Annual Reviews in Control 56 (2023) 100917

Contents lists available at ScienceDirect

Annual Reviews in Control

journal homepage: www.elsevier.com/locate/arcontrol

Full length article

Value-gradient iteration with quadratic approximate value functions

Alan Yang *, Stephen Boyd

Department of Electrical Engineering, Stanford University, Stanford, CA, USA

ARTICLE INFO

Keywords:
Approximate dynamic programming
Stochastic control
Convex optimization
Value function approximation
Supply chain optimization

ABSTRACT

We propose a method for designing policies for convex stochastic control problems characterized by random linear dynamics and convex stage cost. We consider policies that employ quadratic approximate value functions as a substitute for the true value function. Evaluating the associated control policy involves solving a convex problem, typically a quadratic program, which can be carried out reliably in real-time. Such policies often perform well even when the approximate value function is not a particularly good approximation of the true value function. We propose value-gradient iteration, which fits the gradient of value function, with regularization that can include constraints reflecting known bounds on the true value function. Our value-gradient iteration method can yield a good approximate value function with few samples, and little hyperparameter tuning. We find that the method can find a good policy with computational effort comparable to that required to just evaluate a control policy via simulation.

1. Introduction

We consider convex approximate dynamic programming (ADP) policies for convex stochastic control problems, which involve systems with known random linear dynamics and convex stage costs. Evaluating an ADP policy reduces to solving a convex optimization problem involving a convex approximate value function. We focus on fitting quadratic approximate value functions, and refer to the associated policies as quadratic approximate dynamic programming (QADP) policies. While QADP policies are optimal for problems with convex quadratic stage cost (Barratt & Boyd, 2021; Bertsekas, 2012), they can also serve as effective heuristics for other problem types. It has been observed that ADP policies can perform well even when using imperfect approximations of the true value function (Bertsekas, 2019; Keshavarz & Boyd, 2014; Powell, 2007).

In this work, we propose an approximate value iteration method for finding quadratic approximate value functions for convex stochastic control problems, which we refer to as *value-gradient iteration* (VGI). In principle, an optimal value function may be found by iterating the Bellman operator, which maps real-valued functions on the state space to real-valued functions on the state space (Bellman, 1954). Since it is not possible in general to exactly represent functions on $\mathbf{R}^n$, we incorporate a function approximation step after each application of the Bellman operator, a general approach called fitted value iteration (FVI). In our proposed VGI, instead of directly fitting the value function, we fit the gradient of the value function with respect to the state vector.

It is sufficient to approximate the gradient since constant offsets in the value function have no impact on the associated ADP policy. In addition, the gradient of the value function carries more information than the value function itself (Dayan & Singh, 1995; Fairbank, 2008). If the gradient is well approximated at a set of states, then the value function is also well approximated locally around those states, up to an additive constant which does not affect the policy. However, having a good approximation of only the value at a set of states does not imply that the value function is well approximated locally around those states.

Most importantly, VGI is practical to implement for QADP. We show that, when it exists, the gradient of the Bellman operator applied to a convex quadratic function can be obtained at any state by evaluating a particular optimal dual variable associated with the QADP policy. Since the gradient of a convex quadratic is an affine function, in each iteration we fit an affine function to a set of pairs of states and value-gradients. This fitting problem is a convex optimization problem. Therefore, VGI involves solving a sequence of convex optimization problems, which can be carried out reliably.

We also consider several techniques for enhancing the reliability of VGI, including damping, a robust Huber fitting loss, and the incorporation of prior knowledge constraints and regularization. VGI remains effective even when the state space dimension is large relative to the number of fitting samples, as we will demonstrate with several numerical examples. Finally, we note that the computational effort of obtaining a good QADP policy using VGI is small enough that it is comparable to that of simply evaluating the policy through simulation.

* Corresponding author.
E-mail address: yalan@stanford.edu (A. Yang).

https://doi.org/10.1016/j.arcontrol.2023.100917
Received 13 July 2023; Received in revised form 2 November 2023; Accepted 3 November 2023
Available online 10 November 2023

1.1. Related work

Dynamic programming. Dynamic programming (DP) provides techniques for computing the optimal value function and policy for general Markov decision processes. The optimal policy is evaluated by solving an optimization problem, where the control is chosen by minimizing the current stage cost plus the expected value function at the next state. For convex stochastic control problems, this is a convex optimization problem (Bellman, 1954; Bertsekas, 2017; Bertsekas & Shreve, 1996; Puterman, 2014). However, it is possible to exactly represent and find the value function in a only few special cases, for example when the state space is discrete (Sutton & Barto, 2018), or when we have a convex stochastic control problem with a convex extended quadratic stage cost (Barratt & Boyd, 2021).

Approximate dynamic programming. ADP (Bertsekas, 2012, 2019; Powell, 2007) methods are heuristics used in stochastic control when the problem cannot be solved by applying DP directly. Typically, these methods either approximate the value function in DP or tune the parameters of a parametric policy. In some contexts, approximate value functions are known as control Lyapunov functions (Corless & Leitmann, 1988; Freeman & Primbs, 1996).

One approach to ADP is to approximate the value function by relaxing the Bellman equation to an inequality, and then solving a convex optimization problem involving a model of the dynamics and stage cost. When the state and input spaces are finite, this leads to a linear program (LP) (De Farias & Van Roy, 2003). When the dynamics are affine, the stage cost is quadratic, and the input is constrained to be in a convex set, quadratic approximate value functions can be obtained using semidefinite programming (Wang & Boyd, 2009; Wang, O'Donoghue, & Boyd, 2015). In both cases, the resulting approximate value functions are lower bounds on the true value function.

Other value function approximation methods search for an approximate value function that satisfies the Bellman equation along simulated trajectories. This includes the method proposed in this paper, which is closely related to fitted (or projected) value iteration (Bellman & Dreyfus, 1959; Bertsekas, 2012; Keshavarz & Boyd, 2014). Other methods, which do not assume that a model of the dynamics and stage cost are available, include Q-iteration (Antos, Szepesvári, & Munos, 2007; Bertsekas, 2012), Q-learning (Sutton & Barto, 2018; Watkins & Dayan, 1992), and temporal difference learning (Bertsekas, Borkar, & Nedic, 2004; Sutton, 1988).

Instead of approximating the value function, other ADP techniques directly optimize the parameters of a parametric policy to improve performance along system trajectories. Stochastic gradient descent and its variants have been employed to tune convex optimization control policies (Agrawal, Barratt, Boyd, & Stellato, 2020) and controllers based on Proportional-Integral-Derivative (PID) control (Åström, Hägglund, Hang, & Ho, 1993; Minorsky, 1922) and model predictive control (MPC) (Amos, Jimenez, Sacks, Boots, & Kolter, 2018; Camacho & Bordons, 2013). Policy gradient methods provide a method for differentiating through policies parametrized by neural networks (Mnih et al., 2016; Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

Reinforcement learning. Reinforcement learning (RL) methods (Bertsekas, 2019; Sutton & Barto, 2018) can be considered a form of approximate dynamic programming (ADP), although their primary focus is on learning from interactions with the system or a simulator, rather than relying on explicit mathematical models of the system dynamics or stage cost. In this work, we assume that models of the dynamics and stage cost are either known or have been estimated or learned beforehand. This is similar to some model-based RL methods that learn a policy and a model of the dynamics jointly (Deisenroth & Rasmussen, 2011; Sutton, 1990). In the context of control, the process of learning the dynamics is typically referred to as system identification (Ljung, 1998).

Value gradients. When considering a differentiable approximate value function, it is advantageous to have accurate approximations of its derivatives with respect to the state, *i.e.*, the value gradient. If the value gradient is well-approximated along a simulated trajectory, then the approximate value function also provides a good local approximation around that trajectory (Dayan & Singh, 1995). Notably, it is only necessary to approximate the value gradient since constant offsets in the approximate value function do not affect the associated policy.

On the other hand, solely having a good approximation of the value function itself along a trajectory does not ensure a good local approximation. In many cases, value function approximation methods rely on stochastic local exploration, such as dithering (Bertsekas, 2012; Sutton & Barto, 2018), to overcome this limitation. Indeed, value-gradient-based RL methods such as dual heuristic programming (DHP), Werbos (1999) globalized DHP (Prokhorov & Wunsch, 1997), value-gradient learning (Fairbank, 2008; Fairbank & Alonso, 2012), and stochastic value gradients (Amos, Stanton, Yarats, & Wilson, 2021; Heess et al., 2015) have been shown to find better policies using less simulation than value function approximation methods that do not directly approximate the value gradient.

VGI differs from the aforementioned value-gradient-based methods in that it does not require stochastic approximations of the value gradient. Fitted value iteration with value gradients is tractable for convex stochastic control problems, since we can exactly evaluate the gradient of the Bellman operator applied to a convex approximate value function by solving a convex optimization problem.

Convex optimization control policies. For convex stochastic control, the policy associated with a convex quadratic approximate value function can be evaluated by solving a convex optimization problem, *i.e.*, it is a convex optimization control policy (COCP) (Agrawal et al., 2020). COCPs are typically evaluated by solving quadratic programs (QPs), which can often be done efficiently in real-time (Wang & Boyd, 2010). Evaluating a COCP may also involve minimizing a more complex convex function, such as one parametrized by a neural network (Amos, Xu, & Kolter, 2017) To enable embedded applications, code generation tools like CVXGEN (Mattingley & Boyd, 2012) and CVXPYgen (Schaller et al., 2022) can be utilized.

Other examples of COCPs include convex model predictive control (MPC) (Borrelli, Bemporad, & Morari, 2017; Garcia, Prett, & Morari, 1989) and convex approximate dynamic programming (Bertsekas, 2012; Keshavarz & Boyd, 2014). COCPs can also be tuned by differentiating through their solution maps (Agrawal et al., 2020; Amos et al., 2018).

1.2. Outline

In Section 2, we introduce the convex stochastic control problem and solution methods, via dynamic programming and model predictive control. Approximate dynamic programming with quadratic approximate value functions is described in Section 3, value-gradient iteration is introduced in Section 4, and extensions and variations are discussed in Section 5. In Section 6, we present three numerical examples: an input-constrained linear quadratic regulator (LQR) problem, a commitments planning problem involving alternative investments, and a supply chain optimization problem.

