SUBOPTIMAL CONTROL POLICIES VIA CONVEX OPTIMIZATION

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

Brendan O'Donoghue
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) Principal 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.

(Emmanuel Candés)

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.

(Sanjay Lall)

Approved for the University Committee on Graduate Studies

Abstract

In this dissertation we consider several suboptimal policies for the stochastic control problem with discounted objective and full state information. In the general case, this problem is difficult to solve exactly. However, solutions can be found for certain special cases. When the state and action spaces are finite, for example, the problem is readily solved [13]. Another such case is when the state and action spaces are finite dimensional real vector spaces, the system dynamics are linear, the cost function is convex quadratic, and there are no constraints on the action or the state. In this case optimal control policy is affine in the state variable, with coefficients that are readily computable [13, 16, 25, 98].

One general method for finding the optimal control policy is to use dynamic programming (DP). DP relies on characterizing the *value-function* of the stochastic control problem. The value function evaluated at a state is the expected cost incurred by an optimal policy starting from that state. The optimal policy can be written as an optimization problem involving the value function [13, 16, 18]. However, due to the 'curse of dimensionality', even representing the value function can be intractable when the state or action spaces are infinite, or as a practical matter, when the number of states or actions is very large. Even in cases where the value function can be represented, evaluating the optimal policy may still be intractable. In such cases a common alternative is to use approximate dynamic programming (ADP) [19,143,179]. In ADP we replace the true value function with an approximate value function in the expression for the optimal policy. The goal is to choose the approximate value function so that the performance of the resulting policy is close to optimal, or at least, good.

In the first part of this dissertation we develop two ADP control policies which

we refer to as the *min-max ADP* policy and the *iterated approximate value function (AVF)* policy respectively. Both of these policies rely on our ability to parameterize a family of lower bounds on the true value function of the stochastic control problem. The condition we use to parameterize our family of lower bounds is related to the 'linear programming approach' to ADP, which was first introduced by Manne in 1960 [118], and extended to approximate dynamic programming in [47, 159]. The basic idea is that any function which satisfies the Bellman inequality is a pointwise lower bound on the true value function [13, 16].

The min-max ADP policy uses the pointwise supremum of the family of lower bounds as a surrogate value function. Evaluating the control policy involves the solution of a min-max or saddle-point problem. For the iterated AVF policy we first develop a sequence of approximate value functions which are optimized to a trajectory of states, we then perform control at each time-step using the corresponding member of the sequence. The trajectory and approximate value functions are generated simultaneously as the solutions and dual variables to a single convex optimization problem.

For the class of problems we consider, finding the control action under either policy requires solving a convex optimization problem. For the min-max ADP policy we solve a semidefinite program (SDP) [26, 30] at each time-step. The iterated AVF policy requires solving a single SDP offline, and then solving a much smaller convex problem at each iteration.

Model predictive control (MPC) is a widespread technique for generating a suboptimal control policy that often performs very well in practice [37, 64, 76, 105, 117, 125, 162, 184]. In the second part of this dissertation we introduce a new algorithm for solving optimal control problems, of which model predictive control is a special case. The algorithm, which we refer to as *operator splitting for control* (OSC), solves MPC problems quickly and robustly. In many cases the resulting algorithm can be implemented in fixed-point arithmetic and is thus suitable for embedded applications. The algorithm relies on an operator splitting technique, referred to as the alternating direction method of multipliers (ADMM), or as Douglas-Rachford (D-R) splitting [28, 57, 62, 63, 70].

The third part of this document investigates the efficacy and computational burden of the suboptimal policies we developed in earlier sections through an in-depth multi-period portfolio optimization example [1, 55, 94, 141, 166, 191]. We exhibit a lower bound on the performance (our problem statement involves minimizing an objective) which we compare to two of the policies detailed in the previous chapters. We present timing results and demonstrate that the performance for both policies is very close to the lower bound and thus very close to optimal for several numerical instances.

Acknowledgements

In my years as a student, at Stanford and elsewhere, I have had the enormous privilege of interacting with, and learning from, many talented and generous people. It would be impossible to list everyone who has made a contribution to my life and education, but I will try to list some of the main characters.

Firstly, I must thank my advisor, Stephen Boyd. His unwavering encouragement and support have helped me navigate through the more difficult aspects of research. My first interaction with Stephen was indirect, I didn't take his class during my first quarter at Stanford. But I soon learned about him from my friends who would wax lyrical about the class (and, sometimes, complain about how hard it was too). I resolved to take the next class, offered the following quarter, just to see what all the fuss was about. After that, I was hooked - I have taken every class Stephen has offered since. Not only is Stephen a fantastic teacher and lecturer when addressing 400 students in a lecture hall, but he is also approachable and welcoming to any student with an idea or a question on a one-on-one basis. Plus, he has such a wicked sense of humour that we in his class felt as though we all shared in a set of in-jokes; we were all co-conspirators in Stephen's hilarious anecdotes and insights. Early on, I decided that Stephen would be a great fit as an advisor and a mentor. Over the following years I learned so much about how to conduct research, how to teach, and how to present my work in a clear manner, both on paper and in person. Of course, I learned a thing or two about control and optimization too...

I must also thank Emmanuel Candès who was an incredibly supportive and magnanimous mentor, especially during my last two years at Stanford. I had the enormous privilege of taking two of his classes at Stanford, which taught cutting edge material

in optimization and compressed sensing. Before each class I had a palpable sense of excitement about the marvellous new ideas Emmanuel would teach us that day. A project I presented in one of these classes became a paper that Emmanuel and I worked on together. I am very grateful for his guidance and support during our work together, and for his time and feedback on my oral defense committee.

I am also very grateful to Sanjay Lall who is an extraordinarily gifted teacher and academic. I have taken two of Sanjay's classes, one on control and one on optimization, and in both classes I was blown away by how deep and elegant the theory was, and how clear Sanjay was able to make difficult concepts. Sanjay had the most amazing insights to problems we discussed, suggesting ideas and approaches that never failed to amaze me. I am very grateful for his support, insights, and for serving on my defense committee.

I must thank the chair of my defense committee, David Miller. I took his quantum mechanics class in my first semester at Stanford and thoroughly enjoyed it. I enjoyed it so much in fact that I went back several years later to TA it for him!

Others that deserve much gratitude include my advisors and mentors at Cambridge: Julian Allwoord, David Holburn, Rob Miller, Malcolm Smith, Glenn Vinnicombe, Hugh Hunt, Chris Bishop, Jimmy Altham, and many others, and my friends at Credit Suisse: Luca, Ed, Jacky, Matt, Gina, Grace, Dino, and everyone else. Of course I must thank my fellow members of the research group located in the lair of Packard 243 and our collaborators over the years: Jacob, Yang, Ekine, Matt, Eric, Neal, Arezou, Thomas, Borja, Argyrios, Madeleine, Joseph, Ivan, George, Tom, Evan, Chung-Ching, and Alex. Thank you for the many great discussions on research, life, careers and funny things we've found on the internet.

I must thank my friends here at Stanford, Cambridge, Dublin, MIT, and elsewhere. You are too numerous to name but I must thank you all for the games, adventure, sport, arguments and laughter, not to mention the late night drinks and early morning conversation.

Finally, and most importantly, I must thank my family, my mother Maelíosa, my father Donal and my sister Niamh, plus all the extended family on both sides. If this thesis is anything it's a tribute to your unending love, support, and sacrifice.

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Overview

1.1 Outline

This document is organized into three parts. Part I gives a high-level overview of the content of this thesis. Part II describes two particular approximate dynamic programming control policies which we refer to as the *min-max ADP* policy [135] and the *iterated approximate value function* policy [139]. In Part III we describe an operator splitting algorithm for solving optimal control problems (for which model predictive control is a special case) quickly and robustly. We refer to this algorithm as *operator splitting for control* [138]. Part IV presents the particular example of finite horizon multi-period portfolio optimization [27]. We apply some of the policies discussed in previous chapters and discuss results. Finally in part V we conclude and discuss possible extensions.

The material for chapters 2-5 was taken from a series of papers. Therefore, each chapter is self-contained and can be read independently (however, as a result background material is duplicated in some chapters).

The material for part II was taken from the following two papers:

> B. O'Donoghue, Y. Wang, and S. Boyd. Min-max approximate dynamic programming. *Proceedings IEEE Multi-Conference on Systems and Control*, pages 424-431, September 2011. Available at:

`http://www.stanford.edu/~boyd/papers/min_max_ADP.html`.

B. O'Donoghue, Y. Wang, and S. Boyd. Iterated approximate value functions. Submitted to *European Control Conference*, 2012. Manuscript available at:

`http://www.stanford.edu/~boyd/papers/it_avf.html`.

The content for part III was taken from:

B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. Submitted to *IEEE Transactions on Control Systems Technology*. Manuscript available at:

`http://www.stanford.edu/~boyd/papers/oper_splt_ctrl.html`.

The content for part IV was taken from:

S. Boyd, M. Mueller, B. O'Donoghue, and Y. Wang. Performance Bounds and Suboptimal Policies for Multi-Period Investment. Submitted to *SIAM Journal on Financial Mathematics*. Manuscript available at:

`http://www.stanford.edu/~boyd/papers/port_opt_bound.html`.

1.2 Prior and related work

Each chapter in this thesis includes an accompanying section detailing the related literature relevant to that material. Here we describe some important work related to the overall theme of this dissertation.

This thesis concerns stochastic control and the approximate solution methodologies referred to as approximate dynamic programming (ADP) and model predictive control (MPC). The study of these fields goes back many years and a full treatment of the literature would be prohibitively long. Instead we direct the interested reader to some standard texts in the field. For a general overview of stochastic control see [13, 14, 16, 18]. Dynamic programming (DP) is a general technique for solving

stochastic control problems, [9, 50, 145, 153, 184], however for most problems of interest DP is intractable. However, solutions can be found for certain special cases. When the state and action spaces are finite, for example, the problem is readily solved [13]. Another such case is when the state and action spaces are finite dimensional real vector spaces, the system dynamics are linear, the cost function is convex quadratic, and there are no constraints on the action or the state. In this case optimal control policy is affine in the state variable, with coefficients that are readily computable [13, 16, 25, 98].

This thesis is primarily concerned with stochastic control problems that cannot be solved exactly and so we must resort to suboptimal policies. In this document we discuss several different policies which can be computed efficiently and perform well in practice. The first two policies we develop in part I of this dissertation are forms of approximate dynamic programming [16, 19, 47, 48, 118, 143]. These techniques rely on the parameterization of an approximate value function that is simple to represent and easy to optimize. The second policy we examine is referred to as model predictive control (which can be considered as a particular special case of approximate dynamic programming). MPC is commonly used in practice and often exhibits very good performance [37, 64, 76, 105, 117, 125, 162, 184], although it may be computationally expensive to carry out. In MPC at each iteration we solve an open loop problem with a finite horizon to generate a planned trajectory. We take the first action in this trajectory advance to the next iteration and repeat. Part II of this document develops a new algorithm to solve MPC problems quickly and robustly. The algorithm we use is a form of operator splitting, known as Douglas-Rachford splitting or the alternating direction method of multipliers (ADMM) [28, 57, 62, 63, 70].

Throughout this document we make use of recent advances in convex optimization. Typically, for the policies we discuss, the control action at each time-period will be found by solving a convex optimization problem. For a general introduction to convex optimization see, [17, 24, 30, 131]. For details on convex analysis see [88, 151]. For discussion on numerical algorithms for solving convex optimization problems see [116, 133, 134, 154, 188, 189]. For linear algebra and numerical methods see [49, 87, 172].

Part II

Approximate Dynamic Programming Policies

Chapter 2

Min-Max Approximate Dynamic Programming

In this chapter we describe an approximate dynamic programming policy for a discrete-time dynamical system perturbed by noise. The approximate value function is the pointwise supremum of a family of lower bounds on the value function of the stochastic control problem; evaluating the control policy involves the solution of a min-max or saddle-point problem. For a quadratically constrained linear quadratic control problem, evaluating the policy amounts to solving a semidefinite program at each time step. By evaluating the policy, we obtain a lower bound on the value function, which can be used to evaluate performance: When the lower bound and the achieved performance of the policy are close, we can conclude that the policy is nearly optimal. We describe several numerical examples where this is indeed the case.

2.1 Introduction

We consider an infinite horizon stochastic control problem with discounted objective and full state information. In the general case this problem is difficult to solve, but exact solutions can be found for certain special cases. When the state and action spaces are finite, for example, the problem is readily solved. Another case for which the problem can be solved exactly is when the state and action spaces are finite

dimensional real vector spaces, the system dynamics are linear, the cost function is convex quadratic, and there are no constraints on the action or the state. In this case optimal control policy is affine in the state variable, with coefficients that are readily computable [13, 16, 25, 98].

One general method for finding the optimal policy is to use dynamic programming (DP). DP represents the optimal policy in terms of an optimization problem involving the value function of the stochastic control problem [13, 16, 18]. However, due to the 'curse of dimensionality', even representing the value function can be intractable when the state or action spaces are infinite, or as a practical matter, when the number of states or actions is very large. Even when the value function can be represented, evaluating the optimal policy can still be intractable. As a result approximate dynamic programming (ADP) has been developed as a general method for finding suboptimal control policies [19, 143, 179]. In ADP we substitute an approximate value function for the value function in the expression for the optimal policy. The goal is to choose the approximate value function (also known as a control-Lyapunov function) so that the performance of the resulting policy is close to optimal, or at least, good.

In this chapter we develop a control policy which we call the *min-max approximate dynamic programming policy*. We first parameterize a family of lower bounds on the true value function; then we perform control, taking the pointwise supremum over this family as our approximate value function. The condition we use to parameterize our family of bounds is related to the 'linear programming approach' to ADP, which was first introduced in [118], and extended to approximate dynamic programming in [47, 159]. The basic idea is that any function which satisfies the Bellman inequality is a lower bound on the true value function [13, 16].

It was shown in [180] that a better lower bound can be attained via an iterated chain of Bellman inequalities, which we use here. We relate this chain of inequalities to a forward look-ahead in time, in a similar sense to that of model predictive control (MPC) [64, 117]. Indeed many types of MPC can be thought of as performing min-max ADP with a particular (generally affine) family of underestimator functions.

In cases where we have a finite number of states and inputs, evaluating our policy requires solving a linear program at every time step. For problems with an infinite

number of states and inputs, the method requires the solution of a semi-infinite linear program, with a finite number of variables, but an infinite number of constraints (one for every state-control pair). For these problems we can obtain a tractable semidefinite program (SDP) approximation using methods such as the $\mathcal{S}$-procedure [179, 180]. Evaluating our policy then requires solving an SDP at each time step [26, 30].

Much progress has been made in solving structured convex programs efficiently (see, *e.g.*, [122, 123, 181, 182]). These fast optimization methods make our policies practical, even for large problems, or those requiring fast sampling rates.

2.2 Stochastic control

Consider a discrete time-invariant dynamical system, with dynamics described by

$$x_{t+1} = f(x_t, u_t, w_t), \quad t = 0, 1, \ldots, \tag{2.1}$$

where $x_t \in \mathcal{X}$ is the system state, $u_t \in \mathcal{U}$ is the control input or action, $w_t \in \mathcal{W}$ is an exogenous noise or disturbance, at time t, and $f : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathcal{X}$ is the state transition function. The noise terms w_t are independent identically distributed (IID), with known distribution. The initial state x_0 is also random with known distribution, and is independent of w_t. We consider causal, time-invariant state feedback control policies

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

where $\phi : \mathcal{X} \to \mathcal{U}$ is the *control policy* or *state feedback function*.

The stage cost is given by $\ell : \mathcal{X} \times \mathcal{U} \to \mathbf{R} \cup \{+\infty\}$, where the infinite values of ℓ encode constraints on the states and inputs: The state-action *constraint set* $\mathcal{C} \subset \mathcal{X} \times \mathcal{U}$ is $\mathcal{C} = \{(z, v) \mid \ell(z, v) < \infty\}$. (The problem is unconstrained if $\mathcal{C} = \mathcal{X} \times \mathcal{U}$.)

The *stochastic control problem* is to choose ϕ in order to minimize the infinite horizon discounted cost

$$J_\phi = \mathbf{E} \sum_{t=0}^{\infty} \gamma^t \ell(x_t, \phi(x_t)), \tag{2.2}$$

where $\gamma \in (0, 1)$ is a discount factor. The expectations are over the noise terms w_t,

$t = 0, 1, \ldots$, and the initial state x_0. We assume here that the expectation and limits exist, which is the case under various technical assumptions [13, 16]. We denote by $J^\star$ the optimal value of the stochastic control problem, *i.e.*, the infimum of J_ϕ over all policies $\phi : \mathcal{X} \to \mathcal{U}$.

2.2.1 Dynamic programming

In this section we briefly review the dynamic programming characterization of the solution to the stochastic control problem. For more details, see [13, 16].

The *value function* of the stochastic control problem, $V^\star : \mathcal{X} \to \mathbf{R} \cup \{\infty\}$, is given by

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

subject to the dynamics (2.1) and $x_0 = z$; the infimum is over all policies $\phi : \mathcal{X} \to \mathcal{U}$, and the expectation is over w_t for $t = 0, 1, \ldots$. The quantity $V^\star(z)$ is the cost incurred by an optimal policy, when the system is started from state z. The optimal total discounted cost is given by

$$J^\star = \mathop{\mathbf{E}}_{x_0} V^\star(x_0),$$

where $x_0 \in \mathcal{X}$ is the initial state. It can be shown that the value function is the unique fixed point of the Bellman equation

$$V^\star(z) = \inf_v \left(\ell(z, v) + \gamma \mathop{\mathbf{E}}_w V^\star(f(z, v, w)) \right)$$

for all $z \in \mathcal{X}$. We can write the Bellman equation in the form

$$V^\star = \mathcal{T} V^\star, \tag{2.3}$$

where we define the Bellman operator $\mathcal{T}$ as

$$(\mathcal{T} g)(z) = \inf_v \left(\ell(z, v) + \gamma \mathop{\mathbf{E}}_w g(f(z, v, w)) \right)$$

for any $g : \mathcal{X} \to \mathbf{R} \cup \{+\infty\}$.

An optimal policy for the stochastic control problem is given by

$$\phi^\star(z) = \operatorname*{argmin}_v \left(\ell(z, v) + \gamma \mathop{\mathbf{E}}_w V^\star(f(z, v, w)) \right), \tag{2.4}$$

for all $z \in \mathcal{X}$.

2.2.2 Approximate dynamic programming

In many cases of interest, it is intractable to compute (or even represent) the value function $V^\star$, let alone carry out the minimization required evaluate the optimal policy (2.4). In such cases, a common alternative is to replace the value function with an *approximate value function* $\hat{V}$ [19, 143, 179]. The resulting policy, given by

$$\hat{\phi}(z) = \operatorname*{argmin}_v \left(\ell(z, v) + \gamma \mathop{\mathbf{E}}_w \hat{V}(f(z, v, w)) \right),$$

for all $z \in \mathcal{X}$, is called an *approximate dynamic programming* (ADP) policy. Clearly, when $\hat{V} = V^\star$, the ADP policy is optimal. The goal of approximate dynamic programming is to find a $\hat{V}$ for which the ADP policy can be easily evaluated (for instance, by solving a convex optimization problem), and also attains near-optimal performance.

2.3 Min-max approximate dynamic programming

We consider a family of linearly parameterized (candidate) value functions $V_\alpha : \mathcal{X} \to \mathbf{R}$,

$$V_\alpha = \sum_{i=1}^{K} \alpha_i V^{(i)}, \tag{2.5}$$

where $\alpha \in \mathbf{R}^K$ is a vector of coefficients and $V^{(i)} : \mathcal{X} \to \mathbf{R}$ are fixed basis functions. Now suppose we have a set $\mathcal{A} \subset \mathbf{R}^K$ for which

$$V_\alpha(z) \leq V^\star(z), \quad \forall z \in \mathcal{X}, \quad \forall \alpha \in \mathcal{A}.$$

Thus $\{V_\alpha \mid \alpha \in \mathcal{A}\}$ is a parameterized family of underestimators of the value function. (We will discuss how to obtain such a family later.) For any $\alpha \in \mathcal{A}$ we have

$$V_\alpha(z) \leq \sup_{\alpha \in \mathcal{A}} V_\alpha(z) \leq V^\star(z), \quad \forall z \in \mathcal{X},$$

i.e., the pointwise supremum over the family of underestimators must be at least as good an approximation of $V^\star$ as any single function from the family. This suggests the ADP control policy

$$\tilde{\phi}(z) = \operatorname*{argmin}_v \left(\ell(z, v) + \gamma \operatorname*{\mathbf{E}}_w \sup_{\alpha \in \mathcal{A}} V_\alpha(f(z, v, w)) \right),$$

where we use $\sup_{\alpha \in \mathcal{A}} V_\alpha(z)$ as an approximate value function. Unfortunately, this policy may be difficult to evaluate, since evaluating the expectation of the supremum can be hard, even when evaluating $\mathbf{E}\, V_\alpha(f(z, v, w))$ for a particular α can be done.

Our last step is to exchange expectation and supremum to obtain the *min-max control policy*

$$\begin{aligned} \phi^{\mathrm{mm}}(z) = \operatorname{argmin}_v \sup_{\alpha \in \mathcal{A}} (&\ell(z, v) \\ &+ \gamma\, \mathbf{E}_w\, V_\alpha(f(z, v, w))) \end{aligned} \tag{2.6}$$

for all $z \in \mathcal{X}$. Computing this policy involves the solution of a min-max or saddle-point problem, which we will see is tractable in certain cases. One such case is where the function $\ell(z, v) + \mathbf{E}_w\, V_\alpha(f(z, v, w))$ is convex in v for each z and α and the set $\mathcal{A}$ is convex.

2.3.1 Bounds

The optimal value of the optimization problem in the min-max policy (2.6) is a lower bound on the value function at every state. To see this we note that

$$\begin{aligned}
&\inf_v \sup_{\alpha\in\mathcal{A}} \Big(\ell(z,v) + \gamma\, \mathbf{E}\, V_\alpha(f(z,u,w))\Big) \\
&\leq \inf_v \Big(\ell(z,v) + \gamma\, \mathbf{E} \sup_{\alpha\in\mathcal{A}} V_\alpha(f(z,u,w))\Big) \\
&\leq \inf_v \Big(\ell(z,v) + \gamma\, \mathbf{E}\, V^\star(f(z,u,w))\Big) \\
&= (\mathcal{T}V^\star)(z) = V^\star(z),
\end{aligned}$$

where the first inequality is due to Fatou's lemma [38], the second inequality follows from the monotonicity of expectation, and the equality comes from the fact that $V^\star$ is the unique fixed point of the Bellman operator.

Using the pointwise bounds, we can evaluate a lower bound on the optimal cost $J^\star$ via Monte Carlo simulation:

$$J^{\mathrm{lb}} = (1/N)\sum_{j=1}^{N} V^{\mathrm{lb}}(z_j)$$

where $z_1, \ldots, z_N$ are drawn from the same distribution as x_0 and $V^{\mathrm{lb}}(z_j)$ is the lower bound we get from evaluating the min-max policy at z_j.

The performance of the min-max policy can also be evaluated using Monte Carlo simulation, and provides an upper bound on the optimal cost, which we shall denote by J^{ub}. Ignoring Monte Carlo error we have

$$J^{\mathrm{lb}} \leq J^\star \leq J^{\mathrm{ub}}.$$

Theses upper and lower bounds on the optimal value of the stochastic control problem are readily evaluated numerically, through simulation of the min-max control policy. When J^{lb} and J^{ub} are close, we can conclude that the min-max policy is almost optimal. We will use this technique to evaluate the performance of the min-max policy for our numerical examples.

2.3.2 Evaluating the min-max control policy

Evaluating the min-max control policy often requires exchanging the order of minimization and maximization. For any function $f : \mathbf{R}^p \times \mathbf{R}^q \to \mathbf{R}$ and sets $\mathcal{W} \subseteq \mathbf{R}^p$, $\mathcal{Z} \subseteq \mathbf{R}^q$, the *max-min inequality* states that

$$\sup_{z\in\mathcal{Z}} \inf_{w\in\mathcal{W}} f(w,z) \leq \inf_{w\in\mathcal{W}} \sup_{z\in\mathcal{Z}} f(w,z). \tag{2.7}$$

In the context of the min-max control policy, this means we can swap the order of minimization and maximization in (2.6) and maintain the lower bound property. To evaluate the policy, we solve the optimization problem

$$\begin{array}{ll} \text{maximize} & \inf_v \left(\ell(z,v) + \gamma\, \mathbf{E}_w\, V_\alpha(f(z,v,w))\right) \\ \text{subject to} & \alpha \in \mathcal{A} \end{array} \tag{2.8}$$

with variable α. If $\mathcal{A}$ is a convex set, (2.8) is a convex optimization problem, since the objective is the infimum over a family of affine functions in α, and is therefore concave. In practice, solving (2.8) is often much easier than evaluating the min-max control policy directly.

In addition, if there exist $\tilde{w} \in \mathcal{W}$ and $\tilde{z} \in \mathcal{Z}$ such that

$$f(\tilde{w}, z) \leq f(\tilde{w}, \tilde{z}) \leq f(w, \tilde{z}),$$

for all $w \in \mathcal{W}$ and $z \in \mathcal{Z}$, then we have the *strong max-min property* (or *saddle-point property*) and (2.7) holds with equality. In such cases the problems (2.6) and (2.8) are equivalent, and we can use Newton's method or duality considerations to solve (2.6) or (2.8) [12, 30].

2.4 Iterated Bellman inequalities

In this section we describe how to parameterize a family of underestimators of the true value function. The idea is based on the Bellman inequality, [47, 179, 180], and results in a convex condition on the coefficients α that guarantees $V_\alpha \leq V^\star$.

2.4.1 Basic Bellman inequality

The basic condition works as follows. Suppose we have a function $V : \mathcal{X} \to \mathbf{R}$, which satisfies the Bellman inequality

$$V \leq \mathcal{T}V. \tag{2.9}$$

Then by the monotonicity of the Bellman operator, we have

$$V \leq \lim_{k\to\infty} \mathcal{T}^k V = V^\star,$$

so any function that satisfies the Bellman inequality must be a value function underestimator. Applying this condition to V_α and expanding (2.9) we get

$$V_\alpha(z) \leq \inf_v \left(\ell(z,v) + \gamma \mathop{\mathbf{E}}_w V_\alpha(f(z,v,w))\right),$$

for all $z \in \mathcal{X}$. For each z, the left-hand side is linear in α since it is of the form given in eq. (2.5), and the right-hand side is a concave function of α, since it is the infimum over a family of affine functions. Hence, the Bellman inequality leads to a convex constraint on α.

2.4.2 Iterated Bellman inequalities

We can obtain better (*i.e.*, larger) lower bounds on the value function by considering an iterated form of the Bellman inequality [180]. Suppose we have a sequence of functions $V_i : \mathcal{X} \to \mathbf{R}$, $i = 0, \ldots, M$, that satisfy a chain of Bellman inequalities

$$V_0 \leq \mathcal{T}V_1, \quad V_1 \leq \mathcal{T}V_2, \quad \ldots \quad V_{M-1} \leq \mathcal{T}V_M, \tag{2.10}$$

with $V_{M-1} = V_M$. Then, using similar arguments as before we can show $V_0 \leq V^\star$. Restricting each function to lie in the same subspace

$$V_i = \sum_{j=1}^{K} \alpha_{ij} V^{(j)},$$

we see that the iterated chain of Bellman inequalities also results in a convex constraint on the coefficients α_{ij}. Hence the condition on α_{0j}, $j = 1, \ldots, K$, which parameterizes our underestimator V_0, is convex. It is easy to see that the iterated Bellman condition must give bounds that are at least as good as the basic Bellman inequality, since any function that satisfies (2.9) must be feasible for (2.10) [180].

2.5 Examples

2.5.1 Box constrained linear quadratic control

This section follows a similar example presented in [180]. We have $\mathcal{X} = \mathbf{R}^n, \mathcal{U} = \mathbf{R}^m$, with linear dynamics

$$x_{t+1} = Ax_t + Bu_t + w_t,$$

where $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{n\times m}$. The noise has zero mean, $\mathbf{E}\, w_t = 0$, and covariance $\mathbf{E}\, w_t w_t^T = W$. (Our bounds and policy will only depend on the first and second moments of w_t.) The stage cost is given by

$$\ell(z, v) = \begin{cases} v^T R v + z^T Q z, & \|v\|_\infty \leq 1 \\ +\infty, & \|v\|_\infty > 1, \end{cases}$$

where $R = R^T \succeq 0$, $Q = Q^T \succeq 0$.

Iterated Bellman inequalities

We look for convex quadratic approximate value functions

$$V_i(z) = z^T P_i z + 2p_i^T z + s_i, \quad i = 0, \ldots, M,$$

where $P_i = P_i^T \succeq 0$, $p_i \in \mathbf{R}^n$ and $r_i \in \mathbf{R}$, are the coefficients of our linear parameterization. The iterated Bellman inequalities are

$$V_{i-1}(z) \leq \ell(z, v) + \gamma\, \mathbf{E}\, V_i(Az + Bv + w),$$

for all $\|v\|_\infty \leq 1$, $z \in \mathbf{R}^n$, $i = 1, \ldots, M$. Defining

$$S_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & P_i & p_i \\ 0 & p_i^T & s_i \end{bmatrix}, \quad L = \begin{bmatrix} R & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$G_i = \begin{bmatrix} B^T P_i B & B^T P_i A & B^T p_i \\ A^T P_i B & A^T P_i A & A^T p_i \\ p_i^T B & p_i^T A & \mathbf{Tr}(P_i W) + s_i \end{bmatrix},$$

for $i = 0, \ldots, M$, we can write the Bellman inequalities as a quadratic form in $(v, z, 1)$

$$\begin{bmatrix} v \\ z \\ 1 \end{bmatrix}^T (L + \gamma G_i - S_{i-1}) \begin{bmatrix} v \\ z \\ 1 \end{bmatrix} \geq 0, \tag{2.11}$$

for all $\|v\|_\infty \leq 1$, $z \in \mathbf{R}^n$, $i = 1, \ldots, M$.

We will obtain a tractable sufficient condition for this using the $\mathcal{S}$-procedure [26, 180]. The constraint $\|v\|_\infty \leq 1$ can be written in terms of quadratic inequalities

$$1 - v^T (e_i e_i^T) v \geq 0, \quad i = 1, \ldots, m,$$

where e_i denotes the ith unit vector. Using the $\mathcal{S}$-procedure, a sufficient condition for (2.11) is the existence of diagonal matrices $D_i \succeq 0$, $i = 1, \ldots, M$ for which

$$L + \gamma G_i - S_{i-1} + \Lambda_i \succeq 0, \quad i = 1, \ldots M, \tag{2.12}$$

where

$$\Lambda_i = \begin{bmatrix} D_i & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\mathbf{Tr}(D_i) \end{bmatrix}. \tag{2.13}$$

Finally we have the terminal constraint, $S_{M-1} = S_M$.

Min-max control policy

For this problem, it is easy to show that the strong max-min property holds, and therefore problems (2.6) and (2.8) are equivalent. To evaluate the min-max control policy we solve problem (2.8), which we can write as

$$\begin{array}{ll} \text{maximize} & \inf_v(\ell(z,v) + \gamma\, \mathbf{E}_w\, V_0(Az + Bv + w)) \\ \text{subject to} & (2.12), \quad S_{M-1} = S_M, \quad P_0 \succeq 0 \\ & P_i \succeq 0, \quad D_i \succeq 0, \quad i = 1, \ldots, M, \end{array}$$

with variables P_i, p_i, s_i, $i = 0, \ldots, M$, and diagonal D_i, $i = 1, \ldots, M$. We will convert this max-min problem to a max-max problem by forming the dual of the minimization part. Introducing a diagonal matrix $D_0 \succeq 0$ as the dual variable for the box constraints, we obtain the dual function

$$\inf_v \begin{bmatrix} v \\ z \\ 1 \end{bmatrix}^T (L + \gamma G_0 - \Lambda_0) \begin{bmatrix} v \\ z \\ 1 \end{bmatrix},$$

where Λ_0 has the form given in (2.13). We can minimize over v analytically. If we block out the matrix $L + \gamma G_0 - \Lambda_0$ as

$$(L + \gamma G_0 - \Lambda_0) = \begin{bmatrix} M_{11} & M_{12} \\ M_{12}^T & M_{22} \end{bmatrix} \tag{2.14}$$

where $M_{11} \in \mathbf{R}^{m \times m}$, then

$$v^\star = -M_{11}^{-1} M_{12} \begin{bmatrix} z \\ 1 \end{bmatrix}.$$

Thus our problem becomes

$$\begin{array}{ll} \text{maximize} & \left[\begin{array}{c} z \\ 1 \end{array}\right] (M_{22} - M_{12}^T M_{11}^{-1} M_{12}) \left[\begin{array}{c} z \\ 1 \end{array}\right] \\ \text{subject to} & (2.12), \quad S_{M-1} = S_M \\ & P_i \succeq 0, \quad D_i \succeq 0, \quad i = 0, \ldots, M, \end{array}$$

which is a convex optimization problem in the variables P_i, p_i, r_i, D_i, $i = 0, \ldots, M$, and can be solved as an SDP.

