MPC approach, but in both the cost-to-go approximation and on-line control calculation, extrapolation is controlled by adding a penalty term based on local data distribution. This way, the search for optimal control moves is restricted to keep the system within the parts of the state space with adequate data density. The second one aims at improving a given control policy in a continual manner without having to build a model. For both approaches, a localized approximator is employed for the cost-to-go function approximation. The approaches are illustrated with a diabatic CSTR example to highlight the difference with the conventional MPC approach.

8.2 Model Predictive Control Using a NARX Model

In this section, as a representative approach for using an empirical model for nonlinear control, we present a MPC formulation based on a NARX model structure, which is given as

$$y(k+1) = f\left(y(k), \cdots, y(k-n_y), u(k), \cdots, u(k-n_u)\right) + e(k+1) \tag{150}$$

where e is a white-noise. f can be parameterized in many different forms such as neural network, polynomial expansion, etc.

A state space realization of (150) can be constructed as

$$x(k) = \left[y(k), \cdots, y(k-n_y), u(k-1), \cdots, u(k-n_u)\right]^T \tag{151}$$

$$x(k+1) = \begin{bmatrix} 0 & \cdots & 0 & 0 & & & & \\ I & & & 0 & & & & \\ & \ddots & & \vdots & & & & \\ & & I & 0 & & & & \\ & & & & 0 & \cdots & 0 & 0 \\ & & & & I & & & 0 \\ & & & & & \ddots & & \vdots \\ & & & & & & I & 0 \end{bmatrix} x(k) + \begin{bmatrix} f(x(k),u(k)) \\ 0 \\ \vdots \\ 0 \\ u(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} I \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} e(k+1) \quad (152)$$

$$y(k) = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} x(k) \quad (153)$$

We denote (152) as

$$x(k+1) = F(x(k),u(k)) + Ge(k+1) \quad (154)$$

where

$$G = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix}^T \quad (155)$$

In using the model for process control, we need to compensate for a possible plant/model mismatch and ensure that the resulting controller will have the integral action. For this, we add to the output of the NARX model an integrated white noise term $\zeta(k)$. Hence the overall model becomes

$$x^{aug}(k) = \begin{bmatrix} x(k) \\ \zeta(k) \end{bmatrix} = \begin{bmatrix} F(x(k-1),u(k-1)) \\ \zeta(k-1) \end{bmatrix} + \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} e_1(k) \\ e_2(k) \end{bmatrix} \quad (156)$$

where $e_1(k)$ and $e_2(k)$ are independent white noises with covariance matrices R_1 and R_2, respectively.

The above can be compactly written as

$$\begin{aligned} x^{aug}(k) &= \mathcal{F}(x^{aug}(k-1),u(k-1)) + \Gamma^e e(k) \\ y(k) &= Hx^{aug}(k) + \nu(k) \end{aligned} \quad (157)$$

where

$$H = \begin{bmatrix} I & 0 & \cdots & 0 & I \end{bmatrix} \quad (158)$$

and

$$\Gamma^e = \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix} \tag{159}$$

Here artificial white noise ν is added in the model output for tuning purposes. The augmented state can be estimated using the following conventional EKF approach:

Prediction

$$\hat{x}^{aug}(k|k-1) = \mathcal{F}\left(\hat{x}^{aug}(k-1|k-1), u(k-1)\right) \tag{160}$$

$$\hat{y}(k|k-1) = H\hat{x}^{aug}(k|k-1) \tag{161}$$

Measurement correction

$$\hat{x}^{aug}(k|k) = \hat{x}^{aug}(k|k-1) + L(k)\left(\tilde{y}(k) - \hat{y}(k|k-1)\right) \tag{162}$$

$$\hat{y}(k|k) = H\hat{x}^{aug}(k|k) \tag{163}$$

EKF gain calculation

$$L(k) = \Sigma(k|k-1)H^T\left(H\Sigma(k|k-1)H^T + R^\nu\right)^{-1} \tag{164}$$

$$\Sigma(k|k-1) = \Phi(k-1)\Sigma(k-1|k-1)\Phi(k-1)^T + \Gamma^e R^e (\Gamma^e)^T \tag{165}$$

$$\Sigma(k|k) = \left(I - L(k)H\right)\Sigma(k|k-1) \tag{166}$$

where $\Phi(k-1) = \begin{bmatrix} A(k-1) & 0 \\ 0 & I \end{bmatrix}$, $R^e = \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}$, $A(k-1) = \frac{\partial F}{\partial x}\big|_{\hat{x}(k-1|k-1),u(k-1)}$, and $\tilde{y}$ is a measured output.

Given the state estimate at each time, a multi-step prediction of y can be computed recursively for the purpose of optimal control move calculation. A successive linearization based scheme such as the one suggested by Lee and Ricker [61] computes the output predictions using the nonlinear model under the assumption of constant input and then adds the effect of input changes based on a linearized model in order to obtain an affine prediction model with respect to the input moves. This yields a quadratic program (QP) instead of a nonlinear program (NLP) for the on-line optimal control calculation, which is much easier to handle computationally. The detailed algorithm can be found in the paper. We shall

compare the performances of this algorithm as well as the full nonlinear MPC based on solving the NLP with those of the proposed ADP based methods.

In the following two sections, we develop two different approaches based on the ADP framework to address the aforementioned drawbacks of MPC in using empirical models for control of nonlinear processes.

8.3 Empirical Model Based ADP Approach: J-Learning

The first approach is essentially same as the penalty function based ADP algorithm discussed in Chapter 5 except that an identification based on the input-output data precedes the procedure. Since the same data are used for identification and derivation of an improved control policy, the quantification of the accuracy of a cost-to-go estimate based on the local data density naturally incorporates cautious utilization of a given model into the design of a controller. As discussed in Chapter 5, the suggested ADP approach defines a 'risk' term that is included in the objective function to discourage the controller from entering the regions of the state space for which the confidence is low. This way, the optimizer can be coaxed to use the empirical model only in the regions of the state space with adequate data density, in both the value iteration step and the on-line optimization.

In this work, we employ a distance-weighted k-nearest neighbor approximation scheme of (67) – (69), local density estimator of (102) and (103), and the corresponding penalty function method described in Chapter 5.

With the above, we can attempt to learn an improved control policy while utilizing the identified model cautiously. The procedure is described as follows:

1. Perform closed-loop identification experiments in all possible operating regions by injecting dither signals into the control actions.

2. Identify a NARX model by fitting a parameterized structure (e.g., neural network, polynomial, etc.) to the data.

3. Perform the value iteration with the identified model and the initial cost-to-go data until the cost-to-go values for all the visited states converge.

4. On-line control action is calculated according to (49).

Since the cost-to-go function represented by the local averager is non-smooth in general, manipulated variable u is discretized into a set of values for the global optimization in order to avoid local minima.

8.4 Model-Free Approach: Q-learning

In this section, we propose a different approach to designing a cost-to-go function based controller, which does not involve building an explicit input-output model and allows for a continual improvement of a given control policy through periodic updates based on obtained on-line data. In order to learn a control policy in the absence of an input-output model, the definition of cost-to-go function needs to be changed slightly. Whereas the cost-to-go for the previous approach has the state vector as a sole argument, the cost-to-go for the new approach, which we refer to as Q-function, maps the state and action pair to the future cost-to-go value. The optimal Q-function satisfies

$$Q^*\left(x(k), u(k)\right) = \phi\left(x(k), u(k)\right) + \alpha \min_{u' \in \mathcal{U}} Q^*\left(x(k+1), u'\right) \tag{167}$$

which is equivalent to

$$Q^*\left(x(k), u(k)\right) = \phi\left(x(k), u(k)\right) + \alpha J^*(x(k+1)) \tag{168}$$

where $x(k+1)$ is the successor state of $x(k)$ after applying $u(k)$. Equations (167) and (168) imply that Q value is the sum of all the single-stage costs over the infinite horizon starting with a given state and action pair assuming the optimal control policy is enforced from the next time step on.

As discussed in Chapter 2, this approach was originally proposed within the AI community [132, 133] to solve discrete MDPs where the number of state and action pairs is finite. The conventional way to solve for the optimal Q function is to iterate (167) by performing a large number of on-line experiments to allow for multiple visits to a same state with different action values. General update rule of the Q-value is

$$Q^{i+1}(x(k), u(k)) = (1-\gamma) Q^i(x(k), u(k)) + \gamma \left\{ \phi(x(k), u(k)) + \alpha \min_{u' \in \mathcal{U}} Q^i(x(k+1), u') \right\} \tag{169}$$

where γ is a learning rate parameter between 0 and 1 [132, 133].

However, the conventional Q-learning is not well-suited for process control problems due to the continuous nature of typical state and action spaces. States collected from transient trajectories will tend to be distinct and multiple visits to a same state unlikely even with a large number of experiments. In other words, different control policies and randomization do not ensure multiple visits to exact same states.

Recently, several modified Q-learning methods capable of handling continuous state and action spaces have been proposed. Most of them are based on discretizing state and action spaces and then posing the problem as a discrete MDP. However, the discretization approach is limited to small-size problems and the result can be very sensitive to the discretization method [121]. One noteworthy work for continuous Q-learning was proposed by Smart and Kaelbling [111], who named it the HEDGER algorithm. The main idea is to use existing approximated Q-values for training *neighboring* Q-values and to use a hyper-elliptic hull to prevent extrapolation. The strategy of updating neighboring points based on distance metric is attractive, but the algorithm has some disadvantages in terms of application to process control problems. Not only is the task of building the independent variable hull (IVH) at each decision point computationally very expensive but the conservativeness of IVH can oftentimes result in no control action taken by the controller. Such a learning scheme may be acceptable for robot-learning problems, where 'learning-by-mistakes' is an acceptable practice, but for chemical process control problems, one must be able to provide a reasonable guarantee against catastrophic failures before a controller can be put on-line. Based on this consideration, we propose a modified Q-learning algorithm suited for process control problems. Important features of the modified approach are

- A memory-based local averager is employed for the approximation of Q values.
- In the implementation of the approximated Q function, the same quadratic penalty adjustment as in (104) (but with J replaced by Q) is made to calculate the control action. The distance metric is based on the concatenated vector composed of the state and action vectors.

- To update the Q values in the memory, the exact Q values are calculated from on-line.

The corresponding Q-learning procedure for learning an improved control policy in the absence of a model is as follows:

1. Perform closed-loop experiments in all possible operating regimes by injecting dither signals into the control actions.

