# CONVEX METHODS FOR APPROXIMATE DYNAMIC PROGRAMMING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT
OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Arezou Keshavarz

December 2012

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

**Stephen Boyd, Primary Adviser**

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

**Benjamin Van Roy**

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

**Kenneth Judd**

Approved for the Stanford University Committee on Graduate Studies.

**Patricia J. Gumport, Vice Provost Graduate Education**

*This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.*

# Abstract

This thesis studies the use of convex optimization in stochastic control problems. We propose methods based on convex optimization for approximate dynamic programming.

Dynamic programming is a standard approach to many stochastic control problems, which involves decomposing the problem into a sequence of subproblems to solve for a global minimizer, called the value function. Traditional dynamic programming approaches focus on problems with finite state space and action space, and finding a solution gets prohibitively difficult as the state space or action space grows. With the exception of a few special cases, dynamic programming (DP) is difficult to carry out for general problems. In such cases, approximate dynamic programming (ADP) gives a method for finding a good, if not optimal, policy.

In this work, we rely on our ability to (numerically) solve convex optimization problems with great speed and reliability. Using custom generated solvers we can speed up computation by orders of magnitude. In this thesis, we use convex optimization in conjunction with approximate dynamic programming to find the optimal (or suboptimal, but good) policies for an array of stochastic control problems.

In the first part of the thesis, we consider applications where we have access to a good controller (which may be very complex): this could be a pilot controlling

an aircraft, or a model predictive controller with a long lookahead horizon. In such cases, we can observe the actions chosen in different states, but we do not have access to the underlying objective function used by the controller (such as the pilot). We propose methods that use convex optimization to impute the underlying objective function. This results in a controller that can mimic the behavior of the original controller, often with the complexity reduced greatly. In the second part of the thesis, we develop the mathematical framework and present algorithms for carrying out projected value iteration using quadratic approximate value functions. Although there are no theoretical guarantees, we observe that in practice we achieve very good performance in a reasonable number of steps.

We will consider problems in a variety of applications, such as consumer behavior modeling, robotics, finance, input-constrained control, and supply chain management.

# Acknowledgement

The past five years at Stanford have been one of the best times of my life. I have had the chance to meet amazing people that have touched my life on many levels. Without them this work would not have been possible, and the least I can do is to acknowledge their contribution and express my deepest gratitude for the impact that they have had on my life.

First and foremost, I would like to thank my advisor Stephen Boyd. I took EE263 in my first quarter at Stanford, and since then I have been truly mesmerized by his knowledge, teaching, and character. I have learned so much from him: not only about convex optimization and control, but also on how to think clearly and write clearly. I would also like to thank him for giving me the opportunity to teach: working as a teaching assistant for his courses and taking part in the creation of the new course EE365 has been an invaluable experience. And besides his excellent academic guidance and advice, Stephen is really the best advisor I could ask for: he has believed in me when I didn't, has motivated me when I was disappointed, and has always been a great source of support. For all this, I can't thank him enough.

I would also like to thank my associate advisor Ben Van Roy for his valuable feedback on my work and teaching great courses on dynamic programming. I thank Ken Judd for taking the time to tell me about all the exciting applications of ADP

when I was just starting out my research, and for being a member of my orals and reading committees. I thank Boris Murmann for chairing my orals committee. I thank Stanford Graduate Fellowship for partially supporting my research. I am forever indebted to Hamid Aghajan and Parham Aarabi for introducing me to the world of academic research when I was an undergraduate student.

I have many fond memories of the days in Packard 243 (the group office). I would like to thank current and former members of the convex optimization group: Yang, Brendan, Eric, Matt, Ekine, Jacob, Thomas, Neal, and Madeleine. We've had a lot of fun discussions ranging from complexity theory to career choices. I especially thank Yang for always being interested and available; he has helped me tremendously.

Words can't describe the extent of my gratitude to my friends for their support and love. I want to thank my great friends Leila, Haleh, Maryam and Sahar for being there for me in joyful and gloomy days; I am very lucky to have them as friends. I also like to thank Mohammad, Roozbeh, Hossein, Narges, Pedram, Reza, Ali, Shahriar, Parnian, Bernd, Parisa, Ali, Mohammed, AJ, Sammy, Sanaz, Mahmoud, Parastou, Hamid, Arash, and many more friends. My Stanford experience would be dramatically different if it wasn't for them. I am glad to have come to known such great people and I hope that our friendship continues in the years to come. I want to extend a special thanks to Marzieh for always being there for me, even from miles away. She has been a true sister that I never had, and I can't thank her enough.

I am grateful to Soraya, Hosein, Payam and Laila for their love and support. I thank my brother Pedram for always making me laugh and laughing with me. I would like to thank my parents, Fardaneh and Hamid, for their unconditional love and support for me. My dad has inspired me to try hard in life and not give in, and he has taught me how to find inspiration and motivation when the times are tough.

My mom has been a pillar of support for me and a great friend: she has given me confidence when I needed it the most and helped me make my own decisions and do what I love all throughout my life. Finally, I would like to thank my dear Pedram for his true love, unwavering support and encouragement. He has been the strongest wall to lean on, and the biggest source of energy for me. Thank you for holding my hand and lifting me up in difficult days and nights, and cheering for me all throughout.

# Contents

**Abstract** **iv**

**Acknowledgement** **vi**

**1 Overview** **1**

1.1 Outline . . . . . 1

1.2 Notation and Style . . . . . 2

**2 Imputing a Convex Objective Function** **4**

2.1 Introduction . . . . . 4

2.2 Prior and related work . . . . . 5

2.3 Problem statement . . . . . 8

2.4 Description of the method . . . . . 11

2.5 Applications . . . . . 14

2.5.1 Consumer behavior . . . . . 14

2.5.2 Controller fitting . . . . . 18

2.5.3 Single commodity trans-shipment network . . . . . 23

2.5.4 Portfolio management . . . . . 29

2.6 Conclusion . . . . . 34

**3 Quadratic ADP for Input-Affine Systems** **35**

3.1 Introduction . . . . . 35

3.2 Input-affine convex stochastic control . . . . . 41

3.2.1 The common model . . . . . 41

3.2.2 The problems . . . . . 44

3.3 Dynamic and approximate dynamic programming . . . . . 45

3.3.1 Linear-convex dynamic programming . . . . . 48

3.3.2 Linear-quadratic dynamic programming . . . . . 50

3.3.3 Approximate dynamic programming . . . . . 52

3.3.4 Quadratic ADP . . . . . 53

3.4 Projected value iteration . . . . . 54

3.4.1 Quadratic projected iteration . . . . . 56

3.5 Input-constrained linear quadratic control . . . . . 57

3.6 Multi-period portfolio optimization . . . . . 61

3.7 Vehicle control . . . . . 65

3.8 Conclusions . . . . . 70

**4 Case Study: Supply Chain Control** **71**

4.1 Introduction . . . . . 71

4.2 Problem description . . . . . 71

4.3 Methods . . . . . 74

4.4 Method parameters . . . . . 76

4.4.1 Imputed objective . . . . . 76

4.4.2 Projected value iteration . . . . . 77

4.5 Problem instance . . . . . 78

4.5.1 Numerical instance . . . . . 79

4.6 Results . . . . . 80
4.6.1 Imputed objective . . . . . 80
4.6.2 Projected value iteration . . . . . 80
4.6.3 Performance comparison . . . . . 80
4.6.4 Runtime comparison . . . . . 84
4.6.5 Parallelization . . . . . 85
4.7 Conclusion . . . . . 85

**A Expectation of quadratic function** **87**

**B Receding horizon control** **89**

**Bibliography** **91**

# List of Figures


2.1 Scatter plot of the desired demand level versus the realized demand level. The dashed line is the $y = x$ line. . . . . 17

2.2 Histogram of stage costs for RHC (top) and ADP (bottom). Vertical lines indicate the average cost incurred over the 1000 step simulation. 22

2.3 Example of a single commodity network. . . . . 25

2.4 Imputed cost functions (dashed) and the true cost functions (solid) for each production node. . . . . 27

2.5 Histogram of stage costs for RHC (top) and ADP (bottom). Vertical lines indicate the average cost incurred over the 1000 step simulation. 33

3.1 $J^{(k)}$ (blue solid), and $J^{(k)}_{\text{naive}}$ (red dashed). . . . . 60

3.2 $\hat{J}^{(k)}$ (blue solid), and a lower bound $J_{\text{lb}}$ (red dashed). . . . . 63

3.3 Vehicle diagram. The forces are applied at two positions, shown with large circles. At each position there are two forces applied: a lateral force ($u_1$ and $u_3$), and a longitudinal force ($u_2$ and $u_4$). . . . . 67

3.4 Desired trajectory (red dashed), two sample trajectories obtained using $\hat{V}_t, t = 1, \ldots, T$ (left), and two sample trajectories obtained using $V_t^{\text{quad}}$ (right). . . . . 69

4.1 The supply chain and the corresponding connections. Links 1 and 2 are supply links, and links 7 and 8 are demand links. . . . . 79
4.2 Scatter plot of the flows generated using MPC, versus the flows realized using the imputed ADP policy (IMP). The dashed line is the $y = x$ line. 81
4.3 Scatter plot of the flows generated using MPC, versus the flows realized using the ADP policy from projected value iteration (PVI). The dashed line is the $y = x$ line. . . . . 82
4.4 Performance of the three ADP methods: model predictive control (MPC), imputed objective (IMP), and projected value iteration (PVI). 83

# Chapter 1

# Overview

In this thesis, we develop a mathematical framework for using convex optimization to solve stochastic control problems. We will consider continuous state space and action space problems, for which the use of traditional dynamic programming methods fail in most cases. We describe two methods for approximate dynamic programming that rely on our ability to solve convex optimization problems.

We outline the thesis in §1.1, and offer a brief description of each chapter. In §1.2, we explain the notation and the style adopted throughout this thesis.

## 1.1 Outline

In chapter 2, which is based on [38], we consider an optimizing process (or parametric optimization problem), *i.e.*, an optimization problem that depends on some parameters. We assume that we can observe the choices made by an optimal or nearly optimal controller (such as a pilot, or a complex controller). We present a method for imputing or estimating the objective function, based on observations of optimal or nearly optimal choices of the variable for several values of the parameter, and prior

knowledge (or assumptions) about the objective. Applications include estimation of consumer utility function from purchasing choices, estimation of value functions in control problems, estimation of cost functions in a flow network, and estimation of the cost function of a trader managing a portfolio of assets. In each of these cases, we design simple controllers that mimic the behavior of much more complex controllers.

In chapter 3, which is based on [37], we consider the use of quadratic approximate value functions for stochastic control problems with input-affine dynamics and convex stage cost and constraints. Evaluating the approximate dynamic programming policy in such cases requires the solution of an explicit convex optimization problem, such as a quadratic program, which can be carried out efficiently. We describe a simple and general method for approximate value iteration, that also relies on our ability to solve convex optimization problems, in this case typically a semidefinite program. While we have no theoretical guarantee on the performance attained using our method, we observe that very good performance can be obtained in practice.

In chapter 4, we consider the problem of supply chain management as a case study. We will consider the flow of a single commodity through a network of nodes (such as warehouses and buffers). We develop the mathematical formulation, and compare variants of the methods described in chapters 2 and 3 to the supply chain problem. The two methods take different angles at the problem, but both of them achieve very good performance.

## 1.2 Notation and Style

We use $\mathbf{R}$ to denote the set of real numbers, $\mathbf{R}^n$ to denote the set of real $n$-dimensional vectors, and $\mathbf{R}^{n\times m}$ to denote the set of real $n\times m$ matrices. The set of $n\times n$ symmetric

matrices is denoted by $\mathbf{S}^n$, the set of $n \times n$ symmetric positive semidefinite matrices is denoted by $\mathbf{S}^n_+$, and the set of symmetric positive definite matrices is denoted by $\mathbf{S}^n_{++}$. We delimit vectors and matrices with square brackets, with the components separated by space. We use parentheses to construct column vectors from comma separated lists. For example, if $a, b, c \in \mathbf{R}$, we have

$$(a, b, c) = \left[\begin{array}{c} a \\ b \\ c \end{array}\right] = \left[\begin{array}{ccc} a & b & c \end{array}\right]^T,$$

which is an element of $\mathbf{R}^3$. The symbol $\mathbf{1}$ denotes a vector whose components are all one (with dimension determined from context). We use $\mathbf{diag}(x_1, \ldots, x_n)$, or $\mathbf{diag}(x)$, $x \in \mathbf{R}^n$, to denote a $n \times n$ diagonal matrix with diagonal entries $x_1, \ldots, x_n$. We let $\mathbf{Tr}(A)$, where $A \in \mathbf{R}^{n \times n}$, denote the sum of the diagonal entries of $A$.

The inequality symbol $\geq$ and its strict form $>$ is used to denote elementwise vector (or matrix) inequality. The curled inequality symbol $\succeq$ (and its strict form $\succ$) is used to denote matrix inequality. For example, for $A, B \in \mathbf{R}^{n \times n}$, $A \succeq B$ means that the matrix $A - B$ is positive semidefinite.

# Chapter 2

# Imputing a Convex Objective Function

## 2.1 Introduction

Parametric optimization is a tool for calculating the optimal solution of a problem for a given set of parameters. In this chapter, we consider the reverse problem: given a set of optimal (or nearly optimal) solutions, corresponding to different parameter values, how can we impute or estimate the underlying objective function? This is a very practical problem that arises in many fields such as control, robotics, economics and societal networks. There are many scenarios where we can observe a process involving agents that behave optimally (or approximately optimally), and we want to be able to say something about what is being optimized. For instance, economists assume that consumer purchases maximize a so-called utility function, minus the price paid; the utility function describes how much satisfaction the consumer receives from purchasing a product or set of products. In reality, we do not have access to a

consumer's utility function; indeed, the consumer cannot express the utility function either. But we can observe consumer purchases in response to price changes. Using the methods presented in this chapter, we can impute the underlying utility function based on observations of the consumer's behavior.

As a different domain where this method can be applied, we consider controller complexity reduction for general control problems. In many cases, a good but sophisticated controller may be available, for example model predictive control [29, 13, 70] (or a human expert), but we might desire a lower complexity controller (or one that can be automated). We can generate optimal (or closely optimal) observations using the more sophisticated controller, and then use the paradigm described in this chapter to find a controller of lower complexity.

In this chapter we consider a parametric optimization problem on a convex set, but with an unknown convex objective function. Given a series of observations, consisting of actions that correspond to different parameter values, we will show that we can impute (or approximately impute) the unknown objective function by solving another convex optimization problem. In cases where the underlying objective is not convex, we can use this method to obtain a convex approximation to the objective function.

## 2.2 Prior and related work

Many disciplines, from networks to robotics and economics, have considered the problem of imputing the underlying objective function of a process based on available observations. The work of Burton, Pulleyback and Toint [20] and Burton and Toint [21] focuses on solving the inverse shortest path problem of imputing the weight of

the graph edges, given a set of observations of shortest path traversals. Ng and Russell [52] and Abbeel and Ng [2] pose the problem of inverse reinforcement learning, where the assumption is that some information about the optimal policy is available (perhaps through observing an expert's behavior) and the goal is to learn the reward function. They consider a discrete and finite problem, which becomes intractable for large state spaces or action spaces. For infinite dimensional problems, they use an affine representation of the reward function, but since the state space is discretized, the approach still suffers from the so-called 'curse of dimensionality' [55]. There are many similar works in this domain, including [66, 51, 58, 57]. This method has been very successfully applied to helicopter robots [1]. There is a large body of literature on estimating utility functions using choice models, *e.g.*, [63, 35, 36, 62, 3, 5, 53]. Rust [62] offers a review of MDP methods in econometrics and gives examples in optimal replacement of bus engines and optimal retirement from a firm. Furthermore, in [63], we see the same techniques used to develop an empirical model the behavior of the person in charge of replacing bus engines. Berry, Levinsohn and Pakes [9] use a discrete choice model and propose techniques for fitting the cost function and aggregate demand functions as a function of product and consumer characteristics. Keane and Wolpin [35] estimate discrete choice utility functions using an approximate dynamic programming method that relies on Monte Carlo simulation to estimate expectations, and uses interpolations to estimate the value function at states that were not sampled in the fitting procedure. In a later work, Keane and Wolpin [36] use similar methods to study the career decisions of young men over a period of 11 years. More recently Nielsen and Jensen [53], consider the problem of modeling a decision maker's utility function from (possibly) inconsistent behavior and offer a solution based on influence diagrams, which can explain inconsistent behavior as random deviations from

an underlying true utility function. Ackerberg, Benkard, Berry and Pakes [3] and Bajari, Benkard and Levin [5] study the problem of estimating demand and production functions in the presence of imperfect competitions.

A closely related problem is the inverse problem of optimal control. In optimal control, given the stage cost function and state dynamics, we want to calculate the control policy and the Lyapunov or Bellman functions. In inverse optimal control, we are given the control policy, but want to learn the stage cost function and the Lyapunov function (or determine that none exist that make the given policy optimal). This topic has a long history, tracing from Kalman's original paper, *When is a Control System Optimal?* [34]. For a more recent treatment, see [16, Chapter 10.6], where it is shown that for a linear quadratic control problem with stage cost $x^TQx + u^TRu$, given a linear control policy $u = Kx$, we can recover $Q$, $R$ (if they exist) and calculate the Lyapunov function by solving a semidefinite program (SDP).

Another related area of research is dynamic programming (DP); see [11, 12] for an overview. The underlying idea in dynamic programming is to decompose the problem into a sequence of subproblems, and solve for a global minimizer, called the value function. The Hamilton-Jacobi-Bellman equation [6] is a fixed point equation for the value function, and the dynamic programming literature has offered various algorithms for calculating the value function, including policy iteration, value iteration, and linear programming [11, 12]. The difference between our work and dynamic programming is that we assume that we are given observations of the policy's behavior (*i.e.*, a set of decisions corresponding to different parameter values), whereas in most algorithms in dynamic programming, the policy is not provided and is estimated in each iteration of the algorithm. In cases where the exact value function cannot be

determined, mostly due to the so-called curse of dimensionality [55], approximate dynamic programming algorithms aim to calculate an approximate value function; see [15, 24, 12, 55]. Usually an approximate value function is obtained and is represented as a linear combination of a set of basis functions. One such algorithm, Q-learning, which was first introduced in [74, 75], has been used as a method for learning the Q-function, which is closely related to the value function, based on observing the state and action over time. Other approaches include approximate version of value iteration, policy iteration, and the linear programming formulation of DPs, where the exact value function is replaced with a linear combination of a set of basis functions [24, 12, 26].

One main difference between our approach and the works outlined above is that we do not limit the state space to discrete and finite set of values. Furthermore, our method is based on solving a convex optimization problem, which means that we avoid the curse of dimensionality [55] (at the cost of a restricted set of problems we can handle).

## 2.3 Problem statement

We consider an optimizing process, that is, a system in which a decision $x$ is made by optimizing an objective subject to constraints, where both objective and constraints can depend on a parameter $p$. (This is sometimes referred to as a parametric optimization problem.) Our goal is to learn or estimate the objective function, given a set of observations consisting of parameter values $p^{(k)}$ and associated optimal decisions $x^{(k)}$, for $k = 1, \ldots, N$. We are also given prior information, which tells us the form of the objective function and constraint functions. We refer to an objective found from

observations of optimal decisions as an *imputed objective function*.

We will focus here on the case when the optimizing process involves a convex optimization problem [19],

$$\begin{array}{ll} \text{minimize} & f(x,p) \\ \text{subject to} & g_i(x,p) \leq 0, \quad i=1,\ldots,m \\ & A(p)x = b(p), \end{array} \tag{2.1}$$

where $x \in \mathbf{R}^n$ is the variable, $f$, $g_i$, $i = 1,\ldots,m$ are differentiable and convex in $x$ for each value of $p \in \mathcal{P}$ (the set of allowable parameter values), $A : \mathcal{P} \to \mathbf{R}^{q\times n}$, and $b : \mathcal{P} \to \mathbf{R}^q$. We say that $x \in \mathbf{R}^n$ is optimal for $p \in \mathcal{P}$ if it is a solution of problem (2.1). We do not assume that for each $p$, there is only one solution of (2.1); in other words, we can have several $x$'s that are optimal for a given $p$.

