Proceedings of Machine Learning Research vol 144:1–13, 2021

Approximate Midpoint Policy Iteration for Linear Quadratic Control

Benjamin Gravell BENJAMIN.GRAVELL@UTDALLAS.EDU
University of Texas at Dallas

Iman Shames IMAN.SHAMES@UNIMELB.EDU.AU
University of Melbourne

Tyler Summers TYLER.SUMMERS@UTDALLAS.EDU
University of Texas at Dallas

Abstract

We present a midpoint policy iteration algorithm to solve linear quadratic optimal control problems in both model-based and model-free settings. The algorithm is a variation of Newton's method, and we show that in the model-based setting it achieves *cubic* convergence, which is superior to standard policy iteration and policy gradient algorithms that achieve quadratic and linear convergence, respectively. We also demonstrate that the algorithm can be approximately implemented without knowledge of the dynamics model by using least-squares estimates of the state-action value function from trajectory data, from which policy improvements can be obtained. With sufficient trajectory data, the policy iterates converge cubically to approximately optimal policies, and this occurs with the same available sample budget as the approximate standard policy iteration. Numerical experiments demonstrate effectiveness of the proposed algorithms.

Keywords: Optimal control, linear quadratic regulator (LQR), optimization, Newton method, midpoint Newton method, data-driven, model-free.

1. Introduction

With the recent confluence of reinforcement learning and data-driven optimal control, there is renewed interest in fully understanding convergence, sample complexity, and robustness in both "model-based" and "model-free" algorithms. Linear quadratic problems in continuous spaces provide benchmarks where strong theoretical statements can be made. In "model-based" methods, trajectory data is used to estimate a model of the system dynamics, then an approximately optimal control policy is computed based on the estimated model, invoking certainty-equivalence or using robust control approaches to explicitly account for model uncertainty. The analysis in recent works by Mania et al. (2019); Oymak and Ozay (2019); Coppens and Patrinos (2020); Dean et al. (2018, 2019); Gravell and Summers (2020); Coppens et al. (2020) have focused on providing finite-sample performance/suboptimality guarantees. "Model-free" methods do not explicitly learn a model of the dynamics, but instead optimize the control policy directly or learn a value function from which policies are constructed. For example, policy gradient has received attention recently for standard LQR (Fazel et al. (2018); Bu et al. (2020)), multiplicative-noise LQR (Gravell et al. (2019)), Markov jump LQR (Jansch-Porto et al. (2020)), and LQ games related to $\mathcal{H}_\infty$ robust control (Zhang et al. (2019); Bu et al. (2019)). Approximate policy iteration has also been studied by Bradtke et al. (1994); Krauth et al. (2019); Luo et al. (2020); Gravell et al. (2021); Al-Tamimi et al. (2007); Fazel et al. (2018); Bu et al. (2020) (note that this method is sometimes called quasi-Newton or Q-learning).

It has been long-known, but perhaps underappreciated, that application of Newton's method to find the root of the functional Bellman equation in stochastic optimal control is equivalent to the dynamic programming algorithm of policy iteration (Puterman and Brumelle (1979); Madani (2002)). In linear-quadratic problems, the Bellman equation becomes a matrix algebraic Riccati equation, and application of the Newton method to the Riccati equation yields the well-known Kleinman-Hewer algorithm.[1] The Newton method has many variations devised to improve the convergence rate and information efficiency, including higher-order methods e.g. Halley (Cuyt and Rall (1985)), and multi-point methods (Traub (1964)), which compute derivatives at multiple points and of which the midpoint method is the simplest member. Some of these have been applied to solving Riccati equations by Anderson (1978); Guo and Laub (2000); Damm and Hinrichsen (2001); Freiling and Hochhaus (2004); Hernández-Verón and Romero (2018), but without consideration of the situation when the dynamics are not perfectly known. Our main contributions are:

1. We present a midpoint policy iteration algorithm to solve linear quadratic optimal control problems when the dynamics are both known (Algorithm 1) and unknown (Algorithm 4).
2. We demonstrate that the method converges, and does so at a faster *cubic* rate than standard policy iteration or policy gradient, which converge at quadratic and linear rates, respectively.
3. We show that approximate midpoint policy iteration converges faster in the model-free setting even with the same available sample budget as the approximate standard policy iteration.
4. We present numerical experiments that illustrate and demonstrate the effectiveness of the algorithms and provide an open-source implementation to facilitate their wider use.

An extended version of this paper (Gravell et al. (2020)) is available at
https://arxiv.org/abs/2011.14212.

2. Preliminaries

Notation: Let $\mathbb{R}^{n\times m}$ denote the space of real-valued $n \times m$ matrices, $\mathbb{S}^n$ the space of symmetric real-valued $n \times n$ matrices, $\rho(M)$ the spectral radius of square matrix M, $\|M\|$ the spectral norm of matrix M, $\mathrm{svec}(M)$ the vector formed by stacking columns of the upper triangular part of matrix M with off-diagonal entries multiplied by $\sqrt{2}$, $\mathrm{smat}(v)$ the matrix formed by the inverse operation of $\mathrm{svec}(\cdot)$ such that $\mathrm{smat}(\mathrm{svec}(M)) = M$, $M \succ (\succeq)\ 0$ that matrix M is positive (semi)definite, and $M \succ (\succeq)\ N$ that matrix $M - N \succ (\succeq)\ 0$.