To implement the min-max control policy, at each time t, we solve the above problem with $z = x_t$, and let

$$u_t = -M_{11}^{\star -1} M_{12}^{\star} \left[\begin{array}{c} x_t \\ 1 \end{array}\right],$$

where $M_{11}^{\star}$ and $M_{12}^{\star}$ denote the matrices M_{11} and M_{12}, computed from $P_0^{\star}$, $p_0^{\star}$, $s_0^{\star}$, $D_0^{\star}$.

Interpretations

We can easily verify that the dual of the above optimization problem is a variant of model predictive control, that uses both the first and second moments of the state. In this context, the number of iterations, M, is the length of the prediction horizon, and we can interpret our lower bound as a finite horizon approximation to an infinite horizon problem, which underestimates the optimal infinite horizon cost. The $\mathcal{S}$-procedure relaxation also has a natural interpretation: in [66], the author obtains similar LMIs by relaxing almost sure constraints into constraints that are only required to hold in expectation.

Numerical instance

We consider a numerical example with $n = 8$, $m = 3$, and $\gamma = 0.9$. The parameters Q, R, A and B are randomly generated; we set $\|B\| = 1$ and scale A so that $\max |\lambda_i(A)| = 1$ (*i.e.*, so that the system is marginally stable). The initial state x_0 is Gaussian, with zero mean.

Policy / Bound	**Value**
MPC policy	1.3147
Min-max policy	1.3145
Lower bound	1.3017

Table 2.1: Performance comparison, box constrained example.

Table 2.1 shows the performance of the min-max policy and certainty equivalent MPC, both with horizons of $M = 15$ steps, as well as the lower bound on the optimal cost. In this case, both the min-max policy and MPC are no more than around 1% suboptimal, modulo Monte Carlo error.

2.5.2 Dynamic portfolio optimization

In this example, we manage a portfolio of n assets over time. Our state $x_t \in \mathbf{R}^n$ is the vector of dollar values of the assets, at the beginning of investment period t. Our action $u_t \in \mathbf{R}^n$ represents buying or selling assets at the beginning of each investment period: $(u_t)_i > 0$ means we are buying asset i, for dollar value $(u_t)_i$, $(u_t)_i < 0$ means we sell asset i. The post-trade portfolio is then given by $x_t + u_t$, and the total gross cash put in is $\mathbf{1}^T u_t$, where $\mathbf{1}$ is the vector with all components one.

The portfolio propagates over time (*i.e.*, over the investment period) according to

$$x_{t+1} = A_t(x_t + u_t)$$

where $A_t = \mathbf{diag}(\rho_t)$, and $(\rho_t)_i$ is the (total) return of asset i in investment period t. The return vectors ρ_t are IID, with first and second moments

$$\mathbf{E}(\rho_t) = \mu, \qquad \mathbf{E}(\rho_t \rho_t^T) = \Sigma.$$

(Here too, our bounds and policy will only depend on the first and second moments of ρ_t.) We let $\hat{\Sigma} = \Sigma - \mu\mu^T$ denote the return covariance.

We now describe the constraints and objective. We constrain the risk of our post-trade portfolio, which we quantify as the portfolio return variance over the period:

$$(x_t + u_t)^T \hat{\Sigma} (x_t + u_t) \leq l,$$

where $l \geq 0$ is the maximum variance (risk) allowed. Our action (buying and selling) u_t incurs a transaction cost with an absolute value and a quadratic component. This cost is given by $\kappa^T |u| + u^T R u$, where $\kappa \in \mathbf{R}^n_+$ is the vector of linear transaction cost rates (and $|u|$ is taken to mean elementwise), and $R \in \mathbf{R}^{n \times n}$, which is diagonal with positive entries, contains the quadratic transaction cost coefficients. Linear transactions costs are used to model effects such as crossing the bid-ask spread or brokerage fees, while quadratic transaction costs can model the effect of price-impact. Thus at time t, we put into our portfolio the net cash amount

$$g(u_t) = \mathbf{1}^T u_t + \kappa^T |u_t| + u_t^T R u_t.$$

(When this is negative, it represents revenue.) The first term is the gross cash in from purchases and sales; the second and third terms are the transaction fees. The stage cost, including the risk limit, is

$$\ell(z, v) = \begin{cases} g(v), & (z+v)^T \hat{\Sigma} (z+v) \leq l \\ +\infty, & \text{otherwise.} \end{cases}$$

Our goal is to minimize the discounted cost (or equivalently, to maximize the discounted revenue). In this example, the discount factor has a natural interpretation as reflecting the time value of money.

Iterated Bellman Inequalities

We incorporate another variable, $y \in \mathbf{R}^n$, to remove the absolute value term from the stage cost function, and add the constraints

$$-(y_t)_i \leq (v_t)_i \leq (y_t)_i, \quad i = 1, \ldots, n. \tag{2.15}$$

We define the stage cost with these new variables to be

$$\ell(z,v,y) = \mathbf{1}^T v + \kappa^T y + (1/2)v^T R v + (1/2)y^T R y$$

for (z,v,y) that satisfy

$$-y \leq v \leq y, \quad (z+v)\hat{\Sigma}(z+v) \leq l, \tag{2.16}$$

and $+\infty$ otherwise. Here, the first set of inequalities is interpreted elementwise.

We look for convex quadratic candidate value functions, *i.e.*

$$V_i(z) = z^T P_i z + 2p_i^T z + s_i, \quad i = 0, \ldots, M,$$

where $P_i \succeq 0$, $p_i \in \mathbf{R}^n$, $s_i \in \mathbf{R}$ are the coefficients of our linear parameterization. Defining

$$L = \begin{bmatrix} R/2 & 0 & 0 & \mathbf{1}/2 \\ 0 & R/2 & 0 & \kappa/2 \\ 0 & 0 & 0 & 0 \\ \mathbf{1}^T/2 & \kappa^T/2 & 0 & 0 \end{bmatrix}, \quad G_i = \begin{bmatrix} P_i \circ \Sigma & 0 & P_i \circ \Sigma & p_i \circ \mu \\ 0 & 0 & 0 & 0 \\ P_i \circ \Sigma & 0 & P_i \circ \Sigma & p_i \circ \mu \\ (p_i \circ \mu)^T & 0 & (p_i \circ \mu)^T & 0 \end{bmatrix},$$

$$S_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & P_i & p_i \\ 0 & 0 & p_i^T & s_i \end{bmatrix}, \quad i = 0, \ldots, M,$$

where $\circ$ denotes the Hadamard product, we can write the iterated Bellman inequalities

as

$$\begin{bmatrix} v \\ y \\ z \\ 1 \end{bmatrix}^T (L + \gamma G_i - S_{i-1}) \begin{bmatrix} v \\ y \\ z \\ 1 \end{bmatrix} \geq 0,$$

for all (v, y, z) that satisfy (2.16), for $i = 1, \ldots, M$.

A tractable sufficient condition for the Bellman inequalities is (by the $\mathcal{S}$-procedure) the existence of $\lambda_i \geq 0$, $\nu_i \in \mathbf{R}^n_+$, $\tau_i \in \mathbf{R}^n_+$, $i = 1, \ldots, M$ such that

$$L + \gamma G_i - S_{i-1} + \Lambda_i \succeq 0, \quad i = 1, \ldots, M, \tag{2.17}$$

where

$$\Lambda_i = \begin{bmatrix} \lambda_i \hat{\Sigma} & 0 & \lambda_i \hat{\Sigma} & \nu_i - \tau_i \\ 0 & 0 & 0 & \nu_i + \tau_i \\ \lambda_i \hat{\Sigma} & 0 & \lambda_i \hat{\Sigma} & 0 \\ \nu_i^T - \tau_i^T & \nu_i^T + \tau_i^T & 0 & -\lambda_i l \end{bmatrix}. \tag{2.18}$$

Lastly we have the terminal constraint, $S_{M-1} = S_M$.

Min-max control policy

The discussion here follows almost exactly the one presented for the previous example. It is easy to show that in this case we have the strong max-min property. At each step, we solve problem (2.8) by converting the max-min problem into a max-max problem, using Lagrangian duality. We can write problem (2.8) as

$$\begin{array}{ll} \text{maximize} & \inf_{v,y}(\ell(z, v, y) + \gamma \, \mathbf{E}_\rho \, V_0(A(z + v))) \\ \text{subject to} & (2.17), \quad S_{M-1} = S_M \\ & P_i \succeq 0, \quad i = 0, \ldots, M, \end{array}$$

with variables P_i, p_i, s_i, $i = 0, \ldots, M$, and $\lambda_i \in \mathbf{R}_+$, $\nu_i \in \mathbf{R}^n_+$, $\tau_i \in \mathbf{R}^n_+$, $i = 1, \ldots, M$.

Next, we derive the dual function of the minimization part. We introduce variables $\lambda_0 \in \mathbf{R}_+$, $\nu_0 \in \mathbf{R}^n_+$, $\tau_0 \in \mathbf{R}^n_+$, which are dual variables corresponding to the constraints (2.16) and (2.15). The dual function is given by

$$\inf_{v,y} \begin{bmatrix} v \\ y \\ z \\ 1 \end{bmatrix}^T (L + \gamma G_0 + \Lambda_0) \begin{bmatrix} v \\ y \\ z \\ 1 \end{bmatrix},$$

where Λ_0 has the form given in (2.18). If we define

$$L + \gamma G_0 + \Lambda_0 = \begin{bmatrix} M_{11} & M_{12} \\ M_{12}^T & M_{22} \end{bmatrix}, \tag{2.19}$$

where $M_{11} \in \mathbf{R}^{2n \times 2n}$, then the minimizer of the Lagrangian is

$$(v^\star, y^\star) = -M_{11}^{-1} M_{12} \begin{bmatrix} z \\ 1 \end{bmatrix}.$$

Thus our problem becomes

$$\begin{array}{lll} \text{maximize} & \begin{bmatrix} z \\ 1 \end{bmatrix} (M_{22} - M_{12}^T M_{11}^{-1} M_{12}) \begin{bmatrix} z \\ 1 \end{bmatrix} & \\ \text{subject to} & (2.17), \quad S_{M-1} = S_M & \\ & P_i \succeq 0, \quad i = 0, \ldots, M, & \end{array}$$

which is convex in the variables P_i, p_i, s_i, $i = 0, \ldots, M$, and $\lambda_i \in \mathbf{R}_+$, $\nu_i \in \mathbf{R}^n_+$, $\tau_i \in \mathbf{R}^n_+$, $i = 0, \ldots, M$.

To implement the policy, at each time t we solve the above optimization problem

(as an SDP) with $z = x_t$, and let

$$u_t = -\left[\begin{array}{cc} I_m & 0 \end{array}\right] M_{11}^{\star -1} M_{12}^{\star} \left[\begin{array}{c} x_t \\ 1 \end{array}\right],$$

where $M_{11}^{\star}$ and $M_{12}^{\star}$ denote the matrices M_{11} and M_{12}, computed from $P_0^{\star}$, $p_0^{\star}$, $s_0^{\star}$, $\lambda_0^{\star}$, $\nu_0^{\star}$, $\tau_0^{\star}$.

Numerical instance

We consider a numerical example with $n = 8$ assets and $\gamma = 0.96$. The initial portfolio x_0 is Gaussian, with zero mean. The returns follow a log-normal distribution, *i.e.*, $\log(\rho_t) \sim \mathcal{N}(\tilde{\mu}, \tilde{\Sigma})$. The parameters μ and Σ are given by

$$\mu_i = \exp(\tilde{\mu}_i + \tilde{\Sigma}_{ii}/2), \quad \Sigma_{ij} = \mu_i \mu_j \exp(\tilde{\Sigma}_{ij}).$$

Table 2.2 compares the performance of the min-max policy for $M = 40$ and $M = 5$, and certainty equivalent model predictive control with horizon $T = 40$, over 1150 simulations each consisting of 150 time steps. We can see that the min-max policy significantly outperforms MPC, which actually makes a loss on average (since the average cost is positive). The cost achieved by the min-max policy is close to the lower bound, which shows that both the policy and the bound are nearly optimal. In fact, the gap is small even for $M = 5$, which corresponds to a relatively myopic policy.

Bound / Policy	**Value**
MPC policy, $T = 40$	25.4
Min-max policy, $M = 5$	-224.1
Min-max policy, $M = 40$	-225.1
Lower bound, $M = 40$	-239.9
Lower bound, $M = 5$	-242.0

Table 2.2: Performance comparison, portfolio example.

2.6 Summary

In this chapter we introduced a control policy which we referred to as the *min-max approximate dynamic programming* policy. Evaluating this policy at each time step requires the solution of a min-max or saddle point problem. In addition, our technique yields a pointwise lower bound on the true value function of the stochastic control problem, which can used to bound (via Monte Carlo simulation) the performance of the optimal policy.

We demonstrated the method with two examples where the policy can be evaluated by solving a convex optimization problem at each time-step. In both examples the lower bound and the achieved performance are very close, certifying that the min-max ADP policy is close to optimal.

Chapter 3

Iterated Approximate Value Functions

In this chapter we introduce a control policy which we refer to as the iterated approximate value function policy. The generation of this policy requires two steps. In the first step we simultaneously compute a trajectory of moments of the state and action and a sequence of approximate value functions optimized to that trajectory. This pre-computation is performed off-line. The next step is to perform control using the generated sequence of approximate value functions. This amounts to a time-varying policy, even in the case where the optimal policy is time-invariant.

We restrict our attention to the case with linear dynamics and quadratically representable stage cost function. In this scenario the pre-computation stage requires the solution of a semidefinite program (SDP) which is tractable. Finding the control action at each time-period requires solving a small convex optimization problem which can be carried out quickly. We conclude with some examples.

3.1 Introduction

We consider an infinite horizon discounted stochastic control problem with full state information. In general this problem is difficult to solve exactly, although there are some special cases in which is is tractable. When the state and action space are

finite and not too large, it is readily solved. Another example is the case of linear dynamics and convex quadratic stage cost, possibly with linear constraints. In this case the optimal action is affine in the state variable, and the coefficients are readily computed.

A general method for solving stochastic control problems is Dynamic Programming (DP). DP relies on characterizing the *value-function* of the stochastic control problem. The value function captures the expected cost incurred by an optimal policy starting from any state. The optimal policy can then be written as an optimization problem involving the value function. However, in most cases the value function is hard to represent, let alone compute. Even in cases where this is not the case, it may be hard to solve the associated optimization problem to obtain the policy.

In such cases a common alternative is to use Approximate Dynamic Programming (ADP). In ADP the value function is replaced with a surrogate function, often referred to as an approximate value function (AVF). The approximate policy is found by solving the policy optimization problem, where we replace the true value function with the surrogate. The goal, then, in ADP is to choose an approximate value function so that the problem of computing the approximate policy is tractable and the approximate policy performs well. In this chapter, we introduce an approximate dynamic programming policy, in which, however, the surrogate function we use to find the control action changes at each time-step.

The policy we develop in this chapter relies on parameterizing a family of lower bounds on the true value function. The condition we use to guarantee lower-boundedness is referred to as the iterated Bellman inequality [180], and is related to the 'linear programming approach' to ADP [47, 118, 159]. In a series of recent papers Wang and Boyd demonstrated a technique for generating performance bounds for a class of stochastic control problems using the iterated Bellman inequality. Their method derives a numerical performance bound via the solution of an optimization problem. As a by-product of the optimization problem they generate a sequence of functions, each of which is a pointwise underestimator of the true value function. In this chapter we justify the use of these functions as a sequence of approximate value functions, sometimes referred to as control-Lyapunov functions. We do this by demonstrating

that the dual variables of the optimization problem correspond to a trajectory of first and second moments of the state and action of the system, under the policy admitted by the sequence of AVFs. We restrict our attention to the case with linear dynamics and quadratically representable stage cost, in which case the performance bound problem can be expressed as a semidefinite program (SDP). We conclude with some numerical examples to illustrate the technique.

3.2 Stochastic control

We begin by briefly reviewing the basics of stochastic control and the dynamic programming 'solution'. For more detail the interested reader is referred to, *e.g.*, [13,16]. We consider a discrete time dynamical system, with dynamics described by

$$x_{t+1} = f(x_t, u_t, w_t), \quad t = 0, 1, \ldots, \tag{3.1}$$

where $x_t \in \mathcal{X}$ is the system state, $u_t \in \mathcal{U}$ is the control input or action, $w_t \in \mathcal{W}$ is an exogenous noise or disturbance, all at time t, and $f : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathcal{X}$ is the state transition function. The noise terms w_t are independent identically distributed (IID), with known distribution. The initial state x_0 is also random with known distribution, and is independent of w_t.

The stage cost function is denoted $\ell : \mathcal{X} \times \mathcal{U} \to \mathbf{R} \cup \{+\infty\}$, where the infinite values of ℓ encode constraints on the states and inputs: The state-action *constraint set* is $\mathcal{C} = \{(z, v) \mid \ell(z, v) < \infty\} \subseteq \mathcal{X} \times \mathcal{U}$. (The problem is unconstrained if $\mathcal{C} = \mathcal{X} \times \mathcal{U}$.)

Consider causal time-invariant state feedback control policies of the form

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

where $\phi_t : \mathcal{X} \to \mathcal{U}$ is the *control policy* or *state feedback function* at time t. The *stochastic control problem* is to choose ϕ_t in order to minimize the infinite horizon discounted cost

$$J_\phi = \mathbf{E} \sum_{t=0}^{\infty} \gamma^t \ell(x_t, \phi_t(x_t)), \tag{3.2}$$

where $\gamma \in (0,1)$ is a discount factor. The expectations are over the noise terms w_t, $t = 0, 1, \ldots$, and the initial state x_0. We assume here that the expectation and limits exist, which is the case under various technical assumptions [13, 16]. We denote by $J^\star$ the optimal value of the stochastic control problem, *i.e.*, the infimum of J_ϕ over all policies $\phi : \mathcal{X} \to \mathcal{U}$. When the control policy functions ϕ_t do not depend on t, they are called time-invariant. For the stochastic control problem we consider it can be shown that there is always an optimal policy that is time-invariant.

3.2.1 Dynamic programming

In this section we briefly review the dynamic programming characterization of the solution to the stochastic control problem. For more details, see [13, 16].

The *value function* of the stochastic control problem, $V^\star : \mathcal{X} \to \mathbf{R} \cup \{\infty\}$, is given by

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

subject to the dynamics (3.1) and $x_0 = z$; the infimum is over all policies $\phi : \mathcal{X} \to \mathcal{U}$, and the expectation is over w_t for $t = 0, 1, \ldots$. The quantity $V^\star(z)$ is the expected cost incurred by an optimal policy, when the system is started from state z at time t. The optimal total discounted cost is given by

$$J^\star = \mathop{\mathbf{E}}_{x_0} V^\star(x_0). \tag{3.3}$$

It can be shown that the value function is the unique fixed point of the Bellman equation [9]

$$V^\star(z) = \inf_v \left(\ell(z, v) + \gamma \mathop{\mathbf{E}}_w V^\star(f(z, v, w)) \right)$$

for all $z \in \mathcal{X}$. We can write the Bellman equation in the form

$$V^\star = \mathcal{T} V^\star, \tag{3.4}$$

where we define the Bellman operator $\mathcal{T}$ as

$$(\mathcal{T}g)(z) = \inf_v \left(\ell(z,v) + \gamma \mathop{\mathbf{E}}_w g(f(z,v,w))\right)$$

for any $g : \mathcal{X} \to \mathbf{R} \cup \{+\infty\}$. Moreover, iteration of the Bellman operator, starting from any initial function, converges to the value function $V^\star$ (see [13, 16] for many more technical details and conditions). This procedure is referred to as value iteration.

An optimal policy (which is time-invariant) for the stochastic control problem is given by

$$\phi^\star(z) = \operatorname*{argmin}_v \left(\ell(z,v) + \gamma \mathop{\mathbf{E}}_w V^\star(f(z,v,w))\right), \tag{3.5}$$

for all $z \in \mathcal{X}$. (We drop the subscript t since this policy is time-invariant.)

3.2.2 Approximate dynamic programming

In many cases of interest, it is intractable to compute (or even represent) the value function $V^\star$, let alone carry out the minimization required evaluate the optimal policy (3.5). A common alternative is to replace the value function with an *approximate value function* $\hat{V}$ [19, 143, 179]. The resulting policy, given by

$$\hat{\phi}(z) = \operatorname*{argmin}_v \left(\ell(z,v) + \gamma \mathop{\mathbf{E}}_w \hat{V}(f(z,v,w))\right),$$

for all $z \in \mathcal{X}$, is called an *approximate dynamic programming* (ADP) policy. When $\hat{V} = V^\star$, the ADP policy is optimal. The goal of approximate dynamic programming is to find a $\hat{V}$ for which the ADP policy can be easily evaluated (for instance, by solving a convex optimization problem), and also attains near-optimal performance. We can also consider time-varying ADP policies, obtained from a sequence of approximate value functions $\hat{V}_t$, which results in a time-varying AVF policy. While an optimal policy can always be chosen to be time-invariant, a time-varying AVF policy can give better performance control than a time-invariant AVF policy.

3.3 Iterated approximate value function policy

In this section we introduce the iterated AVF policy. We begin by reviewing the iterated Bellman inequality and discuss how it can be used to generate performance bounds on stochastic control problems. Finally we introduce the iterated AVF policy.

3.3.1 Iterated Bellman inequality

Any function which satisfies the Bellman *inequality*,

$$V \leq \mathcal{T}V, \tag{3.6}$$

where the inequality is pointwise, is a guaranteed pointwise lower bound on the true value function (under some additional mild technical conditions) [13, 16, 47, 118, 159]. The basic condition works as follows. Suppose $V : \mathcal{X} \to \mathbf{R}$ satisfies $V \leq \mathcal{T}V$. Then by the monotonicity of the Bellman operator and convergence of value iteration [13, 16, 184], we have

$$V \leq \lim_{k\to\infty} \mathcal{T}^k V = V^\star,$$

so any function that satisfies the Bellman inequality must be a value function underestimator.

We can derive a weaker condition for being a lower bound on $V^\star$ by considering an iterated form of the Bellman inequality. Suppose we have a sequence of functions $V_t : \mathcal{X} \to \mathbf{R}$, $t = 0, \ldots, T+1$, that satisfy a chain of Bellman inequalities

$$V_0 \leq \mathcal{T}V_1, \quad V_1 \leq \mathcal{T}V_2, \quad \ldots \quad V_T \leq \mathcal{T}V_{T+1}, \tag{3.7}$$

with $V_T = V_{T+1}$. Then, using similar arguments as before, we can show that $V_t \leq V^\star$ for $t = 0, \ldots .T+1$. The condition is weaker since any function feasible for the single Bellman inequality is also feasible for the iterated Bellman inequality.

If we parameterize the functions to be linear combinations of k fixed basis functions

$V^{(i)} : \mathcal{X} \to \mathbf{R}$ with coefficient vectors $\alpha_t \in \mathbf{R}^k$, *i.e.*,

$$V_t = \sum_{i=1}^{k} \alpha_{ti} V^{(i)}, \tag{3.8}$$

for $t = 0, \ldots, T+1$, then the Bellman inequalities lead to convex constraints on the coefficients α_t. To see this, we write the Bellman inequality relating V_t and V_{t+1} as

$$V_t(z) \leq \inf_v \left(\ell(z, v) + \gamma \mathop{\mathbf{E}}_w V_{t+1}(f(z, v, w)) \right),$$

for all $z \in \mathcal{X}$. For each z, the left hand side is linear in α_t, and the right hand side is a concave function of α_{t+1}, since it is the infimum over a family of affine functions. Hence, the set of α_t, $t = 0, \ldots, T+1$ that satisfy the iterated Bellman inequalities (3.7) is convex [30].

3.3.2 Performance bound

Now that we have a tractable condition on value function lower-boundedness we can use it to generate a performance bound on the stochastic control problem, since if $V_0 : \mathcal{X} \to \mathbf{R}$ satisfies $V_0(x) \leq V^\star(x)$ for all $x \in \mathcal{X}$, then

$$J^{\mathrm{lb}} = \mathop{\mathbf{E}}_{x_0} V_0(x_0) \leq \mathop{\mathbf{E}}_{x_0} V^\star(x_0) = J^\star.$$

To find the best (*i.e.*, largest) lower bound we solve the following problem:

$$\begin{array}{ll} \text{maximize} & \mathbf{E}_{x_0} V_0(x_0) \\ \text{subject to} & V_t \leq \mathcal{T} V_{t+1}, \quad t = 0, \ldots, T \\ & V_T = V_{T+1} \end{array} \tag{3.9}$$

over variables $\alpha_0, \ldots, \alpha_{T+1}$. Since the iterated Bellman condition is a convex constraint on the coefficients $\alpha_t \in \mathbf{R}^K$, $t = 0, \ldots, T+1$, and the objective is linear in α_0 this is a convex optimization problem [30]. For (much) more detail on deriving bounds for stochastic control problems see [179, 180, 183] and the references therein.

Note that in [180], where the iterated Bellman inequality was first introduced, the authors used $V_0 = V_{T+1}$ as the terminal constraint. We replace that with $V_T = V_{T+1}$ here, which generally gives better numerical bounds.

3.3.3 Policy

By solving the performance bound problem (3.9) we obtain a sequence of approximate value functions $V_0, \ldots, V_{T+1}$, each of which is a lower bound on the true value function. The iterated AVF policy is given by

$$\phi_t(x) = \operatorname*{argmin}_u \left(\ell(x,u) + \gamma \, \mathbf{E} \, V_{t+1}(f(x,u,w))\right) \tag{3.10}$$

for $0 \leq t \leq T$, and

$$\phi_t(x) = \operatorname*{argmin}_u \left(\ell(x,u) + \gamma \, \mathbf{E} \, V_{T+1}(f(x,u,w))\right) \tag{3.11}$$

for $t > T$.

Note that for the problem we consider an optimal policy is time-invariant. However, the iterated AVF policy is *time-varying*. It may be advantageous to use a time-varying policy because, typically, an approximate value function cannot be a good approximation of the true value function everywhere, so knowledge about the initial state of the system, and subsequent states under the iterated AVF policy, can be exploited to our advantage. The rest of this chapter is a justification for the use of this time-varying policy in the particular case of linear dynamics and quadratically representable stage cost.

We briefly mention another policy, the pointwise maximum policy:

$$\phi(x) = \operatorname*{argmin}_u \left(\ell(x,u) + \gamma \max_{t=0,\ldots,T} \mathbf{E} \, V_t(f(x,u))\right). \tag{3.12}$$

Note that this policy is time-invariant. Since each V_t is an underestimator of the true value function, the pointwise maximum of these is also an underestimator and is at least as good an approximation of the true value function as any individual V_t.

However, this policy is much more expensive to compute, and complexity grows with horizon T.

3.4 Quadratically representable case

We restrict our attention to the case with linear dynamics and quadratically representable stage cost and constraint set. We consider this limited case for simplicity, but the results in this chapter extend to other cases with some minor modifications, such as time-varying dynamics, random dynamics, and a finite horizon.

We consider the case where the state and action spaces are finite dimensional vector spaces, *i.e.*, $x_t \in \mathbf{R}^n$ and $u_t \in \mathbf{R}^m$, and the dynamics equation has the form

$$x_{t+1} = f(x_t, u_t, w_t) = Ax_t + Bu_t + w_t, \quad t = 0, 1, \ldots$$

for some matrices $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{n\times m}$. We will write the dynamics as

$$G \begin{bmatrix} x_{t+1} \\ u_{t+1} \\ 1 \end{bmatrix} = F \begin{bmatrix} x_t \\ u_t \\ 1 \end{bmatrix} + \begin{bmatrix} w_t \\ 1 \end{bmatrix}$$

where

$$F = \begin{bmatrix} A & B & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad G = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

We pad the state-action vector with an additional 1 so that it has dimension $l = n + m + 1$, which will allow more compact notation in the sequel. We assume that the noise term has zero mean, *i.e.*, $\mathbf{E}(w_t) = 0$ for all t, and has second moment $\mathbf{E}\, w_t w_t^T = \hat{W}$ for all t. For compactness of notation we let

$$\mathbf{E} \begin{bmatrix} w_t \\ 0 \end{bmatrix} \begin{bmatrix} w_t \\ 0 \end{bmatrix}^T = \begin{bmatrix} \hat{W} & 0 \\ 0 & 0 \end{bmatrix} = W$$

where $W \in \mathbf{R}^{(n+1)\times(n+1)}$.

We consider stage cost functions of the form

$$\ell(x,u) = \begin{cases} \bar{\ell}(x,u) & (x,u) \in \mathcal{C} \\ \infty & \text{otherwise,} \end{cases}$$

where

$$\bar{\ell}(x,u) = \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T L \begin{bmatrix} x \\ u \\ 1 \end{bmatrix},$$

$L \in \mathbf{R}^{l\times l}$, is a convex quadratic function and where $\mathcal{C}$ denotes the feasible set of state-action pairs. We assume that we can write the feasible set as the intersection of $k+1$ convex quadratic constraint sets, *i.e.*,

$$\mathcal{C} = \left\{ (x,u) \left| \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \Sigma_i \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} \leq 0, \ i = 0, \ldots, k \right. \right\} \tag{3.13}$$

where $\Sigma_i \in \mathbf{R}^{l\times l}$, $i = 0, \ldots, k$.

We shall parameterize the approximate value functions to be convex quadratic functions of the state, *i.e.*,

$$V_t(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^T P_t \begin{bmatrix} x \\ 1 \end{bmatrix}$$

for some $P_t \in \mathbf{R}^{n+1\times n+1}$. With this choice of approximate value function the parameter P_t takes role of α_t in (3.8) and (3.9). Thus the iterated Bellman inequalities are a set of convex constraints on the parameters $P_0, \ldots, P_{T+1}$. This choice of approximate value function also ensures that the policy problems (3.10) and (3.11) are convex optimization problems and can be solved efficiently [30, 182].

3.4.1 Iterated Bellman inequalities

With the notation we have established, we can write

$$\mathbf{E}\,V(x_{t+1}) = \mathbf{E}\,V(Ax_t + Bu_t + w_t) = \begin{bmatrix} x_t \\ u_t \\ 1 \end{bmatrix}^T F^T P F \begin{bmatrix} x_t \\ u_t \\ 1 \end{bmatrix} + \mathbf{Tr}(PW).$$

We denote by $\mathcal{U}(z) = \{u \mid (z,u) \in \mathcal{C}\}$ the set of feasible actions at a given state z. The single Bellman inequality is written

$$V(x) \leq \min_{u \in \mathcal{U}(x)} \left(\ell(x,u) + \gamma\,\mathbf{E}\,V(Ax + Bu + w)\right),$$

for all $x \in \mathcal{X}$, which with our notation is

$$\begin{bmatrix} x \\ 1 \end{bmatrix}^T P \begin{bmatrix} x \\ 1 \end{bmatrix} \leq \gamma\,\mathbf{Tr}(PW) + \min_{u \in \mathcal{U}(x)} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T (L + \gamma F^T P F) \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}$$

for all x such that $\mathcal{U}(x)$ is non-empty, or equivalently

$$\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T (L + \gamma F^T P F - G^T P G) \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} + \gamma\,\mathbf{Tr}(PW) \geq 0 \tag{3.14}$$

for all $(x,u) \in \mathcal{C}$. This constraint is convex (indeed affine) in the variable P. However it is semi-infinite, since it is a family of constraints parametrized by the (infinite) set $(x,u) \in \mathcal{C}$. The $\mathcal{S}$-procedure [26] provides a sufficient condition that ensures (3.14) holds for all states. By using the $\mathcal{S}$-procedure, we can approximate the iterated Bellman inequalities as linear matrix inequalities (LMIs) and, in turn, (3.9) can be approximated as an SDP, which can be solved efficiently. Since the $\mathcal{S}$-procedure is sufficient, but not necessary, the resulting approximate value functions found by

solving the SDP will still be pointwise underestimators of the true value function, and the numerical performance bound will still be valid.

The set $\mathcal{C}$ is defined by quadratic inequalities parameterized by Σ_i, $i = 0, \ldots, k$. From the $\mathcal{S}$-procedure we have that the Bellman inequality will hold if there exists $\lambda \in \mathbf{R}_+^{k+1}$ such that

$$L + \gamma F^T P F - G^T P G + \gamma \, \mathbf{Tr}(PW) e_l e_l^T \succeq -\textstyle\sum_{i=0}^k \lambda^i \Sigma_i$$

where e_l is the unit vector in the lth coordinate and λ^i denotes the ith component of the vector λ.

