Journal of Machine Learning Research () Submitted ; Published

Kernel-Based Reinforcement Learning Using Bellman Residual Elimination

Brett Bethke BBETHKE@MIT.EDU
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Jonathan P. How JHOW@MIT.EDU
Department of Aeronautics and Astronautics, MIT

Asuman Ozdaglar ASUMAN@MIT.EDU
Department of Electrical Engineering and Computer Science, MIT

Editor:

Abstract

This paper presents a class of new approximate policy iteration algorithms for solving infinite-horizon, discounted Markov decision processes (MDPs) for which a model of the system is available. The algorithms are similar in spirit to Bellman residual minimization methods. However, by exploiting kernel-based regression techniques with nondegenerate kernel functions as the underlying cost-to-go function approximation architecture, the new algorithms are able to explicitly construct cost-to-go solutions for which the Bellman residuals are identically zero at a set of chosen sample states. For this reason, we have named our approach *Bellman residual elimination* (BRE). Since the Bellman residuals are zero at the sample states, our BRE algorithms can be proven to reduce to *exact* policy iteration in the limit of sampling the entire state space. Furthermore, by exploiting knowledge of the model, the BRE algorithms eliminate the need to perform trajectory simulations and therefore do not suffer from simulation noise effects. The theoretical basis of our approach is a pair of reproducing kernel Hilbert spaces corresponding to the cost and Bellman residual function spaces, respectively. By construcing an invertible linear mapping between these spaces, we transform the problem of performing BRE into a simple regression problem in the Bellman residual space. Once the solution in the Bellman residual space is known, the corresponding cost function is found using the linear mapping. This theoretical framework allows any kernel-based regression technique to be used to perform BRE. The main algorithmic results of this paper are two BRE algorithms, BRE(SV) and BRE(GP), which are based on support vector regression and Gaussian process regression, respectively. BRE(SV) is presented first as an illustration of the basic idea behind our approach, and this approach is then extended to develop the more sophisticated BRE(GP). BRE(GP) is a particularly useful algorithm, since it can exploit techniques from Gaussian process regression to automatically learn the kernel parameters (via maximization of the marginal likelihood) and provide error bounds on the solution (via the posterior covariance). Experimental results demonstrate that both BRE(SV) and BRE(GP) produce good policies and cost approximations for a classic reinforcement learning problem.

Keywords: Kernel methods, Reinforcement learning, Approximate dynamic programming, Bellman residual elimination, Support vector regression, Gaussian process regression

1. Introduction

Markov Decision Processes (MDPs) are a powerful framework for addressing problems involving sequential decision making under uncertainty (Bertsekas, 2007; Puterman, 1994). Such problems arise frequently in a number of fields, including engineering, finance, and operations research. It is well-known that MDPs suffer from the curse of dimensionality, which implies that the size of the state space, and therefore the amount of time necessary to compute the optimal policy, increases exponentially rapidly with the size of the problem. The curse of dimensionality renders most MDPs of practical interest very difficult to solve exactly using standard methods such as value iteration or policy iteration. To overcome this challenge, a wide variety of methods for generating approximate solutions to large MDPs have been developed, giving rise to the field of *reinforcement learning* (sometimes also referred to as *approximate dynamic programming* or *neuro-dynamic programming*) (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998).

Approximate policy iteration is a central idea in many reinforcement learning methods. In this approach, an approximation to the cost-to-go vector of a fixed policy is computed; this step is known as *policy evaluation*. Once this approximate cost is known, a *policy improvement* step computes a new, potentially improved policy, and the process is repeated. In many problems, the policy improvement step involves a straightforward minimization over a finite set of possible actions, and therefore can be performed exactly. However, the policy evaluation step is generally more difficult, since it involves solving the fixed-policy Bellman equation:

$$T_\mu J_\mu = J_\mu. \tag{1}$$

Here, J_μ represents the cost-to-go vector of the policy μ, and T_μ is the fixed-policy dynamic programming operator (these objects will be fully explained in the next section). Eq. (1) is a linear system of dimension $|\mathcal{S}|$, where $|\mathcal{S}|$ denotes the size of the state space $\mathcal{S}$. Because $|\mathcal{S}|$ is typically very large, solving Eq. (1) exactly is impractical, and an alternative approach must be taken to generate an approximate solution. Much of the research done in reinforcement learning focuses on how to generate these approximate solutions, which will be denoted in this paper by $\widetilde{J}_\mu$.