2. Estimate initial Q values for each visited state-action pair. Based on these values, build a local averager (e.g. distance-weighted k-nearest neighborhood), of which the input argument is a concatenated vector of state-action pair. A feature weight can be assigned to each component of the vector to make the prediction more accurate.

$$Q^0\left(x(k), u(k)\right) = \sum_{t=k}^{\infty} \alpha^{t-k} \phi(x(t), u(t)) \tag{170}$$

$$\tilde{Q}(z) = \sum_{z_j \in N_k(z)} w_j Q(z_j) \tag{171}$$

where

$$z = [x \quad u]^T \tag{172}$$

,

$$w_j = \frac{1/d_j}{\sum_j 1/d_j} \tag{173}$$

and

$$d_j = \sqrt{(z - z_j)^T W (z - z_j)} \tag{174}$$

3. Perform a closed-loop experiment under the policy based on $\tilde{Q}^i$, which is

$$u(k) = \mu^i(x(k)) = \arg\min_{u \in \mathcal{U}} \tilde{Q}^i(x(k), u) \tag{175}$$

with the penalty term added to $\tilde{Q}^i$ as in the J-learning's case. In the above optimization, input action set $\mathcal{U}$ can be constructed by sampling the neighboring k_u points of the $x(k)$ and also add discretized action sets within the sampled bound for global optimization. This is reasonable in view of the fact that state-action pairs that are far from the available data are unlikely to be chosen due to the penalty term anyhow.

In essence, this results in discretization and global search only in the relevant region of the action space, which can reduce the computation time.

4. Calculate the Q value (Q^{on}) for each visited state and action pair (x_0, u_0) from the on-line experiment according to

$$Q^{on}(x_0, u_0) = \sum_{t=0}^{\infty} \alpha^t \phi \tag{176}$$

5. Update the Q values using the new information Q^{on}

$$Q^{i+1}(x, u) = Q^i(x, u) + \gamma \left\{ Q^{i,on}(x_0, u_0) - Q^i(x, u) \right\} \tag{177}$$

where i is the off-line update index, and the learning rate parameter γ is assigned as follows:

- If (x_0, u_0) is the same point (within some tolerance) with (x, u)

$$\gamma = \frac{1}{\sqrt{i+1}} \tag{178}$$

Otherwise, add the point (x_0, u_0) to the memory.

- In both cases, neighboring points, $(x, u) \in N_{k_u}(x_0, u_0)$, are also updated using the distance-weighted factor.

$$\gamma = \frac{1/d_j}{\sum_{j=1}^{k_u} 1/d_j} \tag{179}$$

where $N_{k_u}(x_0, u_0)$ is the set composed of the k_u-nearest neighbors of (x_0, u_0).

6. Repeat the procedure from step 3 until no appreciable improvement is observed.

8.5 Simulation Example: CSTR

In this section, an example of CSTR with a first-order exothermic reaction adopted from Hernández and Arkun [46] is considered. The proportional-integral (PI) controllers, the successive linearization based MPC (slMPC) method described in [61], nonlinear programming based MPC (NMPC), J-learning-based controller, and Q-learning-based controller are compared to illustrate some key aspects of the suggested approaches.

The dynamic equations of the system is given in dimensionless form by

$$\begin{aligned} \dot{x}_1 &= -x_1 + D_a(1-x_1)\exp\left(\frac{x_2}{1+x_2/\varphi}\right) \\ \dot{x}_2 &= -x_2 + BD_a(1-x_1)\exp\left(\frac{x_2}{1+x_2/\varphi}\right) + \beta(u-x_2) \\ y &= x_1 \end{aligned} \tag{180}$$

where x_1 and x_2 are the dimensionless reactant concentration and reactor temperature, respectively. The input u is the cooling jacket temperature. D_a, φ, B, and β are Damköhler number, dimensionless activation energy, heat of reaction, and heat-transfer coefficients, respectively. With the choice of the following parameters,

$$D_a = 0.072, \quad \varphi = 20.0, \quad B = 8.0, \quad \beta = 0.3 \tag{181}$$

the system shows three steady states, the middle one of which is unstable as shown in Figure 33. The control objective is to take the system from a stable equilibrium point $(x_1 = 0.144, x_2 = 0.886, u = 0.0)$ to an unstable one $(x_1 = 0.445, x_2 = 2.7404, u = 0.0)$.

In the following, we first illustrate how the model-based controllers can overuse an identified model leading to poor control performances, and then show how the overuse can be prevented by the proposed approaches.

8.5.1 Identification

A NARX structure with a feedforward neural network was identified using data from closed-loop simulations under a PI controller. One can determine the structure of a NARX model by using the stepwise model building algorithm discussed in [55]. With the dimensionless sample time of 0.5, the output of next time step $y(k+1)$ was found to be a function of $[y(k), \cdots, y(k-3), u(k), \cdots, u(k-3)]$ [46]. Thus, the state vector x is defined as in (151) with $n_y = 3$ and $n_u = 3$.

To cover pertinent operating ranges, different controller gains were used under the set-point of 0.4450. The selected gain values were 9, 6.75, and 4.5 with the same integral time of 83.3 (sample time). For each closed-loop experiment, the input was dithered at each sample time with a random noise generated from the uniform distribution within [-0.03 0.03]. 12 set of simulations were conducted by changing the set-point from the low steady state to

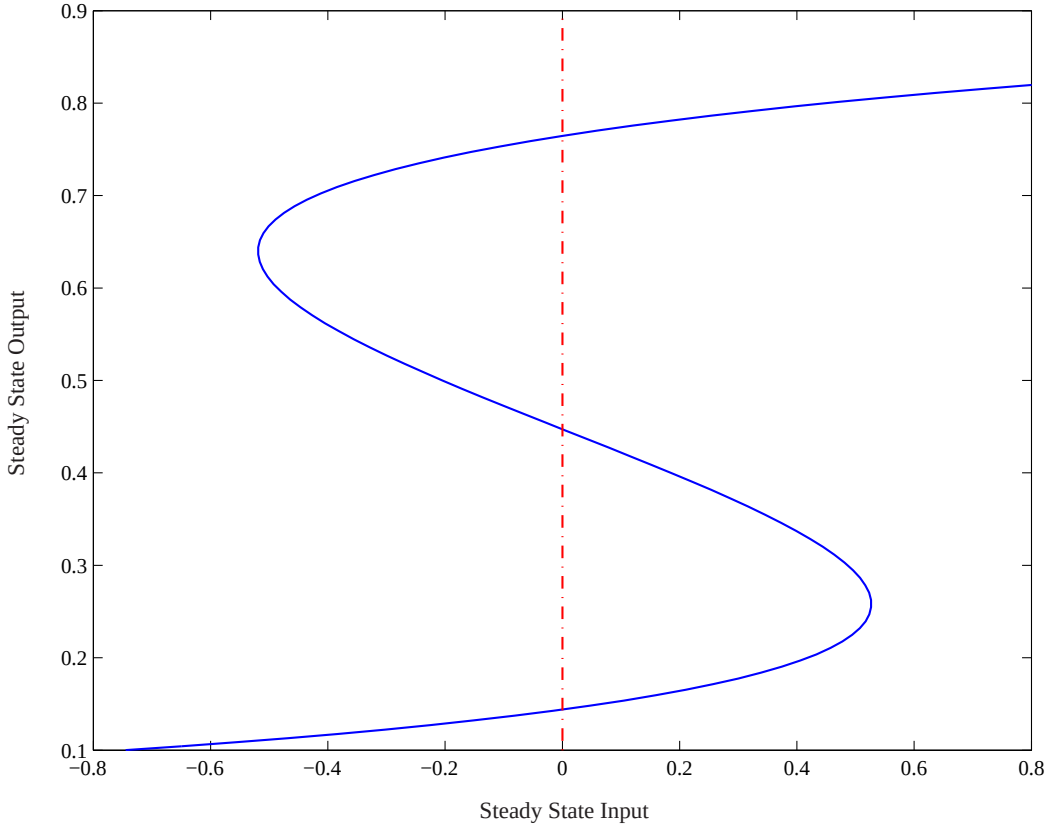

Figure 33: Steady-state output vs. steady-state input for CSTR example.

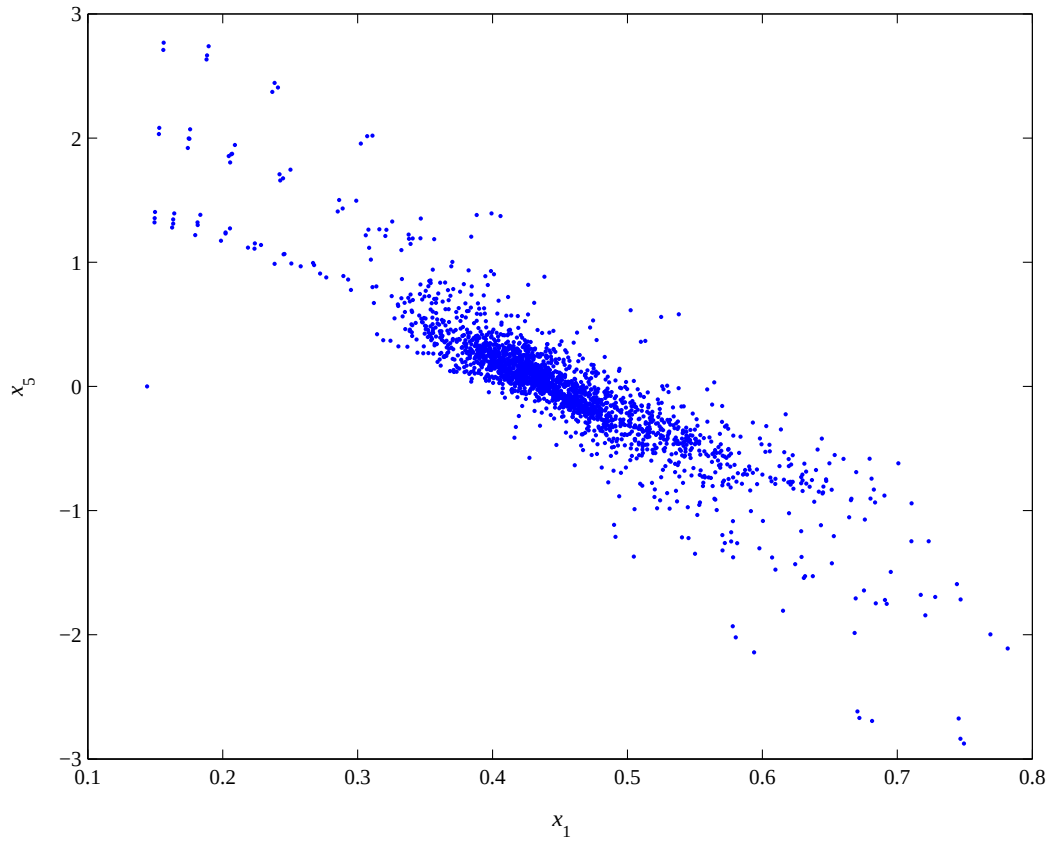

Figure 34: State space plot of the identification data: plot of x_1 vs. x_5.

the unstable one. From the simulations, we collected 2820 input-output data points. The identification data (x_1 vs. x_5) are plotted in Figure 34.