**Optimality and approximate optimality** We will assume throughout that an appropriate constraint qualification holds, so for a given $p \in \mathcal{P}$ the necessary and sufficient (Kharush-Kuhm-Tucker or KKT) conditions for $x$ to be optimal are the existence of $\lambda \in \mathbf{R}^m_+$ and $\nu \in \mathbf{R}^q$ that satisfy the following conditions:

$$\begin{array}{l} g_i(x,p) \leq 0, \quad i=1,\ldots,m, \\ A(p)x = b(p), \\ \nabla f(x,p) + \sum_{i=1}^m \lambda_i \nabla g_i(x,p) + A(p)^T\nu = 0, \\ \lambda_i g_i(x,p) = 0, \quad i=1,\ldots,m, \end{array} \tag{2.2}$$

where the gradient is taken with respect to $x$. Like $x$, the dual variables $\lambda$ and $\nu$ depend on the parameter $p$, but to keep the notation light, we will not explicitly show this dependence. In (2.2), the first and second conditions are primal feasibility, the

third condition is stationarity, and the fourth condition is complementary slackness.

We use these optimality conditions to define what it means for $x \in \mathbf{R}^n$ to be *approximately optimal* for the problem (2.1): Given a parameter $p \in \mathbf{R}^q$, $x \in \mathbf{R}^n$ is said to be approximately optimal if the conditions in (2.2) hold approximately. To make this more precise, we define the residuals

$$\begin{aligned} r_{\text{ineq}} &= (g_i(x,p))_+, \quad i = 1, \ldots, m, \\ r_{\text{eq}} &= A(p)x - b(p), \\ r_{\text{stat}}(\alpha, \lambda, \nu) &= \nabla f(x,p) + \textstyle\sum_{i=1}^m \lambda_i \nabla g_i(x,p) + A(p)^T\nu, \\ r_{\text{comp}}(\lambda) &= \lambda_i g_i(x,p), \quad i = 1, \ldots, m. \end{aligned}$$

The first two residuals, $r_{\text{ineq}}$ and $r_{\text{eq}}$ correspond to primal feasibility, and are thus not a function of the dual variables.

Given a parameter $p$, if the point $x$ is optimal (and thus feasible), there exist $\lambda \in \mathbf{R}^m_+$ and $\nu \in \mathbf{R}^q$ such that all residuals are exactly zero. We say that a point $x$ is *approximately optimal* for the problem (2.1) if the primal residuals $r_{\text{ineq}}$ and $r_{\text{eq}}$ are close to zero, and there exist $\lambda \in \mathbf{R}^m_+$ and $\nu \in \mathbf{R}^q$ such that $r_{\text{stat}}$ and $r_{\text{comp}}$ are close to zero.

**The imputed objective problem** We assume that the inequality constraint functions $g_1, \ldots, g_m$ are known, as are $A$ and $b$ (which are functions from $\mathcal{P}$ into $\mathbf{R}^{q\times n}$ and $\mathbf{R}^q$, respectively). In addition, we are given a set of $N$ optimal (or approximately optimal) points along with the associated parameter values:

$$(x^{(k)}, p^{(k)}), \quad k = 1, \ldots, N.$$

We say that $f : \mathbf{R}^n \times \mathcal{P} \to \mathbf{R}$ is a consistent objective if $x^{(k)}$ is optimal for $p^{(k)}$, for all $k$.

There can be many consistent objective functions. For example, if the given observations are feasible, $f = c$, where $c$ is any constant, is always consistent. If $f$ is a consistent objective, so is $G \circ f$, where $G$ is any convex increasing function. This issue will be handled by normalization and regularization, described below.

In addition to the problem of non-uniqueness, we also encounter (in applications) the problem of non-existence of (exactly) consistent objectives. This can be due to measurement or modeling error, which render the samples $x^{(k)}$ only approximately optimal. When an exactly consistent objective cannot be found, we look for an imputed objective function that is approximately consistent with our data.

We will restrict ourselves to the case where $f$ has the finite dimensional affine parametrization

$$f = \sum_{i=0}^{K} \alpha_i f_i, \quad \alpha \in \mathcal{A},$$

where $f_i$ are pre-selected basis functions, and $\mathcal{A}$ is a convex subset of $\mathbf{R}^{K+1}$. The set $\mathcal{A}$ collects our prior information about the objective $f$. For example, if the basis functions $f_i$ are convex, then $\mathcal{A} = \mathbf{R}_+^{K+1}$ is sufficient to ensure that $f$ is a convex function. Our goal is to find $\alpha \in \mathcal{A}$ for which $f$ is consistent (or approximately consistent) with the data, which is a finite dimensional optimization problem with variable $\alpha$.

## 2.4 Description of the method

Our method for computing an imputed objective function involves finding weights $\alpha \in \mathcal{A}$ so that each decision $x^{(k)}$ is approximately optimal for the associated parameter

value $p^{(k)}$.

For each sample $(x^{(k)}, p^{(k)})$ we first introduce dual variables $\lambda^{(k)} \in \mathbf{R}^m$ and $\nu^{(k)} \in \mathbf{R}^q$. We let $r_{\text{ineq}}^{(k)}$, $r_{\text{eq}}^{(k)}$, $r_{\text{comp}}^{(k)}$ and $r_{\text{stat}}^{(k)}$ denote the residuals corresponding to each sample and its dual variables $(x^{(k)}, p^{(k)}, \lambda^{(k)}, \nu^{(k)})$. The primal residuals $r_{\text{ineq}}^{(k)}$ and $r_{\text{eq}}^{(k)}$ are fixed for each sample, and do not depend on $\alpha$. Thus, to compute an imputed objective we solve the problem

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^{N} \phi\left(r_{\text{stat}}^{(k)}, r_{\text{comp}}^{(k)}\right) \\ \text{subject to} & \lambda^{(k)} \succeq 0, \quad k = 1, \ldots, N, \quad \alpha \in \mathcal{A}, \end{array} \tag{2.3}$$

with variables $\alpha$, $\lambda^{(k)}$, $\nu^{(k)}$, $k = 1 \ldots, N$. Here, $\phi : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}_+$ is a nonnegative convex penalty function, which satisfies

$$\phi(r_{\text{stat}}, r_{\text{comp}}) = 0 \quad \Longleftrightarrow \quad r_{\text{stat}} = 0, \quad r_{\text{comp}} = 0.$$

The choice of penalty function $\phi$ will affect the distribution of the residuals as well as the imputed objective. As an example, we can take $\phi$ to be any norm on $\mathbf{R}^n \times \mathbf{R}^m$. Other choices for the penalty function include Huber, deadzone-linear, or log-barrier functions [19, §6.1].

We make a few comments about this optimization problem. First, since $r_{\text{stat}}^{(k)}$ and $r_{\text{comp}}^{(k)}$ are linear in $\alpha$, $\lambda^{(k)}$ and $\nu^{(k)}$, it is easy to see that the objective function is convex. In addition, the constraints are clearly convex, so (2.3) is a finite-dimensional convex optimization problem, which can be efficiently solved. If we solve (2.3) and find that the optimal value is equal to zero, then we must have

$$r_{\text{stat}}^{(k)}(\alpha, \lambda^{(k)}, \nu^{(k)}) = 0, \quad r_{\text{comp}}^{(k)}(\lambda^{(k)}) = 0, \quad k = 1, \ldots, N.$$

Furthermore, if the samples are also primal feasible, *i.e.*, $r_{\text{ineq}}^{(k)} = 0$, $r_{\text{eq}}^{(k)} = 0$, $k = 1, \ldots, N$, then our imputed objective is exactly consistent with our data. On the other hand if we solve (2.3) and find that the many of the dual residuals at the optimum are very large, we can conclude that our optimizing process cannot be a good model for our data.

**Trivial solutions and normalization** It is very important that the set $\mathcal{A}$ contains enough prior information about our objective $f$; otherwise trivial solutions may easily arise. For example, when $\mathcal{A} = \mathbf{R}_+^{K+1}$, a simple solution to (2.3) is $\alpha = 0$, $\lambda^{(k)} = 0$, $\nu^{(k)} = 0$, $k = 1, \ldots, N$, since $r_{\text{stat}}^{(k)}$ and $r_{\text{comp}}^{(k)}$ are homogeneous in $(\alpha, \lambda^{(k)}, \nu^{(k)})$. Another case is if we can find $\alpha \in \mathcal{A}$ for which $\sum_i \alpha_i f_i = c$, where $c$ is a constant function of $x$ and $p$. Then setting $\lambda^{(k)} = 0$ and $\nu^{(k)} = 0$ will solve (2.3).

In some cases this may be the desired result, but for most applications we will consider, we have prior knowledge that the underlying objective function has a non-trivial dependence on $x$ and $p$. For these problems a constant imputed objective is implausible, and should be excluded from the set $\mathcal{A}$. The appropriate method for doing this depends on the specific application, but here we give a normalization approach that is applicable to a large class of problems.

We assume that $\mathcal{A} = \mathbf{R}_+^{K+1}$, and $f_0, \ldots, f_K$ are non-constant, convex functions. For many applications, we know a part of the objective ahead of time. We encode this knowledge in $\mathcal{A}$ by adding the condition $\alpha_0 = 1$, so $f_0$ is the part of the objective that is fixed. This is a generic normalization method; we will see other normalization methods, more appropriate for specific problems, in the examples considered later.

## 2.5 Applications

Here we discuss potential applications of imputing an objective. One generic application is prediction, *i.e.*, guessing the decision $x$, given a parameter $p$. Another generic application is control or manipulation, *i.e.*, choosing a value of $p$ that produces a desired $x$. More specific applications are discussed below.

### 2.5.1 Consumer behavior

We consider a set of $n$ products with prices $p_i$, $i = 1, \ldots, n$. Let $x_i$ be the consumer demand for product $i$. We assume that the consumer chooses $x$ to maximize an (unknown) concave and nondecreasing utility $U$, minus the cost $p^T x$. The optimizing process involves solving the convex optimization problem

$$\begin{array}{ll} \text{minimize} & p^T x - U(x) \\ \text{subject to} & x \geq 0, \end{array} \tag{2.4}$$

with variable $x \in \mathbf{R}^n$ and parameter $p \in \mathbf{R}^n$. We are given samples $(x^{(k)}, p^{(k)})$ (for example, based on demand patterns observed in some test situation), and our goal is to impute the utility function $U : \mathbf{R}^n_+ \to \mathbf{R}$.

Once we have imputed a utility function, we can use it to find a set of prices $p_{\text{des}}$ that would achieve a target demand level $x_{\text{des}}$.

**Imputed objective problem.** For any given price $p$, the consumer spends a total of $p^T x$ and derives a utility of $U(x)$ from the purchase. For this application, we will consider a concave quadratic representation for $U$, *i.e.*, $U(x) = x^T Q x + 2r^T x$, where $Q \in \mathbf{S}^n_-$ (the set of symmetric nonpositive definite matrices), $r \in \mathbf{R}^n$. Note that we

assume that the utility function has zero offset, since adding a constant to the utility function will not affect the optimal consumption levels. Furthermore, we will assume that $U$ is nondecreasing over the range $[0, x_{\max}]$, where $x_{\max}$ is taken as the maximum demand level (which can be calculated from the training data). This means that $\nabla U \geq 0$ on that range, *i.e.*,

$$2Qx_{\max} + 2r \geq 0, \quad r \geq 0.$$

Thus, the constraint set $\mathcal{A}$ is

$$\mathcal{A} = \{(Q, r) \mid Qx_{\max} + r \geq 0,\ r \geq 0,\ Q \preceq 0\}.$$

In this problem we know the first part of the objective function is $p^T x$, so we implicitly use the normalization $\alpha_0 = 1$, given in §2.4. To compute an approximate imputed objective we solve (2.3), taking $\phi(r_{\text{stat}}, r_{\text{comp}}) = \|r_{\text{stat}}\|_2^2 + \|r_{\text{comp}}\|_2^2$.

**Numerical example.** As an example, consider a problem with $n = 5$ products and $N = 200$ observations of consumer demand. We generate the prices from a uniform distribution, *i.e.*, $p_i^{(k)} \sim U[p_{\min}, p_{\max}]$, with $p_{\min} = 8$ and $p_{\max} = 12$, and calculate the demand levels by minimizing (2.4) with

$$U_{\text{true}}(x) = \mathbf{1}^T \sqrt{Ax^{(k)} + b},$$

where $A \in \mathbf{R}_+^{n \times n}$ and $b \in \mathbf{R}_+^n$. (Note that this 'true' utility is concave and increasing, but not quadratic.)

We solve (2.3) and impute a quadratic utility function with coefficients $Q$ and $r$. We assess the performance of the imputed utility function on the training data as

well as on non-trained test data for validation. The relative training error is 5.4% and the relative test error is 6%.

**Usage.** We use the imputed objective function to set prices $p_{\text{des}}$ to achieve a certain demand level $x_{\text{des}}$. Since the underlying demand behavior is modeled by $U_{\text{true}}$, the actual realized demand level, $x_{\text{real}}$ will be different from $x_{\text{des}}$. Figure 2.1 shows a scatter plot of the desired demand level $x_{\text{des}}$ and the realized demand level $x_{\text{real}}$.

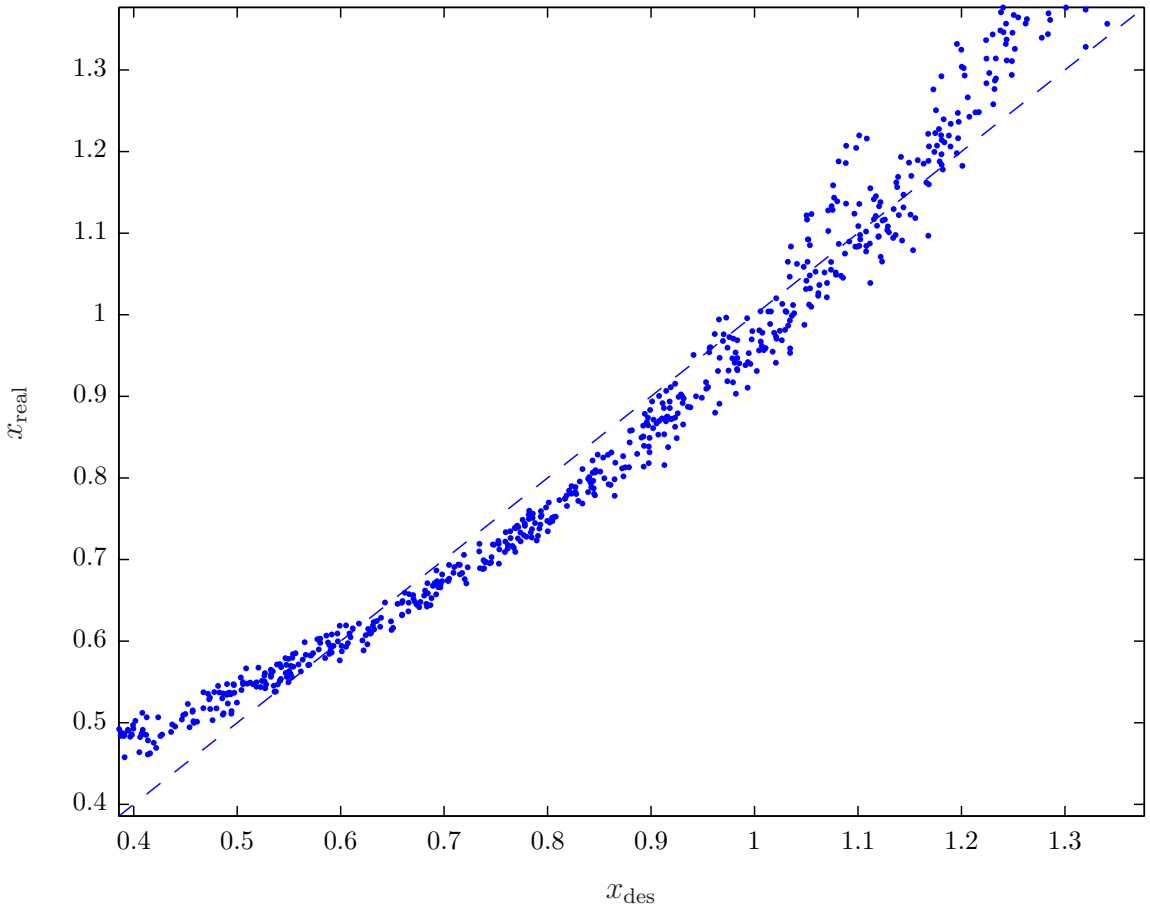


**Figure 2.1:** Scatter plot of the desired demand level versus the realized demand level. The dashed line is the $y = x$ line.

### 2.5.2 Controller fitting

In this example we fit an optimizing process to samples of a control policy, to mimic the control behavior. This allows us to simplify complex control policies, such as receding horizon control (RHC) [41], or human expert control, by approximating their action with a relatively simple optimizing process.

**Stochastic control.** We consider a linear dynamical system,

$$x_{t+1} = Ax_t + Bu_t + w_t, \quad t = 0, 1, \ldots,$$

where $x_t \in \mathbf{R}^n$ is the state, $u_t \in \mathbf{R}^m$ is the input, and $w_t \in \mathbf{R}^n$ is the noise (or exogenous input), at time $t$. We assume that $w_0, w_1, w_2, \ldots$, are zero mean IID with known distribution. The matrices $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{n\times m}$ define the system dynamics.

We look for state feedback control policies, $u_t = \phi(x_t)$, where $\phi : \mathbf{R}^n \to \mathbf{R}^m$ is the state feedback function. For a fixed state feedback function, the state and input trajectories become stochastic processes. Our objective is

$$\limsup_{T\to\infty} \frac{1}{T} \mathbf{E} \sum_{t=1}^{T-1} \ell(x_t, u_t),$$

where $\ell : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ is a convex stage cost. We also have constraints on the input

$$Fu_t \leq h \ \text{(a.s.)}, \quad t = 0, 1, \ldots,$$

where $F \in \mathbf{R}^{p\times m}$ and $h \in \mathbf{R}^p$.

The stochastic control problem is to find a control policy that minimizes the objective, among all policies that satisfy the input constraints. Here we omit technical details, such as conditions under which the expectation/limit exists. For a full technical discussion of stochastic control, see [11, 12].

**Imputing a control policy.** We are given samples $(u^{(k)}, x^{(k)})$ that come from running a suboptimal control policy, where $u^{(k)}$ is the input applied when the state is $x^{(k)}$. These samples can come from a policy such as receding horizon control (RHC, also known as model predictive control or MPC), or proportional-integral-derivative (PID) control, but more generally, they can be samples of an expert operator, such as an experienced pilot.

We model the action of the control policy by the optimizing process

$$\begin{array}{ll} \text{minimize} & \ell(x,u) + \mathbf{E}\, V(Ax + Bu + w_t) \\ \text{subject to} & Fu \leq h, \end{array}$$

with variable $u$ and parameter $x$. Here, the function $\ell : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ is the given stage cost function, and $V : \mathbf{R}^n \to \mathbf{R}$ is called an approximate value function. We refer to this control policy as the approximate dynamic programming (ADP) policy [55, 12].

In this example, we look for a convex quadratic $V$ with the form $V(z) = z^T P z$. Taking the above expectation analytically, our optimizing process simplifies to

$$\begin{array}{ll} \text{minimize} & \ell(x,u) + z^T P z \\ \text{subject to} & Fu \leq h, \quad z = Ax + Bu, \end{array} \tag{2.5}$$

with variables $z$ and $u$, and parameter $x$. The imputed objective problem is to find a

symmetric $P \succeq 0$ that is approximately consistent with the samples $((u^{(k)}, z^{(k)}), x^{(k)})$, $k = 1, \ldots, N$, where we let $z^{(k)} = Ax^{(k)} + Bu^{(k)}$. Once we have imputed a $P$, we can use our optimizing process as a suboptimal controller. Thus using this method we can approximate potentially complex control behaviors with a simple optimizing process model.

Note that in this problem we implicitly use the normalization given in §2.4, since we know part of the objective (the stage cost function), and require $P \succeq 0$. To compute an approximate imputed objective we solve (2.3), taking $\phi(r_{\text{stat}}, r_{\text{comp}}) = \|(r_{\text{stat}}, r_{\text{comp}})\|_2^2$.