The infinite-horizon average-cost time-invariant linear quadratic regulator (LQR) problem is

$$\begin{aligned} \underset{\pi\in\Pi}{\text{minimize}} \quad & \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{x_0,w_t} \sum_{t=0}^{T} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}, \\ \text{subject to} \quad & x_{t+1} = Ax_t + Bu_t + w_t, \end{aligned} \tag{1}$$

where $x_t \in \mathbb{R}^n$ is the system state, $u_t \in \mathbb{R}^m$ is the control input, and w_t is i.i.d. process noise with zero mean and covariance matrix W. The state-to-state system matrix $A \in \mathbb{R}^{n\times n}$ and input-to-state system matrix $B \in \mathbb{R}^{n\times m}$ may or may not be known; we present algorithms for both settings. The optimization is over the space Π of (measurable) history dependent feedback policies $\pi = \{\pi_t\}_{t=0}^{\infty}$ with $u_t = \pi_t(x_{0:t}, u_{0:t-1})$. The penalty weight matrix $Q \in \mathbb{S}^{n+m}$ has blocks Q_{xx}, Q_{uu}, $Q_{xu} = Q_{ux}^\mathsf{T}$ which quadratically penalize deviations of the state, input, and product of state and input from the

1. Kleinman (1968) introduced this for continuous-time systems, and Hewer (1971) studied it for discrete-time systems.

origin, respectively. We assume the pair (A, B) is stabilizable, the pair $(A, Q_{xx}^{1/2})$ is detectable, and the penalty matrices satisfy the definiteness condition $Q \succ 0$, in order to ensure feasibility of the problem (see Anderson and Moore (2007)). Dynamic programming can be used to show that the optimal policy that solves (1) is linear state-feedback $u_t = Kx_t$, where the gain matrix $K = \mathcal{K}(P)$ is expressed through the linear-fractional operator $\mathcal{K}$

$$\mathcal{K}(P) \coloneqq -\left(Q_{uu} + B^\intercal P B\right)^{-1}\left(Q_{ux} + B^\intercal P A\right),$$

and P is the optimal value matrix found by solving the algebraic Riccati equation (ARE) expressed with the quadratic-fractional Riccati operator $\mathcal{R}$

$$\mathcal{R}(P) \coloneqq -P + Q_{xx} + A^\intercal P A - \left(Q_{xu} + A^\intercal P B\right)\left(Q_{uu} + B^\intercal P B\right)^{-1}\left(Q_{ux} + B^\intercal P A\right) = 0, \quad (2)$$

The optimal gain and value matrix operators can be expressed more compactly as

$$\mathcal{K}(P) = -\mathcal{H}_{uu}^{-1}(P)\mathcal{H}_{ux}(P), \qquad \mathcal{R}(P) = -P + \mathcal{H}_{xx}(P) - \mathcal{H}_{xu}(P)\mathcal{H}_{uu}^{-1}(P)\mathcal{H}_{ux}(P)$$

where $\mathcal{H}$ is the state-action value matrix operator

$$\mathcal{H}(P) = \begin{bmatrix} \mathcal{H}_{xx}(P) & \mathcal{H}_{xu}(P) \\ \mathcal{H}_{ux}(P) & \mathcal{H}_{uu}(P) \end{bmatrix} \coloneqq Q + \begin{bmatrix} A & B \end{bmatrix}^\intercal P \begin{bmatrix} A & B \end{bmatrix}.$$

The discrete-time Lyapunov equation with matrix F and symmetric matrix S is $X = F^\intercal X F + S$, whose solution we denote by $X = \texttt{DLYAP}(F, S)$, which is unique if F is Schur stable. The first total derivative of the Riccati operator evaluated at point $P \in \mathbb{S}^n$ is denoted as $\mathcal{R}'(P) \in \mathbb{S}^n \times \mathbb{S}^n$. With a slight abuse of notation, the first directional derivative of the Riccati operator evaluated at point P in direction X is denoted as $\mathcal{R}'(P, X) \in \mathbb{S}^n$. Considering two symmetric matrices P, X and the related gains $K = \mathcal{K}(P)$, $L = \mathcal{K}(X)$, then $\mathcal{R}'(P, X)$ can be rewritten in the compact form

$$\mathcal{R}'(P, X) = -X + (A + BK)^\intercal X (A + BK), \quad (3)$$

and we have the identity

$$\mathcal{R}(P) - \mathcal{R}'(X, P) = \begin{bmatrix} I \\ K \end{bmatrix}^\intercal Q \begin{bmatrix} I \\ K \end{bmatrix} + (A + BK)^\intercal P (A + BK) - (A + BL)^\intercal P (A + BL). \quad (4)$$

3. Exact midpoint policy iteration

First we consider finding a solution to the generic vector equation $f(x) = 0$ where $f : \mathbb{R}^n \to \mathbb{R}^n$, whose total derivative at a point x is $f'(x) \in \mathbb{R}^{n \times n}$. The midpoint Newton method, due originally to Traub (1964), begins with an initial guess x_0 then proceeds with iterations

$$x_{k+1}^N = x_k - f'(x_k)^{-1} f(x_k), \qquad x_k^M = \frac{1}{2}(x_k + x_{k+1}^N), \qquad x_{k+1} = x_k - f'(x_k^M)^{-1} f(x_k),$$

until convergence. Each iteration in this technique uses derivative information at two points, x_k and x_k^M. This method has been shown to achieve cubic convergence in a neighborhood of the root by Nedzhibov (2002); Homeier (2004); Babajee and Dauhoo (2006).

We now consider application of the midpoint Newton method to the Riccati equation (2):

$$P_{k+1}^N = P_k - \mathcal{R}'(P_k)^{-1}(\mathcal{R}(P_k)), \;\; P_k^M = \frac{1}{2}\left(P_{k+1}^N + P_k\right), \;\; P_{k+1} = P_k - \mathcal{R}'(P_k^M)^{-1}(\mathcal{R}(P_k)),$$

The updates can be rearranged into the Newton equations

$$\mathcal{R}'(P_k, P_{k+1}^N) = \mathcal{R}'(P_k, P_k) - \mathcal{R}(P_k), \qquad \mathcal{R}'(P_k^M, P_{k+1}) = \mathcal{R}'(P_k^M, P_k) - \mathcal{R}(P_k).$$

Applying (3) to the left- and (4) to the right-hand side, these become the Lyapunov equations

$$P_{k+1}^N = \texttt{DLYAP}(F^N, S^N), \qquad P_{k+1} = \texttt{DLYAP}(F^M, S^M),$$

where F^N, S^N, F^M, S^M are defined in the full midpoint policy iteration in Algorithm 1.

