J Control Theory Appl 2011 9 (3) 336–352
DOI 10.1007/s11768-011-0313-y

A review of stochastic algorithms with continuous value function approximation and some new approximate policy iteration algorithms for multidimensional continuous applications

Warren B. POWELL, Jun MA
Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544, U.S.A.

Abstract: We review the literature on approximate dynamic programming, with the goal of better understanding the theory behind practical algorithms for solving dynamic programs with continuous and vector-valued states and actions and complex information processes. We build on the literature that has addressed the well-known problem of multidimensional (and possibly continuous) states, and the extensive literature on model-free dynamic programming, which also assumes that the expectation in Bellman's equation cannot be computed. However, we point out complications that arise when the actions/controls are vector-valued and possibly continuous. We then describe some recent research by the authors on approximate policy iteration algorithms that offer convergence guarantees (with technical assumptions) for both parametric and nonparametric architectures for the value function.

Keywords: Approximate dynamic programming; Reinforcement learning; Optimal control; Approximation algorithms

1 Introduction

Dynamic programming has a rich history, with roots that span disciplines such as engineering and economics, computer science, and operations research, centered around the solution to a set of optimality equations that go under names like Bellman's equations, Hamilton-Jacobi equations, or the general purpose Hamilton-Jacobi-Bellman (HJB) equations. Perhaps the two most dominant lines of investigation assume either a) discrete states, discrete actions, and discrete time (most commonly found in operations research and computer science) or b) continuous states and actions, often in continuous time (most commonly found in engineering and economics). Discrete models are typically described using a discrete state S and action a, while continuous models are typically described using state x and control u.

If we use the language of discrete states and actions, Bellman's equation (as it is most commonly referred to in this community) would be written as

$$V(S) = \max_{a\in\mathcal{A}} \big(C(S,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)V(s')\big), \quad (1)$$

where $C(S,a)$ is the expected reward if we are in state S and take action a, and $p(s'|s,a)$ is the one-step transition matrix. Equation (1) can be written equivalently in its expectation form as

$$V(S) = \max_{a\in\mathcal{A}} \big(C(S,a) + \gamma \mathrm{E}V(S')\big). \quad (2)$$

A rich literature has grown around the solution to (1), starting with the work of Bellman [1], progressing through a series of notable contributions, particularly [2], and [3], which serve as a capstone summary of an extensive history of contributions to this field. In this problem class, solution algorithms depend on our ability to compute $V(s)$ for each discrete state $s \in \mathcal{S}$.

For many practical problems, the state variable is a vector. For discrete states, the size of the state space $\mathcal{S}$ grows exponentially with the number of dimensions, producing what is widely referred to as the 'curse of dimensionality' in dynamic programming. In fact, there are potentially three curses of dimensionality: the state space, the outcome space (hidden in the expectation in (2)), and the action space (the action a may also be a vector, complicating the search for the best action). The curse (or curses) of dimensionality is a direct byproduct of a desire to work with discrete representations, which are particularly easy to deal with on the computer.

In the engineering literature, the Hamilton-Jacobi equation is more typically written as

$$J(x) = \max_{u\in\mathcal{U}} \Big(g(x,u) + \gamma \int_{x'} P(x'|x,u)J(x')\Big), \quad (3)$$

where $P(x'|x,u)$ is called the transition kernel, which gives the density of state x' given we are in state x and apply control u. The control theory community often starts with the transition function

$$x' = S^{\mathrm{M}}(x,u,w), \quad (4)$$

where w is a 'noise term' and $S^{\mathrm{M}}(\,\cdot\,)$ is the system model or transition function. For example, there are many problems in engineering where the transition function is linear and can be written as

$$x' = Ax + Bu + w$$

Received 13 December 2010.

with a quadratic reward function that might be written as

$$g(x,u) = u^{\mathrm{T}}Qu.$$

Such problems are referred to as linear, quadratic control, and lend themselves to analytic solutions or simple algebraic models (see [4,5]).

Our presentation uses a merger of the two notational systems. We use x for the state and u for the decision, but $V(x)$ for the value of being in state x, a decision we made because this paper is appearing in a control theory journal. We use $C(x,u)$ as the reward (cost if we are minimizing) when we are in state x and apply decision u.

There are, of course, many problems where states and actions/controls are continuous, but where we lose the nice property of additive noise in the transition function or quadratic rewards. For example, if we are allocating resources such as energy, water, or money, the control u is a vector of flows of resources between supplies and demands (possibly through a network), subject to constraints on the availability of resources (such as water in the reservoir) and demands (such as the need for electricity). In this setting, randomness can appear in the constraint set and in the parameters in the objective function.

At the same time, we do not enjoy the discrete structure assumed in (1), which otherwise does not make any assumption about problem structure but which is severely limited in its ability to handle vector-valued states, actions, or random information. For this reason, these two communities are seeing an unusual convergence toward approximation strategies that fall under names such as reinforcement learning (used in computer science), or approximate dynamic programming, used in operations research and increasingly in engineering (see, for example, Chapter 6 in [6]). Other names are adaptive dynamic programming and neurodynamic programming. Often, cosmetic differences in names and notation hide more substantive differences in the characteristics of the problems being solved. In computer science, the vast majority of applications assume a relatively small number of discrete (or discretized) actions. In engineering, a control vector is generally continuous with a 'small' number of dimensions (e.g., less than 20). In operations research, decisions may be discrete or continuous but often have hundreds or thousands of dimensions (see http://www.castlelab.princeton.edu/wagner.htm for an illustration).

These efforts are focused on two complementary paths: approximating the value function or approximating the policy. If we are approximating the value function, we might write the policy $\pi(x)$ as

$$\pi(x) = \arg\max_{u\in\mathcal{U}} \big(C(x,u) + \gamma \mathrm{E}\bar{V}(f(x,u,w))\big), \quad (5)$$

where $\bar{V}(x')$ is some sort of statistical approximation of the value of being in state x'. For example, we might write the approximation in the form

$$\bar{V}(x) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(x),$$

where $\phi_f(x), f \in \mathcal{F}$ is a set of user-specified basis functions, and θ is a vector of regression parameters to be determined. There is a rich and growing literature where the value function is approximated using neural networks [7,8] or a host of other statistical methods [9].

Alternatively, we might specify some functional form for $\pi(x|\theta)$ governed by a vector θ of tunable parameters. This problem is often written as

$$\max_{\theta} \mathrm{E} \sum_{t=0}^{\infty} C(x_t, \pi(x_t)). \quad (6)$$

In theory, the policy can be the same as (5), although more often it is given a specific functional form that captures the behavior of the problem. However, it is possible to approximate the policy using the same family of statistical techniques that are being used to approximate value functions (see [10–12]).

This paper represents a modern survey of approximate dynamic programming and reinforcement learning, where we make an effort (albeit an imperfect one) of covering contributions from different communities. Our primary interest is in developing an understanding of the convergence theory of algorithms that are specifically designed for problems with multidimensional states and actions, and where the expectation cannot be computed exactly. The bulk of our survey is provided in Section 2. Then, we provide a summary of some recent algorithmic work aimed at this problem class. Section 3 provides some mathematical foundations for continuous Markov decision processes (MDPs). Section 4 then presents some recent work on algorithms specifically tailored to the challenges of multidimensional and continuous states, actions, and information.

2 Literature review

In this section, we survey stochastic algorithms with continuous function approximations from different communities including dynamic programming, reinforcement learning, and control. Each community has its unique perspective to solve stochastic optimization problems, generally motivated by specific problem classes. Some are designed to apply continuous function approximations to discrete problems, while others work for problems with continuous states and actions.

There are a number of dimensions to the general problem of finding optimal or near-optimal policies for sequential decision problems. Some of these include:

- The choice of policies for choosing states, which can be divided between:

  1) on-policy algorithms where the choice of the next state is determined by the estimation policy, which is being optimized,

  2) off-policy algorithms that evaluate the estimation policy by using a sampling policy (or behavior policy) to determine the next state to visit.

This issue is often referred to as the exploration vs. exploitation problem in approximate dynamic programming, and it remains an active area of research.

- Computing the expectation. Some algorithms assume known distribution (transition matrix) or computable expectation, while others use online Monte Carlo simulation to estimate the expectation.

- Problem structure. Some algorithms assume special problem structures such as linear control and transition, quadratic rewards, and linear additive noise (sometimes Gaussian).

. Approximation architecture. We can approximate the value function using lookup tables (discrete representation), parametric beliefs (e.g., basis functions), and various forms of nonparametric representations (neural networks, kernel regression, support vector machines, and hierarchical aggregation).

. Performance guarantees. Some algorithms have convergence guarantees that are almost sure/with probability 1, in probability, in expectation, or provide performance bounds, while others provide good empirical performance without rigorous convergence support.

Given space constraints and the current level of maturity in the field, our review is necessarily incomplete. For a good introduction to the field from the perspective of computer science, we recommend reference [13]. Reference [7] and Chapter 6 in [4] provide a rigorous theoretical foundation. Reference [14] provides an introduction to ADP for an engineering audience with an emphasis on modeling and algorithms, with a presentation oriented toward complex problems. Reference [15] is a recent research monograph with numerous algorithms. References [16–18] are excellent research monographs that are more oriented toward the optimization of simulation models, which is a closely related problem class.

Below we provide a summary of some of the research in this growing field, with the goal of touching on the major issues and algorithmic strategies that arise in the design of effective algorithms.

2.1 Early heuristic approximations for discrete problems

We start with algorithms that are designed to solve discrete problems with continuous value function approximation (VFA). The use of compact representations in VFA can be traced to the origins of the field. Reference [19] first uses polynomial representations as a method for breaking the curse of dimensionality in the state space. Both [20] and [21] consider other compact representation methods such as replacement of state and action spaces with subsets. Reference [22] proposes using linear combinations of fixed sets of basis functions to approximate value functions. Both temporal difference (TD) learning in [23] and Q-learning in [24] consider various compact representations such as linear function approximation and artificial neural networks for dynamic programming. However, all these approaches are proposed as heuristics without rigorous convergence analysis, even though there were extraordinarily successful applications like the world-class backgammon player given in [25] and robot navigation given in [26].

2.2 Feature-based function approximation

The first step to set up a rigorous framework combining dynamic programming and compact representations of value functions is given in [27]. Two types of feature-based value iteration algorithms are proposed. One is a variant of the value iteration algorithm that uses a lookup table at an aggregated level (a form of feature) rather than in the original state space. The other value iteration algorithm employs feature extraction and linear approximations with a fixed set of basis functions. Under rather strict technical assumptions on the feature mapping, reference [27] proves the convergence of the value iteration algorithm (not necessarily to the optimal value function unless it is spanned by the basis functions) and provides a bound on the quality of the resulting approximations compared with the optimal value function.

2.3 TD learning algorithms

Reference [27] develops a counter-example to illustrate that a simple combination of value iteration and linear approximation fitted using least squares might lead to divergence of the algorithm. As pointed out by Sutton, the counter-example fails to hold when an online state sampling scheme is employed. As a result, reference [28] considers an online TD learning TD(λ) algorithm using a linear-in-the-parameters model and continuous basis functions. The algorithm assumes a fixed policy, in which case the problem is reduced to a Markov chain. The convergence analysis is established on the assumption of a discrete state Markov chain, even though it is claimed that the proofs can be easily carried over to the continuous case.

2.3.1 Least-squares TD learning

Reference [29] combines a TD learning algorithm with linear function approximation and least-squares updating to build the least-squares TD (LSTD) algorithm for a fixed policy and proves almost sure convergence of the algorithm. It is argued that LSTD is superior to the conventional TD algorithm in terms of convergence properties for the following three reasons: 1) Tuning of step size parameters is not needed in LSTD, overcoming the well-known problem of slow convergence with a poor choice of step sizes; 2) LSTD produces faster convergence because samples are used more efficiently; and 3) LSTD is robust to the initial value of the parameter estimates and choice of basis functions, but TD using a stochastic gradient updating algorithm is not. Reference [30] generalizes the LSTD(λ) algorithm to arbitrary values of $\lambda \in [0, 1]$. Then, LSTD in [29] becomes a special case for $\lambda = 0$. At the other extreme of $\lambda = 1$, the algorithm is an incremental construction of supervised linear regression.