2. Convex stochastic control

2.1. Average-cost convex stochastic control problem

Dynamics. We consider a dynamical system evolving in discrete time $t = 0, 1, 2, \ldots$, with state $x_t \in \mathbf{R}^n$, input $u_t \in \mathbf{R}^m$, and affine dynamics

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

where $A_t \in \mathbf{R}^{n\times n}$, $B_t \in \mathbf{R}^{n\times m}$, and $c_t \in \mathbf{R}^n$ are random. We assume the dynamics are time-invariant, *i.e.*, (A_t, B_t, c_t) are independent and

identically distributed (IID) for different values of t. The initial state x_0 is also random, independent of all (A_t, B_t, c_t). When A_t, B_t, or c_t are not random, *i.e.*, constant, we write them as A, B, or c.

Certainty-equivalent dynamics. We denote the expectations of the dynamics matrices as $\bar{A} = \mathbf{E}A_t$, $\bar{B} = \mathbf{E}B_t$, and $\bar{c} = \mathbf{E}c_t$. We refer to the dynamical system with the matrices replaced by their expectations,

$$z_{t+1} = \bar{A}z_t + \bar{B}_t v_t + \bar{c} \quad t = 0, 1, \dots,$$

with initial condition $z_0 = \mathbf{E}x_0$, as the certainty-equivalent system (with state $z_t \in \mathbf{R}^n$ and input $v_t \in \mathbf{R}^m$).

State-feedback policy. We consider the time-invariant state feedback policy

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

where $\phi : \mathbf{R}^n \to \mathbf{R}^m$ is the policy that maps the state to the input. The closed-loop system dynamics are

$$x_{t+1} = A_t x_t + B_t \phi(x_t) + c_t, \quad t = 0, 1, \dots,$$

which defines a stochastic process for the state x_t.

Stage cost. The stage cost is a function $g : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$, where $g(x_t, u_t)$ is the cost at time t. The stage cost g imposes constraints by taking infinite values at disallowed state-input pairs (x_t, u_t). We assume that the stage cost is a closed convex function. Note that the cost function does not depend on time, *i.e.*, it is time-invariant.

In some applications the cost is also random, *e.g.*, of the form $\tilde{g}_t(x_t, u_t)$, where $\tilde{g}_t$ is IID, and independent of A_t, B_t, c_t, and therefore also of x_t. Since we will work with the expected value of the stage cost, we can handle this situation by taking $g(x, t) = \mathbf{E}\tilde{g}_t(x, t)$, where the expectation is over the random stage cost. For simplicity we assume that this expectation may be computed analytically. In other cases, the expectation may be approximated, for example using a sample average.

Average cost. The infinite-horizon average cost is given by

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

Here, we assume that the limit and expectations exist.

We exclusively consider the average-cost problem, and do not consider the closely-related discounted infinite horizon problem and finite horizon problem, which may have time-varying stage cost. However, our approach is readily extended to those problem settings, as discussed in Section 5.

Convex stochastic control problem. The convex stochastic control problem is to choose the policy ϕ so as to minimize the cost J. We will denote an optimal policy as $\phi^\star$, and assume that it exists. We let $J^\star$ denote the optimal value, *i.e.*, the cost J with an optimal policy. The data in this problem are the distributions of (A_t, B_t, c_t) (which do not depend on t), the distribution of x_0, and the stage cost function g.

2.2. Dynamic programming

The optimal control problem is readily solved, at least in principle, using dynamic programming (DP) (Bellman, 1954; Bertsekas, 2012; Bertsekas & Shreve, 1996; Pontryagin, 1987; Puterman, 2014). An optimal policy may be expressed in terms of a so-called Bellman or optimal value function $V^\star : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$, which roughly speaking represents the optimal long-term cost of being in a given state.

An optimal policy can be expressed in terms of a value function as

$$\phi^\star(x) = \operatorname*{argmin}_u \left(g(x, u) + \mathbf{E}V^\star(A_t x + B_t u + c_t)\right). \tag{2}$$

If there are multiple minima, we can arbitrarily choose one. The first term in the quantity that is minimized is the immediate stage cost incurred by the input choice u. The second term reflects the optimal expected long-term cost of starting from the next state. The optimal policy balances these two costs.

The policy does not change when we add a constant to a value function. Without loss of generality we can remove this ambiguity by insisting that $V^\star(x^{\mathrm{ref}}) = 0$, where x^{ref} is a reference state (for which there is an optimal value function with finite value). The value function

$$V^{\mathrm{rel}}(x) = V^\star(x) - V^\star(x^{\mathrm{ref}})$$

is sometimes called a *relative value function*.

Bellman operator. It can be shown that a value function $V^\star$ and the optimal cost $J^\star$ satisfy

$$V^\star + J^\star = \mathcal{T}V^\star, \tag{3}$$

where $\mathcal{T}$ is the *Bellman operator*, given by

$$(\mathcal{T}h)(x) = \min_u \left(g(x, u) + \mathbf{E}h(A_t x + B_t u + c_t)\right), \tag{4}$$

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

It follows that a relative value function is a fixed point of the Bellman operator $\mathcal{T}$, *i.e.*,

$$V^{\mathrm{rel}} = \mathcal{T}V^{\mathrm{rel}}.$$

This fixed point condition implies (3), with optimal cost $J^\star = \mathcal{T}V^\star(x^{\mathrm{ref}})$.

Value iteration. The relative value function V^{rel} may be found by fixed point iteration. Under certain technical conditions, the so-called value iteration (or relative value iteration)

$$V^{k+1} = \mathcal{T}V^k - \mathcal{T}V^k(x^{\mathrm{ref}}), \qquad k = 1, 2, \dots \tag{5}$$

converges, *i.e.*, $V^k - V^k(x^{\mathrm{ref}}) \to V^{\mathrm{rel}}$ and $\mathcal{T}V^k(x^{\mathrm{ref}}) \to J^\star$ (Bertsekas, 2012; Puterman, 2014; White, 1969).

For future reference we mention a variation on value iteration called *damped value iteration*, which has the form

$$V^{k+1} = \rho_k \left(\mathcal{T}V^k - \mathcal{T}V^k(x^{\mathrm{ref}})\right) + (1 - \rho_k)V^k, \quad k = 1, 2, \dots, \tag{6}$$

where $\rho_k \in (0, 1]$ with $\sum_k \rho_k(1 - \rho_k) = \infty$. Damped value iteration also satisfies $V^k - V^k(x^{\mathrm{ref}}) \to V^{\mathrm{rel}}$ and $\mathcal{T}V^k(x^{\mathrm{ref}}) \to J^\star$ under certain technical conditions.

The value function is convex. The Bellman operator (4) maps convex functions to convex functions, since expectation and partial minimization preserve convexity (see, *e.g.*, Boyd and Vandenberghe (2004, §3.2.1, §3.2.5)). With any convex V^1 (*e.g.*, the zero function), it follows that all iterates of value iteration are convex, which implies that its limit $V^\star$ is convex.

One implication is that evaluating the policy (2), *i.e.*, minimizing

$$g(x, u) + \mathbf{E}V^\star(A_t x + B_t u + c_t)$$

over u, is a convex optimization problem. To see this, we observe that $A_t x + B_t u + c_t$ is an affine function of u, so by the affine pre-composition rule, $V^\star(A_t x + b_t u + c_t)$ is a convex function of u. Adding this to $g(x, u)$ and taking expectation preserve convexity, so the function that is minimized is a convex function of u.

Since evaluating the policy (2) involves solving a convex optimization problem, we refer to it as a *convex optimization control policy*.

Linear quadratic regulator. The dynamic programming approach can only be carried out in practice in special cases. The most widely known example is when the stage cost is a (convex) quadratic function, in which case the optimal control problem is called the *linear quadratic regulator* (LQR). For LQR the Bellman operator preserves convex quadratic functions, so it follows that the limit $V^\star$ is also convex quadratic, and the optimal policy is affine, *i.e.*, $\phi^\star(x) = Kx + l$, where $K \in \mathbf{R}^{m\times n}$ and $l \in \mathbf{R}^m$ (see Barratt and Boyd (2021)). Value iteration for LQR can be carried out using basic linear algebra operations, and so is tractable. Most importantly we have a practical way to represent the Bellman iterates, and also their limit, by a finite set of parameters, the coefficients of a quadratic function.

Dynamic programing in the general case. Beyond the special case of LQR described above, there are a handful of other very specific stochastic control problems that are tractable to solve. These cases follow the same general story line as LQR: There is a class of functions that is preserved under the Bellman operator. One example is Merton's portfolio problem, which considers the allocation of wealth between various assets over time, and admits a closed-form solution (Merton, 1969). Problems with a finite state space may, in principle, be solved by DP, by representing the value function with a table of values. This is referred to as the tabular case (Sutton & Barto, 2018). When the state space is continuous but low-dimensional, say, with $n \leq 4$, the region of interest in the state space may be represented using a finite number of points, for example a uniform grid. Tabular DP may then be used, in combination with an interpolation over those points, to give a good approximation of the value function. However, this approach does not scale to problems with larger state dimension, since the number of points needed to represent the value function to a given accuracy grows exponentially with the state dimension.

The challenge in carrying out dynamic programming in more general cases is simple: There is no practical way to represent an arbitrary convex function on $\mathbf{R}^n$.

2.3. Certainty-equivalent steady-state optimal state-input pair

For many stochastic control problems, certainty-equivalent approximations may be used to obtain heuristic policies without dynamic programming. In this section we explain the idea of an optimal steady-state certainty-equivalent optimal state-input pair. We start by making two very crude approximations of the stochastic control problem. First, we ignore all uncertainty by replacing the dynamics matrices with their mean values (also called certainty-equivalent). Second, we assume that the system is in steady-state, with constant state $z \in \mathbf{R}^n$ and constant input $v \in \mathbf{R}^m$, *i.e.*, $z = \bar{A}z + \bar{B}v + \bar{c}$. Then we choose z and v to minimize the objective, which with the assumptions above reduces to $g(z, v)$. Thus we solve the convex optimization problem

$$\begin{array}{ll} \text{minimize} & g(z, v) \\ \text{subject to} & z = \bar{A}z + \bar{B}v + \bar{c}, \end{array} \tag{7}$$

with variables $z \in \mathbf{R}^n$ and $v \in \mathbf{R}^m$. We refer to a solution of this problem $(z^\star, v^\star)$ as a certainty-equivalent steady-state optimal (CE-SSO) state-input pair, and denote it as $(x^{\text{sso}}, u^{\text{sso}})$. For some problems, such as the example considered in Section 6.2, the constant policy $\phi(x) = u^{\text{sso}}$ is a reasonable heuristic.

2.4. Certainty-equivalent model predictive control

Certainty-equivalent model predictive control (CE-MPC) is another heuristic policy for stochastic control (Borrelli et al., 2017; Garcia et al., 1989). CE-MPC is not our focus, but the methods of this paper can also be used to develop a good CE-MPC policy.