The extension of the single Bellman inequality to the iterated case is written

$$L + \gamma F^T P_{t+1} F - G^T P_t G + \gamma \, \mathbf{Tr}(P_{t+1} W) e_l e_l^T \succeq -\textstyle\sum_{i=0}^k \lambda_t^i \Sigma_i$$

for $t = 0, \ldots, T$, along with the end condition $P_T = P_{T+1}$. This condition is convex in parameters $P_0, \ldots, P_{T+1} \in \mathbf{R}^{l \times l}$ and $\lambda_0, \ldots, \lambda_T \in \mathbf{R}_+^k$, in particular it is a linear matrix inequality (LMI) [26, 30].

3.4.2 Performance bound problem

In this section we will define the set $\mathcal{P}_t^T \subset \mathbf{R}^{(n+1)\times(n+1)}$, which is the set of convex quadratics,parameterized by P_t, for which there exists $P_{t+1}, \ldots, P_{T+1} \in \mathbf{R}^{(n+1)\times(n+1)}$ that together satisfy $T + 1 - t$ iterated Bellman inequalities. Thus any $P_t \in \mathcal{P}_t^T$ defines a convex quadratic function that is a guaranteed lower bound on the true value function.

First, the iterated Bellman inequalities must hold, *i.e.*,

$$L + \gamma F^T P_{\tau+1} F - G^T P_\tau G + \gamma \, \mathbf{Tr}(P_{\tau+1} W) e_l e_l^T \succeq -\textstyle\sum_{i=0}^k \lambda_\tau^i \Sigma_i, \quad \tau = t, \ldots, T \tag{3.15}$$

where

$$\lambda_\tau \in \mathbf{R}_+^{k+1}, \quad \tau = t, \ldots, T. \tag{3.16}$$

For convexity of the approximate value functions we require

$$P_\tau = \begin{bmatrix} \hat{P}_\tau & p_\tau \\ p_\tau^T & r_\tau \end{bmatrix}, \ \hat{P}_\tau \in \mathbf{S}_+^n, \quad \tau = t, \ldots, T+1. \tag{3.17}$$

Finally we require the terminal constraint

$$P_T = P_{T+1}. \tag{3.18}$$

We denote by $\mathcal{P}_t^T$ the set of parameters that satisfy these conditions, *i.e.*,

$$\begin{aligned} \mathcal{P}_t^T \ = \ & \{P_t \mid \exists \ P_{t+1}, \ldots, P_{T+1}, \lambda_t, \ldots, \lambda_T, \\ & \qquad \text{such that } (3.15), (3.16), (3.17), (3.18) \text{ are satisfied}\}. \end{aligned}$$

With this notation the problem of finding a lower bound on the performance of the optimal policy can be expressed as an SDP in the variables $P_0, \ldots, P_{T+1}$ and $\lambda_0, \ldots, \lambda_T$,

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}(P_0 X_0) \\ \text{subject to} & P_0 \in \mathcal{P}_0^T \end{array} \tag{3.19}$$

where

$$X_0 = \mathbf{E} \begin{bmatrix} x_0 \\ 1 \end{bmatrix} \begin{bmatrix} x_0 \\ 1 \end{bmatrix}^T$$

contains the first and second moments of the initial state.

3.5 Iterated Quadratic AVFs

The goal in this section is to justify the iterated AVF policy given in (3.10) and (3.11), *i.e.*, using the chain of value functions, arising from solving (3.19), as a sequence of approximate value functions. We assume throughout that (3.19) is feasible and that the optimal value is attained.

3.5.1 Relaxed policy problem

Here we introduce what we refer to as the relaxed policy problem. This is the problem of minimizing the *expected* cost in the policy problems (3.10) or (3.11), where instead of exact knowledge of the state we know only its first and second moments, and where we relax the constraints to hold *in expectation*. (The relaxed policy problem reduces to the standard policy problem when the state is known exactly). For an in-depth treatment on the use of moment relaxations in optimal control see [157]. This problem is written

$$\begin{array}{ll} \text{minimize} & \mathbf{E}\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T (L + \gamma F^T P_{t+1} F) \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} \\ \text{subject to} & \mathbf{E}\begin{bmatrix} x \\ 1 \end{bmatrix}\begin{bmatrix} x \\ 1 \end{bmatrix}^T = X_t \\ & \mathbf{E}\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \Sigma_i \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} \leq 0, \quad i = 0, \ldots, k \end{array}$$

where X_t contains the first and second moment information about the state at time t. If we let

$$Z_t = \mathbf{E}\left(\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T\right) \in \mathbf{S}_+^{l \times l}$$

we can write the problem as

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}(L + \gamma F^T P_{t+1} F) Z_t \\ \text{subject to} & G Z_t G^T = X_t \\ & Z_t \succeq 0 \\ & \mathbf{Tr}(\Sigma_i Z_t) \leq 0, \quad i = 0, \ldots, k \end{array} \tag{3.20}$$

over variable $Z_t \in \mathbf{R}^{l\times l}$. The solution matrix $Z_t^\star$ contains the first and second moments of the state and action that minimize the expected cost-to-go, using V_{t+1} as the value function. Since we have relaxed the constraints to hold in expectation, we shall refer to Z_t as the *relaxed* second moment at time t.

3.5.2 Saddle point problems

In this subsection we shall show that the dual variables of the performance bound problem (3.19) correspond to the relaxed second moments of the state and action under the policy admitted by the sequence of approximate value functions. We will show that the sequence of approximate value functions is optimized to this trajectory, which concludes our justification.

We start by forming the partial Lagrangian of the problem, where we introduce dual variable $Z_0 \succeq 0$ for the Bellman inequality constraint relating P_0 and P_1, which is written

$$\begin{aligned} L(P_0, P_1, Z_0) &= \mathbf{Tr}(P_0 X_0) + \gamma\,\mathbf{Tr}(P_1 W) e_l^T Z_0 e_l \\ &\quad + \mathbf{Tr}\, Z_0 \left(L + \gamma F^T P_1 F - G^T P_0 G - \textstyle\sum_{i=0}^k \lambda_0^i \Sigma_i\right) \end{aligned}$$

where we have the additional constraint that $P_1 \in \mathcal{P}_1^T$. If we analytically maximize this over P_0 and λ_0 we obtain the following constraint set of constraints on Z_0:

$$\mathcal{Z}_0 = \left\{ Z \mid GZG^T = X_0,\ \mathbf{Tr}(\Sigma_i Z) \leq 0,\ Z \succeq 0 \right\}.$$

These constraints correspond exactly to the constraints in the relaxed second moment problem (3.20). The first constraint above requires Z_0 to be in consensus with the supplied second order information about the state, and the second constraint requires the state and action to satisfy the constraints (3.13) in expectation. From the first constraint we also have that $e_l^T Z_0 e_l = 1$. Making the appropriate substitutions we

arrive at the saddle point problem

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}(LZ_0) + \gamma \max\limits_{P_1 \in \mathcal{P}_1^T} \mathbf{Tr}(FZ_0F^T + W)P_1 \\ \text{subject to} & Z_0 \in \mathcal{Z}_0 \end{array} \tag{3.21}$$

over Z_0. If we swap the order of maximization and minimization in (3.21) we have

$$\begin{array}{ll} \text{maximize} & \gamma\,\mathbf{Tr}(P_1W) + \min\limits_{Z_0 \in \mathcal{Z}_0} \mathbf{Tr}(L + \gamma F^T P_1 F)Z_0 \\ \text{subject to} & P_1 \in \mathcal{P}_1^T \end{array} \tag{3.22}$$

over variables $P_1, \ldots, P_T$ and $\lambda_1, \ldots, \lambda_{T-1}$. Assuming strong duality, problems (3.21) and (3.22) are equivalent and attain the same optimal value as (3.19). The second term in problem (3.22) is identical to problem (3.20) for $t = 0$. This implies that the optimal Z_0 is the optimal second moment of the *relaxed* problem (3.20) at $t = 0$ using V_1 as the value function, *i.e.*, we can interpret Z_0 as

$$Z_0 = \mathbf{E}\left(\begin{bmatrix} x_0 \\ u_0 \\ 1 \end{bmatrix}\begin{bmatrix} x_0 \\ u_0 \\ 1 \end{bmatrix}^T\right).$$

With this we can write

$$\begin{aligned} FZ_0F^T + W &= \mathbf{E}\begin{bmatrix} Ax_0 + Bu_0 + w \\ 1 \end{bmatrix}\begin{bmatrix} Ax_0 + Bu_0 + w \\ 1 \end{bmatrix} \\ &= \mathbf{E}\begin{bmatrix} x_1 \\ 1 \end{bmatrix}\begin{bmatrix} x_1 \\ 1 \end{bmatrix}^T, \end{aligned}$$

and thus we can rewrite the second term in problem (3.21) as

$$\max_{P_1 \in \mathcal{P}_1^T} \mathbf{Tr}(FZ_0F^T + W)P_1 = \max_{P_1 \in \mathcal{P}_1^T} \mathbf{E} \begin{bmatrix} x_1 \\ 1 \end{bmatrix}^T P_1 \begin{bmatrix} x_1 \\ 1 \end{bmatrix}.$$

This implies that the optimal P_1 is the quadratic lower bound that maximizes the expected cost over states at time $t = 1$ (and satisfies the iterated Bellman inequalities).

The above argument is repeated inductively: At iteration t we introduce a new dual variable Z_t for the Bellman inequality involving P_t and P_{t+1} and show that it corresponds to the relaxed second moment of the state and action at time t, provided the same holds for Z_{t-1}. Then, since the relaxed second moment of the state at time $t + 1$ is determined by applying the dynamics equations to Z_t, it follows that P_{t+1} maximizes the expected cost over states at time $t + 1$ (while satisfying the Bellman inequalities). This is repeated up to $t = T$. Finally, if the relaxed second moment of the state converges to a steady-state and the horizon T is large enough, then, by the argument above, P_T (and therefore P_{T+1}) is optimized to that steady-state distribution, which justifies the long-term policy (3.11).

3.6 Examples

Here we introduce two examples to demonstrate the efficacy of the iterated AVF policy. For other practical examples of formulating the bound problem (3.9) as an SDP see [27, 135, 179, 180].

3.6.1 One-dimensional example

In this instance we take $n = m = 1$ and $\gamma = 0.99$, the dynamics were given by $x_{t+1} = x_t + u_t$ and the cost function was chosen to be

$$\ell(x, u) = \begin{cases} x^2 + (0.1)u^2 & |u| \leq 1, \\ \infty & \text{otherwise.} \end{cases}$$

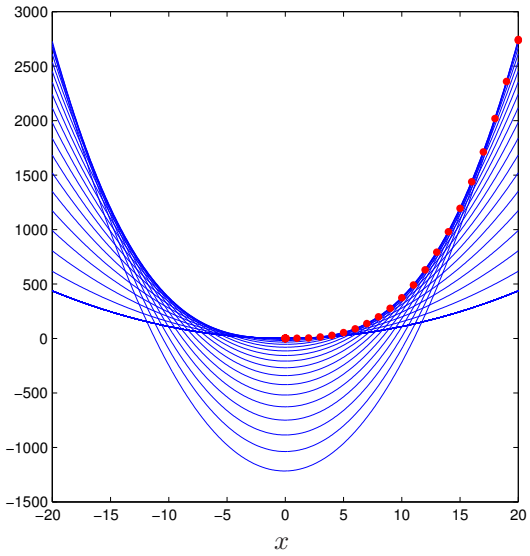

Figure 3.1: AVF sequence and state trajectory.

This allows us to visually inspect the approximate value functions and the true value function for this particular problem.

We solve the performance bound problem (3.19) with a horizon of $T = 25$ and with exact knowledge of the initial state, which we take to be $x_0 = 20$. Figure 3.1 shows the sequence of value functions generated by solving problem (3.19). The red dots are the trajectory of first moments of the state x_t, extracted from the dual variables Z_t, the blue curves are the corresponding quadratic value functions defined by P_t. Note that each quadratic is a good approximation in some regions, namely where the state is expected to be at that time, and a bad approximation in other regions. In this case we can find the true value function by discretizing the state and action spaces and performing value iteration [13, 16]. Figure 3.2 shows the approximate value function given by the pointwise maximum over all 25 quadratics in blue and the true value function in red. In this case the two are indistinguishable.

The performances of various policies from x_0 were evaluated using Monte Carlo. The iterated AVF policy obtained 2747.6, almost identical to the lower bound of 2745.0; a single approximate value function achieved 2750.1, a small but significant difference. The single AVF was generated by solving (3.9) with $T = 0$. The more

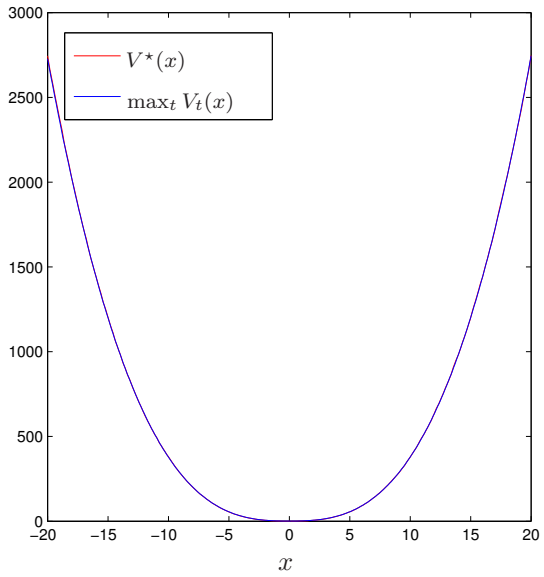

Figure 3.2: True value function and pointwise max over AVFs.

computationally expensive model predictive control (MPC) policy [64, 117, 125, 138] with a lookahead of 50 time-steps, achieved a performance of 2745.1.

3.6.2 Box constrained quadratic control

This example is similar to an example presented in [135]. We control a linear dynamical system with stage cost

$$\ell(x,u) = \begin{cases} x^T Q x + u^T R u & \|u\|_\infty \leq 1 \\ \infty & \text{otherwise,} \end{cases}$$

for some $Q \succeq 0$ and $R \succeq 0$, *i.e.*, our action is constrained to lie in a box at all time periods. The constraint that $\|u_t\|_\infty \leq 1$ can be rewritten as

$$(u_t)_i^2 \leq 1, \quad i = 1, \ldots, m.$$

We randomly generated a numerical instance with $n = 12$ and $m = 4$. The dynamics matrix A was randomly generated, and then scaled to have spectral radius 1. The

Policy	Performance
lower bound	50737
MPC	50923
iterated AVF	51286
single AVF	52479

Table 3.1: Lower bound, and policy performances.

horizon length for both MPC and the lower bound calculation was set to 50, the value of γ was chosen to be 0.99. We ran 1000 Monte Carlo simulations of 500 time periods to estimate performance of various policies. We compared the performance of the iterated AVF policy with a single AVF policy, where we solve (3.9) with $T = 0$. We also compared to MPC with a 50 step-lookahead. The results are summarized in table 3.1. Custom interior point solvers were generated by CVXGEN [122] and used to solve the policy problems for both the AVF policies and the MPC policy. Computation was carried out on a 4-core Intel Xeon processor, with clock speed 3.4GHz and 16Gb of RAM, running Linux. On average, it took 9ms to solve the MPC problem at each iteration, whereas it only took 83μs to solve the AVF policy problem, more than two orders of magnitude faster.

From the lower bound we can certify that MPC is no more than 0.4% suboptimal, the iterated AVF policy is no more than 1.1% suboptimal, and the single AVF policy is no more than 3.4% suboptimal.

3.7 Summary

In this chapter we introduced the iterated approximate value function policy, which is a time-varying policy even in the case where the optimal policy is time-invariant. The sequence of approximate value functions we use is derived from a performance bound problem, which, for the case we consider, is a convex optimization problem and thus tractable. We justified the use of this sequence of approximate value functions by considering the dual of the performance bound problem. We showed that the dual

variables of this problem could be interpreted as a trajectory of moments of the state and action for the stochastic control problem. We concluded with some examples to show the performance of our policy.

Part III

Model Predictive Control Policies

Chapter 4

A Splitting Method for Optimal Control

We apply an operator splitting technique to a generic linear-convex optimal control problem, which results in an algorithm that alternates between solving a quadratic control problem, for which there are efficient methods, and solving a set of single-period optimization problems, which can be done in parallel, and often have analytical solutions. In many cases the resulting algorithm is division-free (after some off-line pre-computations) and so can be implemented in fixed-point arithmetic, for example on a field-programmable gate array (FPGA). We demonstrate the method on several examples from different application areas.

4.1 Introduction

We consider a linear-convex optimal control problem, *i.e.*, the problem of finding a trajectory of state-control pairs that satisfy a set of linear dynamics, and minimize a sum of convex stage-cost functions. This problem arises naturally in many application areas, including model predictive control (MPC), moving horizon estimation (MHE), and trajectory planning. Finding an optimal trajectory involves solving a convex optimization problem, which can in principle be done efficiently, using generic

techniques for convex optimization, or methods developed specifically for the linear-convex optimal control problem, such as custom interior-point methods. We are interested here in real-time applications, which require very high execution speed, and simplicity of the algorithm. On the other hand most real-time applications do not require the solution to be computed to high accuracy; assuming the variables have been appropriately scaled, three digits of accuracy is usually more than adequate.

In this chapter we describe yet another method to solve linear-convex optimal control problems quickly. The algorithm we present relies on an operator splitting technique, referred to as the Alternating Direction Method of Multipliers (ADMM), or as Douglas-Rachford (D-R) splitting. Operator splitting breaks the problem into two parts, a quadratic optimal control problem (which can be solved very efficiently), and a set of single period optimization problems (which can be solved in parallel, often, analytically). An iteration that alternates these two steps then converges to a solution. We demonstrate that our method can solve optimal control problems to an acceptable accuracy very rapidly, indicating that it is suitable for use in, *e.g.*, high-frequency control applications. Another advantage of our method is that in many cases, after some off-line pre-computation, the algorithm requires no division operations. In these cases it can be implemented in fixed-point arithmetic, for example on a field-programmable gate array (FPGA) for high-speed embedded control.

4.2 Convex optimal control problem

4.2.1 Problem description

We consider the (deterministic, discrete-time, finite-horizon) linear-convex optimal control problem

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T} \left(\phi_t(x_t, u_t) + \psi_t(x_t, u_t)\right) \\ \text{subject to} & x_{t+1} = A_t x_t + B_t u_t + c_t, \quad t = 0, \ldots, T-1 \\ & x_0 = x_{\text{init}}, \end{array} \tag{4.1}$$

with variables (states) $x_t \in \mathbf{R}^n$ and (controls) $u_t \in \mathbf{R}^m$, $t = 0, \ldots, T$. The stage cost function is split into a convex quadratic part ϕ_t and a convex non-quadratic part ψ_t. The convex quadratic terms $\phi_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ have the form

$$\phi_t(x,u) = (1/2)\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} Q_t & S_t & q_t \\ S_t^T & R_t & r_t \\ q_t^T & r_t^T & 0 \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}$$

where

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

(*i.e.*, is symmetric positive semidefinite). The non-quadratic terms $\psi_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$, are closed proper convex functions (see, *e.g.*, [151]). Infinite values in the functions ψ_t encode convex constraints on the states and controls. For example, when ψ_t is the indicator function of a closed convex set $\mathcal{C}_t \subset \mathbf{R}^{n+m}$,

$$\psi_t(x_t, u_t) = I_{\mathcal{C}_t}(x_t, u_t) = \begin{cases} 0 & (x_t, u_t) \in \mathcal{C}_t \\ \infty & \text{otherwise,} \end{cases}$$

the stage cost term $\psi_t(x_t, u_t)$ in (4.1) simply imposes the state-control constraint $(x_t, u_t) \in \mathcal{C}_t$.

The problem data are the initial state $x_{\text{init}} \in \mathbf{R}^n$, the dynamics matrices A_t, B_t, and c_t, for $t = 0, \ldots, T-1$, the quadratic cost term coefficients Q_t, R_t, S_t, q_t, and r_t, for $t = 0, \ldots, T$, and the non-quadratic cost functions ψ_t, $t = 0, \ldots, T$. The decomposition of the stage cost into the quadratic part ϕ_t and the non-quadratic part ψ_t is in general not unique, since a quadratic stage cost term can be included in the non-quadratic term.

Problem (4.1) is a convex optimization problem and as such it can solved efficiently by a variety of generic methods including, *e.g.*, interior point methods [30, 187]. Here 'efficiently' means that the computational effort required to obtain a solution grows no faster than a polynomial of the problem size. This chapter is about solving the

linear-convex optimal control problem (4.1) very quickly, even by comparison to these generic methods.

4.2.2 Usage scenarios

We list here some ways in which a solver for the problem (4.1) can be used.

Cold start. Here we solve (4.1) just once, or many times, but with very different (unrelated) data each time.

Warm start. Here we solve (4.1) many times sequentially, where the data in each problem is similar to the data for the previously solved problem. In this case we initialize the algorithm with the solution from the previous problem to obtain a speed-up over the cold start scenario; the speed-up will depend on how similar the data are from one iteration to the next.

Constant quadratic case. Here we solve (4.1) many times, where the quantities Q_t, R_t, S_t, $t = 0, \ldots, T$ and A_t, B_t, $t = 0, \ldots, T-1$, do not change. (The initial state and non-quadratic stage cost terms can change in each instance.) In this case we can cache some quantities that are pre-computed, allowing us to reduce the computation required to solve instances of the problem. Depending on the non-quadratic stage cost terms, our algorithm can be division free; as a result it can be implemented in fixed-point arithmetic on, *e.g.*, an FPGA.

Warm start constant quadratic. Here we have both the warm start and constant quadratic cases, in which case the computational savings stack.

4.2.3 Notation

We use $x = (x_0, \ldots, x_T)$ and $u = (u_0, \ldots, u_T)$ to denote the concatenated states and controls (*i.e.*, their trajectories), and $(x, u) \in \mathbf{R}^{(n+m)(T+1)}$ to denote the whole

state-control trajectory. We define

$$\mathcal{D} = \{(x, u) \mid x_0 = x_{\text{init}},\, x_{t+1} = A_t x_t + B u_t + c_t,\, t = 0, \ldots, T-1\},$$

which is the set of state-control pairs that satisfy the dynamics of the problem (4.1), and we use $I_{\mathcal{D}}$ to denote the indicator function of the set $\mathcal{D}$, which is a closed proper convex function. We define

$$\phi(x, u) = \sum_{t=0}^{T} \phi_t(x_t, u_t), \qquad \psi(x, u) = \sum_{t=0}^{T} \psi_t(x_t, u_t),$$

which are the quadratic and non-quadratic costs over the whole trajectory (x, u). With this notation the convex optimal control problem can be expressed as

$$\text{minimize} \quad I_{\mathcal{D}}(x, u) + \phi(x, u) + \psi(x, u)$$

with variables $(x, u) \in \mathbf{R}^{(n+m)(T+1)}$.

4.2.4 Prior and related work

In this section we give a brief overview of some of the important prior work in several related areas.

Interior-point methods. A generic interior-point solver for (4.1) that does not exploit the problem structure would scale in complexity with the cube of the time-horizon [22]. If the structure of the problem is exploited, however, the complexity only grows linearly. In [181] the authors developed a custom interior-point method that can solve quadratic optimal control problems with box constraints very rapidly by exploiting problem structure. A similar approach was taken in [147]. For work detailing efficient primal-dual interior-point methods to solve the quadratic programs (QPs) that arise in optimal control see [2, 79, 80].

Automatic code generation. Typically creating a custom interior-point solver is a very labor-intensive exercise. In [122] the authors describe the automatic generation of high speed custom solvers directly from high level descriptions of the problem. These automatically generated custom solvers are tailored to the problem at hand, providing dramatic speed-ups over generic solvers. For work detailing custom code generation specifically for optimal control problems see, *e.g.*, [52, 89, 124, 136, 137].

Explicit MPC. Explicit model predictive control is a technique for solving quadratic optimal control problems with polyhedral constraints [11, 167], with all data fixed except the initial state. In this case the solution is a piecewise affine function of the initial state. The polyhedra that define the regions, and the associated coefficients in the affine function, can be computed off-line. Solving the problem then reduces to searching in a lookup table, and then evaluating the affine function (which is division free). Due to the exponential growth in the number of regions, explicit MPC can realistically only be applied to systems with very modest numbers of states and constraints. For an extension that can handle larger problems by using partial enumeration see [140].

Active set methods. Active set methods are a set of techniques for solving QPs that are closely related to the simplex method for linear programming. They rely on identifying the set of constraints that are active at the optimum and then solving a simpler problem using just these constraints [7, 21, 68, 69, 71]. The use of active set methods to solve the QPs that arise in control has been explored by Ferreau et al. in [60, 61].

Fast gradient methods. Fast gradient methods, inspired by Nesterov's accelerated first order methods [130, 132], have been applied to the optimal control problem [100–102, 148, 150]. These techniques typically require only the evaluation of a gradient and a projection at each iteration. Thus, they generally require less computation than, say, interior-point methods, at the expense of high accuracy.

Embedded control. There has been much recent interest in using MPC in an embedded control setting, for example in autonomous or miniature devices. The challenge is to develop algorithms that can solve (4.1) quickly, robustly and within the limitations of on-board chip architectures. Many techniques have been investigated, including interior-point methods, active set methods and others; see, *e.g.*, [93,106,110, 115]. In this chapter we develop an algorithm that (in some cases) can be implemented without division, which allows us to use fixed point arithmetic. For other work exploring the use of fixed point arithmetic in control see [92,149,150]. For other work on implementing control algorithms on FPGAs, see, *e.g.*, [81, 91, 177, 185, 186].

Operator splitting. The technique we employ in this chapter relies on the work done on monotone operators and operator splitting methods. The history of operator splitting goes back to the 1950s; ADMM itself was introduced in the mid-1970s by Glowinski and Marrocco [70] and Gabay and Mercier [63]. It was shown in [62] that ADMM is a special case of a splitting technique known as Douglas-Rachford splitting, and Eckstein and Bertsekas [57] showed in turn that Douglas-Rachford splitting is a special case of the proximal point algorithm. For convergence results for operator splitting methods see, *e.g.*, [57, 62, 83, 84]. For (much) more detail about operator splitting algorithms and example applications see [28] and the references therein. Operator splitting has seen use in many application areas, see, *e.g.*, [6,39,178]. In [108] the authors use operator splitting to develop sparse feedback gain matrices for linear-quadratic control problems.

Recently, accelerated variants of operator splitting methods have been proposed. Although we do not use these variants in this chapter we mention them here for completeness. These techniques apply a Nesterov-type acceleration to the iterates [130], which under certain conditions can dramatically improve the rate of convergence [72, 73, 75].

4.3 Splitting method

4.3.1 Consensus form

We can write the optimal control problem (4.1) in the following form:

$$\begin{array}{ll} \text{minimize} & (I_{\mathcal{D}}(x,u) + \phi(x,u)) + \psi(\tilde{x},\tilde{u}) \\ \text{subject to} & (x,u) = (\tilde{x},\tilde{u}), \end{array} \tag{4.2}$$

with variables $(x,u) \in \mathbf{R}^{(n+m)(T+1)}$, $(\tilde{x},\tilde{u}) \in \mathbf{R}^{(n+m)(T+1)}$. This form is referred to as *consensus*; see [28, §7]. Here we split the objective into two separate parts, with different variables. The first term contains the quadratic objective and the dynamic constraints; the second term is separable across time, and encodes the constraints and non-quadratic objective terms on the states and control in each period. The equality constraint requires that they be in consensus.

4.3.2 Operator splitting for control

The consensus form of operator splitting is the following algorithm. Starting from any initial $(\tilde{x}^0,\tilde{u}^0)$ and (z^0,y^0), for $k = 0,1,\ldots$,

$$\begin{aligned} (x^{k+1},u^{k+1}) &:= \underset{(x,u)}{\operatorname{argmin}} \left(I_{\mathcal{D}}(x,u) + \phi(x,u) + (\rho/2)\|(x,u) - (\tilde{x}^k,\tilde{u}^k) - (z^k,y^k)\|_2^2\right) \\ (\tilde{x}^{k+1},\tilde{u}^{k+1}) &:= \underset{(\tilde{x},\tilde{u})}{\operatorname{argmin}} \left(\psi(\tilde{x},\tilde{u}) + (\rho/2)\|(\tilde{x},\tilde{u}) - (x^{k+1},u^{k+1}) + (z^k,y^k)\|_2^2\right) \qquad (4.3) \\ (z^{k+1},y^{k+1}) &:= (z^k,y^k) + (\tilde{x}^{k+1},\tilde{u}^{k+1}) - (x^{k+1},u^{k+1}). \qquad (4.4) \end{aligned}$$

Here $\rho > 0$ is an algorithm parameter, k is the iteration counter, and $(z^k,y^k) \in \mathbf{R}^{(n+m)(T+1)}$ is a (scaled) dual variable associated with the consensus constraint. We shall refer to this technique as operator splitting for control (OSC).

The first step entails minimizing a sum of convex quadratic functions of the state and control, subject to the dynamics, *i.e.*, a convex quadratic control problem. The objective in the second step is separable across time, so the minimization for each

time period can be carried out separately, as

$$(\tilde{x}_t^{k+1}, \tilde{u}_t^{k+1}) := \underset{(\tilde{x}_t, \tilde{u}_t)}{\operatorname{argmin}} \left(\psi_t(\tilde{x}_t, \tilde{u}_t) + (\rho/2) \|(\tilde{x}_t, \tilde{u}_t) - (x_t^{k+1}, u_t^{k+1}) + (z_t^k, y_t^k)\|_2^2 \right),$$

for $t = 0, \ldots, T$. The right-hand side is the *proximal* (or prox) operator of ψ_t, evaluated at

$$(x_t^{k+1} - z_t^k, u_t^{k+1} - y_t^k).$$

For more details about prox operators and derivations of prox operators for some common functions see, *e.g.*, [40, 174].

Convergence and stopping criteria. Under some mild conditions, the splitting algorithm converges to the solution (assuming there is one); see [28, §3.2]. The *primal residual* for (4.2) is given by

$$r^k = (x^k, u^k) - (\tilde{x}^k, \tilde{u}^k),$$

and the *dual residual* as

$$s^k = \rho((\tilde{x}^k, \tilde{u}^k) - (\tilde{x}^{k-1}, \tilde{u}^{k-1})).$$

It can be shown that r^k and s^k converge to zero under OSC. A suitable stopping criterion is when the residuals are smaller than some threshold, *i.e.*,

$$\|r^k\|_2 \leq \epsilon^{\text{pri}}, \quad \|s^k\|_2 \leq \epsilon^{\text{dual}},$$

where $\epsilon^{\text{pri}} > 0$ and $\epsilon^{\text{dual}} > 0$ are tolerances for primal and dual feasibility respectively. One way to assign these tolerances is the following [28, §3.3.1]:

$$\begin{aligned} \epsilon^{\text{pri}} &= \epsilon^{\text{abs}} \sqrt{(T+1)(n+m)} + \epsilon^{\text{rel}} \max\{\|(x^k, u^k)\|_2, \|(\tilde{x}^k, \tilde{u}^k)\|_2\} \\ \epsilon^{\text{dual}} &= \epsilon^{\text{abs}} \sqrt{(T+1)(n+m)} + \epsilon^{\text{rel}} \|(z^k, y^k)\|_2, \end{aligned} \tag{4.5}$$

where $\epsilon^{\text{abs}} > 0$ and $\epsilon^{\text{rel}} > 0$ are absolute and relative tolerances respectively. This choice ensures that the primal and dual tolerances scale with the size of the problem and the scale of the typical variable values. In any particular application, however, it can be simpler to choose fixed values for the primal and dual tolerances ϵ^{pri} and ϵ^{dual}.

Douglas-Rachford splitting can take many iterations to converge to high accuracy. However, modest accuracy is typically achieved within a few tens of iterations. This is in contrast to, *e.g.*, interior-point methods which can converge to high accuracy in a few tens of iterations, but the per iteration cost is generally much higher. In many practical cases, including much of control, extremely high accuracy is not required and the moderate accuracy provided by the splitting technique is sufficient.

The best known theoretical guarantees of Douglas-Rachford splitting are rather weak: To achieve an error of less than ϵ, D-R splitting requires at most $\mathcal{O}(1/\epsilon^2)$ steps [28, §A]. However, typically, the empirical convergence rate is much faster than this worst-case estimate.