Algorithm 1 Exact midpoint policy iteration (MPI)

Input: System matrices A, B, penalty matrix Q, initial value matrix $P_0 \succ 0$, tolerance ε

Initialize: $P_{-1} = \infty I_n$ and $k = 0$
while $\|P_k - P_{k-1}\| > \varepsilon$ **do**
  Compute $K_k = \mathcal{K}(P_k)$.
  Compute $F^N = A + BK_k$, and $S^N = \begin{bmatrix} I & K_k^\intercal \end{bmatrix} Q \begin{bmatrix} I & K_k^\intercal \end{bmatrix}^\intercal$.
  Solve $P_{k+1}^N = \texttt{DLYAP}(F^N, S^N)$
  Compute $M_k = \frac{1}{2}(P_k + P_{k+1}^N)$ and $L_k = \mathcal{K}(M_k)$.
  Compute $F^M = A + BL_k$, and
  $S^M = \begin{bmatrix} I & K_k^\intercal \end{bmatrix} Q \begin{bmatrix} I & K_k^\intercal \end{bmatrix}^\intercal + (A+BK_k)^\intercal P_k (A+BK_k) - (A+BL_k)^\intercal P_k (A+BL_k)$.
  Solve $P_{k+1} = \texttt{DLYAP}(F^M, S^M)$.
  $k \leftarrow k+1$

Output: P_k, $K_k = \mathcal{K}(P_k)$

Proposition 1 *Consider Exact Midpoint Policy Iteration in Algorithm 1. For any feasible problem instance, there exists a neighborhood around the optimal gain K^* from which any initial gain K_0 yields cubic convergence, i.e. $\|K_{k+1} - K^*\| \leq \mathcal{O}\left(\|K_k - K^*\|^3\right)$ where $\|\cdot\|$ is any matrix norm.*

Proof The claim follows by generic cubic convergence results for midpoint policy iteration (Homeier (2004)) and invertibility and smoothness of $\mathcal{R}$ within a neighborhood of K^*; a complete proof is given in the extended paper (Gravell et al. (2020)). ■

4. Approximate midpoint policy iteration

In the model-free setting we do not have access to the dynamics matrices (A, B), so we cannot execute the updates in Algorithm 1. However, the gain $K = \mathcal{K}(P)$ can be computed solely from the state-action value matrix $H = \mathcal{H}(P)$ as $K = -H_{uu}^{-1} H_{ux}$. Thus, if we can obtain accurate estimates of H, we can use the estimate of H to compute K and we need not perform any other updates that depend explicitly on (A, B). We begin by summarizing an existing method in the literature for estimating state-action value functions from observed state-and-input trajectories.

4.1. State-action value estimation

First, we connect the matrix H with the (relative) state-action value ($\mathcal{Q}$) function, which determines the (relative) cost of starting in state $x = x_0$, taking action $u = u_0$, then following the policy $u_t = Kx_t$ thereafter:

$$\begin{aligned}\mathrm{Tr}(PW) + \mathcal{Q}_K(x,u) &= \begin{bmatrix} x \\ u \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \mathop{\mathbb{E}}_{w}\Big[(Ax+Bu+w)^\mathsf{T} P (Ax+Bu+w)\Big] \\ &= \begin{bmatrix} x^\mathsf{T} & u^\mathsf{T} \end{bmatrix} H \begin{bmatrix} x^\mathsf{T} & u^\mathsf{T} \end{bmatrix}^\mathsf{T} \\ \text{where } H = \mathcal{H}(P), \quad P &= \texttt{DLYAP}\left(A+BK, \begin{bmatrix} I & K^\mathsf{T} \end{bmatrix} Q \begin{bmatrix} I & K^\mathsf{T} \end{bmatrix}^\mathsf{T}\right). \end{aligned} \tag{5}$$

From this expression it is clear that a state-input trajectory, or "rollout," $\mathcal{D} = \{x_t, u_t\}_{t=0}^{\ell}$ must satisfy this cost relationship, which can be used to estimate H. In particular, least-squares temporal difference learning for $\mathcal{Q}$-functions (LSTDQ) was originally introduced by Lagoudakis and Parr (2003) and analyzed by Abbasi-Yadkori et al. (2019); Krauth et al. (2019), and is known to be a consistent and unbiased estimator of H. Following the development of Krauth et al. (2019), the LSTDQ estimator is summarized in Algorithm 2.

Algorithm 2 `LSTDQ`: Least-squares temporal difference learning for Q-functions

Input: Rollout $\mathcal{D} = \{x_t, u_t\}_{t=0}^{\ell}$, gain matrix K^{eval}, penalty matrix Q.

1: Compute augmented rollout $\{z_t, v_t, c_t\}_{t=0}^{\ell}$ where $z_t = \begin{bmatrix} x \\ u \end{bmatrix}$, $v_t = \begin{bmatrix} x \\ K^{\text{eval}} x \end{bmatrix}$, $c_t = z_t^\mathsf{T} Q z_t$.

2: Use feature map $\phi(z) = \mathrm{svec}\left(zz^\mathsf{T}\right)$ and compute the parameter estimate

$$\hat{\Theta} = \Big(\textstyle\sum_{t=1}^{\ell} \phi(z_t)(\phi(z_t) - \phi(v_{t+1})^\mathsf{T}\Big)^\dagger \sum_{t=1}^{\ell} \phi(z_t) c_t.$$

Output: $\hat{H} = \mathrm{smat}(\hat{\Theta})$.

We collect rollouts to feed into Algorithm 2 via Algorithm 3, i.e. by initializing the state with x_0 drawn from the given initial state distribution $\mathcal{X}_0$, then generating control inputs according to $u_t = K^{\text{play}} x_t + u_t^{\text{explore}}$ where K^{play} is a stabilizing gain matrix, and u_t^{explore} is an exploration noise drawn from a distribution $\mathcal{U}_t$, assumed Gaussian in this work, to ensure persistence of excitation.

Algorithm 3 `ROLLOUT`: Rollout collection

Input: Gain K^{play}, rollout length ℓ, initial state distribution $\mathcal{X}_0$, exploration distributions $\{\mathcal{U}_t\}_{t=0}^{\ell}$.