**Numerical example.** We consider a control problem with $n = 10$ states and $m = 4$ inputs, where the dynamics matrices $A$ and $B$ are randomly generated, with entries drawn from $\mathcal{N}(0, 1)$; $A$ is then scaled so $|\lambda_{\max}(A)| < 1$. The disturbance $w_t$ has distribution $\mathcal{N}(0, I)$. Our stage cost is quadratic, with the form $\ell(x, u) = \|x\|_2^2 + \|u\|_2^2$, and we have box constraints $\|u\|_\infty \leq 0.1$.

To generate the data, we simulate the system under RHC (which we describe in appendix B), for 1000 time steps. Using the states and inputs obtained in the first 100 steps we impute a quadratic approximate value function by solving (2.3). We then repeat the simulation (using the same disturbance trajectory), with the system running under our ADP policy. The results are presented in Figure 2.2, which shows histograms of stage costs for RHC (top), and ADP (bottom). The vertical lines show the average cost incurred over the 1000 step simulation. For RHC the average cost is $J_{\text{rhc}} = 173.0$, for ADP the average cost is $J_{\text{adp}} = 173.4$, which is almost identical. Thus, we have obtained similar control performance to RHC using an ADP policy, which requires solving a smaller optimization problem with significantly fewer variables and constraints at each time step. Indeed, when the constraints are

polyhedral and the approximate value function is quadratic, the ADP policy can be computed extremely fast; see, *e.g.*, [73, 8].

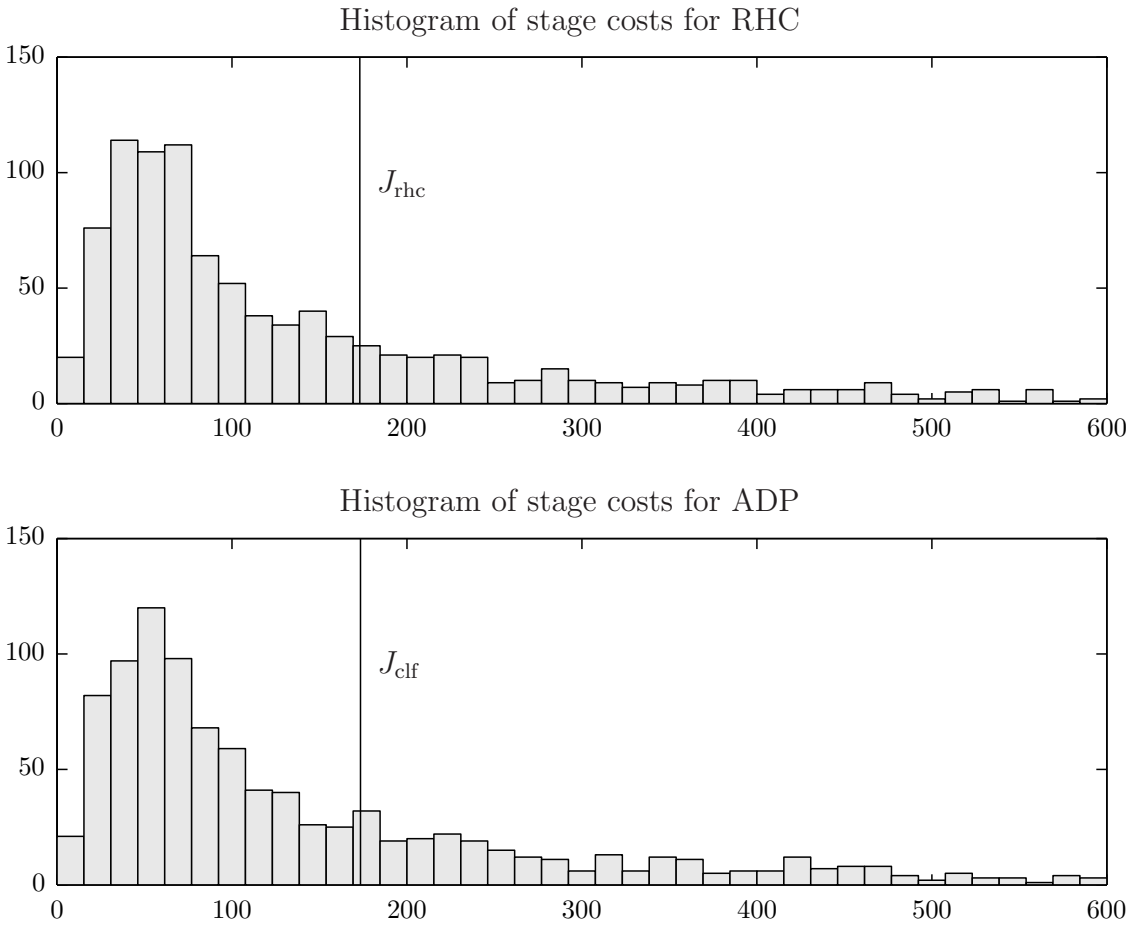


**Figure 2.2:** Histogram of stage costs for RHC (top) and ADP (bottom). Vertical lines indicate the average cost incurred over the 1000 step simulation.

### 2.5.3 Single commodity trans-shipment network

In this application we consider a single commodity trans-shipment network (for example, electricity), with a total of $n$ nodes, where nodes $i = 1, \ldots, p$ are producers and the remaining nodes $i = p+1, \ldots, n$ are consumers. There are $m$ edge between the nodes, and we are given the edge-node incidence matrix $R \in \mathbf{R}^{n \times m}$:

$$R(i,j) = \begin{cases} 1 & \text{if node } i \text{ is the starting node for edge } j \\ -1 & \text{if node } i \text{ is the ending node for edge } j \\ 0 & \text{otherwise.} \end{cases}$$

We assume that the edges have a maximum capacity of $f^{\max}$; furthermore, we also assume that the network obeys conservation of flows, *i.e.*, for a given flow on the network $f \in \mathbf{R}^m$, production level $y \in \mathbf{R}^p$ and demand level $d \in \mathbf{R}^{n-p}$, we have $Rf = (y, d)$.

A negative value for $s_i$ means that there was an outflow at node $i$ (which would be common in the demand nodes), and a positive $s_i$ means that there is an injection at node $i$ (which would be common in the producer nodes). Each producer $i = 1, \ldots, p$ can produce a maximum of $y_i^{\max}$ units of commodity, and incurs a cost of $C^{(i)}(y_i)$ for producing $y_i$ units. We also assume that there is a linear transportation cost $w_i f_i$ associated with each edge $i = 1, \ldots, m$. Finally, we can write the underlying optimizing process as an optimization problem with parameter $d$ and variable $y$:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{p} C^{(i)}(y_i) + \sum_{i=1}^{m} w_i f_i \\ \text{subject to} & 0 \leq y \leq y^{\max}, \\ & 0 \leq f \leq f^{\max}, \\ & Rf = \begin{bmatrix} y \\ d \end{bmatrix}. \end{array} \tag{2.6}$$

**Imputed objective problem.** We are given a set of $N$ different demand scenarios $d^{(k)}$, the corresponding flows $f^{(k)}$ and the producer levels $y^{(k)}$, based on historical operation of the network. The variable in this problem is $x = (y, f)$ and the parameter is $d$. For this example, we will focus on a polynomial representation for the producer cost functions, so the objective can be written as

$$\sum_{i=1}^{p} C^{(i)}(y_i) + \sum_{i=1}^{m} w_i f_i = \sum_{i=1}^{p} \sum_{j=1}^{K} A_{i,j} y_i^j + \sum_{i=1}^{m} w_i f_i,$$

where $y_i^j$, $i = 1, \ldots, p$, $j = 1, \ldots, K$ are the basis functions for the production level $y$ and $f_i$, $i = 1, \ldots, m$ are basis functions for the flows $f$, and

$$\mathcal{A} = \{(A, w) \mid A \geq 0,\ w \geq 0,\ \mathbf{1}^T(w + A\mathbf{1}) = 1\}.$$

Note that unlike the examples in §2.5.1 and §2.5.2, in this example we do not have access to a part of the objective function $f_0$. Since we are not interested in a trivial objective function, we normalize the weights by requiring $\mathbf{1}^T(w + A\mathbf{1}) = 1$. Since the objective is homogenous in the variables, this normalization will result in the same solution up to a multiplicative factor.

To impute the objective function, we solve (2.3), taking $\phi(r_{\text{stat}}, r_{\text{comp}}) = \|r_{\text{stat}}\|_2^2 +$

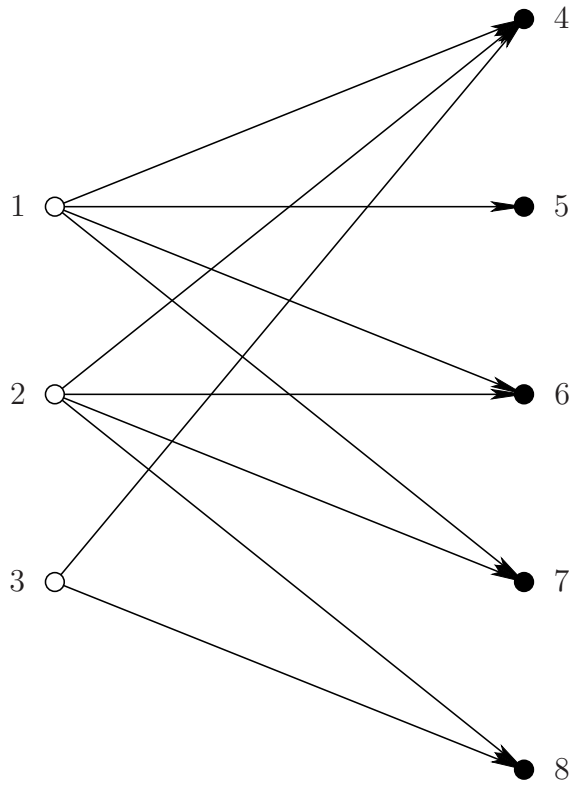


**Figure 2.3:** Example of a single commodity network.

$\|r_{\text{comp}}\|_2^2$.

**Numerical example.** As an example, consider a problem instance with $n = 8$ nodes, $p = 3$ producers and $N = 240$ observations of network flows and production levels and $K = 5$ polynomials for basis functions. The consumers and producers are connected according to the graph in figure 2.3. For each consumer, we generate negative demand levels randomly from a uniform distribution $U[-1, 0]$. The production

levels and edge flows are generated using the cost functions

$$C^{(1)}_{\text{true}}(y_1) = e^y - 1,$$
$$C^{(2)}_{\text{true}}(y_2) = y^2,$$
$$C^{(3)}_{\text{true}}(y_3) = 2y^3.$$

The edge weights are chosen randomly from a $U[2.5, 7.5]$ distribution, and the maximum edge capacity is set to $f^{\max} = 1.2$ for all edges. We use different production limits for each producer, $y^{\max} = (2, 1.1, 1.7)$.

As described above, we obtain a polynomial imputed objective function. The true cost functions (used to generate the data) and the imputed cost functions are plotted in figure 2.4. (The imputed cost functions are scaled to be on the same scale as the true cost functions for illustration purposes). To assess the performance of the imputed utility function, we used it to generate production level and edge flows on the training data as well as non-trained test data for cross validation. The average relative error in production level is around 7.8% for the training data and 9.6% for the test data.

**Usage.** Once we have imputed an objective function, we can use it to predict the production level and edge flows for a given demand level. Furthermore, the dual variable $\nu$ associated with the equality constraint $Rf = (y, d)$ provides a set of shadow prices. The values of $\nu_1, \ldots, \nu_p$ correspond to production prices and the values of $\nu_{p+1}, \ldots, \nu_n$ provide a set of consumer prices. Thus, for a given demand level, we can solve the convex optimization problem (2.6) and dynamically adjust the consumer prices by using $\nu_{p+1}, \ldots, \nu_n$. As a numerical example, consider the network in figure 2.3. For a demand level of $d = (1, 0.1, 0.2, 0.3, 3)$, we get $\nu_8 = 0.35$, while $\nu_i$ is on the

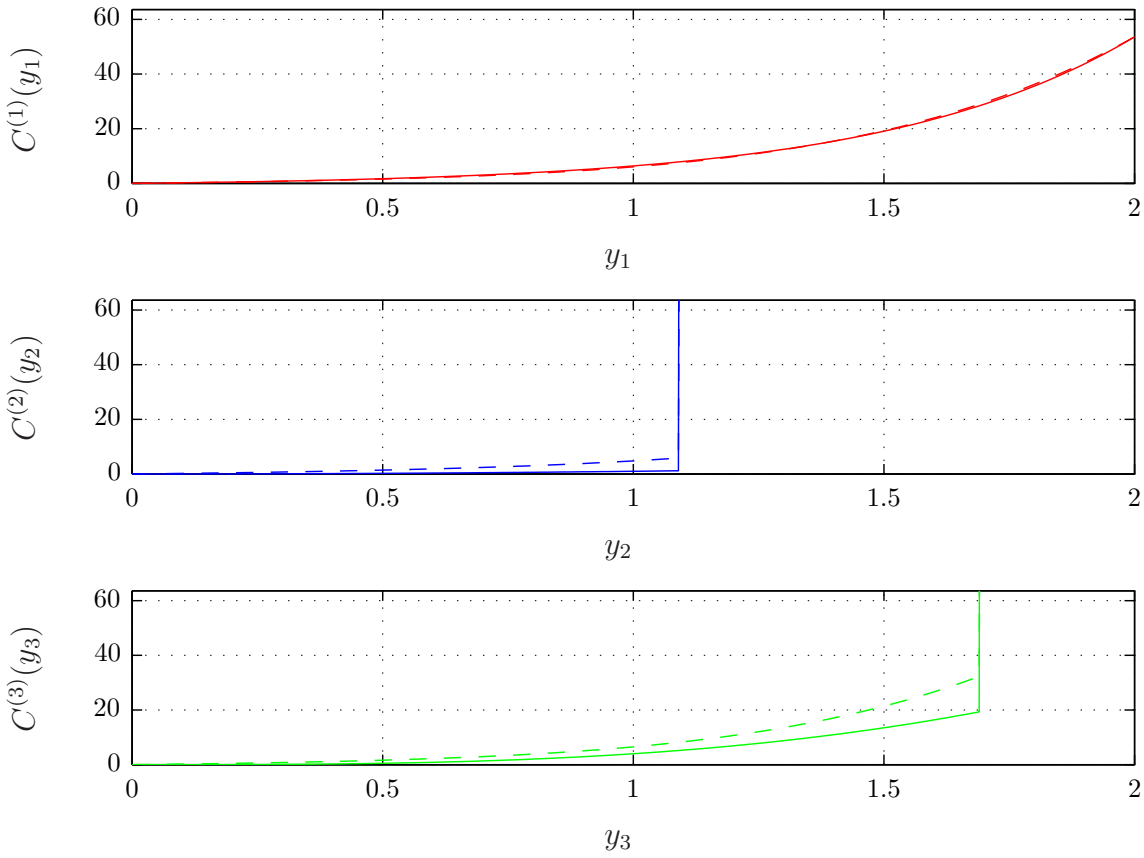


**Figure 2.4:** Imputed cost functions (dashed) and the true cost functions (solid) for each production node.

order of 0.01 or smaller for all other demand nodes. This means that the consumer at node 8 should be charged a much higher price than other customers, since it is only connected to the producers that have a small maximum production capacity, and it is putting a very large demand on the network.

### 2.5.4 Portfolio management

In this example we observe the trades performed by an expert, and we impute the parameters of the trader's cost function in order to mimic its behavior. We consider an average cost problem of optimizing a portfolio of $n$ assets. The position at time $t$ is denoted by $x_t \in \mathbf{R}^n$, where $(x_t)_i$ denotes the dollar value of asset $i$ at the beginning of time period $t$. At each time $t$, we can buy and sell assets. We denote the trades by $u_t \in \mathbf{R}^n$, where $(u_t)_i$ denotes the trade for asset $i$. A positive $(u_t)_i$ means that we buy asset $i$, and a negative $(u_t)_i$ means that we sell asset $i$ at time $t$. The position evolves according to the dynamics with

$$x_{t+1} = \mathbf{diag}(r_t)(x_t + u_t),$$

where $r_t$ is the vector of asset returns in period $t$. The return vectors are IID with mean $\mathbf{E}\, r_t = \bar{r}$ and covariance $\mathbf{E}(r_t - \bar{r})(r_t - \bar{r})^T = \Sigma$.

The stage cost consists of the following terms: the total gross cash put in, which is $\mathbf{1}^T u_t$; a risk penalty (a multiple of the variance of the post-trade portfolio); a quadratic transaction cost; and the constraint that the portfolio is long-only (which is $x_t + u_t \geq 0$). It is given by

$$\ell(x,u) = \begin{cases} \mathbf{1}^T u + c(x+u)^T \Sigma (x+u) + u^T \mathbf{diag}(s) u & x + u \geq 0 \\ \infty & \text{otherwise,} \end{cases}$$

where $c \geq 0$ is the risk aversion parameter, and $s_i \geq 0$ models the price-impact cost for asset $i$. We assume that the initial portfolio $x_0$ is given.

**Imputing the trader's policy.** We are given samples $(u^{(k)}, x^{(k)})$ that come from running a suboptimal trading policy, where $u^{(k)}$ is the trade performed when the position is $x^{(k)}$. We do not know the trader's risk aversion parameter $c$ or its cost impact parameter $s$. We will, however, assume that we know the mean and covariance matrix of the returns, $\overline{r}$ and $\Sigma$. We model the action of the trader by the following optimizing process

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T u + c(x+u)^T\Sigma(x+u) + u^T\,\mathbf{diag}(s)u + \mathbf{E}\,V(\mathbf{diag}(r)(x+u)) \\ \text{subject to} & x+u \geq 0, \end{array}$$

with variable $u$ and parameter $x$. Here the function $V : \mathbf{R}^n \to \mathbf{R}$ is an approximate value function for the trader. In this example, we consider a convex quadratic $V$ with the form $V(z) = (1/2)z^T Pz + p^T z$. Thus, our optimizing process simplifies to

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T u + c(x+u)^T\Sigma(x+u) + u^T\,\mathbf{diag}(s)u + (1/2)(x+u)^T A(x+u) + a^T(x+u) \\ \text{subject to} & x+u \geq 0, \end{array}$$

where

$$A = P \circ (\Sigma + \overline{r}\overline{r}^T), \quad a = p \circ \overline{r}.$$

Here $\circ$ denotes the Hadamard (elementwise) product. Note that $A \succeq 0$, since the Hadamard product of symmetric positive semidefinite matrices is positive semidefinite.

The imputed objective problem is to find $P \succeq 0$, $p$, $c$ and $s$ that is approximately consistent with the samples $(u^{(k)}, x^{(k)})$, $k = 1, \ldots, N$. To do so, we solve (2.3), taking $\phi(r_{\text{stat}}, r_{\text{comp}}) = \|r_{\text{stat}}\|_2^2 + \|r_{\text{comp}}\|_2^2$. Once we have imputed these coefficients, we can use them as a suboptimal trading policy. Thus, we can use this method to mimic the

behavior of a much more complicated trader with a simple optimizing process.

**Numerical example.** We consider $n = 8$ assets and an initial position of $x_0 = 0$. The returns $r_t$ are IID and have a log-normal distribution, *i.e.*, $\log r_t \sim \mathcal{N}(\mu, \tilde{\Sigma})$. The means $\mu_i$ are chosen from a $\mathcal{N}(0, 0.01^2)$ distribution; the diagonal entries of $\tilde{\Sigma}$ are chosen from a uniform $[0, 0.01]$ distribution, and the off-diagonal entries are generated using $\tilde{\Sigma}_{ij} = C_{ij}(\Sigma_{ii}\Sigma_j j)^{1/2}$, where $C$ is a correlation matrix generated randomly. Thus we have

$$\overline{r} = \exp(\mu + (1/2)\,\mathbf{diag}(\tilde{\Sigma})), \quad \Sigma_{ij} = \overline{r}_i\overline{r}_j(\exp\tilde{\Sigma}_{ij} - 1).$$

To simulate the expert trader's behavior, we will use a receding horizon controller for 1000 time steps, assuming that we know the trader's true cost function parameters $c^{\text{true}}$ and $s^{\text{true}}$. For a detailed explanation of receding horizon control (RHC) see appendix B. We will use $c^{\text{true}} = 0.5$ and generate each entry of $s^{\text{true}}$ from a uniform distribution. Then, we use the positions visited and trades performed in the first $N = 100$ steps to impute the objective function.

