Value Function Approximation Methods for Linearly-solvable Markov Decision Process

Mingyuan Zhong

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Emanual Todorov, Chair

Randy Leveque

Eric Shea-Brown

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

Value Function Approximation Methods for Linearly-solvable Markov Decision Process

Mingyuan Zhong

Chair of the Supervisory Committee:
Professor Emanual Todorov
Department of Applied Mathematics

Optimal control provides an appealing machinery to complete complicated control tasks with limited prior knowledge. Both global methods and online trajectory optimization methods are powerful techniques for solving optimal control problems; however, each has limitations. The global methods are directly or indirectly based on the Bellman equation, which originates from dynamical programming. Finding the solution of Bellman equation, the value function, or cost-to-go function, suffers from multiple difficulties, including the curse of dimensionality. In the linearly-solvable Markov Decision Process (LMDP) framework, the Bellman equation can be linearized despite nonlinearity in the stochastic dynamical models. This fact permits efficient algorithms and motivates specialized function approximation schemes.

In the average-cost setting, the Bellman equation in LMDP can be reduced to computing the principal eigenfunction of a linear operator. To solve for the value function of the Bellman equation in this cases, we designed two methods, moving least squares approximation and aggregation methods, to avoid matrix factorization and take advantage of sparsity by using efficient iterative solvers. In the moving least square approximation methods, value function is approximated by linear basis constructed from moving least squares fitting. In the aggregation methods, LMDP is approximated by using soft state aggregation over a continuous space. Adaptive schemes for basis placement are developed to provide higher resolution at the regions of state space that are visited most often. Numerical results are

provided.

Approximating value function is not sufficient to apply LMDP in more realistic tasks. We demonstrated that value function methods may require an unrealistic number of base functions to control certain dynamical systems. In order to mitigate the undesirable properties of local and global methods, we explore the possibility of combining value function approximation methods in LMDP with model predictive control (MPC). Exploiting both the value function and the policy generated by solving the LMDP, MPC is able to perform at a level similar to that of MPC alone with long time horizon, but now we may drastically shorten the time horizon of MPC. This also allows LMDP value function approximation methods to be applied to more problems. The results of the implementation of these methods show that global and local methods can and should be combined in real applications to benefit both.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ACKNOWLEDGMENTS

I would first like to thank my advisor, Professor Emanuel Todorov. I can hardly find words to describe how grateful I am to him. In a nutshell, I learned the specifics of control, optimization, research and development, and many aspects of life from him. His guidance and support made my work possible, his enthusiasm always inspires my research and me through the worst of times.

I would like to thank all other members of the committee, Professor Randall LeVeque, Professor Eric Shea-Brown and Professor Brian Fabien. I am honored to have been advised by such great individuals. I really enjoyed courses I took from Professor LeVeque, and I appreciate his efforts on all students in the department including me.

I would like to thank the entire movement control lab. Everyone in the lab are really helpful and insightful. I especially want to thank Dvijotham Krishnamurthy, Dr. Evangelos Theodorou, Dr. Yuval Tassa, Dr. Tom Erez, and Mikala Johnson for their helpful discussions and cooperation. I would also sincerely thank Zhe Xu, Vikash Kumar, Paul Kulchenko, and many others for supports in other aspects. I am really grateful to be part of the lab, and I love everyone in the lab.

I would like to thank my family who continue to support me. Their support and encouragement have driven me forward through the worst of times.

I would like to thank the entire department of Applied Mathematics. I appreciate the courses I took from Professor Bernard Deconinck, Professor Robert O'Malley, Professor Loyce Adams and many other professors in the Applied Mathematics department, and I am grateful for all supports from them. I would like to thank Shari Jacobs, Keshanie Dissanayake, and many others for supporting.

I also would like to thank many other people in the University of Washington. I especially want to thank Professor Paul Tseng for optimization courses I took under his guid and I

am so sadden that I cannot say thank you to him personally any more.

I would also like to thank the National Science Foundation for its support.

DEDICATION

To my parents and my late grandmother

Chapter 1

INTRODUCTION

1.1 Prologue: optimality

Optimality is perhaps one of the most widespread concepts in our daily life. For example, in general, both rational businesspersons and customers tend to *maximize* profit and *minimize* cost. People are likely to believe events that *most* likely happened. The intent to maximize "benefits" also exists in most biological systems, and it is one of the key elements of "decision making".

Scientists have discovered that many processes can be described in the language of optimality (i.e., maximizing or minimizing a certain quantity). One of the earliest attempts at optimality is Fermat's Principle in optics. This principle states that the path taken between two points by a ray of light is the path that can be traversed in the least amount of time. Fermat's Principle is not the cause of the reflection and refraction behaviors it describes, yet it provides a simple mathematical description and calculation. Later, the principle of least action was proposed and serves as the basis for the Lagrangian and Hamiltonian mechanics, which expand the application of traditional mechanic to more complicated systems and provides the mathematical basis for modern physics.

Optimality helps to explain complicated decisions behaviors. In microeconomics and finance, decisions are based on how to maximizing "utilities". Optimality may at least help to explain behaviors qualitatively. However, "utilities" are not well-defined quantities and are hard to obtain quantitatively and may differ from individuals, making the optimality less strict.

Specifically, optimality principles are found in sensorimotor control in biological systems[53], and they may work as an analog of those "first principles" in physics. The optimality principles in sensorimotor control may help us generate a telescopic view of biological processes within complicated mechanisms and may shed light on the movement control of robotics.

When we drive a car, we do our best to ensure a safe, comfortable, and efficient journey, rather than following instructions about how to manipulate steering wheels and pedals. When we are instructing a robot to drive, should we micromanage the robot by telling it every detail, or should we invent intelligent control tools and allow robots to decide those details automatically? Optimality is the answer for how to achieve complicated control tasks.

Many mathematical tools have been invented to enable works based on optimality. Calculus of variations, a field of mathematical analysis that deals with maximizing or minimizing functions, serves as the basis of the Lagrangian and Hamiltonian mechanics. Mathematical optimization provides numerous numerical methods to find the optimal value. Ideally, when a real problem is expressed in optimization, mathematical optimization can automatically find the optimal value and solve the problem.

In this thesis, we propose to study the value function approximation method in the theoretical framework of optimal control, where control design and optimality are brought together.

1.2 Optimal control and robotics

Optimal control[38] deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved. An optimal control problem is formulated as a control problem with a cost function that is a function of state and control variables. The optimal control for a system describes the paths of the control variables or the function of control variables depending on the states and time that minimize the cost function. The optimal control can be obtained by using Pontryagin's maximum principle or by solving the Bellman equation (including the Hamitonian-Jacobi-Bellman equation).

Research in optimal control started in the 1950s and includes applications in multiple fields other than control systems or robotics. For example, as proposed by Nobel laureate in Economics Robert C. Merton, Merton's portfolio problem [31] is a stochastic optimal control problem with an analytical solution describing the decision of an investor about how much to consume and how much to allocate between stocks and a risk-free asset over a time period.

In the field of computer graphics, motions of characters can be generated through optimal control directly or indirectly. Examples include human locomotion [27], locomotion of artificial animals[58] and hand manipulation of objects[28]. These methods use optimal control directly or indirectly to expedite the control design of characters in animation, and will profoundly benefit the movie and video game industries. On the other hand, especially in movie production, those methods usually do not really need to run algorithms as fast as physical world, and they do not need to deal with noises and inaccuracies arising in reality, so optimal control in computer graphics is easier than that of the physical world.

Optimal control originates from and is widely used in the control systems and robotics. The result of a special class of optimal control problems, Linear Quadratic Regulator(LQR), is widely used. Certain optimal control problems, such as minimum jerk[20], have analytical solutions for the trajectory. More general optimal control results without such analytical solutions include but are not limited to helicopter[1, 2] and pingpong[26].

Optimal control may have different goals in different circumstances. In problems with sufficient actuators, control tasks may be easy and optimality is likely to explain behaviors under redundancy. In those cases, optimal control can be used to actually minimize/maximize certain meaningful quantities such as fuel efficiency, time to reach target, and so on. Alternatively, when the controller is difficult to design, optimal control is used to find a feasible solution. In the second case, the exact form of the cost function of a "good" controller is unknown.

Certain control tasks such as locomotion are still hard to design today, and optimal control clearly can play a significant role in the design process. Real-time control requires a method that computes faster than the physical world, which motivates more efficient optimal control methods. The rapid development of computational infrastructures and efficient physical simulators [56] gradually make these possible, yet efficient optimal control methods are still demanding. A special class of optimal control problems can linearize the equations. We would like to explore the numerical methods for the value function approximation approach in such problems, and we hope research in this field will contribute to our long-term goal: automate control design to intelligently achieve complicated control tasks such as locomotion.

1.3 Numerical methods for optimal control

Due to the complexity and nonlinearity of control tasks, optimal control in applications is frequently solved by numerical algorithms. Similar to the theoretical basis of optimal control, the Pontryagin's maximum principle and the Bellman equation, the numerical methods for optimal control can also be divided into two categories: local methods and global methods. Local methods solve for a trajectory, and global methods solve for a control policy that is dependent on state variables.

Local methods are widely used in applications. They are mostly based on Pontryagin's maximum principle but some works include local methods based on Bellman equations such as iLQG or DDP. It generally scales better in systems with high degrees of freedom, but its efficiency is significantly affected by the length of the planned trajectory. The downside of local methods are the following: (1) the solution may converge to a local minimum; (2) local methods consider only the system around the trajectory, so the resulting controller may not be sufficiently robust during perturbations.

On the other hand, global methods may avoid local minima, but they may suffer from the curse of dimensionality. They are generally based on the Bellman equation or its variants and solve for indirect quantities such as the value function, Q function, and so forth, or direct quantities such as control policy. Accurately expressing those functions in a control problem is generally prohibitive due to high dimensionality. Successful global methods generally require carefully approximating the value function or other globally defined functions, the skills of which are not transferable between dynamical systems. Among them, value function approximation, which is based on value iteration, is the fastest of global methods.

Value function approximations suffer from the curse of dimensionality, while the Linearly-solvable Markov Decision Process(LMDP)[25, 47, 48] avoids discretization in control space, thus making the problem more tangible. This fact makes designing methods for value function approximation in LMDP promising.

1.4 Contributions

In this thesis, we aim to contribute to answering the big question: how to automate control design in complicated control problems such as locomotion. We focus on the value function approximation method in the global method of optimal control in the Linearly-solvable Markov Decision Process(LMDP). We would like to design numerical methods tailored to solve LMDP problems efficiently and reliably, and explore their applications as well as their limitations.

First, this thesis explores function approximation methods, moving-least-square approximations[61] and aggregation methods [60], which are tailored to solve for the leading eigenpair of a linear approximator in order to solve for the value function of LMDP problems. These methods exploit the benefits brought by LMDP framework and enrich both value function approximation methods.

Second, this thesis demonstrates limitations in value function approaches of LMDP. It also proposes methods to detour such limitations. Notably, one way to expand LMDP's application is to combine it with MPC[59]. This combination improves both MPC and LMDP methods.

1.5 Outline

In this thesis, we will first introduce background materials in Chapter 2. Then we will illustrate the basics of Linearly-solvable MDP in Chapter 3. After that, we will introduce our major methods: moving least squares in Chapter 4 and aggregation methods in Chapter 5. Then we will discuss limitations of LMDP based value function approximation methods in Chapter 6. Finally, in Chapter 7 we will combine MPC with the aggregation method in applications such as acrobot.

Chapter 2

BACKGROUND

In this chapter, we will first briefly review traditional results in control theory. Then we will provide a big picture of optimal control and numerical methods. Finally we will focus on the Markov Decision Process and derive the Bellman equations for stochastic optimal control.

2.1 Classical control theory and methods

2.1.1 Overview

Control theory [33] is an interdisciplinary branch of engineering and mathematics that deals with the behavior of dynamical systems with inputs. The inputs and outputs of a continuous control system are generally related by differential equations. If those dynamical systems are linear with constant coefficients, based on Laplace transformation, the differential equations can be converted to algebraic equations with polynomials and the solution can be computed analytically. This provides the basis for classical control theory, yet it is in theory limited to linear systems.

In contrast to the frequency domain analysis of the classical control theory described above, modern control theory utilizes the time-domain state space representation, a mathematical model of a physical system as a set of input, output, and state variables related by first-order differential equations. The time-domain approach (also known as the "state space representation") provides a convenient and compact way to model and analyze systems with multiple inputs and outputs. Unlike the frequency domain approach, the use of the state space representation is not limited to systems with linear components and zero initial conditions.

2.1.2 PID control

PID control is one of the most widely used controllers in industry. A PID controller considers an "error" value, which is the difference between a measured variable and a desired value. The controller attempts to minimize the error by adjusting the inputs. The input/control signal $u(t)$ is determined by

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t), \tag{2.1}$$

where $e(t)$ represents the error term. The tuning parameters are (1) K_p proportional gain, (2) K_i integral gain (3) K_d Derivative gain. If $K_d = 0$, this controller becomes a PI controller. Similarly, if $K_i = 0$, this controller becomes a PD controller. PID controller design requires carefully tuning those parameters, which can be done manually or automatically.

A PID controller can only control a system that is fully actuated, and it requires a "desired value". In some realistic tasks, desired value or reference trajectory may be either absent or unknown. So although PID controllers are widely used, it is still helpful to improve other methods to control more complicated systems.

2.1.3 Stability

Stability is the most important feature of a desired controller. In a linear dynamical system, stability can be judged easily. There are extensive results in judging the stability of complicated linear systems, and commercial softwares are available to adjust parameters to achieve stability.

In a nonlinear system, stability can be judged by the Lyapunov function $\Lambda(x)$, which is 0 at equilibrium ($\Lambda(x) = 0$) and decreases over time $\frac{d}{dt}\Lambda(x) \leq 0$. In practice, finding a Lyapunov function is very hard and it is even hard to find a controlled-Lyapunov function. For a control system, a Lyapunov function may be found with SOS (sums of squares) programming[13, 24, 30, 34, 35]. The Lyapunov function is different from the value function we are going to describe later, (To avoid confusion, we used Λ to represent Lyapunov function, which is more often expressed in V in other literature).

2.1.4 Controllability and Under-actuation

The concept of controllability denotes the ability to move a system around in its entire configuration space using only certain admissible manipulations.

If an external input can move the internal state of a system from any initial state to any other final state in a finite time interval, the system is controllable.

Different systems' controllability can be judged by different criteria, and they are related to system's behavior over a time horizon. For example, in a linear-time-invariant system, $\dot{x}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^r$, the controllability matrix is given by

$$R = \left[B\, AB\, ...\, A^{n-1}B\right], \tag{2.2}$$

and the system is controllable if R has full rank.

Any system in engineering should be controllable in the desired state space, but the controllability of a system does not describe the difficulty of the control task in that system. For example, when learning to drive, the skill of parallel parking is a difficult skill to learn because a car cannot move left or right without moving forward or backward. On the other hand, parallel parking is possible; in the formal language, the dynamical system of a car is controllable.

Underactuation[46] is an extreme of such case. It describes a mechanical device that has a lower number of actuators than degrees of freedom. That makes the control design harder (PID controller cannot be directly applied here). On the other hand, it may make the control design more efficient. Underactuation is a key feature of a walking robot, which cannot fully control the contact force between the ground and itself.

2.2 Optimal control theory

2.2.1 Overview

Optimal control[38, 54] is a mathematical branch with applications in science engineering and even finance. In theory it is an extension of the calculus of variations, and it yields control policies based on mathematical optimization methods. The development of this field dates back to 1950s by Lev Pontryagin and his collaborators in the Soviet Union and

Richard Bellman and his collaborators in the United States.

Let $\mathbf{x}$ denote the state of a dynamical system and $\mathbf{u}$ denotes the action that the controller chooses. Let $f(\mathbf{x}, \mathbf{u})$ represents the state results from applying action $\mathbf{u}$ at state $\mathbf{x}$, and $l(\mathbf{x}, \mathbf{u})$ represents the cost of applying action $\mathbf{u}$ in state $\mathbf{x}$. The problem can be formulated by: finding a control sequence $\mathbf{u}_0, ..., \mathbf{u}_{n-1}$ and a state sequence $\mathbf{x}_0, ..., \mathbf{x}_n$ to minimize the total cost $J(\mathbf{x}, \mathbf{u})$

$$J(\mathbf{x}_0, \mathbf{u}_0) = \Sigma_{i=0}^{n-1} l(\mathbf{x}_i, \mathbf{u}_i). \tag{2.3}$$

In continuous time setting, the problem can be formulated similarly by replacing summation with an integral: finding a control sequence $\mathbf{u}(t)$ and a state sequence $\mathbf{x}(t)$ to minimize the total cost $J(\mathbf{x}, \mathbf{u})$

$$J(\mathbf{x}(0), \mathbf{u}(0)) = \int_0^t l(\mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau, \tag{2.4}$$

where $l(\mathbf{x}(\tau), \mathbf{u}(\tau))$ is *cost rate* describing the rate of accumulating cost.

The optimization conditions of total cost J functions leads to either the Bellman equation or the maximum principle. We will briefly introduce them in this section and provide a more formal explanation of the Bellman equation in our next section about the Markov Decision Process.

2.2.2 Bellman equation

If a given state-action sequence $\mathbf{x}_i, ..., \mathbf{x}_n, \mathbf{u}_i, ..., \mathbf{u}_{n-1}$ is optimal, then the remaining sequence removing the first state and action $\mathbf{x}_{i+1}, ..., \mathbf{x}_n, \mathbf{u}_{i+1}, ..., \mathbf{u}_{n-1}$ is also optimal. This is known as the Bellman optimality principle, and this principle enables us to solve this question via dynamic programming (DP).

This principle relies rely on a function summarizing the cumulative cost of subsequence, the optimal value function (or optimal cost-to-go function) $V(\mathbf{x})$, which denotes the minimal total cost for completing the task starting from state $\mathbf{x}$. Then value function can be calculated backwards as follows: considering every action available at the current state,

adding its immediate cost to the optimal value of the resulting next state, and choosing an action for which the sum is minimal. This can be summarized in the following equation

$$v(\mathbf{x}) = \min_{\mathbf{u}} \left[l(\mathbf{x}, \mathbf{u}) + v(f(\mathbf{x}, \mathbf{u})) \right], \tag{2.5}$$

$$u(\mathbf{x}) = \arg\min_{\mathbf{u}} \left[l(\mathbf{x}, \mathbf{u}) + v(f(\mathbf{x}, \mathbf{u})) \right]. \tag{2.6}$$

(2.5) and (2.6) are called Bellman equations. Bellman equation differs in formulation, and we will discuss in greater details in the MDP section.

In continuous-time formulation, the Bellman equation will become the Hamiltonian-Jacobi-Bellman equation, which is a partial differential equation.

$$-v_t(\mathbf{x}) = \min_{\mathbf{u}} \left[l(\mathbf{x}, \mathbf{u}) + f(x, u)^T v_x(f(\mathbf{x}, \mathbf{u})) \right], \tag{2.7}$$

$$u(\mathbf{x}) = \arg\min_{\mathbf{u}} \left[l(\mathbf{x}, \mathbf{u}) + f(x, u)^T v_x(f(\mathbf{x}, \mathbf{u})) \right]. \tag{2.8}$$

We will provide more details about variations of Bellman equations according to different ways to define total cost J in later sections. Bellman equation or HJB equation is easy to derive, and the hard part is finding the value function.

2.2.3 Maximum principle

Alternatively, we can find a necessary condition for the optimality of trajectory.

In continuous-time formulation, define a co-state $p(t)$ corresponding to v_x in the HJB equation. The Hamiltonian function is defined as $H(\mathbf{x}, \mathbf{u}, \mathbf{p}, t) = l(\mathbf{x}, \mathbf{u}, t) + \mathbf{f}(\mathbf{x}, \mathbf{u})^T \mathbf{p}$ and the Maximum principle is expressed as

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \frac{\partial}{\partial \mathbf{p}} H(\mathbf{x}, \mathbf{u}, \mathbf{p}, t) \\ -\dot{\mathbf{p}}(t) &= \frac{\partial}{\partial \mathbf{x}} H(\mathbf{x}, \mathbf{u}, \mathbf{p}, t) \\ \mathbf{u}(t) &= \arg\min_{\mathbf{u}} H(\mathbf{x}, \mathbf{u}, \mathbf{p}, t) \end{aligned} \tag{2.9}$$

Similar results exist in discrete cases. Compared to the HJB equation, the maximum principle leaves an ordinary differential equation and avoids solving for a function defined on the entire state space. The computational complexity for ODE solutions based on the

maximum principle grows linearly with the state dimensionality in ideal cases, so the curse of dimensionality is avoided. One drawback is that maximum principle may have multiple solutions (one of which is the optimal solution). Here is an example. When using maximum principle to control a robot to reach arbitrary choice of multiple targets, plans to reach each of those targets may be locally optimal. Another drawback is that the solution to the maximum principle is valid for a single initial state, and if the initial state were to change we would have to solve the problem again.

2.2.4 LQR- a special case

In a very special case, where dynamics is linear and cost function l is quadratic to both state and control variables, the Bellman equation can be solved analytically using the Linear Riccati equation. Imagine the following continuous-time linear deterministic optimal control problem

$$\begin{aligned} dynamics: \dot{\mathbf{x}} &= A\mathbf{x} + B\mathbf{u} && (2.10)\\ cost\,rate: l(\mathbf{x},\mathbf{u}) &= \frac{1}{2}\mathbf{u}^T R\mathbf{u} + \frac{1}{2}\mathbf{x}^T Q\mathbf{x} && (2.11)\\ & && (2.12) \end{aligned}$$

The value function is

$$v(\mathbf{x},t) = \mathbf{x}^T V(t)\mathbf{x}/2, \tag{2.13}$$

and the control policy is

$$u(\mathbf{x},t) = -R^{-1}B^T V(t)\mathbf{x} = -R^{-1}B^T v_{\mathbf{x}}(\mathbf{x},t). \tag{2.14}$$

The matrix $V(t)$ is symmetric and it is determined by the continuous-time linear Riccati equation

$$-\dot{V}(t) = Q + A^T V(t) + V(t)A - V(t)BR^{-1}B^T V(t). \tag{2.15}$$

In discrete-time, it can also be formulated into the Riccati equation for matrix V. These results give us the analytical results of the Bellman equation and can be used directly without mathematical optimization algorithms. Because of this, LQR is widely used in

applications where linearization will not bring significant error. (In Matlab, continuous-time and discrete-time linear Riccati equations can be solved by command “care” or “dare” respectively.)

2.2.5 Stochastic optimal control

We discussed optimal control in deterministic dynamical systems above. In optimal control problems in the stochastic dynamical system, the results above will be further adjusted. The Bellman equation approaches will not be very different from those of the deterministic case. The Bellman equation will contain expectation terms, and the HJB equation becomes a second-order PDE. If the noise is gaussian, LQR will becomes LQG(Linear quadratic gaussian control), yields exactly the same Riccati equation as in deterministic cases. On the other hand, maximum principle will cease to remain an ODE, which means we will normally need a global value function in stochastic cases.

There are benefits considering a control problem in stochastic optimal control rather than the deterministic counterparts. First, stochastic dynamical models directly arise from dynamical models with uncertainty, and stochastic optimal control helps to control the system when noise is present. In contrast to applications in space(such as controlling satellites), where optimal control was first used, in applications involving real robots, noise in both control and estimation can dramatically affect control policy. Moreover, stochastic dynamical models bring computational convenience. For example, stochastic optimal control smoothes the value function[44] and handles collision better[45] than the deterministic optimal control. Linearly-solvable MDP, which is based on stochastic models, linearizes the Bellman equation despite dynamics being nonlinear (see Chapter 3).

2.2.6 Discrete time vs. continuous time formulation

Traditionally, optimal control is based on continuous time formulation because physical models are defined in continuous time. On the other hand, discrete time formulation can handle cases with discontinuous dynamics. Discontinuous dynamics naturally arises in tasks such as object manipulation and walking. Moreover, when relying on a computer to control

a system, the signal is discrete in time. In the later chapters we will mostly focus on discrete time formulation.