1: Initialize state $x_0 \sim \mathcal{X}_0$
2: **for** $t = 0, 1, 2, \ldots, \ell$ **do**
3: Sample exploratory control input $u_t^{\text{explore}} \sim \mathcal{U}_t$ and disturbance $w_t \sim W$
4: Generate control input $u_t = K^{\text{play}} x_t + u_t^{\text{explore}}$
5: Record state x_t and input u_t
6: Update state according to $x_{t+1} = Ax_t + Bu_t + w_t$

Output: $\mathcal{D} = \{x_t, u_t\}_{t=0}^{\ell}$.

Note that LSTDQ is an off-policy method, and thus the gain K^{play} used to generate the data in Algorithm 3 and the gain K^{eval} whose state-action value matrix is estimated in Algorithm 2 need

not be identical. We will use this fact in the next section to give an off-policy, offline (OFF) and on-policy, online (ON) version of our algorithm. Likewise, the penalty matrix Q used in Algorithm 2 need not be the same as the one in the original problem statement, which is critical to developing the model-free midpoint update in the next section.

4.2. Derivation of approximate midpoint policy iteration

We have shown that estimates of the state-action value matrix H can be obtained by `LSTDQ` using either off-policy or on-policy data. In the following development, (OFF) denotes a variant where a single off-policy rollout $\mathcal{D}$ is collected offline before running the system, and (ON) denotes a variant where new on-policy rollouts are collected at each iteration. Also, an overhat symbol "$\hat{}$" denotes an estimated quantity while the absence of one denotes an exact quantity.

In approximate policy iteration, we can simply form the estimate $\hat{H}_k$ using `LSTDQ` (see Krauth et al. (2019)). For approximate midpoint policy iteration, the form of $\hat{H}_k$ is more complicated and requires multiple steps. To derive approximate midpoint policy iteration, we will re-order some of the steps in the loop of Algorithm 1. Specifically, move the gain calculation in step 3 to the end after step 9. We will also replace explicit computation of the value function matrices with estimation of state-action value matrices, i.e. subsume the pairs of steps 4, 5 and 8,9 into single steps, and work with H instead of P. Thus, at the beginning of each iteration we have in hand an estimated state-action value matrix $\hat{H}_k$ and gain matrix $\hat{K}_k$ satisfying $\hat{K}_k = -\hat{H}_{uu,k}^{-1}\hat{H}_{ux,k}$.

First we translate steps 4, 5, 6, and 7 to a model-free version. Working backwards starting with step 7, in order to estimate L_k, it suffices to estimate $\mathcal{H}(M_k)$ since $L_k = -\mathcal{H}(M_k)_{uu}^{-1}\mathcal{H}(M_k)_{ux}$. To find $\mathcal{H}(M_k)$, notice $\mathcal{H}(X)$ is linear in X, so $\mathcal{H}(M_k) = \left(\mathcal{H}(P_k) + \mathcal{H}(P_{k+1}^N)\right)/2$. Therefore we can estimate $\mathcal{H}(M_k)$ by estimating $\mathcal{H}(P_k)$ and $\mathcal{H}(P_{k+1}^N)$ separately and taking their midpoint. Since the estimate $\hat{H}_k$ of $\mathcal{H}(P_k)$ is known from the prior iteration, what remains is to find an estimate $\hat{H}_{k+1}^N$ of $\mathcal{H}(P_{k+1}^N)$ by collecting $\mathcal{D}^N = \texttt{ROLLOUT}(\hat{K}_k, \ell, \mathcal{X}_0, \{\mathcal{U}_t\}_{t=0}^{\ell})$ (ON) or using $\mathcal{D}^N = \mathcal{D}$ (OFF), and estimating $\hat{H}_{k+1}^N = \texttt{LSTDQ}(\mathcal{D}^N, \hat{K}_k, Q)$. Then we form the estimated gain $\hat{L}_k = -\hat{H}_{uu,k}^{M\ -1}\hat{H}_{ux,k}^M$ where $\hat{H}_k^M = \frac{1}{2}(\hat{H}_k + \hat{H}_{k+1}^N)$.

Now we translate steps 8, 9, and 2 to a model-free version. Working backwards, starting with step 2, in order to estimate K_{k+1}, it suffices to find an estimate $\hat{H}_{k+1}$ of matrix $\mathcal{H}(P_{k+1})$ since $K_{k+1} = -\mathcal{H}(P_{k+1})_{uu}^{-1}\mathcal{H}(P_{k+1})_{ux}$. From steps 8 and 9, we want to estimate

$$H_{k+1} = \mathcal{H}(P_{k+1}) = Q + \begin{bmatrix} A & B \end{bmatrix}^\intercal P_{k+1} \begin{bmatrix} A & B \end{bmatrix}, \tag{6}$$

where

$$P_{k+1} = \texttt{DLYAP}\left(F^M, S^M\right), \tag{7}$$

$$F^M = A + BL_k,$$

$$S^M = \begin{bmatrix} I \\ K_k \end{bmatrix}^\intercal \left(Q + \begin{bmatrix} A & B \end{bmatrix}^\intercal P_k \begin{bmatrix} A & B \end{bmatrix}\right) \begin{bmatrix} I \\ K_k \end{bmatrix} - \begin{bmatrix} I \\ L_k \end{bmatrix}^\intercal \begin{bmatrix} A & B \end{bmatrix}^\intercal P_k \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} I \\ L_k \end{bmatrix}.$$

Comparing the two arguments to $\texttt{DLYAP}(\cdot,\cdot)$ in (5) and (7), we desire both

$$A + BK = A + BL_k, \qquad \begin{bmatrix} I & K^\intercal \end{bmatrix} Q^M \begin{bmatrix} I & K^\intercal \end{bmatrix}^\intercal = S^M. \tag{8}$$

Clearly it suffices to take $K = L_k$ in (8). Notice that, critically, all quantities in S^M on the right-hand side of (8) have been estimated already, i.e. $\hat{K}_k$, $\hat{L}_k$, $\hat{H}_k$ have been calculated already and

$$Q + \begin{bmatrix} A & B \end{bmatrix}^\intercal P_k \begin{bmatrix} A & B \end{bmatrix} = H_k, \qquad \begin{bmatrix} A & B \end{bmatrix}^\intercal P_k \begin{bmatrix} A & B \end{bmatrix} = H_k - Q.$$