Relaxation. In some cases we can improve the convergence rate of OSC by applying relaxation. Here we replace (x^{k+1}, u^{k+1}) in equations (4.3) and (4.4) with

$$\alpha(x^{k+1}, u^{k+1}) + (1-\alpha)(\tilde{x}^k, \tilde{u}^k)$$

where $\alpha \in (0, 2)$ is a relaxation parameter; when $\alpha < 1$ this is referred to as under-relaxation, when $\alpha > 1$ it is referred to as over-relaxation. This scheme is analyzed in [57], and experiments in [56, 58] show that values of $\alpha \in [1.5, 1.8]$ can improve empirical convergence.

4.3.3 Quadratic control step

The first step in the splitting algorithm can be expressed as a linearly constrained quadratic minimization problem

$$\begin{array}{ll} \text{minimize} & (1/2)w^T E w + f^T w \\ \text{subject to} & Gw = h, \end{array}$$

over variable $w \in \mathbf{R}^{(T+1)(n+m)}$, where

$$w = \begin{bmatrix} x_0 \\ u_0 \\ \vdots \\ x_T \\ u_T \end{bmatrix}, \qquad f = \begin{bmatrix} q_0 - \rho(\tilde{x}_0^k + z_0^k)) \\ r_0 - \rho(\tilde{u}_0^k + y_0^k) \\ \vdots \\ q_T - \rho(\tilde{x}_T^k + z_T^k) \\ r_T - \rho(\tilde{u}_T^k + y_T^k) \end{bmatrix}, \qquad h = \begin{bmatrix} x_{\text{init}} \\ c_0 \\ \vdots \\ c_{T-1} \end{bmatrix},$$

$$E = \begin{bmatrix} Q_0 + \rho I & S_0 & \cdots & 0 & 0 \\ S_0^T & R_0 + \rho I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & Q_T + \rho I & S_T \\ 0 & 0 & \cdots & S_T^T & R_T + \rho I \end{bmatrix},$$

and

$$G = \begin{bmatrix} I & 0 & \cdots & 0 & 0 & 0 & 0 \\ -A_0 & -B_0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \\ 0 & 0 & \cdots & I & 0 & 0 & 0 \\ 0 & 0 & \cdots & -A_{T-1} & -B_{T-1} & I & 0 \end{bmatrix}.$$

The matrix E is block diagonal with $T+1$ blocks of size $(n+m) \times (n+m)$; it is positive definite since $\rho > 0$. The matrix G has full (row) rank (*i.e.*, $(T+1)n$), due to the identity blocks.

Necessary and sufficient optimality conditions are (the KKT equations)

$$\begin{bmatrix} E & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda \end{bmatrix} = \begin{bmatrix} -f \\ h \end{bmatrix}, \tag{4.6}$$

where $\lambda \in \mathbf{R}^{(T+1)n}$ are dual variables associated with the equality constraints. This is a set of $(T+1)(2n+m)$ equations in $(T+1)(2n+m)$ variables; the coefficient matrix (which is called the KKT matrix) is invertible since E is positive definite and G is full rank. We must solve (4.6) multiple times, once per iteration of the splitting algorithm, using the same KKT matrix, but with different values of f.

The traditional approach to solving (4.6) is to eliminate the variable w using

$$w = -E^{-1}G^T\lambda - E^{-1}f, \tag{4.7}$$

which results in the *reduced equation*

$$GE^{-1}G^T\lambda = -h - GE^{-1}f,$$

for the dual variable λ. The matrix $GE^{-1}G^T$ is block tri-diagonal, and can therefore be solved by a Riccati-like recursion in order Tn^3 floating-point operations (flops) [30, §10.4]. The variable w is then reconstructed from (4.7). The overall complexity (including forming $GE^{-1}G^T$ and computing z) is order $T(n+m)^3$ flops. Careful analysis of the Riccati algorithm reveals that if we cache or pre-compute various matrices that arise in solving (4.6), we can solve subsequent instances, with the same KKT matrix but different values of f, in order $T(m+n)^2$ flops.

A modern approach to solving (4.6) is to use a sparse LDL^T decomposition. We factor the KKT matrix as

$$\begin{bmatrix} E & G^T \\ G & 0 \end{bmatrix} = PLDL^TP^T,$$

where P is a permutation matrix, L is lower triangular with diagonal entries one, and D is block diagonal with 1×1 or 2×2 blocks. The permutation matrix P is chosen based on the sparsity pattern of the KKT matrix, in order to yield a factor L with

few nonzeros and a factorization that is stable [33, 49]. We then solve (4.6) using

$$\begin{bmatrix} w \\ \lambda \end{bmatrix} = P\left(L^{-T}\left(D^{-1}\left(L^{-1}\left(P^T\begin{bmatrix} -f \\ h \end{bmatrix}\right)\right)\right)\right).$$

Multiplication by L^{-1} is carried out by forward substitution, and multiplication by L^{-T} is carried out by backward substitution; these operations do not require division. The forward and backward substitution steps require a number of flops on the order of the number of nonzero entries in L. Inverting D amounts to inverting 2×2 matrices; if we pre-compute the inverse of D, which is block diagonal with the same structure as D, the solve step above requires no division; it relies on addition and multiplication. To solve the KKT system many times, with the same KKT matrix but different values of f, we compute and cache P, L, and D^{-1}; subsequently we only carry out the solve step above.

It can be shown that the Riccati recursion is equivalent to the factor-solve method for a particular choice of P [30, §10.4]. When the blocks in E and G are all dense, the factor-solve method has the same complexity: order $T(m+n)^3$ flops to carry out the initial factorization, and order $T(m+n)^2$ flops to carry out subsequent solves. But the general LDL^T method can allow us to exploit additional sparsity within the matrices to reduce the complexity of both the factor and solve steps.

Regularization. To ensure that the factorization always exists and that the factorization algorithm is stable we can regularize the system (for discussion on the stability of factorization algorithms see, *e.g.*, [49,87,172]). Instead of the original KKT matrix, we factor the regularized KKT matrix

$$\begin{bmatrix} E & G^T \\ G & -\epsilon I \end{bmatrix},$$

where $\epsilon > 0$ is a small constant. The regularized matrix is *quasi-definite* [176], which implies that for any permutation P the factorization exists, with D diagonal, and

is numerically stable. It is suggested in [156] that a value as small as $\epsilon = 10^{-8}$ gives stability without seriously altering the problem. It was shown in [57] that the operator splitting method still converges to a solution when the subproblems are not solved exactly, as long as certain conditions on the solutions are met; for example, we can reduce the regularization parameter ϵ as iteration proceeds. The effects of the regularization can also be removed or reduced using iterative refinement [53, §4.11], [122]. In practice we find that a fixed and small regularization parameter value works very well, and that iterative refinement is not needed.

4.3.4 Single period proximal step

The second step of the splitting algorithm involves solving T problems of the form

$$\text{minimize} \quad \psi_t(x_t, u_t) + (\rho/2)\|(x_t, u_t) - (v_t, w_t)\|_2^2, \tag{4.8}$$

over $x_t \in \mathbf{R}^n$ and $u_t \in \mathbf{R}^m$, where $v_t \in \mathbf{R}^n$ and $w_t \in \mathbf{R}^m$ are data. When the non-quadratic cost term ψ_t is separable across the state and control, it splits further into a proximal optimization for the state, and one for the control. In the extreme case when ψ_t is fully separable (down to the individual scalar entries), the problem (4.8) reduces to solving $m + n$ scalar proximal minimization problems.

When ψ_t is a indicator function of a set, the proximal optimization step reduces to a projection of (v_t, w_t) onto the set. For example, suppose the non-quadratic terms ψ_t are used to enforce state and control constraints of the form $\|x_t\|_\infty \leq 1$, $\|u_t\|_\infty \leq 1$, so ψ_t is the indicator function of the unit box (and is separable to the components). Then the proximal minimization has the simple solution

$$x_t = \underset{[-1,1]}{\mathbf{sat}}(v_t), \qquad u_t = \underset{[-1,1]}{\mathbf{sat}}(w_t),$$

where **sat** is the standard saturation function. The examples below will show other cases where we can solve the single period proximal minimizations analytically.

We can always solve the single period proximal problems using a general numerical method, such as an interior-point method. Assuming the dominant cost is solving

for the search step in each interior-point iteration, the cost is order $(n+m)^3$ flops. Therefore, the total cost of finding the proximal step is no more than order $T(n+m)^3$ flops, executed on a single processor; we can reduce this by employing multiple processors and carrying out the proximal optimizations in parallel.

4.3.5 Robust control

Here we mention briefly how our algorithm generalizes to the robust control case. In robust control we typically have uncertainty in the dynamics, initial state, or both. One common technique to deal with this uncertainty is to generate a set of candidate dynamics equations and initial states and to find a single sequence of control inputs that minimizes the expected costs over these possibilities.

In other words our system has N possible dynamics equations

$$x_t^{(i)} = A_t^{(i)} x_t^{(i)} + B_t^{(i)} u_t + c_t^{(i)}, \quad t = 0, \ldots, T-1,$$

from initial state $x_0^{(i)}$, for $i = 1, \ldots, N$. We assume that the probability of the system being in state $x_0^{(i)}$ and begin subject to the ith set of dynamics equations is given by p_i. These candidates could have been generated, for example, by sampling from a distribution over the parameters and initial states. The goal is to find a single sequence of inputs, u_t for $t = 0, \ldots, T$, that minimizes the expected cost.

This problem fits our problem description if we make the following substitutions: The state is constructed by stacking the candidate states, *i.e.*,

$$x_t = \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(N)} \end{bmatrix}$$

and the dynamics of our stacked system is

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

where

$$A_t = \begin{bmatrix} A_t^{(1)} & 0 & \cdots & 0 \\ 0 & A_t^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_t^{(N)} \end{bmatrix}, \quad B_t = \begin{bmatrix} B_t^{(1)} \\ B_t^{(2)} \\ \vdots \\ B_t^{(N)} \end{bmatrix}, \quad c_t = \begin{bmatrix} c_t^{(1)} \\ c_t^{(2)} \\ \vdots \\ c_t^{(N)} \end{bmatrix}.$$

The stage cost at time t is written

$$\sum_{i=1}^{N} p_i \left(\phi_t(x_t^{(i)}, u_t) + \psi_t(x_t^{(i)}, u_t) \right),$$

which is a sum of convex quadratics and convex non-quadratic terms and so it matches our framework.

Note that this problem has additional structure that can be exploited. In particular, the KKT matrix in the quadratic control step has significant extra sparsity from the dynamics equations. This will be exploited by the permuted LDL^T algorithm. Similarly for the proximal step: If the ψ_t terms in the objective are separable into terms involving only the state and only the input, then the proximal step problem will decompose over the N state trajectories (in addition to decomposing across time periods), which can then be solved in parallel.

4.4 Examples

In this section we illustrate OSC on four examples, and report on numerical results. For each example we explain the details of the splitting technique, explain how we generated the data for the problem instances, and compare the OSC solution time with CVX [77, 78], a parser-solver that uses SDPT3 [170]. When we report CVX times we report the solve time only, and not the time required to parse and transform the problem. For each example we present three instances with different problem dimensions.

All computation, with the exception of the results presented in §4.4.5, was carried

out on a system with a quad-core Intel Xeon processor, with clock speed 3.4GHz and 16Gb of RAM, running Linux. All examples were implemented in C, and all but one example were implemented in single thread. Note that SDPT3 is able to exploit the structure inherent in the problem and is automatically multi-threaded over the 4 cores using hyperthreads.

When running OSC we report the time and number of iterations required to reach an accuracy corresponding to ϵ^{abs} and ϵ^{rel} in (4.5) both set to 10^{-3}. With these tolerances the objective value obtained by $(\tilde{x}, \tilde{u})$ at termination was never more than 1% suboptimal, as judged by the objective value obtained by CVX. (Note that $(\tilde{x}, \tilde{u})$ at termination may very slightly violate the equality constraints in (4.1), but are guaranteed to satisfy any constraints embedded in the non-quadratic cost function terms).

In each case we give the time required for the factorization of the KKT matrix, cold start solve, and warm start solve. We break down the cost of each iteration into solving the linear quadratic system and solving the single period proximal problems. The results for each example are presented in a table; all reported times are in milliseconds to an accuracy of 0.1 ms. In certain cases the reported time is 0.0 ms, which indicates that the time required for this step was less than 0.05 ms. In the case where we parallelized the single period proximal problems across multiple cores we mention the single thread and multi-thread solve times.

When forming the KKT system we added regularization of $\epsilon = 10^{-6}$ to all examples to ensure existence of the LDL^T factorization. We found that iterative refinement was not needed in any of the examples. To solve the KKT system we used the sparse LDL and AMD packages written by Tim Davis et al. [5, 45, 46].

For the cold start case we initialize variables $(\tilde{x}^0, \tilde{u}^0)$ and (z^0, y^0) to zero for all examples and solve the problem. The time required to solve the problem under a cold start can vary by a small amount, so we report the average solve time over 100 runs (the number of iterations is always the same). The worst-case times were never more than 5% more than the average times.

For warm start we initialize variables $(\tilde{x}^0, \tilde{u}^0)$ and (z^0, y^0) to those returned at the termination of the algorithm with unperturbed data. When warm starting we

consider perturbations to the initial state, *i.e.*, we replace x_{init} with some $\hat{x}_{\text{init}}$; this type of perturbation is common in, *e.g.*, MPC. Since no other data are perturbed the matrix does not have to be re-factorized; this corresponds to the warm start constant quadratic usage scenario. For each example we run 100 warm starts and report *average* numbers. Across all examples we found that the *worst-case* solve time out of the 100 warm starts was no more than 20% longer than the average time.

4.4.1 Box constrained quadratic optimal control

We consider a quadratic optimal control problem with box constraints on the input,

$$\begin{array}{ll} \text{minimize} & (1/2)\sum_{t=0}^{T}(x_t^T Q_t x_t + u_t^T R_t u_t) \\ \text{subject to} & x_{t+1} = A_t x_t + B_t u_t, \quad t = 0, \ldots, T-1 \\ & x_0 = x_{\text{init}} \\ & \|u_t\|_\infty \leq 1, \end{array}$$

where $Q_t \succeq 0$ and $R_t \succ 0$. We encode the constraints on the action by setting

$$\psi_t(x_t, u_t) = I_{\|u_t\|_\infty \leq 1},$$

so proximal minimization is just saturation.

Numerical instances. For simplicity we consider the time-invariant case, *i.e.*, $A_t = A$, $B_t = B$, $Q_t = Q$ and $R_t = R$ for all t. The data were all generated randomly; the matrix A was scaled to be marginally stable, *i.e.*, have a largest eigenvalue magnitude of one. The initial state x_{init} was scaled so that at least one input was saturated for at least the first 2/3 of the horizon. For all instances below we chose $\rho = 50$ and relaxation parameter $\alpha = 1.8$. In this example the time required to take the proximal step was negligible (less than 0.05 ms for the largest instance), as such there was no benefit to parallelization and all computations were performed serially on a single core. For the warm start we perturbed each entry of the initial vector by a random proportion sampled from a uniform distribution on $[-0.1, 0.1]$.

	small	**medium**	**large**
state dimension n	5	20	50
input dimension m	2	5	20
horizon length T	10	20	30
total variables	77	525	2170
CVX solve time	400 ms	500 ms	3400 ms
fast MPC solve time	1.5 ms	14.2 ms	2710 ms
factorization time	0.1 ms	1.3 ms	16.8 ms
KKT solve time	0.0 ms	0.1 ms	0.9 ms
cold start OSC iterations	92	46	68
cold start OSC solve time	0.4 ms	4.4 ms	60.5 ms
warm start OSC iterations	72.6	35.1	39.5
warm start OSC solve time	0.3 ms	3.4 ms	35.2 ms

Table 4.1: OSC results for the box constrained optimal control example.

For this example we can compare the performance of OSC to fast MPC [181]. CVXGEN [122] can solve the smallest problem instance in about 0.1 ms, but the current version of CVXGEN cannot scale to the larger problems. The results are summarized in table 4.1.

4.4.2 Multi-period portfolio optimization

In this example we consider the problem of managing a portfolio of n assets over a finite time horizon, as described in [27]. In this case our state $x_t \in \mathbf{R}^n$ is the dollar amount invested in each asset; the control $u_t \in \mathbf{R}^n$ is the amount of each asset we buy (or sell, when negative) at time t. The dynamics is given by

$$x_{t+1} = \mathbf{diag}(r_t)(x_t + u_t), \quad t = 0, \ldots, T-1,$$

where $r_t > 0$ is the vector of (estimated) returns in period t. This has our form, with $A_t = B_t = \mathbf{diag}(r_t)$, and $c_t = 0$. The stage cost has form

$$\mathbf{1}^T u_t + \kappa_t^T |u_t| + u_t^T \mathbf{diag}(s_t) u_t + \lambda_t (x_t + u_t)^T \Sigma_t (x_t + u_t),$$

where $\kappa_t \geq 0$, $s_t \geq 0$, $\Sigma_t \succ 0$, and $\lambda_t > 0$ are data; the absolute value is element-wise. These terms are, respectively, gross cash in for sales and purchases, a bid-ask spread transaction fee, a quadratic price impact transaction cost, and a quadratic risk penalty. We are subject to the constraints that $x_{\text{init}} = x_T + u_T = 0$, *i.e.*, the trading starts and ends with no holdings, and a long-only constraint, *i.e.*, $x_t + u_t \geq 0$ for all t.

We split the stage cost as

$$\phi_t(x_t, u_t) = \mathbf{1}^T u + u_t^T \mathbf{diag}(s_t) u_t + \lambda_t (x_t + u_t)^T \Sigma_t (x_t + u_t)$$

$t = 0, \ldots, T$,

$$\psi_t(x_t, u_t) = \kappa_t^T |u_t| + I_{x_t + u_t \geq 0}.$$

$t = 0, \ldots, T-1$ and

$$\psi_T(x_T, u_T) = \kappa_T^T |u_T| + I_{x_T + u_T = 0}.$$

In this example, ψ_t is not state-control separable, but it is separable across assets; that is, it is a sum of terms involving $(x_t)_i$ and $(u_t)_i$. To evaluate the proximal operator for $t < T$, we need to solve a set of problems of the form (using the subscript to denote the component, not time period)

$$\begin{array}{ll} \text{minimize} & \kappa_i |u|_i + (\rho/2)\left((x_i - v_i)^2 + (u_i - w_i)^2\right) \\ \text{subject to} & x_i + u_i \geq 0, \end{array}$$

with (scalar) variables x_i and u_i. The solution relies on the proximal operator for the absolute value, which is given by soft-thresholding:

$$S_\gamma(z) = \operatorname*{argmin}_y \left(\gamma |y| + (1/2)(y - z)^2\right) = z(1 - \gamma/|z|)_+.$$

	small	**medium**	**large**
number of assets n	10	30	50
horizon length T	30	60	100
total variables	620	3660	10100
CVX solve time	800 ms	2100 ms	10750 ms
factorization time	0.7 ms	13.3 ms	73.6 ms
KKT solve time	0.1 ms	0.7 ms	3.2 ms
cold start OSC iterations	27	41	53
cold start OSC solve time	1.5 ms	30.8 ms	177.7 ms
warm start OSC iterations	5.1	5.9	4.8
warm start OSC solve time	0.3 ms	4.4 ms	16.1 ms

Table 4.2: OSC results for the portfolio optimization problem.

The solution to the above problem is as follows: If $v_i + S_{\kappa_i/\rho}(w_i) \geq 0$ then $x_i = v_i$ and $u_i = S_{\kappa_i/\rho}(w_i)$, otherwise $u_i = S_{\kappa_i/2\rho}((w_i - v_i)/2)$ and $x_i = -u_i$. For $t = T$ the solution is simply $u_i = S_{\kappa_i/2\rho}((w_i - v_i)/2)$ and $x_i = -u_i$.

Numerical instances. For simplicity we consider the time-invariant case, that is the data are constant over time. For more details about how the data were generated see [27]. The correlations between assets were selected to be between -0.4 and 0.6. The expected returns were chosen so that all the assets made returns from 0.01% to 3% per period. For all instances below we chose $\rho = 0.1$ and relaxation parameter $\alpha = 1.8$. Again in this case the time required to calculate the proximal step was negligible (less than 0.05 ms for the largest instance) and thus all computation was performed on a single core. For the warm start we replaced the initial portfolio with a random portfolio sampled from $\mathcal{N}(0, I)$; this produced initial portfolios with a norm of roughly 20% the norm of the maximum position taken in the unperturbed solution. The results are summarized in table 4.2.

4.4.3 Robust state estimation

In this example we use noisy state measurements to estimate the state sequence for a dynamical system acted on by a process noise. The system evolves according to

$$x_{t+1} = A_t x_t + B_t u_t, \quad t = 0, \ldots, T-1,$$

where here u_t is a process noise. Our measurements are

$$y_t = C_t x_t + v_t, \quad t = 0, \ldots, T,$$

where $v_t \in \mathbf{R}^p$ is a measurement noise. We assume the process and measurement noises are independent. We know the initial state of the system, x_{init}.

The maximum-likelihood estimate of the state sequence is found by solving the problem

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T} h_t(u_t) + g_t(y_t - C_t x_t) \\ \text{subject to} & x_{t+1} = A_t x_t + B_t u_t, \quad t = 0, \ldots, T-1 \\ & x_0 = x_{\text{init}}. \end{array}$$

where h_t is the negative log density of u_t, and g_t is the negative log density of v_t. Assuming these are convex functions (*i.e.*, u_t and v_t have log-concave distributions), this problem has our form. The stage cost is state-control separable. If these distributions are Gaussian, this problem is quadratic.

As a more specific example, we take v_t to have a Gaussian distribution, so g_t is convex quadratic, but we take h_t to be the circular Huber function with half-width M, *i.e.*,

$$h_t(u_t) = \begin{cases} (1/2)\|u_t\|_2^2 & \|u_t\|_2 \leq M \\ M(\|u_t\|_2 - M/2) & \|u_t\|_2 > M. \end{cases}$$

This corresponds to a process noise that is Gaussian for $\|u_t\|_2 \leq M$, with fat (exponential) tails for $\|u_t\|_2 > M$.

The proximal operator for the circular Huber function has the simple expression

$$\underset{u}{\text{argmin}}(h_t(u) + (\rho/2)\|u - v\|_2^2) = \left(1 - \min\left(\frac{1}{1+\rho}, \frac{M}{\rho\|v\|_2}\right)\right) v.$$

Numerical instances. We consider the case where $A_t = A$, $B_t = I$ and $C_t = C$ for all t. All data were generated randomly, and we scaled A so that the system was marginally stable. The measurement noise was sampled from a standard Normal distribution, *i.e.*, $v_t \sim \mathcal{N}(0, I)$. The process noise was generated as follows:

$$u_t \sim \begin{cases} \mathcal{N}(0, I) & \text{with probability } 0.75 \\ \mathcal{N}(0, 10I) & \text{with probability } 0.25, \end{cases}$$

so 25% of the noise samples were large outliers. We took the half-width of the Huber function to be $M = 1$. For all instances we chose $\rho = 0.1$ and relaxation parameter $\alpha = 1.8$. Again the time required to calculate the proximal step was negligible (less than 0.05 ms for the largest instance) and thus all computation was single threaded. For the warm start we perturbed each entry of the initial vector by a random proportion sampled from a uniform distribution on $[-0.1, 0.1]$. The results are summarized in table 4.3.

4.4.4 Supply chain management

We consider a single commodity supply chain management problem. The supply chain network is described by a directed graph, with n nodes corresponding to warehouses or storage locations, and the m edges corresponding to shipment links between warehouses, as well as from sources and to sinks (which supply or consume the good). The state $x_t \in \mathbf{R}_+^n$ is the amount of the commodity stored in each of the n warehouses, and the control $u_t \in \mathbf{R}_+^m$ is the amount of the commodity shipped along each of the m edges. The dynamics is

$$x_{t+1} = x_t + (B^+ - B^-)u_t,$$

	small	**medium**	**large**
state dimension n	10	30	50
input dimension m	10	30	50
measurement dimension p	5	10	20
horizon length T	30	60	100
total variables	620	3660	10100
CVX solve time	700 ms	1900 ms	8000 ms
factorization time	0.5 ms	8.7 ms	48.3 ms
KKT solve time	0.0 ms	0.6 ms	2.5 ms
cold start OSC iterations	21	25	29
cold start OSC solve time	0.9 ms	14.9 ms	76.7 ms
warm start OSC iterations	7.5	8.0	7.7
warm start OSC solve time	0.3 ms	4.8 ms	20.5 ms

Table 4.3: OSC results for the robust state estimation problem.

where $B^+_{ij} = 1$ if edge j enters node i, $B^-_{ij} = 1$ if edge j leaves node i. Each column of the matrix $(B^+ - B^-)$ corresponding to an edge between warehouses has exactly one entry 1 and one entry -1, and all other entries are zero. If the edge connects to a source then that column has 1 and no other entries, if it connects to a sink then that column has just a -1.

The warehouses have a capacity $C \in \mathbf{R}^n_+$, so $0 \leq x_t \leq C$. We cannot ship out of any warehouse more than is on hand, so we have the constraint $B^- u_t \leq x_t$, in addition to $0 \leq u_t \leq U$, where U gives the maximum possible shipping levels. (For edges that come from sources, we interpret the corresponding entry of U as a maximum possible supply rate; for edges that go to sinks, we interpret the corresponding entry of U as a maximum demand.)

The stage cost consists of quadratic storage charges, linear transportation charges, the linear cost of acquiring the commodity, and the linear revenue we earn from sinks:

$$q_t^T x + \tilde{q}_t^T x^2 + r_t^T u_t$$

where $q_t > 0$ and $\tilde{q}_t > 0$ give the storage cost, and the entries of the vector r_t give the transportation cost, (for edges between warehouses), the cost of acquisition (for sources), and the revenue (for sinks).

We split the stage cost as

$$\phi_t(x_t, u_t) = q_t^T x_t + \tilde{q}_t^T x_t^2 + r_t^T u_t,$$

and ψ_t the indicator function of the constraints,

$$B^- u_t \leq x_t \leq C, \qquad 0 \leq u_t \leq U.$$

The proximal step can be found by first noting that the problem can be decomposed across the n nodes, including all outgoing edges from the node. (Edges coming in from sources are readily handled separately, since their only constraint is that they lie in the interval.) For each node we solve a problem of the form

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^k (u_j - w_j)^2 + (x - v)^2 \\ \text{subject to} & \sum_{j=1}^k u_j \leq x \leq C \\ & 0 \leq u_j \leq U_j, \quad j = 1, \ldots, k, \end{array}$$

over $x \in \mathbf{R}$ and $u \in \mathbf{R}^k$, where k is the out-degree of the node. Introducing a Lagrange multiplier λ for the constraint $\sum_{j=1}^k u_j \leq x$, we end up with

$$u_i = \underset{[0,U]}{\mathbf{sat}}(w_i - \lambda), \qquad x = \underset{[0,C]}{\mathbf{sat}}(v + \lambda).$$

The solution is found as the smallest value of $\lambda \geq 0$ for which $\sum_{j=1}^k u_j \leq x$ holds. This value can be found using, for example, bisection.

Numerical instances. To generate the graph we distributed warehouses randomly on a unit square, and connected warehouses together that were closer than a threshold, selected so that the graph was fully connected. The cost to traverse an edge was proportional to the distance between the warehouses on the square, plus a small

	small	**medium**	**large**
warehouses n	10	20	40
edges m	25	118	380
horizon length T	20	20	20
total variables	735	2898	8820
CVX solve time	500 ms	1200 ms	3300 ms
factorization time	0.3 ms	1.3 ms	4.7 ms
KKT solve time	0.0 ms	0.1 ms	0.3 ms
single-thread prox step time	0.1 ms	0.4 ms	1.3 ms
multi-thread prox step time	0.0 ms	0.1 ms	0.4 ms
cold start OSC iterations	82	77	116
cold start OSC solve time	4.6 ms	19.1 ms	88.1 ms
warm start OSC iterations	21.9	31.0	24.2
warm start OSC solve time	1.2 ms	7.5 ms	18.5 ms

Table 4.4: OSC results for the supply chain example.

perturbation. We created dummy warehouses to act as sources in one corner of the square and others to act as sinks in the opposite corner, and connected them to nearby warehouses. This ensured that to get from a source to a sink required traversing through several intermediary warehouses. The warehouse capacities were all chosen to be 2, the edge limits were all chosen to be 1. The revenues from sinks were chosen to have a mean of -15 the cost of acquisition from sources were chosen to have a mean of 5.

For all instances we chose $\rho = 2.5$ and relaxation parameter $\alpha = 1.8$. In this case the time required to calculate the proximal step was not negligible, so we report times for both single-threaded and multi-threaded prox step calculation. For the warm start we perturbed each entry of the initial vector by a random proportion sampled from a uniform distribution on $[-0.1, 0.1]$. The results are summarized in table 4.4.

	small	**medium**	**large**
box constrained quadratic	8.4 ms	99.9 ms	1750.3 ms
portfolio optimization	29.4 ms	879.1 ms	5036.0 ms
robust state estimation	17.7 ms	430.3 ms	2244.5 ms
supply chain management	125.9 ms	595.9 ms	2817.7 ms

Table 4.5: Embedded processor timing results, cold start.

4.4.5 Embedded results

In this section we present timing results for an embedded processor running the same examples as above. For these experiments computation was carried out on a Raspberry Pi, which is a credit-card sized, single-board computer costing about $25 and capable of running Linux. The Raspberry Pi has an ARM1176 core with clock speed 1GHz, 256Mb of RAM and a floating point unit (FPU). It typically draws less than 2W of power. With the exception of the matrix factorization all computation was performed in single-precision. Tables 4.5 and 4.6 summarize the timing results for the cold start and warm start cases respectively. The embedded processor requires roughly an order of magnitude more time to solve the problems than the quad-core Intel Xeon machine, with larger problems requiring greater factors. However, even examples with more than 10000 variables are solved in just a few seconds, and in all cases the embedded processor solved the problem quicker than CVX on the quad-core machine. Typically, embedded applications do not require very high accuracy solutions, and so, in practice, computation times could be reduced by decreasing the termination tolerances.

4.5 Conclusions

In this chapter we demonstrated an operator splitting technique that can solve optimal control problems rapidly and robustly. Small problems are solved in a few milliseconds or less; example problems with on the order of 10000 variables are solved in tens of milliseconds. The speedup of OSC over other solvers, including fast and custom

	small	**medium**	**large**
box constrained quadratic	5.5 ms	76.4 ms	1019.9 ms
portfolio optimization	5.2 ms	129.1 ms	458.8 ms
robust state estimation	6.3 ms	138.6 ms	602.5 ms
supply chain management	34.0 ms	245.3 ms	592.8 ms

Table 4.6: Embedded processor timing results, warm start.

solvers specifically for the particular problem type, range from tens to thousands. When the dynamics data do not change, the splitting method yields a division-free algorithm, which can be implemented in fixed-point arithmetic.

We close with some comments about the applicability of this technique to the case when the stage-cost functions ψ_t are non-convex. This situation is not uncommon in practice. One common example is when a device (say, an internal combustion engine or turbine) can be operated over some range, and in addition switched on or off; an extreme example is where the control u_t is restricted to a finite set of allowed values. In this case the optimal control problem (4.1) is non-convex, and generally hard to solve (exactly). For many of these cases, the proximal step can be carried out efficiently (even though the problem that must be solved is non-convex), so the operator splitting technique described in this chapter can be applied. In this case, though, we have no reason to believe that the OSC method will converge, let alone to a solution of the problem. On the other hand, when OSC is used on non-convex problems, it is observed in practice that it typically does converge, to a quite good solution. For a more detailed discussion of Douglas-Rachford splitting applied to non-convex problems see [28, §9].

Part IV

Example

Chapter 5

Multi-Period Investment

In this chapter we apply some of the techniques from the previous chapters to a particular example, and examine performance and timing results. Our example is that of dynamic trading of a portfolio of assets in discrete periods over a finite time horizon, with arbitrary time-varying distribution of asset returns. The goal is to maximize the total expected revenue from the portfolio, while respecting constraints on the portfolio such as a required terminal portfolio and leverage and risk limits. The revenue takes into account the gross cash generated in trades, transaction costs, and costs associated with the positions, such as fees for holding short positions. Our model has the form of a stochastic control problem with linear dynamics and convex cost function and constraints. While this problem can be tractably solved in several special cases, such as when all costs are convex quadratic, or when there are no transaction costs, our focus is on the more general case, with nonquadratic cost terms and transaction costs.

We show how to use linear matrix inequality techniques and semidefinite programming to produce a quadratic bound on the value function, which in turn gives a bound on the optimal performance. This performance bound can be used to judge the performance obtained by any suboptimal policy. As a by-product of the performance bound computation, we obtain an approximate dynamic programming policy that requires the solution of a convex optimization problem, often a quadratic program,

to determine the trades to carry out in each step. While we have no theoretical guarantee that the performance of our suboptimal policy is always near the performance bound (which would imply that it is nearly optimal) we observe that in numerical examples the two values are typically close.