2.2.7 Cost function

The cost functions of optimal controls are defined to achieve desired tasks. There are two ways to look at cost function and the goal of optimal control. If the model is accurate and the controller is easy to design, optimal control can be used to actually minimize/maximize certain meaningful quantities such as fuel efficiency, time to reach target and so on. In these cases, cost functions are defined carefully to represent such goals. Alternatively, when the controller is difficult to design, cost functions are designed to help the optimal controller to discover a feasible solution. In this case, the cost function is complicated. For example, when controlling a robot, the least desired behavior is falling off, yet we may still want it to go to certain target, so the cost function includes information about both safety and target-reaching strategies. Designing cost function in these cases is difficult and requires carefully tuning the parameters. Alternatively, inverse optimal control learns and constructs cost function automatically. In the later chapters, we will use arbitrarily designed and carefully tuned cost functions.

2.3 Numerical methods for optimal control

2.3.1 Overview

Similar to the two pillars in optimal control theory, methods in the optimal control can be classified into local methods and global methods. Here we briefly introduce major methods classified into two categories.

2.3.2 Local methods

Local methods, or trajectory-based methods, solve for a series of states and control signal over a time horizon. The solution of these algorithms is state control sequence $\mathbf{u}_0, ..., \mathbf{u}_{n-1}$, $\mathbf{x}_0, ..., \mathbf{x}_n$

Naturally the maximum principle leads to these methods. It can be formulated as a

numerical solution of a boundary value problem with a final state fixed, or an initial value problem with final cost fixed.

In certain contexts, this approach is directly derived from state-control representation. It minimizes total cost J directly, subject to constraints on the dynamical model. The solution may not be feasible due to violations of the physical model, although minor violations of the physical model may be OK in certain applications.

Alternatively, certain methods, such as iLQG[52] or DDP[23], are based on local quadratic approximation of the Bellman equation instead. They apply a series of Bellman equation over the future trajectory and use tricks to update policy without violating local approximation.

Local methods avoid the curse of dimensionality caused by approximating a function over the entire state space; they are very powerful and have proven performance in both simulation and real robots. On the other hand, local methods have several drawbacks. (1) Local methods need good initialization. Local methods applied on practical problems normally form a non-convex optimization problem, so good initial solutions are crucial to performance. This is especially difficult in certain problems such as underactuated problems, where meaningful initial control and state sequences are very hard to find. (2) Local methods lack feedback. A local method solves only for a trajectory from an initial state, but in reality, both estimator and actuators bring errors to the system and the future steps can hardly remain exactly as planned. To recover from diverging from the planned trajectory, a special class of local methods, called receding horizon control, or model predictive control, is proposed. It uses only the first control signal at each time and solves the problem again with the local method in a new time horizon. This provides feedback towards errors, but the resolve step also makes MPC slower than local methods without resolving the planned trajectory.

2.3.3 Global methods

Overview

Global methods generally are based on the Bellman equation or its variants[10, 11, 41]. They approximate certain functions defined on the entire state, and/or control space and in most cases it eventually solves for a control signal dependent on state $u(x)$. The time-invariant feature brings convenience and requires less storage, but the history of previous steps is also absent in $u(x)$. Global methods attempting to express the time-dependent control signal $u(x, t)$ may carry time-dependent control policy back, but it requires much more storage or a parameterized form.

Model-based vs model-free

One way to classify global methods is by the existence of a dynamical model. Assuming we have an accurate dynamical model, we use methods such as value iteration or policy iteration, which are directly derived from the Bellman equation. In value iteration, the value function $v(x)$ is expressed globally with a function approximator. The algorithm iteratively updates $v(x)$ until it converges. The control policy $u(x)$ is therefore obtained by the $\arg\min$ operator. In policy iteration, both $v(x)$ and $u(x)$ are expressed globally with function approximators. The algorithms iteratively update both $v(x)$ and $u(x)$ until $u(x)$ converges.

The dynamical model for an actual robot may be very complicated with unknown parameters. Such complexity may be overcome by assuming that we do not have any dynamical model and constructing global methods with actual data. These methods are called reinforcement learning, and some such methods in reinforcement learning rely on the quantity Q function $Q(x, u)$, which is related to the value function as $v(x) = \min_u Q(x, u)$. Model-free methods are normally slower but can deal with errors in dynamical models. Since dynamical models and corresponding software for complicated systems such as humanoid are normally available yet inaccurate; there is no reason to always start from scratch. Model-based methods can be used as a warm start to model-free methods.

Direct vs indirect

Certain methods attempt to approximate control policy directly to avoid errors caused by retrieving policy from either hte value function $v(x)$ or the Q function $Q(x, u)$. Examples are policy search methods[32] and policy gradient methods[40]. Direct methods may require more storage and are slower than their indirect counterparts focusing on the value function or Q function.

Curse of dimensionality

The biggest challenge for global methods is the curse of dimensionality. In robotics, the dimensionality of state space is usually high. Therefore it is prohibitive to represent a real-valued or vector-valued function over the entire state space (or combined state and control space) with grids (storage requirement growing exponentially with the increase of dimensionality). The difficulty can be lessened by dimensionality reduction or solved using the Bellman equation analytically when possible (e.g. LQR).

2.4 Markov decision process

2.4.1 Markov decision process and Bellman equation

The Markov Decision Process is a stochastic optimal control in discrete state space with discrete time steps. It provides the mathematical foundation for a decision making process where outcomes are partially random and partially controlled. It is used in a wide area of disciplines, including robotics, economics, and manufacturing.

Definition

The Markov Decision Process is a framework describing a discrete time stochastic control process. MDP can be expressed as a 4-tuple (S, A, P, R), which is described below. (Some contexts use 5-tuple, but the meanings are similar.)

1. S is a set of states.

2. A is a set of available actions.

3. P(s', a, s) represents the transition probability.

4. R(s', a, s) represents the rewards at each time step.

At each time step, the process is in some state $s \in S$, and the agent (decision maker) may choose any action a that is available in state s. At the next step, the process randomly moves into a new state $s' \in S$, and gives the agent a corresponding reward $R(s, a, s')$ (rewards can be understood as negative cost).

The transition probability between current state s and the next step state s', $P(s'|s, a)$ is influenced by both current state s and action a, but it is irrelevant to all previous states and actions. Thus the state transition of MDP processes the Markov property. In other words, Markov Decision Process (MDP) is memoryless.

MDP is an extension to Markov chain; if there is only one action available, MDP will reduce itself to a Markov chain. The difference is the addition of actions and rewards, which represent the decision maker's available actions and their outcomes.

Problem

MDP seek to find a policy $\pi : S \to A$ maximizing the accumulated rewards (minimizing accumulated costs) over time in some way. π specifies the action the agent will make at each state s. Different ways to accumulate rewards create different formulations of the Bellman equation, which we will explain in greater details later. The accumulated cost is called *value function* or *cost-to-go function*.

Let's take the *discounted cost infinite horizon formulation* as an example. In this case, the value function is accumulated as $\Sigma_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})$,

γ is the discount factor that satisfies $0 < \gamma < 1$. γ represents to what extent we want to discount the outcomes of future actions.

Bellman equation

Solving for optimal policy π is possible through either dynamical programming or linear programming. This yields the Bellman equation.

In the **discounted cost infinite horizon formulation**, the policy specific value function is

$$V^{\pi}(s_0) = E\{\Sigma_{t=0}^{\infty}\gamma^t R(s_t, \pi(s_t), s_{t+1}\}. \tag{2.16}$$

the Bellman equation and optimal policy are formulated as,

$$\pi^*(s) = \arg\max_a E_{s'\sim P(\cdot|s,a)}\{R(s,a,s') + \gamma V^*(s')\}, \tag{2.17}$$

$$V^*(s) = E_{s'\sim P(\cdot|s,a)}\{R(s,\pi^*(s),s') + \gamma V^*(s')\}. \tag{2.18}$$

In the **finite horizon formulation**, value function only summate rewards R over a finite time horizon in the future, ignoring future consequences. We need to define *final cost* $V_f : S \to R$ describes the future rewards. So the policy-dependent value function is

$$V_t^{\pi}(s_t) = E\{\Sigma_{\tau=t}^{t_f-1} R(s_\tau, \pi_\tau(s_\tau), s_{\tau+1} + V(s_{t_f})\}. \tag{2.19}$$

This yields the Bellman equation in the following form;

$$V_t^*(s) = E_{s'\sim P(\cdot|s,\pi_t^*(s))}(R(s,\pi_t^*(s),s') + V_{t+1}(s')), \tag{2.20}$$

with the time dependent policy as

$$\pi_t^*(s) = \arg\max_a E_{s'\sim P(\cdot|s,a)}(R(s,a,s') + V_{t+1}(s')), \tag{2.21}$$

In the **first exit formulation**, the value function will keep accumulating until it reach certain "exit states", such as target or bad states. We call the set of those exit states A, and the corresponding cost is $V(s) = g(s), s \in E$.

$$V^{\pi}(s_0) = E\{\Sigma_{\tau=0}^{t_f-1} R(s_\tau, \pi_\tau(s_\tau), s_{\tau+1}) + g(s_{t_f})\}. \tag{2.22}$$

Thus the Bellman equation for this formulation is

$$V^*(s) = E_{s'\sim P(\cdot|s,\pi^*(s))}(R(s,\pi^*(s),s') + V(s')), V^*(s) = g(s), s \in E, \tag{2.23}$$

$$\pi^*(s) = \arg\max_a E_{s'\sim P(\cdot|s,a)}(R(s,a,s') + V(s')). \tag{2.24}$$

In the **infinite horizon average-cost formulation**, we consider the limiting average value function when the time horizon goes to infinity. Thus the policy specific value function is

$$\tilde{V}^{\pi}(s_0) = \lim_{t_f \to \infty} \frac{1}{t_f} V^{\pi}(s_0) = \lim_{t_f \to \infty} \frac{1}{t_f} E\{\Sigma_{\tau=0}^{t_f-1} R\left(s_\tau, \pi(s_\tau), s_{\tau+1}\right)\}. \tag{2.25}$$

The resulting Bellman equation is

$$\pi^*(s) = \arg\max_a E_{s' \sim P(\cdot|s,a)}(R(s,a,s') + \tilde{V}^*(s')), \tag{2.26}$$

$$\tilde{V}^*(s) + c = E_{s' \sim P(\cdot|s,\pi^*(s))}(R(s,\pi^*(s),s') + \tilde{V}^*(s')). \tag{2.27}$$

2.4.2 Algorithms for MDP

Value iteration

In the value iteration method, value function $v(s)$ in the Bellman equation is updated iteratively, and the policy $\pi(s)$ is not stored; it is only calculated when necessary. For example, in the discounted-cost formulation,

$$V^{(n+1)}(s) = \max_a E_{s' \sim P(\cdot|s,a)}(R(s,a,s') + \gamma V^{(n)}(s')). \tag{2.28}$$

Policy iteration

In the policy iteration method, policy $\pi(s)$ and value function given policy $V^{\pi}(s)$ are updated iteratively. For example, in the discounted-cost formulation,

$$\pi^{(n+1)}(s) = \max_a E_{s' \sim P(\cdot|s,a)}(R(s,a,s') + \gamma V^{\pi^{(n)}}(s')). \tag{2.29}$$

Policy iteration always converges in a finite number of iterations. This is because (1) the algorithm cannot cycle because each iteration is an improvement and (2) the number of different policies is finite (the state and control spaces are finite).

Q learning

Reinforcement learning, [41], attempts to deal with cases where transition probability or rewards are unknown. Q learning is an important branch of methods in reinforcement

learning and it is closely related to the value function approaches discussed here.

Instead of considering value function $V(s)$, the Q learning considers Q function $Q(s,a)$, which describes the quality of state-action combination. The Q function can be related to a value function as $V(s) = \min_a Q(s,a)$. Q learning relies on sampling based on the actual dynamical model rather than fixed probability transition. In theory, Q learning may converge to the results of value iteration given certain assumptions.

An example of Q learning is

$$Q^{(n+1)}(s,a) = (1-\alpha)Q^{(n)}(s,a) + \alpha(R(s,a,s') + \gamma \max_a Q^{(n)}(s',a)), \tag{2.30}$$

where α is the learning rate describing to what extent the newly acquired information will override the old information. $\alpha = 0$ would make the Q function not learn anything, while $\alpha = 1$ would make the agent consider only the most recent information.

2.4.3 Motivation of this thesis

Although MDP is a promising framework with applications, none of the methods discussed above scale easily to certain problems with high dimensionalities: (1) they normally express a function in a very large space, which make them too slow, with huge storage demand; (2) they requires computation of max or min operators over all available actions in each iteration. Manually tuning parameters and restricting the number of available actions can make MDP applicable to certain hard problems; yet such practice is neither automatic nor scalable. LMDP framework avoids discretization in the control space and replaces the minimize operator with a linear operator, so this framework is more promising in developing scalable numerical methods than MDP in general.

‘

Chapter 3

LINEARLY SOLVABLE MARKOV DECISION PROCESS

3.1 Linearly-solvable MDPs

3.1.1 Overview

Consider an MDP with state $\mathbf{x} \in X \subseteq \mathbf{R}^n$. Let $p(\mathbf{x}'|\mathbf{x}, u) \equiv u(\mathbf{x}'|\mathbf{x})$ denote the transition probability given a certain control signal u, and $p(\mathbf{x}'|\mathbf{x})$ denote the transition probability without any control, also known as passive dynamics. $u(\mathbf{x}'|\mathbf{x})$ and $p(\mathbf{x}'|\mathbf{x})$ have to lie in the same subspace, i.e. $u(\mathbf{x}'|\mathbf{x}) = 0$ where $p(\mathbf{x}'|\mathbf{x}) = 0$.

The optimal cost-to-go function is given by the Bellman equation

$$v(\mathbf{x}) = \min_u \{l(\mathbf{x}, u) + E_{\mathbf{x}' \sim u(\cdot|\mathbf{x})}[v(\mathbf{x}')]\}. \tag{3.1}$$

Note that u under min is the transition probability distribution $u(\mathbf{x}'|\mathbf{x})$ rather than a control vector.

For this problem class, the immediate cost function is defined as

$$\tilde{l}(\mathbf{x}, u) = \tilde{q}(x) + KL(u(\cdot|\mathbf{x})||p(\cdot|\mathbf{x})), \tag{3.2}$$

where KL denotes the Kullback-Leibler divergence between two probability distributions. Define the *desirability* function $z(\mathbf{x}) = \exp(-v(\mathbf{x}))$, where $v(\mathbf{x})$ is the optimal cost-to-go function. Then (3.1) and (3.2) become (shown in [48])

$$-\log(z(\mathbf{x})) = \tilde{q}(x) \min_u \{E_{\mathbf{x}' \sim u(\cdot|\mathbf{x})}[\frac{u(\mathbf{x}'|\mathbf{x})}{p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')}]\}.$$

The minimizing expression resembles KL divergence between $u(\mathbf{x}'|\mathbf{x})$ and $p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')$ except $p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')$ is not normalized to 1. Because KL divergence gets its global minimum 0 when two distributions are equal, the optimal control law can be expressed as follows

without minimization

$$u^*(\mathbf{x}'|\mathbf{x}) = \frac{p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')}{\mathcal{G}[z](\mathbf{x})}, \tag{3.3}$$

where the operator $\mathcal{G}$ is defined as

$$\mathcal{G}[z](\mathbf{x}) = \int_{\mathbf{x}' \in X} p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')d\mathbf{x}'. \tag{3.4}$$

The Bellman equation for infinite-horizon average-cost problems can be written as

$$\exp\left(\tilde{q}(\mathbf{x}) - \tilde{c}\right) z(\mathbf{x}) = \mathcal{G}[z](\mathbf{x}). \tag{3.5}$$

The desirability function is the principal eigenfunction of $\exp\left(-\tilde{q}(\mathbf{x})\right)\mathcal{G}[z](\mathbf{x})$ and is guaranteed to be positive. The corresponding eigenvalue is $\lambda = \exp(-\tilde{c})$, where $\tilde{c}$ is the average cost per step.

In summary, LMDP consider the transition probability $p(\mathbf{x}'|\mathbf{x}, u) \equiv u(\mathbf{x}'|\mathbf{x})$ as a control variable. Two conditions have to be met:(1)$u(\mathbf{x}'|\mathbf{x}) = 0$ where $p(\mathbf{x}'|\mathbf{x}) = 0$;(2)cost is in KL divergence (3.2). Then the Bellman equation is linear for desirability function $z(\mathbf{x}) = \exp(-v(\mathbf{x}))$.

3.1.2 Other formulations

For the first-exit formulation we have

$$\exp\left(\tilde{q}(\mathbf{x})\right) z(\mathbf{x}) = \mathcal{G}[z](\mathbf{x}), z(\mathbf{x}) = \exp(-q_T(\mathbf{x})), x \in A, \tag{3.6}$$

where A are terminal states. This formulation makes LMDP a linear equation rather than an eigenfunction problem.

For the finite horizon formulation, we have

$$\exp\left(\tilde{q}(\mathbf{x})\right) z(\mathbf{x}, t) = \mathcal{G}[z](\mathbf{x}, t+1) \tag{3.7}$$

For the discounted cost formulation, we have

$$\exp\left(\tilde{q}(\mathbf{x})\right) z(\mathbf{x}) = \mathcal{G}[z^\alpha](\mathbf{x}). \tag{3.8}$$

This is no longer a linear equation, yet it still avoids the minimization operator

3.2 Linearly-solvable controlled diffusion

Here we consider a class of continuous-time optimal control problems with the following stochastic dynamics:

$$d\mathbf{x} = \mathbf{a}(\mathbf{x})dt + B(\mathbf{x})(\mathbf{u}dt + \sigma d\omega), \tag{3.9}$$

where $\omega(t)$ represents Brownian motion, and σ is the noise magnitude. The cost function of state and control is in the form

$$l(\mathbf{x}, \mathbf{u}) = q(\mathbf{x}) + \frac{1}{2\sigma^2}\|\mathbf{u}\|^2, \tag{3.10}$$

where $q(\mathbf{x})$ is a state cost function. Note that here $l(\mathbf{x}, \mathbf{u})$ is defined on control vector $\mathbf{u}$ rather than $u(\mathbf{x}'|\mathbf{x})$ defined in 3.1. The noise is assumed to lie in the same subspace as the control. The fact that the noise amplitude also appears in the cost function is unusual; however, $l(\mathbf{x}, \mathbf{u})$ can be scaled by σ^2 without changing the optimal control law, and this scaling factor can be absorbed in the state cost $q(\mathbf{x})$, so this is not a restriction. Now we can discretize this dynamical system. The one-step transition probability under the passive dynamics is approximated as a Gaussian distribution

$$p(\mathbf{x}'|\mathbf{x}) = \mathcal{N}(\mathbf{x} + h\mathbf{a}(\mathbf{x}), h\Sigma(\mathbf{x})), \tag{3.11}$$

where h is the time duration of this step, $h\Sigma(\mathbf{x}) = h\sigma^2 B(\mathbf{x})B(\mathbf{x})^T$ is the noise covariance.

The transition probability with control $\mathbf{u}$ is

$$p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) = \mathcal{N}(\mathbf{x} + h\mathbf{a}(\mathbf{x}) + hB(\mathbf{x})\mathbf{u}, h\Sigma(\mathbf{x})), \tag{3.12}$$

while the formula for KL divergence between Gaussians gives

$$KL(p(\mathbf{x}'|\mathbf{x}, \mathbf{u})||p(\mathbf{x}'|\mathbf{x})) = \frac{h}{2\sigma^2}\|\mathbf{u}\|^2. \tag{3.13}$$

Thus, the familiar quadratic energy cost is a special case of the KL divergence cost defined earlier. It can be shown that in the limit $h \to 0$, the solution to the above discrete-time problem converges to the solution of the underlying continuous-time problem. Therefore, if we define

$$\tilde{q}(\mathbf{x}) = hq(\mathbf{x}), \tilde{c} = hc, \tag{3.14}$$

the continuous problem is approximated as a continuous-space discrete-time LMDP. In the infinite-horizon average-cost setting, (3.5) becomes

$$\exp\left(hq(\mathbf{x}) - hc\right) z(\mathbf{x}) = \mathcal{G}[z](\mathbf{x}). \tag{3.15}$$

In linearly-solvable controlled diffusions, Bellman equations can also be linearized in a continuous-time formulation and represented with derivatives. That yields another set of Bellman equations, yet the continuous-time formulation generally yield worse numerical performance and are not directly related to this thesis.

3.3 Numerical methods for value function approximation in LMDP

[49] attempts to numerically solve for the value function in Bellman equations in LMDP setting. The first method directly discretizes the solution on dense grids(MDP) and uses the eigen solver to solve for the leading eigenpair. On the downside, discretizing on grids is unscalable to problems even with a few degrees of freedom.

Another attempt use a linear combination of gaussian base functions to express the desirability function.

$$z(\mathbf{x}) = \Sigma_i \phi_i(\mathbf{x}),\ \phi_i(\mathbf{x}) = \exp(-(\mathbf{x} - \mathbf{m_i})^T S_i (\mathbf{x} - \mathbf{m_i})/2),\ w_i \geq 0. \tag{3.16}$$

Then Bellman equations can be approximated accordingly. We can solve for the weights to minimize differences on both sides of the linear equation

$$\lambda F w = G w, \tag{3.17}$$

with both weights $\mathbf{w} = (w_1, ..., w_N)^T$ and eigenvalue λ unknown.

Weights can be solved iteratively by quadratic programming in both weights and eigenvalue. This approach approximates the desirability function and has shown effectiveness on simple problems such as car-on-the-hill. However, because we have an unknown eigenvalue λ, this approach has often failed to converge to the leading eigenpair, and the results are confusing.

Moreover, parameters in gaussian base functions are hard to tune without prior insights into both dynamical system and function approximators. A gradient descend method is

proposed to adjust such parameters, yet this may also lead to higher tendency to converge to the wrong value function.

3.4 Other works in LMDP framework

3.4.1 Stochastic maximum principle

Pontryagin's Maximum Principle characterizes locally-optimal trajectories as solutions to an ODE in deterministic problems. In stochastic problems, noise makes every trajectory dependent on its neighbors; thus, isolated trajectories are meaningless. Stochastic maximum principles are PDEs that depends on global solutions, and in numerical methods' view they are closer to the Bellman equation.

The LMDP framework provided a trajectory-based maximum principle for stochastic control[50]. The most likely trajectory of a LMDP problem in finite-horizon formulation is equivalent to finding the locally-optimal trajectory in a corresponding deterministic optimal control problem.

For linealy solvable controlled diffusions (3.9), the corresponding deterministic optimal control problem is

$$\dot{\mathbf{x}} = \mathbf{a}(\mathbf{x}) + B(\mathbf{x})\mathbf{u} \tag{3.18}$$

$$l(\mathbf{x}, \mathbf{u}) = q(\mathbf{x}) + \frac{1}{2\sigma^2}\|\mathbf{u}\|^2 + \frac{1}{2} div\, \mathbf{a}(\mathbf{x}) \tag{3.19}$$

The extra term in cost function pushes the trajectory away when the system is unstable. This work means there is almost no reason to develop a stand-alone local method for LMDP framework.

3.4.2 Compositionality of optimal control laws

In the first-exit formulations or finite-horizon formulations of LMDP, there exist a theory of compositionality showing how task-optimal controllers can be constructed from certain primitives[55]. Considering K primitive LMDP problems that share the same state cost, their final costs are $f_k(\mathbf{x})$ respectively. If the new problem has a final cost as

$$f(x) = -\log\left(\Sigma_{k=1}^{K} w_k \exp(-f_k(\mathbf{x}))\right), \tag{3.20}$$

the optimal control policy can be constructed based on a composite of optimal control policies of LMDP primitive problems without solving for a new policy. This result led to interesting applications in animation [17]

Chapter 4

MOVING LEAST SQUARES APPROXIMATION FOR LINEARLY-SOLVABLE MDP

(Most of the contents in this chapter appeared in [61])

4.1 Introduction

4.1.1 Motivation

As mentioned before, some numerical methods applicable to LMDP problems have previously been developed, in particular direct MDP discretization [48] and function approximation using Gaussian bases [49]. Discretization is useful in terms of obtaining "ground-truth" solutions in low-dimensional problems and comparing to the results of more advanced algorithms that need fine-tuning, but it is not applicable to higher-dimensional problems. Gaussian bases are promising, but they have some disadvantages. First, the resulting problem is weighted (in the average-cost setting, it is in the form $\lambda F\mathbf{w} = G\mathbf{w}$ instead of $\lambda\mathbf{w} = G\mathbf{w}$) which slows down the solver and may converge to suspicious solution. Second, when λ is also unknown, this method might converge to the wrong eigenvector. Third, positivity of the solution (which is required because we are solving for the exponent of a function) is hard to enforce without introducing inequality constraints. Fourth, Gaussians have too many shapes of parameters that need to be adjusted: it takes $O\left(n^2\right)$ scalars to specify a covariance matrix in an n-dimensional space.