2.3.2 Least squares policy iteration

Motivated by the LSTD algorithm, reference [31] proposes the least squares policy iteration (LSPI) algorithm that combines VFA with linear architectures and approximate policy iteration. LSPI is presented as a model-free, off-policy, offline approximate policy iteration algorithm for finite MDPs that uses LSTD-Q (a modified version of LSTD) to evaluate state-action Q-factors of a fixed policy, and a generic deterministic error bound of the approximate policy iteration as in [7] is provided as convergence support for the algorithm.

2.3.3 Representative policy iteration

Reference [32] extends the LSPI algorithm within a novel spectral framework called representative policy iteration (RPI). By representing a finite sample of state transitions induced by the MDP as an undirected graph, the algorithm automatically generates subspaces on which to project the orthonormal basis functions using spectral analysis of graph Laplacian operators. The algorithm provides a potential solution to one of the major open problems in approximate dynamic programming, which is feature (basis function) selection.

2.3.4 Off-policy TD learning

Reference [33] introduces an off-policy TD learning algorithm that is stable with linear function approximation using importance sampling. The algorithm converges almost surely given training under any ϵ-soft policy (Boltzmannn exploration). Reference [34] presents an off-policy TD algorithm, called gradient temporal-difference (GTD). The learning algorithm uses i.i.d. sampling of initial states combined with on-policy transitions to perform stochastic gradient descent and to update the VFA estimates. The algorithm converges almost surely to the same solution as conventional TD and LSTD with linear complexity.

2.3.5 Fitted TD learning

Reference [35] presents a convergent fitted TD learning (value iteration) algorithm with function approximations that are contraction mappings, such as k-nearest neighbor, linear interpolation, some types of splines, and local weighted average. Interestingly, linear regression and neural networks do not fall into this class, and they can in fact diverge. The main reason for divergence is the exaggeration feature that small local changes can lead to large global shifts of the approximation. Reference [36] proves a weaker convergece result (converge to a bounded region almost surely) for linear approximation algorithms such as $\mathrm{SARSA}(0)$ (a Q-learning-type algorithm presented in [37]) and $V(0)$ (a value-iteration-type algorithm introduced by [38]).

2.4 Residual gradient algorithm

To overcome the instability of Q-learning or value iteration when implemented directly with a general function approximation, residual gradient algorithms, which perform gradient descent on the mean-squared Bellman residual rather than the value function or Q-function, are proposed in [39]. For a deterministic MDP, the change of weight w of the linear approximation is

$$\begin{aligned} w^{n+1} &= w^n + \Delta w \\ &= w^n + \alpha_n(C(x^n, \pi(x^n)) + \gamma V(x'|w^n) \\ &\quad -V(x^n|w^n))\big(\frac{\partial}{\partial w}\gamma V(x'|w^n) - \frac{\partial}{\partial w}V(x^n|w^n)\big), \end{aligned}$$

where α_n is the step size, $C(x^n, \pi(x^n))$ is the reward, $V(x|w)$ is the value function for state x given weight vector w, and $x' = \pi(x)$ is the next state if we are following policy π. Convergence of the algorithms to a local minimum of the Bellman residual is guaranteed only for the deterministic case.

2.5 Bridge algorithm

Reference [40] describes the Bridge algorithm for TD learning applied to a fixed policy. The VFA is derived from linear combinations of a specified set of basis functions. Graphically, the algorithm uses a novel projection operator called B instead of the standard operator to 'build' a bridge across the hypothesized class of functions towards the optimal solution for VFA updates. The algorithm can be applied to any 'agnostically learnable' hypothesis class other than the class of linear combination of fixed basis functions, and it is provably convergent to an approximate global optimum given approximation error bounds.

2.6 State-action-reward-state-action (SARSA) updating

SARSA is an algorithm whose name reflects the notation of reinforcement learning that is designed for discrete states and actions. Assume we are in state s, choose action a according to some policy (such as ϵ-greedy), observe a reward r and then the next state s', after which we again choose an action a' based on the same policy. This is an on-policy (the value of being in state s is based on the same policy for choosing action a as we use to determine the action a' out of the next state s'), convergent algorithm as long as the policy guarantees that each state is visited infinitely often. Reference [41] presents a model-free type of approximate policy iteration that combines online SARSA updates (see [42]) with linear state-action Q-factor approximation for policy evaluation and uses a new policy improvement operator for policy improvement. Under certain technical assumptions on the policy improvement operator, the algorithm is provably convergent to a unique solution from any initial policy.

2.7 Linear quadratic regulation

In the 1990s, convergence proofs of dynamic programming algorithms were established for a special problem class with continuous states called linear quadratic regulation (LQR), which has been studied extensively in control theory. LQR is a control problem with linear dynamic transition and quadratic cost/reward function for continuous state and action spaces. What follows is a canonical form of LQR. The state system evolves as $x_{t+1} = A_t x_t + B_t u_t + w_{t+1}$, where w_t is random at time t and the objective is to find an optimal control law minimizing the quadratic cost function:

$$\mathrm{E}\{\sum_{t=0}^{\infty}\gamma^t(x_t' Q_t x_t + u_t' R_t u_t)|x_0 = x\},$$

where t is the time index, x_t is the state vector of dimension n, u_t is the control vector of dimension m, A_t, B_t, Q_t, R_t are matrices with appropriate dimensions, and w_ts are independent random vectors with zero mean and finite variance and they do not depend on the state and control. Furthermore, Q_t is assumed to be positive semidefinite and R_t is positive definite so that the objective function is convex and a unique solution exists.

One nice feature of the LQR-type problems is that they can be solved analytically using algebraic Riccati equations. However, computing these equations can be computationally challenging. In addition, many applications require a model-free setting with the ability to compute the control policy online. Under these circumstances, the transition matrices A, B and cost matrices Q, R are unknown, but rewards/penalties and state transitions can be directly observed from a physical process. As a result, recursive stochastic algorithms are proposed to find the optimal control, and we would expect that an approximation dynamic programming algorithm with quadratic basis functions would converge to the true value function and optimal control. However, such convergence results only exist for the deterministic case by Brandke and Landelius and for the stochastic case by [43].

2.7.1 Bradtke's reinforcement learning algorithms

Reference [44] proposes two algorithms based on Q-learning to apply to deterministic infinite-horizon LQR problems (without the noise term w_t in the transition). One resembles a policy iteration algorithm, while the other belongs to optimistic policy iteration or classic value iteration algorithm. Optimistic policy iteration, introduced in [45], represents a form of partial policy evaluation. The policy iteration algorithm is provably convergent to the optimal control policy, but an example shows that the value iteration algorithm only converges locally.

For a deterministic LQR with fixed, stable policy π, as shown before, the value function has the form $v^{\pi}(x_t) = x_t K^{\pi} x_t$, where K^{π} is a positive semidefinite matrix. The Q-factor for a stable control policy π is defined to be

$$Q^{\pi}(x_t, u_t) = C(x_t, u_t) + \gamma Q^{\pi}(x_{t+1}, \pi(x_{t+1})).$$

With straightforward computation, it can be written as

$$Q^{\pi}(x, u) = [x, u]' H^{\pi} [x, u],$$

where

$$H^{\pi} = \begin{pmatrix} Q + \gamma A' K^{\pi} A & \gamma A' K^{\pi} B \\ \gamma B' K^{\pi} A & R + \gamma B' K^{\pi} B \end{pmatrix}.$$

The quadratic function can be rewritten as a linear function, e.g., $x'Qx = \Theta(Q)'\phi$, where ϕ is the vector of quadratic basis functions and $\Theta(Q)$ is the corresponding parameter vector. Then, the recursive formula becomes

$$C(x_t, u_t) = ([x_t, u_t] - \gamma[x_{t+1}, \pi(x_{t+1})])'\Theta(H^{\pi}) = \phi_t' \theta^{\pi},$$

where $[x, u]$ is a concatenation of x and u.

With this formulation, the least-squares method can be applied to estimate θ^{π}. Recursive least squares (RLS) is used in actual implementations since observations come sequentially and updates of parameters are also done sequentially. Reference [46] proved that starting with a stabilizing control and adding in a noise term to serve as a form of exploration (also known as persistent excitations), if a sufficiently accurate estimation of H^{π} for policy π_n is obtained before updating to an improved policy π_{n+1}, then the policy iteration algorithm is guaranteed to converge to the optimal control law π^*. On the other hand, a numerical example with scalar state and action in [44] shows that the value iteration algorithm only converges locally. If the initial policy is set close enough to an unstable solution, the algorithm converges to the unstable solution. This is because the unstable initial policy is very likely to generate a concave quadratic function approximation rather than a convex one in successive estimation. As a result, the policy obtained using differentiation maximizes rather than minimizes the value function in LQR. Failure of convergence to the true optimal policy is no surprise, and the algorithm should converge to the optimal control law if the initial estimation of the value function is convex and the control policy is set accordingly.

2.7.2 Landelius' heuristic dynamic programming algorithms

Reference [47] presents convergence analysis of several adaptive critic algorithms applied to LQR problems including heuristic dynamic programming (HDP, another name for approximate dynamic programming), dual heuristic programming (DHP, which works with derivatives of the value function instead), action-dependent HDP (ADHDP, another name for Q-learning), and action-dependent DHP (ADDHP), which were first introduced by [48]. The one that is of the most interest to us is the HDP algorithm since it uses greedy iteration (value iteration) and it is supposed to provide faster convergence.

For an infinite horizon problem, reference [47] uses the following updating formula for the parameter matrix:

$$\begin{aligned} K_{n+1} = K_n - \alpha x_n x_n' \{ x_n' [& K_n - Q - L_n' R L_n \\ & -(A + BL_n)' K_n (A + BL_n)] x_n \}, \end{aligned}$$

where $L_n = -(R + B'K_n B)B'K_n A$ and $0 \leqslant \alpha < 1$ is the fixed step size. Then, reference [47] transforms from matrix form to vector form by letting $\mathrm{vec}(K)$ be the vector formed by stacking columns of K. Then, letting $k_n = \mathrm{vec}(K_n)$ and $v_n = \mathrm{vec}(x_n x_n')$, the updating rule becomes

$$\begin{aligned} k_{n+1} = k_n - \alpha v_n v_n' \mathrm{vec}([& K_n - Q - L_n' R L_n \\ & -(A + BL_n)' K_n (A + BL_n)]). \end{aligned}$$

With this transformation, reference [47] is able to prove convergence of the algorithm. The basic difference between HDP and the Q-learning algorithm proposed in [44] is the parameter updating rule. Reference [44] uses RLS, while reference [47] uses a stochastic gradient method that may introduce a scaling problem.

2.7.3 Szita's stochastic algorithm

It seems reasonable to extend the two algorithms from the deterministic case to the stochastic case, but convergence for the stochastic gradient method is established in [43] only for the case of Gaussian noise that arises when observing the state, as is standard in many control theory applications. The state evolves as follows:

$$x_{t+1} = Ax_t + Bu_t + w_{t+1}, \quad y_{t+1} = Hx_t + \xi_{t+1},$$

where y_{t+1} is a noisy observation and w_{t+1}, ξ_{t+1} are uncorrelated, zero mean noise terms with covariance matrices C^w and C^{ξ}. The evolution of estimates of the state is

$$\begin{aligned} \hat{S}_{t+1} &= A\hat{S}_t + Bx_t + K_t(y_{t+1} - H\hat{S}_t), \\ K_t &= A\Sigma_t H^{\mathrm{T}} (H\Sigma_t H^{\mathrm{T}} + C^{\xi})^{-1}, \\ \Sigma_{t+1} &= C^w + A\Sigma_t A^{\mathrm{T}} - K_t H \Sigma_t A^{\mathrm{T}}. \end{aligned}$$

The updating rule for the parameter matrix is essentially the same as the algorithm in [47], but reference [43] uses the estimate of states and a stochastic step size. With the assumption that the state is bounded in expectation (equivalent to say that the state is bounded with high probability), reference [43] shows almost sure convergence of the algorithm. The proof relies heavily on the following lemma in [36].