5.1 Introduction

5.1.1 Overview

In this chapter we formulate the discrete-time finite horizon time-varying multi-period investment problem as a stochastic control problem. By using state variables that track the *value* of the assets, instead of more traditional choices of states such as the number of shares or the fraction of total value, the stochastic control problem has *linear* (but random) dynamics. Assuming that the costs and constraints are convex, we arrive at a linear convex stochastic control problem.

This problem can be effectively solved in two broad cases. When there are no transaction costs, the multi-period investment problem can be reduced to solving a set of standard single-period investment problems; the optimal policy in this case is to simply rebalance the portfolio to a pre-computed optimal portfolio in each step. Another case in which the problem can be effectively solved is when the costs are quadratic and the only constraints are linear equality constraints. In this case standard dynamic programming (DP) techniques can be used to compute the optimal trading policies, which are affine functions of the current portfolio. We describe these special cases in more detail in §5.3.3 and §5.3.2. The problem is also tractable when the number of assets is very small, say two or three, in which case brute force dynamic programming can be used to compute an optimal policy.

Most problems of interest, however, include significant transaction costs, or include terms that are not well approximated by quadratic functions. In these cases, the optimal investment policy cannot be tractably computed. In such situations, several approaches can be used to find suboptimal policies, including approximate dynamic programming (ADP) and model predictive control (MPC). The performance of any

suboptimal policy can be evaluated using Monte Carlo analysis, by simulation over many return trajectories. An obvious practical (and theoretical) question is, how suboptimal is the policy? In this chapter we address this question.

Using linear matrix inequality (LMI) techniques widely used in control system analysis and design [26,59,146], we construct a (numerical) bound on the best performance that can be attained, for a given problem. The method requires the construction and solution of a semidefinite program (SDP), a convex optimization problem involving matrix inequalities. We can compare the bound on performance with the performance attained by any suboptimal policy; when they are close, we conclude that the policy is approximately optimal (and that the performance bound is nearly tight). Even when the performance bound and suboptimal policy performance are not close, we at least have a bound on how suboptimal our suboptimal policy can be.

The performance bound computation yields a quadratic approximation (in fact, underestimator) of the value functions for the stochastic control problem. These quadratic value function approximations can be used in an ADP policy, or as the terminal cost in an MPC policy. While we have no a priori guarantee that the gap between the performance bound and the performance of the ADP policy will always be small, simulations show that the ADP and MPC policies achieve performance that is often nearly optimal.

Our methods for computing the performance bound, as well as implementing the ADP and MPC suboptimal policies, rely on (numerically) solving convex optimization problems, for which there are efficient and reliable algorithms available [30, 133, 134, 142,188]. The performance bound computation requires solving SDPs [30,175], which can be done using modern interior-point cone solvers such as SeDuMi or SDPT3 [164, 169, 173]. Parser-solvers such as CVX or YALMIP [78, 114] allow the user to specify the SDPs in a natural high-level mathematical description form, greatly reducing the time required to form and solve the SDPs. The SDPs that we solve involve T matrices of size $n \times n$, where n is the number of assets, and T is the trading period horizon. These SDPs can be challenging to solve (depending on n and T, of course), using generic methods; but this computation is done once, off-line, before trading begins.

Evaluating the ADP suboptimal policy in each period (*i.e.*, determining the trades

to execute) requires solving a small and structured convex optimization problem with (on the order of) n scalar variables. Solving these problems using generic solvers might take seconds, or even minutes, depending on the problem size and types of constraints and objective terms. But recent advances have shown that if the solver is customized for the particular problem family, orders of magnitude speed up is possible [120–123, 182]. This means that the ADP trading policies we design can be executed at time scales measured in milliseconds or microseconds for modest size problems (say, tens of assets), even with complex constraints. In addition, the trading policies we design can be tested and verified via Monte Carlo simulation very efficiently. For example, the simulation of the numerical examples of the ADP policies reported in this chapter required the solution of around 50 million quadratic programs (QPs). These were solved in a few hours on a desktop computer using custom solvers generated by CVXGEN, a code generator for embedded convex optimization [122].

Evaluating the MPC policy also requires the solution of a structured convex optimization problem, with (on the order of) nT variables. If a custom solver is used, the computational effort required to solve this optimization problem is approximately T times the effort required to evaluate the ADP policy. One major advantage of MPC is that it does not require any pre-computation; to implement the ADP policy, we must first solve a large SDP to find the approximate value functions. As a result, MPC can directly incorporate real-time signals, such as changes in future return statistics.

5.1.2 Prior and related work

Portfolio optimization has been studied and used for more than 60 years. In this section our goal is to give a brief overview of some of the important research in this area, focussing on work related to our approach. Readers interested in a broader overview of the applications of stochastic control and optimization to economics and finance should refer to, *e.g.*, [1, 55, 94, 141, 166, 191].

Single-period portfolio optimization. Portfolio optimization was first introduced by Markowitz in 1952 [119]. Markowitz formulated a single period portfolio investment problem as a quadratic optimization problem with an objective that

trades off expected return and variance. Since this first work, many papers have extended the single period portfolio optimization framework. For example, Goldsmith [74] is one of the first papers to include an analysis of the effect of transaction costs on portfolio selection. Modern convex optimization methods, such as second-order cone programming (SOCP), are applied to portfolio problems with transaction costs in [112,113]. Convex optimization methods have also been used to handle more sophisticated measures of risk, such as conditional value at risk (CVaR) [103, 152].

Dynamic multi-period portfolio optimization. Early attempts to extend the return-variance tradeoff to multi-period portfolio optimization include [129,168]. One of the first works on multi-period portfolio investment in a dynamic programming framework is by Merton [126]. In this seminal paper, the author considers a problem with one risky asset and one risk-free asset; at each continuous time instant, the investor chooses what proportion of his wealth to invest and what to consume, seeking to maximize the total utility of the wealth consumed over a finite time horizon. When there are no constraints or transaction costs, and under some additional assumptions on the investor utility function, Merton derived a simple closed-form expression for the optimal policy. In a companion paper [155], Samuelson derived the discrete-time analog of Merton's approach.

Constantinides [41] extended Samuelson's discrete-time formulation to problems with proportional transaction costs. In his paper, Constantinides demonstrated the presence of a convex 'no-trade cone'. When the portfolio is within the cone the optimal policy is not to trade; outside the cone, the optimal policy is to trade to the boundary of the cone. (We will see that the policies we derive in this chapter have similar properties.) Davis and Norman [44] and Dumas and Lucian [54] derived similar results for the continuous-time formulation. In [43], the authors consider a specific multi-period portfolio problem in continuous time, where they derive a formula for the minimum wealth needed to hedge an arbitrary contingent claim with proportional transaction costs. More recent work includes [34,35,171]; in these the authors develop affine recourse policies for discrete time portfolio optimization.

Log-optimal investment. A different formulation for the multi-period problem was developed by Kelly [99], where it was shown that a log-optimal investment strategy maximizes the long-term growth rate of cumulative wealth in horse-race markets. This was extended in [31] to general asset returns and further extended to include all frictionless stationary ergodic markets in [3] and [42]. More recently, Iyengar [90] extended these problems to include proportional transaction costs.

Optimal execution. An important special case of the multi-period portfolio problem is the optimal execution problem, where we seek to execute a large block of trades while incurring as small a cost as possible. In [20] Bertsimas and Lo model the practical problem of price impact, where the act of trading affects the asset prices, and derive an optimal trading policy using dynamic programming methods. Almgren and Chriss [4] address the optimal execution problem, including volatility of revenue. They show that the optimal policy can be obtained with additional restrictions on the price dynamics.

Linear-quadratic multi-period portfolio optimization. Optimal policies for unconstrained linear-quadratic portfolio problems have been derived for continuous-time formulations by Zhou and Li [190], where the authors solve a continuous-time Riccati equation to compute the value function. In [107] this was extended to include a long-only constraint. Skaf and Boyd [161], and Gârleanu and Pederson [65], point out that the multi-period portfolio optimization problem with linear dynamics and convex quadratic objective can be solved exactly. For problems with more complex objective terms, such as proportional transaction costs, Skaf and Boyd use the value functions for an associated quadratic problem as the approximate value functions in an ADP policy. In [86] the authors formulate a multi-period portfolio problem as a linear stochastic control problem, and propose an MPC policy.

Performance bounds. In problems for which an optimal policy can be found, the optimal performance serves as a (tight) bound on performance. The present chapter focuses on developing a numerical bound on the optimal performance for problems

for which the optimal policy cannot be found. Brown and Smith, in [32], compute a bound on optimal performance and derive a heuristic policy that achieves a performance close to the bound. In their work, the bound is given by the performance of an investor with perfect information about future returns, plus a penalty for clairvoyance. In [82] the authors construct an upper bound on a continuous time portfolio utility maximization problem with position limits. They do this by solving an unconstrained 'fictitious problem' which provides an upper bound on the value function of the original problem. In [128], the authors describe a class of linear rebalancing policies for the discrete-time portfolio optimization problem. They develop several bounds, including a bound based on a clairvoyant investor and a bound obtained by solving an unconstrained quadratic problem. Desai et al., in [51], develop a bound for an optimal stopping problem, which is useful in a financial context for the pricing of American or Bermudan derivatives amongst other applications. The bound is derived from a dual characterization of optimal stopping problems as optimization problems over the space of martingales.

5.1.3 Outline

We structure this chapter as follows. In §5.2 we formulate a general multi-period investment problem as a linear convex stochastic control problem, using somewhat nontraditional state variables, and give examples of (convex) stage cost terms and portfolio constraints that arise in practical investment problems, as well as mentioning some nonconvex terms and constraints that do not fit our model. In §5.3 we review the dynamic programming solution of the stochastic control problem, including the special case when the stage costs are convex quadratic. In §5.4 we give our method for finding a performance bound in outline form; lengthy derivations for this section are pushed to appendices A.2–A.1. We describe MPC in §5.6. In §5.7 we report numerical results for several examples, using both ADP and MPC trading policies.

5.2 Stochastic control formulation

5.2.1 Model

Portfolio. We manage a portfolio of n assets over a finite time horizon, which is divided into discrete time periods $t = 0, 1, \ldots, T$, which need not be uniformly spaced in real time. We let $x_t \in \mathbf{R}^n$ denote the portfolio (or vector of positions) at time t, where $(x_t)_i$ is the *dollar value* of asset i at the beginning of time period t, with $(x_t)_i < 0$ meaning a short position in asset i. For discussion on the relative merits of tracking the value of the assets rather than the number of units of each asset see, *e.g.*, [95–97, 158]. The dollar value is computed using the current *reference price* for each asset, which can differ from the current prices in an order book, such as the bid or ask price; a reasonable choice is the average of the bid and ask prices. We assume that the initial portfolio, x_0, is given. One of the assets can be a risk-free or cash account, as in a traditional formulation, but since we will be separately tracking cash that enters and leaves the portfolio, this is not needed.

Trading. We can buy and sell assets at the beginning of each time period. We let $u_t \in \mathbf{R}^n$ denote the dollar values of the trades: $(u_t)_i > 0$ means we buy asset i at the beginning of time period t and $(u_t)_i < 0$ means we sell asset i at the beginning of time period t. We define the post-trade portfolio as

$$x_t^+ = x_t + u_t, \quad t = 0, 1, \ldots, T,$$

which is the portfolio in time period t immediately after trading. For future reference, we note that $\mathbf{1}^T x_t$ is the total value of the portfolio (before trading), $\mathbf{1}^T u_t$ is the total cash we put into the portfolio to carry out the trades (not including transaction costs, discussed below), and $\mathbf{1}^T x_t^+$ is the total value of the post-trade portfolio, where $\mathbf{1}$ denotes the vector with all entries one.

Investment. The post-trade portfolio is held until the beginning of the next time period. The portfolio at the next time period is given by

$$x_{t+1} = R_{t+1}x_t^+, \quad t = 0, 1, \ldots, T-1, \tag{5.1}$$

where $R_{t+1} = \mathbf{diag}(r_{t+1}) \in \mathbf{R}^{n\times n}$ is the diagonal matrix of asset returns, and r_{t+1} is the vector of asset returns, from period t to period $t+1$. The return r_{t+1} is of course not known at the beginning of period t, so the choice of trades u_t must be made without knowledge of r_{t+1}. The dynamics (5.1) is *linear* (but unknown at time t); this is not the case when other state variables are chosen, such as the number of shares of each asset, or the fractional value of each asset in the portfolio.

Return model. We assume that r_t are independent random (vector) variables, with known distributions, with mean and covariance

$$\mathbf{E}\, r_t = \overline{r}_t, \qquad \mathbf{E}(r_t - \overline{r}_t)(r_t - \overline{r}_t)^T = \Sigma_t, \quad t = 1, \ldots, T.$$

Our assumption of independence of returns in different periods means that our formulation does not handle phenomena such as momentum or mean-reversion, or more sophisticated models for variance such as GARCH. The returns are typically nonnegative, but we will not use this assumption in the sequel. Time variation of the return distribution can be used to model effects such as predictable time variation in volatility, or non-uniformly spaced time periods. We note for future use that the total value of the portfolio after the investment period (or equivalently, at the beginning of the next time period) is given by $\mathbf{1}^T x_{t+1} = r_{t+1}^T x_t^+$.

Trading policy. The trades are determined in each period by the trading policy $\phi_t : \mathbf{R}^n \to \mathbf{R}^n$:

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

Thus, at time t, the trades u_t depend only on the portfolio positions x_t. (For the problem we consider, it can be shown that there is no advantage to including past

state values, or past returns, in the trading policy; see, *e.g.*, [13, 16, 18].) For fixed $\phi_0, \ldots, \phi_T$, the portfolio and trades $x_0, \ldots, x_T$ and $u_0, \ldots, u_T$ become random variables. Since x_0 is given, we can assume that ϕ_0 is a constant function.

Cash in. The amount of cash we put into the portfolio at the beginning of time period t is given by $\ell_t(x_t, u_t)$, $t = 0, \ldots, T$, where $\ell_t : \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$ is a closed convex function we will describe in detail in §5.2.2; it includes the gross cost of (or revenue from) trades, transaction costs, and other costs associated with holding the portfolio. We will refer to $\ell_t(x_t, u_t)$ as the *stage cost* for period t, and $-\ell_t(x_t, u_t)$ as the *revenue* or income from the portfolio in time period t. We refer to $\ell_T(x_T, u_T)$ as the *terminal cost*. Infinite values of $\ell_t(x_t, u_t)$ are used to encode hard constraints on the portfolio, described in detail in §5.2.2, such as leverage or risk limits or a required final portfolio.

Time variation in the stage cost functions can be used to model many effects. Simple examples include handling terminal costs and constraints, and including a discount factor to take into account the time value of the revenue in different periods. We can also model predictable variation in transaction cost parameters, or enforce leverage or risk limits that vary with time.

Objective. Our overall objective is the expected total cost,

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

where the expectation is over the returns sequence $r_1, \ldots, r_T$, and $u_t = \phi_t(x_t)$. (We assume the expectations exist.) Thus $-J$ is the total expected revenue from the portfolio. If a discount factor has been incorporated into the stage cost functions, $-J$ is the expected present value of the revenue stream.

Stochastic control. The optimal investment problem is to determine a trading policy ϕ_t, $t = 0, \ldots, T$, that minimizes J, *i.e.*, maximizes the expected total revenue. We let $J^\star$ denote the optimal value of J, and we let $\phi_t^\star$, $t = 0, \ldots, T$, denote an

optimal policy. This is a stochastic control problem with linear dynamics and convex stage cost. The data for the problem is the distribution of r_t, the stage cost functions ℓ_t, and the initial portfolio x_0.

Our goal here is not to address the many technical conditions arising in stochastic control (some of which can have ramifications in practical problems). For example, the problem may be unbounded below, *i.e.*, we can find policies that make the total expected revenue arbitrarily negative, in which case $J^\star = -\infty$. As another example, the optimal value $J^\star$ can be finite, but a policy that achieves the optimal value does not exist. For discussion of these and other pathologies, and more technical detail, see [13, 16, 18].

Real-time signals. Our model assumes that the return statistics (but of course not the returns) are known ahead of time, over the whole trading period. Indeed, we will see that the optimal trading policies, as well as our suboptimal trading policies, depend on all values of $\overline{r}_t$ and Σ_t. Thus, our formal model does *not* allow for the use of *real-time signals* in the return distribution model, *i.e.*, the use of real-time data, financial or non-financial, to update or predict the future return distributions during the trading period.

Signals can be formally incorporated into the model, for example, by assuming that the returns are independent, given the current signal values. While much of what we discuss in this chapter can be generalized to this setting, it is far more complex, so we defer it to future work. As a practical matter, however, we note that the trading algorithms we describe can readily incorporate the use of signals, at least informally. We simply re-compute the policies whenever our estimates of the future return statistics change due to signals.

5.2.2 Stage cost function

In this section we detail some of the possible forms that the stage cost function can take. Its general form is

$$\ell_t(x,u) = \begin{cases} \mathbf{1}^T u + \psi_t(x,u) & x+u \in \mathcal{C}_t \\ \infty & \text{otherwise,} \end{cases}$$

where $\mathcal{C}_t \subseteq \mathbf{R}^n$ is the post-trade portfolio constraint set, and $\psi_t : \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$ is a cost, with units of dollars, for period t. We assume that ψ_t and $\mathcal{C}_t$ are closed convex, with $\mathcal{C}_t$ nonempty. The term $\mathbf{1}^T u$ is the gross cash we put into the portfolio due to the trades, without transaction costs. The term $\psi_t(x,u)$ represents any additional amount we pay for the portfolio and trades. The *post-trade constraint set* $\mathcal{C}_t$ will capture constraints on the post-trade portfolio. We will refer to ψ_t as the *transaction cost* function, even though it can also include terms related to positions. The transaction cost ψ_t is typically nonnegative, but we do not need to assume this in the sequel; we intepret $-\psi_t(x_t,u_t)$ as additional revenue when $\psi_t(x_t,u_t)$ is negative.

5.2.3 Post-trade constraints

The post-trade constraint set (or more simply, the constraint set) $\mathcal{C}_t$ defines the set of acceptable post-trade portfolios. Since $\mathcal{C}_t$ is nonempty, it follows that for any value of x_t, we can find a u_t for which $x_t^+ = x_t + u_t \in \mathcal{C}_t$.

We impose explicit constraints only on the post-trade portfolio x_t^+, and not on the portfolio itself. One reason is that we have control over the post-trade portfolio, by buying and selling (*i.e.*, through u_t); whereas the (pre-trade) portfolio x_t is determined by the (random) return r_t in the previous period, and is not directly under our control. In many cases a constraint on the post-trade portfolio implies a similar constraint on the portfolio, with some reasonable assumption about the return distribution (such as nonnegativity). For example, if we impose a nonnegativity (long-only) constraint on x_t^+ and the returns are always nonnegative, then the portfolio values x_t are also nonnegative (long-only).

We now describe examples of portfolio constraints that are useful in practice; we can of course impose any number of these, taking $\mathcal{C}_t$ to be the intersection.

Position limits. The portfolio may be subject to constraints on the post-trade positions, such as minimum and maximum allowed positions for each asset:

$$x_t^{\min} \leq x_t^+ \leq x_t^{\max},$$

where the inequalities are elementwise and $x^{\min}$ and $x^{\max}$ are given (vectors of) minimum and maximum asset holdings, given in dollars. For this constraint $\mathcal{C}_t$ is a box in $\mathbf{R}^n$. One special case is $x_t^+ \geq 0$, which means our (post-trade) portfolio can include only long positions. This corresponds to $\mathcal{C}_t = \mathbf{R}_+^n$, where $\mathbf{R}_+$ is the set of nonnegative reals.

Position limits can also be expressed relative to the total portfolio value; for example,

$$x_t^+ \leq (\mathbf{1}^T x_t^+)\alpha_t,$$

with $\alpha_t \in \mathbf{R}^n$ with positive entries, requires the value in asset i does not exceed the fraction $(\alpha_t)_i$ of the total portfolio value. This constraint is convex, with $\mathcal{C}_t$ a polyhedron.

Some simple position limits that are not convex, and so cannot be used in our model, include minimum nonzero positions, or the restriction that assets are held in integer multiples of some block size (such as 100 shares).

Total value minimum. We can require that the post-trade portfolio maintain minimum total value $v_t^{\min}$, which is the constraint $\mathbf{1}^T x_t^+ \geq v_t^{\min}$. When the pre-trade portfolio value falls $\mathbf{1}^T x_t$ below $v_t^{\min}$ (say, because of a very unfavorable return in the last period), we are required to put cash into the portfolio to bring the total value back up to $v_t^{\min}$. Several other portfolio constraints described below indirectly impose a minimum post-trade portfolio value, typically, zero. This constraint corresponds to $\mathcal{C}_t$ being a halfspace.

Terminal portfolio constraints. The simplest terminal constraint is $x_T^+ = x^{\text{term}}$, where x^{term} is a given portfolio, *i.e.*, $\mathcal{C}_T = \{x^{\text{term}}\}$. (This constraint is the same as fixing the last trade to be $u_T = x^{\text{term}} - x_T$.) In this case our multi-period trading problem is the *optimal execution problem*: We seek the policy that starts from the given initial portfolio at $t = 0$, achieves the given final (post-trade) portfolio at $t = T$, while minimizing expected cost. When x_0 is nonzero and $x^{\text{term}} = 0$, the problem is to 'unwind' the positions x_0, *i.e.*, to cash out of the market over the given period, maximizing expected revenue. When $x_0 = 0$ and x^{term} is some given portfolio, the problem is to make investments over the period to achieve a desired portfolio, at minimum cost. A special case is $x_0 = x^{\text{term}} = 0$, which means the trading starts and ends with no holdings.

Short position and leverage limits. In the simplest case we impose a fixed limit on the total short position in the post trade portfolio:

$$\mathbf{1}^T (x_t^+)_- \leq S_t^{\max},$$

where $(x)_- = \max(-x, 0)$ and $S_t^{\max} \geq 0$ is the maximum allowed total short position. This constraint is convex (in fact, polyhedral), since the lefthand side is a piecewise linear convex function.

We can also limit the total short position relative to the total value of the post-trade portfolio:

$$\mathbf{1}^T (x_t^+)_- \leq \eta_t \mathbf{1}^T x_t^+, \tag{5.2}$$

where $\eta_t \geq 0$, sets the maximum ratio of total short position to total portfolio value. This constraint is convex (in fact, polyhedral), since the lefthand side is a piecewise linear convex function, and the righthand side is a linear function of x_t^+. This limit requires the total post-trade value to be nonnegative; for $\eta_t = 0$ it reduces to a long-only constraint.

The limit (5.2) can be written in several other ways, for example, it is equivalent to

$$\mathbf{1}^T (x_t^+)_- \leq \frac{\eta_t}{1 + \eta_t} \mathbf{1}^T (x_t^+)_+,$$

where $(x)_+ = \max(x, 0)$. In other words, we limit the ratio of the total short to total long positions.

A traditional leverage limit, which limits the ratio of the total short plus total long positions to the total assets under management, can be expressed as

$$\|x_t^+\|_1 = \mathbf{1}^T(x_t^+)_- + \mathbf{1}^T(x_t^+)_+ \leq L_t^{\max} A,$$

where $L_t > 0$ is the leverage limit, and $A > 0$ is the total value of the assets under management. (As a variation on this, we can replace A with $A + \mathbf{1}^T x_t^+$.)

Sector exposure limits. We can include the constraint that our post-trade portfolio has limited exposure to a set of economic sectors (such as manufacturing, energy, technology) or factors (determined by statistical analysis of returns). These constraints are expressed in terms of a matrix $F_t \in \mathbf{R}^{k\times n}$, called the *factor loading matrix*, that relates the portfolio to a vector of k sector exposures: $(F_t x_t^+)_j$ is the sector exposure to sector j. A sector exposure limit can be expressed as

$$s_t^{\min} \leq F_t x_t^+ \leq s_t^{\max},$$

where $s_t^{\min}$ and $s_t^{\max}$ are (vectors of) given lower and upper limits on sector exposures. A special case is sector neutrality, which is the constraint

$$(F_t x_t^+)_j = 0 \tag{5.3}$$

(for sector j neutrality). Sector exposure limits are linear inequality constraints, and sector neutrality constraints are linear equality constraints on x_t^+.

Sector limits can also be expressed relative to the total portfolio value, as in

$$F_t x_t^+ \leq (\mathbf{1}^T x_t^+)\alpha_t,$$

which limits the (positive) exposure in sector i to no more than the fraction $(\alpha_t)_i$ of the total portfolio value.

Concentration limit. A concentration limit requires that no more than a given fraction of the portfolio value can be held in some given fraction (or just a specific number p) of assets. This can be written as

$$\sum_{i=1}^{p}(x_t^+)_{[i]} \leq \beta_t \mathbf{1}^T x_t^+,$$

where $\beta_t > 0$, and the notation $a_{[i]}$ refers to the ith largest element of the vector a. The lefthand side is the sum of the p largest post-trade positions, which is a convex function of x_t^+, and the righthand side is a linear function, so this constraint is convex (in fact, polyhedral) [30, §3.2.3]. This constraint implies that the post-trade portfolio has nonnegative value.

Variance and standard deviation risk limits. We can limit the risk in the post-trade portfolio, using the traditional measure of risk based on variance of post-trade portfolio value over the next period, that is, the variance given x_t and u_t,

$$\mathbf{var}(\mathbf{1}^T x_{t+1} \mid x_t^+) = (x_t^+)^T \Sigma_{t+1} x_t^+.$$

This is a (convex) quadratic function of x_t^+. A simple risk limit can then be expressed as

$$(x_t^+)^T \Sigma_{t+1} x_t^+ \leq \gamma_t,$$

where $\gamma_t > 0$ is a given maximum variance (with units of dollars squared). This constraint is convex; in fact, the associated constraint set is an ellipsoid.

The risk limit above is fixed (for each t). By working with the standard deviation, we can develop risk limits that scale with total portfolio value. This allows us to limit the risk (measured in standard deviation, which has units of dollars) to some fraction of the post-trade portfolio value,

$$((x_t^+)^T \Sigma_{t+1} x_t^+)^{1/2} = \|\Sigma_{t+1}^{1/2} x_t^+\|_2 \leq \delta_t \mathbf{1}^T x_t^+,$$

where $\delta_t > 0$ (and is unitless, since left and righthand sides have units of dollars).

These are convex constraints, in fact, second-order cone (SOC) constraints [30, §4.4.2]. They are also homogeneous constraints; that is, the allowed risk scales with the value of the post-trade portfolio. These constraints require the post-trade portfolio value to be nonnegative.

More sophisticated risk limits. There are many risk measures that are more sophisticated than variance, but are also convex in x_t^+, and therefore fit into our framework. For example, suppose that we do not know the return covariance matrix, but are willing to assume it lies in the convex hull of a set of given covariance matrices $\Sigma^1, \ldots, \Sigma^q$. (We can think of these as the return covariance under q market regimes, or under q possible scenarios.) We can impose a constraint that limits our return variance, under all such scenarios, as

$$(x_t^+)^T \Sigma^i x_t^+ \leq \gamma_t, \quad i = 1, \ldots, q.$$

The associated constraint set is the intersection of ellipsoids (and therefore convex).

Moving beyond quadratic risk measures, we can limit the expected value of a function of period loss,

$$\mathbf{E}(\chi(\mathbf{1}^T x_{t+1} - \overline{r}_{t+1}^T x_t^+) \mid x_t^+) \leq \gamma_t,$$

where $\chi : \mathbf{R} \to \mathbf{R}$ is convex (and typically decreasing). For $\chi(v) = v^2$ we recover quadratic risk; for $\chi(v) = (v)_-^2$, this is downside quadratic risk; for $\chi(v) = (v - v_0)_-$, it is related to conditional value at risk (CVaR) [8, 152]. For such measures the constraint is convex, but not easily handled; typically, one has to resort to Monte Carlo methods to evaluate the risk measure, and stochastic optimization to handle them in an optimization setting; see, *e.g.*, [12, 23, 36, 144, 160].

5.2.4 Transaction and position costs

In this section we describe some of the possible forms that the transaction (and position) cost function ψ can take. Any number of the terms below can be combined

by simple addition, which preserves convexity.

Broker commission. When trades are executed through a broker, we can be charged a commission for carrying out the trade on our behalf. In a simple model this cost is proportional to the total trade volume, which gives

$$\psi_t(x_t, u_t) = \kappa_t^T |u_t|,$$

where $\kappa_t \geq 0$ is the vector of commission rates, and the absolute value is element-wise. When the commission rates are equal across assets, this reduces to $\psi_t(x_t, u_t) = \kappa_t \|u_t\|_1$, where $\kappa_t \geq 0$ is a scalar. A more general form charges different rates for buying and selling:

$$\psi_t(x_t, u_t) = (\kappa_t^{\text{buy}})^T (u_t)_+ + (\kappa_t^{\text{sell}})^T (u_t)_-,$$

where κ_t^{buy} and κ_t^{sell} are nonnegative vectors of buying and selling commission rates. These are all convex functions.

Some broker commission charges, such as charging a flat fee for any (nonzero) amount of trading in an asset, or offering a relative discount for large orders, are nonconvex; see, *e.g.*, [112] for methods for dealing with these nonconvex terms using convex optimization.

Bid-ask spread. The prices at which we buy or sell an asset are different, with the difference referred to as the bid-ask spread; the *mid-price* is the average of the buy and sell prices. If we use the mid-price for each asset as our reference price for converting shares held into our portfolio vector x_t, this means that we sell each asset for a price lower than the mid-price and buy each asset for a price higher than the mid-price. We can model this phenomenon as an additional transaction cost of the form

$$\psi_t(x_t, u_t) = \kappa_t^T |u_t|,$$

where $(\kappa_t)_i$ is one-half the bid-ask spread for asset i. (This has the same form as the broker commission described above.)

Price-impact. When a large order is filled, the price moves against the trader as orders in the book are filled. This is known as price impact. A simple model for price-impact cost is quadratic,

$$\psi_t(x_t, u_t) = s_t^T u_t^2,$$

where $(s_t)_i \geq 0$, and the square above is elementwise. Many other models can be used, such as a 3/2 power transaction cost, $\psi_t(x_t, u_t) = s_t^T |u_t|^{3/2}$, or a general convex piecewise linear transaction cost, which models the depth of the order book at each price level. These are all convex functions of u_t.

We are *not* modeling multi-period price impact, which is the effect of a large order in one period affecting the price in future periods.

Borrowing/shorting fee. When going short, an asset must be borrowed from a third party, such as a broker, who will typically charge a fee for this service proportional to the value of the assets borrowed per period. This gives the (convex) position cost

$$\psi_t(x_t, u_t) = c_t^T (x_t^+)_-,$$

where $(c_t)_i \geq 0$ is the fee rate, in period t, for shorting asset i.

More generally we can pay a fee for our total short position, perhaps as a default insurance policy premium. Such a cost is an increasing convex function of $\mathbf{1}^T (x_t^+)_-$, and is therefore also convex in x_t.

Risk penalty. We have described risk limits as constraints in §5.2.3 above. We can also take risk into account as a real or virtual additional charge that we pay in each period. (For example, the charge might be paid to an internal risk management desk.) In this case $\ell_t(x_t, u_t)$ is the risk-adjusted cost in time period t. Indeed, traditional portfolio optimization is formulated in terms of maximizing risk-adjusted return (the

negative of risk-adjusted cost).

The traditional risk adjustment charge is proportional to the variance of the next period total value, given the current post-trade position, which corresponds to

$$\psi_t(x_t, u_t) = \lambda_t \, \mathbf{var}(\mathbf{1}^T x_{t+1} \mid x_t^+) = \lambda_t (x_t^+)^T \Sigma_{t+1} x_t^+,$$

where $\lambda_t \geq 0$ is called the risk aversion parameter. This traditional charge is not easy to interpret, since λ_t has units of inverse dollars. (The main appeal of using variance is analytical tractability, which is not an issue when convex optimization beyond simple least-squares is used.)

We can levy a charge based on standard deviation rather than variance, which gives

$$\psi_t(x_t, u_t) = \lambda_t \|\Sigma_{t+1}^{1/2} x_t^+\|_2,$$

where in this case $\lambda_t \geq 0$ is dimensionless, and has the interpretation of standard deviation cost rate, in dollar cost per dollar standard deviation. More generally, the charge can be any increasing convex function of $\|\Sigma_{t+1}^{1/2} x_t^+\|_2$, which includes both the standard deviation and variance charges as special cases. All such functions are convex.

5.2.5 Quadratic and QP-representable stage cost

Here we describe two special forms for the stage cost, for future use.