The accuracy of an approximate solution $\widetilde{J}_\mu$ generated by a reinforcement learning algorithm is important to the ultimate performance achieved by the algorithm. A natural criterion for evaluating solution accuracy in this context is the *Bellman error* BE:

$$BE \equiv ||\widetilde{J}_\mu - T_\mu \widetilde{J}_\mu|| = \left(\sum_{i \in \mathcal{S}} |\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)|^2 \right)^{1/2}. \tag{2}$$

The individual terms

$$\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)$$

which appear in the sum are referred to as the *Bellman residuals*. Designing a reinforcement learning algorithm that attempts to minimize the Bellman error over a set of candidate cost solutions is a sensible approach, since achieving an error of zero immediately implies that the exact solution has been found. However, it is difficult to carry out this minimization directly, since evaluation of Eq. (2) requires that the Bellman residuals for every state in the state space be computed. To overcome this difficulty, a common approach is to

generate a smaller set of representative sample states $\widetilde{\mathcal{S}}$ (using simulations of the system, prior knowledge about the important states in the system, or other means) and work with an approximation to the Bellman error $\widetilde{BE}$ obtained by summing Bellman residuals over only the sample states: (Bertsekas and Tsitsiklis, 1996, Ch. 6):

$$\widetilde{BE} \equiv \left(\sum_{i \in \widetilde{\mathcal{S}}} |\widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i)|^2 \right)^{1/2}. \tag{3}$$

It is then practical to minimize $\widetilde{BE}$ over the set of candidate functions. This approach has been investigated by several authors, including (Schweitzer and Seidman, 1985; Baird, 1995; Munos and Szepesvári, 2008; Antos et al., 2008), resulting in a class of reinforcement learning algorithms known as *Bellman residual methods*.

The set of candidate functions is usually referred to as a *function approximation architecture*, and the choice of architecture is an important issue in the design of any reinforcement learning algorithm. Numerous approximation architectures, such as neural networks (Hornik et al., 1989; Bishop, 1995; Haykin, 1994; Tesauro, 1995, 1992), linear architectures (Lagoudakis and Parr, 2003; Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007; Si and Wunsch, 2004; Bertsekas and Ioffe, 1996; de Farias and Van Roy, 2003; Valenti, 2007), splines (Trick and Zin, 1997), and wavelets (Maggioni and Mahadevan, 2006; Mahadevan and Maggioni, 2005) have been investigated for use in reinforcement learning. Recently, motivated by the success of kernel-based methods such as support vector machines (Burges, 1998; Christianini and Shawe-Taylor, 2000; Smola and Schölkopf, 2004) and Gaussian processes (Candela and Rasmussen, 2005; Rasmussen and Williams, 2006) for pattern classification and regression, researchers have begun applying these powerful techniques in the reinforcement learning domain. Dietterich and Wang investigated a kernelized form of the linear programming approach to dynamic programming (Dietterich and Wang, 2001). Ormoniet and Sen presented a model-free approach for doing approximate value iteration using kernel smoothers, under some restrictions on the structure of the state space (Ormoneit and Sen, 2002). Using Gaussian processes, Rasmussen and Kuss derived analytic expressions for the approximate cost-to-go of a fixed policy, in the case where the system state variables are restricted to evolve independently according to Gaussian probability transition functions (Rasmussen and Kuss, 2004). Engel, Mannor, and Meir applied a Gaussian process approximation architecture to TD learning (Engel et al., 2003, 2005; Engel, 2005), and Reisinger, Stone, and Miikkulainen subsequently adapted this framework to allow the kernel parameters to be estimated online (Reisinger et al., 2008). Tobias and Daniel proposed a LSTD approach based on support vector machines (Tobias and Daniel, 2006). Several researchers have investigated designing specialized kernels that exploit manifold structure in the state space (Sugiyama et al., 2006, 2007; Mahadevan, 2005; Mahadevan and Maggioni, 2005; Belkin and Niyogi, 2005, 2004; Smart, 2004). Deisenroth, Jan, and Rasmussen used Gaussian processes in an approximate value iteration algorithm for computing the cost-to-go function (Deisenroth et al., 2008). Similar to the well-studied class of linear architectures, kernel-based architectures map an input pattern into a set of features; however, unlike linear architectures, the effective feature vector of a kernel-based architecture may be infinite-dimensional. This property gives kernel methods a great deal of flexibility and makes them particularly