As described above, we impute a quadratic value function $V(z) = (1/2)z^TPz + p^Tz$, as well as the parameters $c$ and $s$. To assess the performance of the ADP controller resulting from the imputed objective function, we simulate it for 1000 time steps and compare the histogram of the costs with that of RHC.

Figure 2.5 shows the two histograms, with the top one corresponding to RHC and the bottom one corresponding to the imputed objective function. For RHC, the average cost is $J_{\text{rhc}} = -0.0121$, and for ADP the average cost is $J_{\text{adp}} = -0.0119$, which means that the performance of ADP and RHC is the same within numerical accuracy. We also note that the histograms are very close to each other. This means that we have been able to mimic the performance of the expert trader with a much more

simpler trading policy which requires solving a much smaller optimization problem with fewer variables and constraints at each time.

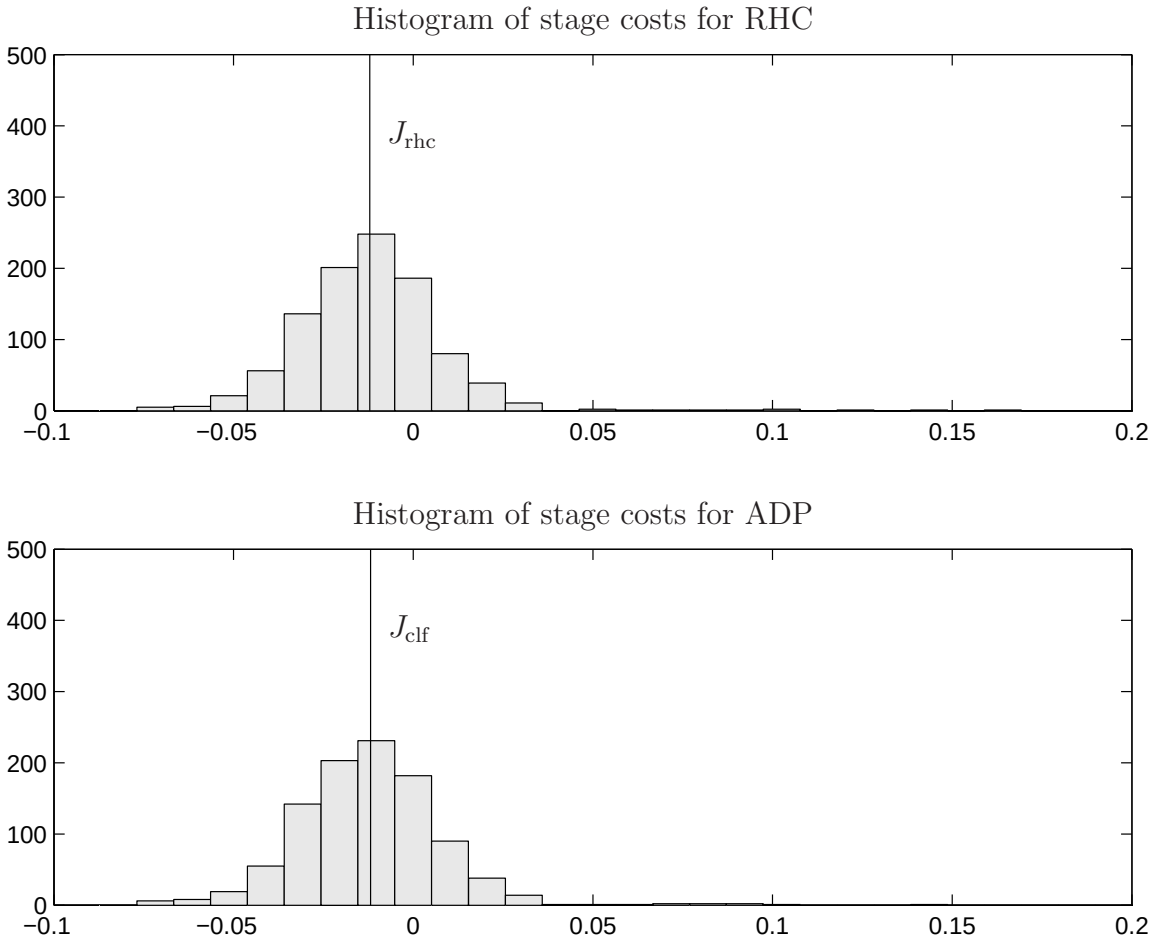


**Figure 2.5:** Histogram of stage costs for RHC (top) and ADP (bottom). Vertical lines indicate the average cost incurred over the 1000 step simulation.

## 2.6 Conclusion

The problem of imputing an objective function is a practical problem that arises frequently, and is closely related to problems in machine learning, dynamic programming, robotics, and economics. In this chapter we presented a framework for imputing a convex objective function, based on observations of the underlying optimizing process. We develop this framework by noting that if the data is optimal with respect to the imputed objective, the primal and dual variables must satisfy the KKT conditions; since the dual variables and the objective parameters enter these conditions linearly, we have a convex optimization problem. In the presence of noise or modeling errors, we minimize the residuals of the KKT conditions under a suitable regularization function.

We show the application of this method to examples in consumer behavior, control, single commodity networks, and portfolio management. We observe that the method works very well in these examples. Furthermore, we note that we can use this framework to reduce the complexity of a controller by observing solutions derived from a more complicated controller.

# Chapter 3

# Quadratic ADP for Input-Affine Systems

## 3.1 Introduction

We consider stochastic control problems with input-affine dynamics and convex stage costs, which arise in many applications in nonlinear control, supply chain, and finance. Such problems can be solved in principle using dynamic programming (DP), which expresses the optimal policy in terms of an optimization problem involving the value function (or a sequence of value functions in the time-varying case), which is a function on the state space $\mathbf{R}^n$. The value function (or functions) can be found, in principle, by an iteration involving the Bellman operator, which maps functions on the state space into functions on the state space, and involves an expectation and a minimization step; see [6] for early work on dynamic programming, and [11, 12, 56] for a more detailed overview. For the special case when the dynamics is linear and the stage cost is convex quadratic, DP gives a complete and explicit solution of the problem, since in

this case the value function is also quadratic, and the expectation and minimization steps both preserve quadratic functions, and can be carried out explicitly.

For other input-affine convex stochastic control problems, DP is difficult to carry out. The basic problem is that there is no general way to (exactly) represent a function on $\mathbf{R}^n$, much less carry out the expectation and minimization steps that arise in the Bellman operator. For small state space dimension, say $n \leq 4$, the important region of state space can be gridded (or otherwise represented by a finite number of points), and functions on it can be represented by a finite-dimensional set of functions, such as piecewise-affine. Brute force computation can then be used to find the value function (or more accurately, a good approximation of it), and also the optimal policy. But this approach will not scale to larger state space dimensions, since in general the number of basis functions needed to represent a function to a given accuracy grows exponentially in the state space dimension $n$ (this is the curse of dimensionality).

For problems with larger dimensions, approximate dynamic programming (ADP) gives a method for finding a good, if not optimal, policy. In ADP the control policy uses a surrogate or approximate value function in place of the true value function. The distinction with the brute force numerical method described above is that in ADP, we abandon the goal of approximating the true value function well, and instead only hope to capture some of its key attributes. The approximate value function is also chosen so that the optimization problem that must be solved to evaluate the policy is tractable. It has been observed that good performance can be obtained with ADP, even when the approximate value function is not a particularly good approximation of the true value function. (Of course, this problem depends on how the approximate value function is chosen.)

There are many methods for finding an appropriate approximate value function;

for an overview on approximate dynamic programming see [12]. The approximate value function can be chosen as the true value function of a simplified version of the stochastic control problem, for which we can easily obtain the value function. For example, we can form a linear-quadratic approximation of the problem, and use the (quadratic) value function obtained for the approximation as our approximate value function. Another example, used in model predictive control (MPC) or certainty-equivalent roll out (see, *e.g.*, [13, 29]) is to replace the stochastic terms in the original problem with constant values, say, their expectations. This results in an ordinary optimization problem, which (if it is convex) we can solve to obtain the value function. Recently, Wang et al have developed a method that uses convex optimization to compute a quadratic lower bound on the true value function, which provides a good candidate for an approximate value function, and also gives a lower bound on the optimal performance [71, 72]. This was extended in [54] to use as approximate value function the pointwise supremum of a family of quadratic lower bounds on the value function. These methods, however, cannot be used for general input-affine convex problems.

ADP is closely related to inverse optimization, which has been studied in various fields such as control, revenue management, robotics and economics [38, 16, 42, 76, 28, 48, 52, 1, 3, 5, 53]. In inverse optimization, the goal is to find (or estimate or impute) the objective function in an underlying optimizing process, given observations of optimal (or nearly optimal) actions. In a stochastic control problem, the objective is related to the value function, so the goal is to approximate the value function given samples of optimal or nearly optimal actions. Watkins et al introduced an algorithm called Q-learning [74, 75], which observes the action-state pair at each step and updates a quality function based on the obtained reward/cost.

MPC, also known as receding-horizon control (RHC) or rolling-horizon planning [39, 77, 30], is also closely related to ADP. In MPC, at each step we plan for the next $T$ time steps, using some estimates or predictions of future unknown values, and then execute only the action for the current time step. At the next step, we solve the same problem again, now using the value of the current state and updated values of the predictions based on any new information available. MPC can be interpreted as ADP, where the approximate value function is the optimal value of an optimization problem.

In this chapter we take a simple approach, by restricting the approximate value functions to be quadratic. This implies that the expectation step can be carried out exactly, and that evaluating the policy reduces to a tractable convex optimization problem, often a simple quadratic program (QP), in many practical cases.

To choose a quadratic approximate value function, we use the same Bellman operator iteration that would result in the true value function (when some technical conditions hold), but we replace the Bellman operator with an approximation that maps quadratic functions to quadratic functions. The idea of value iteration, followed by a projection step back to the subset of candidate approximate value functions, is an old one, explored by many authors [12, Chapter 6], [33, Chapter 12], [55]. In the reinforcement learning community, two closely related methods are fitted value iteration and fitted Q-iteration [27, 59, 49, 50, 4]. Most of these works consider problems with continuous state space and finite action space, and some of them come with theoretical guarantees on convergence and performance bounds. For instance [27] propose a family of tree-based methods for fitting approximate Q-functions and provide convergence guarantees and performance bounds for some of the proposed methods. Riedmiller considers the same framework, but employs a multi-layered

perceptron to fit (model-free) Q-functions and provides empirical performance results [59]. Munos and Szepesvari develop finite-time bounds for fitted value iteration for discounted problems with bounded rewards [50]. Antos et al extend this work to continuous action space and develop performance bounds for a variant of the fitted Q-iteration using stationary stochastic policies, where greedy action selection is replaced with searching on a restricted set of candidate policies [4].

There are also approximate versions of policy iteration and the linear programming formulation of DPs [24, 25, 26], where the value function is replaced by a linear combination of a set of basis functions. Lagoudakis et al consider the discounted finite state case and propose a projected policy iteration method [40] which minimizes the $\ell_2$ Bellman residual to obtain an approximate value function and policy at each step of the algorithm. Tsitsiklis and Van Roy established strong theoretical bounds on the suboptimality of the ADP policy, for the discounted cost finite state case [67, 61].

Our contribution is to work out the details of a projected ADP method for the case of input-affine dynamics and convex stage costs, in the relatively new context of availability of extremely fast and reliable solvers for convex optimization problems.

We rely on our ability to (numerically) solve convex optimization problems with great speed and reliability. Recent advances have shown that using custom-generated solvers will speed up computation by orders of magnitude [8, 70, 45, 44, 46, 43]. With solvers that are orders of magnitude faster, we can carry out approximate value iteration for many practical problems and obtain approximate value functions that are simple, yet achieve very good performance.

The method we propose comes with no theoretical guarantees. There is no theoretical guarantee the modified Bellman iterations will converge, and when they do (or when we simply terminate the iterations early), we do not have any bound on how

suboptimal the resulting control policy is. On the other hand, we have found that the methods described in this chapter work very well in practice. In cases in which the performance bounding methods of Wang et al can be applied, we can confirm that our quadratic ADP policies are very good, and in many cases, nearly optimal, at least for specific cases.

Our method has many of the same characteristics as proportional-integral-derivative (PID) control, widely used in industrial control. There are no theoretical guarantees that PID control will work well; indeed it is easy to create a plant for which no PID control results in a stable closed-loop system (which means the objective value is infinite). On the other hand, PID control, with some tuning of the parameters, can usually be made to work very well, or at least well enough, in many (if not most) practical applications.

**Style of this chapter.** The style of this chapter is informal, but algorithmic and computationally oriented. We are very informal and cavalier in our mathematics, occasionally referring the reader to other work that covers the specific topic more formally and precisely. We do this so as to keep the ideas simple and clear, without becoming bogged down in technical details; and in any case, we do not make any mathematical claims about the methods described. The methods presented are offered as methods that can work very well in practice, and not methods for which we make any more specific or precise claim. We also do not give the most general formulation possible, instead sticking with what we hope gives a good trade-off of clarity, simplicity, and practical utility.

**Assumptions.** We assume that all expectations exist, all sums exist, and all limits exist. In fact these assumptions might not be true in all cases; see *e.g.*, [14, Chapter

5.5] for much more detail and technical conditions. The statements in this chapter only hold when the disturbance is drawn from a countable set. We suspect that many of the statements generalize to other cases, but we cannot make any claims.

**Outline.** In §3.2 we describe three variations on the stochastic control problem. In §3.3 we review the dynamic programming solution of the problems, including the special case when the dynamics is linear and the stage cost functions are convex quadratic, as well as quadratic approximate dynamic programming, which is the method we propose. We describe a projected value iteration method for obtaining an approximate value function in §3.4. In the remaining three sections we describe numerical examples: a traditional control problem with bounded inputs (in §3.5), a multi-period portfolio optimization problem (§3.6), and a vehicle control problem (§3.7).

## 3.2 Input-affine convex stochastic control

In this section we set up three variations on the input-affine convex stochastic control problem: The finite-horizon case, the discounted infinite horizon case, and the average cost case. We start with the assumptions and definitions that are common to all cases.

### 3.2.1 The common model

**Input-affine random dynamics.** We consider a dynamical system with state $x_t \in \mathbf{R}^n$, input or action $u_t \in \mathbf{R}^m$, and input-affine random dynamics

$$x_{t+1} = f_t(x_t) + g_t(x_t)u_t,$$

where $f_t : \mathbf{R}^n \to \mathbf{R}^{n\times n}$ and $g_t : \mathbf{R}^n \to \mathbf{R}^{n\times m}$. We assume that $f_t$ and $g_t$ are (possibly) random, *i.e.*, depend on some random variables, with $(f_t, g_t)$ and $(f_\tau, g_\tau)$ independent for $t \neq \tau$. The initial state $x_0$ is also a random variable, independent of $(f_t, g_t)$. We say the dynamics is time-invariant if the random variables $(f_t, g_t)$ are identically distributed (and therefore IID).

**Linear dynamics.** We say the dynamics is linear (or more accurately, affine), if $f_t$ is an affine function of $x_t$, and $g_t$ does not depend on $x_t$, in which case we can write the dynamics in the more familiar form

$$x_{t+1} = A_t x_t + B_t u_t + c_t.$$

Here $A_t \in \mathbf{R}^{n\times n}$, $B_t \in \mathbf{R}^{n\times m}$, and $c_t \in \mathbf{R}^n$ are random.

**Closed-loop dynamics.** We consider state feedback policies, *i.e.*,

$$u_t = \phi_t(x_t), \quad t = 0, 1, \ldots,$$

where $\phi_t : \mathbf{R}^n \to \mathbf{R}^m$ is the policy in time period $t$. The closed-loop dynamics are then

$$x_{t+1} = f_t(x_t) + g_t(x_t)\phi_t(x_t),$$

which recursively defines a stochastic process for $x_t$ (and $u_t = \phi_t(x_t)$). We say the policy is time-invariant if it does not depend on $t$, in which case we denote it by $\phi$.

**Stage cost.** The stage cost is given by $\ell_t(z, v)$, where $\ell_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$ is an extended valued closed convex function. We use infinite stage cost to denote

unallowed state-input pairs, *i.e.*, (joint) constraints on $z$ and $v$. We will assume that for each state $z$ there exists an input $v$ with finite stage cost. The stage cost is time-invariant if $\ell_t$ does not depend on $t$, in which case we write it as $\ell$.

**Convex quadratic stage cost.** An important special case is when the stage cost is a (convex) quadratic function and has the form

$$\ell_t(z,v) = (1/2)\begin{bmatrix} z \\ v \\ 1 \end{bmatrix}^T \begin{bmatrix} Q_t & S_t & q_t \\ S_t^T & R_t & r_t \\ q_t^T & r_t^T & s_t \end{bmatrix} \begin{bmatrix} z \\ v \\ 1 \end{bmatrix},$$

where

$$\begin{bmatrix} Q_t & S_t \\ S_t^T & R_t \end{bmatrix} \succeq 0,$$

*i.e.*, is positive semidefinite.

**QP-representable stage cost.** A stage cost is QP-representable if $\ell_t$ is convex quadratic plus a convex piecewise linear function, subject to polyhedral constraint (which can be represented in the cost function using an indicator function). Minimizing such a function can be done by solving a QP.

**Stochastic control problem.** In the problems we consider, the overall objective function, which we denote by $J$, is a sum or average of the expected stage costs. The associated stochastic control problem is to choose the policies $\phi_t$ (or policy $\phi$ when the policy is time-invariant) so as to minimize $J$. We let $J^\star$ denote the optimal (minimal) value of the objective, and we let $\phi_t^\star$ denote an optimal policy at time period $t$.

When the dynamics is linear, we call the stochastic control problem a linear-convex

stochastic control problem. When the dynamics is linear and the stage cost is convex quadratic, we call the stochastic control problem a linear-quadratic stochastic control problem.

### 3.2.2 The problems

**Finite-horizon.** In the *finite-horizon problem*, the objective is the expected value of the sum of the stage costs up to a horizon $T$:

$$J = \sum_{t=0}^{T} \mathbf{E}\, \ell_t(x_t, u_t).$$

**Discounted infinite-horizon.** In the *discounted infinite-horizon problem*, we assume that the dynamics, stage cost, and policy are time-invariant. The objective in this case is

$$J = \sum_{t=0}^{\infty} \mathbf{E}\, \gamma^t \ell(x_t, u_t),$$

where $\gamma \in (0, 1)$ is a discount factor.

**Average cost.** In the *average-cost problem*, we assume that the dynamics, stage cost, and policy are time-invariant, and take as objective the average stage cost,

$$J = \lim_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{T} \mathbf{E}\, \ell(x_t, u_t).$$

We will simply assume that the expectation exists in all three objectives, the sum exists in the discounted average cost, and the limit exists in the average cost.

## 3.3 Dynamic and approximate dynamic programming

The stochastic control problems described above can be solved in principle using dynamic programming, which we sketch here for future reference. See [12, 56, 14] for (much) more detail. Dynamic programming makes use of a function (time-varying in the finite-horizon case) $V : \mathbf{R}^n \to \mathbf{R}$ that characterizes the cost of starting from that state, when an optimal policy is used. The definition of $V$, its characterization, and methods for computing it differ (slightly) for the three stochastic control problems we consider. In all three scenarios discussed below, we will assume that an optimal stationary policy exists; the required technical conditions can be found in detail in [14].

**Finite-horizon.** In the finite-horizon case, we have a value function $V_t$ for each $t = 0, \ldots, T$. The value function $V_t(z)$ is the minimum cost-to-go starting from state $x_t = z$ at time $t$, using an optimal policy for time periods $t, \ldots, T$:

$$V_t(z) = \sum_{\tau=t}^{T} \mathbf{E}\left(\ell_\tau(x_\tau, \phi_\tau^\star(x_\tau)) \mid x_t = z\right), \quad t = 0, \ldots, T.$$

The optimal cost is given by $J^\star = \mathbf{E}\, V_0(x_0)$, where the expectation is over $x_0$.