Lemma 1 [36] Let J be a differentiable function, bounded below by J^*, and let ∇J be Lipschitz continuous. Suppose the sequence satisfies $w_{t+1} = w_t + \alpha_t s_t$ for a random vector s_t independent of $w_{t+1}, w_{t+2}, \dots$. Suppose s_t is a descent direction for J in the sense that $\mathrm{E}(s_t | w_t)^{\mathrm{T}} \nabla J(w_t) > \delta(\epsilon) > 0$ whenever $J(w_t) > J^* + \epsilon$. Suppose that

$$\mathrm{E}(\|s_t\|^2 | w_t) \leqslant K_1 J(w_t) + K_2 \mathrm{E}(s_t | w_t)^{\mathrm{T}} \nabla J(w_t) + K_3,$$

and α_t satisfies $\alpha_t > 0$, $\sum\limits_t \alpha_t = \infty$ and $\sum\limits_t \alpha_t < \infty$. Then,

$J(w_t) \to J^*$ with probability 1.

Here, reference [43] takes $J(w) = \frac{1}{2}\|w - w^*\|^2$ and the same descent direction used in [47].

2.7.4 Nonlinear HJB solution using approximate dynamic programming

From a control theory perspective, reference [49] considers a value-iteration-based approximate dynamic programming algorithm without knowledge of the internal dynamics of the system. Under the assumption of exact updates of the value function and action, it is proven that the algorithm converges to the optimal control and the optimal value function that solves the HJB equation in infinite-horizon discrete-time nonlinear optimal control. The algorithm is often used with neural networks for approximation: one critic network to approximate the value function and another actor network to approximate the optimal control policy. Specifically, LQR is a special case of the exact solution assumption for the nonlinear systems and the algorithm returns zero approximation error.

2.7.5 Discrete-time linear quadratic zero-sum game

With a model-free setting, reference [50] proposes a Q-learning-type approximate dynamic programming (forward in time) algorithm to find the optimal solution to discrete-time linear system quadratic zero-sum games (with continuous state and action spaces) related to the H-infinity optimal control problem. The algorithm adaptively trains the state-action Q-factor of the zero-sum game instead of the state value function forward in time using adaptive critic methods, and it is provably convergent to the equilibrium of the game. Reference [51] extends the work in [50] to two schemes: a) HDP to solve for value function and b) dual HDP to solve for the costate of the game.

2.8 Batch reinforcement learning

A series of papers [52–55] derive finite-sample probably approximately correctness (PAC) bounds for batch reinforcement learning problems. These performance bounds depend on the mixing rate of the sample trajectory (which captures the degree to which the stochastic process moves between states while following a policy), the smoothness properties and controllability of the underlying MDP, the approximation power and capacity of the function approximation method used, the iteration counter of the policy improvement step, and the sample size for policy evaluation. With these properties, they show the algorithms produce near-optimal policies with high probability. More specifically, reference [54] presents a sampling-based fitted value iteration algorithm (a special form of approximate value iteration) for MDP problems with large-scale or continuous state spaces but finite action spaces in an offline setting where a known system model can generate sample transitions from any initial state for each possible action. At each time step, the Monte Carlo simulation is used to generate estimates of the optimal value function with a PAC bound that controls the approximation error.

Assuming a model-free setting, references [52, 53] consider a fitted policy iteration algorithm based on a single trajectory of following some fixed sampling policy (off-policy) to solve MDPs with continuous state space and finite action space. The algorithm provides a finite-sample, high-probability bound on the performance of the VFA capacity. In addition, it is shown that the algorithm is equivalent to LSPI if linear approximation is used. In order to handle problems with continuous actions, reference [55] steps forward to extend previous algorithms to a variant of fitted Q-iteration, in which policy improvement is done by maximizing the average action values in a restricted set of potential policies rather than selecting the greedy policy. By imposing regularity conditions on the action space, finite-time, high-probability performance bounds can be obtained for the algorithm.

2.9 Kernel-based reinforcement learning

Reference [56] presents a kernel-based approach to approximate the value function, avoiding the problem of designing basis functions. The algorithm progresses by approximating the transition probabilities for each action using Monte Carlo samples, limiting the idea to relatively small action spaces. Then, the optimal policy is found given this approximation using conventional contraction arguments.

The algorithm is provably convergent to a unique solution to an approximate Bellman's equation regardless of its initial values for a finite training data set of historical transitions. As the size of the transition data set goes to infinity, the approximate value function and approximate policy converge in probability to the optimal value function and optimal policy, respectively. Moreover, it is shown that the value function estimate of the limiting distribution is Gaussian with the kernel-based approach.

Reference [57] considers a kernelized version of LSPI in [31], which applies the kernel RLS algorithm developed in [58] to LSTD-Q for approximating state-action Q-factors of a fixed policy. The kernel RLS algorithm is a nonlinear version of the RLS algorithm, which applies linear regression to a high-dimensional feature space induced by a Mercer kernel to recursively solve the nonlinear least-squares problem. To strike a balance between policy quality and learning efficiency, reference [57] uses approximate linear dependence developed in [58] for online sparsification by sequentially eliminating samples that can be well approximated by a linear span of other samples. A Kernel-based LSPI performs empirically better than LSPI in terms of learning efficiency and policy quality on typical nonlinear reinforcement learning tasks such as ship heading control and the inverted pendulum.

Reference [59] extends LSTD to the framework of a subspace-based variant least-squares support vector machine (LS-SVM) to handle online learning problems with a large training sample size and a sequential data process. Adapted from Gaussian process regression, the algorithm has only linear computational complexity compared to the original optimization problems of cubic complexity.

Reference [60] proposes another algorithm with online nonparametric approximation of the value function, called equigradient descent algorithm (EGD). It is a TD algorithm using kernel regression with least absolute shrinkage and selection operator (LASSO) for regularization, and the computational cost of the algorithm is kept reasonable.

References [61] present an approximate policy iteration algorithm using support vector regression (SVR) for Bellman residual minimization at selected states. When the entire state space is sampled, the algorithm converges to the same result as exact policy iteration in the limit under certain conditions about the chosen feature mapping and the associated kernel function.

Reference [62] proposes a fitted Q-iteration algorithm that can be combined with either linear or nonlinear function approximation and uses L_2 regularization to control the complexity of the value function in problems with continuous state space. Reference [63] extends reference [62] to an approximate policy iteration algorithm that applies nonparametric methods with L_2 regularization to policy evaluation approaches such as Bellman residual minimization and LSTD. Finite-sample performance bounds are provided for both algorithms to show that the optimal rates of convergence can be achieved under the certain conditions.

2.10 Gaussian process models

Another nonparametric approach to optimal control problems is the Gaussian process model, in which value functions are modeled with Gaussian processes. For fixed policy evaluation, reference [64] presents the Gaussian process TD algorithm, which is a Bayesian approach to VFA with continuous state spaces. The algorithm imposes a Gaussian prior over value functions and assumes a Gaussian noise model. Then, it updates the posterior Gaussian process parameters using an online sparsification method. Reference [65] then extends GPTD in [64] to the framework of approximate policy iteration. Reference [66] introduces a Gaussian process dynamic programming (GPDP) algorithm. It is an approximate value iteration algorithm, which models value functions in the dynamic programming recursion with Gaussian processes. These GP algorithms have successful applications in complex nonlinear control problems without convergence guarantees.

2.11 Other convergent algorithms

Reference [67] proves convergence of policy iteration algorithms for average cost optimal control problems with unbounded cost and general state space. The algorithm assumes countable action space and requires exact computation of the expectation. With further assumptions of a c-regular initial policy (a strong stability condition where c represents the cost function) and irreducibility of the state space, the algorithm generates a sequence of c-regular policies that converge to the optimal average cost policy.

One extension of the TD algorithm to the continuous domain can be found in [68,69], which show the convergence of a Q-learning algorithm with linear function approximation under a fixed learning policy for MDP problems with continuous state space but finite action space. Under strong technical conditions on the structure of basis functions, reference [68] proves almost sure convergence of Q-learning used with linear function approximation. Reference [69] shows convergence with probability 1 of Q-learning and SARSA algorithms with linear approximation under strong assumptions on the sampling policy.

2.12 Examples of divergence

Value iteration-type algorithms with linear function approximation can fail to converge. Reference [70] describes a counter-example of $\mathrm{TD}(0)$ algorithm with stochastic gradient updating for the parameters. It is shown that the algorithm converges but can generate a poor approximation of the optimal value function in terms of Euclidean distance. Reference [27] presents a similar counter-example as in [70] but with a least-squares updating rule for the parameter estimates, in which case divergence happens even when the optimal value function can be perfectly represented by the linear approximator. Reference [71] illustrates that divergent behavior occurs for value iteration algorithms with a variety of function approximation techniques such as polynomial regression, back-propagation, and local weighted regression when the algorithm is applied to simple nonlinear problems.

2.13 Continuous-time algorithms

With respect to the timescale of control input signals being sent to the system, the control community often distinguishes algorithms between discrete-time and continuous-time. However, continuous-time algorithms are out of the scope of this paper and interested readers can find details of the algorithms in the references [72–77].

3 Mathematical foundations for continuous MDPs

We begin by defining the MDP and the idea of post-decision states, which we use to avoid the imbedded expectation within the minimization operator. We then describe some important properties of MDPs with continuous states, and close with a brief introduction to policy iteration, which we use as the basis for our work.

We use as our basic model the MDP described in the introduction, where Bellman's equation is given by

$$V_t(x_t) = \sup_{u_t \in \mathcal{U}} \{C(x_t, u_t) + \gamma \mathrm{E}[V_{t+1}(x_{t+1})|x_t]\}, \quad (7)$$

where

$$x_{t+1} = S^{\mathrm{M}}(x_t, u_t, W_{t+1}). \quad (8)$$

For infinite horizon problems, we simply drop the time index throughout, recognizing that W is a random variable at time t. The following two assumptions on MDPs are imposed to facilitate convergence analysis.

Assumption 1 The state space $\mathcal{X}$, the decision space $\mathcal{U}$, and the outcome space $\mathcal{W}$ are convex, compact, and Borel subsets of $\mathbb{R}^m$, $\mathbb{R}^n$, and $\mathbb{R}^l$, respectively.

Assumption 2 The contribution function C, the state transition function S^{M}, and the transition probability density function $Q : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \rightarrow \mathbb{R}_+$ are all continuous.

3.1 Post-decision state variable

In order to avoid computing the expectation within the Bellman's equation directly, we use the idea of the post-decision state (see [78,79], also known as end-of-state [80] and after-state [42]). The transition function (8) is broken into the two steps:

$$x_t^u = S^{\mathrm{M},u}(x_t, u_t), \quad (9)$$
$$x_{t+1} = S^{\mathrm{M},W}(x_t^u, W_{t+1}). \quad (10)$$

x_t^u denotes the post-decision state immediately after a decision is made. To illustrate, take a simple reservoir management problem for example. Let x_t be the water supply at time t, u_t be how much to release or store, and W_{t+1} be random precipitation. Then, using post-decision states, the state transition becomes

$$x_t^u = x_t - u_t, \;\; x_{t+1} = x_t^u + W_{t+1}.$$

Constructing a post-decision state typically reflects the structure of the problem. We might be traversing a graph, where when we arrive at a nice i, we are able to see sample realizations of costs c_{ij} on links emanating from node i. The pre-decision state would be $x_t = (i, (c_{ij})_j)$, consisting of the current node (node i) and the costs on links emanating from node i. If we decide to go to node k, the post-decision state would be $x_t^u = k$, since we have not actually arrived at node k, and have not observed the costs out of k. Another form of post-decision state is $x_t^u = (x_t, u_t)$ (which is what is done with Q-learning), but if u_t is vector-valued, the resulting state variable becomes much more difficult to work with. See Chapters 4 and 5 in [14] for more thorough discussions of the concept of post-decision states.