appropriate in reinforcement learning, where the structure of the cost function may not be well understood.

The focus of this paper is on the development of a new class of kernel-based reinforcement learning algorithms that are similar in spirit to traditional Bellman residual methods. Similar to traditional methods, the new algorithms are designed to minimize an approximate form of the Bellman error as given in Eq. (3). The motivation behind our work is the observation that, given the approximate Bellman error $\widetilde{BE}$ as the objective function to be minimized, we should seek to find a solution for which the objective is identically zero, the smallest possible value. The ability to find such a solution depends on the richness of the function approximation architecture employed, which in turn defines the set of candidate solutions. Traditional, parametric approximation architectures such as neural networks and linear combinations of basis functions are finite-dimensional, and therefore it may not always be possible to find a solution satisfying $\widetilde{BE} = 0$ (indeed, if a poor network topology or set of basis functions is chosen, the minimum achievable error may be large). In contrast, in this paper we shall show that by exploiting the richness and flexibility of kernel-based approximation architectures, it is possible to construct algorithms that always produce a solution for which $\widetilde{BE} = 0$. As an immediate consequence, our algorithms have the desirable property of reducing to *exact* policy iteration in the limit of sampling the entire state space, since in this limit, the Bellman residuals are zero everywhere, and therefore the obtained cost function is exact ($\widetilde{J}_\mu = J_\mu$). Furthermore, by exploiting knowledge of the system model (we assume this model is available), the algorithms eliminate the need to perform trajectory simulations and therefore do not suffer from simulation noise effects. We refer to our approach as *Bellman residual elimination* (BRE), rather than Bellman residual minimization, to emphasize the fact that the error is explicitly forced to zero. To the best of our knowledge, this work is the first to propose Bellman residual elimination as an approach to reinforcement learning.

The theoretical basis of our approach relies on the idea of Reproducing Kernel Hilbert Spaces (RKHSs) (Aronszajn, 1950; Berg et al., 1984; Girosi, 1998; Saitoh, 1988; Schölkopf, 1997; Schölkopf and Smola, 2002; Wahba, 1990), which are a specialized type of function space especially useful in the analysis of kernel methods. Specifically, the paper presents two related RKHSs, one corresponding to the space of candidate cost-to-go functions, and one corresponding to the space of Bellman residual functions. We show how an invertible linear mapping between these spaces can be constructed, using properties of the Bellman equation and the assumption that the kernel is non-degenerate. This mapping is useful because the desired property $\widetilde{BE} = 0$ is naturally encoded as a simple regression problem in the Bellman residual space, allowing the construction of algorithms that find a solution with this property in the Bellman residual space. Any kernel-based regression technique, such as support vector regression or Gaussian process regression, can be used to solve the regression problem, resulting in a class of related BRE algorithms. Once the solution is known, the linear mapping is used to find the corresponding cost-to-go function. The use of a nondegenerate kernel function (that is, one with an infinite-dimensional feature vector) is key to the success of this approach, since the linear mapping is not always invertible when using a finite-dimensional architecture such as those used in (Lagoudakis and Parr, 2003; Bertsekas and Ioffe, 1996; de Farias and Van Roy, 2003).

