INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL
*Int. J. Robust. Nonlinear Control* (2012)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/rnc.2894

# Quadratic approximate dynamic programming for input-affine systems

Arezou Keshavarz*,† and Stephen Boyd

*Electrical Engineering, Stanford University, Stanford, CA, USA*

SUMMARY

We consider the use of quadratic approximate value functions for stochastic control problems with input-affine dynamics and convex stage cost and constraints. Evaluating the approximate dynamic programming policy in such cases requires the solution of an explicit convex optimization problem, such as a quadratic program, which can be carried out efficiently. We describe a simple and general method for approximate value iteration that also relies on our ability to solve convex optimization problems, in this case, typically a semidefinite program. Although we have no theoretical guarantee on the performance attained using our method, we observe that very good performance can be obtained in practice. Copyright © 2012 John Wiley & Sons, Ltd.

Received 10 February 2012; Revised 21 May 2012; Accepted 29 July 2012

KEY WORDS: approximate dynamic programming; stochastic control; convex optimization

## 1. INTRODUCTION

We consider stochastic control problems with input-affine dynamics and convex stage costs, which arise in many applications in nonlinear control, supply chain, and finance. Such problems can be solved in principle using dynamic programming (DP), which expresses the optimal policy in terms of an optimization problem involving the value function (or a sequence of value functions in the time-varying case), which is a function on the state space $\mathbf{R}^n$. The value function (or functions) can be found, in principle, by an iteration involving the Bellman operator, which maps functions on the state space into functions on the state space and involves an expectation and a minimization step; see [1] for an early work on DP and [2–4] for a more detailed overview. For the special case when the dynamics is linear and the stage cost is convex quadratic, DP gives a complete and explicit solution of the problem, because in this case, the value function is also quadratic and the expectation and minimization steps both preserve quadratic functions and can be carried out explicitly.

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

*Correspondence to: Arezou Keshavarz, Electrical Engineering, Stanford University, Packard 243, 350 Serra Mall, Stanford, CA 94305, USA.
†E-mail: arezou@stanford.edu

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

There are many methods for finding an appropriate approximate value function; for an overview on ADP, see [3]. The approximate value function can be chosen as the true value function of a simplified version of the stochastic control problem, for which we can easily obtain the value function. For example, we can form a linear–quadratic approximation of the problem and use the (quadratic) value function obtained for the approximation as our approximate value function. Another example used in model predictive control (MPC) or certainty-equivalent roll out (see, e.g., [5]) is to replace the stochastic terms in the original problem with constant values, say, their expectations. This results in an ordinary optimization problem, which (if it is convex) we can solve to obtain the value function. Recently, Wang *et al.* have developed a method that uses convex optimization to compute a quadratic lower bound on the true value function, which provides a good candidate for an approximate value function and also gives a lower bound on the optimal performance [6, 7]. This was extended in [8] to use as approximate value function the pointwise supremum of a family of quadratic lower bounds on the value function. These methods, however, cannot be used for general input-affine convex problems.

Approximate dynamic programming is closely related to inverse optimization, which has been studied in various fields such as control [9, 10], robotics [11, 12], and economics [13–15]. In inverse optimization, the goal is to find (or estimate or impute) the objective function in an underlying optimizing process, given observations of optimal (or nearly optimal) actions. In a stochastic control problem, the objective is related to the value function, so the goal is to approximate the value function, given samples of optimal or nearly optimal actions. Watkins *et al.* introduced an algorithm called Q-learning [16, 17], which observes the action–state pair at each step and updates a quality function on the basis of the obtained reward/cost.

Model predictive control, also known as receding-horizon control or rolling-horizon planning [18–20], is also closely related to ADP. In MPC, at each step, we plan for the next $T$ time steps, using some estimates or predictions of future unknown values and then execute only the action for the current time step. At the next step, we solve the same problem again, now using the value of the current state and updated values of the predictions based on any new information available. MPC can be interpreted as ADP, where the approximate value function is the optimal value of an optimization problem.