The neural network we fitted has seven hidden nodes with eight inputs, $x(k), u(k)$, and one output, $y(k+1)$. The parameters were identified using the MATLAB Neural Network Toolbox [37] with the fitting tolerance (MSE) set as 1e-5. The resulting neural network gave excellent predictions for the test data lying in the regions of the state space occupied by the identification data as shown in Figure 35.

8.5.2 Model Predictive Control

We tested both slMPC and NMPC coupled with the EKF estimator. With a prediction horizon of 7 and a control horizon of 1, the MPCs solve the following objective function

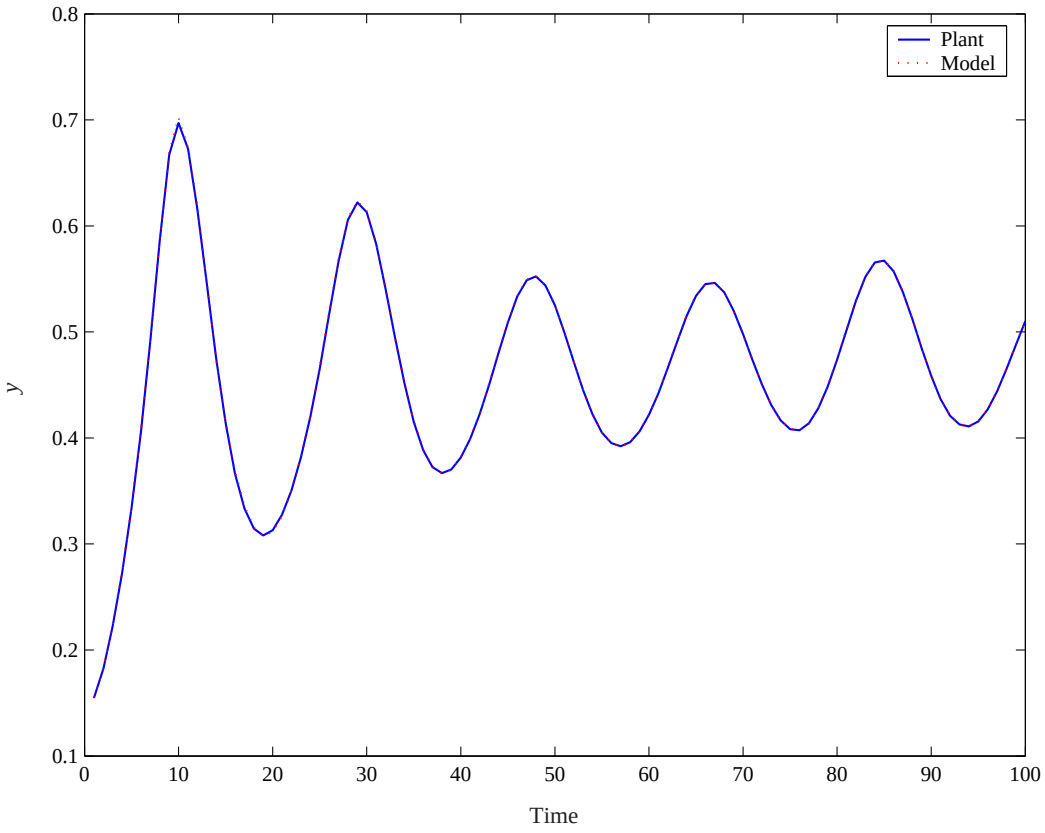

Figure 35: A sample plot of prediction performance of the NARX model for a test data set.

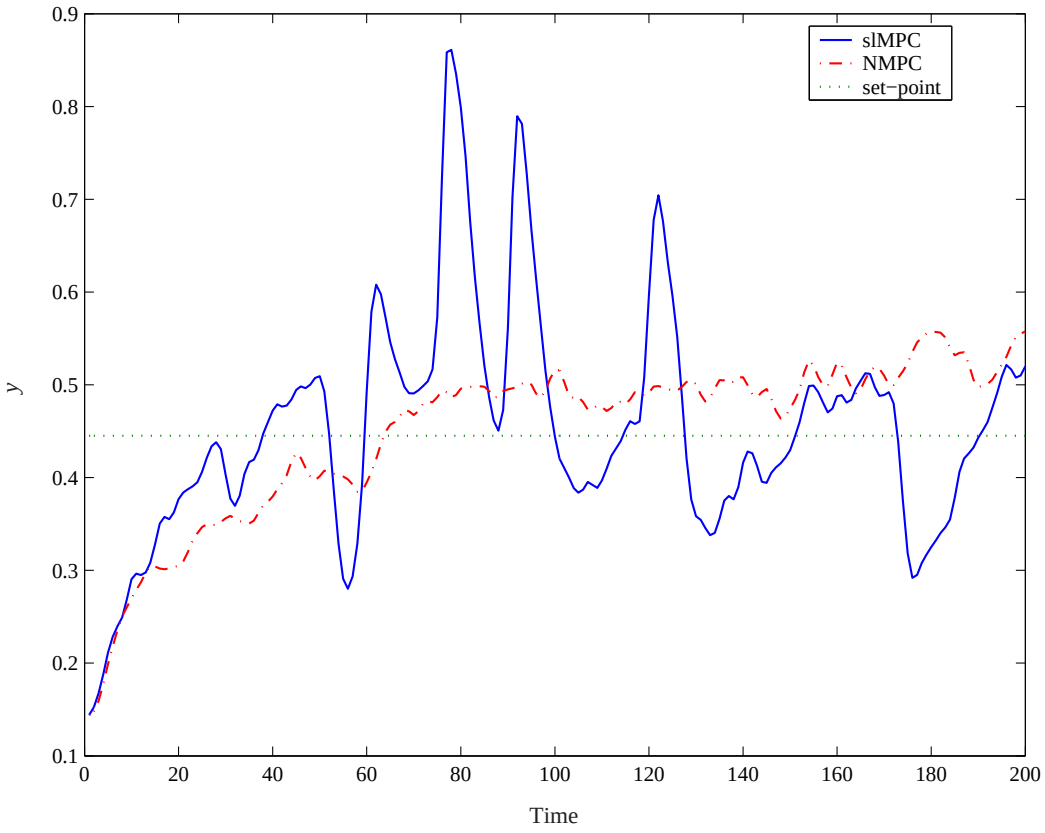

Figure 36: Regulation performances of slMPC and NMPC using the identified model.

on-line at each sample time k:

$$\min_{u(k)} \left[\sum_{t=1}^{7} 50 \left\{ 0.4450 - y(k+t) \right\}^2 + \Delta u(k)^2 \right] \tag{182}$$

The regulation performances are shown in Figure 36. We can see that the system could not be regulated by either of the controllers. Larger control horizon choices were found to yield worse performances, probably due to the optimizer finding local minima. Figure 37 shows that the poor regulation performance is due to the extrapolation of the identified model to unexplored regions of the state space. It is noteworthy that *the output and input weights had to be detuned significantly in order to achieve closed-loop stability.* The ratio of the output weight to the input weight had to be decreased to 5.

8.5.3 J-learning Approach

For the 2820 data, the value iteration was performed with the k-nearest neighbor approximator, which averages four neighboring points ($k = 4$) for the cost-to-go approximation.

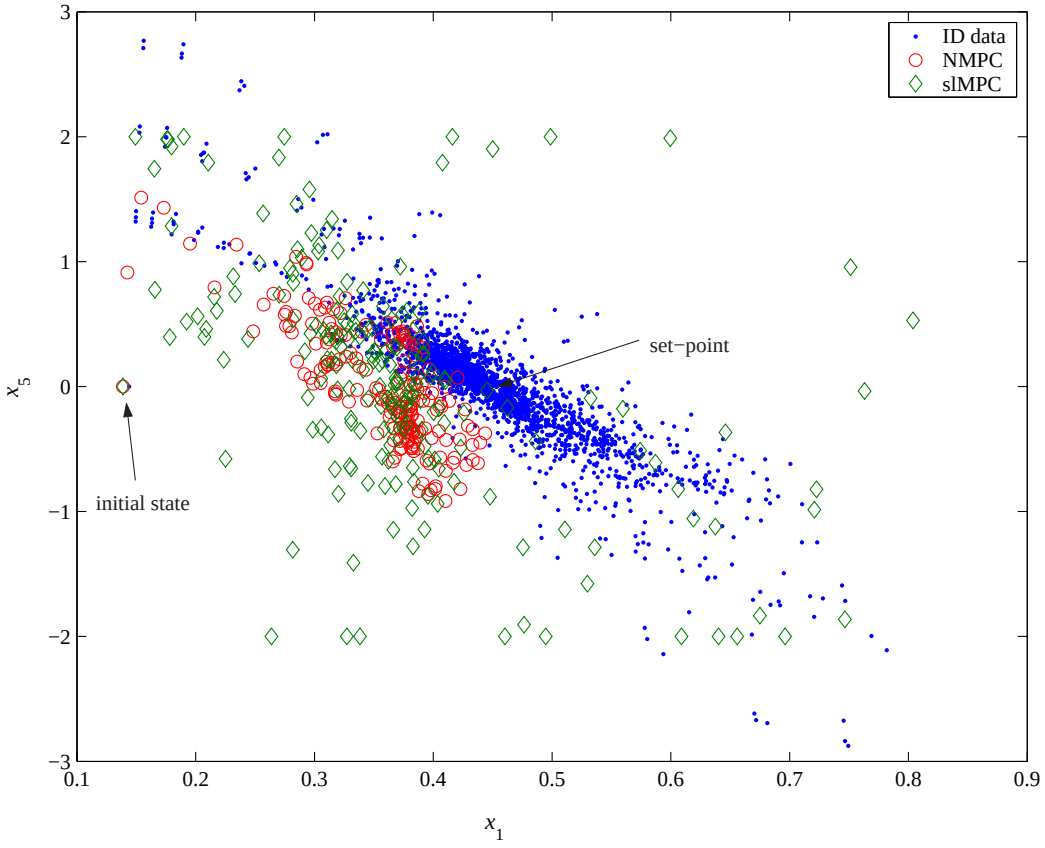

Figure 37: On-line state trajectories of MPCs: plot of x_1 vs. x_5.