The organization of this paper is as follows. Section 2 introduces basic concepts and notation for MDPs, kernel regression, and RKHSs. Section 3 develops a BRE algorithm based on support vector regression, which we call BRE(SV), to illustrate the basic idea behind our approach. Section 4 then develops a general, theoretical BRE framework that can utilize any kernel-based regression technique to perform BRE, and presents proofs showing that the Bellman residual is always zero at the sampled states as claimed. In Section 5, we use this framework to derive BRE(GP), a BRE algorithm based on Gaussian process regression. BRE(GP) is a particularly useful BRE algorithm, since it can exploit well-known techniques from Gaussian process regression to automatically learn the kernel parameters (via maximization of the marginal likelihood) and provide error bounds on the solution (via the posterior covariance). Finally, we present experimental results demonstrating BRE(SV) and BRE(GP) on several test problems. Section 6 applies BRE(SV) and BRE(GP) to several test problems to illustrate the algorithms' performance. Finally, Section 7 presents concluding remarks.

2. Background

2.1 Markov Decision Processes

An infinite horizon, discounted, finite state MDP is specified by $(\mathcal{S}, \mathcal{A}, P, g)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space (assumed to be finite), P is the system dynamics model where $P_{ij}(u)$ gives the transition probability from state i to state j under action u, and $g(i, u)$ gives the immediate cost of taking action u in state i. Future costs are discounted by a factor $0 < \alpha < 1$. A policy of the MDP is denoted by $\mu : \mathcal{S} \to \mathcal{A}$. Given the MDP specification, the problem is to minimize the so-called cost-to-go function $J_\mu : \mathcal{S} \to \mathbb{R}$ over the set of admissible policies Π:

$$\min_{\mu \in \Pi} J_\mu(i_0) = \min_{\mu \in \Pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k g(i_k, \mu(i_k))\right]. \tag{4}$$

Here, $i_0 \in \mathcal{S}$ is an initial state and the expectation $\mathbb{E}$ is taken over the possible future states $\{i_1, i_2, \dots\}$, given i_0 and the policy μ.

In solving the MDP, the primary object of interest is the policy $\mu^\star$ which achieves the minimum in (4). The optimal cost associated with $\mu^\star$, denoted by $J^\star \equiv J_{\mu^\star}$, satisfies the Bellman equation (Bertsekas, 2007)

$$J^\star(i) = \min_{u \in \mathcal{A}} \left(g(i, u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J^\star(j) \right) \quad \forall i \in \mathcal{S}. \tag{5}$$

It is customary to define the *dynamic programmig operator* T as

$$(TJ)(i) \equiv \min_{u \in \mathcal{A}} \left(g(i, u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J(j) \right),$$

so that Eq. (5) can be written compactly as

$$J^\star = TJ^\star.$$

If $J^\star$ can be found by solving Eq. (5), then the optimal policy $\mu^\star$ is given by

$$\mu^\star(i) = \arg\min_{u \in \mathcal{A}} \left(g(i,u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J^\star(j) \right). \tag{6}$$

Eq. (6) establishes a relationship between the policy and its associated cost function. In this paper, we assume that the minimization in Eq. (6) can be performed exactly since $\mathcal{A}$ is a finite set. Therefore, the bulk of the work in solving an MDP involves computing the optimal cost function by solving the nonlinear system of equations given in (5).

One approach to computing the optimal cost is to solve Eq. (5) in an iterative fashion using value iteration. An alternative approach arises from the observation that if a policy μ is fixed, then the nonlinear system Eq. (5) reduces to a linear system which is easier to solve:

$$J_\mu(i) = g(i,\mu(i)) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(\mu(i)) J_\mu(j) \quad \forall i \in \mathcal{S}. \tag{7}$$

For notational convenience, the fixed-policy cost and state transition functions are defined as

$$g_i^\mu \equiv g(i,\mu(i)) \tag{8}$$
$$P_{ij}^\mu \equiv P_{ij}(\mu(i)). \tag{9}$$

In vector-matrix notation, Eq. (7) can also be expressed as

$$\underline{J}_\mu = (I - \alpha P^\mu)^{-1} \underline{g}^\mu, \tag{10}$$

where $\underline{g}^\mu$ is the vector of immediate costs $g(i,\mu(i))$ over the state space $\mathcal{S}$, and similarly, $\underline{J}_\mu$ is the vector of cost-to-go values over $\mathcal{S}$.