4.1.2 Contribution and limitations

Our method avoids the above limitations. It is motivated by the moving least-squares (MLS) methods [6, 7, 9], also known as local regression or locally weighted scatter plot smoothing (LOWESS). The new approximation scheme developed here leads to simple eigen-problems in the form $\lambda\mathbf{w} = G\mathbf{w}$ (here $G = QP$ in (4.26)) and guarantees the positivity of the

solution by enforcing the positivity of G. It also adapts the shape of the basis functions automatically, given the set of collocation states.

On the downside, the bases are defined implicitly, as the solution to a linear system with $O(n)$ equations, thus evaluating all bases at one state involves an $O(n^3)$ operation (Cholesky decomposition). This by itself is not a major disadvantage because finding the inverse of a covariance matrix (which is needed when working with Gaussian bases) is also an $O(n^3)$ operation. However, the bases developed here need to be evaluated at more points, because the lack of an analytical expression leads to the use of cubature formulas to compute integrals.

Finally, although this methods works on many simpler problems, we frequently observed that it fails to provide good solutions for some practical problems, including with acrobot.

4.2 Moving least squares and base function construction

4.2.1 Moving least squares

Overview

Moving least squares, also called local regression (LOESS), locally weighted scatterplot smoothing (LOWESS), is developed by Cleveland and colleagues [14, 15]. MLS approximation or locally-weighted regression is a way to approximate a continuous function when data are assigned to discrete points. It builds on "classical" regression methods such as least squares and combines the linear regression with polynomials with the flexibility of nonlinear regression. It is achieved by fitting simple regressions on localized subsets of data.

The advantage of local regression is that users are not required to fit a global model to the data. Instead, users fit models to segments of data. The disadvantages of local regression are (1) additional computational cost, and (2) inefficient use of data. In addition, local regression or moving least squares cannot easily provide an analytical solution to the fitted function. This is both advantages and disadvantages, depending on details of applications.

Moving least squares have applications beyond the field of statistics. In the meshless methods for ordinary differential equations[9], moving least square's ability to construct linear base functions is exploited and used to solve certain differential equations. These

works motivate the work in this chapter.

Moving least squares

MLS is a variation of weighted least squares fitting. Compared to ordinary least squares, both the weight function and coefficients here are no longer constant but are functions of the space variable $\mathbf{x}$. In MLS, the weight function $\omega(\mathbf{x})$ is no longer a constant and is designed to reconstruct $z(\mathbf{x})$ based mainly on the neighboring nodes. In other words, weights would be higher at nearby nodes.

The reconstructed function can be expressed as

$$z^{MLS}(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T \alpha(\mathbf{x}). \tag{4.1}$$

Here, the elements of the vector $\mathbf{h}(\mathbf{x})$ are given bases. For example, in one dimensional space, $\mathbf{h}(\mathbf{x})$ can be expressed as $\mathbf{h}(\mathbf{x}) = [1, x]^T$ in linear fitting. $\mathbf{h}(\mathbf{x})$ should not be confused with the basis functions $\phi_i(\mathbf{x})$ in Section 4.3.1,4.2.2. $\mathbf{h}(\mathbf{x})$ is a by-product of MLS approximation process, while we finally use $\phi_i(\mathbf{x})$ in our solving-procedure, which will be outlined in later sections.

Elements of $\alpha(\mathbf{x})$ are the unknown coefficients of those bases $\mathbf{h}(\mathbf{x})$. Unlike ordinary least squares, here they are functions of $\mathbf{x}$ rather than constants. They are obtained by minimizing the Euclidean norm between the approximation and the given function values z_I at nodes $\mathbf{X}_I$.

$$\alpha(\mathbf{x}) = \arg\min_{\alpha} \Sigma_I \omega(\mathbf{x_I} - \mathbf{x})(\mathbf{h}(\mathbf{x_I})^T \alpha - z_I)^2. \tag{4.2}$$

Here $\omega(\mathbf{x})$ is a given weight function. It is used to make the function approximator local, in other words, fit the value at each state using only the (given) values at nearby states $\mathbf{x}_I$. The minimization process treats the vector α as a free variable at each $\mathbf{x}$.

It is not intuitive but can be easily shown that the resulting $\alpha(\mathbf{x})$ and $z^{MLS}(\mathbf{x})$ are linear in the vector $\mathbf{z}$, whose elements are the given function values z_I. The reason is simple. Because the Euclidean norm is used in (4.2), $\alpha(\mathbf{x})$ can be calculated using linear algebra operations and is linear to vector $\mathbf{z}$. This is a property shared among least squares methods minimizing errors in Euclidean norms, rather than a property specific to moving

least squares. This property holds no matter how many orders of polynomials are used in $\mathbf{h}(\mathbf{x})$.

As a result, the reconstructed function be represented as $z^{MLS}(\mathbf{x}) = \Sigma_I \tilde{\phi}_I(\mathbf{x}) z_I$, making the MLS process a natural candidate for the construction of basis functions.

4.2.2 Construction of basis functions

Now we begin to construct basis functions tailored for solving our problem. In order to solve the leading eigenpair easier, we seek basis functions $\phi_i(x)$ that satisfy the following conditions:

(a) $\phi_j(x_i) = \begin{cases} 1, & \text{when } i = j, \\ 0, & \text{when } i \neq j. \end{cases}$

(b) $\phi_i(\mathbf{x}) \geq 0$ everywhere.

(c) $\Sigma_i \phi_i(x) = 1$.

(a) will make the resulting discretized eigen problem easier to solve. (b) will ensure the non-negativity of our approximated desirability function $z(x)$. (c) guarantees consistency, meaning that if all data indicate that we have a constant function, the approximator will recover a constant function. These properties come for free when using the least-squares bases and $\mathbf{h}(\mathbf{x})$ contains a constant.

To achieve this goal, we first use MLS to generate potential functions $\tilde{\phi}_i(\mathbf{x})$. Then, we truncate and renormalize them to construct basis functions $\phi_i(\mathbf{x})$.

In order to achieve sparseness and computational efficiency, $z(\mathbf{x})$ will only depend on the values at the K-nearest neighbor nodes (also collocation states). Here K is a manually tuned parameter that determines the trade-off between computer time and smoothness of the approximator. Note that very large values of K are undesirable even if we ignore computer time, because they induce too much coupling and effectively decrease the approximating power of the method. Thus $\phi_i(\mathbf{x}) = 0$ if $\mathbf{x}_i$ is not among the K nearest nodes to $\mathbf{x}$. In the following parts, we will use $\mathbf{x}_{(1)}, \mathbf{x}_{(2)} \cdots, \mathbf{x}_{(K)}$ to represent the K nearest neighboring

nodes of $\mathbf{x}$. $\mathbf{x}_{(1)}, \mathbf{x}_{(2)} \cdots, \mathbf{x}_{(K)}$, are sorted in increasing order by distance from $\mathbf{x}$, so we have $\|\mathbf{x} - \mathbf{x}_{(1)}\| \leq \|\mathbf{x} - \mathbf{x}_{(2)}\| \leq \cdots \leq \|\mathbf{x} - \mathbf{x}_{(K)}\|$.

Constraint (a) can be easily obtained by enforcing a linear constraint $\mathbf{h}(\mathbf{x_1})^T\alpha = z_{(1)}$. This will not significantly affect the computational efficiency, because it can be done by introducing a Lagrange multiplier.

To maintain scalability to higher dimensional systems, we use linear basis

$$\mathbf{h}(\mathbf{x}) = [1, x_1, \cdots, x_n]^T. \tag{4.3}$$

Here $x_i, i = 1, \cdots, n$ are elements of $\mathbf{x} \in \Re^n$. Even when the number n of dimensions increases, the time complexity of the algorithm in this step will not increase significantly.

We would like to use a weight function that can automatically adjust to areas with different densities. We currently use $\omega_i(\mathbf{x}) = \mu_i(\mathbf{x})^2$, with

$$\begin{aligned} \mu_i &= C_\mu\Big(\frac{1}{\|\mathbf{x} - \mathbf{x}_{(i)}\| - \|\mathbf{x} - \mathbf{x}_{(1)}\| + \epsilon_\mu} \\ &\quad - \frac{1}{\|\mathbf{x} - \mathbf{x}_{(K)}\| - \|\mathbf{x} - \mathbf{x}_{(1)}\| + \epsilon_\mu}\Big). \end{aligned} \tag{4.4}$$

Here, $i = 2, \cdots, K$. C_μ is a normalization constant adjusted to yield $\Sigma_i \mu_i = 1$, and ϵ_μ is a small constant used to avoid numerical difficulties.

It is possible to enforce the non-negativity constraints in (4.5) using quadratic programming; however, this is slower than solving a linear system. Instead we simply set any negative values to 0 and renormalize to ensure $\Sigma_i \phi_i(\mathbf{x}) = 1$.

To recapitulate, we define $\alpha(\mathbf{x})$ as the solution to the optimization problem

$$\begin{aligned} &\alpha(\mathbf{x}) = \arg\min_\alpha \Sigma_{I=2}^K \omega(\mathbf{x_I} - \mathbf{x})(\mathbf{h}(\mathbf{x_I})^T\alpha - z_{(I)})^2 \\ &\text{s.t.} \quad \mathbf{h}(\mathbf{x_1})^T\alpha = z_{(1)} \end{aligned} \tag{4.5}$$

to obtain $\tilde{\phi}_i(\mathbf{x})$ in

$$z^{MLS}(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T\alpha(\mathbf{x}) = \Sigma_i \tilde{\phi}_i(\mathbf{x}) z_i \tag{4.6}$$

This process is done without knowing z_i or explicitly solving for $\alpha(\mathbf{x})$. Instead, (4.5) is converted into linear equations, and by comparison with (4.6), an algorithm for finding $\tilde{\phi}_i(\mathbf{x})$ is obtained. Details of this process are shown below.

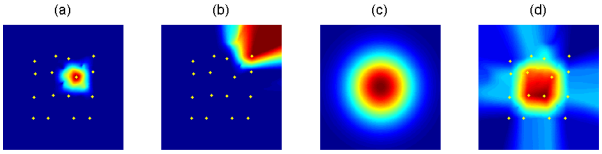


Figure 4.1: (a,b) show example basis functions, (c) shows a Gaussian function and (d) shows the fit to the Gaussian function. Dots represent nodes.

After setting any negative values to 0 and renormalizing, the solution is given by

$$\phi_i(\mathbf{x}) = \begin{cases} C_\phi(\mathbf{x})\tilde{\phi}_i(\mathbf{x}), & \text{if } \tilde{\phi}_i(\mathbf{x}) \geqslant 0, \\ 0, & \text{if } \tilde{\phi}_i(\mathbf{x}) < 0. \end{cases} \tag{4.7}$$

Note that $C_\phi(\mathbf{x})$ is adjusted to ensure $\Sigma_i \phi_i(\mathbf{x}) = 1$

Fig.4.1 gives an example of what these basis functions look like. There are 16 nodes on the plane. We can see that the basis functions have an irregular shape, but nevertheless their support is concentrated around the node. In Fig.4.1(a) we see a basis that stretches to the right, because the right side is "empty". In Fig.1(b) we see a base stretching out to infinity. Suppose the true $z(\mathbf{x})$ is the Gaussian shown in Fig.1(c). The function $z^{MLS}(\mathbf{x})$ fitted by our approximator is shown in Fig.1(d). The reconstructed result is not entirely smooth but captures the shape of the original function.

In summary, we used an MLS method with truncation to construct basis functions whose shape is automatically adapted to the set of collocation states. This method can be viewed as an interpolation scheme, but is modified to ensure a fast yet reliable algorithm to solve the eigenfunction problem.

Computational details of step (4.5) (4.6)

In the main text, we omitted the details about how to obtain the basis functions before truncation and renormalization, i.e., $\tilde{\phi}_i(\mathbf{x})$ in (4.5) and (4.6). The optimization problem (4.5) is solved via a linear equation, the same way as in the ordinary least square methods.

Let us start from the optimization problem (4.5)

$$\begin{aligned}&\alpha(\mathbf{x}) = \arg\min_{\alpha} \Sigma_{I=2}^{K} \omega(\mathbf{x_I} - \mathbf{x})(\mathbf{h}(\mathbf{x_I})^T\alpha - z_{(I)})^2\\ &\text{s.t.} \quad \mathbf{h}(\mathbf{x_1})^T\alpha = z_{(1)}.\end{aligned}$$

Using a Lagrange multiplier $\tilde{\lambda}$, we can rewrite the optimization problem as

$$\begin{aligned}\alpha(\mathbf{x}) \quad = \quad & \arg\min_{\alpha} \Sigma_{I=2}^{K} \omega(\mathbf{x_I} - \mathbf{x})(\mathbf{h}(\mathbf{x_I})^T\alpha - z_{(I)})^2 \\ & +\tilde{\lambda}(\mathbf{h}(\mathbf{x_1})^T\alpha - z_{(1)}).\end{aligned} \tag{4.8}$$

This is equivalent to solving for $\alpha, \tilde{\lambda}$ in the linear equation

$$M_1(\mathbf{x})[\alpha(\mathbf{x})^T, \tilde{\lambda}]^T - M_2(\mathbf{x})\mathbf{z} = 0, \tag{4.9}$$

where

$$\mathbf{z} = [z_{(1)}, \cdots, z_{(K)}]^T \tag{4.10}$$

$$M_1 = \begin{pmatrix} H_2^T W H_2 & H_1^T \\ H_1 & 0 \end{pmatrix}, \tag{4.11}$$

$$M_2 = \begin{pmatrix} \mathbf{0} & H_2^T W \\ 1 & \mathbf{0} \end{pmatrix}, \tag{4.12}$$

$$H_1 = \mathbf{h}(\mathbf{x_1})^T = \begin{pmatrix} 1 & \mathbf{x}_1^T \end{pmatrix}, \tag{4.13}$$

$$H_2 = \begin{pmatrix} \mathbf{h}(\mathbf{x_2})^T \\ \vdots \\ \mathbf{h}(\mathbf{x_K})^T \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{x}_2^T \\ \vdots & \vdots \\ 1 & \mathbf{x}_K^T \end{pmatrix}, \tag{4.14}$$

and W is a diagonal matrix representing weights,

$$W = \begin{pmatrix} \omega(x - x_2) & 0 & \cdots & 0 \\ 0 & \omega(x - x_3) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \omega(x - x_K) \end{pmatrix}. \tag{4.15}$$

Therefore, from (4.9), we have

$$[\alpha(\mathbf{x})^T, \tilde{\lambda}]^T = M_1^{-1}(\mathbf{x})M_2(\mathbf{x})\mathbf{z} \tag{4.16}$$

Comparing with (4.6)

$$z^{MLS}(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T\alpha(\mathbf{x}) = \Sigma_i\tilde{\phi}_i(\mathbf{x})z_i,$$

we have

$$\left[\tilde{\phi}_{(1)}(\mathbf{x}), \cdots, \tilde{\phi}_{(K)}(\mathbf{x})\right] = \mathbf{h}(\mathbf{x})^T \left(M_1(\mathbf{x})^{-1}M_2(\mathbf{x})\right)_{\mathrm{n+1}}, \tag{4.17}$$

where$(\cdot)_{\mathrm{n+1}}$ denotes the operation that takes the first $n+1$ rows of a matrix (because we ignore the Lagrange multiplier), and n is the number of dimensions in the dynamical system.

(4.17) gives the method for how to construct basis functions from the optimization problem (4.5). Since M_1 matrix is a $(n+2)\times(n+2)$ matrix, when we perform $M_1^{-1}M_2$ with Cholesky decomposition, the computational complexity of (4.17) is $O(n^3)$.

4.3 Solution method

In this section we describe how we solve for the desirability function using the above MLS function approximator. First, we show how to discretize the LMDP problem with MLS basis functions. Then, we show how to obtain the discretized linear operator. After that, we describe how to determine the weights of the basis functions. Next, we describe how the (approximately) optimal control law is found from the approximate desirability function. Finally, we describe a procedure for adding collocations states to improve the solution in critical regions (i.e., regions that are visited often under the resulting control law but do not yet contain enough collocation states). Finally, we discuss the cases of settings other than average-cost setting.

4.3.1 Discretizing LMDPs with basis functions

Let ϕ_j denote (defined in Section 4.2.2) basis functions with weights w_j, $j = 1, \cdots, N$. Then the desirability function $z(\mathbf{x})$ can be approximated by

$$z(\mathbf{x}) = \Sigma_{j=1}^{N} w_j\phi_j(\mathbf{x}). \tag{4.18}$$

From (3.15), for the average-cost setting, we aim to solve the following eigenfunction problem with $\lambda = \exp(-hc)$

$$\lambda z(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z](\mathbf{x}). \tag{4.19}$$

Since $\mathcal{G}[z]$ is a linear operator, we have

$$\mathcal{G}[z](\mathbf{x}) = \Sigma_{j=1}^{N} w_j \mathcal{G}[\phi_j](\mathbf{x}). \tag{4.20}$$

Therefore if (4.18) holds, (4.19) becomes

$$\lambda \Sigma_{j=1}^{N} w_j \phi_j(\mathbf{x}) = \exp(-hq(\mathbf{x}))\Sigma_{j=1}^{N} w_j \mathcal{G}[\phi_j]. \tag{4.21}$$

We will solve this equation approximately by introducing collocation points $\mathbf{x_i}, i = 1, \cdots, M$, enforcing the equation at those points. This yields the generalized eigenvalue problem

$$\lambda F\mathbf{w} = QP\mathbf{w} \tag{4.22}$$

Here F and P are $M \times N$ matrices with entries

$$\begin{aligned} &F_{ij} = \phi_j(\mathbf{x}_i), P_{ij} = \mathcal{G}[\phi_j](\mathbf{x}_i), \\ &i = 1, \cdots, M, j = 1, \cdots, N, \end{aligned} \tag{4.23}$$

Q is a $M \times M$ diagonal matrix with diagonal entries $Q_{ii} = \exp(-hq(\mathbf{x_i}))$, $\mathbf{w}$ is a vector of all w_j.

The above outline applies generally to many basis function approximators that are linear in the unknown parameters w_j. For example, in [49], the functions ϕ_j are Gaussians.

Here we will design bases and select collocation states that satisfy the following conditions.

1. Equal number of collocation points and basis functions, i.e., $N = M$.

2. F in (4.22) is the identity matrix, or equivalently

$$\phi_j(x_i) = \begin{cases} 1, & \text{when } i = j, \\ 0, & \text{when } i \neq j. \end{cases} \tag{4.24}$$

3. The basis functions are everywhere non-negative, so

$$\phi_j(x_i) \geq 0 \tag{4.25}$$

There are two reasons for these restrictions. First, when N is large and F is the identity matrix, we can avoid matrix factorization and take advantage of sparsity. Second, $z(\mathbf{x}) = \exp(-v(\mathbf{x})) > 0$ is difficult to enforce if the bases can be negative.

Thus in the average-cost setting, the resulting discretized problem becomes

$$\lambda \mathbf{w} = QP\mathbf{w} \tag{4.26}$$

4.3.2 Computing the discretized integral operator

Starting from our discretized problem, as mentioned before, we choose basis functions as (4.7) in Section 4.2.2, and we use the nodes themselves as collocation points. Then we have

$$\begin{aligned} P_{ij} &= \mathcal{G}[\phi_j](\mathbf{x}_i) \\ &= \int p(\mathbf{x}'|\mathbf{x})\phi_j(\mathbf{x}')d\mathbf{x}'. \end{aligned} \tag{4.27}$$

Because $p(\mathbf{x}'|\mathbf{x})$ is a Gaussian, this integral can be approximated using cubature formulas. Cubature formulas can be thought of as deterministic sampling, designed to match as many moments of the Gaussian distribution as possible.

Calculating the expectation of an arbitrary function under a normal distribution using the cubature method

We want to evaluate the expectation of a function under a normal distribution. This is equivalent to calculating the integral of the product of an arbitrary function and a normal distribution numerically. Here we used the cubature scheme [29].

The goal is to calculate

$$E[g(\mathbf{x})] = \int g(\mathbf{y})p(\mathbf{y}|\mathbf{x})d\mathbf{y}. \tag{4.28}$$

As in (3.11),

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{x} + h\mathbf{a}(\mathbf{x}), h\Sigma(\mathbf{x})), h\Sigma(\mathbf{x}) = CC^T, \tag{4.29}$$

where $C = \sigma B(\mathbf{x})$ for continuous formulation.

We need a variable transformation

$$\mathbf{y} = C\sqrt{2}\xi + \mathbf{x} + h\mathbf{a}(\mathbf{x}) \tag{4.30}$$

to convert the integral into the standard form of cubature formulas.

$$\begin{aligned} E[g(\mathbf{x})] &= C_0 \int f(\xi) exp(-\xi^T \xi) d\xi, \\ C_0 = \frac{1}{(\pi)^{N/2}}, f(\xi) &= g(C\sqrt{2}\xi + \mathbf{x} + h\mathbf{a}(\mathbf{x})). \end{aligned} \tag{4.31}$$

Now we can apply the cubature scheme in [29], which uses the following formula to approximate the integral.

$$\int f(\xi) d\xi \approx Q[f] = \Sigma_{j=1}^{N} w_j f(\xi^j), \tag{4.32}$$

where w_j are weights and ξ^j are points.

Several choices of weights and points exist, and it is not clear which cubature formula is optimal for our problems. We observed that formula (4.33) in [29] works better in some test problems. This is a degree 5 formula with $2n_C^2 + 1$ points, n_C is the number of columns of the $C(\mathbf{x})$ matrix in (4.29), which is the number of dimensions in control space (for under-actuated systems this number may be significantly smaller than the number of dimensions in the system). Here, "full sym." means all possible index permutations and reflections.

$$\begin{aligned} Q[f] \quad = \quad & \frac{n^2 - 7n + 18}{18} \pi^{n/2} f(\mathbf{0}) \\ & + \frac{4 - n}{18} \pi^{n/2} \Sigma_{full\ sym.} f(\sqrt{3/2}, 0, .., 0) \\ & + \frac{1}{36} \pi^{n/2} \Sigma_{full\ sym.} f(\sqrt{3/2}, \sqrt{3/2}, .., 0). \end{aligned} \tag{4.33}$$

In summary, to calculate the integral (4.28) approximately, we used (4.31) and (4.29); therefore, the following formula represents this process.

$$E[g(\mathbf{x})] = \int g(\mathbf{y}) p(\mathbf{y}|\mathbf{x}) d\mathbf{y} = C_0 \Sigma_{j=1}^{N} w_j f(\xi^j). \tag{4.34}$$

4.3.3 Solving for the desirability function

Recall that in the average-cost setting, we need to solve for the leading eigenvalue and eigenvector of a matrix. Many numerical methods are available for solving this kind of problem [57]. Our current implementation uses the classic power iteration method. The reasons for choosing this algorithm are: (1) it does not require matrix factorization, which would be very slow when we have a large amount of nodes; (2) the result is always positive as long as the initialization is positive and all elements of P are nonnegative (which property P satisfies by design). Thus the algorithm is:

(0) Let $\mathbf{w}^0$ be an initial guess

(1) $\mathbf{w}^n = \frac{QP\mathbf{w}^{n-1}}{\|QP\mathbf{w}^{n-1}\|}$

(2) Repeat step (1) until $\|\mathbf{w}^n - \mathbf{w}^{n-1}\| < \epsilon$

After this step, we obtain an approximate cost-to-go function for the optimal control problem. We know that the eigenfunction corresponding to the leading eigenvalue of the linear operator $\mathcal{G}$ is unique. Thus, the discretization QP is also likely to have a unique leading eigenvector, unless the placement of nodes is pathological (which we have not observed in practice). The speed of convergence is governed by the difference between the top two eigenvalues of QP. Alternatively, because an iteration can be viewed as running backwards with step h in an finite horizon formulation, the speed of convergence corresponds to the mixing rate of the Markov chain. In practice we have found this algorithm to converge very quickly (in a few iterations) in our test problems. The time needed to run power iteration is a small fraction of the overall CPU time; most of the CPU time is spent in constructing the discretized operator.