Substituting $K = L_k$ in (8) and comparing coefficients, it suffices to estimate Q^M by

$$\hat{Q}^M = \begin{bmatrix} \begin{bmatrix} I & \hat{K}_k^\intercal \end{bmatrix} \hat{H}_k \begin{bmatrix} I & \hat{K}_k^\intercal \end{bmatrix}^\intercal & 0 \\ 0 & 0 \end{bmatrix} - (\hat{H}_k - Q). \tag{9}$$

At this point, establish the rollout $\mathcal{D}^M$ either by collecting $\mathcal{D}^M = \texttt{ROLLOUT}(\hat{L}_k, \ell, \mathcal{X}_0, \{\mathcal{U}_t\}_{t=0}^{\ell})$ (ON) or using $\mathcal{D}^M = \mathcal{D}$ (OFF). Then the matrix $\hat{H}_{k+1}^O = \texttt{LSTDQ}(\mathcal{D}^M, \hat{L}_k, \hat{Q}^M)$ estimates $H_{k+1}^O = Q^M + \begin{bmatrix} A & B \end{bmatrix}^\intercal P_{k+1} \begin{bmatrix} A & B \end{bmatrix}$. However, we need $H_{k+1} = Q + \begin{bmatrix} A & B \end{bmatrix}^\intercal P_{k+1} \begin{bmatrix} A & B \end{bmatrix}$, which is easily found by offsetting H_{k+1}^O as $H_{k+1} = H_{k+1}^O + (Q - Q^M)$, and thus $\hat{H}_{k+1} = \hat{H}_{k+1}^O + (Q - \hat{Q}^M)$ estimates H_{k+1}. One further consideration to address is the initial estimate $\hat{H}_0$; since we do not have a prior iterate to use, we simply collect $\mathcal{D} = \texttt{ROLLOUT}(\hat{K}_0, \ell, \mathcal{X}_0, \{\mathcal{U}_t\}_{t=0}^{\ell})$ and estimate $\hat{H}_0 = \texttt{LSTDQ}(\mathcal{D}, \hat{K}_0, Q)$ i.e. the first iteration will be a standard approximate policy iteration/Newton step. Importantly, the initial gain $\hat{K}_0$ must stabilize the system so that the value functions are finite-valued. Also, although a convergence criterion such as $\|\hat{H}_k - \hat{H}_{k-1}\| > \varepsilon$ could be used, it is more straightforward to use a fixed number of iterations N so that the influence of stochastic errors in $\hat{H}_k$ does not lead to premature termination of the program. Likewise, a schedule of increasing rollout lengths ℓ could be used for the (ON) variant to achieve increasing accuracy, but finding a meaningful schedule which properly matches the fast convergence rate of the algorithm requires more extensive analysis. The full set of updates are compiled in Algorithm 4.

Algorithm 4 Approximate midpoint policy iteration (AMPI)

Input: Penalty Q, gain $\hat{K}_0$, number of iterations N, rollout length ℓ, distributions $\mathcal{X}_0$, $\{\mathcal{U}_t\}_{t=0}^{\ell}$.

Initialize: $\hat{H}_{-1} = \infty I_{n+m}$ and $k = 0$
Collect $\mathcal{D} = \texttt{ROLLOUT}(\hat{K}_0, \ell, \mathcal{X}_0, \{\mathcal{U}_t\}_{t=0}^{\ell})$
Estimate value matrix $\hat{H}_0 = \texttt{LSTDQ}(\mathcal{D}, K_0, Q)$.
while $k < N$ **do**
  Set $\mathcal{D}^N = \mathcal{D}$ (OFF), or collect $\mathcal{D}^N = \texttt{ROLLOUT}(\hat{K}_k, \ell, \mathcal{X}_0, \{\mathcal{U}_t\}_{t=0}^{\ell})$ (ON)
  Estimate value matrix $\hat{H}_{k+1}^N = \texttt{LSTDQ}(\mathcal{D}^N, \hat{K}_k, Q)$.
  Form the midpoint value estimate $\hat{H}_k^M = \frac{1}{2}(\hat{H}_k + \hat{H}_{k+1}^N)$.
  Compute the midpoint gain $\hat{L}_k = -{\hat{H}_{uu,k}^M}^{-1} \hat{H}_{ux,k}^M$.
  Set $\mathcal{D}^M = \mathcal{D}$ (OFF), or collect $\mathcal{D}^M = \texttt{ROLLOUT}(\hat{L}_k, \ell, \mathcal{X}_0, \{\mathcal{U}_t\}_{t=0}^{\ell})$ (ON)
  Estimate $\hat{H}_{k+1}^O = \texttt{LSTDQ}(\mathcal{D}^M, \hat{L}_k, \hat{Q}^M)$ where $\hat{Q}^M$ computed from (9)
  Compute the estimated value matrix $\hat{H}_{k+1} = \hat{H}_{k+1}^O + (Q - \hat{Q}^M)$.
  Compute the gain $\hat{K}_{k+1} = -\hat{H}_{uu,k+1}^{-1} \hat{H}_{ux,k+1}$.
  $k \leftarrow k + 1$

Output: $\hat{H}_k$, $\hat{K}_k$

Proposition 2 *Consider Approximate Midpoint Policy Iteration in Algorithm 4. As the rollout length ℓ grows to infinity, the state-action value matrix estimate $\hat{H}_k$ converges to the exact value. Thus, in the infinite data limit, for any feasible problem instance, there exists a neighborhood around the optimal gain K^* from which any initial gain $\hat{K}_0$ converges cubically to K^*.*

Proof The claim follows by Proposition 1 and the fact that LSTDQ is a consistent estimator Lagoudakis and Parr (2003); Krauth et al. (2019), i.e. as $\ell \to \infty$ the estimates $\hat{H}$ used in Algorithm 4 approach the true values H indirectly used in Algorithm 1. ■