Quadratic. An important special case occurs when ℓ_t is (convex) quadratic, possibly including linear equality constraints. This means that the transaction cost is quadratic,

$$\psi_t(x, u) = (1/2) \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} A_t & B_t & a_t \\ B_t^T & C_t & c_t \\ a_t^T & c_t^T & d_t \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix},$$

where

$$\left[\begin{array}{cc} A_t & B_t \\ B_t^T & C_t \end{array}\right] \succeq 0,$$

(meaning, the matrix on the lefthand side is symmetric positive semidefinite), and the constraint set is

$$\mathcal{C}_t = \{x \mid G_t x = h_t\}.$$

When the stage cost has this form, we refer to the multi-period portfolio optimization problem as being quadratic. The quadratic problem is quite limited; it can include, for example, a terminal portfolio constraint, a quadratic risk penalty, and a quadratic transaction cost. The other constraints described above yield a problem that is not quadratic.

QP-representable. We say that the stage cost is QP-representable if ℓ_t is convex quadratic plus a convex piecewise linear function, possibly including linear equality and inequality constraints. Such a function can be minimized by transformation to a QP, using standard techniques (described, *e.g.*, in [30, 133], and employed in convex optimization systems such as CVX [78], YALMIP [114], and CVXGEN [122]).

Many of the transaction cost terms and constraints described above are QP-representable, including leverage limits, sector exposure limits, concentration limits, broker fees, bid-ask spread, and quadratic risk penalty. Quadratic and second-order cone risk limits are not QP-representable, but can be represented as a second-order cone problem (SOCP) [30, §4.4], [113].

5.3 Optimal policy

5.3.1 Dynamic programming

In this section we briefly review the dynamic programming characterization of the solution to the stochastic control problem. For more detail, see, *e.g.*, [9, 13, 16, 50, 127, 153].

The (Bellman) value functions $V_t : \mathbf{R}^n \to \mathbf{R}$, $t = 0, \ldots, T+1$ are defined by $V_{T+1} = 0$, and the backward recursion

$$V_t(x) = \inf_u \left(\ell_t(x,u) + \mathbf{E}\, V_{t+1}(R_{t+1}(x+u)) \right), \quad t = T, T-1, \ldots, 0, \tag{5.4}$$

where the expectation is over the return r_{t+1}. We can write this recursion compactly as

$$V_t = \mathcal{T}_t V_{t+1}, \quad t = T, \ldots, 0, \tag{5.5}$$

where $\mathcal{T}_t$ is the Bellman operator, defined as

$$(\mathcal{T}_t h)(x) = \inf_u \left(\ell_t(x,u) + \mathbf{E}\, h(R_{t+1}(x+u)) \right),$$

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

An optimal policy can be expressed in terms of the value functions as

$$\phi_t^\star(x) \in \operatorname*{argmin}_u \left(\ell_t(x,u) + \mathbf{E}\, V_{t+1}(R_{t+1}(x+u)) \right), \tag{5.6}$$

and the optimal cost is given by

$$J^\star = V_0(x_0).$$

This general dynamic programming solution method is not a practical algorithm, since we do not in general have a method to represent V_t, let alone effectively carry out the expectation and partial minimization operations. In the quadratic case (described below), however, the Bellman iteration can be explicitly carried out, yielding an explicit form for V_t and $\phi_t^\star$.

Monotonicity. For future use we note that the Bellman operator $\mathcal{T}_t$ is monotonic: for any $f : \mathbf{R}^n \to \mathbf{R}$ and $g : \mathbf{R}^n \to \mathbf{R}$,

$$f \leq g \quad \Longrightarrow \quad \mathcal{T}_t f \leq \mathcal{T}_t g, \tag{5.7}$$

where the inequalities are interpreted pointwise [13, 16].

Convexity. The value functions $V_0, \ldots, V_{T+1}$ are convex, which we can show by a (backward) recursion. We first observe that $V_{T+1} = 0$ is convex. We will show that the Bellman operators $\mathcal{T}_t$ preserve convexity; this will show that all V_t are convex functions.

To show that the Bellman operator $\mathcal{T}_t$ preserves convexity, suppose h is convex. Then, for fixed R_{t+1}, $h(R_{t+1}(x+u))$ is a convex function of (x, u); since expectation preserves convexity, we conclude that $\mathbf{E}\, h(R_{t+1}(x+u))$ is convex in (x, u). The stage cost $\ell_t(x, u)$ is convex, so $\ell_t(x, u)+\mathbf{E}\, h(R_{t+1}(x+u))$ is convex in (x, u). Finally, partial minimization of this function (in this case, over u) preserves convexity, so we conclude that $\mathcal{T}_t h$ is convex. (For more on the convexity rules used above, see, *e.g.*, [30, Ch. 3].)

One implication of the convexity of V_t is that evaluation of the optimal policy (5.6) requires solving a convex optimization problem.

5.3.2 Quadratic case

When the problem is quadratic, *i.e.*, ℓ_t are quadratic functions plus (possibly) linear equality constraints, we can effectively compute V_t, which are also (convex) quadratic functions. We give the argument in general form here, with more detail given in the appendices.

The argument is similar to that for convexity of V_t, with the attribute 'quadratic' substituted for 'convex'. We note that V_{T+1} is a quadratic function, and we will show that the Bellman operators preserve quadratic functions. It follows that all V_t are quadratic. Moreover we can explicitly compute the coefficients of the quadratic functions, so the method can be implemented.

To show that the Bellman operator $\mathcal{T}_t$ preserves convex quadratic functions, suppose h is convex quadratic. Then, for fixed R_{t+1}, $h(R_{t+1}(x+u))$ is a convex quadratic function of (x, u); since expectation preserves convex quadratic functions, we conclude that $\mathbf{E}\, h(R_{t+1}(x+u))$ is convex quadratic in (x, u). (To explicitly compute its coefficients requires knowledge of $\overline{r}_{t+1}$ and Σ_{t+1}, but no other attribute of the return

distribution.) A detailed derivation, including fomulas for the coefficients, is given in appendix A.2.

The stage cost $\ell_t(x,u)$ is assumed convex quadratic (plus linear equality constraints), so $\ell_t(x,u) + \mathbf{E}\, h(R_{t+1}(x+u))$ is convex quadratic in (x,u) (since convex quadratic functions are closed under addition). Finally, partial minimization of a convex quadratic, possibly subject to linear equality constraints, preserves convex quadratic functions, so we conclude that $\mathcal{T}_t h$ is convex quadratic. See appendix A.3 for a detailed derivation, and explicit formulas for the coefficients.

The optimal trading policies require the minimization of a convex quadratic function of (x,u) over u (possibly, with equality constraints). The minimizer of a convex quadratic function of (x,u), subject to linear equality constraints, can be expressed explicitly as an affine function of x (see appendix A.3). Thus, the optimal trading policies have the form

$$\phi_t^\star(x) = J_t x + k_t, \quad t = 0, \ldots, T, \tag{5.8}$$

where $J_t \in \mathbf{R}^{n\times n}$ and $k_t \in \mathbf{R}^n$ can be explicitly computed. (We can without loss of generality take $J_0 = 0$.) This is one of the few cases for which the value functions (and hence, the optimal policy), can be explicitly computed. The coefficients J_t and k_t in the optimal policy depend on the coefficients in the stage cost functions ℓ_τ (including coefficients in any linear equality constraints), as well as $\overline{r}_\tau$ and Σ_τ, for $\tau = t, \ldots, T$.

5.3.3 No transaction cost case

Here we consider another special case in which we can solve the stochastic control problem. Suppose that $\psi_t(x,u)$ is a function of $x^+ = x+u$, which we write (with some abuse of notation) as $\psi_t(x^+)$. In other words, we exclude transaction costs (broker fees, bid-ask spread, and price impact type terms); the only stage costs we have are functions of the post-trade portfolio. (We have already assumed that the constraints have this form.)

The stage cost then has the form

$$\ell_t(x,u) = \mathbf{1}^T x^+ - \mathbf{1}^T x + \psi_t(x^+) + I_t(x^+),$$

where I_t is the indicator function of $\mathcal{C}_t$. The objective can then be written as

$$\begin{aligned} J &= \mathbf{E}\sum_{t=0}^{T} \ell_t(x_t,u_t) \\ &= \sum_{t=0}^{T} \mathbf{E}\left(\mathbf{1}^T x_t^+ + \psi_t(x_t^+) + I_t(x_t^+)\right) - \mathbf{1}^T x_0 - \mathbf{E}\sum_{t=1}^{T} r_t^T x_{t-1}^+ \\ &= -\mathbf{1}^T x_0 + \sum_{t=0}^{T} \mathbf{E}\left((\mathbf{1}-r_{t+1})^T x_t^+ + \psi_t(x_t^+) + I_t(x_t^+)\right), \end{aligned}$$

where we take $r_{T+1} = 0$. Thus, our problem is equivalent to a stochastic control problem with the modified stage cost

$$\tilde{\ell}_t(x,u) = (\mathbf{1}-r_{t+1})^T x^+ + \psi_t(x^+) + I_t(x^+),$$

which is a function only of $x^+ = x+u$. This stochastic control problem has an associated value function, which we denote by $\tilde{V}$, which satisfies (5.4) with the modified stage cost function.

Using the modified stage cost, the minimization in the optimal policy (5.6) is over a function of x^+. To minimize a function of $x^+ = x+u$ over u, we simply minimize the function over x^+ to find $x^{+\star}$, and then take $u = x^{+\star} - x$. For the optimal policy (5.6), we conclude that $\phi_t^\star(x_t) = x_t^{+\star} - x_t$, where $x_t^{+\star}$ minimizes

$$(\mathbf{1}-\overline{r}_{t+1})^T x_t^+ + \psi_t(x_t^+) + I_t(x_t^+) + \mathbf{E}\,\tilde{V}_{t+1}(R_{t+1}x_t^+)$$

over x_t^+. Moreover, the minimum value of this expression is independent of x, which, together with (5.4), implies $\tilde{V}_t$ are constant functions. It follows that we can find $x_t^{+\star}$ as the minimizers of

$$(\mathbf{1}-\overline{r}_{t+1})^T x_t^+ + \psi_t(x_t^+) + I_t(x_t^+).$$

Thus, an optimal policy can be found as follows. For $t = 0,\ldots,T$, we solve the

problems

$$\begin{array}{ll} \text{minimize} & (\mathbf{1} - \overline{r}_{t+1})^T x_t^+ + \psi_t(x_t^+) \\ \text{subject to} & x_t^+ \in \mathcal{C}_t, \end{array} \tag{5.9}$$

over variables x_t^+, to determine optimal post-trade portfolios $x_t^{+\star}$. An optimal policy simply rebalances to these optimal post-trade portfolios:

$$\phi_t^\star(x) = x_t^{+\star} - x$$

(which is an affine policy, of a special form). The optimal objective $J^\star$ is the sum of the optimal values of the problems (5.9) less the total wealth of the initial portfolio, $\mathbf{1}^T x_0$.

The optimization problems (5.9) are single-stage portfolio problems, which can be solved independently (in parallel). When the stage-cost function is QP-representable, they reduce to QPs; for standard deviation risk limits, they reduce to SOCPs.

In summary, when all stage costs are functions only of the post-trade portfolio, the multi-period investment problem is readily solved using standard single-period portfolio optimization. However, when cost terms that depend on u are included in the model, such as broker commission, bid-ask spread, or price-impact, the full stochastic control formulation is needed. These are significant terms in most practical multi-period investment problems, which provides the motivation for developing a method for handling them.

5.4 Performance bounds

Here we describe methods for computing a lower bound on the optimal value $J^\star$, using techniques described in a recent series of papers [135, 179, 180, 183]. We describe the development of these bounds starting from the high level ideas, and then proceed to lower level details.

These methods are based on finding a set of convex quadratic functions V_t^{lb} that are underestimators of V_t, *i.e.*, satisfy $V_t^{\mathrm{lb}} \leq V_t$, where the inequality means pointwise.

It follows that

$$J^{\mathrm{lb}} = V_0^{\mathrm{lb}}(x_0) \leq V_0(x_0) = J^\star,$$

i.e., J^{lb} is a lower bound on $J^\star$. We let V_t^{lb} take the form

$$V_t^{\mathrm{lb}}(x) = (1/2) \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} P_t & p_t \\ p_t^T & q_t \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix},$$

where $P_t \succeq 0$.

Bellman inequalities. The quadratic underestimators are found as follows. We construct a set of quadratic functions that satisfy the Bellman *inequalities*, [47,85,118],

$$V_t^{\mathrm{lb}} \leq \mathcal{T}_t V_{t+1}^{\mathrm{lb}}, \quad t = T, \ldots, 0, \tag{5.10}$$

with $V_{T+1}^{\mathrm{lb}} = 0$ (*cf.* the Bellman recursion *equalities*, given in (5.5)). By monotonicity of the Bellman operator we get

$$V_T^{\mathrm{lb}} \leq \mathcal{T}_T V_{T+1}^{\mathrm{lb}} \leq \mathcal{T}_T V_{T+1} = V_T.$$

Continuing this argument recursively we see that

$$V_t^{\mathrm{lb}} \leq V_t, \quad t = 0, \ldots, T+1,$$

i.e., V_t^{lb} are underestimators of V_t.

LMI conditions. Now we explain how to obtain quadratic functions that satisfy the Bellman inequalities (5.10). For simplicity, we describe the method when ψ_t is convex quadratic and $\mathcal{C}_t$ has the representation

$$\mathcal{C}_t = \{x \mid g_1(x) \leq 0, \ldots, g_s(x) \leq 0,\ g_{s+1}(x) = \cdots = g_M(x) = 0\}, \tag{5.11}$$

where $g_i : \mathbf{R}^n \to \mathbf{R}$ are convex quadratic constraint functions. (In fact, after appropriate problem transformations, this includes all of the transaction cost terms and constraints described in §5.2.2, except for the more sophisticated risk bounds.)

The Bellman inequality (5.10) can be written as

$$V_t^{\mathrm{lb}}(x) \leq \mathbf{1}^T u + \psi_t(x,u) + \mathbf{E}\, V_{t+1}^{\mathrm{lb}}(R_{t+1}(x+u)), \quad \forall (x+u) \in \mathcal{C}_t.$$

From appendix A.2 we know that when V_{t+1}^{lb} is convex quadratic, $\mathbf{E}\, V_{t+1}^{\mathrm{lb}}(R_{t+1}(x+u))$ is also convex quadratic, so

$$\mathbf{1}^T u + \psi_t(x,u) + \mathbf{E}\, V_{t+1}^{\mathrm{lb}}(R_{t+1}(x+u)) - V_t^{\mathrm{lb}}(x) \tag{5.12}$$

is quadratic in (x,u). Thus, the Bellman inequality (5.10) requires that the quadratic function (5.12) be nonnegative on the set $\mathcal{C}_t$, which is described by the set of quadratic inequalities (5.11).

A sufficient condition for a quadratic function to be nonnegative on a set described by a set of quadratic inequalities can be obtained via the $\mathcal{S}$-procedure [26, §2.6.3], [30, §B.2], which results in a set of linear matrix inequalities (LMIs) in the coefficients of V_t^{lb} (see appendix A.1 for details). If P_t, p_t, q_t, $t = 0, \ldots, T+1$ satisfy these LMIs, then V_t^{lb}, $t = 0, \ldots, T+1$ must satisfy the Bellman inequalities (5.10), and hence are value function underestimators. We can then maximize our lower bound on $J^\star$,

$$J^{\mathrm{lb}} = V_0^{\mathrm{lb}}(x_0) = (1/2) x_0^T P_0 x_0 + p_0^T x_0 + q_0$$

(which is a linear function of the coefficients of V_t^{lb}), subject to these LMIs. This is an SDP with variables P_t, p_t, q_t, $t = 0, \ldots, T+1$.

The details of this method, including a full derivation of the SDPs involved, can be found in appendix A.4 and [180, 183].

Summary. The method described in this section produces a (numerical) lower bound on $J^\star$, the optimal value of the stochastic control problem, given the problem data, *i.e.*, the stage cost function and the first and second moments of the return

distributions. We note that while the performance of any given policy can depend on higher moments of the returns, our performance bound only depends on the first and second moments. The method requires forming and solving an SDP, with on the order of T matrix variables of size $n \times n$.

5.5 Approximate dynamic programming

Except for the special case when the problem is quadratic, described in §5.3.2 above, it is usually impossible to compute (or even represent) the value functions V_t. A common alternative is to replace the value functions appearing in (5.6) with approximate (or surrogate) value functions $\hat{V}_t : \mathbf{R}^n \to \mathbf{R}$, which gives the ADP policy

$$\phi_t^{\mathrm{adp}}(x) \in \operatorname*{argmin}_u \left(\ell_t(x,u) + \mathbf{E}\,\hat{V}_t(R_{t+1}(x+u))\right), \quad t = 0, \ldots, T. \tag{5.13}$$

When $\hat{V}_t = V_t$, the ADP policy is optimal; when $\hat{V}_t = 0$, the ADP policy is the greedy policy, in which trades are chosen to minimize the current stage cost, without regard for any impact on the future (other than those contained in the current stage cost). When $\hat{V}_t$ is convex, evaluating the ADP policy requires solving a convex optimization problem. We let J^{adp} denote the cost achieved by the ADP policy.

A variety of methods, which we do not survey here, can be used to choose approximate value functions (see, *e.g.*, [19, 47, 109, 143]). The goals in choosing approximate value functions $\hat{V}_t$ are the following.

- *Ease of evaluation.* The ADP policy should be easily evaluated, *i.e.*, the minimization over u in (5.13) should be easy to carry out. For example, when the stage cost is QP-representable, and $\hat{V}_t$ are convex quadratic, evaluation of the ADP policy reduces to solving a QP, for which there are very efficient methods.

- *Performance.* The ADP policy should attain near-optimal performance, *i.e.*, $J^{\mathrm{adp}} \approx J^\star$. (A more modest goal is for the ADP policy to obtain good performance, meaning J^{adp} is not too much larger than $J^\star$.) This would be the case if J^{adp} is not much larger than J^{lb}, the lower bound computed using the methods

of §5.4.

One simple and general method for obtaining approximate value functions is to find the (exact) value functions for a quadratic problem that approximates in some sense or relaxes the original problem, for example, by ignoring portfolio constraints other than equality constraints, as well as any part of the transaction cost that is not quadratic (such as bid-ask spread). Then we use these value functions as approximate value functions for the original problem.

5.5.1 Quadratic approximate dynamic programming

When the approximate value functions are convex quadratic, $\mathbf{E}\,\hat{V}_t(R_{t+1}(x+u))$ is a convex quadratic function, whose coefficients can be explicitly computed, using $\overline{r}_{t+1}$ and Σ_{t+1}. (This is shown in appendix A.2.) Thus, evaluating the ADP policy in this case reduces to solving a convex optimization problem, with an explicit objective function that does not involve expectation over returns.

A further simplification occurs when the stage cost is QP-representable. In this case, evaluation of the ADP policy reduces to solving a QP, for which there are a number of effective methods, already mentioned above. It can be shown in this case that the ADP policy is a piecewise affine function of the current portfolio x [11], *i.e.*, it has the form

$$\phi_t^{\mathrm{adp}}(x) = J_t^i x + k_t^i, \quad x \in \mathcal{R}_t^i,$$

where $\{\mathcal{R}_t^1, \ldots, \mathcal{R}_t^N\}$ is a polyhedral partition of $\mathbf{R}^n$ (*cf.* the optimal policy for the quadratic case, given in (5.8)). This observation leads to another method for implementing the ADP policy, called explicit MPC or multi-parametric QP [10, 104]: We compute J_t^i, k_t^i, and (the inequalities that define) $\mathcal{R}_t^i$ explicitly, off-line. On-line, policy evaluation requires two steps. First, the region i containing the current portfolio x_t is determined; then the corresponding affine trading policy is applied. This method works well when the number of assets is small (say, no more than around 5), but quickly becomes intractable for larger portfolios, since N grows exponentially with n.

5.5.2 ADP based on quadratic underestimators

The quadratic underestimators V_t^{lb} found using the performance bounding method described in §5.4 are natural candidates for approximate value functions in ADP. Indeed, ADP policies using $\hat{V}_t = V_t^{\mathrm{lb}}$ have been observed to perform well in many practical applications, including variations on the multi-period portfolio optimization problem considered here [180, 183].

Let us summarize what is required to evaluate this ADP policy.

- *Performance bound computation.* We solve an SDP to obtain J^{lb}, a lower bound on $J^\star$, as well as the (coefficients of) quadratic approximate value functions V_t^{lb}. This SDP involves around T matrix variables of size $n \times n$. This can be done off-line, before the trading begins.

- *On-line policy evaluation.* In each period we solve a small QP, with around n variables, which includes the current portfolio x_t in its data, to determine which trades to carry out.

The first task can involve substantial computation, but is carried out off-line; the second task, which is carried out on-line in each trading period, can be done very quickly.

5.6 Model predictive control

In this section we describe certainty-equivalent model predictive control (MPC), another suboptimal policy for the multi-period investment problem. MPC is a widely used and extensively studied suboptimal policy; see, *e.g.*, [64, 76, 105, 117, 125, 184]. It can also be interpreted as an ADP policy, with an appropriately defined approximate value function.

5.6.1 Policy

MPC is based on a simple idea. To determine u_t, we replace all future (unknown) returns with their mean values, $\overline{r}_\tau$, $\tau = t+1, \ldots, T$. This turns the stochastic control

problem into a standard optimization problem,

$$\begin{array}{ll} \text{minimize} & \sum_{\tau=t}^{T} l_\tau(z_\tau, v_\tau) \\ \text{subject to} & z_{\tau+1} = \mathbf{diag}(\overline{r}_{\tau+1})(z_\tau + v_\tau), \quad \tau = t, \ldots, T-1 \\ & z_t = x_t \end{array} \tag{5.14}$$

with variables z_τ, v_τ, $\tau = t, \ldots, T$. We solve this convex optimization problem to obtain an optimal sequence of trades $v_t^\star, \ldots, v_{T-1}^\star$. This sequence is a plan for future trades over the remaining trading horizon, under the (highly unrealistic) assumption that future returns will be equal to their mean values. The MPC policy takes $u_t = \phi^{\mathrm{mpc}}(x_t) = v_t^\star$. In other words, we execute the first trade in our planned sequence of trades. At the next step, we repeat the process, starting from the new portfolio x_{t+1}.

5.6.2 Implementation

Evaluating $\phi^{\mathrm{mpc}}(x_t)$ requires solving the convex optimization problem (5.14), which is an optimization problem with $2(T-t)n$ variables. Using a generic optimization solver that does not exploit sparsity structure, the computational effort required is order $(T-t)^3n^3$ flops (floating-point operations). The MPC problem (5.14) can be more efficiently solved by exploiting its structure. When the variables are appropriately ordered, the linear equations that are solved in each step of an interior-point method are block tri-diagonal, and can be solved with a computational effort that grows linearly in $T-t$. In this case, the overall computational effort is order $(T-t)n^3$ flops. For details, see, *e.g.*, [2, 124, 147, 181], which describe implementations of the MPC policy for similar control problems. Note that the cost of evaluating the MPC policy decreases with increasing t, since later in the trading interval our planning is done over a shorter horizon. The total cost of the T evaluations of the MPC policy, for $t = 1, \ldots, T$, is order T^2n^3 flops. In contrast, the total cost of T evaluations of the ADP policy is order Tn^3 flops.

Comparison with quadratic ADP policy. We can compare the computational cost of the MPC and ADP policies. The main difference is that the ADP policy

requires significantly more off-line computation, while MPC is more expensive to evaluate on-line.

- *Off-line pre-computation.* The ADP policy requires solving a large SDP to find the quadratic approximate value functions. This is done off-line, before the trading policy is implemented on-line. In contrast, MPC does not require any pre-computation. The only computation that is required is solving (5.14), which is solved on-line at every time step.

- *On-line policy evaluation.* In ADP, after the quadratic approximate value functions have been computed, on-line evaluation requires solving a QP with n variables. The computational effort required is order n^3 flops. In contrast, evaluating the MPC policy requires order $(T-t)n^3$ flops (assuming we exploit the sparsity structure).

Signals. There are computational advantages to using MPC in cases when (estimates of) future return statistics are updated in real-time, using signals. In this case, the expected returns $\overline{r}_t$ are simply replaced with the most recent return estimates. In contrast, for our quadratic ADP policies, when return statistics are updated, the quadratic approximate value function must be re-computed, which requires solving a large SDP. Our ADP policies can be extended to handle some real-time signals without re-solving the SDP, but the method is more complicated, and is beyond the scope of this work.

5.6.3 Interpretation as an ADP policy

Model predictive control is itself an ADP policy, and can be interpreted as a special case of a method called *rollout*, a popular method in approximate dynamic programming [13, §6.4] [14, 15, 165]. In rollout, the approximate value function is taken to be the cost achieved by a heuristic policy called the *base policy*. In most cases, the cost achieved by the base policy can only be evaluated via Monte Carlo simulation, *i.e.*, by 'rolling out' possible sample paths.

In MPC, the base heuristic is an open loop policy, where we compute an optimal sequence of trades starting from an initial portfolio, assuming that the true returns are equal to their mean values $\overline{r}_t$. In this base policy, the planned sequence of trades is executed without recourse, *i.e.*, the trades depend only on the initial portfolio. Evaluating the cost achieved by this policy would require Monte Carlo simulation, which is computationally intensive (We would need to evaluate the cost achieved over a large number of return trajectories and average.)

Thus, in MPC a further simplification is to roll out only one return trajectory—the trajectory of mean returns. We can write the MPC policy as

$$\phi_t^{\mathrm{mpc}}(x) = \operatorname*{argmin}_u \left(\ell_t(x,u) + V_{t+1}^{\mathrm{mpc}}(\overline{R}_{t+1}(x+u))\right), \quad t = 0, \ldots, T,$$

where $V_t^{\mathrm{mpc}}(z)$ is the optimal value of the problem (5.14), starting from $x_t = z$ at period t. We can see that MPC can be interpreted as an ADP policy, with a relatively complex approximate value function (given as the optimal cost of a convex optimization problem).

For more details on interpretations of MPC, and connections to approximate dynamic programming and rollout, see [11, 13–16].

5.6.4 Truncated MPC

The MPC policy described above plans a sequence of trades for the full time interval $t, \ldots, T$. A common variation is to look ahead a limited number of steps, M, into the future. At each time t we solve

$$\begin{array}{ll} \text{minimize} & \sum_{\tau=t}^{t+M-1} l_\tau(z_\tau, v_\tau) + V_{t+M}^{\mathrm{term}}(z_{t+M}) \\ \text{subject to} & z_{\tau+1} = \mathbf{diag}(\overline{r}_{\tau+1})(z_\tau + v_\tau), \quad \tau = t, \ldots, t+M-1 \\ & z_t = x_t \end{array} \tag{5.15}$$

with variables $v_t, \ldots, v_{t+M-1}$, and $z_t, \ldots, z_{t+M}$. Here, M is the number of steps of look-ahead, and V_{t+M}^{term} is the terminal cost (to be chosen). As with full look-ahead MPC, we take $\phi_t^{\mathrm{mpc}}(x_t) = v_t^\star$ and repeat the process at the next time step, starting

from the portfolio x_{t+1}. For $t > T - M$, we use the basic MPC policy (5.14).

An important parameter in this policy is the terminal cost V_{t+M}^{term}. If this cost is appropriately chosen, the truncated MPC policy is exactly the same as the full look-ahead policy [11]. Common choices for terminal costs are approximate value functions obtained via any approximate dynamic programming method.

The idea in truncated MPC is to trade off on-line and off-line computation. If the number of steps of look-ahead M is large, we need a less accurate approximate value function as our terminal cost, so less off-line computation is needed. On the other hand, if we are willing to spend a lot of time off-line computing a good approximate value function, we can set $M \ll T$ and save significant computational effort on-line. Thus, truncated MPC gives a family of policies that sit in between (full look-ahead) MPC and ADP.

5.7 Numerical examples

In this section we present results for several numerical examples. We first describe the five problems we consider.

5.7.1 Problem data

All five examples use zero initial portfolio $x_0 = 0$, and impose a zero terminal portfolio constraint: $x_T^+ = 0$. The five examples use the same return distribution; they differ only in the choice of stage cost function. Our first example is a quadratic problem, for which we can compute the optimal policy exactly. The other four examples use the same transaction cost terms, and differ only in the choice of portfolio constraints. All examples involve $n = 30$ assets, over a time horizon of 100 time steps, *i.e.*, $T = 99$.

Return distribution. The returns r_t are identically distributed with log-normal distribution,

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

where μ and $\tilde{\Sigma}$ are the mean and covariance of the log return. The return mean and covariance are then given by

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

The log return mean and covariance were chosen as follows. First we chose the log return standard deviations $(\tilde{\Sigma}_{ii})^{1/2}$ from a uniform distribution on $[0, 0.01]$. We then formed $\tilde{\Sigma}$ using

$$\tilde{\Sigma}_{ij} = C_{ij}(\tilde{\Sigma}_{ii}\tilde{\Sigma}_{jj})^{1/2}, \quad i,j = 1,\ldots,n,$$

where C is a matrix of correlation coefficients chosen at random, with entries varying from about -0.3 to $+0.9$. To form C we generate a matrix $Z \in \mathbf{R}^{n\times n}$ with all entries drawn from a standard Gaussian distribution, then form $Y = ZZ^T + \zeta\mathbf{1}\mathbf{1}^T$, where $\zeta > 0$, and finally set

$$C = \mathbf{diag}(Y_{11}^{-1/2},\ldots,Y_{nn}^{-1/2})Y\,\mathbf{diag}(Y_{11}^{-1/2},\ldots,Y_{nn}^{-1/2}).$$

We chose ζ so that the entries of C are in the range we desire. The log return means μ_i were chosen from a $\mathcal{N}(0, 0.03^2)$ distribution. The resulting mean returns $\overline{r}_i$ ranged from 0.95 to 1.08; the asset standard deviations $(\Sigma_{ii})^{1/2}$ ranged from 0.03 to 0.10.

Transaction cost. For simplicity we consider a transaction cost that does not vary over time, *i.e.*, $\psi_t = \psi$ for $t = 0,\ldots,T$. For the quadratic example, the transaction cost is

$$\psi(x_t, u_t) = s^T u_t^2 + \lambda(x_t^+)^T\Sigma x_t^+,$$

which includes a quadratic transaction cost and a quadratic risk penalty. For the other examples, the transaction cost is

$$\psi(x_t, u_t) = c^T(x_t^+)_- + \kappa^T|u_t| + s^T u_t^2 + \lambda(x_t^+)^T\Sigma x_t^+,$$

which includes a quadratic risk penalty, a quadratic transaction cost, a shorting fee and an absolute term to model bid-ask spread and broker commissions. The stage cost for the quadratic example is smaller than the stage cost for the other examples, since the additional terms are nonnegative. It follows that the optimal cost for the quadratic example is a lower bound on the optimal cost for the other examples.

The parameter λ was set to 0.5; s_i, κ_i, and c_i were sampled from uniform distributions on $[0, 1]$, $[0, 0.1]$ and $[0, 0.05]$, respectively. These choices resulted in each term giving a significant contribution to the over all objective.

Constraint set. For all five examples we have a zero terminal portfolio constraint, *i.e.*, $\mathcal{C}_T = \{0\}$. The quadratic case has no other constraints, *i.e.*, $\mathcal{C}_t = \mathbf{R}^n$ for $t = 0, \ldots, T-1$. For the other four examples we take one of the following constraints for $t = 0, \ldots, T-1$:

- Unconstrained: $\mathcal{C}_t = \mathbf{R}^n$.
- Long-only: $\mathcal{C}_t = \mathbf{R}^n_+$.
- Leverage limit: $\mathcal{C}_t = \{x \mid \mathbf{1}^T(x)_- \leq \eta \mathbf{1}^T x\}$, with $\eta = 0.3$.
- Sector neutral: $\mathcal{C}_t = \{x \mid Fx = 0\}$, with $F \in \mathbf{R}^{2\times n}$; the rows of F are the eigenvectors associated with the two largest eigenvalues of Σ.

5.7.2 Computation

Cost evaluation and simulation. For the quadratic example, we can evaluate the optimal cost exactly, as $V_0(0)$, as well as the (affine) optimal policy. For the other four problems we use Monte Carlo simulation to compute (approximately) the objective value obtained by the ADP and MPC policies. For the quadratic example, the estimate of objective obtained from Monte Carlo serves as a consistency check. To evaluate the performance of the ADP policy we ran 50000 Monte Carlo simulations for each example. Thus, Monte Carlo simulation of each example required solving around 5 million small QPs. To evaluate the performance of the MPC policy we ran 5000 Monte Carlo simulations for each example, which required solving 500000 larger

MPC QPs. (Of course we could have reduced the number of Monte Carlo samples we needed to take by using one of the many variance reduction techniques, such as control variates or importance sampling.) Our sampling was sufficient to obtain around three significant figures of accuracy in our performance evaluations, as estimated by the empirical variance in our estimates (reported below), and also verified experimentally by re-running the simulations with different initial random number generator seed, which yielded numerical results that agreed to three significant figures.