In other settings, the discretized problem is no longer an eigenvector problem. However, iterative algorithms without matrix factorization are still available.

4.3.4 Finding the optimal control u^*

Recall that the optimal control law (in terms of probability densities) is given by (3.3)

$$u^*(\mathbf{x}'|\mathbf{x}) = \frac{p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')}{\mathcal{G}[z](\mathbf{x})}.$$

The above probability distribution function is known, and its mean can be calculated. In a linearly-solvable diffusion process, $u^*(\mathbf{x}'|\mathbf{x})$ should be a gaussian distribution whose mean moved $hB(\mathbf{x})\mathbf{u}^*$ from the mean of passive dynamics, thus, to calculate a control vector $\mathbf{u}^*$, we can solve the following equation:

$$\mathbf{x} + h\mathbf{a}(\mathbf{x}) + hB(\mathbf{x})\mathbf{u}^* = E_{\mathbf{x}'\sim u^*(\cdot|\mathbf{x})}[\mathbf{x}'] \tag{4.35}$$

From (3.3), we have

$$\begin{aligned} E_{\mathbf{x}'\sim u^*(\cdot|\mathbf{x})}[\mathbf{x}'] &= \int u^*(\mathbf{x}'|\mathbf{x})\mathbf{x}'d\mathbf{x}' \\ &= \frac{\int p(\mathbf{x}'|\mathbf{x})\mathbf{x}'z(\mathbf{x}')d\mathbf{x}'}{\int p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')d\mathbf{x}'}. \end{aligned} \tag{4.36}$$

Both the numerator and the denominator of the right-hand side of (4.36) are the integral of a function multiplied by a normal distribution. Again, we can use the cubature formula to calculate them approximately. By this method, both the numerator and denominator can be expressed as a weighted sum of $z(\xi_i)$, which are the z values at the cubature points.

Numerical errors introduced by (3.3)(4.36) may be problematic when the denominator is very small. However, that happens when $z(\mathbf{x})$ is very small, i.e. the cost-to-go function $v(\mathbf{x})$ is very large, or $\mathbf{x}$ is far away from the region of interest. In such a case we cannot trust the above u^*; instead we replace it with a default linear feedback control law. Another issue is that the cubature formula restricts the left-hand side of (4.36), so that the mean of $u^*(\mathbf{x}'|\mathbf{x})$) lies in the convex hull of the cubature sampling points. This effectively restricts the magnitude of the control signal – which can be good or bad in practice.

4.3.5 Adapting the nodes

Overview

In previous sections, we described how our method works once the nodes/collocation states are chosen. This choice affects performance significantly and needs to be done in a way

suitable for the problem at hand. We have developed an automated procedure for generating problem-specific node placements that optimize performance. Note that the state space is usually very large, so our nodes can only cover a small fraction of it. Ideally the part that is covered will correspond to the region where the optimally controlled system spends most of its time (i.e., where the good states are found). Thus we use an adaptive procedure, placing new nodes in regions that are visited most often under the control law obtained on the previous iteration of the algorithm. The details are as follows.

The method only adds nodes, with the restriction that every new node must be sufficiently removed from any existing node. If nodes are further restricted within a predefined volume, then this method is guaranteed to terminate in a finite number of iterations. After solving for the optimal control law with the current selection of nodes, we generate prospective nodes based on a stochastic simulation starting from the current nodes (or from given initial states). Meanwhile we also introduce random perturbations to the current nodes. Note that regions with lower $z(\mathbf{x})$, or equivalently higher cost-to-go, will end up with smaller node density under this scheme. We also impose a heuristic restriction to avoid generating prospective nodes in regions where it is impossible to add new nodes (because the density is already too high). The distance between nodes should approximately match the characteristic distance of the system determined by $h\mathbf{a}(\mathbf{x})$ and $\sqrt{h}\sigma B(\mathbf{x})$, which are the magnitude of passive dynamics and transition probabilities. This feature means the time step should be big enough to solve the problem with a small number of nodes.

In summary, we used a heuristic strategy to place nodes in regions that are visited more often under the optimal control law.

Details

Several general rules apply. (1) All nodes are classified into normal nodes and "pioneer" nodes. New nodes can only be generated from "pioneer" nodes. (2) All new nodes should lie sufficiently far away from old nodes. This lower bound of distance is basically a constant, but when an old node has a very small z(x) value, it would "occupy" a larger space to avoid creating new nodes.

(iii) The user has to specify a range for states and assume that the range is large enough to cover the region of interest. (iv) Initially, all nodes are pioneer nodes.

(1)If any "pioneer" nodes lie outside of the specified range, they would turn to normal nodes.
(2)If any "pioneer" nodes satisfy $z(x) < \theta_n$ for several iterations, where θ_n is a manually defined parameter, then they would turn to normal nodes.
(3)If any normal nodes satisfy $z(x) < \theta_c$ for several times, where θ_c is a manually defined parameter, then they would occupy a larger region to avoid new nodes to be created.
(4)If any "pioneer" nodes fail to generate new nodes for several iterations, they would turn to normal nodes.
(5)Generate new nodes from "pioneer" nodes. They may come from both dynamics and random perturbation.
(optional) Run dynamical simulation from known initial points, and use the resulting trajectory to generate nodes

Here, (1) is used to avoid exploring less interesting region. (2)and (3) are used to avoid exploring the region where the cost-to-go function is high. (4) is used to avoid exploring where the nodes is dense enough. In (5), we generate new nodes both from calculated optimal control and random perturbation, because the calculated optimal control law might not be accurate. If some information about the system is provided, nodes may be generated from dynamical simulation.

This method would terminate in finite iterations. That is because only those nodes in the specific region can be "pioneer" nodes, and the space in this region is limited, then finally all of those "pioneer" nodes would turn to normal, thus stopping iterations. In practice, our method rarely terminates in this way, because filling a high dimensional requires a tremendous number of nodes. On the other hand, we lack theoretical results about whether the resulting optimal control law will converge to the true solution.

4.3.6 Discretizing and solve for LMDP in other settings

This chapter mainly addresses the LMDP problem in the average-cost setting. Nevertheless, the proposed method is applicable to other settings as well. Below we describe the Bellman equation for other settings, their discretized versions, and the corresponding algorithms. Table 4.1 is provided to summarize the results.

Finite horizon

In the finite horizon setting, costs are accumulated from time 0 to time T. $q_F(\mathbf{x})$ represents the final cost evaluated at time T.

The Bellman equation for the finite-horizon setting[48] is

$$z_t(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z_{t+h}(\mathbf{x})], \tag{4.37}$$

$$z_T(\mathbf{x}) = \exp(-q_F(\mathbf{x})). \tag{4.38}$$

If we represent the desirability function with the same set of basis functions at each time step,

$$z_t(\mathbf{x}) = \Sigma_{i=1}^{N} w_{t,i}\phi_i(\mathbf{x}). \tag{4.39}$$

Defining a column vector $\mathbf{w_t}$ with elements $w_{t,i}$, (4.37) becomes

$$\mathbf{w_t} = QP\mathbf{w_{t+h}}. \tag{4.40}$$

Due to the nature of our basis functions, we can use the following approximation. ($\approx$ would become $=$ if $\mathbf{x} = \mathbf{x_i}$.)

$$z(\mathbf{x}) \approx \Sigma_{i=1}^{N} z(\mathbf{x_i})\phi_i(\mathbf{x}). \tag{4.41}$$

Applying (4.41) to (4.38), we will have

$$\mathbf{w_T} = \exp(-q_F(\mathbf{x_i})). \tag{4.42}$$

Using (4.40) iteratively for T/h times from (4.42), we would obtain the approximated desirability function at time 0

$$z_0(\mathbf{x}) \approx \Sigma_{i=1}^{N} w_{0,i}\,\phi_i(\mathbf{x}). \tag{4.43}$$

Infinite horizon, discounted cost

In the discounted-cost setting, costs are accumulated from time 0 to infinity; however future costs decay exponentially as $\exp(-t/\tau)$. Let $\alpha = \exp(-h/\tau) < 1$ represent the corresponding discretized discount factor, where h is the time step.

The Bellman equation for the discounted-cost setting[48] is

$$z(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z^{\alpha}(\mathbf{x})]. \tag{4.44}$$

We will use our basis functions to approximate the desirability function

$$z(\mathbf{x}) = \Sigma_{i=1}^{N} w_i \phi_i(\mathbf{x}). \tag{4.45}$$

Due to the nature of our basis functions, we can use the following approximation. ($\approx$ would become $=$ if $\mathbf{x} = \mathbf{x_i}$.)

$$z^{\alpha}(\mathbf{x}) \approx \Sigma_{i=1}^{N} w_i^{\alpha} \phi_i(\mathbf{x}). \tag{4.46}$$

Thus the problem is discretized as

$$\mathbf{w} = QP\mathbf{w}^{\alpha}. \tag{4.47}$$

Here $\mathbf{w}$ is a column vector with elements w_i.

There exist alternative ways to solve the equation (4.47), but here, we will focus on the iterative methods to avoid matrix factorization.

One may naturally try to use the following iterative formula (4.48) to solve this problem.

$$\mathbf{w}^{k+1} = QP(\mathbf{w}^{k})^{\alpha}. \tag{4.48}$$

However, this may not work well when α is close to 1. There are two possibilities: (1) the decay of cost is slow; (2) the time step h is too small. In these cases, elements of $\mathbf{w}$ would become undistinguishable from 0, and the rate of convergence would become very slow. This situation is not so important because the results are close to what the average-cost setting may yield. However, if one wants to overcome the numerical difficulty, the following algorithm would work.

$$\begin{aligned} \tilde{\mathbf{w}}^{k+1} &= \tilde{\lambda}_k QP(\tilde{\mathbf{w}}^{k})^{\alpha}, \tilde{\lambda}_k = \|QP(\tilde{\mathbf{w}}^{k})^{\alpha}\|^{-1} \\ \mathbf{w}^{k} &= (\tilde{\lambda}^{k})^{1/(\alpha-1)} \tilde{\mathbf{w}}^{k}. \end{aligned} \tag{4.49}$$

This is similar to the power iteration method in the an average-cost case. In our simulation, (4.48) performs no worse than the simple power iteration in average-cost setting. On the other hand, if we need only the optimal control law, because it is invariant when the desirability function is multiplied by a constant, there is no need to evaluate $\mathbf{w}^{k}$ explicitly.

First exit

In the first-exit setting, costs are accumulated from time 0 to infinity; however, accumulation would stop when the system reaches a terminal state. Here $\mathcal{T} \in \mathcal{R}^n$ represents the subset of terminal states, and $\mathcal{N} \in \mathcal{R}^n$ represents the subset of non-terminal states. $q_{\mathcal{T}}(\mathbf{x})$ represents the cost at terminal states.

The Bellman equation would become a boundary value problem[48]

$$z(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z(\mathbf{x})], \tag{4.50}$$

$$z(\mathbf{x}) = \exp(-q_{\mathcal{T}}(\mathbf{x})), x \in \mathcal{T}. \tag{4.51}$$

(4.50) can be further rewritten as

$$\exp(hq(\mathbf{x}))z(\mathbf{x}) = \mathcal{G}_{\mathcal{N}}[z(\mathbf{x})] + \mathcal{G}_{\mathcal{T}}[\exp(-q_{\mathcal{T}}(\mathbf{x}))] \tag{4.52}$$

$$x \in \mathcal{N}$$

Where operators are defined as integrals on finite regions

$$\mathcal{G}_{\mathcal{N}}[z(\mathbf{x})] = \int_{\mathbf{x}\in\mathcal{N}} z(\mathbf{x}')p(\mathbf{x}'|\mathbf{x})d\mathbf{x}', \tag{4.53}$$

$$\mathcal{G}_{\mathcal{T}}[z(\mathbf{x})] = \int_{\mathbf{x}\in\mathcal{T}} z(\mathbf{x}')p(\mathbf{x}'|\mathbf{x})d\mathbf{x}'. \tag{4.54}$$

We can construct a matrix $P_{\mathcal{N}}$ with integrals only at non-terminal states

$$P_{\mathcal{N},ij} = \mathcal{G}_{\mathcal{N}}[\phi_j](\mathbf{x}_i), \qquad i = 1, \cdots, N, j = 1, \cdots, N. \tag{4.55}$$

and a vector $\mathbf{r}$ with elements as $r_i = \mathcal{G}_{\mathcal{T}}[\exp(-q_{\mathcal{T}}(\mathbf{x}_i))]$. $\int_{\mathbf{x}\in\mathcal{N}}$ is usually not different than the integral on the entire space, because the bases are localized, and terminal states are normally limited in a small region. Therefore, most $P_{\mathcal{N},ij}$ can be evaluated with the cubature formula. Otherwise, if the collocations state is close to the terminal states, discretization in grids is needed to evaluate $P_{\mathcal{N},ij}$.

If we use basis functions to approximate the desirability function (4.45), we will obtain the discretized formula, which is a linear equation.

$$\mathbf{w} = Q(P_{\mathcal{N}}\mathbf{w} + \mathbf{r}) \tag{4.56}$$

Many algorithms are available for solving this kind of problem (backslash in MATLAB, for example). One iterative method will take the form

$$\mathbf{w}^{k+1} = QP_{\mathcal{N}}\mathbf{w}^k + Q\mathbf{r} \tag{4.57}$$

This algorithm would terminate when the modulus of the leading eigenvalue of $QP_{\mathcal{N}}$ is smaller than 1, which is true when $q(\mathbf{x}) > 0$ everywhere, meaning that the first exit problem is well-defined.

Summary

Table 4.1 summarizes the Bellman equations, their discretized correspondences and algorithms. Some details are omitted.

In summary, it is possible to use the MLS approximation to solve the LMDP problems posed in all four settings[48]. Iterative algorithms are available, so no matrix factorization is needed when matrix P ($P_{\mathcal{N}}$ for first-exit) is constructed. This feature provides efficiency when dealing with a large number of base functions.

4.4 Numerical results

In this section, we present numerical results. First we show that the results given by MLS approximation are meaningful on simple test problems. Then we show that the method scales to high-dimensional systems. After that, we will compare them with other methods. Finally we will discuss limitations of this method.

4.4.1 Test problems, solution, and dynamical simulation

Here we use MLS approximation to solve our test problems. We focus on the desirability function $z(\mathbf{x})$, cost-to-go function $v(\mathbf{x})$, optimal control law $u^*(\mathbf{x})$, and dynamical simulations based on $u^*(\mathbf{x})$. When possible, we compare these results with dense MDP discretization[49]. Test problems are formulated as linearly-solvable controlled diffusions as in Section 3.2.

Table 4.1:

Bellman Equation	
Setting	Bellman Equation
average	$z(\mathbf{x}) = \exp(hc - hq(\mathbf{x}))\mathcal{G}[z(\mathbf{x})]$
finite	$z_t(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z_{t+h}(\mathbf{x})]$
discount	$z(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z^\alpha(\mathbf{x})]$
first exit	$z(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z(\mathbf{x})]$

Discretized problem and algorithms		
Setting	Discretized	Algorithm
average	$\mathbf{w} = \lambda QP\mathbf{w}$	$\mathbf{w}^{k+1} = QP\mathbf{w}^k/(\|QP\mathbf{w}^k\|)$
finite	$\mathbf{w_t} = QP\mathbf{w_{t+h}}$	$\mathbf{w_t} = QP\mathbf{w_{t+h}}$
discount	$\mathbf{w} = QP\mathbf{w}^\alpha$	$\mathbf{w}^{k+1} = QP(\mathbf{w}^k)^\alpha$
first exit	$\mathbf{w} = Q(P_{\mathcal{N}}\mathbf{w} + \mathbf{r})$	$\mathbf{w}^{k+1} = QP_{\mathcal{N}}\mathbf{w}^k + Q\mathbf{r}$

Example 1: Car-on-the-hill

This test problem is adapted from [48]. (More details in the appendix.) It has a 2D state space (position and velocity) and 1D control space. This dynamical system simulates a point mass (a car) moving along a curve (inverted Gaussian) in the presence of gravity. The control signal is the force acting in a tangential direction. One interesting property of this model is that the continuous dynamics are augmented with the following rule. When the car hits the "walls" at x_{min} or x_{max} , its speed becomes 0. Such a discontinuity cannot be captured by the diffusion model (3.9), yet it can easily be captured by our LMDP [49]. The reason for constructing a model with collisions is that we hope our methods will work for more complex tasks such as locomotion and hand manipulation, where contact phenomena and discontinuity are essential.

For the average-cost setting, we design a cost function $q(\mathbf{x})$ (Fig.4.2(b)) which encourages the car to pass repeatedly via two targets with non-zero desired velocity – resulting in a

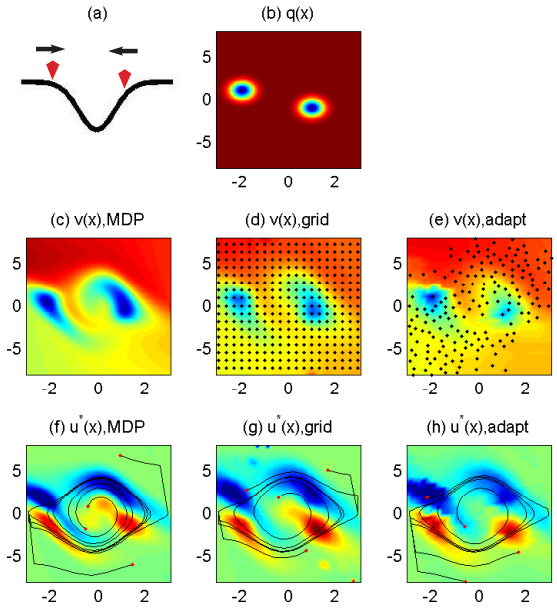


Figure 4.2: (a) illustrates the model and (b) shows the state cost function $q(\mathbf{x})$. (c)-(e) (the second row) show the cost-to-go function $v(\mathbf{x}) = -\log(z(\mathbf{x}))$, while (f)-(h) (the third row) show the optimal control law obtained. In (f-h), dynamical simulations are shown by black curves, while their initial states are shown by red dots. (c),(f) show the results from the MDP method with a 151×151 grid. (d),(g) shows the results from the MLS approximation with 21×21 nodes on a regular grid. (e,h) shows the results from the MLS approximation with 189 nodes generated using our adaptive scheme. Black dots in (d,e) represent nodes. In (b)-(h), the horizontal axis represents position x_p and the vertial axis represents velocity x_v.

limit cycle behavior. As shown in Fig.4.2(c-h), the cost-go function and optimal control law are consistent with the results of dense MDP discretization.

We can also use this dynamical system to define other optimal control problems: discounted, first exit, and finite horizon. For the discounted setting, the cost function is the same as before. For the first exit and finite horizon settings, the goal (encoded as a final cost) is to park the car at horizontal position 2.4; in that case the running state cost $q(\mathbf{x})$ is constant. As shown in Fig.4.3, Fig.4.4 and Fig.4.5, the results are again similar to those obtained by dense MDP discretization.

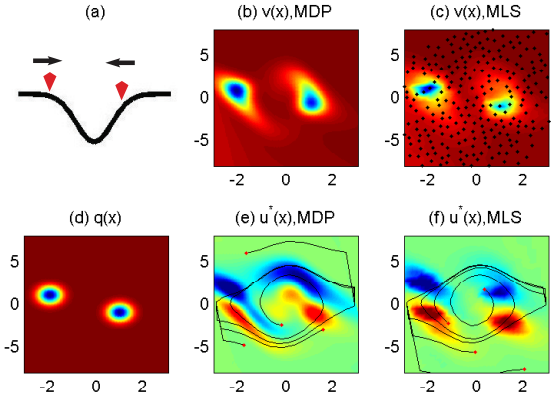


Figure 4.3: (a) illustrates the model and (b) shows the state cost function $q(\mathbf{x})$. (b-c) show the cost-to-go function $v(\mathbf{x}) = -\log(z(\mathbf{x}))$. (e-f) show the optimal control law obtained. In (e-f), dynamical simulations are shown by black curves, while their initial states are shown by red dots. (b),(e) show the results from the MDP method with a 151×151 grid. (c),(f) show the results from the MLS approximation with 251 nodes generated by our adaptive scheme. Black dots in (c) represent nodes. Compared with the average-cost setting, here the optimal control law seems to take advantage of walls (hit the wall, then keep going). MLS with the adaptive scheme is able to capture this property. In (b)-(f), the horizontal axis represents position x_p and the vertial axis represents velocity x_v.

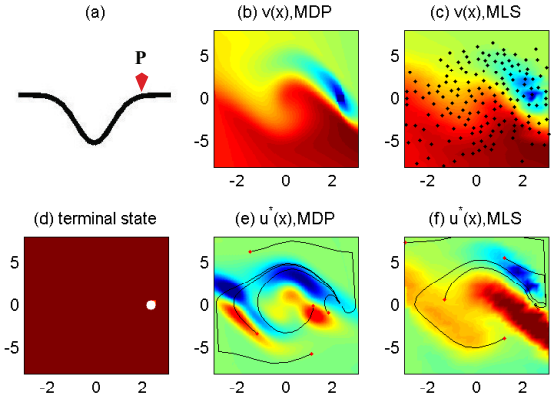


Figure 4.4: (a) illustrates the model and (d) shows the terminal states vs. nonterminal states. Write shows terminal states. (b-c) show the cost-to-go function $v(\mathbf{x}) = -\log(z(\mathbf{x}))$. (e-f) show the optimal control law obtained. In (e-f), dynamical simulations are shown by black curves, while their initial states are shown by red dots. (b),(e) show the results from the MDP method with 151×151 grid. (c),(f) show the results from the MLS approximation with 161 nodes generated by our adaptive scheme. Black dots in (c) represent nodes. The MLS solution is not identical to the MDP solution, yet the resulting control laws are similar. In (b)-(f), the horizontal axis represents position x_p and the vertial axis represents velocity x_v.

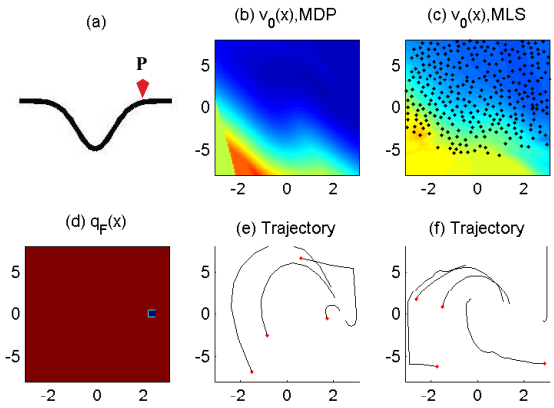


Figure 4.5: (a) illustrates the model and (d) shows the final cost function $q_F(\mathbf{x})$. (b-c) show the cost-to-go function $v(\mathbf{x}) = -\log(z(\mathbf{x}))$ at time 0. In (e-f), dynamical simulations are shown by black curves, while their initial states are shown by red dots. (b),(e) show the results from the MDP method with a 151×151 grid. (c),(f) show the results from the MLS approximation with 334 nodes generated by our adaptive scheme. Black dots in (c) represent nodes. Different from previous settings, here both desirability functions and optimal control law are functions of time, so we focus on showing the dynamical simulation. MLS tends to smooth out the desirability functions and gives a slightly different optimal control law. In (b)-(f), the horizontal axis represents position x_p and the vertial axis represents velocity x_v.

Example 2: Coupling a series of masses on ideal springs

This model simulates several identical masses attached to frictionless springs, which are dynamically independent. More details are described in the Appendix. Each mass can oscillate with any amplitude. The control objective is to generate movements such that: (1) the energy of each mass-spring equals a constant; (2) mass number (i) moves with phase $\pi/2$ ahead of mass number (i+1). We use the average-cost setting to solve the problem, with state cost function designed to enforce the above goals. This model is rather simple, but it has advantages when tuning the algorithm – namely it can be defined for any number of dimensions (masses), and the optimal behavior can be computed analytically: cosine functions $\cos(Ct + \phi)$ with appropriate phase changes.