Using a discount factor of 0.98, the initial cost-to-go values were calculated as an discounted infinite horizon sum of the following one-stage cost:

$$\phi(x(k), u(k)) = 50(0.4450 - y(k+1))^2 + (u(k) - u(k-1))^2 \tag{183}$$

Hence, with the ADP based method, we are attempting to derive a much less detuned controller that still maintains closed-loop stability. The convergence criterion used for the value iteration was

$$e_{rel} = \left\| \frac{J^i(x_k) - J^{i-1}(x_k)}{J^{i-1}(x_k)} \right\|_\infty < 0.01 \tag{184}$$

where $k = 1, \cdots, 2820$ and i is the iteration index. The off-line value iteration converged after 31 steps, during which e_{rel} decreased monotonically as shown in Figure 38. The parameters of the penalty function were set as $\sigma = 0.1587$ (1% of normalized distance range), $\rho = 6.6 \times 10^{-9}$, $A = 0.0696$, and $J_{max} = 200$. Figure 39 shows the improvement in performance from the starting control policies (PI controllers) and Figure 40 illustrates that the suggested strategy uses the model in the vicinity of the data and avoids unreasonable extrapolations.

8.5.4 Q-learning Approach

Using the same starting PI controllers and input dither signals, 3256 points of state and action pairs were collected. Initial Q values for each state and action pair were estimated by (170) with α of 0.98. For the Q function approximation, the k-nearest neighbor approximator with $k = 4$ was employed and $W = \text{diag}[5\ 3\ 3\ 2\ 3\ 2\ 2\ 5]$ was chosen to assign more weights to the current input and output values. $k_u = 15$ was used for action sampling and update of neighboring Q values based on the updated on-line information. The parameters of penalty function were set as $\sigma = 0.1697$, $\rho = 1.2 \times 10^{-9}$, $A = 0.0696$, and $Q_{max} = 200$.

As mentioned earlier, this interactive learning between on-line experiment and off-line redesign of the Q function (and therefore the controller) in the absence of a model involves the update of new information at each iteration, which implies a gradual increase in the number of data points stored in the memory. Table 16 indicates about 100% increase in the number of data points after 20 iteration steps, which is very reasonable.

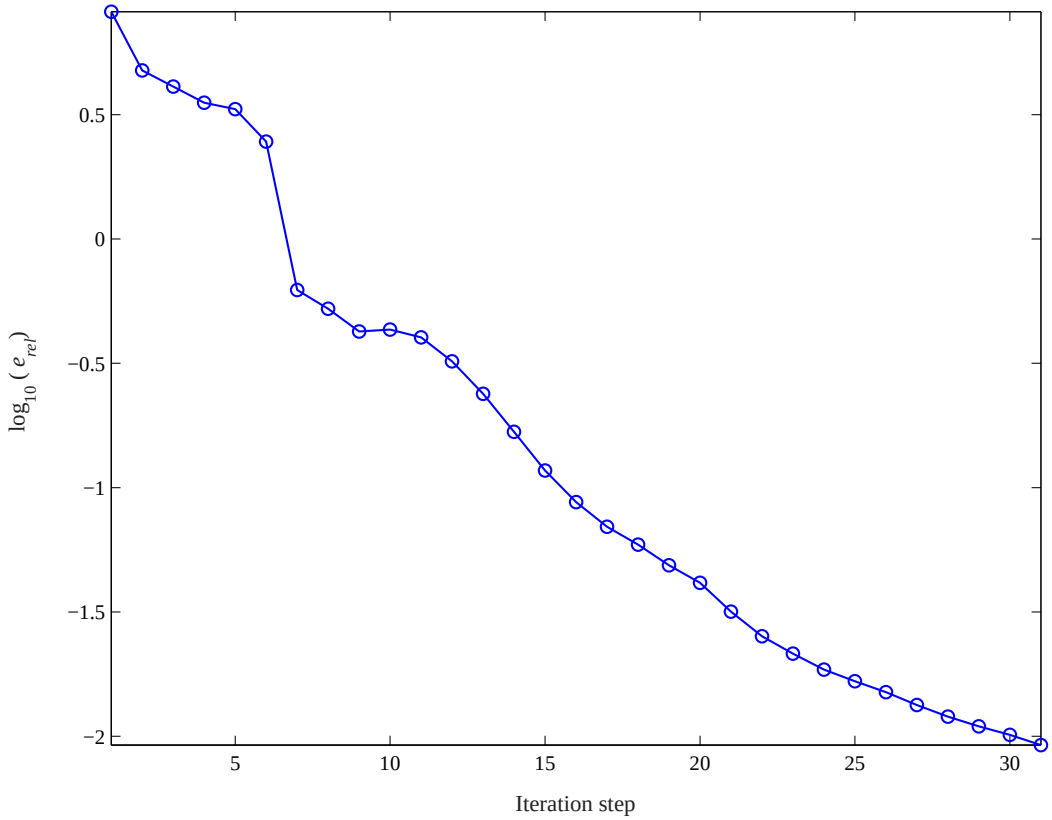

Figure 38: Convergence behavior of the off-line value iteration for J-learning.

Table 16: Number of data points in the memory at each iteration step of Q-learning.

Iteration	Data	Iteration	Data
1	3256	11	4979
2	3442	12	5147
3	3627	13	5242
4	3810	14	5410
5	3990	15	5592
6	4157	16	5756
7	4269	17	5923
8	4451	18	6089
9	4633	19	6250
10	4809	20	6418

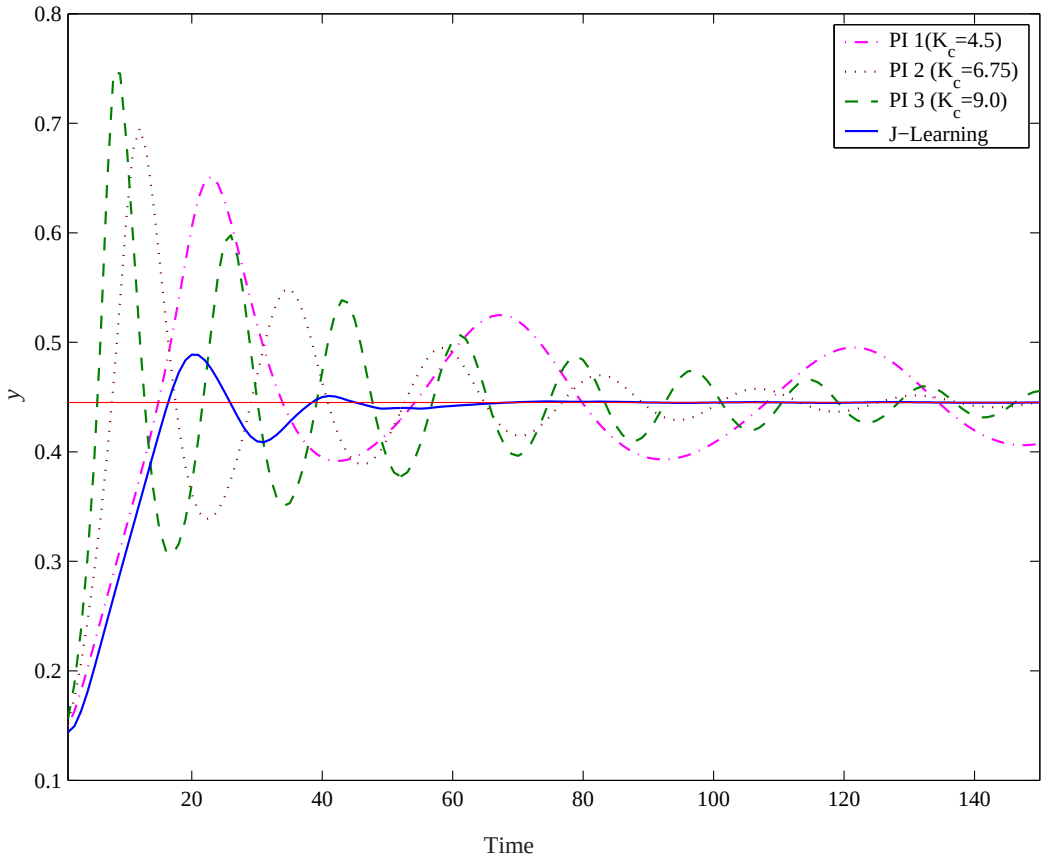

Figure 39: Improved regulation performance of J-learning from starting control policies.

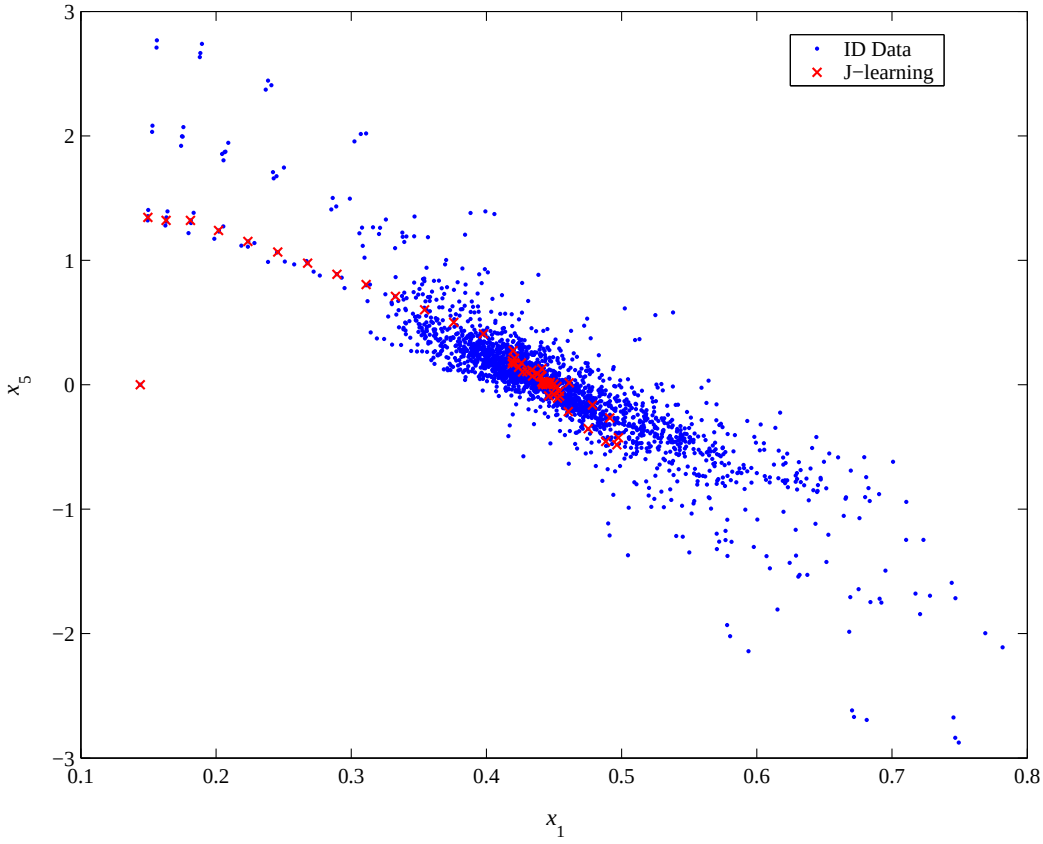

Figure 40: On-line state trajectory of J-learning: plot of x_1 vs. x_5.