To evaluate the CE-MPC policy $\phi^{\text{mpc}}(x)$, we solve an H-step ahead planning problem with certainty-equivalent dynamics. The planning problem is

$$\begin{array}{ll} \text{minimize} & \frac{1}{H+1}\sum_{\tau=1}^{H} g(z_\tau, v_\tau) + V^{\text{mpc}}(z_{H+1}) \\ \text{subject to} & z_{\tau+1} = \bar{A}z_\tau + \bar{B}v_\tau + \bar{c}, \quad \tau = 1, \ldots, H \\ & z_1 = x, \end{array} \tag{8}$$

with variables $z_1, \ldots, z_{H+1}$ and $v_1, \ldots, v_H$. The CE-MPC policy is then $\phi^{\text{mpc}}(x) = v_1^\star$, the first input of an optimal trajectory of the MPC planning problem. (8).

In the CE-MPC problem (8), V^{mpc} is called the terminal cost. It can be chosen to be zero (particularly when H is large enough), or the indicator function of x^{sso}, an optimal certainty-equivalent steady-state state. Another very good choice is $\hat{V}$, an approximation of the value function, which can be found by the methods of this paper.

3. Quadratic approximate dynamic programming

3.1. Approximate dynamic programming

In this paper, we consider ADP policies that replace the optimal value function $V^\star$ in (2) with a convex approximation $\hat{V}$. The ADP policy is of the form

$$\hat{\phi}(x) = \operatorname*{argmin}_u \left(g(x, u) + \mathbf{E}\hat{V}\left(A_t x + B_t u + c_t\right)\right). \tag{9}$$

(We omit the constant or offset term since it does not affect the associated policy.) If there are multiple minima, we can arbitrarily choose one. When $\hat{V}$ is a convex quadratic function, we refer to (9) as a QADP policy.

ADP is a heuristic that addresses the issue mentioned above, that there is no practical way to represent an arbitrary convex function on $\mathbf{R}^n$ (Bellman & Dreyfus, 1959; Bertsekas, 2012; Munos, 2007). The approximate value function $\hat{V}$ is chosen to approximate $V^\star$ in some sense, and to make evaluating the policy (9) tractable. Evaluating $\hat{\phi}$ is always a convex optimization problem; depending on the form of g and $\hat{V}$, the expectation can simplify and the problem can reduce to a common form, such as a quadratic program (QP). When it is not possible to evaluate the expectation in the policy exactly, we can use an estimate obtained by replacing the expectation with a suitable sample average, *i.e.*, a Monte Carlo approximation (Bertsekas, 2012). ADP often works well in practice, even in cases when $\hat{V}$ is not a particularly good approximation of $V^\star$ (Agrawal et al., 2020; Keshavarz & Boyd, 2014).

3.2. Quadratic approximate value functions

In this paper we focus exclusively on quadratic approximate value functions of the form

$$\hat{V}(x) = \frac{1}{2}\begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} P & p \\ p^T & 0 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \frac{1}{2}x^T P x + p^T x, \tag{10}$$

where $P \succeq 0$, *i.e.*, $P \in \mathbf{S}_+^n$, the set of symmetric positive semidefinite (PSD) $n \times n$ matrices.

The QADP policy associated with $\hat{V}$ is parametrized by the $n \times n$ PSD matrix P and n-vector p, which we collectively refer to as $\theta = (P, p)$. All together, the parameter θ contains

$$n(n+1)/2 + n = (1/2)n^2 + (3/2)n \tag{11}$$

scalar parameters, which has order n^2. We define $\Theta = \{\theta \mid P \succeq 0\}$, the set of parameters for which $\hat{V}$ is convex.

3.3. Properties of QADP policies

We now consider several properties of the QADP policies which will be useful in the sequel.

Simplifying the expectation. The QADP policy can be simplified, since the expectation of a quadratic function can be expressed analytically in terms of the first and second moments of its argument. Thus we have

$$\mathbf{E}\hat{V}(A_t x + B_t u + c_t) = \frac{1}{2}\begin{bmatrix} u \\ 1 \end{bmatrix}^T \begin{bmatrix} M & m \\ m^T & \mu(x) \end{bmatrix} \begin{bmatrix} u \\ 1 \end{bmatrix}, \tag{12}$$

where

$$\begin{aligned} M &= \mathbf{E}B_t^T P B_t, \\ m &= \mathbf{E}B_t^T (P A_t x + P c_t) + \bar{B}^T p, \\ \mu(x) &= x^T \mathbf{E}(A_t^T P A_t) x + 2x^T \mathbf{E}(A_t^T P c_t) + 2x^T \bar{A}^T p + 2p^T \bar{c} + \mathbf{E}c_t^T P c_t. \end{aligned}$$

Note that $\mu(x)$ depends on x, and therefore is not constant, but the other coefficients M and m are constant and depend only on the first and second moments of A, B, c (and P and p). These formulas are derived in Appendix A. Finally, we observe that M, m, and $\mu(x)$ are linear functions of θ.

Evaluating the policy. Since $g(x,u)$ is convex, evaluating the quadratic ADP policy reduces to solving a deterministic convex optimization problem. When in addition $g(x,u)$ is QP-representable, *i.e.*, a convex quadratic function plus a convex piecewise linear function, plus the indicator function of linear inequality and equality constraints, evaluating the QADP policy reduces to solving a QP (Wang & Boyd, 2010).

Gradient of the Bellman operator image. Given convex quadratic $\hat{V}$, we may evaluate $\mathcal{T}\hat{V}(x)$, the Bellman operator applied to $\hat{V}$ at any state x, by solving the convex optimization problem associated with the QADP policy. We can also compute $\nabla\mathcal{T}\hat{V}(x)$, where it is differentiable, and a subgradient otherwise.

To do this, we represent $\mathcal{T}\hat{V}(x)$ as the optimal value of the convex optimization problem

$$\begin{array}{ll}\text{minimize} & g(\tilde{x},u)+\mathbf{E}\hat{V}(A_t\tilde{x}+B_tu+c_t)\\ \text{subject to} & \tilde{x}=x,\end{array} \tag{13}$$

where we have introduced the variable $\tilde{x}$. Let $\nu^\star(x)\in\mathbf{R}^n$ represent the optimal Lagrange multiplier associated with the constraint $\tilde{x}=x$. Then, we have $\nabla\mathcal{T}\hat{V}(x)=-\nu^\star(x)$ when the gradient exists (Boyd & Vandenberghe, 2004, §5.6). Otherwise, $-\nu^\star(x)$ is a subgradient, *i.e.*, $-\nu^\star(x)\in\partial\mathcal{T}\hat{V}(x)$.

4. Value-gradient iteration

4.1. Fitted value iteration

We begin by reviewing fitted (or projected) value iteration (FVI), which is an approximation of value iteration (Bellman & Dreyfus, 1959; Bertsekas, 2012; Keshavarz & Boyd, 2014). The issue with value iteration is that in practice, we cannot exactly represent the function $\mathcal{T}V^k$ in the update (6). FVI addresses this by restricting all approximate value function iterates V^k to be convex quadratic functions.

In the kth iteration, we choose a set of states $x^1,\dots,x^N$, and evaluate $\mathcal{T}V^k(x^i)$ for each $i=1,\dots,N$. We can evaluate each $\mathcal{T}V^k(x^i)$ by evaluating (4), which is a convex optimization problem. Then, we fit a convex quadratic function $V^{k+1/2}$ to those points, such that

$$V^{k+1/2}(x^i)\approx\mathcal{T}V^k(x^i),\quad i=1,\dots,N.$$

This leads to the damped fitted value iteration update

$$V^{k+1}=\rho_kV^{k+1/2}+(1-\rho_k)V^k,\quad k=0,1,\dots, \tag{14}$$

which generates a sequence of convex quadratic functions V^k, with associated QADP policies.

Fitting convex quadratic functions. One method for finding parameters $\theta=(P,p)$ for the convex quadratic function $V^{k+1/2}$ is to fit it to a set of points. We first evaluate $v^i=\mathcal{T}V^k(x^i)$ for each $i=1,\dots,N$, and then solve the fitting problem

$$\begin{array}{ll}\text{minimize} & (1/N)\sum_{i=1}^N L\left(V^{k+1/2}(x^i)+c-v^i\right)+r(\theta)\\ \text{subject to} & \theta\in\Theta,\end{array} \tag{15}$$

with variables θ and $c\in\mathbf{R}$, where c is a scalar offset. Here $L:\mathbf{R}\to\mathbf{R}$ is a convex fitting loss function, and $r:\mathbf{S}^n\times\mathbf{R}^n\to\mathbf{R}\cup\{\infty\}$ is a convex regularization function, with infinite values used to impose (convex) constraints on θ. This is a convex optimization problem, since $V^{k+1/2}(x^i)$ is a linear function of θ. Possible choices for L include the squared loss or the robust Huber loss (Huber, 1992), given by

$$L^{\text{hub}}(z)=\begin{cases}(1/2)z^2 & |z|\le M\\ M(|z|-M/2) & |z|>M.\end{cases} \tag{16}$$

The Huber loss is a more robust alternative to the square loss, in the presence of outliers. Possible choices for r include ℓ_2 regularization and prior knowledge constraints, and are discussed in Section 4.3. For simplicity, we consider the standard Huber function, which transitions from the quadratic to absolute value at $M=1$. In general, M may be tuned by cross-validation, using a procedure similar to that described in Section 4.3.

Convergence. Convergence guarantees for FVI are available when the approximation error of $\nabla\mathcal{T}V^k$ is small enough (Bertsekas, 2012; Munos, 2007). However, unlike value iteration, FVI is not guaranteed to converge in general (Baird, 1995; Tsitsiklis & Van Roy, 1996). Nevertheless, with an appropriate approximation $\hat{\nabla}\mathcal{T}V^k$ and damping parameters ρ_k, FVI can often find policies with good performance in practice.

4.2. Value-gradient iteration

VGI is a special case of FVI, where we fit $V^{k+1/2}$ using gradients instead of values. In Section 3.3, we showed that we can evaluate $\nabla\mathcal{T}V^k(x)$ at any state x where $\mathcal{T}V^k$ is differentiable, by evaluating a particular optimal Lagrange multiplier. Therefore, we can find $V^{k+1/2}$ by fitting its gradient.

That is, we choose $V^{k+1/2}(x)=(1/2)x^TPx+p^Tx$ such that $P\succeq0$ and

$$\nabla V^{k+1/2}(x)=Px^i+p\approx\nabla\mathcal{T}V^k(x^i),\quad i=1,\dots,N.$$

Once we have found $V^{k+1/2}$, we apply the damped update (14) to generate the next iterate V^{k+1}. Like in standard FVI, this generates a sequence of convex quadratic functions V^k, with associated QADP policies.