Here, we applied a modification that includes a decay factor when the state is too far away from existing nodes. (This effectively applies an additional control to move the system back to nodes, or a hybrid method with the Gaussian approximator in [49]) . The formula is shown below. r is the distance to nearest node,

$$\tilde{z}^{MLS}(\mathbf{x}) = \begin{cases} z^{MLS}(\mathbf{x}), & r < R, \\ z^{MLS}(\mathbf{x}) \exp(-\lambda(r-R)^2), & r \geqslant R. \end{cases} \tag{4.58}$$

We found that with some prior information (i.e., initializing some nodes near the optimal trajectory), our approximation scheme generates good results even in a 14-dimensional state space (7 masses) and only requires a few thousand nodes, which is less than a regular grid with 2 nodes per dimension! This is because most of the nodes end up being places around the limit cycle. Fig. 4.6 shows the emergence of an attractive trajectory for systems composed of 2 masses and 7 masses respectively.

4.4.2 Numerical issues

Convergence

First we consider convergence for fixed bases. The power iteration method converges geometrically when the leading eigenvalue is real and its corresponding eigenvector is unique. This holds for the original problem (before the approximation), and in practice we have

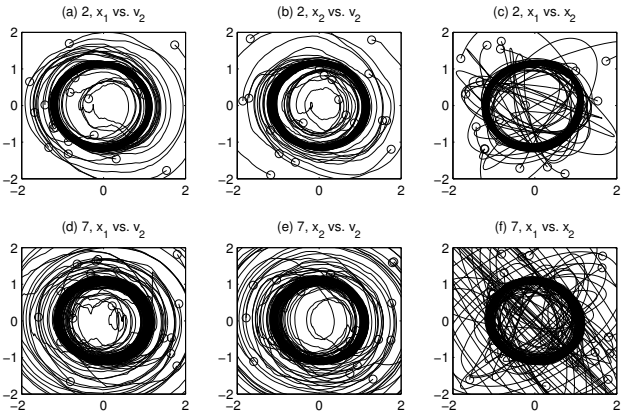


Figure 4.6: The trajectories are shown by projecting to different dimensions. The first row(a-c) shows the results for 2 masses (4-dimensional state space) with 166 nodes. The second row(d-f) shows the results for 7 masses (14-dimensional state space) with 2912 nodes. The first column(a, d) shows the trajectories of the first mass. The second column(b, e) shows the trajectories of the second mass. The last column(c, f) shows the positions of the first and second masses. Note that the analytical solution in each case is a circle with radius 1.

observed that it holds for the approximation as well. The rate of convergence corresponds to the mixing time of the MDP and is problem specific, but normally it is fast enough. We are guaranteed to get a positive solution. In our numerical tests, the CPU time spent on power iteration is always a small fraction of the total CPU time. Another issue is how accurate the solution is. Due to the nature of cubature formula, we cannot make our nodes distributed too densely; however, as shown in the previous part, the solution is good in the region covered by the nodes.

We then consider convergence of the adaptive scheme. The method is trivially guaranteed to terminate in a finite number of iterations, because the approximation volume and node density are limited. In our numerical tests, we observed that the solution improves when the number of nodes increases. The adaptive scheme usually terminates in tens of iterations.

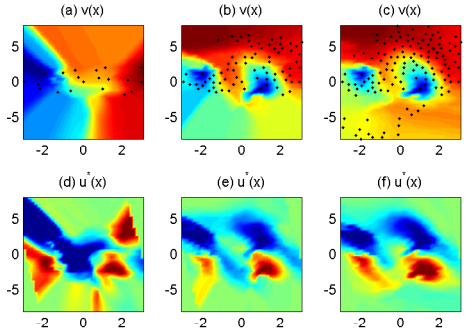


Figure 4.7: Here, we demonstrate the process of the adaptive scheme by showing the intermediate cost-to-go function and desirability functions for the average-cost setting of Example 1. The final result is shown in Fig. 4.2 (e,h), which is obtained with 17 iterations of our adaptive scheme. Here, the first column (a,d) shows results with initial nodes. The second column (b,e) shows the 4th iteration, while the third column(c,f) shows the 9th iteration. The first row (a-c) shows the cost-to-go function, where black dots represent the nodes. The second row (d-f) shows the optimal control law obtained.

Scalability to high-dimensional systems

Scalability to high-dimensional systems is the primary challenge in solving Bellman equations numerically. A successful method has to satisfy several conditions. First, the method should give a controller not far from the true solution. Second, the computational complexity of the method should scale well with the number of parameters (weights, grid points, etc). Third, the number of parameters needed to achieve good accuracy should scale well with dimensionality. The first and second conditions hold for most methods, including MDP discretization. The real challenge, however, is the third condition – which is rarely met in practice. Our numerical results illustrated that the new method has a lot of potential in terms of scaling. This is because the function approximator is automatically adapted to the problem at hand, and because for a fixed set of bases the problem becomes linear and is solved very efficiently. We now discuss these two points in more detail.

First we discuss the case of fixed bases/nodes. Our empirical results show that most of the CPU time is spend in constructing the discretized operator (Section 4.3.2) using the MLS method. The time complexity of this construction is $O(n^3)$, where n is the number of

state space dimensions. The cubature formula has time complexity $O(n_c^2)$ where n_c is the number of control space dimensions. Thus, when the number of nodes is N and the number of nearest neighbors being considered is K, the overall time complexity is $O(NKn^3n_c^2)$. The CPU time for constructing the discretized operator in Example 2, with the final set of nodes, was as follows. For 2 masses, with 100-200 nodes, it takes around 0.02 seconds. For 7 masses, with 2000-4000 nodes, it takes 40-160 seconds. These results are consistent with our theoretical analysis.

Second we discuss the time complexity of the adaptive scheme. We do not yet have theoretical results; with our current settings, the number of iterations did not increase significantly with state space dimensionality (this is possible because multiple nodes are added in each iteration, and the number of new nodes per iteration typically increases with dimensionality). In Example 2, the algorithm converges in 10-30 iterations, regardless of the number of masses. Each iteration involves (1) constructing the discretized operator, (2) finding the principal eigen-pair and corresponding control law, and (3) running a dynamical simulation starting from each current node. Part (2) takes negligible time, while parts (1) and (3) are comparable. The total CPU time for solving Example 2 with the adaptive scheme is as follows; for 2 masses it takes 1-3 seconds. For 7 masses it takes 400-4000 seconds. Our code is a mixture of MATLAB and C MEX files. The speed can be improved significantly by utilizing parallel processing; indeed, both the construction of the discretized operator and the stochastic simulation (starting from many different states) are easily parallelized.

4.4.3 Comparison with other methods

Comparison with MDP method

Previous results have demonstrated the similarity of results given by both methods. Theoretically, the MDP method can naturally give a "ground-truth" solution, while the MLS method tends to yield a bigger error.

However, MLS would outperform MDP in practice. First of all, normally the MLS costs much less CPU time than the MDP. For example, in the example shown in Fig.4.2, the MDP method (Fig. 4.2(c,f)) takes a hour in MATLAB, while the MLS method (Fig. 4.2(d,g))

takes 0.1 seconds without adaption 0.09 seconds with adaption (Fig. 4.2(e,h)).(Programming details and complexity of model may affect those estimates, but normally MDP would run much slower than MLS.) In both cases, most CPU time is spent in constructing a matrix (probability transition matrix for MDP, discretized operator for MLS). In high dimensional practical control problems, the MDP method is not numerically possible, since it needs dense grids on every dimension on state space. Second, since the MDP method is to approximate a problem within a finite box in state space, it may yield suspicious results near boundaries (thus, in Section 4.4.1, we perform the MDP method in a bigger region than shown). The MLS method tends to have less suspicious results near boundaries due to its exploration power.

Comparison with the gaussian method

The method presented here differs significantly from the method using gaussians presented in [49] except for different choices of bases functions. First, the method here uses power iteration rather than quadratic optimization or nonlinear optimization. Second, the method here uses nonparametric ways to adapt the bases. Our comparison need to take these factors into account.

First, we test the gaussian method on Example 2 with a similar nonparametric adaptive scheme as described here in Section 4.3.5. The covariances of gaussians are allocated based on distance between each other, thus providing a fair comparison with the MLS method. MATLAB built-in optimizer "quadprog" is used to perform quadratic programming. The results are similar to that in Secion 4.4.1 with 2, 3, and 4 masses, but fail for more masses. We can conclude that, MLS bases have the following advantages over gaussian bases. First, solving the resulting eigen problem cost less CPU time than gaussian bases. Second, the MLS method is less prone to converge to the wrong eigenvector. The disadvantage of MLS bases is that constructing MLS bases tends to consume more time than constructing gaussian bases. The total CPU time consumed with both methods is similar. In conclusion, when scaling to high dimensionality, if the number of bases grows, or if the results cannot easily be verified, MLS tends to outperform gausian bases.

Then, we test the adaptive scheme in Secion 4.3.5 by comparing above results with results from gaussian bases with gradient descent. MATLAB built-in optimizer "fmincon" is used to perform nonlinear optimization. When gaussian bases are initially placed properly (near trajectories), the gradient descent method may normally converge to acceptable results. The results are similar to that in Section 4.4.1 with 2, 3, 4, and 5 masses, but fail for more masses. The time consumed with gradient descent methods is much higher than non-parametric adaptation if they converge to acceptable results. For example, with 4 masses, gradient descent bases adaptation needs 46 seconds with 50 bases, and adaptive scheme in Section 4.3.5 with 161 gaussian bases needs 0.17 seconds, adaptive scheme in Section 4.3.5 with 191 MLS bases needs 0.06 seconds. This is likely due to the need to repeated construct gaussian bases. Moreover, the gradient descent method is very likely to converge to wrong eigenvector (λ very small) when initially bases are not placed properly. To conclude, the method presented here outperforms the gaussian approximation method with gradient descent presented in [49].

4.4.4 Limitations

On the other hand, we found that this method cannot scale well to some difficult benchmark control problems, such as acrobot or humanoid. In those cases, the resulting control problem may appear to move randomly, indicating that we cannot find an acceptable solution with this method. This is due to inaccuracies in both value function and optimal control policy. We found the value functions are not smooth enough, due to the truncation used. We therefore explore other base functions in later sections.

4.5 Conclusion

Here we developed a function approximation method for linearly-solvable stochastic optimal control problems. This is our first attempt to engineer base functions in order to solve leading eigen-pair more easily. Numerical tests show MLS method's effectiveness in simple problems, yet we lack good results on difficult problems with applications. In addition, the cost to compute base functions on high dimensionality is still high.

For these reasons, although we still believe that MLS may be a good candidate for function approximators in other applications, in the field of solving for LMDP's value function, we will examine other base functions and other ways to obtain control policy in later sections.

Chapter 5

AGGREGATION METHODS FOR LINEARLY-SOLVABLE MDP

(Most of the contents in this chapter are shown in [60].)

5.1 Introduction

5.1.1 Motivation

The motivation of this section shares that of the last section moving least squares. We want a method that improves or supplements the moving least squares method, yet avoid solving a weighted eigen-problem $\lambda F\mathbf{w} = G\mathbf{w}$. We also want a framework that makes first-exit formulation easier to solve and that is based on different theoretical background than MLS. Furthermore, we want a function approximator that is smoother than MLS. That brings us to a normalized gaussian approximator, which is the basis of the method described here.

5.1.2 Contribution

The new method proposed here is different from the other methods in concept. Previous methods are function-approximation schemes. They approximate the desirability function $z(x) = \exp(-v(x))$ defined on the continuous state space and solve the corresponding linear equation; v is the optimal cost-to-go function. Here, we are proposing a problem-approximation scheme. Using aggregation methods, see [10, 36], we approximate the optimal control problem as one defined on discrete states, which we call clusters, and then solve the resulting linearly-solvable MDP (or LMDP). The new approximation scheme leads to simple eigen-problems in the form $\lambda\mathbf{w} = QG\mathbf{w}$, so it always converges to the principal eigenvalue and guarantees positivity of the solution. This scheme also provides extra flexibility in terms of constructing the approximation in first-exit problems (see below). On the other hand, it requires more dynamics evaluations compared to function approximation with Gaussian bases – by a factor of $O(m)$, where m is the state dimensionality – because certain inte-

grals cannot be computed analytically and instead require numerical approximations via Gaussian cubature.

5.2 Soft state aggregation scheme

In this section we first summarize the idea of soft state aggregation [10, 36]. Then we introduce our choice of aggregation and deaggregation probabilities. After that, we show how to recover the solution of the original problem using clusters obtained from aggregation. Issues regarding setting parameters of those clusters are discussed at the end.

5.2.1 Soft state aggregation for LMDPs

To aggregate the entire space into a finite number of clusters, we need to construct the clusters $\mathcal{S}$ and define two choices of probabilities relating the clusters to the original system states.

Definition 1. *(Aggregation and deaggregation probabilities)*

1. *Aggregation probability* $p(i|x) = \phi_i(x)$, $x \in \mathcal{X}, i \in \mathcal{S}$ *is a probability-like quantity that is interpreted as the "degree of membership of* x *in the aggregate state* i*".*

2. *Deaggregation probability,* $p(x|i) = d_i(x)$, $x \in \mathcal{X}, i \in \mathcal{S}$ *is a probability-like quantity that is interpreted as the "degree to which* i *is represented by* x*".*

Aggregation and deaggregation probabilities naturally satisfy the following conditions

Fact 1. *(Aggregation and deaggregation probabilities)*

1. *nonnegativity*

$$\phi_j(x) \geq 0, d_i(x) \geq 0. \tag{5.1}$$

2. *normalization conditions*

$$\sum_{j \in \mathcal{S}} \phi_j(x) = 1, \int_{x \in \mathcal{X}} d_i(x) dx = 1. \tag{5.2}$$

3. Bayes rule

$$\phi_i(x) = p(i|x) = \frac{p(x|i)p(i)}{p(x)} = \frac{d_i(x)p(i)}{p(x)}, i \in \mathcal{S}, x \in \mathcal{X}, \tag{5.3}$$

where $p(i)$ and $p(x)$ are some probability densities over $\mathcal{S}$ and $\mathcal{X}$.

When the aggregation probabilities $\phi_i(x)$ are not all 0 or 1, this method is called *soft aggregation*, which implies that each state of the original system may belong to multiple clusters.

Suppose we have a continuous-state LMDP with passive dynamics $p(x'|x)$, $x, x' \in \mathcal{X}$. We would like to approximate the desirability function $z(x), x \in \mathcal{X}$. Based on the aggregation framework, the transition probabilities among the clusters can be expressed as

$$\begin{aligned} \hat{p}(j|i) &= \iint_{x,x' \in \mathcal{X}} p(j|x')p(x'|x)p(x|i)dxdx' \\ &= \iint_{x',x \in \mathcal{X}} \phi_j(x')d_i(x)p(x'|x)dxdx'. \end{aligned} \tag{5.4}$$

The state cost function over clusters is

$$\exp(-\tilde{q}_i) = \int_{x \in \mathcal{X}} \exp(-hq(x))d_i(x)dx. \tag{5.5}$$

This completes the construction of the approximating LMDP on clusters, which is defined on a finite and discrete state space.

5.2.2 Choice of aggregation and deaggregation probabilities

In this paper we choose the deaggregation probabilities as multidimensional normal distributions:

$$d_i(x) = c_i^{-1} \exp(-(x - m_i)^T S_i(x - m_i)/2), \tag{5.6}$$

where c_i^{-1} denotes the normalization factor. Then $p(x)$ and $p(i)$ in (5.3) are chosen as the following mixtures(5.7):

$$p(i) = c_i, p(x) = \Sigma_{i \in \mathcal{S}} p(i)p(x|i), \tag{5.7}$$

then based on (5.3), the aggregation functions are

$$\phi_i(x) = \frac{\exp(-(x - m_i)^T S_i(x - m_i)/2)}{\sum_{i \in \mathcal{S}} \exp(-(x - m_i)^T S_i(x - m_i)/2)}. \tag{5.8}$$

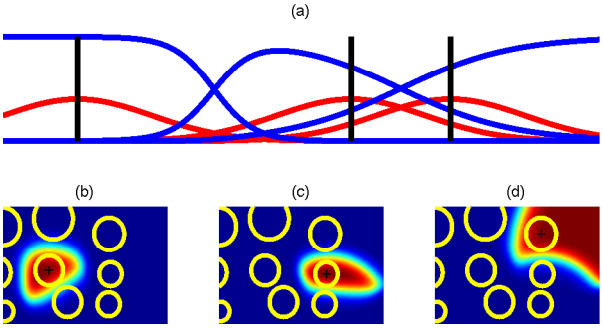


Figure 5.1: This figures demonstrate the shape of aggregation functions $d_i(x)$ and deaggregation functions$\phi_i(x)$ in the aggregation method. (a) Aggregation and deaggregation functions in 1D. Blue lines show aggregation, while red lines show deaggregation functions. Black lines show the means of the clusters. (b-d) shows several aggregation functions in 2D. The ellipses correspond to the contours of the Gaussian deaggregation functions.

The shape of the aggregation and deaggregation functions is illustrated in Fig 5.1. To improve computational efficiency and generate a sparse transition probability matrix, the summation in (5.8) will be limited to nearby clusters, and $\phi_i(x)$ will be set to 0 if the m_i is not among the k nearest neighbors to x. Note that we only choose the shape of the deaggregation functions (Gaussian). The aggregation functions are then constructed automatically via normalization. The result of the normalization is to reshape the Gaussians in a way that fills the gaps and covers the space more uniformly.

5.2.3 Finding the desirability function $z(\mathbf{x})$

Overview

The double integral (5.4) is not trivial to calculate numerically. Currently we are employing a cubature formula [29, 39] that requires evaluating the aggregation function and the $a(x)$ function $O(m)$ times, where m is the state space dimensionality. Once the discrete-space LMDP is constructed, we can solve it by finding the principal eigen-pair for a problem in the form

$$\lambda \mathbf{z} = QP\mathbf{z}. \tag{5.9}$$

Here, $P_{ij} = \hat{p}(j|i)$, $\lambda = \exp(-hc)$, Q is a diagonal matrix with $Q_{ii} = \exp(-\tilde{q}_i)$ in (5.5). One way to solve this problem is the power iteration method (in [57]): $\lambda \mathbf{z}^{k+1} = QP\mathbf{z}^k$, with $\|\mathbf{z}\|$ fixed. The matrix QP is guaranteed to have a principal eigenvalue less than 1, and the corresponding eigenvector is guaranteed to have positive elements, because all elements of the matrix are nonnegative.

For the first-exit formulation [47], the corresponding linear equation is

$$\mathrm{diag}(\exp(\mathbf{q}_{\mathcal{N}})) - P_{\mathcal{N}\mathcal{N}}\mathbf{z}_{\mathcal{N}} = P_{\mathcal{N}\mathcal{T}} \exp(-\mathbf{q}_{\mathcal{T}}). \tag{5.10}$$

Here, $\mathcal{N}$ stands for nonterminal states and $\mathcal{T}$ stands for terminal states, and $P_{\mathcal{N}\mathcal{N}}, P_{\mathcal{N}\mathcal{T}}$ are the corresponding blocks of P in (5.9). We can approximate the continuous-state problem if we allocate several clusters as terminal states. Note that this is not equivalent to the continuous function approximation approach, which requires a boundary condition on $z(x)$. Let $T \subset \mathcal{R}^m$ denote the set of terminal states in the continuous state space. The continuous function-approximation scheme would involve integrals such as $\int_{x' \in \mathcal{R}^m \setminus T} p(x'|x) f(x') dx'$, which would be difficult to evaluate in high-dimensional problems.

With regard to computational complexity, estimating (5.4) tends to be the bottleneck. Finding the K nearest neighbors can be done efficiently using tree decompositions, while evaluating (5.4) requires $O(NKm^3)$ computations where N is the total number of clusters (see below).

Numerical issues for estimating probability transition matrix of clusters

$$\hat{p}(j|i) = \iint_{x', x \in \mathcal{X}} \phi_j(x') d_i(x) p(x'|x) dx dx' \tag{5.11}$$

is not trivial to be calculated numerically. We will use cubature formula again. With $x' = \mathbf{x} + ha(\mathbf{x}) + \sqrt{h}\sigma B(\mathbf{x})\xi$, the integral becomes

$$\hat{p}(j|i) = \iint_{x \in \mathcal{X}, \xi \in \mathcal{R}^{m_c}} \phi_j(\mathbf{x} + ha(\mathbf{x}) + \sqrt{h}\sigma B\xi) d_i(x) \mathcal{N}(\mathbf{0}, I) d\xi. \tag{5.12}$$

Here $\mathcal{N}(\mathbf{0}, I)$ represents the probability density function of a unit Gaussian distribution. $d_i(x)\mathcal{N}(\mathbf{0}, I)$ is a Gaussian function. If the problem is defined in an entire space or the clusters are small, the above integral can be approximately calculated by the cubature formula. Those formulas approximate the integral with a weighted summation of the function

evaluated at carefully selected sampling points.

$$I[\Phi] = \int \Phi(\mathbf{y}) \exp(-\mathbf{y}^T \mathbf{y}) d\mathbf{y} \approx Q[\Phi] = \Sigma_{j=1}^{n_{sample}} w_j \Phi(\mathbf{y}^j) \tag{5.13}$$

We currently are using one with $n_{sample} = 2(m + m_c) + 1$ points, a variation from [39], where m is the dimensionality of the state space and m_c is that of the control space.

$$\begin{aligned} Q[\Phi] &= w_1 \pi^{n/2} \Phi(\mathbf{0}) + w_2 \Sigma_{full\,sym.} \Phi(\sqrt{\lambda}, 0, .., 0), \\ & w_1 = \tfrac{2\lambda - 1}{2\lambda}, w_2 = \tfrac{4}{n}\lambda, \end{aligned} \tag{5.14}$$

λ is an arbitrary parameter. Here, "full sym." means all possible indexes, permutations and reflections. Therefore, for a m dimensional system with N clusters, the time-complexity for computing the entire matrix is $O(NKm^3)$, where K corresponds to how many neighboring points you have chosen to evaluate. That is because :(1) calculating a Gaussian needs $O(m^2)$ time and (2) calculating a row of the probability transition matrix require evaluating Gaussians $O(Km)$ times. In our trials, computing the transition matrix seems to be a bottleneck.

5.2.4 Finding optimal control law $\mathbf{u}^*(\mathbf{x})$

Once the discrete-space LMDP is solved, the continuous-space desirability function is

$$z(x) = \sum_{j \in \mathcal{S}} \phi_j(x) z_j, \tag{5.15}$$

and then the optimal control law $u^*(x)$ can be computed from (3.3). Numerically this can also be calculated through cubature formulae.

However, since aggregation methods map the original continuous space problem to discrete states, we can obtain optimal transition probabilities between clusters and get vector optimal control policy as well.

5.2.5 Determining the position and covariances of clusters

The clusters (defined by the Gaussian deaggregation functions) are characterized by their means and covariances. Obviously the choice of means and covariances will have a large

effect on the quality of the approximation. Here we present a method for choosing these parameters in a suitable way.

The means of clusters are chosen in a similar fashion to that in Section 4.3.5

Once the means of the clusters are chosen, we need to choose the covariances. One possibility is to use gradient descent; however, this is likely to be slow because the covariances have a large number of parameters. Instead, we use a heuristic method to ensure that the clusters have some overlap. Since the solution $z(x)$ tends to be more spread out in the directions where the noise acts, the covariances need to be more elongated in these dimensions.

Our current results involve setting covariances in the following ways. (1) Manually set them, but this involves some insight into the specific system. (2) Assume that they have the same shape, in other words the inverse covariance matrix S_i for all clusters is the same up to multiplication by a scalar; this scalar is determined by the distance to other clusters and how much overlap between clusters we need. Keeping S_i diagonal tends to give good enough results in most of our test problems.

5.3 Numerical results

In this section we present numerical results for simple problems. First we will illustrate that the results given by the aggregation framework are meaningful in simple problems. Second we will illustrate that the method scales to a high-dimensional systems.

5.3.1 Test problems, solutions and dynamical simulations

Here we will focus on the desirability function $z(\mathbf{x})$, cost-to-go function $v(\mathbf{x})$, optimal control law $u^*(\mathbf{x})$, and dynamical simulations based on $u^*(\mathbf{x})$. When possible, we will compare these results to the "ground truth" obtained from an MDP discretization [49] on a dense grid.