Let the post-decision state space denote $\mathcal{X}^u$ and the post-decision value function $V^u : \mathcal{X}^u \to \mathbb{R}$ be

$$V^u(x_t^u) = \mathrm{E}\{V(x_{t+1})|x_t^u\}, \tag{11}$$

where $V^u(x_t^u)$ is the value of being in the post decision states x_t^u. There is a simple relationship between the pre- and post-decision value functions:

$$V(x_t) = \max_{u_t \in \mathcal{U}} \{C(x_t, u_t) + \gamma V^u(x_t^u)\}. \tag{12}$$

The Bellman's equation of post-decision value function is

$$V^u(x_t^u) = \mathrm{E}\{\max_{u_{t+1} \in \mathcal{U}} \{C(x_{t+1}, u_{t+1}) + \gamma V^u(x_{t+1}^u)\}|x_t^u\}. \tag{13}$$

For a fixed policy, the MDP becomes a Markov chain on the post-decision states. The Bellman's equation (13) for the post-decision state is

$$V^\pi(x) = \int_{\mathcal{X}^\pi} P^\pi(x, \mathrm{d}x')(C^\pi(x, x') + \gamma V^\pi(x')), \tag{14}$$

where V^π is the policy value funcion, $P^\pi(\cdot,\cdot)$ is the transition probability function of the chain, $C^\pi(\cdot,\cdot)$ is the stochastic reward function with $C^\pi(x_t^\pi, x_{t+1}^\pi) = C(x_{t+1}, \pi(x_{t+1}))$, and $\mathcal{X}^\pi \subset \mathbb{R}^d$ is the post-decision state space of policy π.

Q-learning is often used to approximate the Q-factor $Q(x,u)$ around the state-action pair, which is impractical for high-dimensional applications. With post-decision states, instead of approximating $Q(x,u)$, we only have to approximate $V^u(x^u)$, which for most applications is much easier. It reduces complexity significantly because the post-decision state is often of much smaller dimensions than the state-action pair.

3.2 Markov chains with continuous state space

Classic approximate dynamic programming methods for discrete states can generally be applied to continuous states without modification, but the proofs can be considerably more intricate. Stability of the underlying process is critical in the convergence analysis of any stochastic algorithms. The underlying stochastic process becomes a Markov chain for fixed policy evaluation. The following definitions related to positive Harris chains are helpful in convergence analysis with Monte Carlo sampling because the well-known strong law of large numbers [81] can be applied.

Definition 1 (ψ-irreducibility) For any measure φ, a Markov chain $\varPhi$ on state space $\mathcal{X}$ is called φ-irreducible if there exists a measure φ on $\mathcal{B}(\mathcal{X})$ such that whenever $\varphi(A) > 0$ for $A \in \mathcal{B}(\mathcal{X})$, we have

$$\mathrm{P}_x\{\varPhi \text{ ever enters } A\} > 0, \;\; \forall x \in \mathcal{X},$$

where P_x denotes the conditional probability on the event that the chain starts in state x. Let ψ be the maximal irreducibility measure among such measures, and the chain is called ψ-irreducible (For the existence of ψ, see Proposition 4.2.2 of [81]).

Definition 2 (Invariant measure) Let $P(\cdot,\cdot)$ be the transition kernel of a chain $\varPhi$ on the state space $\mathcal{X}$. A σ-finite measure μ on $\mathcal{B}(\mathcal{X})$ with the property

$$\mu(A) = \int_{\mathcal{X}} \mu(\mathrm{d}x) P(x, A), \;\; \forall A \in \mathcal{B}(\mathcal{X})$$

will be called invariant.

Definition 3 (Positive Harris chain) The set $A \in \mathcal{B}(\mathcal{X})$ is called Harris recurrent if

$$\mathrm{P}_x\{\varPhi \in A \text{ infinitely often}\} = 1, \;\; \forall x \in \mathcal{X}.$$

A chain $\varPhi$ is called Harris (recurrent) if it is ψ-irreducible, admits an invariant probability measure μ and every set in

$$\mathcal{B}^+(\mathcal{X}) = \{A \in \mathcal{B}(\mathcal{X}) : \psi(A) > 0\}$$

is Harris recurrent.

3.3 Policy iteration

The new algorithms we present below are all variations of policy iteration, which typically consists of two steps: fixed policy evaluation and policy improvement. The convergence result of the exact policy iteration algorithm is well known (e.g., see [82]). However, policy evaluation can rarely be achieved exactly, so approximate policy iteration, in which the policy evaluation is done approximately, is often used in practice. Approximate policy iteration achieves a statistical estimate (possibly biased) of the true value function of the policy based on the sample trajectory of the chain by following a policy. Given a compact state space with the sup norm $\|\cdot\|_\infty$ defined for continuous functions, reference [83] proves mean convergence of approximate policy iteration in the following theorem, which is the foundation for the convergence analysis of all the variations. The proof is an extension of error bounds for approximate policy iteration with discrete and deterministic VFAs in [7].

Theorem 1 (Mean convergence of approximate policy iteration) Let $(\hat{\pi}_i)_{i=0}^n$ be the sequence of policies generated by an approximate policy iteration algorithm and $(\hat{V}^{\hat{\pi}_i})_{i=0}^n$ be the corresponding approximate policy value functions. Further assume that, for each fixed policy $\hat{\pi}_n$, the MDP is reduced to a Markov chain that admits an invariant probability measure $\mu_{\hat{\pi}_n}$. Let $\{\epsilon_n\}$ and $\{\delta_n\}$ be positive scalars that bound the mean errors in approximations to value functions and policies (over all iterations), respectively, that is, $\forall n \in \mathbb{N}$,

$$\mathrm{E}_{\mu^{\hat{\pi}_n}} \|\hat{V}^{\hat{\pi}_n} - V^{\hat{\pi}_n}\|_\infty \leqslant \epsilon_n, \tag{15}$$

$$\mathrm{E}_{\mu^{\hat{\pi}_n}} \|M_{\hat{\pi}_{n+1}} \hat{V}^{\hat{\pi}_n} - M \hat{V}^{\hat{\pi}_n}\|_\infty \leqslant \delta_n. \tag{16}$$

Suppose the sequences $\{\epsilon_n\}$ and $\{\delta_n\}$ converge to 0 and $\lim\limits_{n\to\infty}\sum\limits_{i=0}^{n-1}\gamma^{n-1-i}\epsilon_i = \lim\limits_{i=0}^{n-1}\gamma^{n-1-i}\delta_i = 0$, e.g., $\epsilon_i = \delta_i = \gamma^i$. Then, this sequence eventually produces policies whose performance converges to the optimal performance in the mean: $\lim\limits_{n\to\infty} \mathrm{E}_{\mu^{\hat{\pi}_n}}\|\hat{V}^{\hat{\pi}_n} - V^*\|_\infty = 0$.

4 Algorithms for multidimensional continuous MDPs

In this section, we propose algorithms that are specifically designed for problems with multidimensional and continuous states and actions, and where the expectations cannot be computed exactly. The algorithms use both parametric and nonparametric architectures. We also use on-policy learning, since off-policy sampling strategies can be impractical for multidimensional and continuous action spaces.

The details of the convergence analysis can be found in [83, 84]. We start with the preliminaries that are crucial for understanding the algorithms.

4.1 RLS approximate policy iteration with known basis functions

Reference [83] extends the off-policy LSPI introduced by [31] to an online, on-policy RLS approximate policy iteration (RLSAPI) algorithm, which is summarized in Table 1. LSTD by [7] is used to combine linear VFA with approximate policy iteration for policy evaluation. Instead of working with the state-action Q-factor as in LSPI, RLSAPI applies LSTD to approximate the value functions around the post-decision state to avoid the computation of the expectation. Furthermore, RLSAPI extends LSTD to the continuous problem class and solves the identification problem for parameter convergence with continuous states. RLSAPI is able to handle continuous (and vector-valued) states and actions as long as we have an algorithm that will solve the deterministic optimization problem.

Table 1 Infinite-horizon approximate policy iteration algorithm with RLS method [83].

```
Step 0  Initialization:
        Set the initial values of the value function parameters $\hat{\theta}_0$;
        Set the initial policy $\pi_1(x) = \arg\max_{u\in\Gamma(x)}\{C(x,u) + \gamma\phi(x^u)^{\mathrm{T}}\hat{\theta}_0\}$;
        Set the iteration counter $n = 1$.
Step 1  Do for $n = 1,\dots,N$,
        Set the initial State $x_0^n$.
Step 2  Do for $m = 0,\dots,M$,
        Initialize $\hat{\theta}_{n,m}$ and $\hat{v}_m = 0$;
        Draw randomly or observe $W_{m+1}$ from the chain;
        Do the following:
          Set $u_m^n = \pi_n(x_m^n)$,
          Compute $x_m^{n,\pi} = S^{\mathrm{M},\pi}(x_m^n, u_m^n)$ and $x_{m+1}^n = S^{\mathrm{M},W}(x_m^{n,\pi}, W_{m+1})$,
          Compute $u_{m+1}^n = \pi_n(x_{m+1}^n)$ and $x_{m+1}^{n,\pi} = S^{\mathrm{M},\pi}(x_{m+1}^n, u_{m+1}^n)$,
          Compute input variable/regressor using the corresponding basis function values: $\phi(x_m^{n,\pi}) - \gamma\phi(x_{m+1}^{n,\pi})$.
Step 2  Do the following:
        Compute/observe the response variable $\hat{v}_m = C(x_m^{n,\pi}, x_{m+1}^{n,\pi})$;
        Update parameters $\hat{\theta}_{n,m}$ with LS/RLS method that regresses response $\hat{v}_m$ on regressor $\phi(x_m^{n,\pi}) - \gamma\phi(x_{m+1}^{n,\pi})$.
Step 3  Update the parameter and the policy $\hat{\theta}_{n+1} = \hat{\theta}_{n,M}$, $\pi_{n+1}(x) = \arg\max_{u\in\Gamma(x)}\{C(x,u) + \gamma\phi(x^u)^{\mathrm{T}}\hat{\theta}_n\}$.
Step 4  Return the policy $\pi_{N+1}$ and parameters $\hat{\theta}_N$.
```

To prove convergence and simplify the policy improvement, the following assumption on the policy value function is proposed.

Assumption 3 Assume that the policy value function for a fixed policy π is continuous with a linear architecture, i.e., $V^\pi(x|\theta) = \phi(x)^{\mathrm{T}}\theta$, where $\phi(x) = [\cdots\ \phi_f(x)\ \cdots]$ is the vector of basis functions of dimension $F = |\mathcal{F}|$ and $f \in \mathcal{F}$ ($\mathcal{F}$ denotes the set of features or basis functions).

The key step in RLSAPI is LSTD for policy evaluation. Reference [83] follows [29] to use the error-in-variable linear regression for parameter estimates. Rearranged from Bellman's equation

$$\phi(x)^{\mathrm{T}}\theta^* = \int_{\mathcal{X}^\pi} P^\pi(x, \mathrm{d}x')[C^\pi(x,x') + \gamma\phi(x')^{\mathrm{T}}\theta^*].$$

Rearranging terms, this can be written as

$$C^\pi(x,x') = \big(\phi(x) - \gamma\int_{\mathcal{X}^\pi} P(x,\mathrm{d}x')\phi(x')\big)^{\mathrm{T}}\theta^* + C^\pi(x,x') - \int_{\mathcal{X}^\pi} P(x,\mathrm{d}x')C^\pi(x,x'),$$

where $\phi(x)^{\mathrm{T}}\theta^*$ is the true value function of following the fixed policy π, $P^\pi(\,\cdot,\cdot\,)$ is the transition probability function of the chain, $C^\pi(\,\cdot,\cdot\,)$ is the stochastic reward function with $C^\pi(x_t^\pi, x_{t+1}^\pi) = C(x_{t+1}, \pi(x_{t+1}))$, and $\mathcal{X}^\pi$ is the post-decision state space by following policy π. Then, the estimate of θ^* using m samples is

$$\theta_m = \Big[\frac{1}{m+1}\sum_{i=0}^{m}\phi_i(\phi_i - \gamma\phi_{i+1})^{\mathrm{T}}\Big]^{-1}\Big[\frac{1}{m+1}\sum_{i=0}^{m}\phi_i C_i\Big], \tag{17}$$

where $C_i = C^\pi(x_i, x_{i+1})$ denotes the ith observation of the reward.