Fitting the gradient. In this case, we fit an affine function $\nabla V^{k+1/2}(x)=Px+p$ to a set of points, subject to the constraint that P is symmetric positive semidefinite. In each iteration, we evaluate $g^i=\nabla\mathcal{T}V^k(x^i)$ for each $i=1,\dots,N$, and then solve the fitting problem

$$\begin{array}{ll}\text{minimize} & (1/N)\sum_{i=1}^N L\left(\nabla V^{k+1/2}(x^i)-g^i\right)+r(\theta)\\ \text{subject to} & \theta\in\Theta,\end{array} \tag{17}$$

with variables θ. Here $L:\mathbf{R}^n\to\mathbf{R}$ is a multivariate convex fitting loss function, and r is, like in (15), a convex regularization function. This is also a convex optimization problem, since $\nabla V^{k+1/2}(x^i)$ is a linear function of θ.

Possible choices for L include the squared ℓ_2 norm and the circular Huber loss

$$L^{\text{hub}}(z)=\begin{cases}(1/2)\|z\|_2^2 & \|z\|_2\le M\\ M(\|z\|_2-M/2) & \|z\|_2>M,\end{cases} \tag{18}$$

which extends the scalar Huber loss (16) to the multivariate case. Like in the scalar case, the circular Huber loss is a more robust alternative to the square function, in the presence of outliers.

Choice of sampling points. An important consideration is the choice of the state samples values $x^1,\dots,x^N$ at which we evaluate the policy and $\mathcal{T}V^k(x^i)$. Ideally the samples should reflect the states that the system is likely to be in, *i.e.*, samples from the steady-state distribution of x_t under the policy ϕ^k.

To accomplish this we choose the sample points by simulating the current policy for N steps, using the current policy ϕ^k. In the first iteration $k=1$, we initialize the simulation at a state chosen at random. In subsequent iterations, we initialize the simulation at the last state in the previous iteration.

4.3. Regularization, constraints, and lower bounds

Prior information, if available, can be incorporated as regularization terms or constraints in the fitting problem, through the function $r(\theta)$ in the fitting problem (17). Constraints and lower bounds may be imposed by setting r to have value ∞ when θ is not consistent with the prior information. We now describe a nonexhaustive list of possibilities that may be combined to form $r(\theta)$.

Ridge regularization. We may add an ℓ_2 penalty on the parameters of the value function

$$r(\theta) = \lambda \left(\|P\|_F^2 + \|p\|_2^2 \right),$$

where $\lambda > 0$ is a scalar regularization parameter and $\|\cdot\|_F$ denotes the Frobenius norm. The ℓ_2 regularization ensures that the fitting problem is well-posed and helps mitigate overfitting, and is sometimes referred to as Tikhonov or ridge regularization (Hastie, Tibshirani, & Friedman, 2009; Tikhonov & Arsenin, 1977).

The parameter λ is typically chosen using use out-of-sample or cross-validation. To do this we divide the fitting data (x^i, v^i) into two sets, the training data and the validation data. We fit V using the training data, for a range of values of λ, typically on a log scale with upper limits $\lambda^{\max}$ and $\lambda^{\max}$, and then evaluate the average loss on the validation data for each value of λ. We then choose a value that gives near minimum validation error, with a preference for larger values, *i.e.*, more regularization. This approach is often referred to as grid search. A more thorough method is to use cross-validation (Hastie et al., 2009), and more sophisticated search methods for evaluating scaling parameters may also be considered; see, for example, Jamieson and Talwalkar (2016).

Lasso regularization. The ℓ_1 penalty

$$r(\theta) = \lambda \left(\sum_{i,j=1}^{n} |P_{ij}| + \|p\|_1 \right)$$

with regularization parameter $\lambda > 0$ is known as LASSO (Hastie et al., 2009). This regularization is similar to ridge regression in that both shrink the values of the parameters; however, the LASSO is more likely to produce sparse solutions, *i.e.*, P and p with zero-valued entries. Therefore, the LASSO regularization can be particularly useful for weakly coupled systems.

Like with ridge regression, the value of λ may be tuned using out-of-sample or cross-validation. When multiple regularization terms are used, we can use the same strategy to find a good set of values for each regularization parameter. For example, the case where both ridge and LASSO regularization are employed is known as the elastic net (Zou & Hastie, 2005). In this case, the aforementioned grid search strategy may be used to select the two regularization parameters jointly.

Symmetry. In some cases, we may know that the value function V should be symmetric, *i.e.*, $V(x) = V(-x)$ for any $x \in \mathbf{R}^n$. The LQR example considered in Section 6.1, for example, satisfies this property. For quadratic approximate value functions, symmetry may be implemented by the constraint $p = 0$.

Fixed minimizer. When we can identify a point $x^\star$ in the state space that seems to be the best, we may include the constraint $\operatorname{argmin}_x V(x) = x^\star$ to the fitting problem. This is equivalent to the linear equality constraint $Px^\star + p = 0$. A special case is when V is constrained to be symmetric, in which case $V(x)$ is minimized at zero.

Lower bounds. In some cases, a quadratic pointwise lower bound

$$V^{\mathrm{lb}}(x) = \frac{1}{2} x^T P^{\mathrm{lb}} x + (p^{\mathrm{lb}})^T x = \frac{1}{2} \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} P^{\mathrm{lb}} & p^{\mathrm{lb}} \\ (p^{\mathrm{lb}})^T & 0 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

on $V^\star$ is available up to an additive constant, and may be included as an additional constraint. This may be done by introducing an additional variable s, and imposing the pointwise constraint $V + s \geq V^{\mathrm{lb}}$. This can be expressed as the convex constraint

$$\begin{bmatrix} P - P^{\mathrm{lb}} & p - p^{\mathrm{lb}} \\ (p - p^{\mathrm{lb}})^T & s \end{bmatrix} \succeq 0, \tag{19}$$

as shown in Appendix B. Since $P^{\mathrm{lb}} \succeq 0$, this constraint implies that $P \succeq 0$. So when we add a quadratic lower bound constraint to the fitting problem, we no longer need the constraint $P \succeq 0$.

In many cases we can form a convex quadratic lower bound V^{lb} on the true value function $V^\star$. In the simplest case we can take $V^\star = 0$ when the stage cost is nonnegative. Another method is to form an LQR relaxation of the problem, *i.e.*, to replace g with a quadratic lower bound, for example, by ignoring constraints on u. The resulting LQR problem can be solved exactly, and its value function V^{lqr} is a lower bound on $V^\star$. More sophisticated methods for computing a lower bound on the value function involve solving a convex optimization problem (Wang & Boyd, 2009) or a series of convex problems (O'Donoghue, Wang, & Boyd, 2011).

When the dynamics matrices A_t and B_t are random, a simpler lower bound may be found by considering the (deterministic) LQR relaxation of the CE problem; see Appendix C.

Policy interpolation. Suppose we have a set of states $x^1, \ldots, x^B$, and require that the policy takes on corresponding values $u^1, \ldots, u^B$, *i.e.*,

$$\phi(x^j) = u^j, \quad j = 1, \ldots, B.$$

This condition may be written as

$$0 \in \partial g(x^j, u^j) + \nabla \mathbf{E} V^k (A_t x^j + B_t u^j + c_t), \tag{20}$$

where $\partial g(x^j, u^j)$ is the set of subgradients of $g(x, u)$ with respect to u, evaluated at (x^j, u^j).

In some cases, this constraint has a simple representation. For example, if the stage cost may be written in the form

$$g(x, u) = h(x, u) + I\left((x, u) \in C\right)$$

where h is differentiable and $I\left((x, u) \in C\right)$ is the indicator function of a polyhedral set C, then the constraint may be written as a linear inequality constraint on the parameters P and p. First, note that

$$\partial g(x^j, u^j) = \nabla h(x^j, u^j) + \partial I\left((x, u) \in C\right),$$

where $\partial I\left((x, u) \in C\right)$ is the normal cone to C at (x^j, u^j). Since C is a polyhedron the normal cone is also a polyhedron (Rockafellar, 1970, S23), *i.e.*, representable by a set of linear inequality constraints. Next, from (12) we have

$$\nabla \mathbf{E} V^k (A_t x^j + B_t u^j + c_t) = \mathbf{E}(B_t^T P B_t) u^j + \mathbf{E} B_t^T (P A_t x^j + P c_t) + \tilde{B}^T p,$$

which is a linear function of P and p. Therefore, the policy interpolation constraints (20) may be represented by a set of linear inequality constraints on P and p.

5. Extensions and variations

5.1. Input-affine dynamics

The methods presented in this paper can also be applied in cases where the dynamics are nonlinear but input-affine. That is, the dynamics may be written in the form

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

where $f_t : \mathbf{R}^n \to \mathbf{R}^n$ and $B_t : \mathbf{R}^n \to \mathbf{R}^{n \times m}$ are random functions. We again assume that (f_t, g_t) are IID for different values of t. The affine dynamics described in Section 2 are a special case, where $f_t(x) = A_t x + c_t$ and $g_t(x) = B_t$.

In the input-affine case, the ADP policy (9) is of the form

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

Since the dynamics are affine in u, the expected value $\mathbf{E}\hat{V}(f_t(x) + g_t(x)u)$ is also affine in u, when $\hat{V}$ is convex. When $\hat{V}$ is a convex quadratic function of the form (10), the expected value may be computed exactly, in terms of the first and second moments of $f_t(x)$ and $g_t(x)$ (Keshavarz & Boyd, 2014). Hence, the policy can still be evaluated by solving a convex optimization problem, and VGI can still be performed in a similar manner.

5.2. Alternative cost functions

Discounted infinite-horizon problem. The mean discounted infinite-horizon cost is given by

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

where $\gamma \in (0,1)$ is a discount factor, and the sum and expectations are assumed to exist. In this case, the value function $V^\star$ represents the optimal cost-to-go, and the optimal policy is of the form

$$\phi^\star(x) = \operatorname*{argmin}_u \left(g(x,u) + \gamma \mathbf{E} V^\star(A_t x + B_t u + c_t)\right).$$

For the discounted infinite-horizon problem, VGI proceeds in the same way, except with the Bellman operator defined as

$$(\mathcal{T} h)(x) = \min_u \left(g(x,u) + \gamma \mathbf{E} h(A_t x + B_t u + c_t)\right),$$

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

Finite-horizon problem. In the finite-horizon problem, the cost is given by

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

where the stage cost may be time-varying, and the expectations are assumed to exist. In this case, the value function $V_t^\star$ depends on time, and may be found using a backward recursion. The value iteration starts with

$$V_T^\star(x) = \min_u g_T(x,u),$$

and then proceeds as

$$V_t^\star(x) = \mathcal{T}_t V_{t+1}^\star(x), \quad t = T, T-1, \ldots, 0,$$

where the Bellman operator at time t is defined as

$$\left(\mathcal{T}_t h\right)(x) = \min_u \left(g_t(x,u) + \gamma \mathbf{E} h(A_t x + B_t u + c_t)\right),$$

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

VGI proceeds similarly for the finite-horizon problem, using an analogous function fitting approximation of the Bellman operator.