We define the *fixed-policy dynamic programming operator* T_μ as

$$(T_\mu J)(i) \equiv g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu J(j),$$

so that Eq. (7) can be written compactly as

$$J_\mu = T_\mu J_\mu.$$

Solving Eq. (7) is known as *policy evaluation*. The solution J_μ is the cost-to-go of the fixed policy μ. Once the policy's cost-to-go function J_μ is known, a new, better policy μ' can be constructed by performing a *policy improvement* step:

$$\mu'(i) = \arg\min_{u \in \mathcal{A}} \left(g(i,u) + \alpha \sum_{j \in \mathcal{S}} P_{ij}(u) J_\mu(j) \right).$$

By iteratively performing policy evaluation followed by policy improvement, a sequence of policies that are guaranteed to converge to the optimal policy $\mu^\star$ is obtained (Bertsekas, 2007). This procedure is known as *policy iteration*.

2.2 Support Vector Regression

This section provides a brief overview of support vector regression (SVR); for more details, see (Smola and Schölkopf, 2004). The objective of the SVR problem is to learn a function $f(x)$ of the form

$$f(x) = \sum_{l=1}^{r} \theta_l \phi_l(x) = \langle \underline{\Theta}, \underline{\Phi}(x) \rangle \tag{11}$$

that gives a good approximation to a given set of training data

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

where $x_i \in \mathbb{R}^m$ is the input data and $y_i \in \mathbb{R}$ is the observed output. The vector

$$\underline{\Phi}(x) = (\phi_1(x) \ \ldots \ \phi_r(x))^T$$

is referred to as the *feature vector* of the point x, where each *feature* (also called a *basis function*) $\phi_i(x)$ is a scalar-valued function of x. The vector

$$\underline{\Theta} = (\theta_1 \ \ldots \ \theta_r)^T$$

is referred to as the *weight vector*. The notation $\langle \cdot, \cdot \rangle$ is used to denote the standard inner product.

The training problem is posed as the following quadratic optimization problem:

$$\begin{aligned}
\min_{\underline{\Theta}, \underline{\xi}, \underline{\xi}^\star} \quad & \frac{1}{2}||\underline{\Theta}||^2 + c\sum_{i=1}^{n}(\xi_i + \xi_i^\star) && (12)\\
s.t. \quad y_i - \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle &\leq \epsilon + \xi_i && (13)\\
-y_i + \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle &\leq \epsilon + \xi_i^\star && (14)\\
\xi_i, \xi_i^\star &\geq 0 \qquad \forall i \in \{1, \ldots, n\}. && (15)
\end{aligned}$$

Here, the regularization term $\frac{1}{2}||\underline{\Theta}||^2$ penalizes model complexity, and the $\xi_i, \xi_i^\star$ are slack variables which are active whenever a training point y_i lies farther than a distance ϵ from the approximating function $f(x_i)$, giving rise to the so-called *ϵ-insensitive loss function*. The parameter c trades off model complexity with accuracy of fitting the observed training data. As c increases, any data points for which the slack variables are active incur higher cost, so the optimization problem tends to fit the data more closely (note that fitting too closely may not be desired if the training data is noisy).