5. Numerical experiments

In this section we compare the empirical performance of proposed midpoint policy iteration (MPI) with standard policy iteration (PI), as well as their approximate versions (AMPI) and (API). In all experiments, regardless of whether the exact or approximate algorithm is used, we evaluated the value matrix P_k associated to the policy gains K_k at each iteration k on the true system, i.e. the solution to $P_k = \texttt{DLYAP}\left(A + BK_k, \begin{bmatrix} I & K_k^\intercal \end{bmatrix} Q \begin{bmatrix} I & K_k^\intercal \end{bmatrix}^\intercal\right)$. We then normalized the deviation $\|P_k - P^*\|$, where P^* solves the Riccati equation (2), by the quantity $\|P^*\|$. This gives a meaningful metric to compare different suboptimal gains. We also elected to use the off-policy version (OFF) of AMPI and API in order to achieve a more direct and fair comparison between the midpoint and standard methods; each is given access to precisely the same sample data and initial policy, so differences in convergence are entirely due to the algorithms. Nevertheless, similar results were observed in the on-policy online setting (ON) and for higher dimensional state- and input-spaces; please see the extended paper for these results Gravell et al. (2020). Python code which implements the proposed algorithms and reproduces the experimental results is available at https://github.com/TSummersLab/midpoint-policy-iteration.

5.1. Representative example

Here we consider one of the simplest tasks in the control discipline: regulating an inertial mass using a force input. Forward-Euler discretization of the continuous-time dynamics with sampling time Δt yields the discrete-time dynamics

$$x_{t+1} = Ax_t + Bu_t + w_t \quad \text{where} \quad A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{\Delta t}{\mu} \end{bmatrix},$$

with mass $\mu > 0$, state $x_t \in \mathbb{R}^2$ where the first state is the position and the second state is the velocity, force input $u_t \in \mathbb{R}$, and $w_t \sim \mathcal{N}(0, W)$ with $W = \Delta t \cdot W_c$ where $W_c \succeq 0$. We used $\mu = 1$, $\Delta t = 0.01$, $W_c = 0.01 I_2$, $Q = I_3$. The initial gain was chosen by perturbing the optimal gain K^* in a random direction such that the initial relative error $\|P_k - P^*\|/\|P^*\| = 10$; in particular the initial gain was $K_0 = \begin{bmatrix} -0.044 & -2.084 \end{bmatrix}$. For the approximate algorithms, we used the hyperparameters $\ell = 300$, $\mathcal{X}_0 = \mathcal{N}(0, I_2), \mathcal{U}_t = \mathcal{N}(0, I_2)$.

The results of applying midpoint policy iteration and the standard policy iteration, exact and approximate (OFF), are plotted in Figure 1. Clearly MPI and AMPI converge more quickly to the (approximate) optimal policy than PI and API, with MPI converging to machine precision in 7 iterations vs 9 iterations for PI, and AMPI converging to noise precision in 6 iterations vs 8 iterations for API.

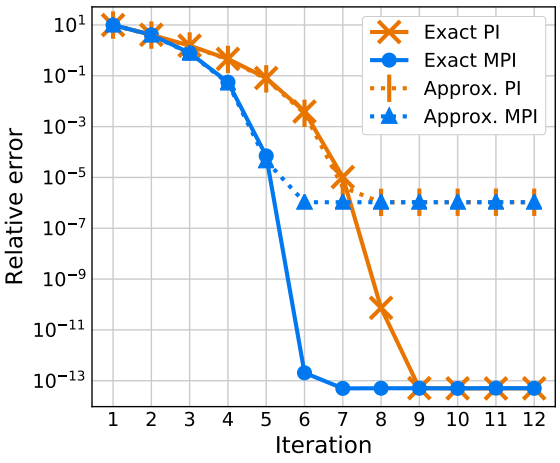


Figure 1: Relative error $\|P_k - P^*\|/\|P^*\|$ vs iteration count k using PI and MPI on the inertial mass control problem.

5.2. Randomized examples

Next we apply the approximate PI (OFF) algorithms on 1000 problem instances in a Monte Carlo-style approach, where problem data was generated randomly with $n = 4$, $m = 2$, entries of A drawn from $\mathcal{N}(0, 1)$ and A scaled so $\rho(A) \sim \text{Unif}([0, 2])$, entries of B drawn from $\text{Unif}([0, 1])$, and $Q = U\Lambda U^\intercal \succ 0$ with Λ diagonal with entries drawn from $\text{Unif}([0, 1])$ and U orthogonal by taking the QR-factorization of a square matrix with entries drawn from $\mathcal{N}(0, 1)$. We used a small process noise covariance of $W = 10^{-6} I_4$ to avoid unstable iterates due to excessive data-based approximation error of H, over all problem instances. All initial gains K_0 were chosen by perturbing the optimal gain K^* in a random direction such that the initial relative error $\|P_k - P^*\|/\|P^*\| = 10$. For the approximate algorithms, we used the hyperparameters $\ell = 100$, $\mathcal{X}_0 = \mathcal{N}(0, I_2)$, $\mathcal{U}_t = \mathcal{N}(0, I_2)$.