5.3. Parallel simulations

In VGI (and FVI in general), we select N sample points by simulating the current policy. We can also select points from more than one simulated trajectory. To do this we choose the sample points by simulating K different trajectories for T steps each, using the current policy. In iteration k, each of these K trajectories gives us T states at which we evaluate the policy ϕ^k, so all together we have $N = TK$ states and associated evaluations of $\nabla \mathcal{T} V^k$ to use in the fitting problem (17). One advantage of this method is that the K trajectories can be evaluated in parallel.

6. Numerical examples

In this section, we present three numerical examples, which involve a box-constrained LQR problem, a commitment planning problem with an alternative investments fund, and a supply chain optimization problem. Comparisons with other ADP methods are given in Section 7.

The code for the examples is available at https://github.com/cvxgrp/vgi. The ADP policies and VGI method are implemented using CVXPY (Agrawal, Verschueren, Diamond, & Boyd, 2018; Diamond & Boyd, 2016). In addition, the code generation tool CVXPYgen (Schaller et al., 2022) was used to create custom solvers for the ADP policies, implemented in C. The experiments were performed on two cores of an Intel Xeon E5-2640 CPU.

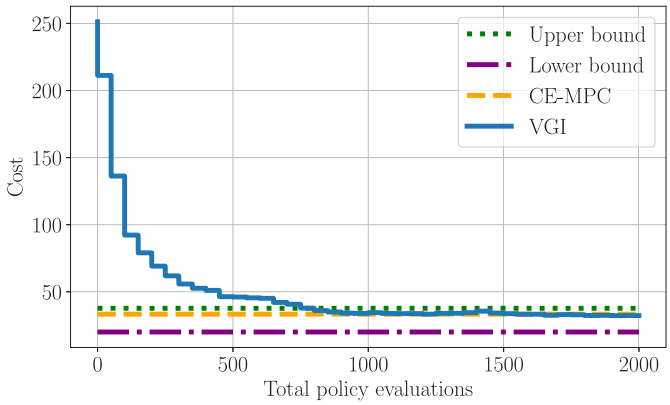

Fig. 1. VGI for the box-constrained LQR problem.

6.1. Box-constrained linear quadratic regulator

We first consider a traditional linear quadratic regulator (LQR) problem. The dynamics are time-invariant, and given by

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

where $A \in \mathbf{R}^{n \times n}$ and $B^{n \times m}$ are known and fixed, and c_t is an IID random variable with zero mean and covariance $\mathbf{E} c_t c_t^T = C$. The stage cost is given by

$$g(x,u) = x^T Q x + u^T R u + I(-u^{\max} \le u \le u^{\max}),$$

where $Q \succeq 0$, $R \succ 0$, and $u^{\max} > 0$ is a maximum input magnitude, in any component of the input.

For this problem, a lower bound J^{lb} on the optimal cost and a quadratic lower bound V^{lb} on the optimal value function can be found by solving a semidefinite program (SDP) (Wang & Boyd, 2009). An upper bound on the optimal cost may be found by evaluating the ADP policy using V^{lb} as the approximate value function.

Numerical example. We consider a problem instance with $n = 12$ and $m = 3$. The entries of A are chosen IID from a uniform distribution on $[-1, 1]$. The matrix A was then rescaled to have a maximum eigenvalue of 1. The entries of B are chosen IID from a uniform distribution on $[-0.5, 0.5]$. The process noise c_t is normally distributed, with zero mean and covariance $0.4I$. The stage cost parameters are given by $Q = I$ and $R = I$, and the maximum input magnitude is $u^{\max} = 0.4$.

Results. We carried out VGI for 40 iterations, starting from the initial value function $V^1(x) = x^T Q x$. We included the symmetry constraint $p = 0$ in the fitting step. In each iteration, the fitting step was performed using $N = 50$ fitting points, obtained by simulating the current policy. The damping coefficient was fixed to $\rho_k = 0.5$.

Fig. 1 shows the average cost versus the number of policy evaluations used to generate the data for the fitting step. Also plotted are the SDP-based upper and lower bounds (Wang & Boyd, 2009) and the average cost of the CE-MPC policy with a horizon of $H = 30$. In this example, VGI converges to a slightly better cost than that of the CE-MPC policy.

6.2. Commitments in an alternative investments fund

Our next example is a practical example, and more specific. We consider a fund that invests in m so-called alternative investment classes, such as venture capital, infrastructure projects, direct lending, or private equity. Alternative investments are found in the portfolios of insurance companies, retirement funds, and university endowments. For more details, see Luxenberg, Boyd, van Beek, Cao, and Kochenderfer (2022) and the papers cited therein.

In each time period (typically quarters) $t = 1, 2, \ldots$, we make nonnegative commitments to the m alternative asset classes. These are amounts we promise to invest, in response to capital calls. Over the next few years, we put money into the investments in response to capital calls, up to the amount of previous commitments. We receive money from each the investments in later years through distributions. Neither the timing nor amounts of the capital calls and distributions are directly under our control, except that the total of the capital calls cannot exceed our total commitments for each asset class.

We first describe some critical quantities.

- $u_t \in \mathbf{R}^m_+$ denotes the amounts that the investor commits in period t, to each of the m asset classes. (These commitments will be the input in our stochastic control problem.)
- $p_t \in \mathbf{R}^m_+$ denotes the amounts that the investor pays in to the investment in response to capital calls in period t.
- $d_t \in \mathbf{R}^m_+$ denotes the amount that the investor receives in distributions from the investments in period t.
- $n_t \in \mathbf{R}^m_+$ denotes the net asset values (NAVs) of the investments in period t.
- $l_t \in \mathbf{R}^m_+$ denotes the total amount of uncalled commitments, *i.e.*, the difference between the total so far committed and the total so far that has been called. (This is a liability, so we use the symbol l.)

The units for all of these is typically millions of USD.

A simple dynamical model relating these variables is

$$n_{t+1} = \mathbf{diag}(r_t)n_t + p_t - d_t, \quad l_{t+1} = l_t - p_t + u_t, \quad t = 1, 2 \ldots,$$

where $r_t \in \mathbf{R}^K_{++}$ is the vector of per-period total returns for the asset classes, assumed to be IID with some known distribution such as log-normal. In words: the value of each investment class in each period is multiplied by its (random) return, increased by the amount paid in, and decreased by the amount distributed; the total uncalled commitments is decreased by the capital calls, and increased by new commitments. The calls and distributions are modeled as

$$p_t = \mathbf{diag}(\gamma_t^{\text{call}})l_t, \quad d_t = \mathbf{diag}(\gamma_t^{\text{dist}})\mathbf{diag}(r_t)n_t, \quad t = 1, 2, \ldots,$$

where γ_t^{call} and γ_t^{dist} are random variables in $(0, 1)^m$, called the call and distribution intensities. We will assume that these are IID, and independent of r_t. In words: In each period and for each asset class, a random fraction of the total liability is called, and a random fraction of the NAV is distributed.

We can express the dynamics as a random linear dynamical system with state $x_t = (n_t, l_t) \in \mathbf{R}^{2m}$ and input $u_t \in \mathbf{R}^m$, with dynamics matrices

$$A_t = \begin{bmatrix} \mathbf{diag}(r_t)(I - \mathbf{diag}(\gamma_t^{\text{dist}})) & \mathbf{diag}(\gamma_t^{\text{call}}) \\ 0 & I - \mathbf{diag}(\gamma_t^{\text{call}}) \end{bmatrix}, \quad B_t = \begin{bmatrix} 0 \\ I \end{bmatrix}, \quad c_t = 0.$$

The goal is to choose commitments so as to reach and maintain a target asset allocation $n^{\text{tar}} \in \mathbf{R}^m_+$, while penalizing deviations of the commitments u_t from the CE-SSO commitment $u^{\text{sso}} \in \mathbf{R}^m_+$. We consider stage cost

$$g(x_t, u_t) = \|n_t - n^{\text{tar}}\|^2 + \lambda\|u_t - u^{\text{sso}}\|^2 + I(0 \le u_t \le u^{\max}),$$

where $\lambda > 0$ is a penalty coefficient and $u^{\max} \in \mathbf{R}^m_+$ are the maximum allowable commitments to each of the asset classes. We take the fixed input u^{sso} is a solution to the certainty-equivalent steady-state problem (7), with the input cost term $\lambda\|u_t - u^{\text{sso}}\|^2$ removed from the stage cost.

For this problem, we find a quadratic lower bound V^{lb} on the value function by relaxing the constraints on the input u_t, replacing A_t with $\bar{A}$, and solving the certainty equivalent LQR problem.