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

To choose a quadratic approximate value function, we use the same Bellman operator iteration that would result in the true value function (when some technical conditions hold), but we replace the Bellman operator with an approximation that maps quadratic functions to quadratic functions. The idea of value iteration, followed by a projection step back to the subset of candidate approximate value functions, is an old one, explored by many authors [3, Chapter 6; 21, Chapter 12; 22]. In the reinforcement learning community, two closely related methods are fitted value iteration and fitted Q-iteration [23–27]. Most of these works consider problems with continuous state space and finite action space, and some of them come with theoretical guarantees on convergence and performance bounds. For instance, the authors in [23] proposed a family of tree-based methods for fitting approximate Q-functions and provided convergence guarantees and performance bounds for some of the proposed methods. Riedmiller considered the same framework but employed a multilayered perceptron to fit (model-free) Q-functions and provided empirical performance results [24]. Munos and



DOI: 10.1002/rnc

Szepesvari developed finite-time bounds for the fitted value iteration for discounted problems with bounded rewards [26]. Antos *et al.* extended this work to continuous action space and developed performance bounds for a variant of the fitted Q-iteration by using stationary stochastic policies, where greedy action selection is replaced with searching on a restricted set of candidate policies [27].

There are also approximate versions of policy iteration and the linear programming formulation of DPs [28, 29], where the value function is replaced by a linear combination of a set of basis functions. Lagoudakis *et al.* considered the discounted finite state case and proposed a projected policy iteration method [30] that minimizes the $\ell_2$ Bellman residual to obtain an approximate value function and policy at each step of the algorithm. Tsitsiklis and Van Roy established strong theoretical bounds on the suboptimality of the ADP policy, for the discounted cost finite state case [31, 32].

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

We rely on our ability to (numerically) solve convex optimization problems with great speed and reliability. Recent advances have shown that using custom-generated solvers will speed up computation by orders of magnitude [33–38]. With solvers that are orders of magnitude faster, we can carry out approximate value iteration for many practical problems and obtain approximate value functions that are simple yet achieve very good performance.

The method we propose comes with no theoretical guarantees. There is no theoretical guarantee that the modified Bellman iterations will converge, and when they do (or when we simply terminate the iterations early), we do not have any bound on how suboptimal the resulting control policy is. On the other hand, we have found that the methods described in this paper work very well in practice. In cases in which the performance bounding methods of Wang *et al.* can be applied, we can confirm that our quadratic ADP policies are very good and, in many cases, nearly optimal, at least for specific cases.

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

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

*Outline.* In Section 2, we describe three variations on the stochastic control problem. In Section 3, we review the DP solution of the problems, including the special case when the dynamics is linear and the stage cost functions are convex quadratic, as well as quadratic ADP, which is the method we propose. We describe a projected value iteration method for obtaining an approximate value function in Section 4. In the remaining three sections, we describe numerical examples: a traditional control problem with bounded inputs (Section 5), a multiperiod portfolio optimization problem (Section 6), and a vehicle control problem (Section 7).

## 2. INPUT-AFFINE CONVEX STOCHASTIC CONTROL

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



DOI: 10.1002/rnc

### 2.1. The common model

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

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

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

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

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

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

*Closed-loop dynamics.* We consider state feedback policies, that is,

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

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

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

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

*Stage cost.* The stage cost is given by $\ell_t(z, v)$, where $\ell_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$ is an extended valued closed convex function. We use infinite stage cost to denote unallowed state–input pairs, that is, (joint) constraints on $z$ and $v$. We will assume that for each state $z$, there exists an input $v$ with finite stage cost. The stage cost is time-invariant if $\ell_t$ does not depend on $t$, in which case we write it as $\ell$.

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

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

where

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

that is, it is positive semidefinite.

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



DOI: 10.1002/rnc

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

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

### 2.2. The problems

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

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

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

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

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

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

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

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

## 3. DYNAMIC AND APPROXIMATE DYNAMIC PROGRAMMING

The stochastic control problems described earlier can be solved in principle using DP, which we sketch here for future reference. See [3, 4, 39] for (much) more details. DP makes use of a function (time-varying in the finite-horizon case) $V : \mathbf{R}^n \to \mathbf{R}$ that characterizes the cost of starting from that state, when an optimal policy is used. The definition of $V$, its characterization, and methods for computing it differ (slightly) for the three stochastic control problems we consider.

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

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

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

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

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



DOI: 10.1002/rnc

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

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

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

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

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

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

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

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

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

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

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

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

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

The value function cannot be constructed using a simple recursion, as in the finite-horizon case, but it is characterized as the unique solution of the Bellman fixed-point equation $\mathcal{T}V = V$, where the Bellman operator $\mathcal{T}$ for the discounted infinite-horizon case is

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

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

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

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

which is called value iteration. Under certain technical conditions, this iteration converges to $V$ [1]; see [39, Chapter 9] for a detailed analysis of the convergence of value iteration.