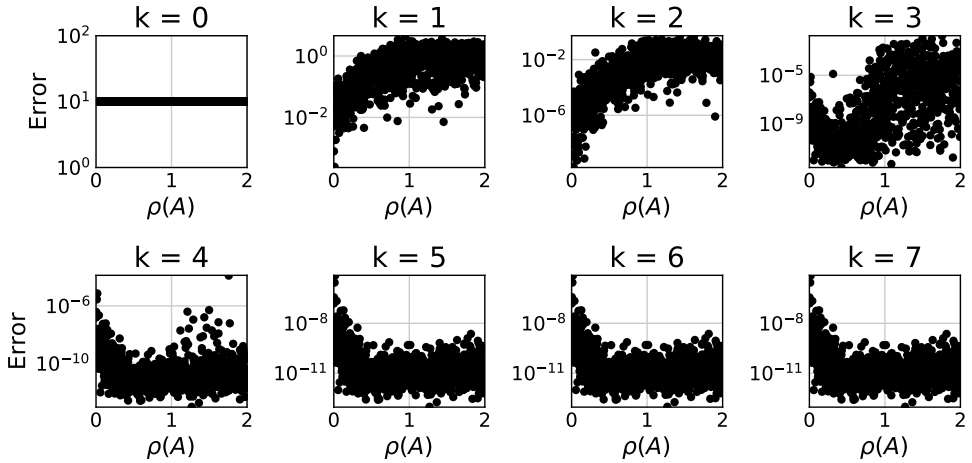
In Figure 2 we plot the relative value error $\|P_k - P^*\|/\|P^*\|$ over iterations. Each scatter point represents a unique Monte Carlo sample, i.e. a unique problem instance, initial gain, and rollout. Each scatter plot shows the empirical distribution of errors at the iteration count k labeled in the subplot titles above each plot. The x-axis is the spectral radius of A which characterizes open-loop stability. In Figure 2 (b), scatter points lying below 1.0 on the y-axis indicate that the midpoint method achieves lower error than the standard method on the same problem instance. From Figure 2 (a), it is clear that AMPI achieves extremely fast convergence to a good approximation of the optimal gain, with the relative error being less than 10^{-6} on almost all problem instances after just 4 iterations. From Figure 2 (b), we see that AMPI achieves significantly lower error than API on iteration counts $2, 3, 4, 5$ for almost all problem instances; recall that Algorithm 4 takes a standard PI step on the first iteration, explaining the identical performance on $k = 1$.

6. Conclusions and future work

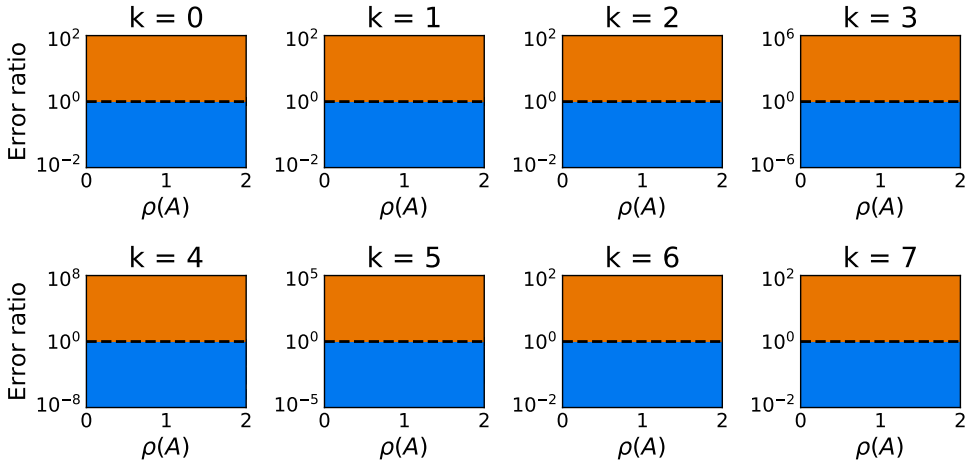
Empirically, we found that regardless of the stabilizing initial policy chosen, convergence to the optimum always occurred when using the exact midpoint method. Likewise, we also found that approximate midpoint and standard PI converge to the same approximately optimal policy, and hence value matrix P, after enough iterations when evaluated on the same fixed off-policy rollout data $\mathcal{D}$. We conjecture that such robust, finite-data convergence properties can be proven rigorously, which we leave to future work.

This algorithm is perhaps most useful in the regime of practical problems in the online setting where it is relatively expensive to collect data and relatively cheap to perform the computations required to execute the updates. In such scenarios, the goal is to converge in as few iterations as possible, and MPI shows a clear advantage. Both the exact and approximate midpoint PI incur a computation cost *double* that of their standard PI counterparts. Theoretically, the faster *cubic* convergence rate of MPI over the *quadratic* convergence rate of PI should dominate this order constant ($2\times$) cost with sufficiently many iterations. However, unfortunately, due to finite machine precision, the total number of useful iterations that increase the precision of the optimal policy is limited, and the per-iteration cost largely counteracts the faster over-iteration convergence of MPI. This phenomenon becomes even more apparent in the model-free case where the "noise floor" is even higher. However, this disadvantage may be reduced by employing iterative Lyapunov equation solvers in Algorithm 1 or iterative (recursive) least-squares solvers in Algorithm 4 and warm-starting the midpoint equation with the Newton solution. Furthermore, the benefit of the faster convergence of the midpoint PI may become more important in extensions to nonlinear systems, where the order constants in Propositions 1 and 2 are smaller.

The current methodology is certainty-equivalent in the sense that we treat the estimated value functions as correct. Future work will explore ways to estimate and account for uncertainty in the value function estimate explicitly to minimize regret risk in the initial transient stage of learning when the amount of information is low and uncertainty is high.



(*a*) Relative error using AMPI



(*b*) Ratio of relative errors using AMPI/API

Figure 2: (a) Relative value error $\|P_k - P^*\|/\|P^*\|$ using AMPI and (b) ratio of relative error using AMPI divided by that using API.

Acknowledgments

This material is based on work supported by the United States Air Force Office of Scientific Research under award number FA2386-19-1-4073 and by the National Science Foundation under award number ECCS-2047040.

References

Yasin Abbasi-Yadkori, Nevena Lazic, and Csaba Szepesvári. Model-free linear quadratic control via reduction to expert prediction. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3108–3117, 2019.

Asma Al-Tamimi, Frank L Lewis, and Murad Abu-Khalaf. Model-free Q-learning designs for linear discrete-time zero-sum games with application to H-infinity control. *Automatica*, 43(3):473–481, 2007.

Brian DO Anderson. Second-order convergent algorithms for the steady-state Riccati equation. *International Journal of Control*, 28(2):295–306, 1978.

Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

D.K.R. Babajee and M.Z. Dauhoo. An analysis of the properties of the variants of Newton's method with third order convergence. *Applied Mathematics and Computation*, 183(1):659 – 684, 2006. ISSN 0096-3003. doi: https://doi.org/10.1016/j.amc.2006.05.116. URL http://www.sciencedirect.com/science/article/pii/S0096300306006011.

Steven J Bradtke, B Erik Ydstie, and Andrew G Barto. Adaptive linear quadratic control using policy iteration. In *Proceedings of 1994 American Control Conference-ACC'94*, volume 3, pages 3475–3479. IEEE, 1994.