The minimization problem [Eqs. (12)-(15)] is difficult to solve when the number of features r is large, for two reasons. First, it is computationally demanding to compute the values of all r features for each of the data points. Second, the number of decision variables in the problem is $r+2n$ (since there is one weight element θ_i for each basis function $\phi_i(\cdot)$ and two slack variables $\xi_i, \xi_i^\star$ for each training point), so the minimization must be carried out in an $(r+2n)$-dimensional space. To address these issues, one can solve the primal problem through its dual, which can be formulated by computing the Lagrangian and minimizing

with respect to the primal variables $\underline{\Theta}$ and $\underline{\xi}, \underline{\xi}_i^\star$ (again, for more details, see (Smola and Schölkopf, 2004)). The dual problem is

$$\max_{\underline{\lambda},\underline{\lambda}^\star} \quad -\frac{1}{2}\sum_{i,i'=1}^{n}(\lambda_i^\star - \lambda_i)(\lambda_{i'}^\star - \lambda_{i'})\langle\underline{\Phi}(x_i),\underline{\Phi}(x_{i'})\rangle - \epsilon\sum_{i=1}^{n}(\lambda_i^\star + \lambda_i) + \sum_{i=1}^{n} y_i(\lambda_i^\star - \lambda_i) \tag{16}$$

$$s.t. \quad 0 \leq \lambda_i, \lambda_i^\star \leq c \quad \forall i \in \{1,\ldots,n\}. \tag{17}$$

Note that the feature vectors $\underline{\Phi}(x_i)$ now enter into the optimization problem only as inner products. This is important, because it allows a *kernel function* $k(x_i, x_{i'}) = \langle\underline{\Phi}(x_i),\underline{\Phi}(x_{i'})\rangle$ to be defined whose evaluation may avoid the need to explicitly calculate the vectors $\underline{\Phi}(x_i)$, resulting in significant computational savings. Also, the dimensionality of the dual problem is reduced to only $2n$ decision variables, since there is one λ_i and one $\lambda_i^\star$ for each of the training points. When the number of features is large, this results in significant computational savings. Furthermore, it is well known that the dual problem can be solved efficiently using special-purpose techniques such as Sequential Minimal Optimization (SMO) (Platt, 1999; Keerthi et al., 1999). Once the dual variables are known, the weight vector is given by

$$\underline{\Theta} = \sum_{i=1}^{n}(\lambda_i - \lambda_i^\star)\underline{\Phi}(x_i), \tag{18}$$

and the function $f(x)$ can be computed using the so-called *support vector expansion*:

$$\begin{aligned} f(x) &= \langle\underline{\Theta},\underline{\Phi}(x)\rangle \\ &= \sum_{i=1}^{n}(\lambda_i - \lambda_i^\star)\langle\underline{\Phi}(x_i),\underline{\Phi}(x)\rangle \\ &= \sum_{i=1}^{n}(\lambda_i - \lambda_i^\star)k(x_i, x). \end{aligned} \tag{19}$$

2.3 Gaussian Process Regression

In this section, another kernel-based regression technique, Gaussian process regression (Rasmussen and Williams, 2006), is reviewed. Gaussian process regression attempts to solve the same problem as support vector regression: given a set of training data $\mathcal{D}$, find a function $f(x)$ that provides a good approximation to the data. Gaussian process regression approaches this problem by defining a probability distribution over a set of admissible functions and performing Bayesian inference over this set. A Gaussian process is defined as a (possible infinite) collection of random variables, any finite set of which is described by a joint Gaussian distribution. The process is therefore completely specified by a mean function

$$m(x) = \mathbb{E}[f(x)]$$

and positive semidefinite covariance (kernel) function

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))].$$

The Gaussian process is denoted by

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

For the purposes of regression, the random variables of a Gaussian process $\mathcal{GP}(m(x), k(x, x'))$ are interpreted as the function values $f(x)$ at particular values of the input x. Note the important fact that given any finite set of input points $\mathcal{X} = \{x_1, \ldots, x_n\}$, the distribution over the corresponding output variables $\{y_1, \ldots, y_n\}$ is given by a standard Gaussian distribution

$$(y_1, \ldots, y_n)^T \sim \mathcal{N}(\underline{\mu}, \Sigma),$$

where the mean $\underline{\mu}$ and covariance Σ of the distribution are obtained by "sampling" the mean $m(x)$ and covariance $k(x, x')$ functions of the Gaussian process at the points $\mathcal{X}$:

$$\begin{aligned} \underline{\mu} &= (m(x_1), \ldots, m(x_n))^T \\ \Sigma &= \mathbb{K}(\mathcal{X}, \mathcal{X}) = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \end{aligned}$$

Here, $\mathbb{K}(\mathcal{X}, \mathcal{X})$ denotes the $n \times n$ Gram matrix of the kernel $k(x, x')$ evaluated for the points $\mathcal{X}$.