While our definition of $V_t$ refers to an optimal policy $\phi_t^\star$, we can find $V_t$ (in principle) using a backward recursion that does not require knowledge of an optimal policy. We start with

$$V_T(z) = \min_v \ell_T(z, v),$$

and then find $V_{T-1}, V_{T-2}, \ldots, V_0$ from the recursion

$$V_t(z) = \min_v \left(\ell_t(z,v) + \mathbf{E}\, V_{t+1}(f_t(z) + g_t(z)v)\right),$$

for $t = T-1, T-2, \ldots, 0$. (The expectation here is over $(f_t, g_t)$).

Defining the Bellman operator $\mathcal{T}_t$ for the finite-horizon problem as

$$(\mathcal{T}_t h)(z) = \min_v \left(\ell_t(z,v) + \mathbf{E}\, h(f_t(z) + g_t(z)v)\right),$$

for $h : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$, we can express the recursion above compactly as

$$V_t = \mathcal{T}_t V_{t+1}, \quad t = T, T-1, \ldots, 0, \tag{3.1}$$

with $V_{T+1} = 0$.

We can express an optimal policy in terms of the value functions as

$$\phi_t^\star(z) = \operatorname*{argmin}_v \left(\ell_t(z,v) + \mathbf{E}\, V_{t+1}(f_t(z) + g_t(z)v)\right), \quad t = 0, \ldots, T.$$

**Discounted infinite-horizon.** For the discounted infinite-horizon problem, the value function is time-invariant, *i.e.*, does not depend on $t$. The value function $V(z)$ is the minimum cost-to-go from state $x_0 = z$ at $t = 0$, using an optimal policy:

$$V(z) = \sum_{t=0}^{\infty} \mathbf{E}\left(\gamma^t \ell(x_t, \phi^\star(x_t)) \;\middle|\; x_0 = z\right).$$

Thus, $J^\star = \mathbf{E}\, V(x_0)$.

The value function cannot be constructed using a simple recursion, as in the finite-horizon case, but it is characterized as the unique solution of the Bellman fixed point

equation $\mathcal{T}V = V$, where the Bellman operator $\mathcal{T}$ for the discounted infinite-horizon case is

$$(\mathcal{T}h)(z) = \min_v \left(\ell(z,v) + \gamma\, \mathbf{E}\, V(f(z) + g(z)v)\right),$$

for $h : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$.

The value function can be found (in principle) by several methods, including iteration of the Bellman operator,

$$V^{k+1} = \mathcal{T}V^k, \quad k = 1, 2, \ldots. \tag{3.2}$$

which is called value iteration. Under certain technical conditions, this iteration converges to $V$ [6]; see [14, Chapter 9] for detailed analysis of convergence of value iteration.

We can express an optimal policy in terms of the value function as

$$\phi^\star(z) = \operatorname*{argmin}_v \left(\ell(z,v) + \gamma\, \mathbf{E}\, V(f(z) + g(z)v)\right).$$

**Average cost.** For the average cost problem, the value function is not the cost-to-go starting from a given state (which would have the same value $J^\star$ for all states). Instead, it represents the differential sum of costs, *i.e.*,

$$V(z) = \sum_{t=0}^{\infty} \mathbf{E}\left(\ell(x_t, \phi^\star(x_t)) - J^\star \mid x_0 = z\right).$$

We can characterize the value function (up to an additive constant) and the optimal average cost as a solution of the fixed point equation $V + J^\star = \mathcal{T}V$, where $\mathcal{T}$,

the Bellman operator $\mathcal{T}$ for the average cost problem, is defined as

$$(\mathcal{T}h)(z) = \min_v \left(\ell(z,v) + \mathbf{E}\, h(f(z) + g(z)v)\right)$$

for $h : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$. Notice that if $V$ and $J^\star$ satisfy the fixed point equation, so do $V + \beta$ and $J^\star$ for any $\beta \in \mathbf{R}$. Thus, we can choose a reference state $x^{\text{ref}}$ and assume $V(x^{\text{ref}}) = 0$ without loss of generality.

Under some technical assumptions [12, 56], the value function $V$ and the optimal cost $J^\star$ can be found (in principle) by several methods, including value iteration, which has the form

$$\begin{aligned} J^{k+1} &= \mathcal{T}V^k(x^{\text{ref}}) \\ V^{k+1} &= \mathcal{T}V^k - J^{k+1}, \end{aligned} \tag{3.3}$$

for $k = 1, \ldots$. $V^k$ converges to $V$, and $J$ converges to $J^\star$ [10, 12].

We can express an optimal policy in terms of $V$ as

$$\phi^\star(z) = \operatorname*{argmin}_v \left(\ell(z,v) + \mathbf{E}\, V(f(z) + g(z)v)\right).$$

### 3.3.1 Linear-convex dynamic programming

When the dynamics is linear, the value function (or functions, in the finite-horizon case) is convex. To see this we first note that the Bellman operator maps convex functions to convex functions. The (random) function

$$f_t(z) + g_t(z)v = A_t z + B_t v + c_t$$

is an affine function of $(z, v)$; it follows that when $h$ is convex,

$$h(f_t(z) + g_t(z)v) = h(A_t z + B_t v + c_t)$$

is a convex function of $(z, v)$. Expectation preserves convexity, so

$$\mathbf{E}\, h(f_t(z) + g_t(z)v) = \mathbf{E}\, h(A_t z + B_t v + c_t)$$

is a convex function of $(z, v)$. Finally,

$$\min_v \left(\ell(z, v) + \mathbf{E}\, h(A_t z + B_t v + c_t)\right)$$

is convex since $\ell$ is convex, and convexity is preserved under partial minimization [19, §3.2.5].

It follows that the value function is convex. In the finite-horizon case, the argument above shows that each $V_t$ is convex. In the infinite-horizon and average cost cases, the argument above shows that value iteration preserves convexity, so if we start with a convex initial guess (say, 0), all iterates are convex. The limit of a sequence of convex functions is convex, so we conclude that $V$ is convex.

Evaluating the optimal policy at $x_t$, *i.e.*, minimizing

$$\ell_t(z, v) + \gamma\, \mathbf{E}\, V_t(f_t(z) + g_t(z)v)$$

over $v$, involves solving a convex optimization problem. In the finite-horizon and the average cost case, the discount factor $\gamma$ is 1. In the infinite-horizon cases (discounted and average cost), the stage cost and value function are time-invariant.

### 3.3.2 Linear-quadratic dynamic programming

For linear-quadratic stochastic control problems, we can effectively solve the stochastic control problem using dynamic programming. The reason is simple: The Bellman operator maps convex quadratic functions to convex quadratic functions, so the value function is also convex quadratic (and we can compute it by value iteration). The argument is similar to the convex case: convex quadratic functions are preserved under expectation and partial minimization.

One big difference, however, is that the Bellman iteration can actually be carried out in the linear-quadratic case, using an explicit representation of convex quadratic functions, and standard linear algebra operations. The optimal policy is affine, with coefficients we can compute. To simplify notation we consider the discounted infinite-horizon case; the other two cases are very similar.

We denote the set of convex quadratic functions on $\mathbf{R}^n$, *i.e.*, those with the form

$$h(z) = (1/2) \begin{bmatrix} z \\ 1 \end{bmatrix}^T \begin{bmatrix} H & h \\ h^T & p \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix},$$

where $H \succeq 0$, by $\mathcal{Q}^n$. With linear dynamics, we have

$$h(A_t z + B_t v + c_t) \;\; = (1/2) \begin{bmatrix} A_t z + B_t v + c_t \\ 1 \end{bmatrix}^T \begin{bmatrix} H & h \\ h^T & p \end{bmatrix} \begin{bmatrix} A_t z + B_t v + c_t \\ 1 \end{bmatrix},$$

where $(A_t, B_t, c_t)$ is a random variable. Convex quadratic functions are closed under expectation, so $\mathbf{E}\, h(A_t z + B_t v + c_t)$ is convex quadratic; the details (including formulas for the coefficients) are given in §A.

The stage cost $\ell$ is convex and quadratic, *i.e.*,

$$\ell(z,v) = (1/2)\begin{bmatrix} z \\ v \\ 1 \end{bmatrix}^T \begin{bmatrix} Q & S & q \\ S^T & R & r \\ q^T & r^T & s \end{bmatrix} \begin{bmatrix} z \\ v \\ 1 \end{bmatrix},$$

where

$$\begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \succeq 0.$$

Thus, $\ell(z,v) + \mathbf{E}\, h(A_t z + B_t v + c_t)$ is convex and quadratic. We write

$$\ell(z,v) + \mathbf{E}\, h(A_t z + B_t v + c_t) = (1/2)\begin{bmatrix} z \\ v \\ 1 \end{bmatrix}^T M \begin{bmatrix} z \\ v \\ 1 \end{bmatrix}.$$

where

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{12}^T & M_{22} & M_{23} \\ M_{13}^T & M_{23}^T & M_{33} \end{bmatrix}, \quad \begin{bmatrix} M_{11} & M_{12} \\ M_{12}^T & M_{22} \end{bmatrix} \succeq 0.$$

Partial minimization of $\ell(z,v) + \mathbf{E}\, h(A_t z + B_t v + c_t)$ over $v$ results in the quadratic function

$$(\mathcal{T}h)(z) = \min_v \left(\ell(z,v) + \mathbf{E}\, h(A_t z + B_t v + c_t)\right) = (1/2)\begin{bmatrix} v \\ 1 \end{bmatrix}^T \begin{bmatrix} S_{11} & S_{12} \\ S_{12}^T & S_{22} \end{bmatrix} \begin{bmatrix} v \\ 1 \end{bmatrix},$$

where

$$\begin{bmatrix} S_{11} & S_{12} \\ S_{12}^T & S_{22} \end{bmatrix} = \begin{bmatrix} M_{11} & M_{13} \\ M_{13}^T & M_{33} \end{bmatrix} - \begin{bmatrix} M_{12} \\ M_{23}^T \end{bmatrix} M_{22}^{-1} \begin{bmatrix} M_{12}^T & M_{23} \end{bmatrix}$$

is the Schur complement of $M$ (with respect to its (2,2) block); when $M_{22}$ is singular, we can replace $M_{22}^{-1}$ with the pseudo-inverse $M_{22}^{\dagger}$ [19, §A5.5]. Furthermore, since

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{12}^T & M_{22} \end{bmatrix} \succeq 0,$$

we can conclude that $S_{11} \succeq 0$, and thus $(\mathcal{T}h)(z)$ is a convex quadratic function, *i.e.*, $\mathcal{T}h \in \mathcal{Q}^n$.

This shows that quadratic convex functions are invariant under the Bellman operator $\mathcal{T}$. From this, it follows that the value function is convex and quadratic. Again, in the finite-horizon case this shows that each $V_t$ is convex and quadratic. In the infinite-horizon and average cost cases, this argument shows that a convex quadratic function is preserved under value iteration, so if the initial guess is convex and quadratic, all iterates are convex and quadratic. The limit of a set of convex quadratic functions is a convex quadratic, and thus we can conclude that $V$ is a convex quadratic function.

### 3.3.3 Approximate dynamic programming

Approximate dynamic programming is a general approach for obtaining a good suboptimal policy. The basic idea is to replace the true value function $V$ (or $V_t$ in the finite horizon case) with an approximation $\hat{V}$ ($\hat{V}_t$), which yields the approximate dynamic programming policies

$$\hat{\phi}_t(x) = \operatorname*{argmin}_u \left( \ell_t(x,u) + \mathbf{E}\, \hat{V}_{t+1}(f_t(x) + g_t(x)u) \right)$$

for the finite-horizon case,

$$\hat{\phi}(x) = \operatorname*{argmin}_u \left( \ell(x,u) + \gamma \, \mathbf{E} \, \hat{V}(f(x) + g(x)u) \right)$$

for the discounted infinite-horizon case, and

$$\hat{\phi}(x) = \operatorname*{argmin}_u \left( \ell(x,u) + \mathbf{E} \, \hat{V}(f(x) + g(x)u) \right)$$

for the average cost case.

These ADP policies reduce to the optimal policy for the choice $\hat{V}_t = V_t$. The goal is to choose these approximate value functions so that

- The minimizations needed to compute $\hat{\phi}(x)$ can be efficiently carried out.
- The objective $\hat{J}$ with the resulting policy is small, hopefully, close to $J^\star$.

It has been observed in practice that good policies (*i.e.*, ones with small objective values) can result even when $\hat{V}$ is not a particularly good approximation of $V$.

### 3.3.4 Quadratic ADP

In quadratic ADP, we take $\hat{V} \in \mathcal{Q}^n$ (or $\hat{V}_t \in \mathcal{Q}^n$ in the time-varying case), *i.e.*,

$$\hat{V}(z) = (1/2) \begin{bmatrix} z \\ 1 \end{bmatrix}^T \begin{bmatrix} P & p \\ p^T & q \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix},$$

where $P \succeq 0$. With input-affine dynamics, we have

$$\hat{V}(f_t(z) + g_t(z)v) = (1/2) \begin{bmatrix} f_t(z) + g_t(z)v \\ 1 \end{bmatrix}^T \begin{bmatrix} P & p \\ p^T & q \end{bmatrix} \begin{bmatrix} f_t(z) + g_t(z)v \\ 1 \end{bmatrix},$$

where $(f_t, g_t)$ is a random variable with known mean and covariance matrix. Thus, the expectation $\mathbf{E}\,\hat{V}(f_t(z) + g_t(z)v)$ is a convex quadratic function of $v$, with coefficients that can be computed using the first and second moments of $(f_t, g_t)$; see appendix A.

A special case is when the cost function $\ell(z, v)$ (and $\ell_t(z, v)$ for the time-varying case) is QP-representable. Then, for a given $z$, the quadratic ADP policy

$$\hat{\phi}(z) = \operatorname*{argmin}_v \Big(\ell(z, v) + \mathbf{E}\,\hat{V}(f_t(z) + g_t(z)v)\Big)$$

can be evaluated by solving a QP.

## 3.4 Projected value iteration

Projected value iteration is a standard method that can be used to obtain an approximate value function, from a given (finite-dimensional) set of candidate approximate value functions, such as $\mathcal{Q}^n$. The idea is to carry out value iteration, but to follow each Bellman operator step with a step that approximates the result with an element from the given set of candidate approximate value functions. The approximation step is typically called a projection and is denoted $\Pi$, since it maps into the set of candidate approximate value functions, but it need not be a projection (*i.e.*, it need not necessarily minimize any distance measure). The resulting method for computing an approximate value function is called projected value iteration.

Projected value iteration has been explored by many authors [12, Chapter 6], [33, Chapter 12], [55], [49], [4]. Here, we are considering scenarios in which the state space and action space are continuous, and establish a framework to carry out projected value iteration for those problems.

Projected value iteration has the following form for our three problems. For

the finite-horizon problem, we start from $\hat{V}_{T+1} = 0$ and compute approximate value functions as

$$\hat{V}_t = \Pi \mathcal{T}_t \hat{V}_{t+1}, \quad t = T-1, T-2, \ldots, 1.$$

For the discounted infinite-horizon problem, we have the iteration

$$\hat{V}^{(k+1)} = \Pi \mathcal{T} \hat{V}^{(k)}, \quad k = 1, 2, \ldots.$$

There is no reason to believe that this iteration always converges, although as a practical matter it typically does. In any case, we terminate after some number of iterations, and we judge the whole method not in terms of convergence of projected value iteration, but in terms of performance of the policy obtained.

For the average cost problem, projected value iteration has the form

$$\begin{aligned} \hat{J}^{(k+1)} &= \Pi \mathcal{T} \hat{V}^{(k)}(x^{\mathrm{ref}}) \\ \hat{V}^{(k+1)} &= \Pi \mathcal{T}(\hat{V}^{(k)} - \hat{J}^{(k+1)}), \quad k = 1, 2, \ldots. \end{aligned}$$

Here too we cannot guarantee convergence, although it typically does in practice.

In the expressions above, $\hat{V}^k$ are elements in the given set of candidate approximate value functions, but $\mathcal{T}\hat{V}^k$ is usually not; indeed, we have no general way of representing $\mathcal{T}\hat{V}^k$. The iterations above can be carried out, however, since $\Pi$ only requires the evaluation of $(\mathcal{T}\hat{V}^k)(x^{(i)})$ (which are numbers) at a finite number of points $x^{(i)}$.

### 3.4.1 Quadratic projected iteration

We now consider the case of quadratic approximate dynamic programming, *i.e.*, our subset of candidate approximate Lyapunov functions is $\mathcal{Q}^n$. In this case the expectation appearing in the Bellman operator can be carried out exactly. We can evaluate $(\mathcal{T}\hat{V}^k)(x)$ for any $x$, by solving a convex optimization problem; when the stage cost is QP-representable, we can evaluate $(\mathcal{T}\hat{V}^k)(x)$ by solving a QP.

We form $\Pi\mathcal{T}\hat{V}^k$ as follows. We choose a set of sampling points $x^{(1)}, \ldots, x^{(N)} \in \mathbf{R}^n$, and evaluate $z^{(i)} = (\mathcal{T}\hat{V}^k)(x^{(i)})$, for $i = 1, \ldots, N$. We take $\Pi\mathcal{T}\hat{V}^k$ as any solution of the least-squares fitting problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^N (V(x^{(i)}) - z^{(i)})^2 \\ \text{subject to} & V \in \mathcal{Q}^n, \end{array}$$

with variable $V$. In other words, we simply fit a convex quadratic function to the values of the Bellman operator at the sample points. This requires the solution of a semidefinite program (SDP) [16, 68].

Many variations on this basic projection method are possible. We can use any convex fitting criterion instead of a least-squares criterion, for example, the sum of absolute errors. We can also add additional constraints on $V$, that can represent prior knowledge (beyond convexity). For example, we may have a quadratic lower bound on the true value function, and we can add this condition as a stronger constraint on $V$ than simply convexity. (Quadratic lower bounds can be found using the methods described in [71, 72, 17].)

Another variation on the simple fitting method described above adds proximal regularization to the fitting step. This means that we add a term of the form

$(\rho/2)\|V - V^k\|_F^2$ to the objective, which penalizes deviations of $V$ from the previous value. Here $\rho \geq 0$ is parameter, and the norm is the Frobenius norm of the coefficient matrices,

$$\|V - V^k\|_F = \left\| \begin{bmatrix} P & p \\ p^T & q \end{bmatrix} - \begin{bmatrix} P^k & p^k \\ (p^k)^T & q^k \end{bmatrix} \right\|_F.$$

**Choice of sampling points.** The choice of the evaluation points affects our projection method, so it is important to use at least a reasonable choice. An ideal choice would be to sample points from the state in that period, under the final ADP policy chosen. But this cannot be done: We need to run projected value iteration to get the ADP policy, and we need the ADP policy to sample the points to use for $\Pi$.

A good method is to start with a reasonable choice of sample points, and use these to construct an ADP policy. Then we run this ADP policy to obtain a new sampling of $x_t$ for each $t$. We now repeat the projected value iteration, using these sample points. This sometimes gives an improvement in performance.

## 3.5 Input-constrained linear quadratic control

Our first example is a traditional time-invariant linear control system, with dynamics

$$x_{t+1} = Ax_t + B_t u_t + w_t,$$

which has the form used in this chapter with

$$f_t(z) = Az + w_t, \quad g_t(z) = B.$$

Here $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{n\times m}$ are known, and $w_t$ is an IID random variable with zero mean $\mathbf{E}\, w_t = 0$ and covariance $\mathbf{E}\, w_t w_t^T = W$. The stage cost is the traditional quadratic one, plus a constraint on the input amplitude:

$$\ell(x_t, u_t) = \begin{cases} x_t^T Q x_t + u_t^T R u_t & \|u_t\|_\infty \leq U_{\max} \\ \infty & \|u_t\|_\infty > U_{\max}, \end{cases}$$

where $Q \succeq 0$ and $R \succeq 0$. We consider the average cost problem.