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

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

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

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

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

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



DOI: 10.1002/rnc

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

Under some technical assumptions [3, 4], the value function $V$ and the optimal cost $J^\star$ can be found (in principle) by several methods, including value iteration, which has the form

$$\begin{aligned} J^{k+1} &= \mathcal{T}V^k\left(x^{\mathrm{ref}}\right), \\ V^{k+1} &= \mathcal{T}V^k - J^{k+1}, \end{aligned} \tag{3}$$

for $k = 1, \ldots$. $V^k$ converges to $V$, and $J$ converges to $J^\star$ [3,40].

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

$$\phi^\star(z) = \arg\min_v \left(\ell(z, v) + \mathbf{E}V(f(z) + g(z)v)\right).$$

### *3.1. Linear–convex dynamic programming*

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

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

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

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

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

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

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

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

is convex because $\ell$ is convex, and convexity is preserved under partial minimization [41, Section 3.2.5].

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

Evaluating the optimal policy at $x_t$, that is, minimizing

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

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

### *3.2. Linear–quadratic dynamic programming*

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

One big difference, however, is that the Bellman iteration can actually be carried out in the linear–quadratic case, using an explicit representation of convex quadratic functions, and standard linear



DOI: 10.1002/rnc

algebra operations. The optimal policy is affine, with coefficients we can compute. To simplify notation, we consider the discounted infinite-horizon case; the other two cases are very similar.

We denote the set of convex quadratic functions on $\mathbf{R}^n$, that is, those with the form

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

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

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

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

The stage cost $\ell$ is convex and quadratic, that is,

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

where

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

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

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

where

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

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

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

where

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

is the Schur complement of $M$ (with respect to its $(2,2)$ block); when $M_{22}$ is singular, we can replace $M_{22}^{-1}$ with the pseudo-inverse $M_{22}^{\dagger}$ [41, Section A5.5]. Furthermore, because

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

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

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



DOI: 10.1002/rnc

### 3.3. Approximate dynamic programming

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

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

for the finite-horizon case,

$$\hat{\phi}(x) = \arg\min_u \left(\ell(x,u) + \gamma\mathbf{E}\hat{V}(f(x) + g(x)u)\right)$$

for the discounted infinite-horizon case, and

$$\hat{\phi}(x) = \arg\min_u \left(\ell(x,u) + \mathbf{E}\hat{V}(f(x) + g(x)u)\right)$$

for the average-cost case.

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

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

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

### 3.4. Quadratic approximate dynamic programming

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

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

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

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

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

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

$$\hat{\phi}(z) = \arg\min_v \left(\ell(z,v) + \mathbf{E}\hat{V}(f_t(z) + g_t(z)v)\right)$$

can be evaluated by solving a QP.

## 4. PROJECTED VALUE ITERATION

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



DOI: 10.1002/rnc

not minimize any distance measure). The resulting method for computing an approximate value function is called projected value iteration.

Projected value iteration has been explored by many authors [3, Chapter 6; 21, Chapter 12; 22; 25; 27]. Here, we consider scenarios in which the state space and action space are continuous and establish a framework to carry out projected value iteration for those problems.

Projected value iteration has the following form for our three problems. For the finite-horizon problem, we start from $\hat{V}_{T+1} = 0$ and compute approximate value functions as

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

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

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

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

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

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

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

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

### *4.1. Quadratic projected iteration*

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

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

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

with variable $V$. In other words, we simply fit a convex quadratic function to the values of the Bellman operator at the sample points. This requires the solution of a semidefinite program [10, 42].

Many variations on this basic projection method are possible. We can use any convex fitting criterion instead of a least squares criterion, for example, the sum of absolute errors. We can also add additional constraints on $V$ that can represent prior knowledge (beyond convexity). For example, we may have a quadratic lower bound on the true value function, and we can add this condition as a stronger constraint on $V$ than simply convexity. (Quadratic lower bounds can be found using the methods described in [6, 43].)