J. Bu, A. Mesbahi, and M. Mesbahi. LQR via first order flows. In *2020 American Control Conference (ACC)*, pages 4683–4688, 2020. doi: 10.23919/ACC45564.2020.9147853.

Jingjing Bu, Lillian J Ratliff, and Mehran Mesbahi. Global convergence of policy gradient for sequential zero-sum linear quadratic dynamic games. *arXiv preprint arXiv:1911.04672*, 2019.

Peter Coppens and Panagiotis Patrinos. Sample complexity of data-driven stochastic LQR with multiplicative uncertainty. *arXiv preprint arXiv:2005.12167*, 2020.

Peter Coppens, Mathijs Schuurmans, and Panagiotis Patrinos. Data-driven distributionally robust LQR with multiplicative noise. In *Learning for Dynamics and Control*, pages 521–530. PMLR, 2020.

Annie AM Cuyt and Louis B Rall. Computational implementation of the multivariate halley method for solving nonlinear systems of equations. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):20–36, 1985.

Tobias Damm and Diederich Hinrichsen. Newton's method for a rational matrix equation occurring in stochastic control. *Linear Algebra and its Applications*, 332:81–109, 2001.

Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. Regret bounds for robust adaptive control of the linear quadratic regulator. In *Advances in Neural Information Processing Systems*, pages 4188–4197, 2018.

Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. On the sample complexity of the linear quadratic regulator. *Foundations of Computational Mathematics*, Aug 2019. ISSN 1615-3383.

Maryam Fazel, Rong Ge, Sham Kakade, and Mehran Mesbahi. Global convergence of policy gradient methods for the linear quadratic regulator. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1467–1476. PMLR, 10–15 Jul 2018.

G Freiling and A Hochhaus. On a class of rational matrix differential equations arising in stochastic control. *Linear Algebra and its Applications*, 379:43 – 68, 2004. ISSN 0024-3795. doi: https://doi.org/10.1016/S0024-3795(02)00651-1. URL http://www.sciencedirect.com/science/article/pii/S0024379502006511. Special Issue on the Tenth ILAS Conference (Auburn, 2002).

Benjamin Gravell and Tyler Summers. Robust learning-based control via bootstrapped multiplicative noise. In Alexandre M. Bayen, Ali Jadbabaie, George Pappas, Pablo A. Parrilo, Benjamin Recht, Claire Tomlin, and Melanie Zeilinger, editors, *Proceedings of Machine Learning Research*, volume 120, pages 599–607. PMLR, 2020.

Benjamin Gravell, Peyman Mohajerin Esfahani, and Tyler Summers. Learning robust control for LQR systems with multiplicative noise via policy gradient. *CoRR*, 2019. URL http://arxiv.org/abs/1905.13547.

Benjamin Gravell, Iman Shames, and Tyler Summers. Approximate midpoint policy iteration for linear quadratic control. *arXiv preprint arXiv:2011.14212*, 2020.

Benjamin Gravell, Karthik Ganapathy, and Tyler Summers. Policy iteration for linear quadratic games with stochastic parameters. *IEEE Control Systems Letters*, 5(1):307–312, 2021. doi: 10.1109/LCSYS.2020.3001883.

Chun-Hua Guo and Alan J Laub. On a Newton-like method for solving algebraic Riccati equations. *SIAM Journal on Matrix Analysis and Applications*, 21(2):694–698, 2000.

Miguel Angel Hernández-Verón and N Romero. Solving symmetric algebraic Riccati equations with high order iterative schemes. *Mediterranean Journal of Mathematics*, 15(2):51, 2018.

G Hewer. An iterative technique for the computation of the steady state gains for the discrete optimal regulator. *IEEE Transactions on Automatic Control*, 16(4):382–384, 1971.

H.H.H Homeier. A modified newton method with cubic convergence: the multivariate case. *Journal of Computational and Applied Mathematics*, 169(1):161 – 169, 2004. ISSN 0377-0427. doi: https://doi.org/10.1016/j.cam.2003.12.041. URL http://www.sciencedirect.com/science/article/pii/S0377042703010215.

J. P. Jansch-Porto, B. Hu, and G. E. Dullerud. Convergence guarantees of policy optimization methods for markovian jump linear systems. In *2020 American Control Conference (ACC)*, pages 2882–2887, 2020. doi: 10.23919/ACC45564.2020.9147571.

David Kleinman. On an iterative technique for Riccati equation computations. *IEEE Transactions on Automatic Control*, 13(1):114–115, 1968.

Karl Krauth, Stephen Tu, and Benjamin Recht. Finite-time analysis of approximate policy iteration for the linear quadratic regulator. In *Advances in Neural Information Processing Systems*, pages 8512–8522, 2019.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4: 1107–1149, December 2003. ISSN 1532-4435.

B. Luo, Y. Yang, and D. Liu. Policy iteration Q-learning for data-based two-player zero-sum game of linear discrete-time systems. *IEEE Transactions on Cybernetics*, pages 1–11, 2020.

Omid Madani. On policy iteration as a Newton's method and polynomial policy iteration algorithms. In *AAAI/IAAI*, pages 273–278, 2002.

Horia Mania, Stephen Tu, and Benjamin Recht. Certainty equivalent control of LQR is efficient. *ArXiv*, abs/1902.07826, 2019.

Gyurhan Nedzhibov. On a few iterative methods for solving nonlinear equations. *Application of mathematics in engineering and economics*, 28:1–8, 2002.

Samet Oymak and Necmiye Ozay. Non-asymptotic identification of LTI systems from a single trajectory. In *2019 American Control Conference (ACC)*, pages 5655–5661. IEEE, 2019.

Martin L. Puterman and Shelby L. Brumelle. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 4(1):60–69, 1979. ISSN 0364765X, 15265471. URL http://www.jstor.org/stable/3689239.

Joe Fred Traub. *Iterative methods for the solution of equations*. Prentice-Hall, 1964.

Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. Policy optimization provably converges to Nash equilibria in zero-sum linear quadratic games. In *Advances in Neural Information Processing Systems*, pages 11598–11610, 2019.