**Numerical instance.** We consider a problem with $n = 10$, $m = 2$, $U_{\max} = 1$, $Q = 10I$ and $R = I$. The entries of $A$ are chosen from $\mathcal{N}(0, I)$, and the entries of $B$ are chosen from a uniform distribution on $[-0.5, 0.5]$. The matrix $A$ is then scaled so that $\lambda_{\max}(A) = 1$. The noise distribution is normal, $w_t \sim \mathcal{N}(0, I)$.

**Method parameters.** We start from the $x_0 = 0$ at iteration 0, and $\hat{V}^{(1)} = V^{\text{quad}}$, where $V^{\text{quad}}$ is the (true) quadratic value function when the stage cost is replaced with $\ell^{\text{quad}}(z, v) = z^T Q z + v^T R v$ for all $(z, v)$.

In each iteration of projected value iteration, we run the ADP policy for $N = 1000$ time steps and evaluate $\mathcal{T}\hat{V}^{(k)}$ for each of these sample points. We then fit a new quadratic approximate value function $\hat{V}^{(k+1)}$, enforcing a lower bound $V^{\text{quad}}$ and a proximal term with $\rho = 0.1$. Here the proximal term has little effect on the performance of the algorithm, and we could have eliminated the proximal at not much cost to the performance of the algorithm. We use $x_N$ to start the next iteration. For comparison, we also carry out our method with the more naive initial choice $\hat{V}^{(1)} = Q$, and we drop the lower bound $V^{\text{quad}}$.

**Results.** Figure 3.1 shows the performance $J^k$ of the ADP policy (evaluated by a simulation run of 10000 steps), after each round of projected value iteration. The performance of the policies obtained using the naive parameter choices are denoted by $J^{(k)}_{\text{naive}}$. For this problem, we can evaluate a lower bound $J_{\text{lb}} = 785.4$ on $J^*$ using the method of [71]. This shows that the performance of our ADP policy at termination is at most 24% suboptimal (but likely much closer to optimal). The plots show that the naive initialization only results in a few more projected value iteration steps. Close examination of the plots shows that the performance need not improve in each step. For example, the best performance (but only by a very small amount) with the more sophisticated initialization is obtained in iteration 9.

**Computation timing.** The timing results here are reported for a 3.4GHz Intel Xeon processor running Linux. Evaluating the ADP policy requires solving a small QP with 10 variables and 8 constraints. We use CVGXEN [43] to generate custom C code for this QP, which executes in around 50 $\mu$s. The simulation of 1000 time steps, which we use in each projected value iteration, requires around 0.05 s. The fitting is carried out using CVX [31], and takes around 1 s (this time could be speeded up considerably).

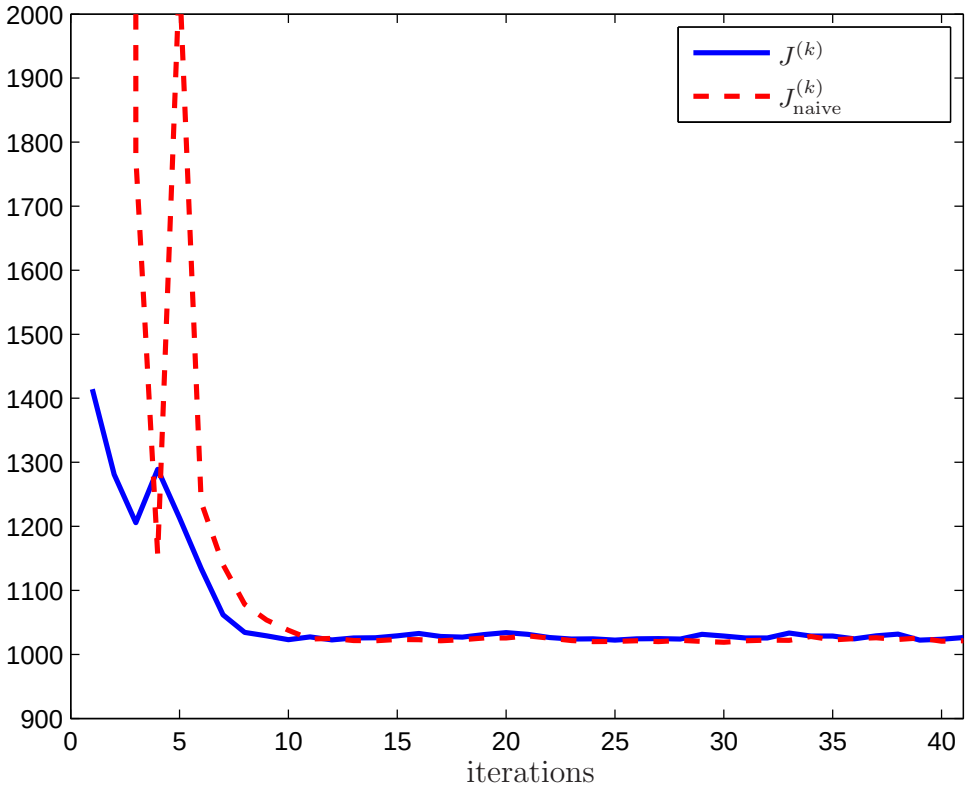


**Figure 3.1:** $J^{(k)}$ (blue solid), and $J^{(k)}_{\text{naive}}$ (red dashed).

## 3.6 Multi-period portfolio optimization

We consider a discounted infinite-horizon problem of optimizing a portfolio of $n$ assets with discount factor $\gamma$. The positions at time $t$ is denoted by $x_t \in \mathbf{R}^n$, where $(x_t)_i$ denotes the dollar value of asset $i$ at the beginning of time period $t$. At each time $t$, we can buy and sell assets. We denote the trades by $u_t \in \mathbf{R}^n$, where $(u_t)_i$ denotes the trade for asset $i$. A positive $(u_t)_i$ means that we buy asset $i$, and a negative $(u_t)_i$ means that we sell asset $i$ at time $t$. The position evolves according to the dynamics with

$$x_{t+1} = \mathbf{diag}(r_t)(x_t + u_t),$$

where $r_t$ is the vector of asset returns in period $t$. We express this in our form as

$$f_t(z) = \mathbf{diag}(r_t)z, \quad g_t(z) = \mathbf{diag}(r_t).$$

The return vectors are IID with mean $\mathbf{E}\, r_t = \bar{r}$ and covariance $\mathbf{E}(r_t - \bar{r})(r_t - \bar{r})^T = \Sigma$.

The stage cost consists of the following terms: the total gross cash put in, which is $\mathbf{1}^T u_t$; a risk penalty (a multiple of the variance of the post-trade portfolio); a quadratic transaction cost; and the constraint that the portfolio is long-only (which is $x_t + u_t \geq 0$). It is given by

$$\ell(x,u) = \begin{cases} \mathbf{1}^T u + \lambda (x+u)^T \Sigma (x+u) + u^T \mathbf{diag}(s) u & x + u \geq 0 \\ \infty & \text{otherwise,} \end{cases}$$

where $\lambda \geq 0$ is the risk aversion parameter, and $s_i \geq 0$ models the price-impact cost for asset $i$. We assume that the initial portfolio $x_0$ is given.

**Numerical instance.** We will consider $n = 10$ assets, a discount factor of $\gamma = 0.99$, and an initial portfolio of $x_0 = 0$. The returns $r_t$ are IID and have a log-normal distribution, *i.e.*,

$$\log r_t \sim \mathcal{N}(\mu, \tilde{\Sigma}).$$

The means $\mu_i$ are chosen from a $\mathcal{N}(0, 0.01^2)$ distribution; the diagonal entries of $\tilde{\Sigma}$ are chosen from a uniform $[0, 0.01]$ distribution, and the off-diagonal entries are generated using $\tilde{\Sigma}_{ij} = C_{ij}(\Sigma_{ii}\Sigma_{jj})^{1/2}$, where $C$ is a correlation matrix, generated randomly. Since the returns are log-normal, we have

$$\bar{r} = \exp(\mu + (1/2)\,\mathbf{diag}(\tilde{\Sigma})), \quad \Sigma_{ij} = \bar{r}_i\bar{r}_j(\exp\tilde{\Sigma}_{ij} - 1).$$

We take $\lambda = 0.5$, and choose the entries of $s$ from a uniform $[0, 1]$ distribution.

**Method parameters.** We start with $\hat{V}^{(1)} = V^{\mathrm{quad}}$, where $V^{\mathrm{quad}}$ is the (true) quadratic value function when the long-only constraint is ignored. Furthermore, $V^{\mathrm{quad}}$ is a lower bound on the true value function $V$.

We choose the evaluation points in each iteration of projected value iteration by starting from $x_0$ and running the policy induced by $\hat{V}^{(k)}$ for 1000 steps. We then fit a quadratic approximate value function, enforcing a lower bound $V^{\mathrm{quad}}$ and a proximal term with $\rho = 0.1$. The proximal term in this problem has a much more significant effect on the performance of projected value iteration – smaller values of $\rho$ require many more iterations of projected value iteration to achieve the same performance.

**Results.** Figure 3.2 demonstrates the performance of the ADP policies, where $J^{(k)}$ corresponds to the ADP policy at iteration $k$ of projected value iteration. In each case we run 1000 Monte Carlo simulations, each for 1000 steps and report the average total

discounted cost across the 1000 time steps. We also plot a lower bound $J_{\text{lb}}$ obtained using the methods described in [73, 17]. We can see that after around 80 steps of projected value iteration, we arrive at a policy that is nearly optimal. It is also interesting to note that it takes around 40 projected value iteration steps before we produce a policy that is profitable (*i.e.*, has $J \leq 0$).

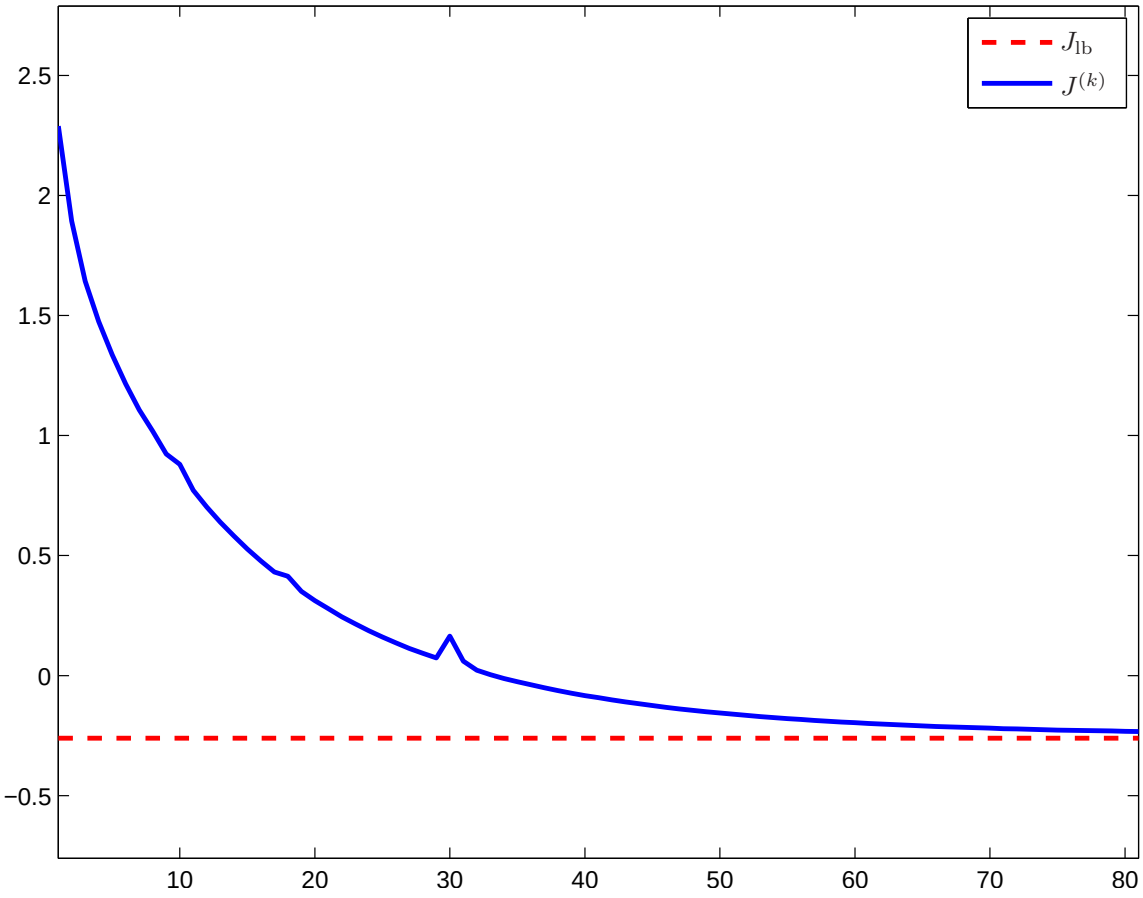


**Figure 3.2:** $\hat{J}^{(k)}$ (blue solid), and a lower bound $J_{\text{lb}}$ (red dashed).

**Computation timing.** Evaluating the ADP policy requires solving a small QP with 10 variables and 10 constraints. We use CVGXEN [43] to generate custom C

code for this QP, which executes in around 22 $\mu$s. The simulation of 1000 time steps, which we use in each projected value iteration, requires around 0.022 s. The fitting is carried out using CVX [31], and takes around 1.2 s (this time could be speeded up considerably).

## 3.7 Vehicle control

We will consider a rigid body vehicle moving in 2 dimensions, with continuous time state $x_c = (p, v, \theta, \omega)$, where $p \in \mathbf{R}^2$ is the position, $v \in \mathbf{R}^2$ is the velocity, $\theta \in \mathbf{R}$ is the angle (orientation) of the vehicle, and $\omega$ is the angular velocity. The vehicle is subject to control forces $u_c \in \mathbf{R}^k$, which put a net force and torque on the vehicle. The vehicle dynamics in continuous time is given by

$$\dot{x}_c = A_c x_c + B_c(x_c)u_c + w_c,$$

where $w_c$ is a continuous time random noise process and

$$A_c = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B_c(x) = \begin{bmatrix} 0 \\ R(x_5)C/m \\ 0 \\ D/I \end{bmatrix},$$

with

$$R(\theta) = \begin{bmatrix} \sin(\theta) & \cos(\theta) \\ -\cos(\theta) & \sin(\theta) \end{bmatrix}.$$

The matrices $C \in \mathbf{R}^{2\times k}$ and $D \in \mathbf{R}^{1\times k}$ depend on where the forces are applied with respect to the center of mass of the vehicle.

We consider a discretized forward Euler approximation of the vehicle dynamics with discretization epoch $h$. With $x_c(ht) = x_t$, $u_c(ht) = u_t$, and $w_c(ht) = w_t$, the

discretized dynamics has the form

$$x_{t+1} = (I + hA_c)x_t + hB_c(x_t)u_t + hw_t,$$

which has input-affine form with

$$f_t(z) = (I + hA_c)z + hw_t, \quad g_t(z) = hB_c(z).$$

We will assume that $w_t$ are IID with mean $\bar{w}$ and covariance matrix $W$.

We consider a finite-horizon trajectory tracking problem, with desired trajectory given as $x_t^{\mathrm{des}}$, $t = 1, \ldots, T$. The stage cost at time $t$ is

$$\ell_t(z, v) = \begin{cases} v^T R v + (z - x_t^{\mathrm{des}})^T Q (z - x_t^{\mathrm{des}}) & |u| \leq u_{\max} \\ \infty & |u| > u_{\max}, \end{cases}$$

where $R \in \mathbf{S}_+^k$, $Q \in \mathbf{R}^{n\times\ n}$, $u_{\max} \in \mathbf{R}_+^k$, and the inequalities in $u$ are element-wise.

**Numerical instance.** We consider a vehicle with unit mass $m = 1$ and moment of inertia $I = 1/150$. There are $k = 4$ inputs applied to the vehicle, shown in Figure 3.3, at distance $r = 1/40$ to the center of mass of the vehicle. We consider $T = 151$ time steps, with a discretization epoch of $h = \pi/150$. The matrices $C$ and $D$ are

$$C = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \quad D = \frac{r}{\sqrt{2}} \begin{bmatrix} 0 & 1 & 0 & -1 \end{bmatrix}.$$

The maximum allowable input is $u_{\max} = (1.5, 2.5, 1.5, 2.5)$, and the input cost matrix is $R = I/1000$. The state cost matrix $Q$ is a diagonal matrix, chosen as follows: we first choose the entries of $Q$ to normalize the range of the state variables, based on

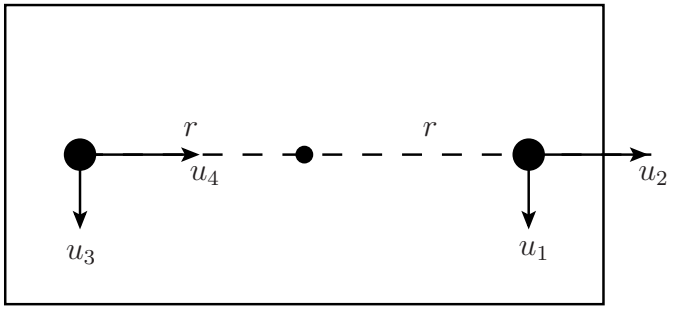


**Figure 3.3:** Vehicle diagram. The forces are applied at two positions, shown with large circles. At each position there are two forces applied: a lateral force ($u_1$ and $u_3$), and a longitudinal force ($u_2$ and $u_4$).

$x^{\text{des}}$. Then we multiply the weight of the first and second state by 10 to penalize a deviation in the position more than a deviation in the speed or orientation of the vehicle.

**Method parameters.** We start at the end of the horizon, with $V_T = \min_v \ell(z, v)$, which is quadratic and convex in $z$. At each step of projected value iteration $t$, we choose $N = 1000$ evaluation points, generated using a $\mathcal{N}(x_t^{\text{des}}, 0.5I)$ distribution. We evaluate $\mathcal{T}_t\hat{V}_{t+1}$ for each evaluation point and fit the quadratic approximate value function $\hat{V}_t$.

**Results.** We calculate the average total cost using Monte Carlo simulation with 1000 samples, *i.e.*, 1000 simulations of 151 steps. We compare the performance of two ADP policies: the ADP policy resulting from $\hat{V}_t$ results in $J = 106$, and the ADP policy resulting from $V_t^{\text{quad}}$ results in $J^{\text{quad}} = 245$. Here $V_t^{\text{quad}}$ is the quadratic value function obtained by linearizing the dynamics along the desired trajectory, and ignoring the actuator limits. Figure 3.4 shows two sample trajectories generated using

$\hat{V}_t$ and $V_t^{\mathrm{quad}}$.

**Computation timing.** Evaluating the ADP policy requires solving a small QP with 4 variables and 8 constraints. We use CVGXEN [43] to generate custom C code for this QP, which executes in around 20 $\mu$s. The simulation of 1000 time steps, which we use in each projected value iteration, requires around 0.02 s. The fitting is carried out using CVX [31], and takes around 0.5 s (this time could be speeded up considerably).