Table 17: Improvement of on-line performance under Q-learning.

Iteration	$\sum_{k=0}^{\infty}\phi$	Iteration	$\sum_{k=0}^{\infty}\phi$
1	44.76	11	30.88
2	39.08	12	29.71
3	41.73	13	29.59
4	38.74	14	29.56
5	34.09	15	29.01
6	35.64	16	29.01
7	32.83	17	29.56
8	31.93	18	28.42
9	31.48	19	27.53
10	29.72	20	27.53

Table 18: Comparison of the closed-loop performance of control polices: infinite horizon cost.

Method	$\sum_{k=0}^{\infty}\phi$
PI (Kc=4.5)	60.9
PI (Kc=6.75)	43.8
PI (Kc = 9.0)	58.0
slMPC	∞
NMPC	∞
slMPC (detuned)	165
NMPC (detuned)	42.0
J-learning	27.2
Q-learning	27.5

Figure 41 and Table 17 show oscillatory behavior in the early phase of learning, but after a while the performance improvement is gradual and reaches very good performance after 20 iteration steps. The state space plot of Figure 42 from the on-line simulation using the Q-function obtained from the 20th iteration shows a well-interpolated trajectory to the target point. In Table 17, undiscounted infinite horizon cost is used for comparison of on-line regulation performances:

$$\sum_{k=0}^{\infty}\phi(x(k),u(k)) \tag{185}$$

Table 18 shows that the ADP schemes improved the starting control policies (PI controllers) while avoiding the extrapolation problem seen in the MPCs.

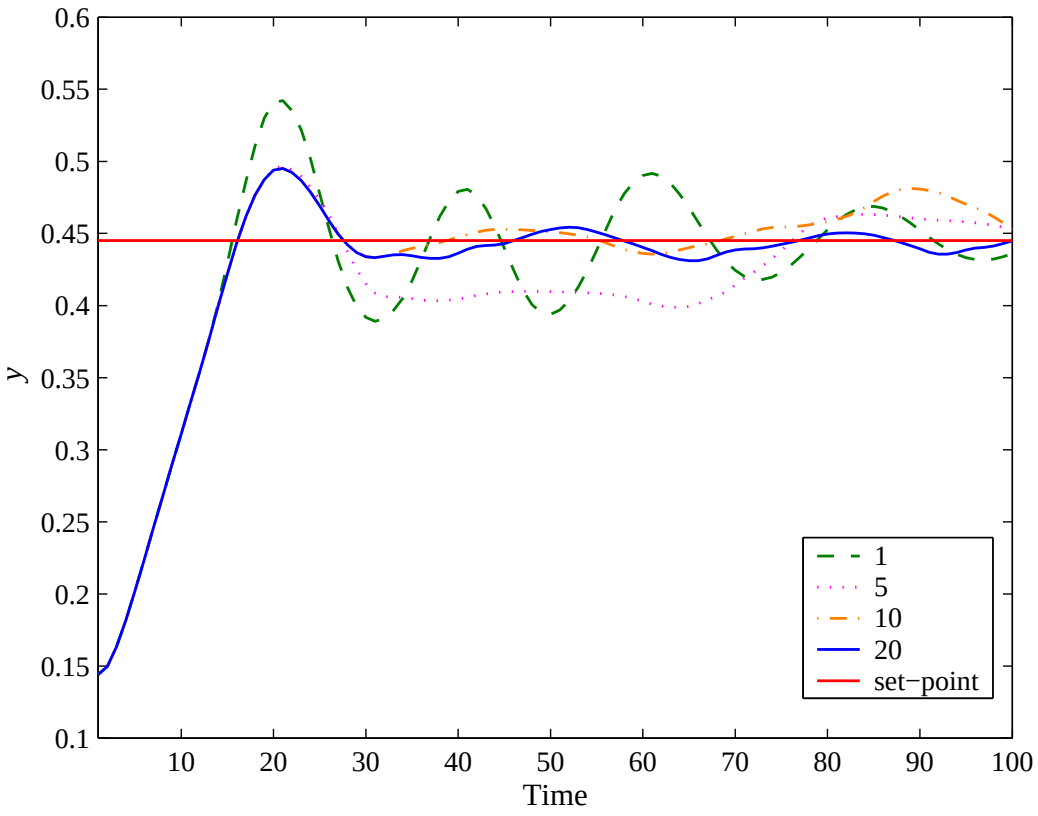

Figure 41: Evolutionary improvement of regulation performance under Q-learning.

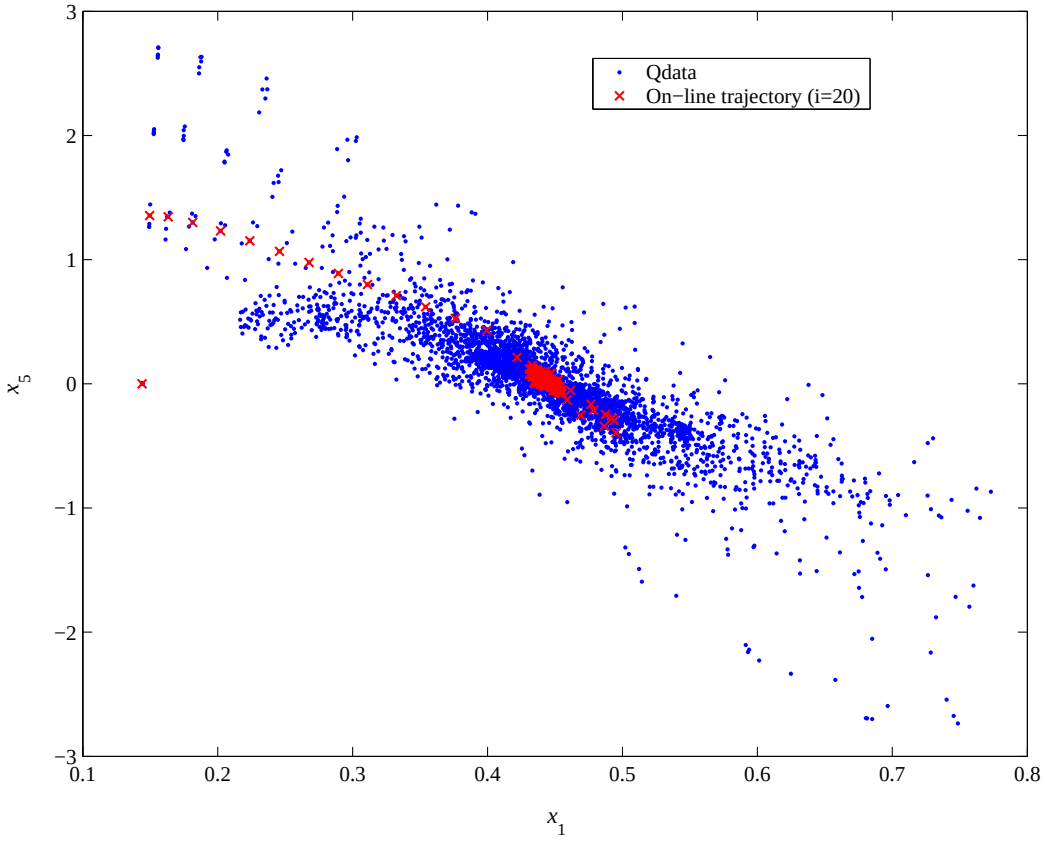

Figure 42: On-line state trajectory of Q-learning: plot of x_1 vs. x_5.

8.6 Conclusions

In this chapter, we presented two input-output-data-based nonlinear control approaches based on the ADP framework, which iteratively learn the cost-to-go function from data collected through identification experiments. The first approach, which we named 'J-learning,' builds an empirical input-output model and then derives the cost-to-go function off-line for use in on-line control. The second approach, named 'Q-learning,' does not build any input-output model but alternates between on-line experiment and approximation of the so called Q function. In both approaches, we discourage the optimizer from finding a solution that pushes the system into the regions of the state space with sparse data density, by adjusting the learned cost-to-go function with a penalty term based on the local data density. Compared to the conventional model-based approaches like MPC, the suggested approaches have the advantage of being able to incorporate the knowledge of identification data distribution (and therefore model's confidence) into the control calculation more easily, thereby yielding a more robust control policy.

CHAPTER IX

CONTRIBUTIONS AND FUTURE WORK

9.1 Contributions

This thesis work was motivated by the necessity of a novel control framework, as chemical processes in practice diversify into complex nonlinear/uncertain systems or even more challenging processes, the dynamics of which can only be described by molecular simulation procedures. The current model-based and on-line optimization control methodologies can only provide limited performance or they cannot be incorporated into such systems. Approximate dynamic programming (ADP) strategies developed in the fields of artificial intelligence (AI) and machine learning (ML) offer attractive advantages in view of the limitations of MPC and conventional DP approaches. However, they are mostly tailored to suit the characteristics of applications in operations research, computer science, and robotics. In this thesis, we have: proposed ADP strategies suitable for chemical process control, addressed issues on the proper choice of a function approximator for cost-to-go and the reliable use of the approximated information, and tackled difficult process control problems using the ADP approaches.

In Chapter 3, we have presented an ADP framework for chemical process control. Process control problems have continuous variables and high-dimensional state space. On-line trial-and-error learning is risky and should be avoided. Because of these factors, some popular algorithms found in the Reinforcement Learning (RL) and Neuro-Dynamic Programming (NDP) literature are not applicable to process control problems. The suggested ADP strategy identifies important regions of the state space using closed-loop simulation data. The starting control policies are judiciously chosen so that satisfactory controls can be found within the limited envelope. The main premise is that only a small fraction of the huge state space would be relevant for optimal or near-optimal control in practice. Then the starting control policies are improved by iterating on the Bellman's optimality equation

off-line using function approximation. Though the working state space may not contain the optimal path, this step brings significant improvement to the initial control policies. For example, one may have several different control policies working satisfactorily only for different regions of the state space. The principle of optimality can find an improved trajectory by interpolating the state points visited by the simulations under the initial policies. Furthermore, on-line implementation of an ADP control policy is relatively simple to solve on-line, compared with the MPC formulation.