Convergence of LSTD requires stability of the underlying Markov chain of a fixed policy π. If a chain is positive Harris recurrent, the invariant measure μ^π satisfies the stability requirement. Unlike discrete states with a lookup table representation, where we have to guarantee that we visit *every* state infinitely often, with basis functions, we only need to ensure that we visit enough states often enough that we can identify the parameter vector θ.

Owing to the assumptions on the state space and transition functions, it suffices to have a ψ-irreducible Markov chain. Hence, the following assumption on the underlying system is imposed.

Assumption 4 Assume that, for all $\pi \in \Pi$, the MDP is reduced to a ψ-irreducible Markov chain Φ^π with state space $\mathcal{X}^\pi$ and the support of ψ has nonempty interior.

Convergence of RLSAPI is fully determined by the convergence of LSTD for policy evaluation. The following theorem illustrates the details.

Theorem 2 (Convergence in the mean of RLSAPI) Under Assumptions 1, 2, and 4, suppose that for any policy $\pi \in \Pi$ the policy value function V^π satisfies Assumption 3 with known basis functions that are either linearly independent (with invertible correlation matrix) or orthonormal with respect to the invariant measure μ^π. Then, LSTD for policy evaluation converges μ^π-almost surely and Theorem 1 holds for RLSAPI.

Note that RLSAPI does not have any explicit exploration policy. Instead, it simply assumes that there is a density μ^π giving the likelihood of visiting each state, which has certain properties. If we assume that a value function is perfectly modeled by a given set of basis functions, exploration is less important. The real issue is identification. Are we visiting enough different states to statistically identify θ? A careful exploration policy, however, can influence the rate of convergence, just as it is easier to fit the slope of a linear function when we measure covariates that are far from the center.

If the basis functions do not capture the function perfectly, then the right choice of θ depends on the probability we will visit different states. Of course, the likelihood that we will visit different states depends on our policy, which in turn depends on the VFA. It is precisely this circular dependence that complicates the identification of optimal policies in approximate dynamic programming. We circumvent this issue with our assumption of an invariant probability measure. This is not a major assumption if our basis functions capture the optimal value function, but becomes more of an issue when the accuracy of our approximation degrades.

4.2 Recursive LSPI policy iteration with Mercer kernels

Using an idea similar to [57], reference [84] presents a kernelized version of the RLS approximate policy algorithm (KRLSAPI) using the post-decision state value function instead of the state-action Q-factor. In order to apply LSTD with nonlinear function approximation, the Mercer kernel trick, which is stated in the following theorem, is applied to convert a nonlinear architecture to a linear architecture (specifically, linear in the parameters). As a result, the algorithm can be shown to be convergent using similar proof techniques for algorithms with linear function approximation.

Theorem 3 (Mercer's theorem) Let $\mathcal{S}$ be a measurable space and the kernel K be a positive and semidefinite function, i.e., $\sum\limits_{i,j} K(s_i,s_j)r_ir_j \geqslant 0$, for any finite subset $\{s_1,\dots,s_n\}$ of $\mathcal{S}$ and any real numbers $\{r_1,\dots,r_n\}$. There exists a function $\phi:\mathcal{S}\to\mathcal{F}$, where $\mathcal{F}$ (feature space) is an inner product space of possibly high dimension, such that $K(x,y)=\langle\phi(x),\phi(y)\rangle$.

The algorithm transforms the original nonlinear function space into a higher-dimensional inner product feature space $\mathcal{F}$, so that linear algorithms such as LSTD can be subsequently applied for estimation. However, the dot products between two vectors of feature function ϕ, which is necessary for parameter estimates, are replaced with the kernel function in the implementation of the algorithm, so the feature vectors do not need to be explicitly calculated. This gives us the attractive capability that infinite-dimensional feature spaces (imagine very high-dimensional polynomial basis functions) can be represented without explicitly calculating these features.

More specifically, for policy evaluation, KRLSAPI uses KLSTD, which maps the nonlinear LSTD function space to a high-dimensional feature space $\mathcal{F}$. Denote the mapping associated with the feature space $\mathcal{F}$ to be $\phi:\mathcal{S}^\pi\to\mathcal{F}$, where $\mathcal{S}^\pi\subset\mathbb{R}^d$ is the original post-decision state space arising when following a fixed policy π, and $\phi(x)$ is the vector of feature functions. Then, the policy value function becomes $\hat{V}^\pi_m(x)=\phi(x)^{\mathrm{T}}\theta_m$, where $\phi(x)$ and θ are both column vectors and m is the number of samples. The Kernel Representor theorem in [85] ensures that the weight vector θ can be rewritten as the weighted sum of state feature vectors, i.e., $\theta_m=\sum\limits_{i=0}^{m}\phi(x_i)\beta_i$, where x_i $(i=1,2,\dots,m)$ are the sampled post-decision states and β_i $(i=1,2,\dots,m)$ are the parameters. Hence, by the reproducing kernel property,

$$\hat{V}^\pi(x)=\phi^{\mathrm{T}}(x)\theta=\sum_{i=1}^{m}K(x,x_i)\beta_i.$$

Then, the least-squares regression equation for KLSTD becomes

$$\sum_{i=1}^{m}\phi(x_i)(\phi(x_i)-\gamma\phi(x_{i+1}))^{\mathrm{T}}\sum_{i=1}^{m}\phi(x_i)\beta_i = \sum_{i=1}^{m}\phi(x_i)c_i+\epsilon_i,$$

where $c_i=C^\pi(x_i,x_{i+1})$ is the ith observation of the contribution and ϵ_i is the noise for each time step. The single-step regression function is

$$\phi(x_i)(\phi(x_i)-\gamma\phi(x_{i+1}))^{\mathrm{T}}\sum_{i=1}^{m}\phi(x_i)\beta_i = \sum_{i=1}^{m}\phi(x_i)c_i+\epsilon_i.$$

Denote

$$\Phi_m=[\phi(x_1)\ \cdots\ \phi(x_m)]^{\mathrm{T}},$$
$$k_m(x_i)=[K(x_1,x_i)\ \cdots\ K(x_m,x_i)]^{\mathrm{T}}.$$

Since all the inner products in the feature space can be replaced with kernel function $K(x,y)=\langle\phi(x),\phi(y)\rangle$ due to the kernel trick, by multiplying Φ_m on both sides of the previous equation, the least-squares regression function becomes

$$\sum_{i=1}^{m} k_m(x_i)(k_m(x_i) - \gamma k_m(x_{i+1}))^{\mathrm{T}}\beta_m = \sum_{i=1}^{m} k_m(x_i)c_i.$$

Let

$$M_m = \sum_{i=1}^{m} k_m(x_i)(k_m(x_i) - \gamma k_m(x_{i+1}))^{\mathrm{T}}$$
$$b_m = \sum_{i=1}^{m} k_m(x_i)c_i.$$

Then, the parameter estimates can be computed recursively as follows:

$$\beta_m = M_m^{-1} b_m$$

with

$$\begin{aligned} M_{m+1} = M_m + k_{m+1}(x_{m+1})(&k_{m+1}^{\mathrm{T}}(x_{m+1}) \\ &-\gamma k_{m+1}^{\mathrm{T}}(x_{m+2})), \end{aligned}$$
$$b_{m+1} = b_m + k_{m+1}(x_{m+1})c_m.$$

Like other policy iteration algorithms, after each policy evaluation step, KRLSAPI makes incremental policy improvements until the optimal policy is reached. It is worth noting that the automatic feature generation property of KRLSAPI provides a solution to the feature selection problem of the LSPI algorithm. The details of the KRLSAPI algorithm are illustrated in Table 2.

Table 2 Infinite-horizon kernel-based approximate policy iteration algorithm with least-squares method [84].

```
Step 0  Initialization:
        Set the initial policy $\pi_0$;
        Set the kernel function $K$;
        Set the iteration counter $n = 0$;
        Set the initial State $x_0^0$.
Step 1  Do for $n = 0, \dots, N$,
        Do for $m = 1, \dots, M$,
        Initialize $\hat{c}_m = 0$;
        Choose one step sample realization $\omega$;
        Do the following:
          Set $u_m^n = \pi_n(x_m^n)$,
          Compute $x_m^{n,\pi} = S^{\mathrm{M},\pi}(x_m^n, u_m^n)$ and $x_{m+1}^n = S^{\mathrm{M}}(x_m^{n,\pi}, u_m^n, W_{m+1}(\omega))$,
          Compute and store the corresponding kernel function value $k_m(S_m^{n,x})$ and $k_m(S_{m+1}^{n,x})$ in dictionary.
Step 2  Do the following:
        Compute and store $\hat{c}_m = C(x_m^n, u_m^n)$ and $M_m$ and $b_m$;
        Update parameters $\hat{\beta}^{n,m} = M_m^{-1} b_m$.
Step 3  Update the parameter and the policy: $\hat{\beta}^{n+1} = \hat{\beta}^{n,M}, \pi_{n+1}(x) = \arg\max_{u \in \mathcal{U}}\{C(x,u) + \gamma k_M(x^u)^{\mathrm{T}}\hat{\beta}^{n+1}\}$.
Step 4  Return the policy $\pi_t^N$ and parameters $\hat{\beta}^N$.
```

For convergence analysis, we use the notion of a reproducing kernel Hilbert space (RKHS). There is a Hilbert space H_K associated with each positive semidefinite kernel function K, and H_K is a vector space containing all linear combinations of the functions $K(\,\cdot, x)$, $f(\,\cdot\,) = \sum_{i=1}^{m} \alpha_i K(\,\cdot, x_i)$. Let $g(\,\cdot\,) = \sum_{j=1}^{n} \beta_j K(\,\cdot, y_j)$. The inner product over H_K is defined as

$$\langle f, g \rangle = \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_i \beta_j K(x_i, y_j),$$

and the norm is $\|f\|_{H_K} = \sqrt{\langle f, f \rangle}$. The following assumption on the policy space Π is imposed for convergence analysis purpose.

Assumption 5 Assume that the policy value function for any fixed policy $\pi \in \Pi$ is in the RKHS H_K.

This is a much weaker assumption than if we required a value function to be in the space spanned by a fixed set of basis functions. Convergence of KLSTD combined with the approximation errors of both policy evaluation and policy improvement determines the convergence of KRLSAPI, and Theorem 4 states the details of the convergence results.

Theorem 4 (Mean convergence of KRLSAPI [84]) Suppose Assumptions 1, 2, and 5 hold and the Markov chain around the post-decision states following any fixed policy $\pi \in \Pi$ is a positive Harris chain having transition kernel $P^{\pi}(x, \mathrm{d}y)$ and invariant probability measure μ^{π}. Further assume the policy value function V^{π} is in the RKHS H_K with the kernel function K being C^{∞} and $\|K\|_{\infty} \leqslant M$ for some constant M. Then, $\hat{V}_m^{\pi} \to V^{\pi}$ in probability where

$$\hat{V}_m^{\pi}(x) = \sum_{i=1}^{m} K(x, x_i)\beta_i,$$

and Theorem 1 holds for the KRLSAPI algorithm in Table 2.

The algorithm in Table 2 requires high computational and memory capacity in implementation since all past samples have to be stored. Kernel sparsification procedures can be applied to deal with this problem, including approximate linear dependence developed in [57, 58] for online sparsification, subspace-based variant of LS-SVM by [59]; Bellman residual minimization methods and SVR by [61]; LASSO L_1 regularization by [60]; and L_2 regularization by [62, 63].