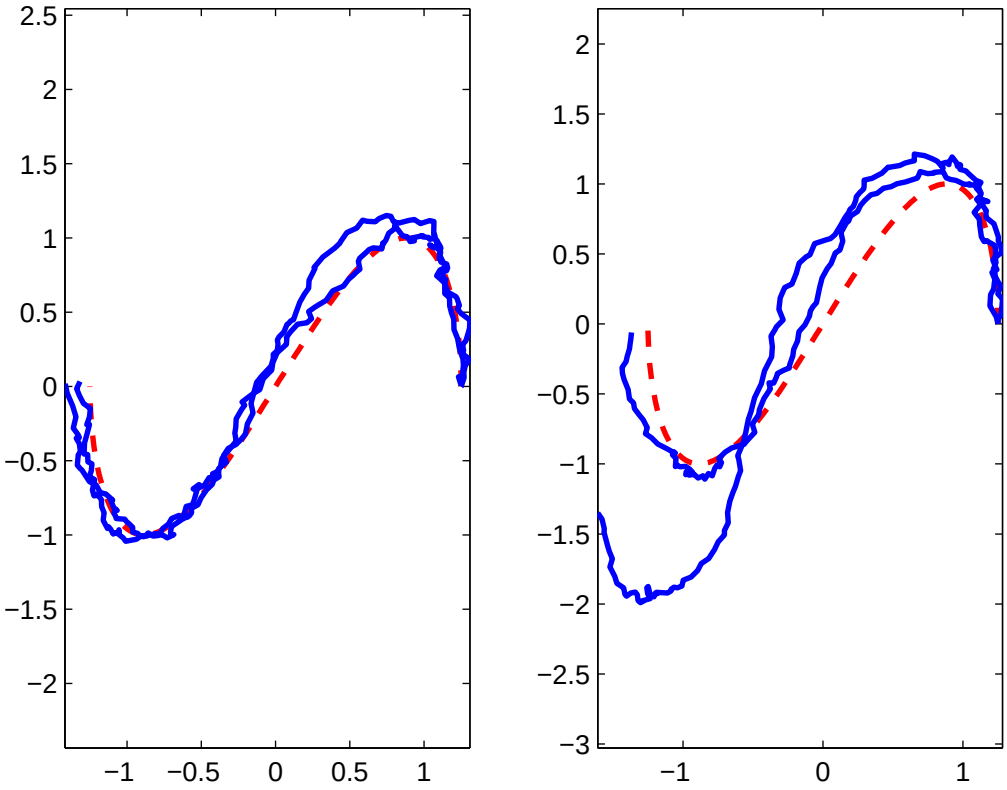


**Figure 3.4:** Desired trajectory (red dashed), two sample trajectories obtained using $\hat{V}_t$, $t = 1, \ldots, T$ (left), and two sample trajectories obtained using $V_t^{\text{quad}}$ (right).

## 3.8 Conclusions

We have shown how to carry out projected value iteration with quadratic approximate value functions, for input-affine systems. Unlike value iteration (when the technical conditions hold), projected value iteration need not converge, but it typically yields a policy with good performance in a reasonable number of steps. To evaluate the quadratic ADP policy requires the solution of a (typically small) convex optimization problem, or more specifically a QP when the stage cost is QP-representable. Recently developed fast methods, and code generation techniques, allow such problems to be solved very quickly, which means that quadratic ADP can be carried out at kilohertz (or faster) rates. Our ability to evaluate the policy very quickly makes projected value iteration practical, since many evaluations can be carried out in reasonable time. While we offer no theoretical guarantees on the performance attained, we have observed that the performance is typically very good.

# Chapter 4

# Case Study: Supply Chain Control

## 4.1 Introduction

In this section we will consider the supply chain management problem as a case study, and will use the methods explained in chapters 2 and 3 to come up with approximate dynamic programming policies.

Supply chain management considers the problem of managing a network of interconnected nodes and the products/services moving through those nodes, with the goal of optimizing an objective (*e.g.*, storage cost, shipping cost, delay, *etc.*). Here we will consider a simple scenario: a single commodity moving through a chain of interconnected nodes.

## 4.2 Problem description

We will manage the flow of a single commodity across a chain consisting of $n$ nodes (warehouses/buffers) and $m$ links. At time $t$, the amount of commodity at node $i$ is $(x_t)_i$, and the amount of commodity transported along link $j$ is $(u_t)_j$. The supply

chain follows linear dynamics over time

$$x_{t+1} = x_t + (A^{\text{in}} - A^{\text{out}})u_t,$$

where

$$A_{i,j}^{\text{in(out)}} = \begin{cases} 1 & \text{link } j \text{ enters (exits) node } i \\ 0 & \text{otherwise.} \end{cases}$$

**Stage cost.** At each time, there is a stage cost which includes the shipping cost and the warehouse cost. Infeasible state-action pairs are denoted by infinite cost. We will consider QP-representable stage costs that are differentiable in $u$.

The stage cost may depend on a disturbance/parameter $w$, which can change with time deterministically, or follow a random process. We can make the dependence of the stage cost on the parameter explicit by writing the stage cost as $\ell(x, u, w)$. We will assume that the parameter $w$, if it is random, is independent and identically distributed (IID) with distribution $\mathcal{W}$. The parameter $w$ can be used to represent various quantities such as price, maximum ordering limit, and varying storage/shipping prices.

The value of $w$ is known at the time of decision making, *i.e.*, the action $u_t$ is chosen after observing $w_t$.

**Objective.** The objective is to minimize the average expected cost,

$$\text{minimize} \quad \lim_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{T} \mathbf{E}\, \ell(x_t, u_t, w_t)$$

where the optimization variable is $u_t$, $t = 0, 1, \ldots$, and the expectation is taken over $w_t$. The initial state, $x_0$, is known and given. The stochastic control problem is to

find a control policy that minimizes the objective. We denote the optimal average cost by $J^\star$.

**Dynamic programming.** Using dynamic programming, we can characterize the value function $V : \mathbf{R}^n \to \mathbf{R}$ (up to an additive constant) and the optimal average cost $J^\star$ as the solution of a fixed point equation:

$$V(x) + J^\star = \mathbf{E} \min_u (\ell(x, u, w) + V(x + (A^{\text{in}} - A^{\text{out}})u)), \tag{4.1}$$

where the expectation is taken over $w$. Here the expectation is outside the minimization, because the value of $w$ is known when choosing $u_t$. Note that if $V$ and $J^\star$ satisfy the above fixed point equation, then so do $V + \beta$ and $J^\star$, for any $\beta \in \mathbf{R}$. Thus, we can choose a state $x^{\text{ref}}$ and assume $V(x^{\text{ref}}) = 0$ without loss of generality. The Bellman operator $\mathcal{T}$ is defined as

$$(\mathcal{T}h)(z) = \mathbf{E} \min_u \left(\ell(x, u, w) + h(x + (A^{\text{in}} - A^{\text{out}})u)\right),$$

where $h : \mathbf{R}^n \to \mathbf{R}$. So we can write the fixed point equation (4.1) in the compact form

$$V + J^\star = \mathcal{T}V.$$

For a given value of the parameter $w$, the optimal policy can be evaluated by solving the following optimization problem

$$\phi^\star(z, w) = \operatorname*{argmin}_u \left(\ell(x, u, w) + V(x + (A^{\text{in}} - A^{\text{out}})u)\right).$$

with variable $u$. Note that we drop the expectation over $w$ in evaluating the control policy, since the value of $w$ is known at the time of decision making. Thus the optimal policy is parametrized by the the parameter $w$.

It is important to note here that the value function $V$ is only a function of $x$, and not of $w$. The value function is modeling the *value* of being in a particular state in the next time step, thus it does not depend on the exact parameter value, since the parameter value for the next time step is unknown. Thus, calculating the Bellman iteration involves taking an expectation over $w$. However, the optimal policy depends on the value of $w$, because the value of $w$ at time $t$ is known before a decision is made. Therefore, the policy is a function of both the variable $x$ and the parameter $w$.

**Approximate dynamic programming.** If we replace $V$ with an approximate value function $\hat{V}$, the resulting control policy is an approximate dynamic programming (ADP) policy:

$$\hat{\phi}(z, w) = \operatorname*{argmin}_u \left( \ell(x, u, w) + \hat{V}(x + (A^{\text{in}} - A^{\text{out}})u) \right).$$

## 4.3 Methods

We will use two different methods of producing an ADP policy.

The first one is the method of chapter 2, which imputes an approximate value function by observing the behavior of an optimal or nearly optimal agent, given in the form

$$(w^{(i)}, x^{(i)}), \quad i = 1, \ldots, N,$$

where $x^{(i)}$ is the action chosen given parameter value $w^{(i)}$. The observed data may

be obtained in a variety of ways. For instance, it may be the data collected from a particular supply chain that is known to be managed very well, with the hope of being able to mimic the same behavior in a different supply chain. Or it may be collected by running a complex controller, such as model predictive control (MPC). We can then use the methods demonstrated in chapter 2 to produce an ADP policy that is simple, efficient to compute, and is able to mimic the behavior of the complex controller, thus providing significant complexity reduction in the management of the supply chain.

The second method is to use projected value iteration to fit a convex quadratic approximate value function, based on the work presented in chapter 3. We will follow each iteration of the Bellman operator with a projection step to make sure that the value function remains a convex quadratic function.

In the following section, we will consider convex quadratic approximate value functions. We denote the set of convex quadratic functions on $\mathbf{R}^n$, *i.e.*, those with the form

$$h(z) = (1/2) \begin{bmatrix} z \\ 1 \end{bmatrix}^T \begin{bmatrix} H & h \\ h^T & p \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix},$$

where $H \succeq 0$, by $\mathcal{Q}^n$.

## 4.4 Method parameters

### 4.4.1 Imputed objective

We consider the class of convex quadratic value functions, *i.e.*, $V \in \mathcal{Q}^n$ with the form

$$V(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} P & p \\ p^T & q \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix},$$

where $P \succeq 0$.

The KKT residuals $r_{\text{stat}}^{(k)}$ and $r_{\text{comp}}^{(k)}$ corresponding to the $k$th samples are

$$r_{\text{stat}}^{(k)} = p^{(k)} - \lambda_1^{(k)} + \lambda_2^{(k)} + A^{\text{out}T}\lambda_3^{(k)} + A^T\lambda_4^{(k)} + 2A^TPAu^{(k)} + 2A^TPx + 2A^Tp,$$

and

$$r_{\text{comp}}^{(k)} = \begin{bmatrix} \lambda_1^{(k)} \circ u^{(k)} \\ \lambda_2^{(k)} \circ (u^{(k)} - (u^{\max})^{(k)}) \\ \lambda_3^{(k)} \circ (A^{\text{out}}u^{(k)} - x^{(k)}) \\ \lambda_4^{(k)} \circ (Au^{(k)} + x^{(k)} - x^{\max}) \end{bmatrix},$$

where $\circ$ denotes the Hadamard (elementwise) product. To impute the value function, we use the method of chapter 2, with $\phi(r_{\text{stat}}, r_{\text{comp}}) = \|(r_{\text{stat}}, r_{\text{comp}})\|_2^2$. The resulting convex optimization problem is

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^N \|r_{\text{stat}}^{(k)}\|_2^2 + \|r_{\text{comp}}^{(k)}\|_2^2 \\ \text{subject to} & \lambda_i^{(k)} \succeq 0, \quad i = 1, \ldots, 4, k = 1, \ldots, N \\ & P \succeq 0, \end{array}$$

with variables $P, p, \lambda_i^{(k)}$, $i = 1, \ldots, 4$, $k = 1, \ldots, N$. Note that we have assumed that

$q = 0$, since the imputed objective function is invariant under scalar addition (since the resulting policy would be the same).

### 4.4.2 Projected value iteration

Projected value iteration has the form

$$\begin{aligned}\hat{J}^{(k+1)} &= \Pi\mathcal{T}\hat{V}^{(k)}(x^{\mathrm{ref}}) \\ \hat{V}^{(k+1)} &= \Pi\mathcal{T}(\hat{V}^{(k)} - \hat{J}^{(k+1)}), \quad k = 1, 2, \ldots.\end{aligned}$$

We will now explain how these two steps are carried out in more detail.

We choose a set of sampling points $x^{(1)}, x^{(2)}, \ldots, x^{(N)} \in \mathbf{R}^n$, and evaluate

$$z^{(i)} = \frac{1}{K}\sum_{k=1}^{K}\left(\ell(x, u, w_k) + V(x + (A^{\mathrm{in}} - A^{\mathrm{out}})u)\right),$$

where $w_k$, $k = 1, \ldots, K$ are parameter values generated according to distribution $\mathcal{W}$. Thus, $z^{(i)}$ is an unbiased estimate of

$$\mathbf{E}\left(\ell(x, u, w) + V(x + (A^{\mathrm{in}} - A^{\mathrm{out}})u)\right),$$

evaluated using a $K$-sample Monte Carlo simulation. We take $\Pi\mathcal{T}\hat{V}^{(k)}$ as any solution of the optimization problem

$$\begin{array}{ll}\text{minimize} & \left(\sum_{i=1}^{N}(V(x^{(i)}) - z^{(i)})^2\right)^{1/2} \\ \text{subject to} & V \in \mathcal{Q}^n,\end{array}$$

with variable $V$. This requires the solution of a semidefinite program (SDP).

## 4.5 Problem instance

We will consider parameters $w = (p, u^{\max})$, where $p$ is the price and $u^{\max}$ is the maximum ordering capacity. The cost function, parameterized by $w$, is

$$\ell(x, u, w) = p^T u + x^T Q x + q^T x,$$

subject to the constraints

$$0 \leq u_i \leq u_i^{\max}, \tag{4.2}$$

$$A^{\text{out}} u \leq x, \tag{4.3}$$

$$x + (A^{\text{in}} - A^{\text{out}})u \leq x^{\max}, \tag{4.4}$$

where $Q \in \mathbf{S}_+^n$, $q \in \mathbf{R}^n$, $x^{\max} \in \mathbf{R}^n$ are given, and $p \in \mathbf{R}^m$, $u^{\max} \in \mathbf{R}^m$ are parameters that vary over time. (The constraints can be incorporated into the cost function through an indicator function).

The first term in the cost function, $p^T u$, corresponds to shipping cost, where $p_i$ is the price for transporting one unit of the commodity across link $i$ and varies with time. The price can be negative, indicating the sales of the commodity to the outside world, or positive, indicating buying supplies from the outside world or incurring transportation costs. The second and third terms in the cost function, $x^T Q x + q^T x$, correspond to warehouse costs.

The first constraint (4.2), enforces non-negativity of flow on the links, and a maximum capacity given by $u^{\max}$, which is a parameter that varies over time. This can reflect varying supply and demand levels. The second constraint (4.3), makes sure that we never ship out more than is available. The third constraint (4.4), enforces maximum warehouse capacity $x^{\max}$.

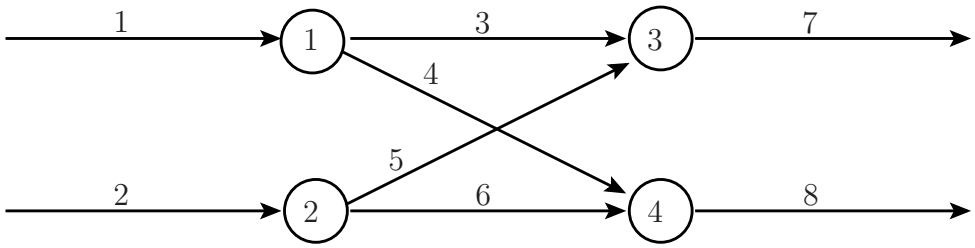


**Figure 4.1:** The supply chain and the corresponding connections. Links 1 and 2 are supply links, and links 7 and 8 are demand links.

### 4.5.1 Numerical instance

We consider a problem with $n = 4$, $m = 8$, and connections as shown in figure 4.1. Links 1 and 2 are for supply shipments, and links 7 and 8 are for demand shipments.

The price is a uniform random variable $p \sim U[a - \frac{w}{2}, a + \frac{w}{2}]$, with mean

$$a = \mathbf{E}\, p = (1, 1, 0.1, 0.1, 0.1, 0.1, -1.5, -1.5),$$

and window length of 0.1 for supply/demand links and 0 for other links, *i.e.*,

$$w = (0.1, 0.1, 0, 0, 0, 0, 0.1, 0.1).$$

The maximum flow $u^{\max}$ is also a uniform random variable $u^{\max} \sim U[b - \frac{w}{2}, b + \frac{w}{2}]$, with mean

$$b = \mathbf{E}\, u^{\max} = (1.5, 1.5, 0.5, 0.5, 0.5, 0.5, 1, 1),$$

and the same window length (0.1 for supply/demand links and 0 for other links).

## 4.6 Results

We will now present the results of applying the two ADP policies (from chapters 2 and 3).

### 4.6.1 Imputed objective

We generate data using MPC, with a lookahead of $T = 20$. We run MPC for 200 time steps, and record the corresponding parameters and variables. We then fit a quadratic approximate value function using the method of chapter 3.

Results are shown in figure 4.2, with the actions chosen using the MPC policy plotted on the $x$-axis, versus the actions chosen using the imputed ADP policy on the $y$-axis.

### 4.6.2 Projected value iteration

We start with $\hat{V}^{(1)} = 0$. In each iteration, we choose $N = 200$ points $x^{(i)}$ and run a Monte Carlo simulation with $K = 500$. We choose $x^{(i)}$ as states visited using an MPC controller with a lookahead of $T = 20$. We run 10 iterations of projected value iteration, and show the performance of the ADP policy obtained at termination. Figure 4.3 plots the actions chosen using the MPC policy on the $x$-axis, versus the actions chosen using the ADP policy on the $y$-axis.

### 4.6.3 Performance comparison

Figure 4.4 compares the performance of three ADP policies: model predictive control (MPC), imputed value function (IMP), and projected value iteration (PVI). We

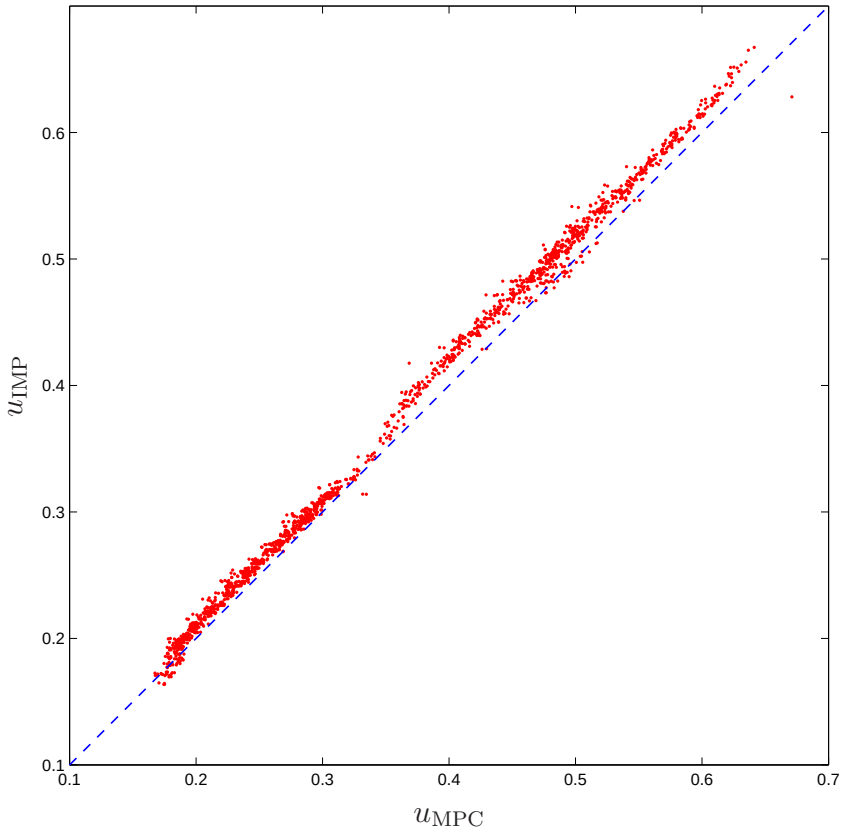


**Figure 4.2:** Scatter plot of the flows generated using MPC, versus the flows realized using the imputed ADP policy (IMP). The dashed line is the $y = x$ line.

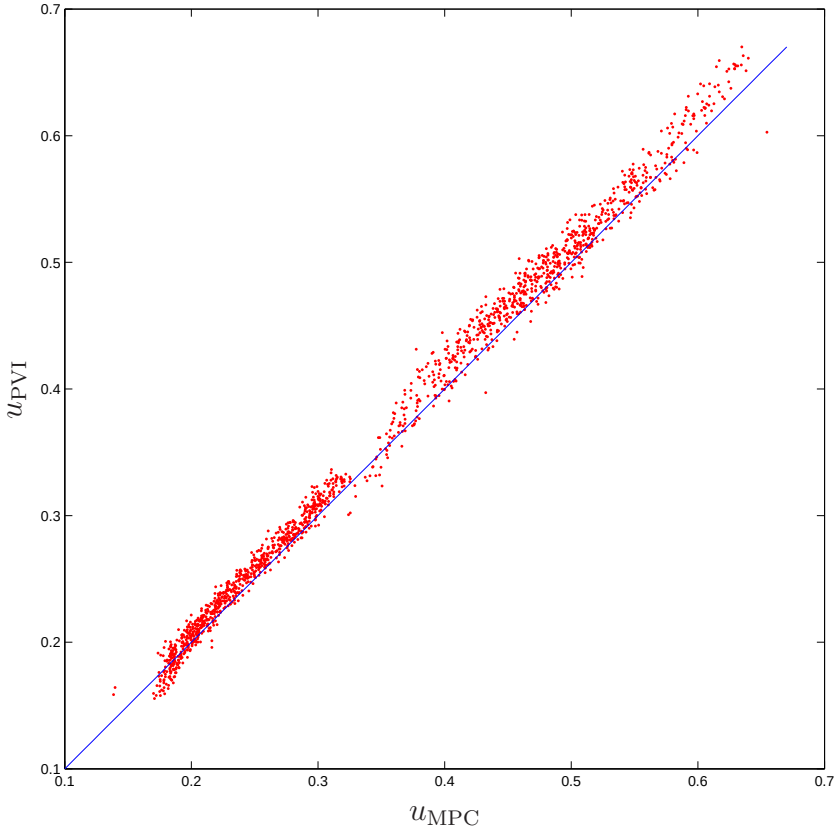


**Figure 4.3:** Scatter plot of the flows generated using MPC, versus the flows realized using the ADP policy from projected value iteration (PVI). The dashed line is the $y = x$ line.