Convex optimization solvers. All computation was carried out on an Intel Xeon processor, with clock speed 3.4GHz, running Linux.

We used CVX [78] to formulate the SDPs required to compute the performance bounds, which in turn uses SeDuMi [164] to solve the transformed SDPs. To give an idea of the solution times involved, consider the long-only example. The SDP required to compute the performance bound had 474100 variables and 59096 constraints, after transformation to a standard form SDP, and took about 24 minutes to solve. (An optimized solver for this type of problem could have solved the SDP much more quickly.)

To solve the QPs required to evaluate the ADP policies we used CVXGEN [122] to generate extremely fast custom solvers. The QP that must be solved in each time step to evaluate the ADP trading policy has 30 variables and 30 constraints; transformed into a standard form QP it has 90 variables and 122 constraints. The CVXGEN generated solver (which is single thread) solves such a problem in around 300μs. (And this computation time could easily have been reduced by a factor of 3 or more, using various tricks for high speed embedded solvers [120–122].) Thus, a simulation run of 100 time periods requires about 30ms; running 50000 simulations in series can be carried out in around 25 minutes. On the 8-core Xeon, the 5 million QPs are solved in slightly more than 3 minutes. (Using CVX, the total time would have been measured in days.) Of course, the Monte Carlo simulations are trivially parallelizable, and could easily have been carried out on k processors with a speed-up of around k.

Evaluating the MPC policy required solving much larger QPs, beyond the size

Example	Lower bound	ADP performance	MPC performance
quadratic	-450.1	-450.0	-444.3
unconstrained	-132.6	-131.9	-130.6
long-only	-41.3	-41.0	-40.6
leverage limit	-87.5	-85.6	-84.7
sector neutral	-121.3	-118.9	-117.5

Table 5.1: Lower bound on performance, and ADP and MPC policy performance.

limits of CVXGEN. To solve these QPs quickly we used an operator splitting technique known as the alternating direction method of multipliers (ADMM) [29], specialized to the case of convex optimal control, and described in full in [138]. The QPs to be solved to evaluate the MPC policy are larger at earlier time periods and get progressively smaller as we approach the final period. For example, the QP solved at the first period for the long-only example has 18243 variables and 9052 constraints (when converted to standard form). It requires about 88ms to solve this QP using ADMM; solving the same problem using CVX takes more than 3 minutes.

5.7.3 Performance bounds and policy performance

The results are summarized in table 5.1. The first column gives the lower bound on performance computed using our method. The second and third columns give the performance attained by the ADP and MPC policies, respectively, estimated via Monte Carlo simulation. For the quadratic example, the lower bound is the optimal performance; the ADP policy is in fact the optimal policy, so the ADP performance entry for the quadratic example serves as a check on our Monte Carlo simulation (which is consistent with our estimate of Monte Carlo error given below). In all cases we see that the policies are very nearly optimal, since the lower bound and performance obtained are very close. In any practical sense, our ADP and MPC trading policies are optimal.

Example	ADP	MPC
quadratic	30.9	40.3
unconstrained	9.6	9.9
long-only	7.1	7.2
leverage limit	9.8	9.9
sector neutral	8.9	9.9

Table 5.2: Empirical standard deviations of total cost for ADP and MPC Monte Carlo simulations.

Monte Carlo error estimation. The empirical standard deviations for the ADP and MPC Monte Carlo simulations are listed in table 5.2. For the quadratic problem the positions typically involve higher leverage than for the other problems, resulting in total cost with higher variance. For the quadratic problem, we can estimate that the Monte Carlo error standard deviation is around $30/\sqrt{50000} \approx 0.13$ for the ADP simulations, and around $40/\sqrt{5000} \approx 0.6$ for the MPC simulations. For the others, we can estimate Monte Carlo error standard deviation as around 0.04 for the ADP simulations and 0.14 for the MPC simulations. Re-running the entire set of simulations, with different random number generator seed, yields numerical results entirely consistent with these estimates of Monte Carlo errors.

Simpler methods. We also evaluated simpler policies for comparison. Our first simple method was to ignore the transaction cost terms, which results in problems that can be solved exactly, as described in §5.3.3. Since the ignored transaction cost terms are nonnegative, this gives a lower bound on the performance, as well as a performance value (evaluated including the transaction costs). The other simple method is to use the (exact) value function for the quadratic example as an approximate value function. Here too we get a lower bound, which is simply the optimal performance for the quadratic example. Table 5.3 lists the lower bound obtained by ignoring the transaction cost, the performance obtained when the transaction cost is ignored, the lower bound attained by ignoring nonquadratic terms, and the performance obtained when the quadratic value function is used in ADP. We can see that the bounds

	Ignoring transaction costs		Ignoring nonquadratic terms	
Example	**lower bound**	**performance**	**lower bound**	**performance**
unconstrained	-4405	2.1×10^6	-450.1	59.2
long-only	-70.6	864.5	-450.1	162.3
leverage limit	-241.4	4525	-450.1	219.5
sector neutral	-4391	2.1×10^6	-450.1	75.6

Table 5.3: Lower bounds and performance obtained by simpler policies, obtained by ignoring transaction costs (first two columns), and by ignoring nonquadratic terms (last to columns).

obtained are quite poor, with the exception of the long-only example, which is merely bad. Note in particular that these simpler policies result in positive objective value for all examples, meaning that they lose money money on average; they are worse than not trading at all (*i.e.*, taking $u_t = 0$ for all t).

5.7.4 Simulation results and trajectories

In this section we give some more detail of the simulations, showing typical and average trajectories of various quantities over the investment period. These simulations show that the problems are not simple, that none of the stage cost terms is negligible, that the constraints are indeed active, and finally, that our ADP and MPC policies are not obvious.

Tables 5.4 and 5.5 summarize average values for various quantities over the 100 time steps and 50000 simulations under the ADP policy, for each example. The quantities are: gross cash in $\mathbf{1}^T u_t$, risk cost $\lambda(x_t^+)^T \Sigma (x_t^+)$, quadratic transaction cost $s^T u_t^2$, linear transaction cost $\kappa^T |u_t|$, and total long, short, and long-short positions. These numbers show that each of the cost terms contributes to the overall objective (except in the quadratic example). For the quadratic example, the stage cost does not include a shorting fee or an absolute value transaction cost, so those numbers are in parentheses; they are what we would have been charged, had these stage cost terms been present.

Example	$\mathbf{1}^T u_t$	$\lambda(x_t^+)^T \Sigma x_t^+$	$s^T u_t^2$	$c^T (x_t^+)_-$	$\kappa^T \lvert u_t \rvert$
quadratic	-9.00	0.77	3.73	(4.93)	(0.70)
unconstrained	-3.82	0.29	1.03	0.96	0.22
long-only	-0.86	0.32	0.08	0	0.04
leverage limit	-2.03	0.54	0.35	0.18	0.11
sector neutral	-3.71	0.19	1.02	1.07	0.23

Table 5.4: Average of various quantities over all simulations under the ADP policy.

Example	$\mathbf{1}^T (x_t^+)_+$	$\mathbf{1}^T (x_t^+)_-$	$\lVert x_t^+ \rVert_1$
quadratic	164.16	168.76	332.92
unconstrained	80.93	47.76	128 .7
long-only	25.7	0	25.7
leverage limit	52.66	12.03	64.7
sector neutral	77.2	53.5	130.7

Table 5.5: Average of various quantities over all simulations under the ADP policy.

To get an idea of the trajectories over the time horizon, figure 5.1 shows several time trajectories for the leverage limit example under the ADP policy. The lighter lines show 5 sample traces from the 50000 Monte Carlo simulations. The darker lines are the average (or in the case of the leverage, the median) over all the runs. The upper left plot shows the total value of the long and short positions (in blue and red, respectively), and the upper right plot shows the leverage ratio. We see that the leverage ratio saturates at the leverage limit, η, which is 0.3 in this case. The middle left plot shows the gross cash in, and the middle right plot shows cumulative net cash in. We can see that (on average) we put money into the portfolio over the first 12 or so steps; after that we derive income from the portfolio, and (of course) cash out on the last step. The cumulative net cash at time T is our performance, for which we have the lower bound of -87.5; this bound is shown with a red dashed line. The bottom left plot shows the total of the quadratic, absolute value, and shorting costs, which is relatively large in the beginning (while we set up our portfolio) and the end (when we cash out). The risk cost is shown in the bottom right plot. This naturally grows as we invest in the portfolio, and then shrinks as we cash out.

Figure 5.2 shows the distribution of total cost for the leverage limit example under the ADP policy over the Monte Carlo simulations. The mean of this distribution is the performance of the policy, shown as a solid blue line; the lower bound is shown in red.

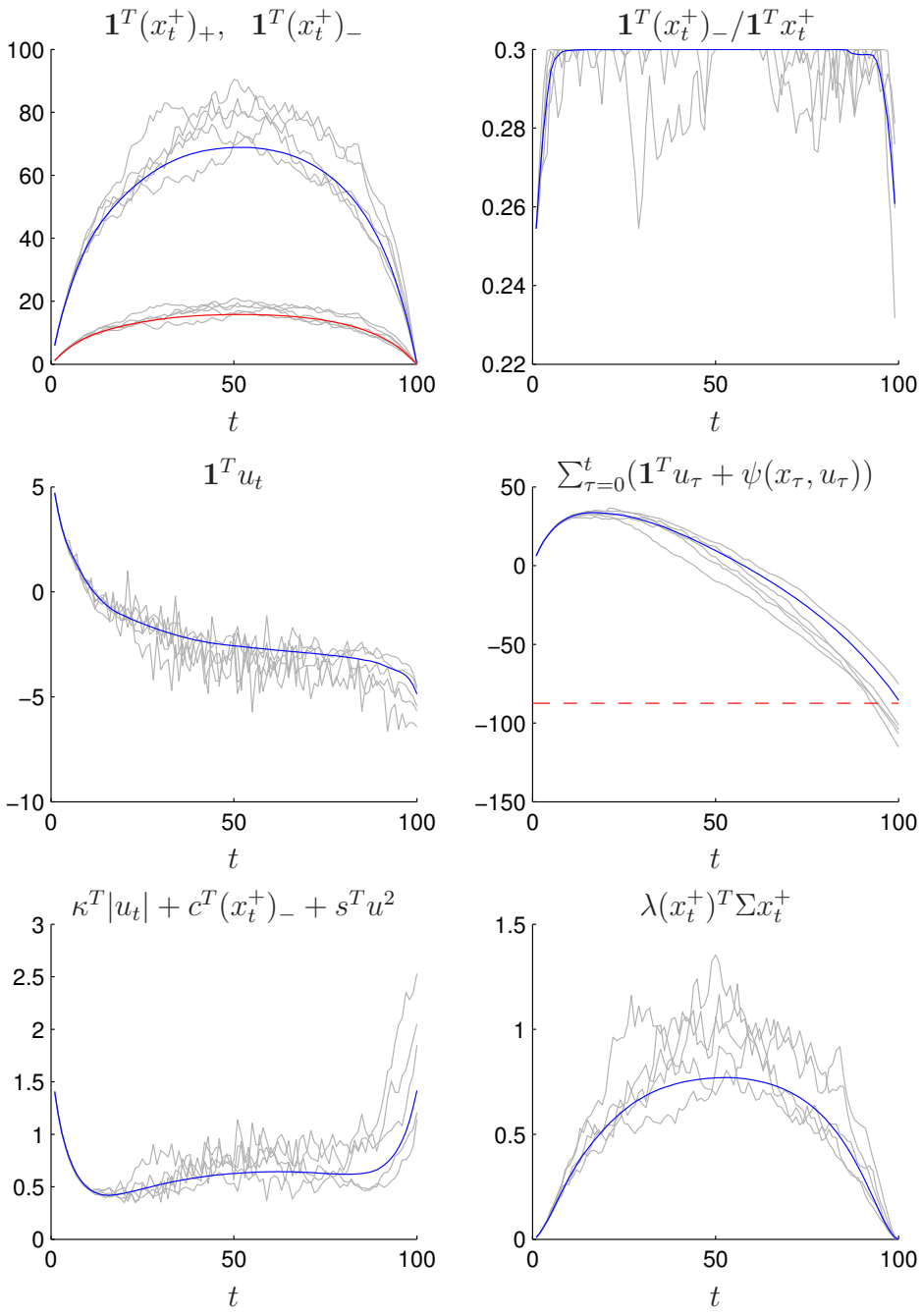

Figure 5.1: Trajectories of various quantities for the leverage limit example under the ADP policy.

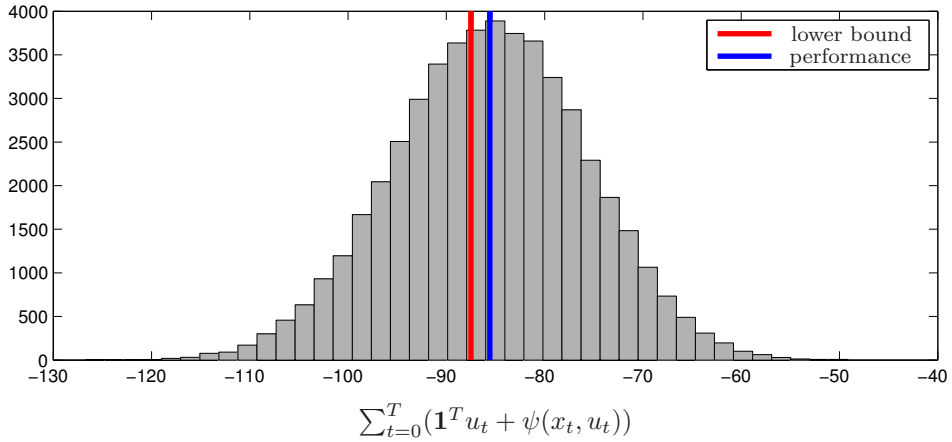

Figure 5.2: Histogram of total cost for the leverage limit example, with ADP policy.

Example	Nominal performance	Worst-case performance
quadratic	-450.0	-425.1
unconstrained	-131.9	-123.4
long-only	-41.0	-38.2
leverage limit	-85.6	-80.3
sector neutral	-118.9	-110.8

Table 5.6: ADP policy performance with perturbed return distribution.

5.7.5 Robustness to model parameters

Here we show that the performance of our ADP policies is not particularly sensitive to the return distribution parameters, even though our problem formulation does not include any explicit robustness requirement. To do this we carry out 10 additional sets of 5000 Monte Carlo simulations for each example, drawing the returns in each Monte Carlo simulation from a log-normal distribution with perturbed mean return $\overline{r}_t^{\text{pert}}$. (We can think of $\overline{r}_t$ as our estimate of the returns, and $\overline{r}_t^{\text{pert}}$ as the true mean return.)

The 10 perturbations were found as follows. We used log return mean $\mu^{\text{pert}} = \mu+\delta$, where μ is the log return mean used in development of the ADP policy, and δ_i were sampled from a uniform distribution on $[-0.003, +0.003]$. Since $\mu_i \sim \mathcal{N}(0, 0.03^2)$, this perturbation amounts to a change of log return means that ranges from around 1% to a factor of 5, including cases in which the signs of μ_i and μ_i^{pert} are different. (In other words, a few assets that are 'winners' under the assumed distribution become 'losers' under the perturbed return statistics, and vice versa.) Thus, the perturbation of the return distribution is not small.

The results are summarized in table 5.6, which gives the *worst* performance values over the 10 perturbed cases. These values are a bit worse than the nominal values, as expected, but quite respectable given the fairly large perturbation of the return distribution. (The average performance values over the 10 perturbations are very close to the nominal performance values.)

5.7.6 Robustness to return distribution

Our performance bounds and policies depend only on the first and second moments of the return distribution, whereas the performance obtained by our policies can depend on higher order moments of the return distribution. In this section we demonstrate that the performance obtained by our policies is not particularly sensitive to these higher order moments, in particular, when the returns come from a distribution with substantially larger tails than a log-normal.

We evaluated the performance of the ADP policy, with the returns sampled from a heavy-tailed distribution with identical first and second moments to the original log-normal distribution (so our numerical bounds are unchanged). We generated the new returns as

$$\log \hat{r}_t \sim \begin{cases} \mathcal{N}(\mu_1, \hat{\Sigma}) & \text{w.p. } 0.8 \\ \mathcal{N}(\mu_2, 10\hat{\Sigma}) & \text{w.p. } 0.2 \end{cases}$$

where μ_1, μ_2 and $\hat{\Sigma}$ were chosen so that the first and second moments of $\hat{r}_t$ matched those of the original distribution. Intuitively, with probability 0.2 the system experiences a significant 'event' that typically moves the asset prices much more drastically than usual.

Figure 5.3 compares the original and the heavy-tailed distribution of the returns of the first asset. The average kurtosis, or fourth standardized moment, of the returns increased from 3.1 for the original distribution to 8.8 under our new distribution, indicating a distribution with heavier tails. Table 5.7 summarizes the performance of the ADP policy under the original and new heavy-tailed distribution for our five examples. The performance of the policy under the new distribution was evaluated via Monte Carlo with 5000 samples. The performance under the heavy-tailed distribution was almost identical to, and in some cases better than, the original performance.

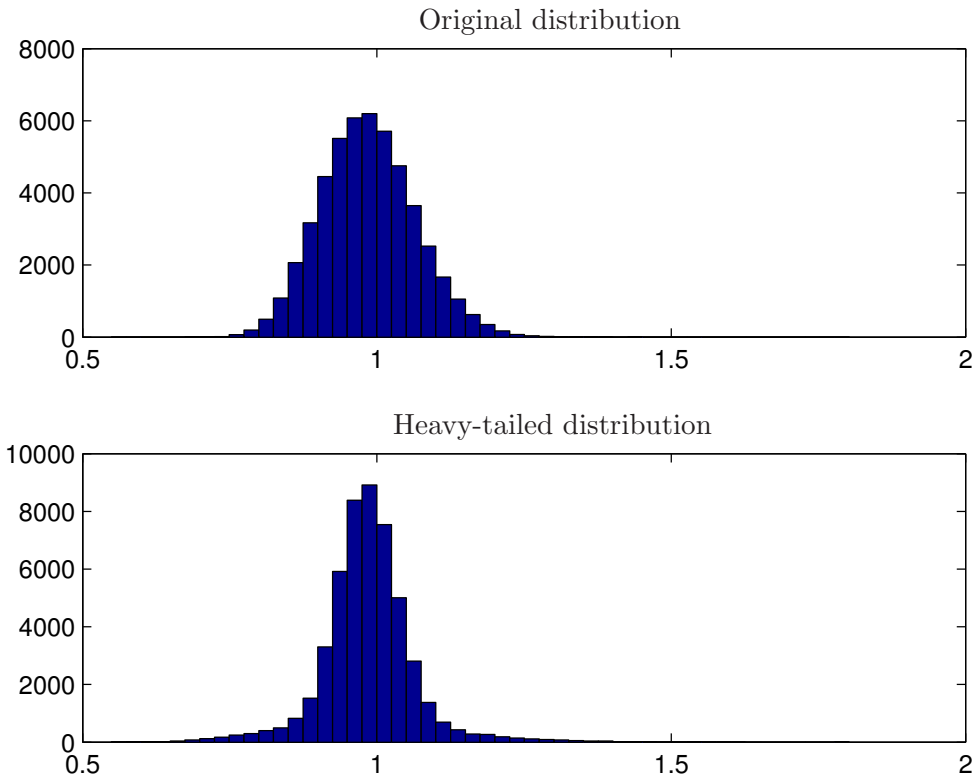

Figure 5.3: A comparison of return distributions for the first asset.

Example	**Original performance**	**Heavy-tailed return performance**
quadratic	-450.0	-450.6
unconstrained	-131.9	-132.0
long-only	-41.0	-41.0
leverage limit	-85.6	-85.5
sector neutral	-118.9	-118.8

Table 5.7: ADP policy performance with heavy-tailed return distribution.

5.8 Summary

In this chapter we formulated the multi-period portfolio optimization problem as a convex stochastic control problem with linear dynamics. Our model captures many features of real multi-period portfolio optimization problems, including limits on leverage, complex transaction costs, and time-varying return statistics. (Features that are not captured by our model include minimum allowed nonzero positions, nonconvex transaction costs, price momentum, and multi-period price impact.) Our formal model does not include the use of signals, *i.e.*, on-line updating of the problem parameters during the trading period; we discuss this below.

We used recently developed methods based on LMIs to compute a performance bound—a specific number—for any particular multi-period portfolio problem. This performance bound can be used as a yardstick against which the performance of any policy can be judged. As a by-product of the performance bound computation, we obtain a quadratic approximation of the value functions, which can be used as the approximate value functions in an ADP policy, or as a terminal cost in an MPC policy. Both the ADP and MPC policies require the solution of a convex optimization problem, typically a QP, to compute the trades to be executed in each step.

We do not currently have an upper bound on how far our performance bound will be from the performance of the ADP or MPC policies. At the moment, all we can do is evaluate the bound, and the performance of the ADP or MPC policies, for any particular multi-period portfolio optimization problem, and subtract them to obtain a gap. We would welcome a result that upper bounds the gap, based on general features of the problem (such as dimensions, or type of stage cost). Such a result would likely require further assumptions about the stage costs and constraints; moreover, it would be unlikely to be sharp enough to be of practical use.

In numerical examples, however, we have seen that the gap is very small, which means that our ADP and MPC policies (and bounds) are nearly optimal for the particular problem instances we consider. We certainly do not claim that the gap will always be small; we merely observe that it often is. Even for problem instances with a larger gap (such as, say, a factor of two), the method seems to us to be very useful:

It provides a good (if not known to be nearly optimal) trading policy, along with an upper bound on how suboptimal it can be (in this case, a factor of two).

Our methods are easily implementable, thanks to recent advances in convex optimization parser-solvers for solving complex SDPs, and code generation tools for generating extremely fast solvers for policy evaluation. In particular we rely on CVX, which calls SDP solvers SeDuMi or SDPT3, and CVXGEN, which is a code generation system for QPs. Without these tools, a user would have to manually transform the SDPs into standard form, and manually write custom solvers for policy implementation (which would take days, if not months). Instead, CVX and CVXGEN allow us to formulate, solve, and implement the policies with relatively little coding, and then carry out extensive Monte Carlo simulation in no more than a few hours, using a standard desktop computer.

We have focused specifically on QP representable problems in this chapter, but the same methods extend easily to general convex stage costs and constraints. In fact, even when the problem is nonconvex, we can still obtain suboptimal policies and performance bounds, via simple relaxations and approximations. For the single period problem, [112] outlines several approaches for handling nonconvex costs.

In our numerical examples, we have seen that our method is quite robust to uncertainties; for instance, when we do not have an accurate model for the return distribution. Robustness can also be directly incorporated into the stochastic control problem, in the computation of the bounds and approximate value functions.

There are more sophisticated methods for both finding performance bounds, as well as approximate policies, which we have not mentioned in this chapter [135, 180]. But these more advanced methods are not needed for the examples considered, since the gap between policy performance and lower bound is small.

We close by addressing again the issue of signals, which are not part of our formal model. Incorporating signals in a formal model is complicated, since it involves the joint distribution of returns and signals. Even if we did incorporate signals into our formal model, it is not clear how much practical significance the optimal performance $J^\star$ would have, since in practice we would certainly not know very much about the joint distribution of signals and returns. Incorporating signals into our formal model

would break much of our analysis, including our lower bound calculations based on Bellman inequalities.

On the other hand, it is easy to think of ways in which updated estimates of future problem data (*i.e.*, stage costs and return statistics) can be incorporated. For an MPC policy it is trivial to incorporate changing estimates of future return statistics: We simply use the most recently future return statistics estimates in the policy, as it is evaluated. This approach has no additional computational cost. For an ADP policy, we would need to re-do the policy synthesis whenever future estimates change, solving a new SDP to find new quadratic approximate value functions.

Part V

Conclusion

Chapter 6

Extensions and Conclusions

In this dissertation we developed two control policies for stochastic control problems, and an algorithm for solving optimal control problems. We then applied these techniques to a practical example. Here we offer some concluding remarks.

6.1 Approximate dynamic programming policies

In part II of this thesis we developed two suboptimal control policies for stochastic control problems that use approximate dynamic programming (ADP). In ADP we search for an easily representable surrogate value function, that provides a tractable and close-to-optimal policy. Both of the policies we developed relied on the iterated Bellman inequality, which provides a sufficient condition for lower-boundedness on the true value function: Any function that satisfies the iterated Bellman inequality belongs to a family of pointwise underestimators of the true value function. In certain cases the iterated Bellman inequality define a convex constraint on the parameters we use to define the surrogate value function.

We referred to the first policy we developed as the min-max ADP policy. The min-max ADP policy uses the pointwise supremum over the family of lower bounds as the surrogate value function. Evaluating the control policy at each iteration amounts to solving a min-max or saddle-point problem, which, for the cases we considered, was a convex optimization problem. We provided examples to show the performance of

the policy.

The second policy we developed was referred to as the iterated approximate value function policy. The generation of this policy requires two steps. In the first step we simultaneously compute a trajectory of moments of the state and action and a sequence of approximate value functions optimized to that trajectory. This pre-computation is performed off-line. The next step is to perform control using the generated sequence of approximate value functions. This amounts to a time-varying policy, even in the case where the optimal policy is time-invariant.

The evaluation of these policies requires the solution of a convex optimization problem at each iteration. There has been much progress recently on fast solvers, custom tailored solvers and solvers designed for embedded applications [52, 89, 122, 138,181,182]. With these advances our policies can realistically be applied at kilohertz rates or higher, even for large problems, making them feasible for use in a broad range of applications.

We focused primarily on the case of linear, time-invariant dynamics and convex stage-cost. However, the techniques apply much more generally. For example, the policies and bounds are readily extensible to cases with time-varying and random dynamics and stage cost, and where the dynamics function and stage cost are polynomial functions. In the case of polynomial dynamics and stage cost we can use sum-of-squares polynomials and Stengle's Positivstellensatz [163] to derive a sufficient condition on the Bellman inequality to hold (indeed the $\mathcal{S}$-procedure is a special case of a Positivstellensatz refutation).

6.2 Operator splitting for control

In part III we introduced the operator splitting for control (OSC) algorithm, which solves optimal control problems quickly and robustly, The technique we develop relies on a form of operator splitting, known as Douglas-Rachford splitting or the alternating direction method of multipliers. By choosing a particular splitting we are able to exploit the inherent structure of the problem. This results in an algorithm that alternates between solving a quadratic control problem, for which there are efficient

methods, and solving a set of single-period optimization problems, which can be done in parallel, and often have analytical solutions. Small problems are solved in a few milliseconds or less; example problems with on the order of 10000 variables are solved in tens of milliseconds. The speedup of this algorithm over other solvers, including fast custom solvers tailored for one particular problem type, range from tens to thousands. Widespread adoption of MPC in practical applications has been limited by the time required to find the control input at each iteration. The speedup that this technique obtains over other algorithms extends the range of MPC to new applications that require control at rates previously infeasible.

When the dynamics data do not change, our method yields a division-free algorithm, which can be implemented in fixed-point arithmetic. This is important for embedded applications, where floating point calculations may be impossible or prohibitively expensive, *e.g.*, on an FPGA.

There are many directions to explore to increase performance of the splitting technique. An open question is how to find the optimal choice of the step-size parameter, which we denoted ρ. In our experiments we tuned ρ off-line, which is practical for most cases. But the question remains: Is there an easy way to select ρ, either a-priori given the problem data or adaptively as the algorithm progresses? Indeed, we could replace ρ with a diagonal or full-rank square matrix, in which case the problem would be to find the coordinate transformation that provided the fastest convergence. This important question is still an area of active research [67].

Another important open question relates to so-called 'Nesterov acceleration'. In 1983 Nesterov demonstrated a simple technique to accelerate the convergence rate of first order convex optimization algorithms applied to smooth functions [130]. ADMM may be interpreted as a first order algorithm on the dual, and one would expect an analog of Nesterov acceleration to provide a significant speedup. If such a scheme existed it could dramatically reduce the number of iterations required to reach a certain accuracy with negligible additional computation. Many authors have looked at this problem with some success [72,73,75], however a general scheme is still lacking.

6.3 Multi-period portfolio optimization

In part IV we applied the techniques developed in previous parts to a practical example. Our example was that of dynamic trading a portfolio of assets in discrete periods over a finite time horizon, with arbitrary time-varying distribution of asset returns. The goal was to maximize the total expected revenue from the portfolio, while respecting constraints, such as a required terminal portfolio and leverage or risk limits. The revenue took into account the gross cash generated in trades, transaction costs, and costs associated with the positions, such as fees for holding short positions. Our model had the form of a stochastic control problem with linear dynamics and convex cost function and constraints. While this problem can be tractably solved in several special cases, such as when all costs are convex quadratic, or when there are no transaction costs, our focus was on the more general case, with nonquadratic cost terms and transaction costs. We applied the iterated AVF policy as well as MPC using OSC. We showed that both policies can be implemented quickly and obtain very close to optimal performance.

The purpose of this example was to illustrate the policies developed in previous chapters. However, the treatment brings up practical questions about using stochastic control in multi-period portfolio optimization. One such question is how best to incorporate changes in the estimates of our risk or returns. MPC can adapt immediately, but the other policies may require the pre-computation to be redone. Another open question is how best to incorporate correlations across time-periods into either policy. Augmenting the state to allow for such correlations renders the dynamics non-linear. Although the iterated AVF policy can be extended to handle polynomial dynamics, the practical performance is unknown.

Appendix A

Appendix

A.1 $\mathcal{S}$-procedure

Let f_i, $i = 0, \ldots, M$ be (not necessarily convex) quadratic functions in the variable $x \in \mathbf{R}^n$,

$$f_i(x) = (1/2) \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} P_i & p_i \\ p_i^T & q_i \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}.$$

We seek a condition on the coefficients P_0, p_0, q_0 under which f_0 is nonnegative on the set defined by the quadratic inequalities and equalities

$$\mathcal{C} = \{x \mid f_1(x) \geq 0, \ldots, f_r(x) \geq 0,\ f_{r+1}(x) = \cdots = f_M(x) = 0\},$$

i.e.,

$$f_0(x) \geq 0, \quad \forall x \in \mathcal{C}. \tag{A.1}$$

Condition (A.1) can be thought of as an infinite number of inequalities in the coefficients P, p and q (one for each $x \in \mathcal{C}$).

An obvious sufficient condition for (A.1) is the existence of $\lambda_1, \ldots, \lambda_r \in \mathbf{R}_+$, $\lambda_{r+1}, \ldots, \lambda_M \in \mathbf{R}$, for which

$$f_0(x) \geq \sum_{i=1}^{M} \lambda_i f_i(x), \quad \forall x \in \mathbf{R}^n.$$

(This follows immediately, since the righthand side is nonnegative for $x \in \mathcal{C}$.) This sufficient condition can be written as a *linear matrix inequality*

$$\left[\begin{array}{cc} P_0 & p_0 \\ p_0^T & q_0 \end{array}\right] - \sum_{i=1}^{M} \lambda_i \left[\begin{array}{cc} P_i & p_i \\ p_i^T & q_i \end{array}\right] \succeq 0,$$

in the variables P_0, p_0, q_0, and $\lambda_1, \ldots, \lambda_M$. (We also have nonnegativity constraints on $\lambda_1, \ldots, \lambda_r$.) Thus, the $\mathcal{S}$-procedure gives a sufficient condition for (A.1) that involves a single LMI in the coefficients P_0, p_0, and q_0, as well as the (multiplier) variables $\lambda_1, \ldots, \lambda_M$.