Example 1: Car-on-the-hill

This test problem has been discussed before in moving least squares. For the average-cost setting, we design a cost-function $q(\mathbf{x})$ (Fig.5.2(b)) that encourages the car to pass through

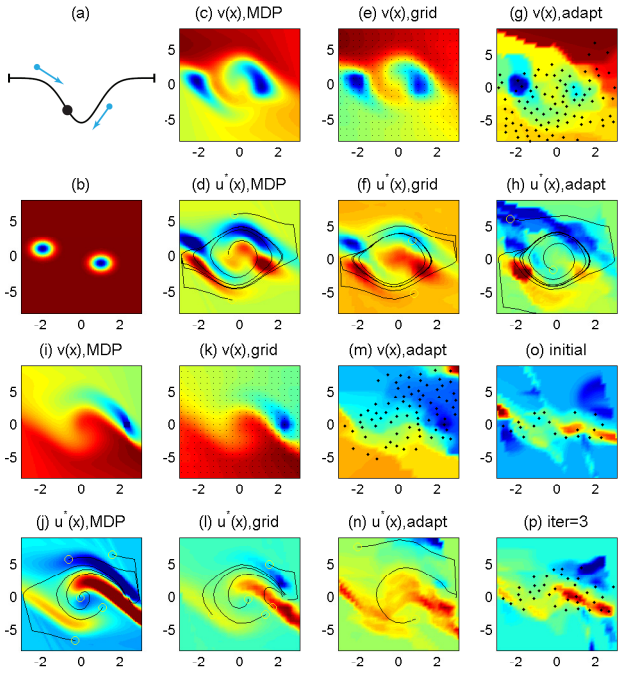


Figure 5.2: (a)demonstrates the problem. Curve represents the "hill" and arrows represents via points in average-cost formulation. (b)demonstrates the state cost function $q(\mathbf{x})$ of the average cost formulation. (c-h) show results given by average-cost formulation. (i-p) show results given by the first-exit problem. Except for (a,b,o,p), the top and third rows show the cost-to-go function$(x) = -\log(z(x))$, with blue representing low cost-to-go or high desirability and dots representing where clusters are. The second and fourth rows show the corresponding optimal control law. Solid lines show the trajectories given by a simulation with several random initial states (indicated by yellow circles). (c-d,i-j) show the results given by MDP with 151 x 151 states sampled on bigger region than shown. (e-f,k-l) show the results given by putting clusters on a 21x21 grid. (g-h,m-n) shows the results given by putting clusters with the adaptive scheme. The average-cost one (g-f) uses 94 clusters, (m-n) the first-exit one uses 78 clusters. For those results by adaptive clusters, their covariances are adaptive to match their distance from the nearest neighbor. (o) shows initial clusters, and (p) shows clusters after 3 iterations. The result given by (m-n) is obtained after the 14th iteration. Except (a), the horizontal axis represents horizontal position and the vertical axis represents velocity.

two targets that have non-zero desired velocities; thus the optimal behavior involves a limit cycle. As shown in Fig.5.2(c-h), the cost-go function and control law given by our approximations are consistent with the dense MDP solution, even though here we use a small number of bases.

We can also use this dynamical model to create a first-exit problem. The goal of this model is to park the car (i.e., achieve zero velocity) at horizontal position 2.35. The running state cost is constant; thus, it effectively acts as a cost for time. Again, in Fig.5.2(i-p), the results are consistent with the MDP method. Subplots (o-p) give the initial clusters and the intermediate clusters during iterations. We can see that there are fewer clusters at the right-bottom corner, where the car visits less frequently.

For the car-on-a-hill model, one interesting result is that our framework could capture collisions properly by solving for control laws taking advantage of collisions. This behavior is difficult to capture using trajectory-based methods.

Example 2: Coupling a series of masses on ideal springs

This simple model has also been used in the section of moving least squares. It is rather simple, but it has advantages when tuning the algorithm. First, it has similar behavior when scaling to higher dimensionality (simply changing the number of masses). Second, the limiting behavior of this system can be easily computed theoretically: it is in the form of cosine functions $\cos(Ct + \phi)$ with appropriate phase changes. Note that the solver does not "know" about this analytical form and instead is applying a generic numerical method.

Results in Fig 5.3 show that we can control systems with 2,3,5 and 7 masses using only hundreds of clusters. Since this model is better used for fine-tuning this method, we will discuss more details of this model in the following section about scalability.

5.3.2 Scalability to high dimensional systems

Scalability to high dimensional systems is the primary challenge of solving Bellman equations numerically. A successful method needs to satisfy several conditions. First, the method should give a controller not far away from the true solution. Second, the computational

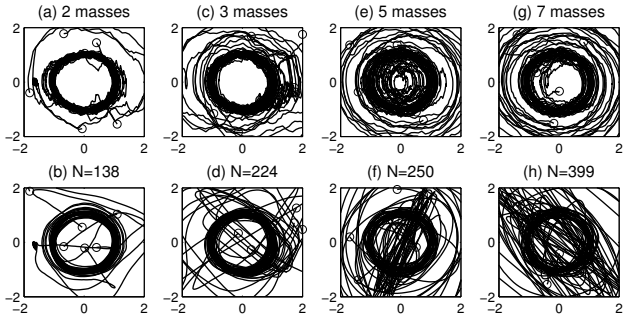


Figure 5.3: Results for N spring-mass model with different dimensionalities. The first row represents simulated trajectories projected to x_1 vs. v_1 plane, while the second row represents those projected to x_1 vs. x_2 plane. (a-b) are for 2 masses, (c-d) are for 3 masses, (e-f) are for 5 masses, (g-h) are for 7 masses. N stands for number of clusters. Circles represent initial states. All of them are calculated with hundreds of clusters. We can see that most trajectories converge to the unit circle on planes, which are projections of the theoretical attracting trajectories.

complexity of the method should scale well with the number of samples used. Third, the number of samples needed to achieve good accuracy should scale well with dimensionality. The first and second conditions normally hold (e.g., MDP), and the real challenge is the third one.

At least in the Example 2 shown, our framework can control a high-dimensional system semi-globally with a surprisingly small number of clusters given the dimensionality. In general, the number of samples needed is likely to be problem specific, and might blow up due to the complexity of a system. Nevertheless, our numerical results to date are encouraging.

We also note that the CPU time scales well with dimensionality. Our code is written in C and is multithreaded. It runs on a quad-core 3GHz Intel i7 processor. Going from 2 masses to 7 masses (4 to 14 state dimensions), the total CPU time increases from 0.5 seconds to 4 seconds. The CPU time for estimating the probability transition matrix increases from around 15 milliseconds to 80 milliseconds. When using 7 masses and 20000 clusters, computing the transition matrix takes 56.5 seconds with 6 threads, as opposed to 243.7 seconds with 1 thread. Thus, our algorithm successfully takes advantage of multithreading.

5.3.3 Comparison with other methods

Comparison with MDP method

The aggregation method can be viewed as a natural extension of the regular discretization MDP method. Previous results have demonstrated the similarity of results given by both methods. Like MLS, however, aggregation may also outperform MDP in practice with good selection of aggregation and deaggregation probabilities.

Comparison with Gaussian method

The method presented here differs significantly from the method using gaussians presented in [49]. First, the method here uses power iteration rather than quadratic optimization or nonlinear optimization. Second, the method here uses nonparametric ways to adapt the bases.

Since the aggregation method is based on LMDP defined in discrete state spaces, it also provides more flexibility than the Gaussian method in first-exit formulation.

Comparison with moving least squares methods

In general, this method outperforms MLS. First, it avoids truncations and provides more smooth value function, frequently give us smoother value function and better control policy. Second, it is faster than MLS. Moreover, it provides more flexibility than MLS in first-exit formulation.

5.3.4 Limitations

On the other hand, we found that this method also cannot scale well to some difficult benchmark control problems, such as acrobot or humanoid. This implies problems with value function approximation, which we will cover in later sections.

5.4 Summary

Here we presented a new problem-approximation scheme, which transforms a linearly solvable optimal control problem with continuous state space into a problem with discrete state

space. This provides extra flexibility in the problem formulation and also avoids numerical issues with function approximation methods. Preliminary results show that it can generate acceptable control laws and is scalable to higher dimensional systems. We also employed an adaptive scheme for aggregation/deaggregation functions that increases the approximation accuracy in the regions of state space that visited most often.

However, on certain problems that are harder, such as acrobot or humanoid, the basic method presented here does not scale automatically. We therefore explore similar value function approximations and other control policy retrieval methods. The source of such limitations and possible ways to enhance the basic aggregation method are proposed in the following chapters.

5.5 Future works

Certain works postdating the initial publication ([60]) of this chapter are discussed in later chapters. In addition, there are still multiple directions that are appealing in improving the basic method of aggregations. First, approximating value function with normalized gaussians is not compulsory in the aggregation setting. Using other distributions rather than normal distribution may be appealing in certain applications. Second, the transition between neighboring clusters can be local controllers such as LQR or Lyapunov-based controllers, which may enhance local stability of the resulting controller. Third, certain actor-critic approaches can be applied to aggregation to adapt clusters and improve the quality of the solution.

Chapter 6

LIMITATIONS OF LMDP GLOBAL METHODS

In previous few chapters, we discussed numerical methods for LMDP and gave the results for some simple test problems. One may naturally ask whether LMDP can help us solve complicated problems arising in applications for robotics, such as walking. However we found that there are obstacles that prevent us from automatically applying LMDP global value function methods in certain applications.

In this chapter, we will first discuss alternative function approximation strategies for value function in LMDP setting to further enrich our tools to solve complicated problems (yet all methods in previous chapters and this section failed on problems such as acrobot). Then we will discuss the limitations of LMDP value function methods. In the end, we will outline how to detour such limitations.

6.1 Alternative function approximation schemes

In addition to the methods in previous chapters, we also have explored several numerical methods to exploit the benefits brought by the linear-solvable framework. Although promising in some aspects, they perform at most on par with the aggregation method. None of this methods discussed is helpful in solving certain harder problems such as cart-pole or acrobot.

6.1.1 Alternative base functions

To obtain a discretized problem in the form of $\lambda \mathbf{w} = QP\mathbf{w}$, alternative approaches exist and may give similar results.

Normalized Gaussian

Similar to the aggregation methods, we can use normalized Gaussian functions (the deaggregation function in the previous chapter) as base functions. This choice of base function avoids the double integral used in the previous chapter, simplifies calculation, yet we still may end up a discretized problem $\lambda F\mathbf{w} = QP\mathbf{w}$ with a nondiagonal F matrix. However, when the covariance matrices in gaussians are big enough, the F matrix may be very close to a diagonal matrix, so we treat it as a diagonal matrix.

The numerical results are similar to those of aggregation methods. The resulting control policy is slightly worse than the aggregation method, and this method consumes less CPU time. However, the approximation of treating F as diagonal lacks sufficient theoretical basis, so we tend not to extend this method.

Similarity weight function

[27] discusses a motion fields method for interactive character locomotion. It uses a function approximator based on "similarity" between two states of characters in locomotion. It can be written as follows,

$$\phi_i(x) = \frac{1}{\eta}\frac{1}{d(x,m_i)^2}, \eta = \Sigma_i \frac{1}{d(x,m_i)^2}, \tag{6.1}$$

where $d(x, m_i)$ is the distance between states andis tailored to walking tasks. m_i are nodes, which are important motion states. It satisfies formulae similar to moving least squares and yields discretized problem $\lambda w = QPw$.

However, the disadvantage of this function approximator is clear. The solution away from the nodes of approximators tends to average the results and may thus give an illusion that the value function away from solved region is more desired. In [27], value functions are likely to be used only very close to previous trajectories measuring with $d(x, m_i)$, and it is not likely what we want. Hence, we do not explore more on this type of base functions further.

6.1.2 Linear base functions in value function space

Since the Bellman equation is linear for $z(\mathbf{x})$, we mainly discussed linear base functions in desirability function $z(\mathbf{x}) = \exp(-v(\mathbf{x}))$ space. On the other hand, we can approximate $v(\mathbf{x})$ with linear base functions and calculate $z(\mathbf{x})$ based on it.

The value function can be approximated as

$$v(\mathbf{x}) = \Sigma_i w_i \phi_i(\mathbf{x}), \tag{6.2}$$

where ϕ_i are base functions, such as normalized gaussians or moving least square base functions.

Approximating in value function space would yields a method similar to power iteration; however, each iterations involves computation with complexity $O(N \times N \times S)$, where N is the number of collocation points and S is the number of samples in collocation methods.

Numerical results shows that the controller based on this method is very similar to those in previous chapters. The benefits of approximating the function in $v(x)$ are minor, with much higher computational costs. Moreover, it cannot exempt us from truncating the moving least square base functions.

6.1.3 Alternative method to find eigenfunction with Gaussian base functions

The major problem for the gaussian base functions [49] approach is converging to the wrong eigenvector. For a given set of base functions, w in $\lambda F w = G w$ may be likely to converge to the wrong eigenvector in the quadratic programming approach.

Alternatively, this can be improved by solving the following nonlinear optimization problem (numerically by quasi-Newton methods such as L-BFGS);

$$\min_w f(w) = \min_w \log(w^T F^T F w) + \log(w^T G^T G w) - 2\log(w^T F^T G w),\ s.t.\ w \geq 0. \tag{6.3}$$

This is inspired by maximizing generalized cosine similarity between vector Fw and Gw (the generalized cosine similarity is $\sqrt{\exp(-f(w))}$, and $f(w)$ is chosen to improve numerical performance and to simplify the calculation of gradient $\nabla f(w)$).

Unfortunately, there is no guarantee for (6.3) to converge to the leading eigenvector. In practice, we did found that (6.3) demonstrated a better chance of converging to the desired solution than the object function in [49], and we will use this object function to find weights for gaussian base functions.

This improvement still cannot help us to solve even harder problems such as acrobot, so (6.3) is just a minor improvement on our existing method[49].

6.1.4 Improvements on solution methods

Power Iteration

Using power iteration to solve the leading eigenpair of the Bellman equations is one of the key aspects to the quality of our solution. There are multiple algorithms for eigenvalue problem [57].

Arnoldi iteration computes the eigenvalue problem in Krylov subspace; it will therefore converge to the leading eigenpair faster. However, positivity of the solution is not guaranteed in Arnoldi iteration, so we still need power iteration as the last steps to refine the results.

However, improvements brought by Arnoldi iteration provide little improvement in the overall time complexity and no improvement in the accuracy of the value function solved. Hence, this is still a minor improvement that cannot help us to expand value function approaches in more applications.

Alternative methods for numerical integration

Numerical integration is the most important step in our algorithms and determines our time complexity and accuracy. Cubature formulae are accurate if the object function is a low order polynomial, which will never happen in our desirability functions. Other methods, such as integrating on grids and random sampling, normally involve more sampling, and they significantly improves the accuracy of integration. However, those alternative approaches also slow down the process and limit the amount of base functions that we can use in a reasonable time frame. Moreover, we have not found any evidence that improving accuracy in numerical integration will improve control policies and help us apply LMDP to more

complicated problems.

When we are integrating with cubature formulae, we can center the distribution on current control policy instead. This is similar to importance sampling, and the details are shown as follows: When calculating the following integral

$$g(x) = \int p(x'|x)f(x')dx, , \tag{6.4}$$

supposing we have an approximated control policy $\mathbf{u}_{guess}$, we can instead rewrite (6.4) as

$$g(x) = \int \frac{p(x'|x)}{p(x'|x, \mathbf{u}_{guess})} f(x')p(x'|x, \mathbf{u}_{guess})dx, . \tag{6.5}$$

Here, $\frac{p(x'|x)}{p(x'|x,\mathbf{u}_{guess})}$ is the exponentiation of a linear function. Then we can treat $p(x'|x, \mathbf{u}_{guess})$ as gaussian, and we treat $\frac{p(x'|x)}{p(x'|x,\mathbf{u}_{guess})} f(x')$ as the arbitrary function. In this way, numerical integration is likely to be more accurate when the desired control policy is large. However, updating (6.5) based on new results on control policy requires calculating linear operators again and slowing down the process. So (6.5) is an improvement on par with our existing methods.

6.1.5 Summary

In this section, we discussed multiple alternative improvements to our existing methods. Our exploration shows that they may not outperform our existing methods and they are not helpful in solving either cart-pole or acrobot. This implies that minor improvements in accuracy of the function approximator may not be the real cause of the problem. We will explore the cause in depth.

6.2 Limitations of value function scheme

6.2.1 LMDP value function approaches fails in acrobot

The acrobot is one of the most difficult benchmark low dimensional problems in the field of optimal control problems. It has 4 dimensions in state space and 1 dimension in control space, making dense grid representation of any global function difficult but not prohibitive. The nonlinearity and instability make the problem even harder. The cart-pole is generally

easier than acrobot, yet it is still underactuated and hard to control, and it can be viewed as an intermediate problem between easier problems and acrobot.

Value function based methods are mostly able to control acrobot yet fail to stabilize it. The only successful result is based on neural network and showing the sharp ridge of value function. On the other hand, several teams succeed in swinging up and stabilizing it based on transformation and trajectory planning with extensive manual control design. MPC also succeeds in swinging up and stabilizing it, which means optimal control schemes can match the results of those more manual control policies.

In our results on LMDP based global methods, both cart-pole and acrobot seem to move randomly in our derived control policy, yet the system may sometimes reach the upright position. A natural control design is a two-step process: let LMDP control the system to the upright position, and let LQR stabilize it locally. In cart-pole, it is possible to let the aggregation method swing up and let the LQR controller stabilize it on the top. However, due to the extreme nonlinearity of acrobot, this approach fails on acrobot.

These results further question how many realistic problems LMDP value function methods can actually solve. For example, humanoid shares all of acrobots's difficulties;(1) instability,(2)nonlinearity, and (3) underactuation. Moreover, humanoid has much more dimensionalities and has to deal with discontinuity in dynamics. Thus, it is important to understand why LMDP based global methods tends to fail with acrobot and similar problems.

6.2.2 Value function approach is indirect

Value function approaches generally (1) approximate value function $v(x)$; (2) use certain rules to generate control $u(x)$, which is a function of state variables ; and (3) generate actual state and control time sequence by simulation. Since it is an indirect method, a close-to-optimal value function cannot even guarantee a close to optimal $u(x)$. Moreover, the users may judge the performance of an optimal controller based on the actual state and control sequence (trajectory). Users may judge optimal controllers with criteria such as stability or response time, rather than cost functions, which are arbitrarily defined, thus

further complicate the problem. For example, when we are using the cruise control of a vehicle and setting the speed as 60 mph, stabilizing the automobile at 58mph may be more preferred by users than keeping it fluctuating between 50mph and 70mph (with mean speed 60mph), even though the latter may yields lower cost and correct expected speed.

The indirectness not only complicates any potential error analysis, but also lessens the impact of value function approaches. To succeed in practice with value function approximation, one may need manual tweaking parameters. We will show the difficulty of parameter tweaking and demonstrate methods to detour such difficulty in later sections and the next chapter.

6.2.3 Certain problems may require prohibitive accuracy in value function approximation to avoid disastrous outcomes

Here we will show that for certain optimal control problems involving instability, underactuation and nonlinearity may require too high density of base functions, proper placement of base functions, or carefully designed form of function approximators.

Method

Here we will isolate the following factors to concentrate on the limitations of value function schemes. First, we will focus on linear systems and their variants to provide "ground truth" solutions of cost-to-go functions by linear Riccati equations. Second, we will only consider continuous and deterministic dynamical systems to avoid extra analysis. (A Linear Riccati equation in continuous LQR system can be solved by build-in command "care" in MATLAB's control toolbox).

Thus the dynamical system would be written as

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x} + B\mathbf{u}. \tag{6.6}$$

The cost function is

$$l(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T Q\mathbf{x}/2 + \mathbf{u}^T R\mathbf{u}/2 \tag{6.7}$$

The corresponding linear Riccati equation is

$$A^T V + VA - VBR^{-1}B^T V + Q = 0. \tag{6.8}$$

The optimal value function is

$$v^*(\mathbf{x}) = \mathbf{x}^T V \mathbf{x}/2. \tag{6.9}$$

The optimal control policy is

$$\mathbf{u}^*(\mathbf{x}) = -R^{-1}B^T V\mathbf{x} = -R^{-1}B^T \nabla v^*(\mathbf{x}). \tag{6.10}$$

We can use function approximators to approximate value function

$$v_{approx}(\mathbf{x}) = -log(\Sigma_i w_i \phi_i(\mathbf{x})), \tag{6.11}$$

and make the approximated value function accurate on nodes/samples, $v_{approx}(\mathbf{x}_I) = v^*(\mathbf{x}_I)$.

Then, we can obtain approximated control policy from $\mathbf{u}_{approx}(\mathbf{x}) = -R^{-1}B^T \nabla v_{approx}(\mathbf{x})$, and test the corresponding trajectories. These setting will encourage the system to go back to origin, and if our approximated control policy fails this purpose, we can call it disastrous.

Please note that no rational engineers should use function approximation schemes at all for LQR problems. This formulation also ignores any delays, which is an important source of difficulty for unstable systems in practice; thus, unstable systems are as easily controlled as stable ones.

Value function approach may yield disastrous behavior when the original dynamical system is unstable AND the number of function approximators is limited

We used a 1D spring-mass-like system, $A = [p, 1; -1, 0]$, $B = [0, 1]^T$, $Q = diag([1, 1]^T)$. p is a variable controlling stability of passive dynamics. If $p > 0$, the system is unstable. We use moving least squares base functions to approximate the base functions.

We solve the problem with sufficient basis functions (101×101) (Fig. 6.1c); the simulated trajectory is identical to that of ground truth. However, with a moderate number of basis functions, (11×11 grid over $[-2, 2] \times [-2, 2]$)(Fig. 6.1b), even the value function seems to

be close to the real solution (Fig.6.1g,6.1h), the simulated trajectory is distorted since the gradient and Hessian of value functions bring error into trajectory, and the controller fails to stabilize the system to origin.

On the other hand, if the system is not unstable, for example $p = 0$, the low density of base function will still be able to stabilize the system at origin (Fig. 6.1d,6.1e,6.1f), although the trajectory slightly differs from the theoretical one. In this case, the errors are not accumulated enough to cause disaster, and the controller can still push the system back to origin.

This means dynamical systems with inherited instability will be more sensitive to deficiencies of value function and may be more apt to yield disastrous results. This may be solved either by better form of function approximator (for example, polynomial in $v(x)$ space), or sufficient number of base functions. However, it is difficult to require too many base functions in problems with higher dimensionalities.

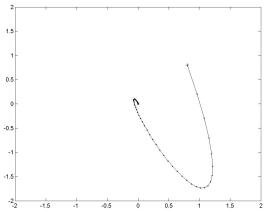
In this specific problem, normalized gaussian bases perform better than moving least square bases by being able to stabilize the system even if the system is unstable. This is due to better smoothness of normalized gaussians, and we will show that this is due to where we place them on an 11×11 grid, where with one of node is the origin itself.

This means for dynamical systems with instability, value function schemes are likely to require more base functions and they make the value function scheme prohibitive in problems with higher dimensionalities.

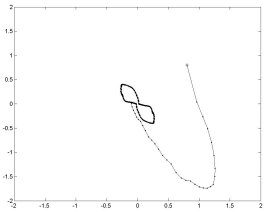
Value function approach may yield disastrous behavior when the original dynamical system is unstable AND function approximators are not used correctly

When the density of basis functions is low, the placement of base functions also matters. Let's start from our "better" base function- normalized gaussians and use the same linear dynamical system as in the previous section.