Numerical example. We consider an example with $m = 6$ asset classes. The returns r_t are distributed according to a log-normal distribution, *i.e.*, $r_t = \exp(z_t)$, with $z_t \sim \mathcal{N}(\mu, \Sigma)$. The parameters μ and Σ were chosen such that the mean quarterly returns have means

$$\mathbf{E}r_t = (1.0, 1.1, 1.1, 1.0, 1.1, 1.1)$$

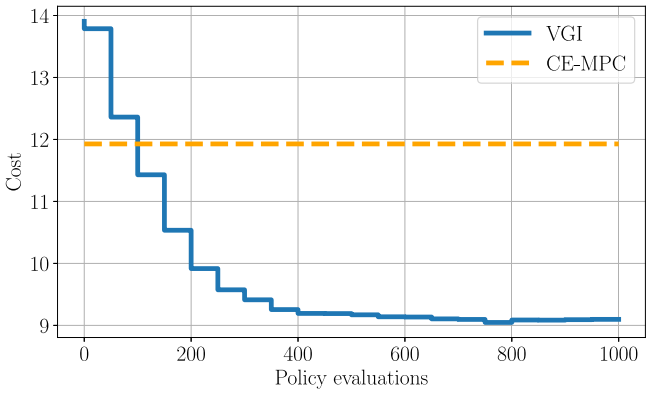

Fig. 2. VGI for the commitments planning problem.

and standard deviations

$$\sigma_t = (0.1, 0.2, 0.2, 0.1, 0.2, 0.1).$$

This leads to annualized returns with means around 20% and standard deviations around 30%. The returns are correlated, with correlation matrix

$$\text{corr}(r_t) = \begin{bmatrix} 1 & -0.06 & -0.05 & 0.62 & -0.32 & -0.44 \\ -0.06 & 1 & -0.21 & 0.18 & 0.80 & -0.12 \\ -0.05 & -0.21 & 1 & 0.35 & -0.27 & -0.19 \\ 0.62 & 0.18 & 0.35 & 1 & 0.18 & -0.15 \\ -0.32 & 0.80 & -0.27 & 0.18 & 1 & 0.37 \\ -0.44 & -0.12 & -0.19 & -0.15 & 0.37 & 1 \end{bmatrix}.$$

The components of γ_t^{call} and γ_t^{dist} are independent and beta-distributed, such that $(\gamma_t^{\text{call}})_i \sim \text{Beta}(\alpha_i^{\text{call}}, \beta_i^{\text{call}})$, where $\alpha_i^{\text{call}} = 2$ for $i = 1, \ldots, m$, and

$$\beta^{\text{call}} = (10.3, 10.0, 12.9, 10.5, 11.8, 10.5).$$

The distribution intensities were also beta distributed, such that $(\gamma_t^{\text{dist}})_i \sim \text{Beta}(\alpha_i^{\text{dist}}, \beta_i^{\text{dist}})$, where $\alpha_i^{\text{dist}} = 3$ for $i = 1, \ldots, m$, and

$$\beta^{\text{dist}} = (13.0, 12.7, 15.9, 12.8, 13.2, 14.2).$$

These parameters lead to typical values of call and distribution intensities around 0.14 and 0.16 respectively. The target asset values n^{tar} are chosen to be between 4 and 5, the maximum commitment is $u^{\max} = 3$, and the penalty coefficient was $\lambda = 0.01$.

Results. We carried out VGI for 20 iterations, starting from $V^1 = V^{\text{lb}}$. In each iteration, the fitting step was performed using $N = 50$ fitting points, obtained by simulating the current policy. The damping coefficient was fixed to $\rho_k = 0.5$.

Fig. 2 plots the average cost versus the number of policy evaluations used, along with the average cost of the CE-MPC policy with a horizon of $H = 30$. Our method converges to a policy that is 25% better than the CE-MPC policy. It is able to significantly outperform the CE-MPC policy because it accounts for the correlation between the returns r_t. The CE-MPC policy, on the other hand, only accounts for the average returns. The average costs were computed by simulating the system for ten thousand steps.

Fig. 3 shows an example trajectory of asset value, liability, and commitments made for one of the six asset classes, using the ADP policy found by VGI. The policy makes commitments when the asset value dips below the target value.

6.3. Supply chain optimization

In our final example, we consider the problem of shipping goods efficiently across a network of warehouses to maximize profit. We consider a single-good, multi-echelon supply chain with $\tilde{n}$ interconnected

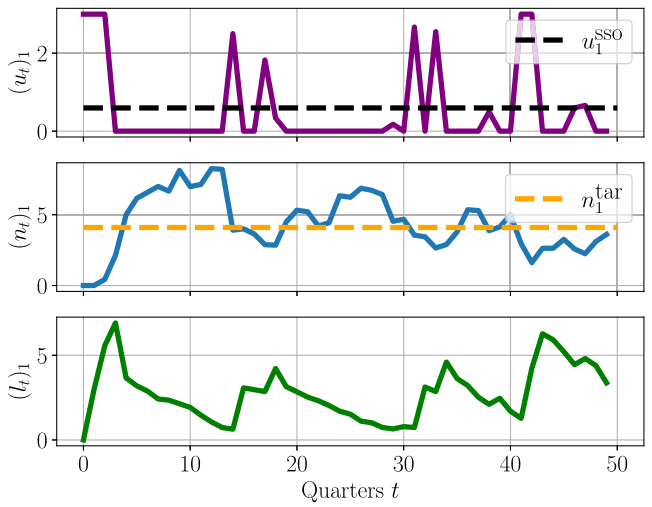

Fig. 3. Commitments, NAV, and uncalled commitments for one asset class. The policy found by VGI makes commitments when the NAV dips below the target value.

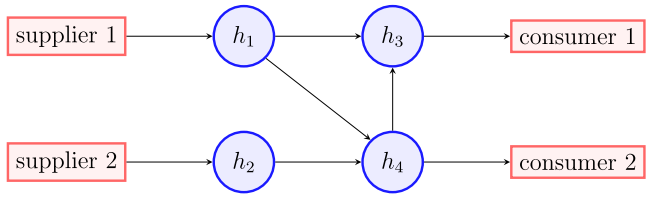

Fig. 4. Supply chain network.

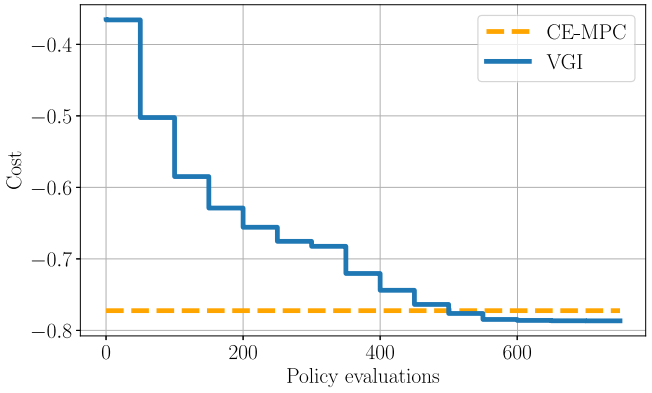

Fig. 5. VGI for the supply chain problem.

warehouses, which are represented by nodes in a graph. There are m directed links over which goods can flow; n_s links connect suppliers to nodes, n_c links connect nodes to consumers, and $m-n_s-n_c$ links connect nodes to each other.

The amount of good held at each node at time t is represented by $h_t \in \mathbf{R}_+^{\tilde{n}}$. The prices at which we can buy the good from the suppliers are denoted by $p_t \in \mathbf{R}_+^{n_s}$, the fixed prices at which goods can be sold to consumers are denoted by $r \in \mathbf{R}_+^{n_c}$, and the consumer demand is $d_t \in \mathbf{R}_+^{n_c}$. The prices and demand are random and independent between time points, but are known at time t for planning. The inputs are $b_t \in \mathbf{R}_+^{n_s}$, amounts bought from the suppliers, $s_t \in \mathbf{R}_+^{n_c}$, the amounts sold to the consumers, and $z_t \in \mathbf{R}_+^{m-n_s-n_c}$, the amounts transported across inter-node links. The dynamics are given by

$$h_{t+1} = h_t + \left(A^{\text{in}} - A^{\text{out}}\right)(b_t, s_t, z_t),$$

where $A^{\text{in}}, A^{\text{out}} \in \mathbf{R}^{n\times m}$; $A_{ij}^{\text{in (out)}}$ is 1 if link j enters (exits) node i and 0 otherwise.

The dynamics may be expressed as a random linear dynamical system with augmented state $x_t = (h_t, p_t, d_t)$, input $u_t = (b_t, s_t, z_t)$, and dynamics matrices

$$A_t = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad B_t = \begin{bmatrix} A^{\text{in}} - A^{\text{out}} \\ 0 \\ 0 \end{bmatrix}, \qquad c_t = \begin{bmatrix} 0 \\ p_{t+1} \\ d_{t+1} \end{bmatrix},$$

such that $x_t \in \mathbf{R}^n$ with $n = \tilde{n} + n_s + n_c$ and $u_t \in \mathbf{R}^m$.

The prices and demand p_t and d_t are included in the state since they are known at time t for planning. However, since they are random and independent between time points, the value function need only be a function of h_t. Moreover, we only require that the stage cost be jointly convex in (h_t, u_t).

The goal is to maximize the revenue from selling goods to customers while minimizing the material costs paid to the suppliers, transportation costs, and holding costs of the goods at each node. Let $\tau \in \mathbf{R}_+^m$ encode the costs of transporting a unit of good across each link, and $\alpha \in \mathbf{R}_+^n$ and $\beta \in \mathbf{R}_+^n$ parametrize the linear and quadratic holding costs of the goods at each node.

The stage cost is

$$g(x_t, u_t) = -r^T s_t + p_t^T b_t + \tau^T z_t + \alpha^T h_t + \beta^T h_t^2 + I(x_t, u_t),$$

where $I(x_t, u_t)$ is the indicator function that encodes the following constraints:

- The warehouses have maximum capacity $h_{\max} > 0$: $0 \le h_{t+1} \le h_{\max}$.
- The links have maximum capacity $u_{\max} > 0$: $0 \le u_t \le u_{\max}$.
- The amounts shipped out should not exceed the current capacities: $A^{\text{out}} u_t \le h_t$.
- The amounts sold to consumers cannot exceed the current demand: $s_t \le d_t$.

For this example, we find a quadratic lower bound V^{lb} on the value function by relaxing the constraints, adding the quadratic penalty $u_t^T u_t - (1/2)u_{\max}\mathbf{1}^\top u_t$ to the stage cost, and solving the resulting LQR problem. The lower bound is valid, since the added penalty is a pointwise lower bound on the indicator of the input constraints, which is zero for $0 \le u_t \le u_{\max}$, and infinity otherwise.

Numerical example. We consider a network with $\tilde{n} = 4$ warehouses, $n_s = 2$ suppliers, $n_c = 2$ consumers, and $m = 8$ links. The network is illustrated in Fig. 4. The supplier prices p_t and customer demands d_t are log-normally distributed, such that $\log p_t \sim \mathcal{N}(\mu_p, \Sigma_p)$ and $\log d_t \sim \mathcal{N}(\mu_d, \Sigma_d)$, with

$$\mu_p = (0.0, 0.1), \quad \Sigma_p = 0.4I, \quad \mu_d = (0.0, 0.4), \quad \Sigma_d = 0.4I.$$

The holding cost parameters are $\alpha = \beta = (0.01)\mathbf{1}$, the transportation cost is $\tau = (0.05)\mathbf{1}$, and the consumer prices are $r = (1.3)\mathbf{1}$. The maximum warehouse capacities are $h_{\max} = (3)\mathbf{1}$, and the maximum link capacities are $u_{\max} = (2)\mathbf{1}$.