Another variation on the simple fitting method described earlier adds proximal regularization to the fitting step. This means that we add a term of the form $(\rho/2)\|V - V^k\|_{\mathrm{F}}^2$ to the objective, which



DOI: 10.1002/rnc

penalizes deviations of $V$ from the previous value. Here, $\rho \geqslant 0$ is a parameter, and the norm is the Frobenius norm of the coefficient matrices,

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

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

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

## 5. INPUT-CONSTRAINED LINEAR–QUADRATIC CONTROL

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

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

which has the form used in this paper with

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

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

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

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

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

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

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

*Results.* Figure 1 shows the performance $J^k$ of the ADP policy (evaluated by a simulation run of 10,000 steps), after each round of projected value iteration. The performance of the policies obtained using the naive parameter choices are denoted by $J^{(k)}_{\mathrm{naive}}$. For this problem, we can evaluate a lower bound $J_{\mathrm{lb}} = 785.4$ on $J^*$ by using the method of [6]. This shows that the performance of our ADP



DOI: 10.1002/rnc

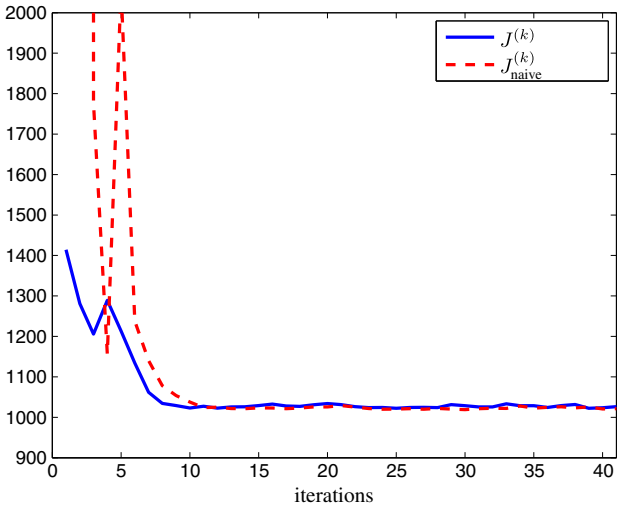


Figure 1. $J^{(k)}$ (solid line) and $J^{(k)}_{\text{naive}}$ (dashed line).

policy at termination is at most 24% suboptimal (but likely much closer to optimal). The plots show that the naive initialization only results in a few more projected value iteration steps. Close examination of the plots shows that the performance need not improve in each step. For example, the best performance (but only by a very small amount) with the more sophisticated initialization is obtained in iteration 9.

*Computation timing.* The timing results here are reported for a 3.4-GHz Intel Xeon processor running Linux. Evaluating the ADP policy requires solving a small QP with 10 variables and 8 constraints. We use CVGXEN [38] to generate custom C code for this QP, which executes in around 50 $\mu$s. The simulation of 1000 time steps, which we use in each projected value iteration, requires around 0.05 s. The fitting is carried out using CVX [44] and takes around 1 s (this time could be speeded up considerably).

## 6. MULTIPERIOD PORTFOLIO OPTIMIZATION

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

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

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

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

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

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

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

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



DOI: 10.1002/rnc

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

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

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

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

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

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

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

*Results.* Figure 2 demonstrates the performance of the ADP policies, where $J^{(k)}$ corresponds to the ADP policy at iteration $k$ of projected value iteration. In each case, we run 1000 Monte Carlo simulations, each for 1000 steps, and report the average total discounted cost across the 1000 time steps. We also plot a lower bound $J_{\mathrm{lb}}$ obtained using the methods described in [43, 45]. We can see that after around 80 steps of projected value iteration, we arrive at a policy that is nearly optimal. It is also interesting to note that it takes around 40 projected value iteration steps before we produce a policy that is profitable (i.e., that has $J \leqslant 0$).

*Computation timing.* Evaluating the ADP policy requires solving a small QP with 10 variables and 10 constraints. We use CVGXEN [38] to generate custom C code for this QP, which executes in around 22 $\mu$s. The simulation of 1000 time steps, which we use in each projected value iteration, requires around 0.022 s. The fitting is carried out using CVX [44] and takes around 1.2 s (this time could be speeded up considerably).