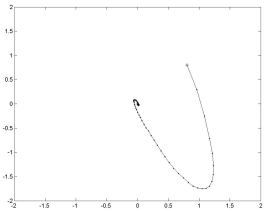
With base functions placed on 11×11 grid, the controller for unstable dynamical systems can still stabilize the system. If we increase the base function to 21×11 and set the diagonal of covariance matrices as an identical matrix, the controller for the unstable system fails



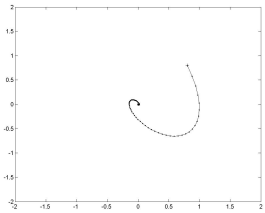
(a) Theoretical - unstable



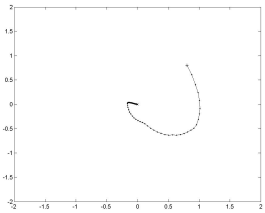
(b) (11×11) - unstable



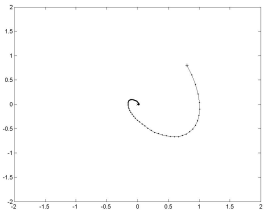
(c) (101×101) - unstable



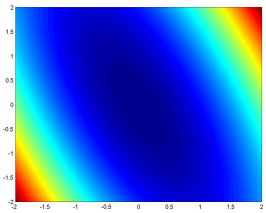
(d) Theoretical - $p = 0$



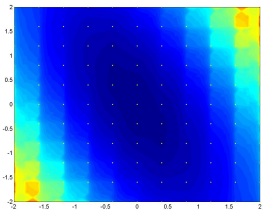
(e) (11×11) - $p = 0$



(f) (101×101) - $p = 0$



(g) Optimal value function - unstable



(h) Approximated value function - unstable

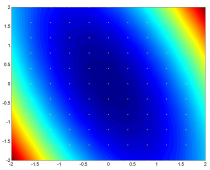
Figure 6.1: Combined impact of insufficient base functions and instability of passive dynamics: Results shows the results obtained by approximating the value function by LQR solution. When passive dynamics is unstable, a lower density of base functions(11× 11) cannot yield an acceptable trajectory and fails to stabilize the system to origin. If the density or number of base functions increases, the control policy will converge to the analytical LQR solution. However, if the passive dynamics is not unstable (p=0), a lower density of base functions(11× 11) can yield an acceptable trajectory and stabilize the system to origin, although it is slightly difference from the outcome of analytical solution of LQR. 6.1g further demonstrates the difference between theoretical value function, and approximated value function and we can see the value function approximation is not bad; and the problem lies in the derivatives of value functions.

even if we have more base functions (Fig. 6.2e,6.2f). It is not surprised that if we put 200 base functions randomly, the controller for unstable system also fails (Fig. 6.2i,6.2j). Ironically, for a stable system with $p = 0$, the optimal controller based on approximated base function can recover from those errors in value function and stabilize the system to origin.

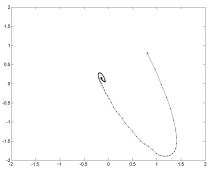
Thus, when using approximated value function to deal with unstable dynamical systems, in order to avoid unexpected consequences, one needs to set base functions in a good way, which is hard in real applications when we are trying to solve this problem automatically. This fact underscores the importance of basis function adaptation, both the gradient-descend method and the heuristic method described in this thesis. All these methods can make basis function adjusting to the real value function well, but none of these basis function adaption methods can guarantee the basis functions will always be perfect. In certain problems such as acrobot, which are extremely sensitive to errors in value function, automatic value function approaches may easily fail to generate an acceptable control policy.

Approximating value function in desirability space further limits flexibility of function approximators

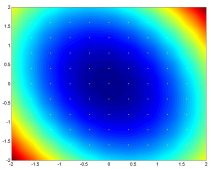
We just illustrated that the actual control policy can be very sensitive for certain function approximators. In the above examples, one way to improve it is to approximate value function as polynomials, i.e. $v(x) = x^T Vx/2$. However, since we are considering Bellman equation in the $z(x) = \exp(-v(x))$ space rather than $v(x)$ space, quadratic function would become a gaussian base function in $z(x)$ space. Directly updating coefficients in polynomials of $v(x)$ space would become a nonlinear problem in the $z(x)$ space. This will bring complications, and we lack sufficient good ways to directly solve for those coefficients. Moreover, in general cases where the problem is no longer linear, we do not know what kind of polynomials we need. Thus the fact that we are working in the desirability function space limits our choice of base functions and exposes our approaches to the complications demonstrated above.



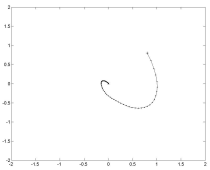
(a) $v(x)$ - unstable



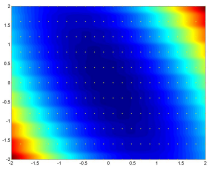
(b) Trajectory - unstable



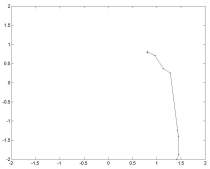
(c) $v(x)$ - $p = 0$



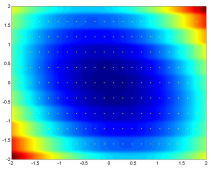
(d) Trajectory - $p = 0$



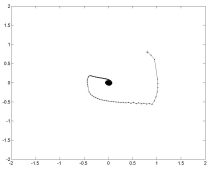
(e) $v(x)$ - unstable



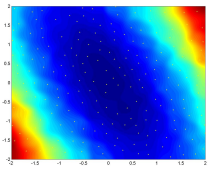
(f) Trajectory - unstable



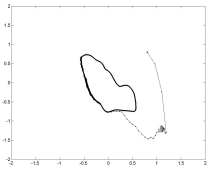
(g) $v(x)$ - $p = 0$



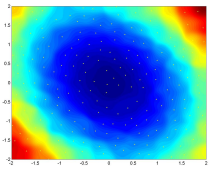
(h) Trajectory - $p = 0$



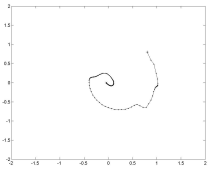
(i) $v(x)$ - unstable



(j) Trajectory - unstable



(k) $v(x)$ - $p = 0$



(l) Trajectory - $p = 0$

Figure 6.2: Misusing basis functions leads to unexpected outcomes in problems with unstable passive dynamics. The first two columns show the value function and simulated trajectory of unstable system and instability is erased in the second two columns. The first row shows the results for normalized gaussian base functions with 11×11 base functions on grid. If we increase the grid to 21×11 with unadjusted covariance matrices, the simulated results are blurred and the controller cannot stabilize the system for unstable passive dynamics.. Similarly, with 200 random base functions, the controller cannot stabilize the system for unstable passive dynamics. These shows that value function approximation maybe blurred if functions are not set "properly" and dynamical systems are hard to control if their passive dynamics are unstable.

Summary

In this section, we demonstrated that even if we solve the Bellman equation perfectly in the simplest problems in optimal control-LQR, we may still fail to control such a dynamical

system with certain function approximators. This behavior is problem-specific. Certain problems, which tends to amplify errors in control policy, may be very sensitive to flaws in function approximators and make value function approximation schemes seems unlikely to succeed. Other problems (e.g. examples in previous chapters), which are more stable, are most easily solved with few base functions.

6.2.4 Discussion

Using value function schemes to compute a global time-invariant policy $u(x)$ is difficult not only because of the curse of dimensionality, but also because of its inability to correct the errors it produces.

The discussions in this section do not rule out possibilities for value function schemes to control difficult problems. However, to solve such problems, one may have to manually tune the parameterization of value functions, and such an approach cannot scale to general cases.

As opposed to manually tuning, we chose to design automatic basis function adaptation, which is harder than manually tuning but is the right direction towards automatic control design in real applications. In certain problems with stability and low dimensionality, value function approximation with automatic basis function adaptation will work well. There are sufficient applications for those simpler problems in design, most likely in the field of navigating service robots. On harder problems similar to acrobot, naive value function may be not economical, and we will explore ways to enhance it.

6.3 Possible ways to apply LMDP in more applications

Previously we have discussed value function schemes' difficulty with certain problems. Unfortunately, many problems arising from applications such as walking involves substantial instability and very high dimensionality. Moreover, we cannot conclude which problems are easier to solve using value function schemes. Despite difficulties, there are still some potential ways to apply LMDP global methods in more applications.

1. Hierarchy control with subproblem suitable for LMDP

Since certain problems can easily be solved by function approximation schemes, we can limit its application to certain problems with low dimensionality and less instability. The example of this category can be a point-mass. Using hierarchy control, this may be a subproblem of a practical task, such as navigating in a complicated environment.

This field involves substantial understanding of individual systems, and hierarchy control is not a new idea in theories and applications. Therefore, we will not expand on this idea in this thesis, although it is one promising way to apply LMDP in broader contexts.

2. Combining with other methods to address stability

   We can combine other approaches with LMDP global value function approximation methods to improve stability.

   As mentioned before, for cart-pole, it is possible to let LMDP controller swing up the pole and use LQR controller to stabilize the pole at the top. This will only improve the control policy near the target and the combination of LQR with LMDP is likely not scalable to general problems, which involves careful planning of trajectories.

   Another option is trajectory-based methods. For example, it has been shown that the MPC controller can swing-up and stabilize the acrobot. In the next chapter, we will discuss combining MPC controllers with LMDP in depth.

Chapter 7

MODEL PREDICTIVE CONTROL AND LMDP-BASED VALUE FUNCTION APPROXIMATION

(This chapter contains part of [59])

7.1 Background

7.1.1 Motivation

While reinforcement learning global methods to find globally defined cost-to-go functions and control policies may be guaranteed to converge in the limit of a large number of basis functions, neurons, or a high-degree of polynomials, in practice, these approximate methods must handle the finite error in representation that they generate. These shortcomings motivate us to find novel, practical methods of exploiting a globally defined cost-to-go function and control policy.

The alternative to global reinforcement learning algorithms is to use trajectory optimization, focusing the limited computational resources on the most relevant states. Model-predictive control (MPC) [42] [43] is an approach that applies, in real time, trajectory optimization to the current state of the system. Computing the policy in real time makes MPC very reactive, allowing it to generate a control law for any state of the system. However, this reliance on online computation is also the main limiting factor for MPC — computational resources are invariably limited, and online trajectory optimization for high-DOF, complex systems is a taxing task as well. The main parameter used to regulate the computational demand of MPC is the length of the planning horizon. Compare to the search depth in classical AI applications (e.g., how many steps ahead a chess software considers before choosing a move), the choice of a particular horizon affects both the computational load (shorter horizons are easier to compute) and the level of performance (planning over shorter horizons result in less-optimal behavior).

7.1.2 Related work

Finding a good way to combine the advantages of local and global dynamic programming is not new. In particular Atkeson and colleagues have been actively exploring this area for the past decade [4][5]. Similar results can also found in real-time dynamical programming [8], heuristics for the go game [21] and the heuristics approach to solve shortest path problem [19].

In some domains we may expect the optimal system to converge to a small volume of state space, e.g., when the optimal behavior leads to some target state or a limit cycle. In these cases, we may compute the accurate value function in that small region, and use it as a final cost for the MPC computation. This scheme is called *infinite-horizon MPC* [12] [22] [18], and it is guaranteed to produce the optimal behavior as long as the trajectory terminates in an area where the terminal cost accurately reflects the value function. Unfortunately, in high-DOF domains the system may find itself very far from the asymptotically-optimal region of state space. For example, consider a walking humanoid that has fallen on the ground; in this case, knowing the value function around the optimal gait does not provide enough information to figure out how to get up.

7.1.3 Contribution

The main contribution of this section is to demonstrate the results of interfacing model predictive control with LMDP based value function approximation. We obtain a global final cost by using the cost-to-go function generated by solving a linearly-solvable Markov Decision Process (LMDP), and we also warm-start MPC's policy search with the control policy also from solving the LMDP (both control policy and value function can be obtained in the same solve). The LMDP is an optimization problem explicitly related to the trajectory optimization problem that MPC solves, but it assumes a particular structure in the mathematical formulation that differs from the MPC optimization problem. This structure makes the problem exactly solvable in theory, but discretization of the problem for numerical solution introduces finite error. We present results in which we use the solution of the LMDP as a final cost for MPC. With this value function approximation as a final cost

and an extra "hint" from the computed LMDP policy, we are able to shorten the horizon of MPC while maintaining solution quality even on underactuated problems; the cart-pole and the acrobot.

This approach means different things in different fields. In the area of Bellman equation-based methods, this approach presents a novel way to recover complicated behaviors even from the coarse approximation of the cost-to-go function. In the area of MPC, this method serves as an automatic way to summarize the final cost.

The results are still limited in several ways: (1) For problems that are already solved by MPC, the actual performance gain of hybrid methods may not be significant enough. (2) It is limited by the shape of function approximators, which are limited in LMDP's case. (3) Interfacing two methods increases error in value function. Despite these limitations, interfacing two methods may still affect certain actual applications and make LMDP value function approximation methods valuable.

7.2 Model Predictive Control and LMDP

7.2.1 Model Predictive Control

Model predictive control, also known as receding horizon control, solves the optimal control problem to a user-chosen time horizon. That is, from the given position in state space, it searches for an optimal policy in the neighborhood of the current position to some finite horizon. It applies the first control of the computed optimal policy, and then it recomputes a new optimal trajectory from the new state (having not completed the old planned policy). Computationally, MPC solves the discrete-time problem

$$\min_{\mathbf{u}_{i,\ldots,i+N-1}} \left[\Sigma_{k=i}^{i+N-1} l(\mathbf{x}_k, \mathbf{u}_k) + v_F(\mathbf{x}_{i+N}) \right], \tag{7.1}$$

where $l(\mathbf{x}, \mathbf{u})$ represents the running cost, v_F is the final cost, and the states $\mathbf{x}_k$ and controls $\mathbf{u}_k$ are constrained by the discretized dynamics $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k, h)$. Here h represents the time step. MPC applies only $\mathbf{u}_i$ to the system and then resolves the problem at the next time step. It is outlined in the following algorithm.

for $i = 0$ to the end of simulation **do**

Obtain intial control $\mathbf{u}_k^{init}$ from interpolation or other warmstarts

Find optimal policy $\mathbf{u}_{i,\ldots,i+N-1}$ from (7.1)

Apply only $\mathbf{u}_i$ to dynamical system, $x_{i+1} = f(x_i, u_i)$

end for

There are multiple trajectory based optimization methods available. However, we utilize exclusively the iLQG method [42] [43][52] in which the trajectory optimizer assumes a locally linear model of the dynamics and maintains a local value function of second-order accuracy.

7.2.2 Model Predictive Control Revisited

Constantly solving for and updating the optimal policy enables cheap trajectory based optimal control, and it is able to adjust to a variety of disturbances. The primary computational bottleneck in preventing MPC from running in real time, especially for high degree of freedom systems, is the need to plan far in advance at nearly every time step. Therefore, it is desirable to add something on top of MPC that permits the same quality of policy at a shortened horizon, or, equivalently, a better policy with the same horizon.

From equation 7.1, one sees that MPC sums the costs up through the end of the finite horizon ($i = 1, ..., N-1$) and then anything that would define the system's behavior from time N to infinity is lumped into the final cost $v_F(\mathbf{x}_N)$. That is, the final cost should give information about the desirability of any state which the finite-horizon planner is able to reach. The more accurately the final cost describes the desirability of the state AND which directions in state-space increase this desirability at the end of MPC's time horizon, the less the method relies on solving the equation 7.1. If the MPC horizon is fixed by limited computational resources, we are still able to obtain a better policy with a more descriptive final cost.

To further characterize a desirable final cost function, recall that MPC is a trajectory-based method of approximating and computing a solution to the Bellman equation

$$\min_{\mathbf{u}} l(\mathbf{x}, \mathbf{u}) + V(f(\mathbf{x}, \mathbf{u})) \tag{7.2}$$

where $V(f(\mathbf{x}, \mathbf{u})$ is the value function evaluated at the state $(\mathbf{x}, \mathbf{u})$ under the dynamics f

and $l(\mathbf{x}, \mathbf{u})$ is the one-step cost function. Note that, the running cost $l(\mathbf{x}, \mathbf{u})$ defines the task, and it also determines the optimal value function.

Putting the Bellman equation and the MPC framework together, we notice that the final cost in MPC $v_F(\mathbf{x}_N)$ is, in the limit an infinite horizon, the optimal value function evaluated at the last state in the finite horizon trajectory. In the next sections, therefore we will give the value function approximated by LMDP to MPC as a final cost, and if the approximation is accurate (or even if it is inaccurate but useful information is still attainable), we may be able to achieve our aim of reducing the need for a long horizon for MPC.

A second aspect of trajectory-based MPC that makes it fast, besides solving the local problem to only a finite horizon, is knowing that if we only employ the first control of a computed optimal policy before recomputing a new trajectory from our new location, we expect that the optimal policy has not changed drastically from that of the previous time step. Thus, MPC uses the entire optimal policy computed at the last time step to start the search for the optimal policy at the new position. One can easily see that this "warm start" of the policy search permits real-time solution at every time step. However, it is also the case that it may constrain the solver in a local minimum; it does not permit the system to look for solutions that are perhaps better but that are too different from the solution it already has in hand. In the next section, we will also discuss using LMDP to enhance the warm-start of MPC.

7.2.3 Using an LMDP solution for value function approximation

We have discussed several methods to solve LMDP problems and obtain both desirability function $z(x)$ and optimal control policy $u^*(x)$. To simplify our discussion, we focus on the aggregation method. After obtaining u^* in a discrete state space and, using the cluster positions, we are able to define desirable locations for the continuous system. In MPC, we set final cost as $v_F(x) = -\log(z(x))$ and set the warm start of new states as LMDP's control policy $u^{init}_{i+N-1} = \mathbf{u}^*(x^{init}_{i+N-1})$.

The probabilistic framework introduces noise that is not present in MPC, so we are not approximating the value function of MPC. However, all that we are interested in is

supplying to MPC is an approximation to the value function that will help nudge it in the right direction.

7.3 Numerical results

We now turn to using the LMDP solution for value function approximation to find a method that works well for the more difficult systems the acrobot.

Building MPC on aggregation methods improved both methods

Solving the LMDP by aggregation methods as described previously, we obtain a global value function approximation. However, using this value function as a final cost fails to provide successful results in achieving either swing up or stabilization in the underactuated system acrobot. Also, recall that in solving the LMDP, we also obtain the LMDP optimal policy. Warm-starting MPC with this control policy and not providing a final cost also fails to improve MPC. Contrary to intuition, when both the control policy and the value function are provided to MPC, the required horizon of planning for MPC can be reduced while still achieving the control tasks.

Figure 7.1b shows the percentage of 100 trials (initializing simulation from a random initial state) in which the controller is able to swing the pendulum to vertical and in which the controller is able to stabilize the pendulum after swing-up. In the acrobot problem, we need a 800 ms time horizon to swing up *and* stabilize the acrobot. This is in a sharp contrast to the MPC method itself, which needs a 1500 ms horizon to obtain similar results on both problems. Thus using both the value function and the control policy derived from LMDP is a way to effectively shrink the horizon MPC needed. Using the value function alone is detrimental, and using the control policy provides little benefit. In both underactuated problems, LMDP helps MPC to quickly discover the target, although too short a horizon limits the hybrid method's ability to stabilize the system. On the other hand, when the horizon is too long, the hybrid method often fails to achieve the task before the simulation is terminated due to the decaying effect of the policy warm-start. We demonstrate an example of those trials in 7.2

This combination in general cannot effectively lower the cost accumulated, since LMDP is actually solving a different optimal control problem. Yet the result for success rates show that the combination of LMDP and MPC would be helpful in solving practical problems.

These results were obtained solving for the value function on a regular grid with approximately 10,000 bases; such grids are too coarse for aggregation methods alone to be effective. The computation time of the LMDP is around 10 seconds with 6 threads and utilizing the Mujoco solver[56]. The solution of the LMDP is done offline. Extracting the control policy and the value function online after the LMDP solve is cheap given proper implementation, so the computational efficiency of MPC can be improved.

We can obtain much better results in single pendulum and double pendulum, and similar results in cart-pole. In double pendulum, with 100ms time horizon in MPC and the solution in LMDP, the double pendulum can be swung up, while MPC alone needs a 1500 ms horizon. However, this is not a surprise since this problem has already be solved roughly in LMDP. In cart-pole, MPC alone needs a 1500 ms horizon, while the combination method needs a 500 ms. We can see that the more difficult the problem becomes, the longer horizon we need in the combinatorial method.

First exit formulation

The flexibility of the LMDP statement of the optimal control problem permits us to solve for a value function and control policy under a first exit formulation as well as in the average cost formulation we used previously. Interestingly, we find that in the first exit formulation, the aggregation method may be used to generate an acceptable control policy even when the state costs are not finely-tuned to describe the control problem. That is, the solution is foremost designed to obtain the goal state, and the state cost is of secondary concern. This thinking puts less of a burden on the control engineer.

Figure 7.1d presents results in which we solved the LMDP for the acrobot under the first exit formulation with the upward position is assigned as the target. In one case of the LMDP the state cost is defined to be constant and the control cost is made very small; such a combination would yield a practically uncontrollable system under other formulations. In

the other case of solution of the LMDP in the first exit formulation, we retain the well-tuned state cost. The percentage of trials in which swing up and stabilization are achieved is nearly equivalent to the case of the solution of the LMDP in an average cost formulation, but the first exit formulation is not reliant on a hand-tuned state cost.

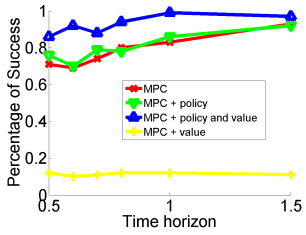
Please do note that first exit formulation theoretically requires MPC to terminate at the exit state, which is not consistent with our MPC formulation with constant horizon. However, heuristically it is helpful in this specific case for swing up, and then MPC can be replaced by a simple feedback control law near the terminal state to maintain equilibrium for some types of control problems.

Resolution of the value function

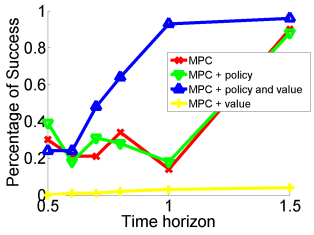
By using more bases in the LMDP solver, the accuracy of the value function will be increased. However, for the acrobot, we can still reduce the MPC planning horizon with as few as 3000 bases, and the addition of more bases does not significantly decrease the needed horizon. It is particularly encouraging that the computational expense of a highly accurate LMDP solution is not required to provide benefit to MPC.

Selection of basis function

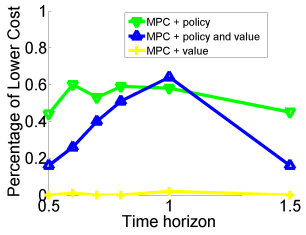
If we do not place base function on grids, LMDP will hardly improve MPC. In parallel with MPC and LMDP combination, we also explored the final cost generated by fitting value function with the information from MPC, and it gives even worse results in acrobot and cart-pole. One observation is that in many failed cases, control policy is not 0 in origin($\nabla v(0) \neq 0$ and $u(0) \neq 0$), which clearly may bring complications. Although MPC provides more tolerance of flaws in function approximation, MPC cannot fix all problems brought by function approximators. Which base functions may robustly improve results of MPC remains an open question.



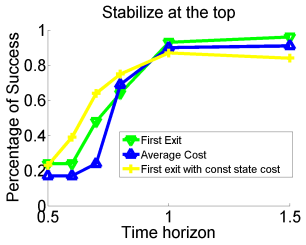
(a) Reach the top



(b) Swing up and stablize



(c) Percentage of lower cost



(d) Compare with first exit

Figure 7.1: Results for swing up and stabilization on acrobot. The success rate is measured as follows: we simulate the dynamical systems from random initial states and find how many times we achieved desired tasks. The success rate is the chance of achieving desired states among all trials and it demonstrates the quality of the global controller we constructed. (a) shows the success rate of swing up, (b) shows the success rate of swing-up and stabilization. Combining MPC with LMDP's value function and policy do shorten the time-horizon of MPC(c) Shows the percentage of lower running cost. In(a-c) Red "x" represents MPC alone, green triangle represents MPC warm-started with control policy, yellow "+" from using LMDP's value function only, and blue triangle represents using both information value function and policy warm-start. (d) Success rate of swing up and stabilization on the acrobot in different formulations. Green triangle represents the first exit LMDP tuned state cost, yellow "+" represents the first exit LMDP constant state cost, and green triangle represents the average cost LMDP. It shows that differences between the formulations are minor.