Results. We carried out VGI for 20 iterations, starting from the quadratic lower bound V^{lb}. In each iteration, the fitting step was performed using $N = 50$ fitting points, obtained by simulating the current policy. The damping coefficient was fixed to $\rho_k = 0.5$. When solving the fitting problem, we add an ℓ_2 (or ridge) regularization, with coefficient $\lambda = 10^{-4}$.

Fig. 5 shows the average cost versus the number of policy evaluations used, along with the average cost of the CE-MPC policy with a horizon of $H = 30$. Our method converges to roughly the same cost as the CE-MPC policy.

Fig. 6 shows the storage h_t for each of the four warehouses over time, for the initial policy using V^{lb} and the final policy after VGI. The plots show average trajectories over 500 simulations, each initialized with a state in $[0, h_{\max}]^4$, chosen uniformly at random.

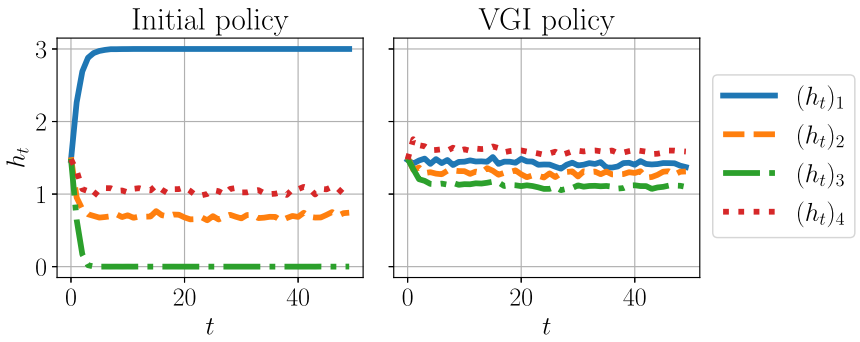

Fig. 6. Supply chain storage h_t for each warehouse over time. Left: initial ADP policy using V^{lb}. Right: final policy after VGI.

On average, the VGI policy is able to keep the storage levels close to half capacity for all warehouses. On the other hand, the initial policy tends to put too much stock in the first warehouse with storage $(h_t)_1$, which can, on average, buy goods at a lower price from the suppliers. Similarly, the policy tends to under-utilize the third warehouse with storage $(h_t)_3$, which experiences lower consumer demand than the fourth warehouse with storage $(h_t)_4$.

Table 1
Comparison of cost and number of policy evaluations used (in thousands).

Method	Box LQR		Commitments		Supply chain	
	Cost	evals. ($\times 10^3$)	Cost	evals. ($\times 10^3$)	Cost	evals. ($\times 10^3$)
VGI	32.3	2	9.1	1	−0.79	0.75
FVI	32.2	20	9.1	4	−0.77	16
COCP gradient	33.2	24	9.4	20	−0.77	70
MPC	33.3	–	11.9	–	−0.77	–

7. Comparison with other methods

In this section, we evaluate VGI against two related ADP methods for finding a quadratic approximate value function: the standard FVI described in Section 4.1 and a COCP gradient method. They are iterative methods that follow the same pattern as VGI: at each iteration, we simulate the system for N steps, and then use the resulting data to update the parameters of the quadratic approximate value function.

COCP gradient method. We compare against a gradient based method that updates the parameters θ of the ADP policy (9) using the derivatives of the cost along simulated trajectories, with respect to θ (Agrawal et al., 2020). At iteration k, the policy ϕ^k with parameters θ^k is used to simulate the system for N steps. The resulting data is used to compute an estimate of the average cost, given by

$$\hat{J}(\theta^k) = \frac{1}{N}\sum_{j=0}^{N-1} g(x_j, \phi^k(x_j)).$$

We then compute $\nabla\hat{J}(\theta^k)$ using the chain rule, and then update the parameters. This approach is known as backpropagation through time (Werbos, 1990). In our experiments, we use the projected stochastic (sub)gradient rule $\theta^{k+1} = \Pi_\Theta(\theta^k - \alpha^k \nabla\hat{J}(\theta^k))$, where Π_Θ is the projection onto Θ, and $\alpha^k > 0$ is a step size.

This approach requires derivatives of the policy with respect to its parameters. Those derivatives may be found by applying the implicit function theorem to the optimality conditions of the convex optimization problem associated with the policy (Agrawal et al., 2019, 2020). Examples of the COCP gradient method used to find quadratic approximate value functions may be found in Agrawal et al. (2020). In our experiments, we used `cvxpylayers` to compute the necessary derivatives (Agrawal et al., 2019).

7.1. Results

In general, FVI and COCP gradient methods required more tuning of hyperparameters than VGI to work well. As shown in Table 1, VGI achieves the best (or close to the best) performance in all three problems, all using far fewer policy evaluations than the FVI and COCP gradient methods. The costs were evaluated in each case by simulating the policy for ten thousand steps.

VGI used the same hyperparameters as in Section 6, *i.e.*, $\rho_k = 0.5$ and $N = 50$. The method was run for 40 iterations for the box-constrained LQR problem, 20 iterations for the commitments example, and 15 iterations for the supply chain problem.

We now discuss the hyperparameters chosen for FVI and the COCP gradient method. All methods were initialized using the same initial quadratic approximate value function. For the box-constrained LQR problem we used $V^1(x) = x^T Q x$, and for the other two problems we used $V^1(x) = V^{\mathrm{lb}}$, the quadratic lower bound on V available for each problem.

Box constrained LQR. FVI was run using $N = 400$ policy evaluations, for a total of 50 iterations. The damping parameter was $\rho_k = 0.5$, and the symmetry constraint $p = 0$ was incorporated into the fitting problem.

The COCP gradient method was run using $N = 300$ policy evaluations, for a total of 80 iterations. The 300 sample points were generated by simulating $K = 3$ trajectories each of length $T = 100$, using the procedure described in Section 5.3. We used a step size of $\alpha^k = 0.01$. The method was initialized with $P = I$, and the symmetry constraint $p = 0$ was incorporated into the fitting problem. VGI took 6 s to complete, FVI took 29 s, and the COCP gradient method took 4 min and 10 s.

Commitments planning. FVI was run using $N = 200$ policy evaluations, for a total of 20 iterations. The sample points were generated by simulating $K = 2$ trajectories each of length $T = 100$. The damping parameter was $\rho_k = 0.5$.

The COCP gradient method was run using $N = 200$ policy evaluations, for a total of 100 iterations. The sample points were generated by simulating $K = 2$ trajectories each of length $T = 100$. We used a step size of $\alpha^k = 10^{-4}$. VGI took 5 s to complete, FVI took 7 s, and the COCP gradient method took 5 min.

Supply chain. FVI was run using $N = 800$ policy evaluations, for a total of 20 iterations. The sample points were generated by simulating $K = 2$ trajectories each of length $T = 400$. The damping parameter was $\rho_k = 0.75$. An ℓ_2 regularization with coefficient $\lambda = 10^{-4}$ was used in the fitting problem.

The COCP gradient method was run using $N = 1000$ policy evaluations, for a total of 70 iterations. The sample points were generated by simulating $K = 10$ trajectories each of length $T = 100$. We used a step size of $\alpha^k = 0.01$. An ℓ_2 regularization with coefficient $\lambda = 10^{-4}$ was added to the cost. VGI took 2 s to complete, FVI took 25 s, and the COCP gradient method took 13 min.

8. Conclusion

In this work, we propose value-gradient iteration, a method for finding a quadratic approximate value function for convex stochastic control. The method is an approximation of value iteration, and we show how we may compute the gradient of the Bellman operator image to fit the gradient of the approximate value function in each iteration. By fitting the gradient of the approximate value function instead of the approximate value function itself, we can find a good policy using far less simulation data. Indeed, we find that the computational effort of obtaining a good approximate value function is comparable to that of evaluating the policy through simulation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Stephen Boyd was partially supported by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHK funding, Hong Kong SAR, and by Office of Naval Research, United States grant N00014-22-1-2121. We would also like to thank Pieter Abeel and Zico Kolter for helpful discussions and feedback.

Appendix A. Expectation of quadratic functions

Let $\hat{V}$ be a convex quadratic function of the form (10). We now show that $\mathbf{E}\hat{V}(A_t x + B_t u + c_t)$ is a convex quadratic function in u, with coefficients that may be written in terms of P, p, and π and the first and second moments of (A_t, B_t, c_t).

Let $\bar{A} = \mathbf{E}A_t$, $\bar{A} = \mathbf{E}A_t$, and $\bar{c} = \mathbf{E}c_t$ denote the expected values. Let $(A_t)_i$, $(B_t)_i$, $\bar{A}_i$, and $\bar{B}_i$ denote the ith columns of A_t, B_t, $\bar{A}$, and $\bar{B}$ respectively. Let Σ_{ij}^{AB} denote the covariance matrix between the ith column of A_t and the jth column of B_t, and let Σ_{ij}^{A} and Σ_{ij}^{B} be defined similarly. Finally, let Σ_i^{Ac} and Σ_i^{Bc} denote the covariances between the ith columns of A_t and B_t with c_t, respectively, and let Σ^c denote the covariance of c_t.

We have

$$\mathbf{E}\hat{V}(A_t x + B_t u + c_t) = \frac{1}{2}\mathbf{E}\left(\begin{bmatrix} A_t x + B_t u + c_t \\ 1 \end{bmatrix}^T \begin{bmatrix} P & p \\ p^T & \pi \end{bmatrix} \begin{bmatrix} A_t x + B_t u + c_t \\ 1 \end{bmatrix}\right).$$

Expanding terms, we obtain

$$\mathbf{E}\hat{V}(A_t x + B_t u + c_t) = \frac{1}{2}\begin{bmatrix} u \\ 1 \end{bmatrix}^T \begin{bmatrix} M & m \\ m^T & \mu \end{bmatrix} \begin{bmatrix} u \\ 1 \end{bmatrix},$$

where

$$M = \mathbf{E}B_t^T P B_t,$$
$$m = \mathbf{E}B_t^T P A_t x + \mathbf{E}B_t^T P c_t + \bar{B}^T p,$$
$$\mu = \pi + x^T \mathbf{E}(A_t^T P A_t)x + 2x^T \mathbf{E}(A_t^T P c_t) + 2x^T \bar{A}^T p + 2p^T \bar{c} + \mathbf{E}c_t^T P c_t.$$