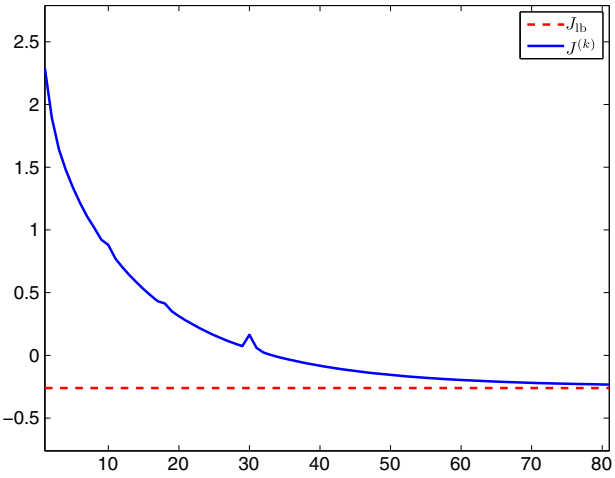


Figure 2. $\hat{J}^{(k)}$ (solid line) and a lower bound $J_{\mathrm{lb}}$ (dashed line).



DOI: 10.1002/rnc

## 7. VEHICLE CONTROL

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

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

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

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

with

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

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

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

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

which has an input-affine form with

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

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

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

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

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

*Numerical instance.* We consider a vehicle with unit mass $m = 1$ and moment of inertia $I = 1/150$. There are $k = 4$ inputs applied to the vehicle, shown in Figure 3, at distance $r = 1/40$ to the center of mass of the vehicle. We consider $T = 151$ time steps, with a discretization epoch of $h = \pi/150$. The matrices $C$ and $D$ are

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

The maximum allowable input is $u_{\max} = (1.5, 2.5, 1.5, 2.5)$, and the input cost matrix is $R = I/1000$. The state cost matrix $Q$ is a diagonal matrix, chosen as follows: we first choose the entries of $Q$ to normalize the range of the state variables, based on $x^{\mathrm{des}}$. Then we multiply the weight of the first and second states by 10 to penalize a deviation in the position more than a deviation in the speed or orientation of the vehicle.



DOI: 10.1002/rnc

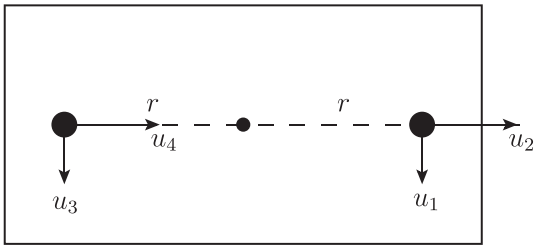


Figure 3. Vehicle diagram. The forces are applied at two positions, shown with large circles. At each position, there are two forces applied: a lateral force ($u_1$ and $u_3$) and a longitudinal force ($u_2$ and $u_4$).

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

*Results.* We calculate the average total cost by using Monte Carlo simulation with 1000 samples, that is, 1000 simulations of 151 steps. We compare the performance of two ADP policies: the ADP policy resulting from $\hat{V}_t$ results in $J = 106$, and the ADP policy resulting from $V_t^{\mathrm{quad}}$ results in $J^{\mathrm{quad}} = 245$. Here, $V_t^{\mathrm{quad}}$ is the quadratic value function obtained by linearizing the dynamics along the desired trajectory and ignoring the actuator limits. Figure 4 shows two sample trajectories generated using $\hat{V}_t$ and $V_t^{\mathrm{quad}}$.

*Computation timing.* Evaluating the ADP policy requires solving a small QP with four variables and eight constraints. We use CVGXEN [38] to generate custom C code for this QP, which executes in around 20 $\mu$s. The simulation of 1000 time steps, which we use in each projected value iteration, requires around 0.02 s. The fitting is carried out using CVX [44] and takes around 0.5 s (this time could be speeded up considerably).