To deal with the continuous variables, a function approximator is necessary in the framework. Since the optimization and function approximation are interlaced and recursively used at every iteration step, a proper choice and design of the approximator was found to be critical for the suggested framework. Though some former papers pointed out the issues, there have been mixed results because of the different natures of the problem. Hence, in Chapter 4, we have examined the suitability of different sorts of approximators for the suggested ADP framework using typical process control examples. They include difficult case studies, such as a state-constrained problem and a high-dimensional nonlinear process. We compared a global parameterized approximator (artificial neural network) with a local averaging instance-based approximator (k-nearest neighbor). We could conclude that a local averager shows better off-line learning behavior and on-line performance, but undue extrapolation is problematic for the high-dimensional case, due to the limited explorations in the state space.

In Chapter 5, we have proposed a penalty function method for ensuring a reliable use of cost-to-go approximation that is valid over limited regions of state space. In the RL/NDP literature there have been few studies dealing with the undue extrapolation issue because they assume either huge amounts of data or trial-and-error learning. The proposed method evaluates local data density around a query point to indicate the 'risk' of cost-to-go estimation. The risk is formulated as a quadratic penalty term on the cost-to-go estimate. The penalty function systematically biases the search of a control action to those regions of adequate data density, thereby providing the most dependable control action.

From Chapters 6 to 8, we have tackled difficult process control problems using the

ADP approach. In Chapter 6, we have solved a dual adaptive control problem where the estimation quality and the control performance are coupled. The optimal controller is known to have dual features, which are regulating actions and active probing actions. Only ad hoc design methods have been suggested to implement the dual features because DP approaches have been considered infeasible. The ADP approach could derive a dual control policy with less computational burden. Stochastic nature was handled using Monte Carlo simulations.

Chapter 7 has shown how the cost-to-go based controller can be used for monitoring and overriding a local linear controller. The proposed scheme switches between an override nonlinear controller and a local linear controller based on the dynamic state of a system. The ease of design and implementation is attractive for industrial applications. Not only can the dual-mode controller improve the performance and stability of a given local controller, but it can also give operators indications on the future performance of the local controller.

In Chapter 8, we have considered a situation where a first-principle model is unavailable. A typical nonlinear MPC approach builds an input-output model and uses it for control calculations, which is considered difficult because the validity of a given model cannot be accounted for conveniently within the predictive control scheme. The ADP framework naturally handles this issue in that the local approximation scheme with the penalty function derives an improved control policy while restricting the search space for optimal control moves to those covered by the identification data. In addition, we have proposed a model-free learning (Q-learning) scheme for continuous state variables. This does not require a model identification step, but improves a control policy by continually updating the cost-to-go function from operation to operation.

9.2 Future Work

There are several directions for further work based on the suggested framework in this thesis. They include:

- **Theoretical guarantees on the use of cost-to-go**

  Though the penalty function method with a local averaging scheme has been shown

to work successfully for a number of practical systems, function approximation can be carried out in a more systematic way if an error bound can be derived for a general class of problems. This analysis is yet to be reported for continuous problems.

- **Systematic exploration**

  Exploration of the state space may give us significant performance improvements. One possible way is to use outer policy iteration, which alternates between the on-line implementation of converged cost-to-go information and the off-line redesign of the cost-to-go much like in the suggested Q-learning scheme in Chapter 8. In this step, one can also inject some input dither signals to yield a randomized control policy. Systematic way of expanding the learning domain in an evolutionary manner is an open question.

- **Sampling and averaging for stochastic systems**

  For a stochastic control problem introduced in Chapter 6, we found that some outliers could occur during the sampling in evaluating the expectation operator and could bias the average cost-to-go significantly. This can be avoided if there are huge amounts of simulation data, but smart handling of these outliers associated with the penalty term can help reduce the computation in the learning step. A proper strategy for sampling and realizations for stochastic systems is an open issue in this regard.

  Evaluation of the expectation can be more systematic and simplified, if we use quadrature approximation given a probability distribution instead of using sample average from Monte Carlo simulation. However, it is still unclear how to conduct simulations in this case.

- **Large action space**

  The suggested algorithms can suffer from the same curse-of-dimensionality as the dimension of action space increases. This is because the action space should be searched over for finding a greedy control action. One approach we employed is to confine the action space by sampling the action values from the suboptimal laws and optimize

over it. More rigorous studies along this line, with an efficient optimization algorithm, would also be beneficial.

- **Parallelization of the off-line iteration**

  Because the value iteration sequentially updates a cost-to-go for each state, the iteration step may require significant amount of computing time when it is applied to large-scale problems. Intelligent allocation of computing time by parallelizing the value iteration procedure can facilitate management of the computational and storage requirements. A proper strategy for partitioning the data set should be explored in this context.

- **Extension to challenging processes**

  Emerging engineering applications deal with more complex systems with different length and time scales. For such a process, it is difficult to apply existing control techniques to calculate a proper control policy. For example, material processing and biological systems are next to impossible to describe using macroscopic conservation equations. For such processes, molecular simulation tools are mainly developed for design and optimization [43]. One of the main advantages of the ADP framework is that it is not limited by specific model forms or processes in principle. Given a simulator and a control objective, one can generate the data and bring evolutionary improvements on given control policies. However, proper representation of the cost-to-go would be important in this application.

  Large scale and distributed systems are common in practice, for which differential algebraic equations (DAEs) are often used. It is more difficult to derive a proper control strategy for DAE systems than the systems with ordinary differential equations because of the issues like high index and consistent initial conditions [58]. Proper interpretation and application of the ADP strategy to the DAE systems is a fruitful avenue to explore.

  Proper integration of relevant techniques into the ADP strategy would also be beneficial for further extension to the large-scale systems. For example, use of data-based

model reduction technique (e.g. POD) to obtain a more compact representation of the state for the cost-to-go information storage will be particularly important in plant-wide centralized control.

- **Applicability of policy space algorithms**

  The ADP framework is concerned with approximating the cost-to-go function aimed at solving the Bellman equation directly. Then the learned cost-to-go function is used to prescribe a near-optimal policy. A new approach recently advocated is to approximate and optimize directly over the policy space, which is called *policy-gradient method* [54, 113, 73]. The method was motivated by the disadvantage of the cost-to-go function based approach that can result in very different actions for "close" states from the greedy policy in the presence of approximation errors. This can be avoided by controlling the smoothness of the policy directly. However, the algorithms still need significant investigation to be recast as an applicable framework for process control problems.

REFERENCES

[1] Ahamed, T. P. I., Rao, P. S. N., and Sastry, P. S., "A reinforcement learning approach to automatic generation control," *Electric Power Systems Research*, vol. 63, no. 1, pp. 9–26, 2002.

[2] Ahn, S.-M., Park, M.-J., and Rhee, H.-K., "Extended Kalman filter-based nonlinear model predictive control for a continuous MMA polymerization reactor," *Industrial & Engineering Chemistry Research*, vol. 38, pp. 3942–3949, 1999.

[3] Akaike, H., "A new look at the statistical model identification," *IEEE Trans. Auto. Cont.*, vol. 19, no. 6, pp. 716–723, 1974.

[4] Anderson, B. D. O., "Adaptive systems, lack of persistency of excitation and bursting phenomena," *Automatica*, vol. 21, no. 3, pp. 247–258, 1985.

[5] Anderson, C. W., "Learning to control an inverted pendulum using neural networks," *IEEE Control Systems Magazine*, vol. 9, no. 3, pp. 31–37, 1989.

[6] Anderson, C. W., Hittle, D. C., Katz, A. D., and Kretchmar, R. M., "Synthesis of reinforcement learning, neural networks and PI control applied to a simulated heating coil," *Artificial Intelligence in Engineering*, vol. 11, pp. 421–429, 1997.

[7] Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K., "Purposive behavior acquisition for a real robot by vision-based reinforcement learning," *Machine Learning*, vol. 23, pp. 279–303, 1996.

[8] Åström, K. J. and Helmersson, A., "Dual control of an integrator with unknown gain," *Comp. & Maths. with Appls.*, vol. 12A, pp. 653–662, 1986.

[9] Åström, K. J. and Wittenmark, B., "Problems of identification and control," *Journal of Mathematical Analysis and Applications*, vol. 34, pp. 90–113, 1971.

[10] Atkeson, C. G. and Santamaria, J., "A comparison of direct and model-based reinforcement learning," in *Proceedings of the International Conference on Robotics and Automation*, 1997.

[11] Atkeson, C. G. and Schaal, S., "Robot learning from demonstration," in *Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*, (San Francisco, CA), pp. 12–20, Morgan Kaufmann, 1997.

[12] Baird III, L., "Residual algorithms: Reinforcement learning with function approximation," in *Proceedings of the International Conference on Machine Learning*, pp. 30–37, 1995.

[13] Barto, A. G., Bradtke, S. J., and Singh, S. P., "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1, pp. 81–138, 1995.

[14] Barto, A. G., Sutton, R. S., and Anderson, C. W., "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 834–846, 1983.

[15] Bellman, R. E., *Dynamic Programming*. New Jersey: Princeton University Press, 1957.

[16] Bemporad, A., Borrelli, F., and Morari, M., "Min-max control of constrained uncertain discrete-time linear systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 9, pp. 1600–1606, 2003.

[17] Bemporad, A. and Morari, M., "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[18] Bequette, B. W., *Process Dynamics: Modeling, Analysis, and Simulation*. Uppper Saddle River, New Jersey: Prentice Hall, 1998.

[19] Bequette, W. B., "Nonlinear control of chemical processes. A review," *Industrial & Engineering Chemistry Research*, vol. 30, no. 7, pp. 1391–1413, 1991.

[20] Bertsekas, D. P., "Neuro-dynamic programming: An overview," in *Proceedings of Sixth International Conference on Chemical Process Control* (Rawlings, J. B., Ogunnaike, B. A., and Eaton, J. W., eds.), 2001.

[21] Bertsekas, D. P. and Castañon, D. A., "Adaptive aggregation for infinite horizon dynamic programming," *IEEE Transactions on Automatic Control*, vol. 34, no. 6, pp. 589–598, 1989.

[22] Bertsekas, D. P. and Gallager, R. G., *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 2nd ed., 1992.

[23] Bertsekas, D. P. and Tsitsiklis, J. N., *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[24] Bertsekas, D. P., *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 2nd ed., 2000.