7.4 Conclusions and future work

7.4.1 Shortening MPC's horizon automatically

The planning horizon required by MPC for effective control of a system is highly dependent upon the descriptiveness of the final cost function. Better methods of function approximation in the highly nonlinear and difficult high-dimensional spaces of complex systems

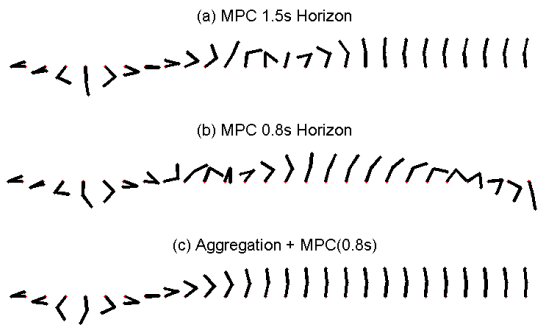


Figure 7.2: Demonstration of the actual movement of acrobot under different approaches.

need to be used to approximate the value function, if any benefit to MPC is to be seen. Our LMDP results show that exploiting both a global value function and optimal policy is required to provide substantial benefit to MPC, which is mostly due to fixing certain errors in the gradient of value functions.

7.4.2 Applying global methods to more problems

These results show that MPC is valuable to the global LMDP method. In our results, we obtain only a very rough estimate of cost-to-go function, which is far from the accuracy required to achieve the desired control task. Yet, even a coarse global solution still contains information about the solution of the optimal control problem. Online adaptation can exploit this helpful yet inaccurate information. Instead of increasing the accuracy of a global method solution by introducing more basis functions or neurons, it may be more practical to include online correction.

Online planning can also expand a global method's applicability in real-time applications. First, the prediction generated by planning provides a last minute safety check of the control policy. Second, it decreases the demand on the limited computational power installed in robots. Rather than storing and computing solutions to the global Bellman equation locally on robots, we have the freedom to compute them on a remote server and send only the planned trajectories.

Our approach of generating a final cost by aggregation methods still suffers from the

approximation structure we have employed. It makes the final cost non-convex and requires warm starting to perform well, and LMDP's control policy serves as a good warm-start for MPC.

Chapter 8

SUMMARY AND DISCUSSIONS

8.1 Summary

In this dissertation, we aim to contribute to answering the big question: how to automate control design in complicated control problems such as locomotion. As a means to provide alternatives for other existing methods, we focus on the value function approximation method in the global method of optimal control in the Linearly-solvable Markov Decision Process (LMDP) to provide alternatives for other existing methods.

First, we designed scalable, numerical methods that are tailored to solve LMDP problem efficiently and reliably. LMDP problems can be reduced to computing the principle eigenfunction of a linear operator. We provide two new frameworks, moving least squares approximation (Chap. 4) and aggregation methods (Chap. 5), to solve for value functions in LMDP problems. The moving least squares approximation method uses linear basis functions constructed by the moving least square process to approximate value function. The aggregation methods utilize aggregation and deaggregation functions to discretize the continuous space MDP in LMDP problems. Both methods avoid matrix factorization and take advantage of sparsity by using efficient iterative solvers. Adaptive schemes for basis placement are developed to provide higher resolution at the regions of state space that are more frequently visited. . By combining with other improvements discussed in Chap. 6 and by applying the preexisting Gaussian basis function approach, these methods in the LMDP framework are able to verify the benefits of LMDP framework, provide scalable efficient solvers for value functions, and can be applied in a sizable range of applications.

Second, we discussed the limitations of the above methods in Chap. 6. These methods alone are less likely to solve certain control problems, such as acrobat or walking, due to the general difficulty in applying value function methods to problems with instability. Those problems require a more delicate design in basis functions; any minor flaws in the

choice of basis function may lead to unexpected results. This fact reinstates the known, nontrivial relationships between errors in value function and control policy, and it limits the automatic control design based on value function approaches. In regard to our efficient LMDP solvers in more applications, we should carefully design basis functions, find suitable stable subproblems for LMDP, or exploit alternative approaches to derive a control policy from value function.

Finally, we demonstrated one way to derive acceptable control policy from the approximate value function solved by LMDP methods: through the application of the Model Predictive Control (MPC)). We used the aggregation methods discussed above to reinforce the MPC method. The combination method not only shortens the MPCs time horizon but also expands LMDP methods on problems LMDP originally cannot solve. We tested this method on difficult benchmark problems such as acrobot, which most value function approximation approaches. have difficulty solving. These results are illustrative of the potential for exploring more robust control design based on suboptimal approximation of value function and suggest that MPC is such an option.

We will further discuss the results from different perspectives in the following sections.

8.2 Applications for value function methods in Linearly solvable framework

The benefits of linearly solvable framework are appealing: By solving Bellman equations linearly, more efficient algorithms can be developed and some complicated problems may be solved using an optimal control approach. In local methods LMDP can be reduced to a regular trajectory optimization problem with modified cost function, so we cannot expect much gain in efficiency. The improvement in actual computation time is significant in global methods, and we have developed multiple methods to exploit such benefits.

However, there are multiple obstacles to using linearly solvable framework global methods to provide accurate solution for the value function. First, the assumption of LMDP limits the freedom of noise assumption and the selection of control cost, so LMDP models are normally already an oversimplified model. Second, a linearly solvable framework limits the choice of function approximations, so although it gives faster algorithms, it may fail to provide accurate enough solutions of value function.

To apply our LMDP global methods to a more difficult problem, acrobot, we demonstrated that combining MPC with LMDP aggregation methods can overcome difficulties mentioned above, i.e. MPC helps correct flaws brought by both incorrect noise model and base functions. The applications of LMDP are expanded by MPC, which is one way to carefully plan trajectories and enhance robustness. To extend value function methods in linearly-solvable frameworks into more applications, we need to further improve methods similar to MPC to help us extract information from inaccurate value function approximators. Alternatively, we can treat LMDP as a way to quickly generate an initial guess of value function and use other value function methods to enhance it.

Formulations not involving $z(\mathbf{x})$ in LMDP may promise more directions, since approximating control policy $u(\mathbf{x})$ or other quantities is likely to be more flexible and generate a more robust control signal in a time horizon.

8.3 Base function in value function approximation

There is no reason to assume that base function selection is a trivial problem in the field of function approximations. It is true that in certain problems, such as car-on-the-hill or other 2D problems, control policy is not sensitive to which base functions and parameters are used. However, we have shown multiple cases where base function selection does matter. An unstable linear system can be sensitive to base function selection. In acrobot, using LMDP solution in random base functions may hardly improve MPC. Similarly, even directly fitting value function may still fail to improve MPC. The selection of base functions plays a significant role in the success of value function approximation, both in stand-alone value function approaches and combinatorial methods with MPC.

8.4 Combining MPC with value function approximation

Combining MPC and value function approximation approaches may be more important than expected.

From value function approximations' perspective, MPC is crucial for bringing value function approximation into more applications. MPC brings extra robustness thereby overcoming certain limitations of value function. Moreover, adding MPC to value function

schemes can also be an economical choice. Moreover, it may also improve the architecture of actual programs by keeping the more computationally expensive value function methods on the cloud.

From MPC's perspective, our results are preliminary yet promising. Combining LMDP can shorten MPC's horizon and in the long run may shorten the running time of MPC and bring MPC to more real-time applications. It also has the potential to free engineers from designing final costs that are tailored to individual problems.

8.5 Future direction

While LMDP value function approximation methods demonstrated applicability to certain problems and exposed limitations, there are still several ways to further improve methods and extend LMDP's impact.

First, more research should be done on hierarchy control based on LMDP value function approaches. Trajectory methods generally may converge to local minimum and fail to navigate in maze-like environments. Global methods may be helpful in such circumstances and be applied to areas such as service robots.

Second, understanding the behavior of value function for certain dynamical systems is very important. As more and more research is conducted on a few dynamical systems such as hand or humanoid, understanding the properties of their value functions and developing corresponding feature functions may be very important because it can help global methods to supplement local methods.

Moreover, more insights into the robustness of MPC are needed. These will not only benefit the area of MPC, but also benefit value function approximation methods.

Finally, more attention should be paid to other frameworks arising from linearly-solvable framework. Certain components of methods derived in this thesis may be transferred to broader contexts, while certain lessons can be helpful in developing suitable numerical methods for those frameworks.

BIBLIOGRAPHY

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, June 2010.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, page 1, 2007.

[3] Nils A Andersen, Lars Skovgaard, and Ole Ravn. Control of an under actuated unstable nonlinear object. In *Experimental Robotics VII*, pages 481–490. Springer, 2001.

[4] C.G. Atkeson et al. Using local trajectory optimizers to speed up global optimization in dynamic programming. *Advances in neural information processing systems*, pages 663–663, 1994.

[5] C.G. Atkeson and J. Morimoto. Nonparametric representation of policies and value functions: A trajectory-based approach. *Advances in neural information processing systems*, page 16431650, 2003.

[6] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning. *Artificial intelligence review*, 11(1-5):11–73, 1997.

[7] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1-5):75–113, 1997.

[8] A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[9] Ted Belytschko, Yury Krongauz, Daniel Organ, Mark Fleming, and Petr Krysl. Meshless methods: an overview and recent developments. *Computer methods in applied mechanics and engineering*, 139(1):3–47, 1996.

[10] Dimitri Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, Belmont Mass., 2nd ed. edition, 2000.

[11] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC Press, 2010.

[12] H. Chen and F. Allgöwer. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1217, 1998.

[13] Graziano Chesi and Didier Henrion. Guest editorial: Special issue on positive polynomials in control. *Automatic Control, IEEE Transactions on*, 54(5):935–936, 2009.

[14] William S Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836, 1979.

[15] William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[16] Rémi Coulom. Reinforcement learning using neural networks, with applications to motor control. 2002.

[17] Marco da Silva, Frédo Durand, and Jovan Popović. Linear bellman combination for control of character animation. In *Acm transactions on graphics (tog)*, volume 28, page 82. ACM, 2009.

[18] T. Erez, Y. Tassa, and E. Todorov. Infinite horizon model predictive control for nonlinear periodic tasks. *Manuscript under review*, 2011.

[19] M. Fink. Online learning of search heuristics. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS 2007)*, pages 114–122, 2007.

[20] Tamar Flash and Neville Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *The journal of Neuroscience*, 5(7):1688–1703, 1985.

[21] S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.

[22] B. Hu and A. Linnemann. Toward infinite-horizon optimality in nonlinear model predictive control. *Automatic Control, IEEE Transactions on*, 47(4):679–682, 2002.

[23] D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. Elsevier, 1970.

[24] Zachary Jarvis-Wloszek, Ryan Feeley, Weehong Tan, Kunpeng Sun, and Andrew Packard. Some controls applications of sum of squares programming. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 5, pages 4676–4681. IEEE, 2003.

[25] H.J. Kappen. Linear theory for control of nonlinear stochastic systems. *Physical review letters*, 95(20):200201.

[26] P. Kulchenko and E. Todorov. First-exit model predictive control of fast discontinuous dynamics: Application to ball bouncing. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2144–2151. IEEE, 2011.

[27] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. Motion fields for interactive character locomotion. In *ACM Transactions on Graphics (TOG)*, volume 29, page 138. ACM, 2010.

[28] C Karen Liu. Dextrous manipulation from a grasping pose. In *ACM Transactions on Graphics (TOG)*, volume 28, page 59. ACM, 2009.

[29] James Lu and David L Darmofal. Higher-dimensional integration with gaussian weight for applications in probabilistic design. *SIAM Journal on Scientific Computing*, 26(2):613–624, 2004.

[30] Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. Control design along trajectories with sums of squares programming. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[31] Richard C Merton. *Optimum consumption and portfolio rules in a continuous-time model*. MIT, 1970.

[32] Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[33] Norman S Nise. *Control systems engineering*, volume 5. Wiley Hoboken, NJ, 2008.

[34] Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, Citeseer, 2000.

[35] Stephen Prajna, Antonis Papachristodoulou, and Pablo A Parrilo. Introducing sostools: A general purpose sum of squares programming solver. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 1, pages 741–746. IEEE, 2002.

[36] S. Singh, T. Jaakkola, and M.I. Jordan. Reinforcement learning with soft state aggregation. *Advances in neural information processing systems*, pages 361–368, 1995.

[37] Mark W Spong. The swing up control problem for the acrobot. *Control Systems, IEEE*, 15(1):49–55, 1995.

[38] Robert F. Stengel. *Optimal Control and Estimation*. Dover Publications, September 1994.

[39] A Stroud. *Approximate calculation of multiple integrals*. Prentice-Hall, Englewood Cliffs N.J., 1971.

[40] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12(22), 2000.

[41] R.S Sutton and A.G Barto. *Reinforcement learning : an introduction*. MIT Press, Cambridge Mass., 1998.

[42] Y. Tassa, T. Erez, and W. Smart. Receding horizon differential dynamic programming. *Advances in Neural Information Processing Systems*, 20:1465–1472, 2008.

[43] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. IROS, 2012.

[44] Yuval Tassa and Tom Erez. Least squares solutions of the HJB equation with neural network value-function approximators. *IEEE Transactions on Neural Networks*, 18(4):1031–1041, July 2007.

[45] Yuval Tassa and Emo Todorov. Stochastic complementarity for local control of discontinuous dynamics. In *Proceedings of Robotics: Science and Systems (RSS)*. Citeseer, 2010.

[46] R. Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for 6.832. 2008.

[47] E. Todorov. Linearly-solvable markov decision problems. *Advances in neural information processing systems*, 19:1369, 2007.

[48] E. Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009.

[49] E. Todorov. Eigenfunction approximation methods for linearly-solvable optimal control problems. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pages 161–168. IEEE, 2009.

[50] E. Todorov. Finding the most likely trajectories of optimally-controlled stochastic systems. In *World Congress of the International Federation of Automatic Control (IFAC)*, 2011.

[51] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. *Under Review*.

[52] E. Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306, Portland, OR, USA, 2005.

[53] Emanuel Todorov. Optimality principles in sensorimotor control. *Nature Neuroscience*, 7(9):907–915, September 2004.

[54] Emanuel Todorov. Optimal control theory. *Bayesian brain: probabilistic approaches to neural coding*, pages 269–298, 2006.

[55] Emanuel Todorov. Compositionality of optimal control laws. *Advances in Neural Information Processing Systems*, 22:1856–1864, 2009.

[56] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[57] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*. Number 50. Society for Industrial and Applied Mathematics, 1997.

[58] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. In *ACM Transactions on Graphics (TOG)*, volume 28, page 60. ACM, 2009.

[59] M. Zhong, M. Johnson, T. Erez, Y. Tassa, and E. Todorov. Value function approximation and model predictive control. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*. IEEE, 2013.

[60] M. Zhong and E. Todorov. Aggregation methods for lineary-solvable markov decision process. In *World Congress of the International Federation of Automatic Control (IFAC)*, 2011.

[61] M. Zhong and E. Todorov. Moving least-squares approximations for linearly-solvable stochastic optimal control problems. *Journal of Control Theory and Applications*, 9(3):451–463, 2011.

Appendix A

DETAILS OF TEST PROBLEMS

A.1 Car-on-the-hill

This test problem is adapted from [49] with 2D state space and 1D control space. This dynamical system simulates a point mass (a car) moving along a curve (inverted Gaussian) in the presence of gravity. The control signal is the force acting in the tangential direction. One interesting property of this model is that the continuous dynamics are augmented with the following rule. When the car hits its "walls" at x_{min} or x_{max} , its speed becomes 0. Such a discontinuity cannot be captured by the diffusion model (3.9), yet it can easily be captured by our LMDP [49]. The reason for constructing a model with collisions is that we hope our methods will work for more complex tasks such as locomotion and hand manipulation, where contact phenomena and discontinuity are essential.

This dynamical system is a point mass(a car) moving along a valley-shaped curve(an inverted Gaussian) with the existence of gravity. Thus $\mathbf{x} = [x_p, x_v]^T$, where x_p denotes horizontal position and x_v denotes tangential velocity. The passive dynamics can be defined as

$$\mathbf{a}(\mathbf{x}) = \begin{pmatrix} x_v(1 + s(x_p)^2)^{-1/2} \\ -9.8sgn(x_p)(1 + s(x_p)^{-2})^{-1/2} \end{pmatrix}, \tag{A.1}$$

where $s(x_p) = x_p \exp(-x_p^2/2)$ is the slope of the inverted gaussian "hill" at x_p position. $B(\mathbf{x}) = [0, 1]^T$. Only the velocity of the car is controlled. Magnitude of noise is $\sigma = 3$ while time step is set as h=0.1. If $x_p < x_{min}$, at the next time step $x_p = x_{min}$. If $x_p > x_{max}$, at the next time step $x_p = x_{max}$. In both cases $B = [0; 0]$(controller would not work when "hit the wall"). The time step is $h = 0.1$ (finite-horizon setting uses $h = 0.04$.) Cost functions for each formulation are discussed in the following.

A.1.1 Infinite-horizon average cost formulation

State cost function is defined as

$$\begin{aligned} q(\mathbf{x}) &= 2(2 - \exp((x_p - 1)^2 + (x_v + 1)^2) \\ &\quad - \exp((x_p + 2)^2 + (x_v - 1)^2). \end{aligned} \tag{A.2}$$

It would keep the car passing those two targets ($x_p = -2, 1$ with velocities $x_v = 1, -1$) and obtain a limit cycle behavior.

A.1.2 Discounted cost formulation

State cost function is the same as before. Costs decline in rate of $\exp(-t/\tau)$, where $\tau = 0.5$.

A.1.3 First-exit formulation

State cost and exit cost are defined as

$$q(\mathbf{x}) = 2, q_T(\mathbf{x}) = 0 \tag{A.3}$$

The terminal states are within $[-2.2, 2.6] \times [-.4, .4]$

A.1.4 Finite-horizon formulation

State cost and terminal cost are defined as

$$q(\mathbf{x}) = 2, \tag{A.4}$$

$$q_F(\mathbf{x}) = \begin{cases} 0, & \text{when} \mathbf{x} \in \text{goal states} \\ 100, & \text{when} \mathbf{x} \notin \text{goal states} \end{cases}$$

The goal states are within $[-2.1, 2.7] \times [-.4, .4]$

A.2 Coupling independent masses

This model simulates M masses attached on ideal frictionless springs (or equivalently electrical oscillators), which are dynamically independent. Each mass yields oscillation on any amplitude. Each mass's determinant dynamical behavior can be expressed as

$$\dot{x}_{p,i} = C x_{v,i}, \dot{x}_{v,i} = -C x_{p,i}, \tag{A.5}$$

where $C = 2\pi$ is a constant. The control is in form of a force which would only apply on the velocity. The time step h is 0.01, $\sigma = 1$. Hence, the state space is $2M$ dimensional, with M freedom of control. Thus $\mathbf{a}(\mathbf{x})$ and $B(\mathbf{x})$ can be easily written as a linear function and a constant matrix, respectively.

The state cost function is designed to achieve the goal, which are, (1) fixing the amplitude to 1 and (2) making the (i)th mass be $\pi/2$ phase ahead of the (i+1)th mass. We would like to use the average-cost formulation to solve the problem.

The state cost function is

$$\begin{aligned} q(\mathbf{x}) &= \Sigma_{i=1}^{m} q_a([x_i, v_i]^T) \\ &\quad + \Sigma_{i=1}^{m-1} q_b([x_i, v_i]^T, [x_{i+1}, v_{i+1}]^T). \end{aligned} \tag{A.6}$$

Here, q_a makes $x^2 + v^2$ be a constant 1, i.e.,

$$q_a([x_i, v_i]^T) = 20(\sqrt{x_i^2 + v_i^2} - 1)^2. \tag{A.7}$$

q_b tries to create the $\pi/2$ phase shift

$$\begin{aligned} & q_b([x_i, v_i]^T, [x_{i+1}, v_{i+1}]^T) \\ & = 50 f_\epsilon(x_i^2 + v_i^2) f_\epsilon(x_{i+1}^2 + v_{i+1}^2) f_\theta(\theta_{i+1}, \theta_i) \end{aligned} \tag{A.8}$$

$$\begin{matrix} x_i = \cos(\theta_i) \\ v_i = \sin(\theta_i) \end{matrix}, f_\epsilon(\eta) = \begin{cases} 1 & \eta >= 0.1 \\ 0 & \eta < 0.1 \end{cases} \tag{A.9}$$

$$f_\theta(\theta_{i+1}, \theta_i) = (1 - \cos(\theta_{i+1} - \theta_i - \pi/2)). \tag{A.10}$$

Here, $f_\theta(\theta_{i+1}, \theta_i)$ is a function designed to achieve desired phase shift, and $f_\epsilon(\eta)$ is a function employed to avoid numerical difficulties when $x_i^2 + v_i^2$ is too close to zero (where angles θ_i are not well defined).

A.3 2-link arm and acrobot

The acrobot is a 2-link arm with control torque only applied to the joint between the two parts. The details of this model is well-explained in [16] and [37]. If we can control both

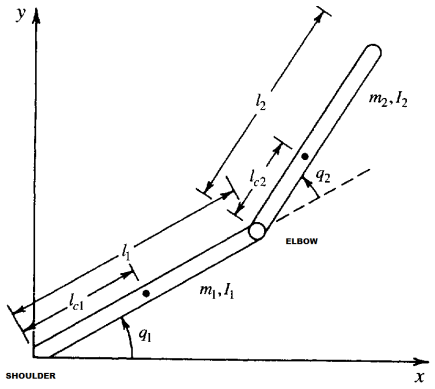


Figure A.1: Two link arm or acrobot: This figure demonstrates two link arm or acrobot. The first arm is linked to the earth via a fixed pivot(shoulder) and the second arm via elbow. For the two-linked arm model, both shoulder and elbow have actuators. For the acrobot model, only the elbow is actuated. This figure is adopted from [37]

the center and root joint, this is a fully actuated 2-link-arm. The goal is to swing the arms to upright position. The cost function is therefore defined accordingly to encourage the arms to swing upward. We used Mujoco[51] to simulate the dynamics and exactly the same parameter and cost function as in the MPC paper controlling acrobot[43]. More details are described in the following paragraphs.

The passive dynamics for acrobot and two-link arm models are the same. The structure of this dynamical system is shown in Fig.A.1. Although we used explicit equations in the early phase of this dissertation, the results shown here are generate from the Mujoco simulator, so we omit the details to avoid confusions.

The goal(the swing-up control task) is to move the system from its stable downward position to its unstable inverted position and balance it in an inverted position. The task is extremely easy for the two-linked arm model where both joints are actuated. For acrobot, several groups have built success controllers based on a carefully controller design in both simulation and experiments [3] [37]. This problem remains hard for optimal control approaches.

In optimal control, there are multiple approaches available to define control cost. Here we follows the convention in [43]. The state cost $q(x)$ is defined in the smoothed absolute

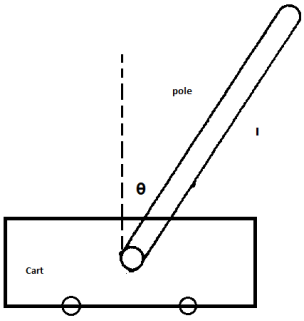


Figure A.2: Cart-pole: This figure demonstrates cart-pole model. The inverted pendulum is linked to the cart which can move horizontally. The only actuator in the system can move the cart horizontally.

value for the distance between the height of the tip of the second link and the desired height.

$$q(x) = C(\sqrt{(h(x) - h_0)^2 + \epsilon^2} - \epsilon), \tag{A.11}$$

where $h(x)$ represents the height of the tip, h_0 represents the desired height$(l_1 + l_2)$ and ϵ is a small parameter to smooth the cost function.

A.4 Cart-pole

The cart pole, a.k.a inverted pendulum in certain literature, is a planar robot composed of a pendulum and a cart, with the pendulums pivot point mounted on the cart. The cart to restricted to move only horizontally, and the actuator can only control the horizontal movement of the cart. (Fig.A.2)

The goal is to swing the pendulum to upright position. The cost function is therefore defined accordingly to encourage the pendulum to swing up. The state cost $q(x)$ is defined in the smoothed absolute value for the distance between the height of the tip of the second link and the desired height. The state cost $q(x)$ is defined in the smoothed absolute value for the distance between the height of the tip of the pendulum and the desired height.

$$q(x) = C(\sqrt{(l\cos(\theta) - l)^2 + \epsilon^2} - \epsilon), \tag{A.12}$$

where $h(x)$ represents the height of the tip of the pendulum, l represents the length of the pendulum and ϵ is a small parameter to smooth the cost function. This cost function is chosen to better compare results with that of acrobot. Mujuco is used to simulate this dynamical system.

VITA

Mingyuan Zhong was born in Tianjin, China, on November 1st, 1986. He entered Department of Physics, Peking University, China in 2002. In 2006, he entered the Department of Applied Mathematics at the University of Washington, Seattle.

He welcomes your comments at `zhongmy@uw.edu`.