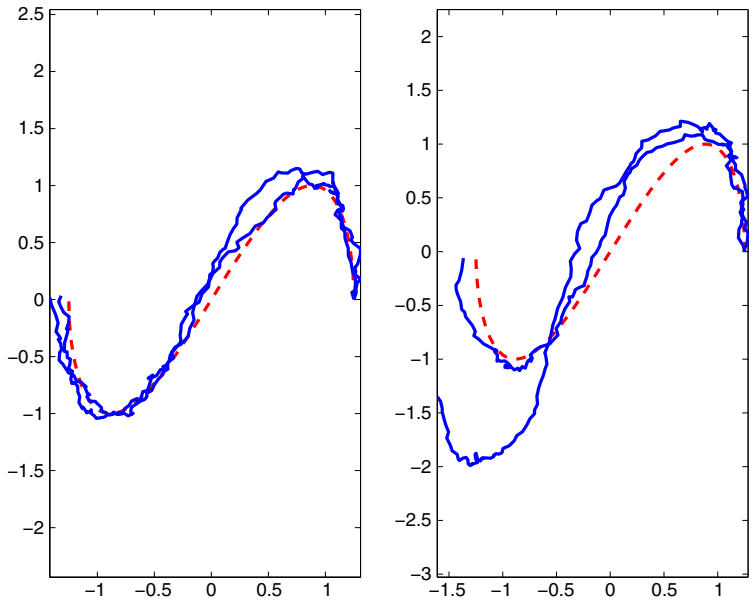


Figure 4. Desired trajectory (dashed line), two sample trajectories obtained using $\hat{V}_t$, $t = 1, \ldots, T$ (left), and two sample trajectories obtained using $V_t^{\mathrm{quad}}$ (right).



DOI: 10.1002/rnc

## 8. CONCLUSIONS

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

## APPENDIX: EXPECTATION OF QUADRATIC FUNCTION

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

$$\mathbf{E}\, h(f_t(z) + g_t(z)v) = (1/2) \begin{bmatrix} v \\ 1 \end{bmatrix}^T \begin{bmatrix} A & a \\ a^T & b \end{bmatrix} \begin{bmatrix} v \\ 1 \end{bmatrix}. \tag{4}$$

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

Suppose $h$ has the form

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

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

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

Taking expectation, we see that $\mathbf{E}\, h(f + gv)$ takes the form of (4), with coefficients given by

$$A_{ij} = \sum_{k,l} P_{kl}\, \mathbf{E}(g_{ki} g_{lj}), \quad i, j = 1, \ldots, n$$

$$a_i = \sum_j p_j\, \mathbf{E}(g_{ji}), \quad i = 1, \ldots, n$$

$$b = q + \sum_{ij} P_{ij}\, \mathbf{E}(f_i f_j) + 2 \sum_i p_i\, \mathbf{E}(f_i).$$

ACKNOWLEDGEMENTS

We would like to thank an anonymous reviewer for pointing us to the literature in reinforcement learning on fitted value iteration and fitted Q-iteration algorithms.

## REFERENCES

1. Bellman R. *Dynamic Programming*. Princeton University Press: Princeton, NJ, USA, 1957.
2. Bertsekas D. *Dynamic Programming and Optimal Control: Volume 1*. Athena Scientific: Nashua, NH, USA, 2005.
3. Bertsekas D. *Dynamic Programming and Optimal Control: Volume 2*. Athena Scientific: Nashua, NH, USA, 2007.
4. Puterman M. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.: New York, NY, USA, 1994.