[25] Bertsekas, D. P. and Tsitsiklis, J. N., *Neuro-Dynamic Programming*. Belmont, Massachusetts: Athena Scientific, 1996.

[26] Borkar, V., "A convex analytic approach to Markov decision processes," *Probability Theory and Related Fields*, vol. 78, pp. 583–602, 1988.

[27] Boyan, J. A. and Moore, A. W., "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems* (Tesauro, G. and Touretzky, D., eds.), vol. 7, Morgan Kaufmann, 1995.

[28] Bradtke, S. J., "Reinforcement learning applied to linear quadratic regulation," in *Advances in Neural Information Processing Systems* (Hanson, S. J., Cowan, J., and Giles, C. L., eds.), vol. 5, Morgan Kaufmann, 1993.

[29] Cacoullos, T., "Estimation of a multivariate density," *Ann. Inst. Statist. Math. (Tokyo)*, vol. 18, no. 2, pp. 179–189, 1966.

[30] Chen, H. and Allgöwer, F., "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability," *Automatica*, vol. 34, no. 10, pp. 1205–1217, 1998.

[31] Chikkula, Y. and Lee, J. H., "Robust adaptive predictive control of nonlinear processes using nonlinear moving average system models," *Industrial & Engineering Chemistry Research*, vol. 39, pp. 2010–2023, 2000.

[32] Crites, R. and Barto, A. G., "Improving elevator performance using reinforcement learning," in *Advances in Neural Information Processing Systems* (Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., eds.), vol. 8, San Francisco, CA: MIT Press, 1996.

[33] Crites, R. and Barto, A. G., "Elevator group control using multiple reinforcement learning agents," *Machine Learning*, vol. 33, pp. 235–262, 1998.

[34] Cutler, C. R. and Ramaker, B. L., "Dynamic matrix control – a computer control algorithm," in *Proceedings of the Joint Automatic Control Conference*, 1980.

[35] Dayan, P., "The convergence of TD(λ) for general λ," *Machine Learning*, vol. 8, pp. 341–362, 1992.

[36] De Farias, D. P. and Van Roy, B., "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850–865, 2003.

[37] Demuth, H. and Beale, M., *Neural Network Toolbox User's Guide (MATLAB)*. Natick, MA: The MathWorks, Inc., 2002.

[38] Denardo, E. V., "On linear programming in a Markov decision problem," *Management Science*, vol. 16, pp. 282–288, 1970.

[39] Fel'dbaum, A. A., "Dual control theory. I," *Automation Remote Control*, vol. 21, pp. 874–880, 1960.

[40] Fel'dbaum, A. A., "Dual control theory. II," *Automation Remote Control*, vol. 21, pp. 1453–1464, 1960.

[41] Fel'dbaum, A. A., "Dual control theory. III," *Automation Remote Control*, vol. 22, pp. 1–12, 1961.

[42] Fel'dbaum, A. A., "Dual control theory. IV," *Automation Remote Control*, vol. 22, pp. 109–121, 1961.

[43] Gear, C. W., Kevrekidis, I. G., and Theodoropoulos, C., "'Coarse' integration/bifurcation analysis via microscopic simulators: micro-Galerkin methods," *Computers & Chemical Engineering*, vol. 26, pp. 941–963, 2002.

[44] Gordon, G. J., "Stable function approximation in dynamic programming," Tech. Rep. CMU-CS-95-103, School of Computer Science, Carnegie Mellon University, 1995.

[45] Henson, M. A., "Nonlinear model predictive control: Current status and future directions," *Computers & Chemical Engineering*, vol. 23, no. 2, pp. 187–202, 1998.

[46] Hernández, E. and Arkun, Y., "Control of nonlinear systems using polynomial ARMA models," *AIChE Journal*, vol. 39, no. 3, pp. 446 – 460, 1993.

[47] Hordijk, A. and Kallenberg, L. C. M., "Linear programming and Markov decision chains," *Management Science*, vol. 25, pp. 352–362, 1979.

[48] Hoskins, J. C. and Himmelblau, D. M., "Process control via artificial neural networks and reinforcement learning," *Computers & Chemical Engineering*, vol. 16, no. 4, pp. 241–251, 1992.

[49] Howard, R. A., *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[50] Jaakkola, T., Jordan, M. I., and Singh, S. P., "On the convergence of stochastic iterative dynamic programming algorithms," *Neural Computation*, vol. 6, no. 6, pp. 1185–1201, 1994.

[51] Kaelbling, L. P., Littman, M. L., and Moore, A. W., "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[52] Kaisare, N. S., Lee, J. M., and Lee, J. H., "Simulation based strategy for nonlinear optimal control: Application to a microbial cell reactor," *International Journal of Robust and Nonlinear Control*, vol. 13, no. 3–4, pp. 347–363, 2002.

[53] Koenig, S. and Simmons, R. G., "Complexity analysis of real-time reinforcement learning," in *Proceedings of the Eleventh National Conference on Artificial Intelligence*, (Menlo Park, CA), pp. 99–105, AAI Press/MIT Press, 1993.

[54] Konda, V. R. and Tsitsiklis, J. N., "Actor-critic algorithms," in *Advances in neural information processing systems* (Solla, S. A., Leen, T. K., and Müller, K.-R., eds.), vol. 12, Cambridge, MA: The MIT Press, 2000.

[55] Kortmann, M. and Unbehauen, H., "Structure detection in the identification of nonlinear systems," *Autom. Prod. Infor. Ind.*, vol. 22, no. 5, pp. 5–25, 1988.

[56] Kothare, M. V., Balakrishnan, V., and Morari, M., "Robust constrained model predictive control using linear matrix inequalities," *Automatica*, vol. 32, no. 10, pp. 1361–1379, 1996.

[57] Kröse, B. J. A. and van Dam, J. W. M., "Adaptive state space quantisation for reinforcement learning of collision-free navigation," in *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Piscataway, NJ), 1992.

[58] Kumar, A. and Daoutidis, P., *Control of Nonlinear Differential Algebraic Equation Systems*. Chapman & Hall/CRC: Boca Raton, 1999.

[59] Kumar, P. R. and Varaiya, P. P., *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Englewood Cliffs, NJ: Prentice Hall, 1986.

[60] Lee, J. H. and Cooley, B., "Recent advances in model predictive control and other related areas," in *Chemical Process Control – V* (Kantor, J. C., Garcia, C. E., and Carnahan, B., eds.), pp. 201–216, American Insititute of Chemical Engineers, 1997.

[61] Lee, J. H. and Ricker, N. L., "Extended Kalman filter based nonlinear model predictive control," *Ind. Eng. Chem. Res.*, vol. 33, no. 6, pp. 1530–1541, 1994.

[62] Lee, J. H. and Yu, Z., "Worst-case formulation of model predictive control for systems with bounded parameters," *Automatica*, vol. 29, pp. 911–928, 1993.

[63] Lee, J. M. and Lee, J. H., "Neuro-dynamic programming approach to dual control problem," in *AIChE Annual Meeting*, (Reno, NV), 2001. paper 276e.

[64] Lee, J. M. and Lee, J. H., "Q-learning approach for identification and control of nonlinear processes," in *AIChE Annual Meeting*, (San Francisco, CA), 2003. paper 436c.

[65] Leonard, J. A., Kramer, M. A., and Ungar, L. H., "A neural network architecture that computes its own reliability," *Computers & Chemical Engineering*, vol. 16, pp. 819–835, 1992.

[66] Lin, J.-S. and Jang, S.-S., "Nonlinear dynamic artificial neural network modeling using an information theory based experimental design approach," *Ind. Eng. Chem. Res.*, vol. 37, pp. 3640–3651, 1998.

[67] Lin, L.-J., "Self-improving reactive agents based on reinforcement learning, plannin and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.

[68] Lindoff, B., Holst, J., and Wittenmark, B., "Analysis of approximations of dual control," *International Journal of Adaptive Control and Signal Processing*, vol. 13, pp. 593–620, 1999.

[69] Lobo, M. S. and Boyd, S., "Policies for simultaneous estimation and optimization," in *Proceedings of the American Control Conference*, vol. 2, pp. 958–964, 1999.

[70] Mahadevan, S. and Connell, J., "Automatic programming of behavior-based robots using reinforcement learning," *Machine Learning*, vol. 55, no. 2–3, pp. 311–365, 1992.

[71] Mahadevan, S., Marchalleck, N., Das, T. K., and Gosavi, A., "Self-improving factory simulation using continuous-time average-reward reinforcement learning," in *Proceedings of 14th International Conference on Machine Learning*, pp. 202–210, Morgan Kaufmann, 1997.

[72] Manne, A. S., "Linear programming and sequential decisions," *Management Science*, vol. 6, no. 3, pp. 259–267, 1960.

[73] Marbach, P. and Tsitsiklis, J. N., "Simulation-based optimization of Markov reward processes," *IEEE Transactions on Automatic Control*, vol. 46, no. 2, pp. 191–209, 2001.

[74] Martinez, E. C., "Batch process modeling for optimization using reinforcement learning," *Computers & Chemical Engineering*, vol. 24, pp. 1187–1193, 2000.

[75] Mayne, D. Q. and Michalska, H., "Receding horizon control of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 35, no. 7, pp. 814–824, 1990.

[76] Mayne, D. Q., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. M., "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, pp. 789–814, 2000.

[77] Meadows, E. S. and Rawlings, J. B., "Model predictive control," in *Nonlinear Process Control* (Henson, M. A. and Seborg, D. E., eds.), pp. 233–310, New Jersey: Prentice Hall, 1997.

[78] Michalska, H. and Mayne, D. Q., "Robust receding horizon control of constrained nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 38, no. 11, pp. 1623–1633, 1993.

[79] Miller, S. and Williams, R. J., "Temporal difference learning: A chemical process control application," in *Applications of Artificial Neural Networks* (Murray, A. F., ed.), Norwell, MA: Kluwer, 1995.

[80] Moore, A. and Atkeson, C., "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces," *Machine Learning*, vol. 21, no. 3, pp. 199–233, 1995.

[81] Moore, A. W. and Atkeson, C. G., "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103–130, 1993.

[82] Morari, M. and Lee, J. H., "Model predictive control: Past, present and future," *Computers & Chemical Engineering*, vol. 23, pp. 667–682, 1999.

[83] Morari, M. and Ricker, N. L., *Model Predictive Control Toolbox User's Guide (MATLAB)*. Natick, MA: The MathWorks, Inc., 1995.

[84] Munos, R., "A convergent reinforcement learning algorithm in the continuous case based on a finite difference method," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.