Finally, we note that

$$\mathbf{E}c_t^T P c_t = \bar{c}^T P \bar{c} + \mathbf{Tr}(P\Sigma^c),$$

and for all indices i and j,

$$(\mathbf{E}A_t^T P A_t)_{ij} = \bar{A}_i^T P \bar{A}_j^T + \mathbf{Tr}(P\Sigma_{ij}^A),$$
$$(\mathbf{E}B_t^T P B_t)_{ij} = \bar{B}_i^T P \bar{B}_j^T + \mathbf{Tr}(P\Sigma_{ij}^B),$$
$$(\mathbf{E}B_t^T P A_t)_{ij} = \bar{B}_i^T P \bar{A}_j^T + \mathbf{Tr}(P\Sigma_{ij}^{AB}),$$
$$(\mathbf{E}A_t^T P c_t)_i = \bar{A}_i^T P \bar{c} + \mathbf{Tr}(P\Sigma_i^{Ac}),$$
$$(\mathbf{E}B_t^T P c_t)_i = \bar{B}_i^T P \bar{c} + \mathbf{Tr}(P\Sigma_i^{Bc}).$$

Appendix B. Lower bounds on quadratic functions

We say that $V_1 \geq V_2$ if $V_1(x) \geq V_2(x)$ for all $x \in \mathbf{R}^n$. We consider the case of convex quadratic functions, where for $i = 1, 2$, V_i is given by

$$V_i(x) = \frac{1}{2}\begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} P_i & p_i \\ p_i^T & \pi_i \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix},$$

where $P_i \geq 0$. Then, $V_1 \geq V_2$ holds if and only if the quadratic function $V_{12} = V_1 - V_2$ is positive semidefinite, *i.e.*

$$V_{12}(x) = \frac{1}{2}\begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} P_1 - P_2 & p_1 - p_2 \\ p_1^T - p_2^T & \pi_1 - \pi_2 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0,$$

for all $x \in \mathbf{R}^n$. The function V_{12} has a minimum value if and only if $P_1 - P_2 \geq 0$ and $p_1 - p_2$ is in the range of the matrix $P_1 - P_2$ (see *e.g.* Boyd and Vandenberghe (2004, § A.5.5)). The range condition may be written as

$$\left[I - (P_1 - P_2)(P_1 - P_2)^\dagger\right](p_1 - p_2) = 0,$$

where $(P_1 - P_2)^\dagger$ is the pseudo-inverse of $(P_1 - P_2)$. In this case, the minimum value is given by

$$\min_x V_{12}(x) = \frac{1}{2}\left(\pi_1 - \pi_2 - (p_1 - p_2)^T (P_1 - P_2)^\dagger (p_1 - p_2)\right).$$

Finally, by the generalized Schur complement, $\min_x V_{12}(x)$ exists and is nonnegative if and only if

$$\begin{bmatrix} P_1 - P_2 & p_1 - p_2 \\ p_1^T - p_2^T & \pi_1 - \pi_2 \end{bmatrix} \geq 0.$$

Appendix C. Lower bound from certainty equivalence

Solving the certainty equivalent problem involves finding a function V^{ce} that satisfies the Bellman equation

$$V^{\mathrm{ce}}(x) = \min_u \left(g(x,u) + V^{\mathrm{ce}}(\bar{A}x + \bar{B}u + \bar{c})\right).$$

By Jensen's inequality,

$$V^{\mathrm{ce}}(\bar{A}x + \bar{B}u + \bar{c}) \leq \mathbf{E}V^{\mathrm{ce}}(Ax + Bu + c).$$

Therefore, we have

$$V^{\mathrm{ce}}(x) \leq \min_u \left(g(x,u) + \mathbf{E}V^{\mathrm{ce}}(Ax + Bu + c)\right) = \mathcal{T}V^{\mathrm{ce}}(x).$$

By the monotonicity of the Bellman operator, we have

$$V^{\mathrm{ce}} \leq \mathcal{T}V^{\mathrm{ce}} \leq \lim_{k\to\infty} \mathcal{T}^k V^{\mathrm{ce}} = V^\star.$$

This implies that V^{ce} is a lower bound on the true value function.

References

Åström, K. J., Hägglund, T., Hang, C. C., & Ho, W. K. (1993). Automatic tuning and adaptation for PID controllers-A survey. *Control Engineering Practice*, *1*(4), 699–714.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., & Kolter, J. Z. (2019). Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, *32*.

Agrawal, A., Barratt, S., Boyd, S., & Stellato, B. (2020). Learning convex optimization control policies. In *Learning for dynamics and control* (pp. 361–373). PMLR.

Agrawal, A., Verschueren, R., Diamond, S., & Boyd, S. (2018). A rewriting system for convex optimization problems. *Journal of Control and Decision*, *5*(1), 42–60.

Amos, B., Jimenez, I., Sacks, J., Boots, B., & Kolter, J. Z. (2018). Differentiable MPC for end-to-end planning and control. *Advances in Neural Information Processing Systems*, *31*.

Amos, B., Stanton, S., Yarats, D., & Wilson, A. G. (2021). On the model-based stochastic value gradient for continuous reinforcement learning. In *Learning for dynamics and control* (pp. 6–20). PMLR.

Amos, B., Xu, L., & Kolter, J. Z. (2017). Input convex neural networks. In *The international conference on machine learning* (pp. 146–155). PMLR.

Antos, A., Szepesvári, C., & Munos, R. (2007). Fitted Q-iteration in continuous action-space MDPs. *Advances in Neural Information Processing Systems*, *20*.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine learning proceedings 1995* (pp. 30–37). Elsevier.

Barratt, S., & Boyd, S. (2021). Stochastic control with affine dynamics and extended quadratic costs. *IEEE Transactions on Automatic Control*, *67*(1), 320–335.

Bellman, R. (1954). The theory of dynamic programming. *American Mathematical Society. Bulletin*, *60*(6), 503–515.

Bellman, R., & Dreyfus, S. (1959). Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 247–251.

Bertsekas, D. P. (2012). *Dynamic programming and optimal control. Vol. 2* (4th ed.). Athena Scientific.

Bertsekas, D. P. (2017). *Dynamic programming and optimal control. Vol. 1* (4th ed.). Athena Scientific.

Bertsekas, D. P. (2019). *Reinforcement learning and optimal control*. Athena Scientific.

Bertsekas, D. P., Borkar, V. S., & Nedic, A. (2004). Improved temporal difference methods with linear function approximation. In *Learning and approximate dynamic programming* (pp. 231–255). New York: IEEE Press.

Bertsekas, D. P., & Shreve, S. E. (1996). *Stochastic optimal control: the discrete-time case. Vol. 5*. Athena Scientific.

Borrelli, F., Bemporad, A., & Morari, M. (2017). *Predictive control for linear and hybrid systems*. Cambridge University Press.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Camacho, E. F., & Bordons, C. (2013). *Model predictive control*. Springer Science & Business Media.

Corless, M., & Leitmann, G. (1988). Controller design for uncertain systems via Lyapunov functions. In *1988 American control conference* (pp. 2019–2025). IEEE.

Dayan, P., & Singh, S. (1995). Improving policies without measuring merits. *Advances in Neural Information Processing Systems*, *8*.

De Farias, D. P., & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, *51*(6), 850–865.

Deisenroth, M., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *The international conference on machine learning* (pp. 465–472).

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, *17*(83), 1–5.

Fairbank, M. (2008). Reinforcement learning by value gradients. arXiv preprint arXiv: 0803.3539.

Fairbank, M., & Alonso, E. (2012). Value-gradient learning. In *The IEEE international joint conference on neural networks* (pp. 1–8).

Freeman, R. A., & Primbs, J. A. (1996). Control Lyapunov functions: New ideas from an old source. In *Proceedings of 35th IEEE conference on decision and control. Vol. 4* (pp. 3926–3931). IEEE.

Garcia, C. E., Prett, D. M., & Morari, M. (1989). Model predictive control: Theory and practice—A survey. *Automatica*, *25*(3), 335–348.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction, Vol. 2*. Springer.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., & Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. *Advances in Neural Information Processing Systems*, *28*.

Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics* (pp. 492–518). Springer.

Jamieson, K., & Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics* (pp. 240–248). PMLR.

Keshavarz, A., & Boyd, S. (2014). Quadratic approximate dynamic programming for input-affine systems. *International Journal of Robust and Nonlinear Control*, *24*(3), 432–449.

Ljung, L. (1998). *System identification*. Springer.

Luxenberg, E., Boyd, S., van Beek, M., Cao, W., & Kochenderfer, M. (2022). Strategic asset allocation with illiquid alternatives. In *Proceedings of the third ACM international conference on AI in finance* (pp. 249–256).

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

Merton, R. C. (1969). Lifetime portfolio selection under uncertainty: The continuous-time case. *The Review of Economics and Statistics*, 247–257.

Minorsky, N. (1922). Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, *34*(2), 280–309.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *The international conference on machine learning* (pp. 1928–1937). PMLR.

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

O'Donoghue, B., Wang, Y., & Boyd, S. (2011). Min-max approximate dynamic programming. In *The IEEE international symposium on computer-aided control system design* (pp. 424–431). IEEE.

Pontryagin, L. S. (1987). *Mathematical theory of optimal processes*. CRC Press.

Powell, W. B. (2007). *Approximate dynamic programming: solving the curses of dimensionality, Vol. 703*. John Wiley & Sons.

Prokhorov, D. V., & Wunsch, D. C. (1997). Adaptive critic designs. *The IEEE Transactions on Neural Networks*, *8*(5), 997–1007.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rockafellar, R. T. (1970). *Convex analysis*: *Vol. 11*, Princeton University Press.

Schaller, M., Banjac, G., Diamond, S., Agrawal, A., Stellato, B., & Boyd, S. (2022). Embedded code generation with CVXPY. *IEEE Control Systems Letters*, *6*, 2653–2658.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings* (pp. 216–224). Elsevier.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT Press.

Tikhonov, A. N., & Arsenin, V. I. (1977). *Solutions of ill-posed problems*. Wiley.

Tsitsiklis, J. N., & Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, *22*(1), 59–94.

Wang, Y., & Boyd, S. (2009). Performance bounds for linear stochastic control. *Systems & Control Letters*, *58*(3), 178–182.

Wang, Y., & Boyd, S. (2010). Fast evaluation of quadratic control-Lyapunov policy. *IEEE Transactions on Control Systems Technology*, *19*(4), 939–946.

Wang, Y., O'Donoghue, B., & Boyd, S. (2015). Approximate dynamic programming via iterated Bellman inequalities. *International Journal of Robust and Nonlinear Control*, *25*(10), 1472–1496.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Werbos, P. J. (1999). Stable adaptive control using new critic designs. In *Ninth workshop on virtual intelligence/dynamic neural networks, Vol. 3728* (pp. 510–579). SPIE.

White, D. J. (1969). *Dynamic programming. Vol. 1*. Oliver & Boyd Edinburgh.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal Of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), 301–320.