4.3 Approximate policy iteration with kernel smoothing

Reference [84] extends the kernel-based value iteration algorithm for continuous MDPs by [56] to an online, on-policy approximate policy iteration algorithm by avoiding two major limitations in [56]: the requirement of a finite action space and the need for an off-policy, offline uniform sampling scheme. We caution the reader that we avoid the need for an offline uniform sampling scheme by introducing a technical assumption on the behavior of the Markov process for a fixed policy.

It starts with a random approximation of the fixed-policy Bellman operator $\hat{M}^{\pi}_{m}$ from a sample trajectory:

$$\hat{M}^{\pi}_{m}V(x)=\sum_{i=0}^{m-1}k(x_i,x)\left(C^{\pi}(x_i,x_{i+1})+\gamma V(x_{i+1})\right),$$

where

$$k(x_i,x)=K(\frac{\|x_i-x\|}{b})/\sum_{i=0}^{m-1}K(\frac{\|x_i-x\|}{b}),$$

and $(x_i)_{i=0}^{m}$ is the sample trajectory of past transitions following the policy π. The kernel weighting function k, which is based on a univariate kernel function K, assigns weights according to the distances between points. By construction, the weights are all nonnegative and add up to 1. Owing to the fixed point property of M^{π}, $\hat{M}^{\pi}_{m}V^{\pi}$ is used to approximate the true post-decision policy value function V^{π} on the finite sample of post-decision states $(x_i)_{i=1}^{m}$. The algorithm searches for the fixed point solution to the approximate Bellman equation

$$\hat{V}^{\pi}=\hat{M}^{\pi}_{m}\hat{V}^{\pi}=\hat{P}^{\pi}[c^{\pi}+\gamma\hat{V}^{\pi}], \tag{18}$$

where $\hat{V}^{\pi}$ is the post-decision state vector of size m, P^{π} is a $m\times m$ stochastic matrix with the i,jth entry being $k(x_{i-1},x_j)$ for $i,j\in\{1,\dots,m\}$, and c^{π} is the reward vector of dimension m with the ith entry being $C^{\pi}(x_{i-1},x_i)$ for $i\in\{1,\dots,m\}$. The existence and uniqueness of the solution is guaranteed, since $\hat{P}^{\pi}$ is a stochastic matrix and $I-\gamma P^{\pi}$ is invertible. The solution is

$$\hat{V}^{\pi}=(I-\gamma\hat{P}^{\pi})^{-1}c^{\pi}. \tag{19}$$

Then, the function estimates are extrapolated to the points (that are not in the sample) $x\in\mathcal{X}^{\pi}\setminus\{x_i\}_{i=1}^{m}$ with

$$\hat{V}^{\pi}(x)=\sum_{i=0}^{m-1}k(x_i,x)(C^{\pi}(x_i,x_{i+1})+\gamma\hat{V}^{\pi}(x_{i+1})). \tag{20}$$

The details of the algorithm are illustrated in Table 3.

Table 3 Infinite-horizon kernel-based approximate policy iteration algorithm [84].

```
Step 0  Initialization:
        Set the initial policy $\pi_0$;
        Set the kernel function $K$;
        Set the iteration counter $n=0$.
Step 1  Do for $n=0,\dots,N$,
        Set the iteration counter $l=0$;
        Set the initial state $x_0^n$;
        Do for $j=0,\dots,m$,
          Set $u_j^n=\pi_n(x_j^n)$ and draw randomly or observe $W_{j+1}$ from the stochastic process,
          Compute $x_j^{n,\pi}=S^{\mathrm{M},\pi}(x_j^n,u_j^n)$ (store also) and $x_{j+1}^n=S^{\mathrm{M},W}(x_j^{n,\pi},W_{j+1})$;
        Let $c^{\pi}$ be a vector of dimensionality $m$ with $i$th entry $C^{\pi}(x_{i-1}^{n,\pi},x_i^{n,\pi})$ for $i=1,\dots,m$;
        Let $\hat{P}^{\pi}$ be a matrix of dimensionality $m\times m$ with $i,j$th entry $k(x_{i-1}^{n,\pi},x_j^{n,\pi})$ for $i,j\in\{1,\dots,m\}$.
Step 2  Solve for $\hat{v}=(I-\gamma\hat{P}^{\pi})^{-1}c^{\pi}$ with $\hat{v}$ being a vector of dimensionality $m$ with $i$th element $\hat{v}(x_i^{n,\pi})$ for $i=1,\dots,m$.
Step 3  Let $\hat{v}^n(x)=\sum_{i=0}^{m-1}k(x_i^{n,\pi},x)\left(C^{\pi}(x_i^{n,\pi},x_{i+1}^{n,\pi})+\gamma\hat{v}(x_{i+1}^{n,\pi})\right)$.
Step 4  Update the policy: $\pi_{n+1}(x)=\arg\max_{u\in\mathcal{U}}\{C(x,u)+\gamma\hat{v}^n(x^u)\}$.
Step 5  Return the policy $\pi_{N+1}$.
```

Under the following additional technical assumptions on state space, reward function, kernel, and the underlying Markov chain, the algorithm is convergent in mean.

Assumption 6 a) For each policy $\pi\in\varPi$, $\mathcal{S}^{\pi}=[0,1]^d$.

b) The contribution function, $C^{\pi}(x,y)$ is a jointly Lipschitz continuous function of x and y, i.e., there exists a $K_C>0$ such that

$$|r(x',y')-r(x,y)|\leqslant K_C\|(x'-x,y'-y)\|. \tag{21}$$

c) The kernel function $K^{+}:[0,1]\to\mathbb{R}^{+}$ is Lipschitz continuous, satisfying

$$\int_0^1 K^{+}(x)\mathrm{d}x=1$$

and K is the completion of K^{+} on $\mathbb{R}$.

d) For each policy $\pi\in\varPi$, the invariant probability measure μ^{π} is absolutely continuous with respect to the Lebesgue measure λ and

$$0<\underline{K}^{\pi}\leqslant\frac{\mathrm{d}\mu^{\pi}}{\mathrm{d}\lambda}\leqslant\bar{K}^{\pi}.$$

In other words, the invariant probability measure μ^{π} has a continuous density function f^{π} such that f^{π} is bounded from above and away from 0 on $\mathcal{S}^{\pi}$.

It is worth noting that the assumption of the post-decision state space being a d-dimensional unit cube $[0,1]^d$ can be relaxed to $\mathcal{X}^{\pi}=[a_1,b_1]\times\dots\times[a_d,b_d]$ since $\mathcal{X}^{\pi}$ is isomorphic to the unit cube $[0,1]^d$ with a bijective linear mapping $L:\mathcal{X}^{\pi}\to[0,1]^d$. The policy value function V^{π} on $\mathcal{X}^{\pi}$ can be easily recovered from policy value functions on the unit cube. With the above assumptions, mean convergence analysis is stated in the following theorem.

Theorem 5 (Convergence in mean of the kernel-based API [84]) Suppose Assumptions 1, 2, and 6 hold and for any policy $\pi\in\varPi$, the Markov chain of the post-decision states follows a positive Harris chain having transition kernel $P(x,\mathrm{d}y)$ and invariant probability measure μ. Let the bandwidth $b(m)$ satisfy $b(m)^{d+1}\sqrt{m}\to\infty$ and $b(m)\to 0$, e.g., $b(m)=m^{-\frac{1}{2(d+2)}}$. Let $\hat{V}^{\pi}_{m}$ be defined as in (20). Then,

$\mathrm{E}_{\mu^\pi}\|\hat{V}_m^\pi - V^\pi\|_\infty \to 0$ as $m \to \infty$, and Theorem 1 applies to the kernel-based approximate algorithm in Table 3.

There is a trade-off between the convergence performance of the algorithm and the sample efficiency. The matrix inversion in Step 2 of the algorithm requires $\mathrm{O}(m^3)$ computational complexity, which causes difficulty if a large sample is used to ensure convergence performance. Hence, reference [84] proposes a hybrid value/policy iteration algorithm, which is illustrated in Table 4. It is called hybrid because a value iteration type updating ($\mathrm{O}(m^2)$ computational complexity) is applied to policy evaluation. In matrix notation, the updating is

$$\hat{V}_{m,k+1}^\pi = \hat{P}^\pi(C^\pi + \gamma\hat{V}_{m,k}^\pi), \tag{22}$$

where $\hat{V}_{m,k}^\pi$ and $\hat{V}_{m,k+1}^\pi$ are the old and updated policy value function estimates, respectively.

Since the random operator $\hat{M}^\pi$ has the contraction property, the Banach fixed point theorem guarantees the convergence of the updating in (22) to a unique fixed point solution satisfying equation (18), which is just $\hat{V}_m^\pi$. Hence, Theorem 6 states that, under the same set of conditions, the hybrid algorithm in Table 4 is convergent in mean as well.

Theorem 6 (Convergence in mean of the hybrid kernel-based API) Suppose the assumptions in Theorem 5 hold. Theorem 1 applies to the hybrid kernel-based approximate algorithm in Table 4 with policy evaluation stopping criterion satisfies

$$\|\hat{V}_{m,k+1}^\pi - \hat{V}_{m,k}^\pi\|_\infty \leqslant \frac{1-\gamma}{2\gamma}\epsilon_n. \tag{23}$$

Table 4 Infinite-horizon kernel-based hybrid value/policy iteration algorithm [84].

```
Step 0  Initialization:
        Set the initial policy $\pi_0$;
        Set the kernel function $K$;
        Set the iteration counter $n = 0$.
Step 1  Do for $n = 0, \ldots, N$,
        Set the iteration counter $l = 0$;
        Set the initial state $x_0^n$;
        Do for $j = 0, \ldots, m$,
          Set $u_j^n = \pi_n(x_j^n)$ and draw randomly or observe $W_{j+1}$ from the stochastic process,
          Compute $x_j^{n,\pi} = S^{\mathrm{M},\pi}(x_j^n, u_j^n)$ (store also) and $x_{j+1}^n = S^{\mathrm{M},W}(x_j^{n,\pi}, W_{j+1})$;
        Initialize $\hat{v}_0$ where $\hat{v}_0$ is a vector of dimensionality $m$ with element $\hat{v}_0(x_i^{n,pi})$ for $i = 1, \ldots, m$;
        Let $C^\pi$ be a vector of dimensionality $m$ with $i$th entry $C^\pi(x_{i-1}^{n,\pi}, x_i^{n,\pi})$ for $i = 1, \ldots, m$;
        Let $\hat{P}^\pi$ be a matrix of dimensionality $m \times m$ with $i,j$th entry $k(x_{i-1}^{n,\pi}, x_j^{n,\pi})$ for $i, j \in \{1, \ldots, m\}$.
Step 2  Do for $l = 0, \ldots, L-1$,
        $\hat{v}_{l+1} = c^\pi + \gamma\hat{P}^\pi\hat{v}_l$.
Step 3  Let $\hat{v}^n(x) = \sum_{i=0}^{m-1} k(x_i^{n,\pi}, x)\big(C^\pi(x_i^{n,\pi}, x_{i+1}^{n,\pi}) + \gamma\hat{v}_L(x_{i+1}^{n,\pi})\big)$.
Step 4  Update the policy: $\pi_{n+1}(x) = \arg\max_{u\in\mathcal{U}}\{C(x,u) + \gamma\hat{v}^n(x^u)\}$.
Step 5  Return the policy $\pi_{N+1}$.
```

4.4 Kernel-based policy iteration with finite horizon approximation

Reference [84] proposes an approximate policy iteration algorithm using kernel smoothing and fixed horizon rewards (KSFHRAPI) with details illustrated in Table 5. The basic idea behind the algorithm is very straightforward. It applies kernel smoothing techniques to finite horizon rewards to approximate policy value function for the infinite horizon problems. More specifically, the infinite horizon post-decision policy value function V^π is approximated with the kth finite horizon policy value function

$$V_k^\pi(x) = \mathrm{E}\big\{\sum_{t=0}^{k}\gamma^t C^\pi(x_t, x_{t+1})|x_0 = x\big\}. \tag{24}$$

Since the problem class is modeled as discounted MDPs, for each $\epsilon > 0$, there exists $k \in \mathbb{N}$ such that for all $x \in \mathcal{X}^\pi$,

$$\begin{aligned}
&|V_k^\pi(x) - V^\pi(x)|\\
&= \big|\mathrm{E}\big\{\sum_{t=k+1}^{\infty}\gamma^t C^\pi(x_t, x_{t+1})|x_0 = x\big\}\big|\\
&\leqslant \frac{\gamma^{k+1}}{1-\gamma}C_{\max} < \epsilon,
\end{aligned}$$

where $C_{\max}$ is an upper bound on the contribution function C^π. As a result, the post-decision policy value function can be approximated arbitrarily close using the finite-horizon counterpart. Then, this algorithm naturally lends itself to different kernel smoothing techniques such as Nadaraya-Watson estimate and local polynomial regression (either recursive or nonrecursive) to estimate the VFA with data structure (x_i, y_i) where

$$y_i = \sum_{\ell=i}^{i+k}\gamma^{\ell-i}C^\pi(x_\ell, x_{\ell+1}). \tag{25}$$

KSFHRAPI is provably convergent in mean. For analysis purpose, the generic form of the kernel smoothing estimates for VFA is considered

$$\hat{m}(x) = \frac{1}{nh^d}\sum_{i=1}^{n}Y_i K(\frac{x - X_i}{h}), \tag{26}$$

where h is a bandwidth and $K : \mathbb{R}^d \to \mathbb{R}$ is a kernel function. This generic form admits Nadaraya-Watson estimators of the regression function and local polynomial estimators with a proper choice of the kernel function K. To show convergence of the algorithm, the kernel smoothing techniques with strong uniform convergence are favored. Due to the dependence structure in the data of MDPs, some technical assumptions, which are listed below, have to be imposed on kernel functions and data structure.