[85] Munos, R., "A study of reinforcement learning in the continuous case by means of viscosity solutions," *Machine Learning Journal*, vol. 40, pp. 265–299, 2000.

[86] Neuneier, R., "Enhancing Q-learning for optimal asset allocation," in *Advances in Neural Information Processing Systems* (Jordan, M., Kearns, M., and Solla, S., eds.), vol. 10, MIT Press, 1997.

[87] Nocedal, J. and Wright, S. J., *Numerical Optimization*. New York: Springer-Verlag, 1999.

[88] Ormoneit, D. and Glynn, P. W., "Kernel-based reinforcement learning in average-cost problems," *IEEE Transactions on Automatic Control*, vol. 47, no. 10, pp. 1624–1636, 2002.

[89] Ormoneit, D. and Sen, S., "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, pp. 161–178, 2002.

[90] Parzen, E., "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, pp. 1065–1076, 1962.

[91] Peng, J., *Efficient Dynamic Programming-Based Learning for Control*. PhD thesis, Northeastern University, Boston, MA, 1993.

[92] Peng, J. and Williams, R. J., "Efficient learning and planning within the Dyna framework," *Adaptive Behavior*, vol. 1, no. 4, pp. 437–454, 1993.

[93] Prokhorov, D. V. and Wunsch II, D. C., "Adaptive critic designs," *IEEE Transactions on Neural Networks*, vol. 8, pp. 997–1007, September 1997.

[94] Puterman, M. L., *Markov Decision Processes*. New York, NY: Wiley, 1994.

[95] Qin, S. J. and Badgwell, T. A., "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003.

[96] Rhodes, C. and Morari, M., "Determining the model order of nonlinear input/output systems," *AICHE Journal*, vol. 44, no. 1, pp. 151–163, 1998.

[97] Richalet, J., Rault, A., Testud, J. L., and Papon, J., "Model predictive heuristic control: Applications to industrial processes," *Automatica*, vol. 14, pp. 413–428, 1978.

[98] Rüde, U., *Mathematical and Computational Techniques for Multilevel Adaptive Methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1993.

[99] Rummery, G. A. and Niranjan, M., "On-line Q-learning using connectionist systems," Tech. Rep. CUED/F-INFENG/TR 166, Engineering Department, Cambridge University, 1994.

[100] Sabes, P., "Approximating Q-values with basis function representations," in *Proceedings of the Fourth Connectionist Models Summer School*, (Hillsdale, NJ), Lawrence Erlbaum, 1993.

[101] Samuel, A. L., "Some studies in machine learning using the game of checkers," *IBM J. Res. Develop.*, pp. 210–229, 1959.

[102] Samuel, A. L., "Some studies in machine learning using the game of checkers II – recent progress," *IBM J. Res. Develop.*, pp. 601–617, 1967.

[103] Schaal, S., "Learning from demonstration," in *Advances in Neural Information Processing Systems* (Mozer, M. C., Jordan, M., and Petsche, T., eds.), vol. 9, pp. 1040–1046, MIT Press, 1997.

[104] Schaal, S. and Atkeson, C., "Robot juggling: An implementation of memory-based learning," *IEEE Control Systems*, vol. 14, no. 1, pp. 57–71, 1994.

[105] Schraudolph, N. N., Dayan, P., and Sejnowski, T. J., "Temporal difference learning of position evaluation in the game of Go," in *Advances in Neural Information Processing Systems* (Cowan, J. D., Tesauro, G., and Alspector, J., eds.), vol. 6, pp. 817–824, San Mateo, CA: Morgan Kaufmann, 1994.

[106] Silverman, B. W., *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[107] Singh, S. and Bertsekas, D., "Reinforcement learning for dynamic channel allocation in cellular telephone systems," in *Advances in Neural Information Processing Systems* (Mozer, M. C., Jordan, M. I., and Petsche, T., eds.), vol. 9, pp. 974–980, MIT Press, 1997.

[108] Singh, S. P. and Sutton, R. S., "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, pp. 123–158, 1996.

[109] Sistu, P. B. and Bequette, B. W., "Model predictive control of processes with input multiplicities," *Chemical Engineering Science*, vol. 50, no. 6, pp. 921–936, 1995.

[110] Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., and Juditsky, A., "Nonlinear black-box modeling in system identification: A unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.

[111] Smart, W. D. and Kaelbling, L. P., "Practical reinforcement learning in continuous spaces," in *Proc. 17th International Conf. on Machine Learning*, pp. 903–910, Morgan Kaufmann, San Francisco, CA, 2000.

[112] Su, H. T. and McAvoy, T. J., "Integration of multilayer perceptron networks and linear dynamic models: A Hammerstein modelling approach," *Ind. Eng. Chem. Res.*, vol. 32, pp. 1927–1936, 1993.

[113] Sutton, R., McAllester, D., Singh, S., and Mansour, Y., "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems* (Solla, S. A., Leen, T. K., and Muller, K.-R., eds.), vol. 12, pp. 1057–1063, MIT Press, 2000.

[114] Sutton, R. S., "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the Seventh International Conference on Machine Learning*, (Austin, TX), Morgan Kaufmann, 1990.

[115] Sutton, R. S., "Planning by incremental dynamic programming," in *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 353–357, Morgan Kaufmann, 1991.

[116] Sutton, R. S., *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[117] Sutton, R. S., "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[118] Sutton, R. S., "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems* (Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., eds.), vol. 8, pp. 1038–1044, The MIT Press, 1996.

[119] Sutton, R. S. and Barto, A. G., "Toward a modern theory of adaptive networks: Expectation and prediction," *Psycol. Rev.*, vol. 88, no. 2, pp. 135–170, 1981.

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

[121] Takeda, M., Nakamura, T., Imai, M., Ogasawara, T., and Asada, M., "Enhanced continuous valued Q-learning for real autonomous robots," *Advanced Robotics*, vol. 14, no. 5, pp. 439–442, 2000.

[122] Tesauro, G., "Practical issues in temporal difference learning," *Machine Learning*, vol. 8, pp. 257–277, 1992.

[123] Tesauro, G., "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[124] Tesauro, G., "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–67, 1995.

[125] Thrun, S., "Learning to play the game of chess," in *Advances in Neural Information Processing Systems* (Tesauro, G., Touretzky, D. S., and Leen, T. K., eds.), vol. 7, Cambridge, MA: The MIT Press, 1995.

[126] Thrun, S. and Schwartz, A., "Issues in using function approximation for reinforcement learning," in *Proceedings of the Fourth Connectionist Models Summer School*, (Hillsdale, NJ), Lawrence Erlbaum, 1993.

[127] Tsai, P.-F., Chu, J.-Z., Jang, S.-S., and Shieh, S.-S., "Developing a robust model predictive control architecture through regional knowledge analysis of artificial neural networks," *Journal of Process Control*, vol. 13, pp. 423–435, 2002.

[128] Tsitsiklis, J. N. and Van Roy, B., "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[129] Tsitsiklis, J. N., "Asynchronous stochastic approximation and Q-learning," *Machine Learning*, vol. 16, pp. 185–202, 1994.

[130] Van de Vusse, J. G., "Plug-flow type reactor versus tank reactor," *Chemical Engineering Science*, vol. 19, pp. 964–996, 1964.

[131] Van Roy, B., "Neuro-dynamic programming: Overview and recent trends," in *Handbook of Markov Decision Processes: Methods and Applications* (Feinberg, E. and Shwartz, A., eds.), Boston, MA: Kluwer, 2001.

[132] Watkins, C. J. C. H., *Learning from delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

[133] Watkins, C. J. C. H. and Dayan, P., "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[134] Werbos, P. J., "Advanced forecasting methods for global crisis warning and models of intelligence," *General Systems Yearbook*, vol. 22, pp. 25–38, 1977.

[135] Werbos, P. J., "Approximate dynamic programming for real-time control and neural modeling," in *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches* (White, D. A. and Sofge, D. A., eds.), pp. 493–525, New York: Van Nostrand Reinhold, 1992.

[136] Whitehead, S. D., "Complexity and cooperation in Q-learning," in *Proceedings of the Eighth International Workshop on Machine Learning*, (Evanston, IL), Morgan Kaufmann, 1991.

[137] Williams, R. J. and Baird III, L. C., "Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems," Tech. Rep. NU-CCS-93-14, Northeastern University, College of Computer Science, Boston, MA, 1993.

[138] Wilson, J. A. and Martinez, E. C., "Neuro-fuzzy modeling and control of a batch process involving simultaneous reaction and distillation," *Computers & Chemical Engineering*, vol. 21S, pp. S1233–S1238, 1997.

[139] Wittenmark, B., "Adaptive dual control," in *Control Systems, Robotics and Automation, Encyclopedia of Life Support Systems (EOLSS), Developed under the auspices of the UNESCO* (Unbehauen, H., ed.), Oxford, UK: Eolss Publishers, 2002. Paper 6.43.15.6.

[140] Wonham, M., "Stochastic control problems," in *Stochastic Problems in Control* (Friedland, B., ed.), New York: ASME, 1968.

[141] Zhang, W., *Reinforcement Learning for Job-Shop Scheduling*. PhD thesis, Oregon State University, 1996. Also available as Technical Report CS-96-30-1.

[142] Zhang, W. and Dietterich, T. G., "A reinforcement learning approach to job-shop scheduling," in *Proceedings of the Twelfth International Conference on Machine Learning (ICML '95)*, (San Francisco, CA), pp. 1114–1120, Morgan Kaufmann, 1995.

[143] Zhang, W. and Dietterich, T. G., "High-performance job-shop scheduling with a time-delay TD(λ) network," in *Advances in Neural Information Processing Systems* (Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., eds.), vol. 8, San Francisco, CA: MIT Press, 1996.

VITA

Jong Min Lee was born in Seoul, Korea on February 3, 1973. He graduated from Sorabol High School in 1991. He then attended Seoul National University at Seoul, Korea in 1992. In February 1996, he graduated Summa Cum Laude with a B.S. in Chemical Engineering. He served in the Korea Army from 1997 to 1998. In August 1999, he attended Purdue University at West Lafayette, Indiana. He then transferred to Georgia Institute of Technology in August 2000. From January to May 2002, he worked for Owens Corning as an engineering intern at Advanced Process Control team in Manufacturing Technology Center, Granville, Ohio. His dissertation title was "A Study on Architecture, Algorithms, and Applications of Approximate Dynamic Programming Based Approach to Optimal Control." He defended his thesis on July 8, 2004 and obtained his Ph.D. in Chemical and Biomolecular Engineering with a Minor in Optimization on July 30, 2004.