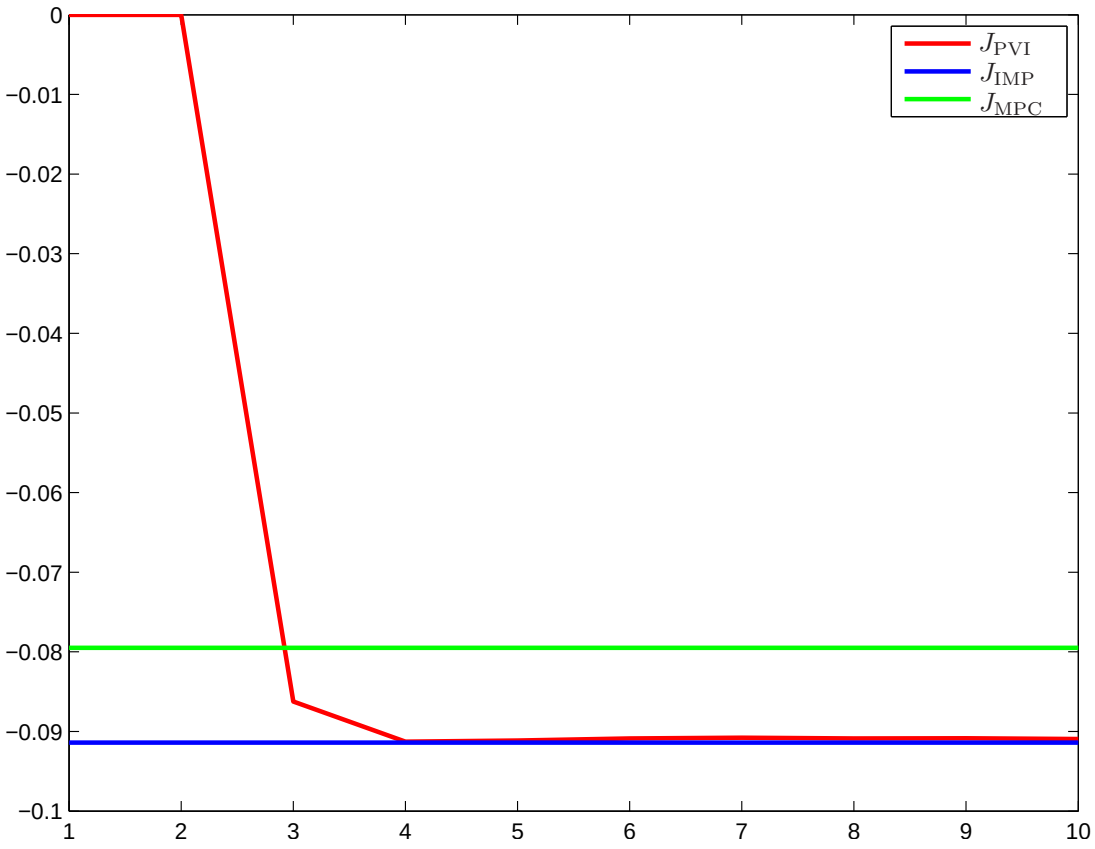


**Figure 4.4:** Performance of the three ADP methods: model predictive control (MPC), imputed objective (IMP), and projected value iteration (PVI).

| calculating $\hat{V}$ | imputed | PVI |
|---|---|---|
| imputing | $232s$ | – |
| evaluating $\hat{\phi}^k$ | – | $200\mu s$ |
| generating 200 samples (500 MC iters for each) | – | $20s$ |
| fitting $\hat{V}^{k+1}$ | – | $0.18s$ |
| total | $232s$ | $101s$ |
| evaluating $\hat{\phi}$ | $200\mu s$ | $200\mu s$ |

**Table 4.1:** Comparing runtime performance of the two methods for producing ADP policies. Top part of the table outlines the time it takes to calculate $\hat{V}$ using either method. Bottom part compares the policy runtime. Imputed refers to the method of chapter 2; PVI refers to the method of chapter 3.

observe that both $V_{\text{IMP}}$ and $V_{\text{AVI}}$ obtain very good performance that is even slightly better than that of the MPC controller, which is of much higher complexity.

### 4.6.4 Runtime comparison

Using the first method (which we will denote by *imputed* in table 4.1), imputing the value function takes around $232s$, resulting in a policy that can be evaluated by solving a quadratic program (QP). Each QP has 8 variables and 24 affine inequality constraints, and can be solved in $200\mu s$.

In the second method (projected value iteration, or PVI), at each iteration we calculate $(\mathcal{T}V)(x^{(i)})$ at 200 points, each of which involves 500 Monte Carlo simulations on $w$. Each calculation is a QP which takes around $200\mu s$. Thus, generating the samples for each iteration, which involves solving 10000 QPs, takes 20 seconds. After generating the samples, we solve an SDP to fit the new approximate value function, which takes 0.18 seconds. The performance doesn't change significantly after the fourth iteration, so the total runtime is around $5 \times (20.18) = 100.9$ seconds.

### 4.6.5 Parallelization

There are many parts of the computation that could be carried out in parallel. We will now explain these briefly.

**Imputed objective method.** We can use the alternating direction of multiplier method (ADMM) to spread the computation along $N$ different cores and enforce a consensus constraint on the imputed function weights, $\alpha_i$. This will reduce the time it takes to impute the value function. For a complete overview of the ADMM algorithm, see [18].

**Projected value iteration.** We can carry out each evaluation of the policy $\hat{\phi}^k(x, w)$ for each $x$ and $w$ in parallel. For instance, if we have 10 cores, the time per iteration would be $2.0 + 0.18 = 2.18$ seconds, which results in a total fitting time of around 11 seconds.

## 4.7 Conclusion

We have demonstrated the use of two methods for designing ADP policies for the supply chain management problem. The first method is based on chapter 2 and uses observations from an optimal or approximately optimal controller to design an ADP policy. The second method uses projected value iteration as explained in chapter 3. We have observed that both methods will generate ADP policies with performance comparable to the more complicated controllers such as MPC. Furthermore, evaluating each of the ADP policies involves solving a QP which can be calculated on the order of $\mu s$.

Even though both methods generate very good ADP policies, there are some fundamental differences between them which we discuss here briefly. The first difference is that the imputed objective method of §2 needs access to data from an optimal or nearly optimal controller, whereas in projected value iteration we do not have this requirement as data is generated using the available estimates of the value function. The second difference is that in the imputed objective method, we will only solve one large optimization problem to impute the value function, whereas in the projected value iteration method, we iterate until satisfactory performance is achieved, or the estimate of value function does not change much. There are no guarantees that projected value iteration will converge, but we have observed that terminating after a reasonable number of steps often results in good performance. An advantage of using the imputed objective method is that it will also provide us with KKT residuals that hint at the plausibility of the model. Lastly, in the imputed objective method the cost function and the constraints must be differentiable in the action $u$, whereas in the projected value iteration method this constraint need not hold (in fact, we only need the cost function and the constraints to be convex).

# Appendix A

# Expectation of quadratic function

In this appendix, we will show that when $h : \mathbf{R}^n \to \mathbf{R}$ is a convex quadratic function, so is $\mathbf{E}\, h(f_t(z) + g_t(z)v)$. We will compute the explicit coefficients $A$, $a$, and $b$ in the expression

$$\mathbf{E}\, h(f_t(z) + g_t(z)v) = (1/2) \begin{bmatrix} v \\ 1 \end{bmatrix}^T \begin{bmatrix} A & a \\ a^T & b \end{bmatrix} \begin{bmatrix} v \\ 1 \end{bmatrix}. \tag{A.1}$$

These coefficients depend only on the first and second moments of $(f_t, g_t)$. To simplify notation, we write $f_t(z)$ as $f$ and $g_t(z)$ as $g$.

Suppose $h$ has the form

$$h(z) = (1/2) \begin{bmatrix} z \\ 1 \end{bmatrix}^T \begin{bmatrix} P & p \\ p^T & q \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix},$$

with $F \succeq 0$. With input-affine dynamics, we have

$$h(f + gv) = (1/2) \begin{bmatrix} f + gv \\ 1 \end{bmatrix}^T \begin{bmatrix} P & p \\ p^T & q \end{bmatrix} \begin{bmatrix} f + gv \\ 1 \end{bmatrix},$$

Taking expectation, we see that $\mathbf{E}\, h(f + gv)$ takes the form of (A.1), with coefficients given by

$$A_{ij} = \sum_{k,l} P_{kl}\, \mathbf{E}(g_{ki} g_{lj}), \quad i, j = 1, \ldots, n$$
$$a_i = \sum_{j} p_j\, \mathbf{E}(g_{ji}), \quad i = 1, \ldots, n$$
$$b = q + \sum_{ij} P_{ij}\, \mathbf{E}(f_i f_j) + 2 \sum_{i} p_i\, \mathbf{E}(f_i).$$

# Appendix B

# Receding horizon control

Here we describe RHC; for more details see, *e.g.*, [41]. RHC works as follows. At each time $t$ we consider a time period extending $T$ steps into the future $t, t+1, \ldots, t+T$. We solve a planning problem

$$\begin{array}{ll} \text{minimize} & \sum_{\tau=t}^{t+T-1} \ell(z_\tau, v_\tau) \\ \text{subject to} & z_{\tau+1} = Az_\tau + Bv_\tau, \quad \tau = t, \ldots, t+T-1 \\ & Fv_\tau \leq h, \quad \tau = 0, \ldots, t+T-1 \\ & z_t = x_t, \end{array}$$

with variables $z_t, \ldots, z_{t+T}$, $v_t, \ldots, v_{t+T-1}$. We let $z_t^\star, \ldots, z_{t+T}^\star$, $v_t^\star, \ldots, v_{t+T-1}^\star$ denote an optimal point; the RHC policy takes $u_t = v_t^\star$. The planning process is repeated at the next time step, using new information that have become available (in this case, the true value of $x_{t+1}$).

Receding horizon control has been successfully applied to many control problems, but one disadvantage is that RHC is computationally intensive, since an optimization problem (with a relatively large number of variables) must be solved at every

time step. On the other hand, the ADP policy requires solving a relatively small optimization problem at each time step.

# Bibliography


[1] P. Abbeel, A. Coates, and A. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29, 2010.

[2] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of 21st International Conference on Machine Learning*, ICML '04. ACM, 2004.

[3] D. Ackerberg, C. Benkard, S. Berry, and A. Pakes. Econometric tools for analyzing market outcomes. In J. Heckman and E. Leamer, editors, *Handbook of Econometrics*, volume 6 of *Handbook of Econometrics*, chapter 63. Elsevier, 2007.

[4] A. Antos, R. Munos, and C. Szepesvari. Fitted Q-iteration in continuous action-space mdps. In *Advances in Neural Information Processing Systems 20*, pages 9–16, 2008.

[5] P. Bajari, C. Benkard, and J. Levin. Estimating dynamic models of imperfect competition. *Econometrica*, 75(5):1331–1370, 2007.

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

[7] R. Bellman and S. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.

[8] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, January 2002.

[9] S. Berry, J. Levinsohn, and A. Pakes. Automobile prices in market equilibrium. *Econometrica*, 63(4):841–90, July 1995.

[10] D. Bertsekas. A value iteration method for the average cost dynamic programming problem, 1995.

[11] D. Bertsekas. *Dynamic Programming and Optimal Control: Volume 1*. Athena Scientific, 2005.

[12] D. Bertsekas. *Dynamic Programming and Optimal Control: Volume 2*. Athena Scientific, 2007.

[13] D. Bertsekas and D. Castañón. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5:89–108, 1999.

[14] D. Bertsekas and S. Shreve. *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 1978.

[15] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[16] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in Systems and Control Theory*. SIAM books, Philadelphia, 1994.

[17] S. Boyd, M. Mueller, B. ODonoghue, and Y. Wang. Performance bounds and suboptimal policies for multi-period investment, 2012. Manuscript.

[18] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[19] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[20] D. Burton, W. Pulleyblank, and Ph. Toint. The inverse shortest path problem with upper bounds on shortest path costs. In *Network Optimization*, volume 450 of *Lecture notes in economics and mathematical systems*, pages 156–171. Springer, 1997.

[21] D. Burton and Ph. Toint. On an instance of the inverse shortest path problem. *Mathematical Programming*, 53:45–61, 1992.

[22] E. Chow and T. Hodgson. Rolling horizon scheduling of multi-factory supply chains. In *Proceedings of the 2003 Winter Simulation Conference*, pages 1409–1416, 2003.

[23] R. Cogill and H. Hindi. Computing policies and performance bounds for deterministic dynamic programs using mixed integer programming. In *American Control Conference (ACC)*, pages 1877–1884, 2011.

[24] D. De Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[25] D. De Farias and B. Van Roy. A cost-shaping linear program for average-cost approximate dynamic programming with performance guarantees. *Mathematics of Operations Research*, 31(3):597–620, August 2006.

[26] V. Desai, V. Farias, and C. Moallemi. Approximate dynamic programming via a smoothed linear program. *Operations Research*, 60(3):655–674, May–June 2012.

[27] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:504–556, 2005.

[28] V. Farias and B. Van Roy. An approximate dynamic programming approach to network revenue management, 2007. Manuscript.

[29] C. Garcia, D. Prett, and M. Morari. Model predictive control: Theory and practice. *Automatica*, 25(3):335–348, 1989.

[30] G. C. Goodwin, M. M. Seron, and J. A. De Doná. *Constrained Control and Estimation*. Springer, 2005.

[31] M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, 2006. Available at `http://www.stanford.edu/~boyd/cvx`.

[32] S. Graves and S. Willems. Optimizing strategic safety stock placement in supply chains. *Manufacturing and Service Operations Management*, 2(1):68–83, 2000.

[33] K. L. Judd. *Numerical Methods in Economics*. MIT Press, 1998.

[34] R. E. Kalman. When is a linear control system optimal? *Transactions of the ASME, Journal of Basic Engineering*, 86:51–60, 1964.

[35] M. Keane and K. Wolpin. The solution and estimation of discrete choice dynamic programming models by simulation and interpolation: Monte carlo evidence. *The Review of Economics and Statistics*, 76(4):648–672, 1994.

[36] M. Keane and K. Wolpin. The career decisions of young men. *Journal of Political Economy*, 105(3):473–522, 1997.

[37] A. Keshavarz and S. Boyd. Quadratic approximate dynamic programming for input-affine systems. *International Journal of Robust and Nonlinear Control*, September 2012.

[38] A. Keshavarz, Y. Wang, and S. Boyd. Imputing a convex objective function. In *Proceedings of the IEEE Multi-Conference on Systems and Control*, pages 613–619, September 2011.

[39] W. H. Kwon and S. Han. *Receding Horizon Control*. Springer-Verlag, 2005.

[40] M. Lagoudakis, R. Parr, and L. Bartlett. Least-squares policy iteration. *Journal of Machine Learning Research*, 2003.

[41] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[42] S. Mariethoz, S. Almer, M. Baja, A. G. Beccuti, D. Patino, A. Wernrud, J. Buisson, H. Cormerais, T. Geyer, H. Fujioka, U. T. Jonsson, C.-Y. Kao, M. Morari, G. Papafotiou, A. Rantzer, and P. Riedinger. Comparison of hybrid control techniques for buck and boost DC-DC converters. *IEEE Transactions on Control Systems Technology*, 18(5):1126–1145, 2010.

[43] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[44] J. Mattingley, Y. Wang, and S. Boyd. Code generation for receding horizon control. In *IEEE Multi-Conference on Systems and Control*, pages 985–992, 2010.

[45] J. E. Mattingley and S. Boyd. Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine*, 23(3):50–61, 2009.

[46] J. E. Mattingley and S. Boyd. Automatic code generation for real-time convex optimization. In D. P. Palomar and Y. C. Eldar, editors, *Convex optimization in signal processing and communications*, pages 1–41. Cambridge University Press, 2010.

[47] S. Minner. Dynamic programming algorithms for multi-stage safety stock optimization. *OR Spectrum*, 19(4):261–271, 1997.

[48] C. Moallemi, S. Kumar, and B. Van Roy. Approximate and data-driven dynamic programming for queueing networks, 2008. Manuscript.

[49] R. Munos. Performance bounds in $L_p$-norm for approximate value iteration. *SIAM Journal on Control and Optimization*, 46(2):541–561, 2007.

[50] R. Munos and C. Szepesvari. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857, 2008.

[51] G. Neu and C. Szepesvari. Apprenticeship learning using reinforcement learning and gradient methods. In *Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 295–302, 2007.

[52] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *In Proceedings of 17th International Conference on Machine Learning*, pages 663–670. Morgan Kauffman, 2000.

[53] T. Nielsen and F. Jensen. Learning a decision maker's utility function from (possibly) inconsistent behavior. *Artificial Intelligence*, 160:53–78, 2004.

[54] B. O'Donoghue, Y. Wang, and S. Boyd. Min-max approximate dynamic programming. In *Proceedings IEEE Multi-Conference on Systems and Control*, pages 424–431, September 2011.

[55] W. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley Series in Probability and Statistics, 2007.

[56] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc. New York, NY, USA, 1994.

[57] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2586–2591, 2007.

[58] N. Ratliff, J. Bagnell, and M. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[59] M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *16th European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[60] S. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, Inc. Orlando, FL, USA, 1983.

[61] B. Van Roy. *Learning and Value Function Approximation in Complex Decision Processes*. PhD thesis, Department of EECS, MIT, 1998.

[62] J. Rust. Structural estimation of markov decision processes. In R. Engle and D. McFadden, editors, *Handbook of Econometrics*, volume 4, chapter 51, pages 3081–3143. Elsevier, 1986.

[63] J. Rust. Optimal replacement of gmc bus engines: An empirical model of harold zurcher. *Econometrica*, 55(5):999–1033, September 1987.

[64] R.S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, 1998.

[65] S. R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[66] U. Syed and R. Schapire. A game-theoretic approach to apprenticeship learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1449–1456. MIT Press, Cambridge, MA, 2008.

[67] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1997.

[68] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[69] W. Wang, D. Rivera, and K. Kempf. Model predictive control strategies for supply chain management in semiconductor manufacturing. *International Journal of Production Economics*, 107(1):57–77, 2007.

[70] Y. Wang and S. Boyd. Fast model predictive control using online optimization. In *Proceedings of the 17th IFAC world congress*, pages 6974–6997, 2008.

[71] Y. Wang and S. Boyd. Performance bounds for linear stochastic control. *Systems and Control Letters*, 58(3):1780–182, March 2009.

[72] Y. Wang and S. Boyd. Approximate dynamic programming via iterated bellman inequalities, 2010. Manuscript.

[73] Y. Wang and S. Boyd. Fast evaluation of quadratic control-lyapunov policy. *IEEE Transactions on control Systems Technology*, PP(99):1–8, June 2010.

[74] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[75] J. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[76] A. Wernrud. *Approximate Dynamic Programming with Applications*. PhD thesis, Department of Automatic Control, Lund University, Sweden, 2008.

[77] P. Whittle. *Optimization over Time: Dynamic Programming and Stochastic Control*. John Wiley & Sons, Inc. New York, NY, USA, 1982.