Table 5 Infinite horizon approximate policy iteration algorithm using kernel smoothing and fixed horizon rewards [84].

```
Step 0  Initialization:
        Set the initial policy $\pi_0$ and the iteration counter $n = 0$;
        Set the kernel function $K$ and fixed horizon number $k$.
Step 1  Do for $n = 0, \ldots, N$,
        Set the iteration counter $m = 0$ and the initial state $x_0^n$.
Step 2  Do for $m = 0, \ldots, M$,
        If $m = 0$, do the following:
          Set the initial state $x_0^n$, initial kernel estimate $\hat{f}_{-1}^n = 0$ and $\hat{v}_m = 0$,
          Draw randomly or observe $W_1, \ldots, W_{k+1}$ from the stochastic process,
          Do for $j = 0, \ldots, k$,
            Set $u_j^n = \pi_n(x_j^n)$,
            Compute $x_j^{n,\pi} = S^{\mathrm{M},\pi}(x_j^n, u_j^n)$ and $x_{j+1}^n = S^{\mathrm{M},W}(x_j^{n,\pi}, W_{j+1})$.
        If $m = 1, \ldots, M$, do the following:
          Draw randomly or observe $W_{m+k+1}$ from the process,
          Set $u_{m+k}^n = \pi_n(x_{m+k}^n)$,
          Compute $x_{m+k}^{n,\pi} = S^{\mathrm{M},\pi}(x_{m+k}^n, u_{m+k}^n)$ and $x_{m+k+1}^n = S^{\mathrm{M}}(x_{m+k}^n, u_{m+k}^n, W_{m+k+1})$,
          Compute $u_{m+k+1}^n = \pi_n(x_{m+k+1}^n)$ and $x_{m+k+1}^{n,\pi} = S^{\mathrm{M},\pi}(x_{m+k+1}^n, u_{m+k+1}^n)$.
        Compute $\hat{v}_m = \sum_{j=0}^{k-1} \gamma^j C^\pi(x_{m+j}^{n,\pi}, x_{m+j+1}^{n,\pi})$;
        Apply kernel sparsification approach and compute kernel estimate $\hat{f}_m^n$ with $(x_j^{n,\pi})_{j=0}^m$ and $(\hat{v}_j)_{j=0}^m$ or
        recursively with $\hat{f}_{m-1}^n$, $x_m^{n,\pi}$ and $\hat{v}_m$.
Step 3  Update the policy: $\pi_{n+1}(x) = \arg\max_{u\in\mathcal{U}}\{C(x,u) + \gamma \hat{f}_M^n(x^u)\}$.
Step 4  Return the policy $\pi_{N+1}$.
```

Assumption 7 Let K be a kernel function. $|K(x)| \leqslant K_1$ for all x and $\int_{\mathbb{R}^d} |K(x)|\mathrm{d}x \leqslant K_2 < \infty$. Furthermore, suppose for some $K_3, C < \infty$, either $|K(x)| = 0$ for all $|x| \geqslant K_3$ and $\forall x, x' \in \mathbb{R}^d$,

$$|K(x) - K(x')| \leqslant C\|x - x'\|,$$

or $\frac{\partial}{\partial x}K(x) \leqslant K_3$ and for some $\nu > 1$,

$$\frac{\partial}{\partial x}K(x) \leqslant K_3\|x\|^\nu$$

for $\|x\| > C$.

Assumption 8 The Markov chain X^π is positive Harris that admits a bounded invariant density. Assume the chain is initialized according to its invariant measure and its mixing coefficients $\alpha(n)$ satisfy $\alpha(n) \leqslant An^{-\beta}$ where $A < \infty$ and $\beta > 1 + d + \frac{d}{q}$ for some $q > 0$.

Assumption 9 The Markov chain X^π satisfies the strong Doeblin condition: there exist $n > 1$ and $\rho \in (0,1)$ such that $p_n(y|x) \geqslant \rho f(y)$ where f is the invariant density of the chain and $p_n(y|x)$ is the nth transition density defined as

$$p_n(y|x) = \int_{\mathcal{X}^\pi} p(y|z)p_{n-1}(z|x)\mathrm{d}z \tag{27}$$

for $n = 1, 2, \ldots$. The transition density $p(y|x)$ is $r \geqslant 1$ times differentiable with $\frac{\partial^r}{\partial y^r}p(y|x)$ being uniformly continuous for all x. $\|x\|^q f(y)$ is bounded for some $q \geqslant d$.

Assumption 7 states that the kernel function K is bounded, integrable, and smooth. Assumptions 8 and 9 are alternatives of one another depending on the applications. Assumption 8 is applied to stationary data (the chain can be initialized at their invariant distribution), while Assumption 9 is considered for the chains initialized at some fixed state or some arbitrary distribution (nonstationary data). With the assumptions, it is ready to state the convergence result.

Theorem 7 (Convergence in mean of KSFHRAPI [84]) Suppose for all policy $\pi \in \Pi$, Assumptions 7 and either 8 or 9 hold. Let the data (X, Y) be of the form as in equation (25), $V_k^\pi(x)$ be defined in equation (24) and $\hat{V}_k^\pi$ defined in the same way as $\hat{m}$ in equation (26). Then, $\sup_x |\hat{V}_k^\pi(x) - V_k^\pi(x)| \to 0$ almost surely and Theorem 1 applies to the kernel-based approximate policy iteration algorithm with finite horizon approximation in Table 5.

The situation of sequential sample observations over time arises naturally in MDPs. Therefore, recursive regression estimation is more favorable than nonrecursive smoothing. Moreover, recursive estimates have the advantage of being less computationally demanding with lower memory requirements, since the estimate updates do not involve matrix inversion and are independent of the previous sample size. The KSFHRAPI algorithm is ready to incorporate a variety of recursive kernel smoothing techniques including the Robbins-Monro procedure by [86] and recursive local polynomials by [87].

4.5 Kernel-based policies and exploration

We made the argument that exploration is not a major issue if we use a parametric representation of the value function and if this representation accurately captures the optimal value function. Using kernel regression, we can no longer make the same argument, since kernels allow us to capture functions exactly by only using local information. Observing the value of being in state s tells us nothing about the value of being in state s' if s and s' are far apart.

Our convergence proof using a kernel-based policy appears to again avoid the need for any explicit exploration policy. We accomplished this by assuming that the strong Doeblin condition holds, which is comparable to assuming that a policy is ergodic. We anticipate that this may not hold in practice, and even if it does, we may see slow convergence. The strength of kernel regression, which is its use of local information to build up the approximation, is also its weakness, in that we learn much less about the function from a few observations. Compare this to problems with discrete states, where observations about one state teach us nothing about the value of another state. For this reason, we anticipate that some form of explicit exploration will be needed.

The reinforcement learning community has long used various exploration heuristics (see Chapter 10 in [14]). Perhaps the most popular is epsilon-greedy, where with probability ϵ we choose an action at random (exploration), while with probability $1-\epsilon$ we choose what appears to be the best action. Of course, such strategies cannot be applied to problems with vector-valued, continuous actions.

A common strategy with continuous actions is to simply add a noise term of some sort to what appears to be the optimal action. Ignoring the lack of any theoretical guarantees, such a strategy can become hard to apply to problems with vector-valued controls. The real problem is the sheer size of both the state space and the control space for these problems. If you are going to run an algorithm for 1000 iterations, 1000 random samples of, say, a 100-dimensional state or control space provide very little information locally about a function. Needless to say, research is needed in this area. As of this writing, there is very little in the way of principled approaches to exploration in dynamic programming even for problems with discrete actions.

5 Conclusions

In this paper, we reviewed many stochastic algorithms with continuous VFA from different perspectives of the algorithms: linear and nonlinear approximation, discrete and continuous application, online and offline, on-policy and off-policy, algorithm types (fixed policy, policy iteration, and value iteration), computable expectation, and special problem structures such as linear transition, quadratic rewards, and linear additive noises. Some of the algorithms are provably convergent (in different ways, such as with probability 1, in probability, and in expectation), while others perform nicely in practice without rigorous convergence guarantees.

We also presented several online, on-policy approximate policy iteration algorithms: parametric models with linear architectures and nonparametric approximations using kernel regression. These algorithms are all provably convergent in mean under a variety of technical assumptions. Approximations with linear architectures work fine if they accurately approximate the problem, but they introduce the challenge of choosing the proper features (basis functions). Kernel-based approximations perform better for nonlinear problems but they suffer from scaling problem and may be slow for high-dimensional applications. Hence, kernel sparsification and methods that adjust the importance of different dimensions are necessary to cope with the curse of dimensionality.

References

[1] R. E. Bellman. *Dynamic Programming*. Princeton: Princeton University Press, 1957.

[2] R. A. Howard. *Dynamic Programming and Markov Process*. Cambridge: MIT Press, 1960.

[3] M. L. Puterman. *Markov Decision Processes*. New York: John Wiley & Sons, 1994.

[4] D. P. Bertsekas. *Dynamic Programming and Optimal Control – II*. Belmont, MA: Athena Scientific, 2007.

[5] F. L. Lewis, V. L. Syrmos. *Optimal Control*. Hoboken, NJ: Wiley-Interscience, 1995.

[6] D. P. Bertsekas. Approximate dynamic programming. *Dynamic Programming and Optimal Control*. 3rd ed. Belmont, MA: Athena Scientific, 2010: 321 – 540.

[7] D. Bertsekas, J. Tsitsiklis. *Neuro-dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[8] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs: Prentice Hall, 1999.

[9] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlag, 2001.

[10] Si. J., A. G. Barto, W. B. Powell, et al. *Handbook of Learning and Approximate Dynamic Programming*. Piscataway: Wiley-IEEE Press, 2004.

[11] P. J. Werbos, W. T. I. I. I. Miller, R. S. Sutton, eds. *Neural Networks for Control*. Cambridge: MIT Press, 1990.

[12] D. A. White, D. A. Sofge. *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. New York: Van Nostrand Reinhold Company, 1992.

[13] R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 1998.

[14] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. New York: John Wiley & Sons, 2007.

[15] L. Busoniu, R. Babuska, B. De Schutter, et al. *Reinforcement Learning and Dynamic Programming using Function Approximators*, New York: CRC Press, 2010.