DOI: 10.1002/rnc

5. Bertsekas DP, Casta nón DA. Rollout Algorithms for Stochastic Scheduling Problems. *Journal of Heuristics* 1999; **5**:89–108.
6. Wang Y, Boyd S. Performance bounds for linear stochastic control. *Systems and Control Letters* March 2009; **58**(3):1780–182.
7. Wang Y, Boyd S. Approximate dynamic programming via iterated bellman inequalities, 2011. Manuscript.
8. O'Donoghue B, Wang Y, Boyd S. Min-max approximate dynamic programming. In *Proceedings IEEE Multi-conference on Systems and Control*, September 2011; 424–431.
9. Keshavarz A, Wang Y, Boyd S. Imputing a convex objective function. In *Proceedings of the IEEE Multi-Conference on Systems and Control*, September 2011; 613–619.
10. Boyd S, El Ghaoui L, Feron E, Balakrishnan V. *Linear matrix inequalities in systems and control theory*. SIAM Books: Philadelphia, 1994.
11. Ng A, Russell S. Algorithms for inverse reinforcement learning. In *Proceedings of 17th International Conference on Machine Learning*. Morgan Kauffman: San Francisco, CA, USA, 2000; 663–670.
12. Abbeel P, Coates A, Ng A. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research* 2010; **29**:1608–1639.
13. Ackerberg D, Benkard C, Berry S, Pakes A. Econometric tools for analyzing market outcomes. In *Handbook of Econometrics,* volume 6 of *Handbook of Econometrics*, Heckman J, Leamer E (eds), chapter 63. Elsevier: North Holland, 2007.
14. Bajari P, Benkard C, Levin J. Estimating dynamic models of imperfect competition. *Econometrica* 2007; **75**(5):1331–1370.
15. Nielsen T, Jensen F. Learning a decision maker's utility function from (possibly) inconsistent behavior. *Artificial Intelligence* 2004; **160**:53–78.
16. Watkins C. Learning from delayed rewards. *Ph.D. Thesis*, Cambridge University, Cambridge, England, 1989.
17. Watkins J, Dayan P. Technical note: Q-learning. *Machine Learning* 1992; **8**(3):279–292.
18. Kwon WH, Han S. *Receding Horizon Control*. Springer-Verlag: London, UK, 2005.
19. Whittle P. *Optimization Over Time*. John Wiley & Sons, Inc.: New York, NY, USA, 1982.
20. Goodwin GC, Seron MM, De Doná JA. *Constrained Control and Estimation*. Springer: New York, USA, 2005.
21. Judd KL. *Numerical Methods in Economics*. MIT Press: Cambridge, Massachusetts, USA, 1998.
22. Powell W. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, Wiley Series in Probability and Statistics. John Wiley & Sons: Hoboken, NJ, USA, 2007.
23. Ernst D, Geurts P, Wehenkel L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 2005; **6**:504–556.
24. Riedmiller M. Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method. In *16th European Conference on Machine Learning*. Springer: Porto, Portugal, 2005; 317–328.
25. Munos R. Performance bounds in $L_p$-norm for approximate value iteration. *SIAM Journal on Control and Optimization* 2007; **46**(2):541–561.
26. Munos R, Szepesvari C. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research* 2008; **9**:815–857.
27. Antos A, Munos R, Szepesvari C. Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems 20*, 2008; 9–16.
28. De Farias D, Van Roy B. The linear programming approach to approximate dynamic programming. *Operations Research* 2003; **51**(6):850–865.
29. Desai V, Farias V, Moallemi C. Approximate dynamic programming via a smoothed linear program. *Operations Research* May–June 2012; **60**(3):655–674.
30. Lagoudakis M, Parr R, Bartlett L. Least-squares policy iteration. *Journal of Machine Learning Research* 2003; **4**:1107–1149.
31. Tsitsiklis JN, Roy B. V. An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control* 1997; **42**:674–690.
32. Van Roy B. Learning and value function approximation in complex decision processes. *Ph.D. Thesis*, Department of EECS, MIT, 1998.
33. Bemporad A, Morari M, Dua V, Pistikopoulos E. The explicit linear quadratic regulator for constrained systems. *Automatica* January 2002; **38**(1):3–20.
34. Wang Y, Boyd S. Fast model predictive control using online optimization. In *Proceedings of the 17th IFAC World Congress*, 2008; 6974–6997.
35. Mattingley JE, Boyd S. Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine* 2009; **23**(3):50–61.
36. Mattingley J, Wang Y, Boyd S. Code generation for receding horizon control. In *IEEE Multi-Conference on Systems and Control*, 2010; 985–992.
37. Mattingley JE, Boyd S. Automatic code generation for real-time convex optimization. In *Convex Optimization in Signal Processing and Communications*, Palomar DP, Eldar YC (eds). Cambridge University Press: New York, USA, 2010; 1–41.
38. Mattingley J, Boyd S. CVXGEN: a code generator for embedded convex optimization. *Optimization and Engineering* 2012; **13**(1):1–27.




DOI: 10.1002/rnc

39. Bertsekas D, Shreve S. *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific: Belmont, Massachusetts, USA, 1978.
40. Bertsekas D. A value iteration method for the average cost dynamic programming problem, 1995.
41. Boyd S, Vandenberghe L. *Convex optimization*. Cambridge University Press: New York, USA, 2004.
42. Vandenberghe L, Boyd S. Semidefinite programming. *SIAM Review* 1996; **38**(1):49–95.
43. Wang Y, Boyd S. Fast evaluation of quadratic control-Lyapunov policy. *IEEE Transactions on control Systems Technology* June 2010; **PP**(99):1–8.
44. Grant M, Boyd S, Ye Y. CVX: Matlab software for disciplined convex programming, 2006. (Available from: http://cvxr.com/cvx/) [Accessed on July 2011].
45. Boyd S, Mueller M, ODonoghue B, Wang Y. Performance bounds and suboptimal policies for multi-period investment, 2012. Manuscript.



DOI: 10.1002/rnc