A.2 Expectation of quadratic function

In this appendix we show that when $h : \mathbf{R}^n \to \mathbf{R}$ is a convex quadratic function, say,

$$h(x) = (1/2) \left[\begin{array}{c} x \\ 1 \end{array}\right]^T \left[\begin{array}{cc} P & p \\ p^T & q \end{array}\right] \left[\begin{array}{c} x \\ 1 \end{array}\right],$$

with $P \succeq 0$, so is $g(x, u) = \mathbf{E}\, h(R_{t+1}(x + u))$. In fact, we will derive explicit formulas for the coefficients of g,

$$g(x, u) = (1/2) \left[\begin{array}{c} x \\ u \\ 1 \end{array}\right]^T \left[\begin{array}{ccc} A & B & a \\ B^T & C & c \\ a^T & c^T & d \end{array}\right] \left[\begin{array}{c} x \\ u \\ 1 \end{array}\right], \tag{A.2}$$

that only require knowledge of the mean and covariance of r_{t+1}. To show that g has this form, we first observe that

$$h(R_{t+1}(x+u)) = (1/2) \left[\begin{array}{c} x \\ u \\ 1 \end{array}\right]^T \left[\begin{array}{cc} R_{t+1} & 0 \\ R_{t+1} & 0 \\ 0 & 1 \end{array}\right] \left[\begin{array}{cc} P & p \\ p^T & q \end{array}\right] \left[\begin{array}{ccc} R_{t+1} & R_{t+1} & 0 \\ 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} x \\ u \\ 1 \end{array}\right]$$

$$= (1/2)\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} R_{t+1}PR_{t+1} & R_{t+1}PR_{t+1} & R_{t+1}p \\ R_{t+1}PR_{t+1} & R_{t+1}PR_{t+1} & R_{t+1}p \\ p^TR_{t+1} & p^TR_{t+1} & q \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}.$$

Taking expectation over R_{t+1} we see that g has the form (A.2) above, with

$$A = B = C = \mathbf{E}\, R_{t+1}PR_{t+1} = P \circ (\Sigma_{t+1} + \overline{r}_{t+1}\overline{r}_{t+1}^T),$$

$$a = c = \mathbf{E}\, R_{t+1}p = p \circ \overline{r}_{t+1}, \quad d = q,$$

where $\circ$ denotes the Hadamard (elementwise) product. This quadratic function is convex, since $P \circ (\Sigma_{t+1} + \overline{r}_{t+1}\overline{r}_{t+1}^T) \succeq 0$ (which follows from the fact that the Hadamard product of symmetric positive semidefinite matrices is positive semidefinite).

A.3 Partial minimization of quadratic function

In this appendix we show that partial minimization of a quadratic function over some of its variables, subject to linear equality constraints, results in a quadratic function in the remaining variables, whose coefficients are readily computed. In addition, we will show that the minimizer can be expressed as an affine function of the remaining variables, whose coefficients are readily computed.

Suppose $g : \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$ is a convex quadratic function,

$$g(x,u) = (1/2)\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} A & B & a \\ B^T & C & c \\ a^T & c^T & d \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}.$$

Let us consider minimization of g over u, subject to $Fx + Gu = h$, with $h \in \mathbf{R}^k$. (We assume that such a minimizer exists.) Let $h(x)$ denote the minimum value, as a function of x.

Necessary and sufficient optimality (KKT) conditions are

$$\begin{bmatrix} C & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} u \\ y \end{bmatrix} = -\left(\begin{bmatrix} c \\ h \end{bmatrix} + \begin{bmatrix} B^T \\ F \end{bmatrix} x \right),$$

where $y \in \mathbf{R}^k$ is a dual variable associated with the equality constraint. It follows that

$$\begin{bmatrix} u \\ y \end{bmatrix} = -\begin{bmatrix} C & G^T \\ G & 0 \end{bmatrix}^\dagger \left(\begin{bmatrix} c \\ h \end{bmatrix} + \begin{bmatrix} B^T \\ F \end{bmatrix} x \right)$$

is a minimizer, where $(\cdot)^\dagger$ is the Moore-Penrose pseudo-inverse. (When the KKT matrix is nonsingular, the pseudo-inverse becomes the inverse, and the minimizer is unique.) This shows that u (a minimizer, or the minimizer when it is unique) is an affine function of x, which we can write as

$$u = Jx + k.$$

Substituting this expression into g we find that h is (convex) quadratic,

$$h(x) = (1/2) \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} I & 0 \\ J & k \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} A & B & a \\ B^T & C & c \\ a^T & c^T & d \end{bmatrix} \begin{bmatrix} I & 0 \\ J & k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}.$$

A.4 LMI sufficient condition for Bellman inequality

We can write a single Bellman inequality in the form

$$V \leq \mathcal{T}V^+,$$

which equivalent to

$$V(x) \leq \mathbf{1}^T u + \psi(x,u) + \mathbf{E}\, V^+(R(x+u)), \quad (x+u) \in \mathcal{C}.$$

(Here we drop all time indices for notational simplicity.) We assume that ψ is convex quadratic,

$$\psi(x,u) = (1/2) \begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} A & B & a \\ B^T & C & c \\ a^T & c^T & d \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix},$$

with

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succeq 0.$$

The constraint set $\mathcal{C}$ defined by quadratic equalities and inequalities,

$$\mathcal{C} = \{x \mid g_1(x) \leq 0, \ldots, g_r(x) \leq 0,\ g_{r+1}(x) = \cdots = g_M(x) = 0\},$$

where

$$g_i(x) = (1/2) \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} G_i & f_i \\ f_i^T & h_i \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}, \quad i = 1, \ldots, M.$$

We assume V and V^+ are convex quadratic,

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

with

$$P \succeq 0, \qquad P^+ \succeq 0.$$

Using the result in appendix A.2, we can write the Bellman inequality as

$$(1/2)\begin{bmatrix} x \\ u \\ 1 \end{bmatrix}^T \begin{bmatrix} \tilde{A} & \tilde{B} & \tilde{a} \\ \tilde{B}^T & \tilde{C} & \tilde{c} \\ \tilde{a}^T & \tilde{c}^T & \tilde{d} \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} \geq 0, \quad (x+u) \in \mathcal{C},$$

where

$$\begin{aligned} \tilde{A} &= A + P^+ \circ (\Sigma + \overline{r}\overline{r}^T) - P \\ \tilde{B} &= B + P^+ \circ (\Sigma + \overline{r}\overline{r}^T) \\ \tilde{C} &= C + P^+ \circ (\Sigma + \overline{r}\overline{r}^T) \\ \tilde{a} &= a + p^+ \circ \overline{r} - p \\ \tilde{c} &= c + \mathbf{1} + p^+ \circ \overline{r} \\ \tilde{d} &= q^+ + d - q. \end{aligned}$$

We note that (the coefficients of) $\tilde{A}$, $\tilde{B}$, $\tilde{C}$, $\tilde{a}$, $\tilde{c}$ and $\tilde{d}$ are affine functions of (the coefficients of) V and V^+.

Applying the $\mathcal{S}$-procedure (appendix A.1), a sufficient condition for the Bellman inequality is the existence of $\lambda_1, \ldots, \lambda_r \in \mathbf{R}_+$, and arbitrary $\lambda_{r+1}, \ldots, \lambda_M \in \mathbf{R}$ for which

$$\begin{bmatrix} \tilde{A} & \tilde{B} & \tilde{a} \\ \tilde{B}^T & \tilde{C} & \tilde{c} \\ \tilde{a}^T & \tilde{c}^T & \tilde{d} \end{bmatrix} \succeq \sum_{i=1}^{M} \lambda_i \begin{bmatrix} G_i & G_i & f_i \\ G_i & G_i & f_i \\ f_i^T & f_i^T & h_i \end{bmatrix}. \tag{A.3}$$

This is an LMI in the coefficients P, p, q, P^+, p^+, q^+, and $\lambda_1, \ldots, \lambda_M$.

A.5 No-trade region

We define the *no-trade region* for a policy ϕ_t in period t as the set of portfolio values in which the policy does not trade:

$$\mathcal{N}_t = \{x \mid \phi_t(x) = 0\}.$$

In this appendix we observe that for the quadratic ADP policy, the presence of linear transaction cost terms in the stage cost lead to $\mathcal{N}_t$ typically having nonempty interior.

For the ADP policy (5.13), $\mathcal{N}_t$ is the set of portfolios x for which $u = 0$ minimizes

$$\ell_t(x, u) + \mathbf{E}\,\hat{V}_t(R_{t+1}(x + u)).$$

When $\hat{V}_t$ is quadratic, the second term is convex quadratic, so we can write it as

$$(1/2)(x + u)^T P_t(x + u) + p_t^T(x + u)$$

where $P_t \succeq 0$. The necessary and sufficient condition for $u = 0$ to minimize the function above is then

$$0 \in \partial_u \ell_t(x, 0) + P_t x + p_t,$$

where ∂_u denotes the subdifferential with respect to u.

Now suppose that

$$\ell_t(x, u) = \tilde{\ell}_t(x, u) + \kappa_+^T(u)_+ + \kappa_-^T(u)_-,$$

where $\tilde{\ell}_t$ is differentiable in u for fixed x, $\kappa_+ > 0$, $\kappa_- > 0$, and (for simplicity) $\mathcal{C}_t = \mathbf{R}^n$. In other words, the stage cost contains a linear transaction cost term (with nonzero cost rates). The subdifferential of the linear transaction cost term at $u = 0$ is the rectangle in $\mathbf{R}^n$ given by $[-\kappa_-, \kappa_+]$, so the no-trade region is characterized by

$$-\left(\nabla_u \tilde{\ell}_t(x, 0) + P_t x + p_t\right) \in [-\kappa_-, \kappa_+]. \tag{A.4}$$

In this case, there is a simple interpretation for the no-trade region. We interpret the

lefthand side as the vector of marginal values of trading the assets in period t; we do not trade when the marginal values are smaller than the linear transaction cost rates.

When the lefthand side above (*i.e.*, the marginal utility) is a continuous function of x, and its range intersects the open rectangle $(-\kappa_-, \kappa_+)$, we conclude that $\mathcal{N}_t$ has non-empty interior.

Short-circuiting. We can use the characterization (A.4) of the no-trade region to (sometimes) speed up the evaluation of $\phi_t(x_t)$. We first check if (A.4) holds; if so, we return $u_t = 0$ as the optimal trade vector. If not, we solve the convex optimization problem required to determine the (nonzero) value of u_t. This trick does not speed up the worst-case time required to evaluate the policy, but it can reduce the average time to evaluate it, if in many periods the policy does not trade. This can be useful in simulations, for example.

Bibliography

[1] J. Adda and R. Cooper. *Dynamic economics: Quantitative methods and applications*. MIT Press, 2003.

[2] M. Åkerblad and A. Hansson. Efficient solution of second order cone program for model predictive control. *International Journal of Control*, 77(1):55–77, January 2004.

[3] P. Algoet and T. Cover. Asymptotic optimality and asymptotic equipartition properties of log-optimum investment. *The Annals of Probability*, 16(2):876–898, April 1988.

[4] R. Almgren and N. Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3(2):5–39, December 2001.

[5] P. Amestoy, T. Davis, and I. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30(3):381–388, 2004.

[6] M. Annergren, A. Hansson, and B. Wahlberg. An ADMM algorithm for solving ℓ_1 regularized MPC, 2012. Manuscript.

[7] R. Bartlett and L. Biegler. QPSchur: a dual, active set, Schur complement method for large-scale and structured convex quadratic programming algorithm. *Optimization and Engineering*, 7:5–32, 2006.

[8] S. Basak and A. Shapiro. Value-at-risk-based risk management: Optimal policies and asset prices. *Review of Financial Studies*, 14(2):371–405, February 2001.

[9] R. Bellman. *Dynamic Programming*. Dover Publications, 1957.

[10] A. Bemporad and C. Filippi. Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming. *Journal of Optimization Theory and Applications*, 117(1):9–38, November 2004.

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

[12] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009.

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

[14] D. Bertsekas. Dynamic programming and suboptimal control: A survey from ADP to MPC. *European Journal of Control*, 11(4–5):310–334, October 2005.

[15] D. Bertsekas. Rollout algorithms for constrained dynamic programming. Technical report, Laboratory for Information and Decision Systems, MIT, 2005.

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

[17] D. Bertsekas, A. Nedić, and A. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, 2003.

[18] D. Bertsekas and S. Shreve. *Stochastic optimal control: The discrete-time case*. Athena Scientific, 1996.

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

[20] D. Bertsimas and A. Lo. Optimal control of execution costs. *Journal of Financial Markets*, 1(1):1–50, April 1998.

[21] M. Best. An algorithm for the solution of the parametric quadratic programming problem. In *Applied Mathematics and Parallel Computing*, pages 57–76. Physica-Verlag: Heidelberg, 1996.

[22] J. Betts. *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*. SIAM, 2010.

[23] J. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, 1997.

[24] J. Borwein and A. Lewis. *Convex Analysis and Nonlinear Optimization: Theory and Examples*. Springer, 2000.

[25] S. Boyd and C. Barratt. *Linear controller design: Limits of performance*. Prentice-Hall, 1991.

[26] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994.

[27] S. Boyd, M. Mueller, B. O'Donoghue, and Y. Wang. Performance bounds and suboptimal policies for multi-period investment. `http://www.stanford.edu/~boyd/papers/port_opt_bound.html`, 2012. Manuscript.

[28] 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.

[29] 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.

[30] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, September 2004.

[31] L. Breiman. Optimal gambling systems for favorable games. In *Proc. 4th Berkeley Symp. Math. Statist. Probab.*, volume 1, pages 65–78, 1961.

[32] D. Brown and J. Smith. Dynamic portfolio optimization with transaction costs: Heuristics and dual bounds. *Management Science*, 57(10):1752–1770, October 2011.

[33] J. Bunch and B. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655, December 1971.

[34] G. Calafiore. Multi-period portfolio optimization with linear control policies. *Automatica*, 44(10):2463–2473, October 2008.

[35] G. Calafiore. An affine control method for optimal dynamic asset allocation with transaction costs. *SIAM J. Control Optim.*, 48(4):2254–2274, June 2009.

[36] G. Calafiore. Random convex programs. *SIAM J. Optim.*, 20(6):3427–3464, December 2010.

[37] E. Camacho and C. Bordons. *Model Predictive Control*. Springer-Verlag, 2004.

[38] D. Cohn. *Measure Theory*. Birkhäuser, 1997.

[39] P. Combettes. The convex feasibility problem in image recovery. *Advances in Imaging and Electron Physics*, 95:155–270, 1996.

[40] P. Combettes and J. Pesquet. Proximal splitting methods in signal processing. *arXiv:0912.3522*, 2009.

[41] G. Constantinides. Multiperiod consumption and investment behavior with convex transaction costs. *Management Science*, 25(11):1127–1137, November 1979.

[42] T. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, January 1991.

[43] J. Cvitanić and I. Karatzas. Hedging and portfolio optimization under transaction costs: A Martingale approach. *Mathematical Finance*, 6(2):133–165, April 1996.

[44] M. Davis and A. Norman. Portfolio selection with transaction costs. *Mathematics of Operations Research*, 15(4):676–713, November 1990.

[45] T. Davis. Algorithm 849: A concise sparse Cholesky factorization package. *ACM Transactions on Mathematical Software*, 31(4):587–591, December 2005.

[46] T. Davis. *Direct Methods for Sparse Linear Systems*. SIAM Fundamentals of Algorithms. SIAM, 2006.

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

[48] D. De Farias and B. Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, August 2004.

[49] J. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[50] E. Denardo. *Dynamic Programming: Models and Applications*. Prentice-Hall, 1982.

[51] V. Desai, V. Farias, and C. Moallemi. Pathwise optimization for optimal stopping problems. *Management Science*, June 2012.

[52] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, and C. Jones. Efficient interior point methods for multistage problems arising in receding horizon control. In *Proceedings of the 51st IEEE Conference on Decision and Control*, December 2012.

[53] I. Duff, A. Erisman, and J. Reid. *Direct methods for sparse matrices*. Oxford University Press, USA, 1989.

[54] B. Dumas and E. Luciano. An exact solution to a dynamic portfolio choice problem under transaction costs. *The Journal of Finance*, 46(2):577–595, June 1991.

[55] J. Dupačová, J. Hurt, and J. Štěpán. *Stochastic modeling in economics and finance*. Applied optimization. Kluwer Academic Publishers, 2002.

[56] J. Eckstein. Parallel alternating direction multiplier decomposition of convex programs. *Journal of Optimization Theory and Applications*, 80(1):39–62, 1994.

[57] J. Eckstein and D. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[58] J. Eckstein and M. Ferris. Operator-splitting methods for monotone affine variational inequalities, with a parallel application to optimal control. *INFORMS Journal on Computing*, 10(2):218–235, 1998.

[59] L. El Ghaoui and S. Niculescu. *Advances in linear matrix inequality methods in control*. Advances in design and control. SIAM, 2000.

[60] H. Ferreau. An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control. Master's thesis, University of Heidelberg, 2006.

[61] H. Ferreau, H. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

[62] D. Gabay. Applications of the method of multipliers to variational inequalities. In M. Fortin and R. Glowinski, editors, *Augmented Lagrangian Methods: Applications to the Solution of Boundary-Value Problems*. North-Holland: Amsterdam, 1983.

[63] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximations. *Computers and Mathematics with Applications*, 2:17–40, 1976.

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

[65] N. Gârleanu and L. Pedersen. Dynamic trading with predictable returns and transaction costs. `http://pages.stern.nyu.edu/~lpederse/papers/DynamicTrading.pdf`, 2011. Manuscript.

[66] A. Gattami. Generalized linear quadratic control theory. In *Proceedings of the 45th Conference on Decision and Control*, pages 1510–1514, December 2006.

[67] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson. On the optimal step-size selection for the alternating direction method of multipliers. In *Proceedings IFAC Workshop on Distributed Estimation and Control in Networked Systems (NecSys)*, September 2012.

[68] P. Gill, N. Gould, W. Murray, M. Saunders, and M. Wright. A weighted Gram-Schmidt method for convex quadratic programming. *Mathematical Programming*, 30:176–195, 1984.

[69] P. Gill, W. Murray, M. Saunders, and M. Wright. Procedures for optimization problems with a mixture of bounds and general linear constraints. *ACM Transactions on Mathematical Software*, 10(3):282–298, 1984.

[70] R. Glowinski and A. Marrocco. Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualité, d'une classe de problems de Dirichlet non lineares. *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, 9:41–76, 1975.

[71] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1983.

[72] D. Goldfarb and S. Ma. Fast multiple splitting algorithms for convex optimization, 2012. Manuscript.

[73] D. Goldfarb, S. Ma, and K. Scheinberg. Fast alternating linearization methods for minimizing the sum of two convex functions. *UCLA CAM technical report, 10-02*, 2010.

[74] D. Goldsmith. Transactions costs and the theory of portfolio selection. *Journal of Finance*, 31(4):1127–39, September 1976.

[75] T. Goldstein, B. O'Donoghue, and S. Setzer. Fast alternating direction optimization methods, 2012. Manuscript.

[76] G. Goodwin, M. Seron, and J. De Doná. *Constrained control and estimation*. Springer, 2005.

[77] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[78] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com/cvx`, April 2011.

[79] A. Hansson. A primal-dual interior-point method for robust optimal control of linear discrete-time systems. *IEEE Transactions on Automatic Control*, 45(9):1639–1655, 2000.

[80] A. Hansson and S. Boyd. Robust optimal control of linear discrete-time systems using primal-dual interior-point methods. In *Proceedings of the American Control Conference*, volume 1, pages 183–187, 1998.

[81] E. Hartley, J. Jerez, A. Suardi, J. Maciejowski, E. Kerrigan, and G. Constantinides. Predictive control of a Boeing 747 aircraft using an FPGA. In *Proceedings of the IFAC NMPC '12 Conference*, August 2012.

[82] M. Haugh, L. Kogan, and J. Wang. Evaluating portfolio policies: A duality approach. *Operations Research*, 54(3):405–418, June 2006.

[83] B. He and X. Yuan. On non-ergodic convergence rate of Douglas-Rachford alternating direction method of multipliers, 2012. Manuscript.

[84] B. He and X. Yuan. On the $O(1/n)$ convergence rate of the Douglas-Rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

[85] O. Hernandez-Lerma and J. Lasserre. Linear programming approximations for Markov control processes in metric spaces. *Acta Applicandae Mathematicae*, 51:123–139, 1998.

[86] F. Herzog, G. Dondi, and H. Geering. Stochastic model predictive control and portfolio optimization. *International Journal of Theoretical and Applied Finance*, 10(2):203–233, 2007.

[87] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

[88] J. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer-Verlag, 2004.

[89] B. Houska, H. Ferreau, and M. Diehl. ACADO toolkit-An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32:298–312, 2011.

[90] G. Iyengar and T. Cover. Growth optimal investment in horse race markets with costs. *IEEE Transactions on Information Theory*, 46(7):2675–2683, November 2000.

[91] J. Jerez, G. Constantinides, and E. Kerrigan. An FPGA implementation of a sparse quadratic programming solver for constrained predictive control. In *Proceedings ACM Symp. Field Programmable Gate Arrays*, pages 209–218, 2011.

[92] J. Jerez, G. Constantinides, and E. Kerrigan. Towards a fixed point QP solver for predictive control. In *Proceedings of the 51st IEEE Conference on Decision and Control*, December 2012.

[93] J. Jerez, G. Constantinides, E. Kerrigan, and K. Ling. Parallel MPC for real-time FPGA-based implementation. In *Proceedings of the 18th IFAC World Congress*, pages 1338 – 1343, August 2011.

[94] K. Judd. *Numerical Methods in Economics*. The MIT Press, 1998.

[95] Y. Kabanov. Hedging and liquidation under transaction costs in currency markets. *Finance and Stochastics*, 3(2):237–248, 1999.

[96] Y. Kabanov, M. Rásonyi, and C. Stricker. On the closedness of sums of convex cones in L^0 and the robust no-arbitrage property. *Finance and Stochastics*, 7(3):403–411, 2003.

[97] Y. Kabanov and C. Stricker. The Harrison–Pliska arbitrage pricing theorem under transaction costs. *Journal of Mathematical Economics*, 35(2):185–196, 2001.

[98] R. Kalman. When is a linear control system optimal? *Journal of Basic Engineering*, 86(1):1–10, 1964.

[99] J. Kelly. A new interpretation of information rate. *IRE Transactions on Information Theory*, 2(3):185–189, September 1956.

[100] M. Kögel and R. Findeisen. A fast gradient method for embedded linear predictive control. In *Proceedings of the 18th IFAC World Conference*, pages 1362–1367, August 2011.

[101] M. Kögel and R. Findeisen. Fast predictive control of linear systems combining Nesterov's gradient method and the method of multipliers. In *Proceedings of the 50th IEEE Conference on Decision and Control*, pages 501–506, December 2011.

[102] M. Kögel and R. Findeisen. Fast predictive control of linear, time-invariant systems using an algorithm based on the fast gradient method and augmented lagrange multipliers. In *Proceedings of the 2011 IEEE International Conference on Control Applications*, pages 780–785, September 2011.

[103] P. Krokhmal, J. Palmquist, and S. Uryasev. Portfolio optimization with conditional value-at-risk objective and constraints. *Journal of Risk*, 4(2):11–27, 2002.

[104] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004.

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

[106] M. Lau, S. Yue, K. Ling, and J. Maciejowski. A comparison of interior point and active set methods for FPGA implementation of model predictive control. In *Proceedings of the ECC 2009*, pages 156 – 161, August 2009.

[107] X. Li, X. Zhou, and A. Lim. Dynamic mean-variance portfolio selection with no-shorting constraints. *SIAM J. Control Optim.*, 40(5):1540–1555, January 2002.

[108] F. Lin, M. Fardad, and M. Jovanovic. Design of optimal sparse feedback gains via the alternating direction method of multipliers. In *Proceedings of the 2012 American Control Conference*, pages 4765–4770, June 2012.

[109] B. Lincoln and A. Rantzer. Relaxing dynamic programming. *IEEE Transactions on Automatic Control*, 51(8):1249–1260, August 2006.

[110] K. Ling, B. Wu, and J. Maciejowski. Embedded model predictive control (MPC) using a FPGA. In *Proceedings of the 17th IFAC World Congress*, pages 15250–15255, July 2008.

[111] A. Lo and M. Mueller. Warning: Physics envy may be hazardous to your wealth!, 2010.

[112] M. Lobo, M. Fazel, and S. Boyd. Portfolio optimization with linear and fixed transaction costs. *Annals of Operations Research*, 152(1):341–365, July 2007.

[113] M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284:193–228, November 1998.

[114] J. Löfberg. YALMIP: A toolbox for modeling and optimization in Matlab. In *Proceedings of the CACSD Conference*, 2004.

[115] S. Longo, E. Kerrigan, K. Ling, and G. Constantinides. Parallel move blocking model predictive control. In *CDC-ECE*, pages 1239–1244, 2011.

[116] D. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. Springer, 2008.

[117] J. Maciejowski. *Predictive Control with Constraints*. Prentice-Hall, 2002.

[118] A. Manne. Linear programming and sequential decisions. *Management Science*, 6(3):259–267, April 1960.

[119] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, March 1952.

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

[121] J. 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.

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

[123] 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.

[124] J. Mattingley, Y. Wang, and S. Boyd. Receding horizon control: Automatic generation of high-speed solvers. *IEEE Control Systems Magazine*, 31(3):52–65, June 2011.

[125] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, June 2000.

[126] R. Merton. Lifetime portfolio selection under uncertainty: The continuous-time case. *The Review of Economics and Statistics*, 51(3):247–257, August 1969.

[127] S. Meyn and R. Tweedie. *Markov chains and stochastic stability*. Communications and control engineering. Cambridge University Press, 2009.

[128] C. Moallemi and M. Sağlam. Dynamic portfolio choice with linear rebalancing rules. `http://www.columbia.edu/~ms3760/linear.pdf`, 2012. Manuscript.

[129] J. Mossin. Optimal multiperiod portfolio policies. *The Journal of Business*, 41(2):215–229, April 1968.

[130] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[131] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Kluwer Academic Publishers, 2004.

[132] Y. Nesterov. Gradient methods for minimizing composite objective function. `http://www.ecore.be/DPs/dp_1191313936.pdf`, 2007. CORE discussion paper.

[133] Y. Nesterov and A. Nemirovsky. *Interior-Point Polynomial Methods in Convex Programming*. SIAM, 1994.

[134] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 1999.

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

[136] T. Ohtsuka. A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4):563–574, April 2004.

[137] T. Ohtsuka and A. Kodama. Automatic code generation system for nonlinear receding horizon control. *Transactions of the Society of Instrument and Control Engineers*, 38(7):617–623, July 2002.

[138] B. ODonoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. `http://www.stanford.edu/~boyd/papers/oper_splt_ctrl.html`, 2012. Manuscript.

[139] B. ODonoghue, Y. Wang, and S. Boyd. Iterated approximate value functions, 2012. Manuscript.

[140] G. Pannocchia, J. Rawlings, and S. Wright. Fast, large-scale model predictive control by partial enumeration. *Automatica*, 43(5):852–860, May 2006.

[141] H. Pham. *Continuous-time stochastic control and optimization with financial applications*. Spinger Series in Stochastic modelling and applied probability. Springer, 2009.

[142] F. Potra and S. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1–2):281–302, 2000.

[143] W. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. J. Wiley & Sons, 2007.

[144] A. Prékopa. *Stochastic programming*. Mathematics and its applications. Kluwer Academic Publishers, 1995.

[145] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[146] M. Rami and L. El Ghaoui. LMI optimization for nonstandard Riccati equations arising in stochastic control. *IEEE Trans. Autom. Control*, 41(11):1666–1671, November 1996.

[147] C. Rao, S. Wright, and J. Rawlings. Application of interior point methods to model predictive control. *Journal of optimization theory and applications*, 99(3):723–757, November 2004.

[148] S. Richter, C. Jones, and M. Morari. Real-time input-constrained MPC using fast gradient methods. In *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 7387 –7393, December 2009.

[149] S. Richter, C. Jones, and M. Morari. Computational complexity certification for real-time MPC with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6), June 2012.

[150] S. Richter, S. Marièthoz, and M. Morari. High-speed online MPC based on a fast gradient method applied to power converter control. In *Proceedings of the 2010 American Control Conference*, pages 4737 –4743, July 2010.

[151] R. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[152] R. Rockafellar and S. Uryasev. Optimization of conditional value-at-risk. *The Journal of Risk*, 2(3):21–42, 2000.

[153] S. Ross. *Introduction to Stochastic Dynamic Programming: Probability and Mathematical*. Academic Press, 1983.

[154] A Ruszczynski. *Nonlinear Optimization*. Princeton University Press, 2006.

[155] P. Samuelson. Lifetime portfolio selection by dynamic stochastic programming. *Review of Economics and Statistics*, 51(3):239–246, August 1969.

[156] M. Saunders. Cholesky-based methods for sparse least squares: The benefits of regularization. In L. Adams and J. L. Nazareth, editors, *Linear and Nonlinear Conjugate Gradient-Related Methods*, pages 92–100. SIAM, Philadelphia, 1996. Proceedings of AMS-IMS-SIAM Joint Summer Research Conference.

[157] C. Savorgnan, J. Lasserre, and M. Diehl. Discrete-time stochastic optimal control via occupation measures and moment relaxations. In *Proc. 48th IEEE Conference on Decision and Control*, pages 519–524, December 2009.

[158] W. Schachermayer. The fundamental theorem of asset pricing under proportional transaction costs in finite discrete time. *Mathematical Finance*, 14(1):19–48, January 2004.

[159] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision process. *Journal of mathematical analysis and applications*, 110(2):568–582, September 1985.

[160] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on stochastic programming: Modeling and theory*. MPS-SIAM series on optimization. SIAM, 2009.

[161] J. Skaf and S. Boyd. Multi-period portfolio optimization with constraints and transaction costs. `http://www.stanford.edu/~boyd/papers/dyn_port_opt.html`, 2008. Manuscript.

[162] R. Soeterboek. *Predictive Control: A Unified Approach*. Prentice Hall, 1992.

[163] G. Stengle. A nullstellensatz and a positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207:87–97, 1974.

[164] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11:625–653, 1999. Software available at `http://sedumi.ie.lehigh.edu/`.

[165] M. Sznaier, R. Suarez, and J. Cloutier. Suboptimal control of constrained nonlinear systems via receding horizon constrained control Lyapunov functions. *International Journal on Robust and Nonlinear Control*, 13(3–4):247–259, March 2003.

[166] C. Tapiero. *Applied stochastic models and control for finance and insurance*. Kluwer, 1998.

[167] P. Tøndel, T. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. In *IEEE Conference on Decision and Control*, pages 1199–1204, 2001.

[168] J. Tobin. The theory of portfolio selection. In F. Hahn and F. Brechling, editors, *The Theory of Interest Rates*. Macmillan, 1965.

[169] K. Toh, M. Todd, and R. Tütüncü. SDPT3—A Matlab software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, 11(1):545–581, 1999.

[170] K. Toh, M. Todd, and R. Tütüncü. SDPT3: A Matlab software package for semidefinite programming. *Optimization Methods and Software*, 11(12):545–581, 1999.

[171] U. Topcu, G. Calafiore, and L. El Ghaoui. Multistage investments with recourse: A single-asset case with transaction costs. In *Proceedings of the 47th Conference on Decision and Control*, pages 2398–2403, Cancun, Mexico, 2008.

[172] L. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[173] R. Tütüncü, K. Toh, and M. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming*, 95(2):189–217, 2003.

[174] L. Vandenberghe. Lecture on proximal gradient method. From `http://www.ee.ucla.edu/~vandenbe/shortcourses/dtu-10/lecture3.pdf`, 2010.

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

[176] R. Vanderbei. Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113, 1995.

[177] P. Vouzis, L. Bleris, M. Arnold, and M. Kothare. A system-on-a-chip implementation for embedded real-time model predictive control. *IEEE Trans. Control Syst. Technol.*, 17(5):1006–1017, 2009.

[178] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang. An ADMM algorithm for a class of total variation regularized estimation problems. In *To appear, Proceedings 16th IFAC Symposium on System Identification*, July 2012.

[179] Y. Wang and S. Boyd. Performance bounds for linear stochastic control. *Systems & Control Letters*, 58(3):178–182, March 2009.

[180] Y. Wang and S. Boyd. Approximate dynamic programming via iterated Bellman inequalities. `http://www.stanford.edu/~boyd/papers/adp_iter_bellman.html`, 2010. Manuscript.

[181] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, March 2010.

[182] Y. Wang and S. Boyd. Fast evaluation of quadratic control-Lyapunov policy. *IEEE Transactions on Control Systems Technology*, 19(4):939–946, July 2011.

[183] Y. Wang and S. Boyd. Performance bounds and suboptimal policies for linear stochastic control via LMIs. *International Journal of Robust and Nonlinear Control*, 21(14):1710–1728, September 2011.

[184] P. Whittle. *Optimization Over Time: Dynamic Programming and Stochastic Control*. John Wiley & Sons, 1982.

[185] A. Wills, G. Knagge, and B. Ninness. Fast linear model predictive control via custom integrated circuit architecture. *IEEE Trans. Contr. Sys. Techn.*, 20(1):59–71, 2012.

[186] A. Wills, G. Knagge, and B. Ninness. Fast linear model predictive control via custom integrated circuit architecture. *IEEE Trans. Control Syst. Technol.*, 20(1):59–71, 2012.

[187] S. Wright. Interior-point method for optimal control of discrete-time systems. *J. Optim. Theory Appl.*, 77:161–187, 1993.

[188] S. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.

[189] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. Wiley, 1997.

[190] X. Zhou and D. Li. Continuous-time mean-variance portfolio selection: A stochastic LQ framework. *Applied Mathematics & Optimization*, 42(1):19–33, 2000.

[191] W. Ziemba and R. Vickson. *Stochastic Optimization Models in Finance*. World Scientific, 2006.