[16] H. Chang, M. Fu, J. Hu, et al. *Simulation-based Algorithms for Markov Decision Processes*. New York: Springer-Verlag, 2007.

[17] X. Cao. *Stochastic Learning and Optimization*, New York: Springer-Verlag, 2007.

[18] A. Gosavi. *Simulation-based Optimization*. Norwell, MA: Kluwer Academic Publishers, 2003.

[19] R. Bellman, S. Dreyfus. Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 1959, 13(68): 247 – 251.

[20] D. Reetz. Approximate solutions of a discounted Markovian decision process. *Bonner Mathematische Schriften*, 1977, 9(8): 77 – 92.

[21] W. Whitt. Approximations of dynamic programs – I. *Mathematics of Operations Research*, 1978, 3(3): 231 – 243.

[22] P. Schweitzer, A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 1985, 110(2): 568 – 582.

[23] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988, 3(1): 9 – 44.

[24] C. Watkins, P. Dayan. Q-learning. *Machine Learning*, 1992, 8(3): 279 – 292.

[25] G. Tesauro. Practical issues in temporal difference learning. *Reinforcement Learning*, 1992, 8(3/4): 257 – 277.

[26] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Reinforcement Learning*, 1992, 8(3/4): 293 – 321.

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

[28] J. Tsitsiklis, B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 1997, 42(5): 674 – 690.

[29] S. Bradtke, A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 1996, 22(1): 33 – 57.

[30] J. Boyan. Least-squares temporal difference learning. *Proceedings of the 16th International Conference on Machine Learning*, San Francisco, CA: Morgan Kaufmann Publishers Inc., 1999: 49 – 56.

[31] M. Lagoudakis, R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 2003, 4(6): 1107 – 1149.

[32] S. Mahadevan, M. Maggioni. Proto-value functions: a Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 2007, 8: 2169 – 2231.

[33] D. Precup, R. Sutton, S. Dasgupta. Off-policy temporal-difference learning with function approximation. *Proceedings of the 18th International Conference on Machine Learning*, San Francisco, CA: Morgan Kaufmann Publishers Inc., 2001: 417 – 424.

[34] R. Sutton, C. Szepesvári, H. Maei. A convergent $\mathrm{O}(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 2009: 1 – 8.

[35] G. Gordon. Stable function approximation in dynamic programming. *Proceedings of the 12th International Conference on Machine Learning*, San Francisco, CA: Morgan Kaufmann Publishers Inc., 1995: 261 – 268.

[36] G. Gordon. Reinforcement learning with function approximation converges to a region. *Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 2001: 1040 – 1046.

[37] G. Rummery, M. Niranjan. *On-line Q-learning using connectionist systems*. Cambridge, U.K.: Engineering Department, Cambridge University, 1994.

[38] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-lever play. *Neural Computation*, 1994, 6(2): 215 – 219.

[39] L. Baird. Residual algorithms: reinforcement learning with function approximation. *Proceedings of the 12th International Conference on Machine Learning*, San Francisco, CA: Morgan Kaufmann Publishers Inc., 1995: 30 – 37.

[40] V. Papavassiliou, S. Russell. Convergence of reinforcement learning with general function approximators. *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, San Francisco, CA: Morgan Kaufmann Publishers Inc., 1999: 748 – 757.

[41] T. Perkins, D. Precup. A convergent form of approximate policy iteration. *Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 2003: 1627 – 1634.

[42] R. Sutton, A. Barto. *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 1998.

[43] I. Szita. *Rewarding Excursions: Extending Reinforcement Learning to Complex Domains*. Budapest, Hungary: Eotvos Lorand University, 2007.

[44] S. Bradtke, B. Ydstie, A. Barto. Adaptive linear quadratic control using policy iteration. *American Control Conference*, New York: IEEE, 1994: 3475 – 3479.

[45] J. N. Tsitsiklis. On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 2002, 3(1): 59 – 72.

[46] S. Bradtke. Reinforcement learning applied to linear quadratic regulation. *Advances in Neural Information Processing Systems*, San Francisco, CA: Morgan Kaufmann Publishers Inc., 1993: 295 – 302.

[47] T. Landelius, H. Knutsson. *Greedy Adaptive Critics for LQR Problems: Convergence Proofs*. Report LiTH-ISY-R-1896. Linköping, Sweden: Computer Vision Laboratory, 1997.

[48] P. Werbos. Approximate dynamic programming for real-time control and neural modeling. *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, New York: Van Nostrand Reinhold, 1992: 493 – 525.

[49] A. Al-Tamimi F. Lewis, M. Abu-Khalaf. Discrete-time nonlinear HJB solution using approximate dynamic programming: convergence proof. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 2008, 38(4): 943 – 949.

[50] A. Al-Tamimi, F. Lewis, M. Abu-Khalaf. Model-free Q-learning designs for linear discrete-time zero-sum games with application to H-infinity control. *Automatica*, 2007, 43(3): 473 – 481.

[51] A. Al-Tamimi, M. Abu-Khalaf, F. Lewis. Adaptive critic designs for discrete-time zero-sum games with application to H_∞ control. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 2007, 37(1): 240 – 247.

[52] A. Antos, C. Szepesvári, R. Munos. Value-iteration based fitted policy iteration: learning with a single trajectory. *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, New York: IEEE, 2007: 330 – 337.

[53] A. Antos, C. Szepesvári, R. Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 2008, 71(1): 89 – 129.

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

[55] A. Antos, R. Munos, C. Szepesvari. Fitted Q-iteration in continuous action-space MDPs. *Advances in Neural Information Processing Systems*, Cambridge: MIT Press, 2008: 9 – 16.

[56] D. Ormoneit, S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 2002, 49(2/3): 161 – 178.

[57] X. Xu, D. Hu, X. Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 2007, 18(4): 973 – 992.

[58] Y. Engel, S. Mannor, R. Meir. The kernel recursive least-squares algorithm. *IEEE Transactions on Signal Processing*, 2004, 52(8): 2275 – 2285.

[59] T. Jung, D. Polani. Least squares SVM for least squares TD learning. *Proceedings of the 17th European Conference on Artificial Intelligence*, Amsterdam, Netherlands: IOS Press, 2006: 499 – 504.

[60] M. Loth, M. Davy, P. Preux. Sparse temporal difference learning using LASSO. *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, New York: IEEE, 2007: 352 – 359.

[61] B. Bethke, J. How, A. Ozdaglar. Approximate dynamic programming using support vector regression. *Proceedings of the 47th IEEE Conference on Decision and Control*, New York: IEEE, 2008: 3811 – 3816.

[62] A. Farahmand, M. Ghavamzadeh, C. Szepesvári, et al. Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems. *American Control Conference*, New York: IEEE, 2009: 725 – 730.

[63] A. Farahmand, M. Ghavamzadeh, C. Szepesvári, et al. Regularized policy iteration. *Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 2008: 441 – 448.

[64] Y. Engel, S. Mannor, R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. *Proceedings of the 20th International Conference on Machine Learning*, Washington D. C., 2003: 154 – 161.

[65] Y. Engel, S. Mannor, R. Meir. Reinforcement learning with Gaussian processes. *Proceedings of the 22nd International Conference on Machine Learning*, New York: ACM, 2005: 201 – 208.

[66] M. P. Deisenroth, J. Peters, C. E. Rasmussen. Approximate dynamic programming with gaussian processes. *American Control Conference*, New York: IEEE, 2008: 4480 – 4485.

[67] S. Meyn. The policy iteration algorithm for average reward Markov decision processes with general state space. *IEEE Transactions on Automatic Control*, 1997, 42(12): 1663 – 1680.

[68] F. Melo, P. Lisboa, M. Ribeiro. Convergence of Q-learning with linear function approximation. *Proceedings of the European Control Conference*, Kos, Greece, 2007: 2671 – 2678.

[69] F. Melo, S. Meyn, M. Ribeiro. An analysis of reinforcement learning with function approximation. *Proceedings of the 25th International Conference on Machine Learning*, New York: ACM, 2008: 664 – 671.

[70] D. Bertsekas. A Counterexample to Temporal Difference Learning. *Neural Computation*, 1995, 7(2): 270 – 279.

[71] J. Boyan, A. Moore. Generalization in reinforcement learning: safely approximating the value function. *Advances in Neural Information Processing Systems* . Cambridge: MIT Press, 1995: 369 – 376.

[72] M. Abu-Khalaf, F. L. Lewis, J. Huang. Policy iterations and the Hamilton-Jacobi-Isaacs equation for H_∞ state feedback control with input saturation. *IEEE Transactions on Automatic Control*, 2006, 51(12): 1989 – 1995.

[73] T. Hanselmann, L. Noakes, A. Zaknich. Continuous-time adaptive critics. *IEEE Transactions on Neural Networks*, 2007, 18(3): 631 – 647.

[74] D. Vrabie, F. Lewis. Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems. *Neural Networks*, 2009, 22(3): 237 – 246.

[75] D. Vrabie, F. Lewis. Generalized policy iteration for continuous-time systems. *Proceedings of the International Joint Conference on Neural Networks*, New York: IEEE, 2009: 2677 – 2684.

[76] D. Vrabie, O. Pastravanu, M. Abu-Khalaf, et al. Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica*, 2009, 45(2): 477 – 484.

[77] S. Balakrishnan, J. Ding, F. Lewis. Issues on stability of ADP feedback controllers for dynamical systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 2008, 38(4): 913 – 917.

[78] B. Van Roy, D. Bertsekas, Y. Lee, et al. A neuro-dynamic programming approach to retailer inventory management. *Proceedings of the 36th IEEE Conference on Decision and Control*, New York: IEEE, 1997: 4052 – 4057.

[79] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses Of Dimensionality*. New York: John Wiley & Sons, 2007.

[80] K. Judd. *Numerical Methods in Economics*. Cambridge: MIT Press, 1998.

[81] S. Meyn, R. Tweedie. *Markov Chains and Stochastic Stability*. New York: Springer-Verlag, 1993.

[82] D. Bertsekas, S. Shreve, *Stochastic Optimal Control: The Discrete-Time Case*. New York: Academic Press. 1978.

[83] J. Ma, W. B. Powell. A convergent recursive least squares approximate policy iteration algorithm for multi-dimensional Markov decision process with continuous state and action spaces. *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, New York: IEEE, 2009: 66 – 73.

[84] J. Ma, W. B. Powell. *Convergence Analysis of Kernel-based On-policy Approximate Policy Iteration Algorithms for Markov Decision Processes with Continuous Multidimensional States and Actions*. Princeton: Department of Operations Research and Financial Engineering, Princeton University, 2010.

[85] G. Kimeldorf, G. Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 1971, 33(1): 82 – 95.

[86] P. Révész, Robbins-Monro procedure in a Hilbert space and its application in the theory of learning processes – I. *Studia Science Mathmatics Hungar*, 1973, 8: 391 – 398.

[87] J. V. Fernández, J. Vilar-Fernández. Recursive estimation of regression functions by local polynomial fitting. *Annals of the Institute of Statistical Mathematics*, 1998, 50(4): 729 – 754.

Warren B. POWELL is a professor in the Department of Operations Research and Financial Engineering at Princeton University, where he has taught since 1981. He is the director of CASTLE Laboratory (www.castlelab.princeton.edu), which specializes in the development of stochastic optimization models and algorithms with applications in transportation and logistics, energy, health, and finance. The author/coauthor of over 160 refereed publications, he is an informs fellow, and the author of Approximate Dynamic Programming: Solving the Curses of Dimensionality (John Wiley & Sons). His primary research interests are in approximate dynamic programming for high-dimensional applications, optimal learning (the efficient collection of information), and their application in energy systems analysis and transportation. He is a recipient of the Wagner Prize and has twice been a finalist in the prestigious Edelman competition. He has also served in a variety of editorial and administrative positions for Informs, including Informs Board of Directors, Area Editor for Operations Research, President of the Transportation Science Section, and numerous prize and administrative committees. E-mail: powell@princeton.edu.

Jun MA is a Ph.D. candidate in the Department of Operations Research and Financial Engineering at Princeton University. E-mail: junma@princeton.edu.