The Gaussian process $\mathcal{GP}(m(x), k(x, x'))$ represents a prior distribution over functions. To perform regression, the training data $\mathcal{D}$ must be incorporated into the Gaussian process to form a posterior distribution, such that every function in the support of the posterior agrees with the observed data. From a probabilistic standpoint, this amounts to conditioning the prior on the data. Fortunately, since the prior is a Gaussian distribution, the conditioning operation can be computed analytically. To be precise, assume that we wish to know the value of the function f at a set of points $\mathcal{X}_\star = \{x_1^\star, \ldots, x_l^\star\}$, conditioned on the training data. Denote the vector $(y_1, \ldots, y_n)^T$ by $\underline{y}$, $(f(x_1), \ldots, f(x_n))^T$ by $\underline{f}(\mathcal{X})$, and $(f(x_1^\star), \ldots, f(x_l^\star))^T$ by $\underline{f}(\mathcal{X}_\star)$. Now, using the definition of the Gaussian process, the joint prior over the outputs is

$$\begin{pmatrix} \underline{f}(\mathcal{X}) \\ \underline{f}(\mathcal{X}_\star) \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} \mathbb{K}(\mathcal{X}, \mathcal{X}) & \mathbb{K}(\mathcal{X}, \mathcal{X}_\star) \\ \mathbb{K}(\mathcal{X}_\star, \mathcal{X}) & \mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star) \end{pmatrix}\right),$$

where $\mathbb{K}(\mathcal{X}, \mathcal{X}_\star)$ denotes the $n \times l$ Gram matrix of covariances between all pairs of training and test points (the other matrices $\mathbb{K}(\mathcal{X}, \mathcal{X})$, $\mathbb{K}(\mathcal{X}_\star, \mathcal{X})$, and $\mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star)$ are defined similarly).

Conditioning this joint prior on the data yields (Rasmussen and Williams, 2006, A.2)

$$\underline{f}(\mathcal{X}_\star) \mid \left(\mathcal{X}_\star, \mathcal{X}, \underline{f}(\mathcal{X}) = \underline{y}\right) \sim \mathcal{N}(\underline{\mu}_{posterior}, \Sigma_{posterior}), \tag{20}$$

where

$$\begin{aligned} \underline{\mu}_{posterior} &= \mathbb{K}(\mathcal{X}_\star, \mathcal{X})\mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}\underline{y}, \\ \Sigma_{posterior} &= \mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star) - \mathbb{K}(\mathcal{X}_\star, \mathcal{X})\mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}\mathbb{K}(\mathcal{X}, \mathcal{X}_\star). \end{aligned}$$

Eq. (20) is a general result that predicts the mean and covariance of the function values at all of the points $\mathcal{X}_\star$. If we wish to query the function at only a single point $x_\star$, Eq. (20) can be simplified. Note that if $|\mathcal{X}_\star| = 1$, then the matrices $\mathbb{K}(\mathcal{X}, \mathcal{X}_\star)$ and $\mathbb{K}(\mathcal{X}_\star, \mathcal{X})$ reduce to a column vector and a row vector, which are denoted by $\underline{k}_\star$ and $\underline{k}_\star^T$, respectively. Similarly,

the matrix $\mathbb{K}(\mathcal{X}_\star, \mathcal{X}_\star)$ reduces to the scalar $k(x_\star, x_\star)$. With this notation, the mean $\bar{f}(x_\star)$ and variance $\mathbb{V}[f(x_\star)]$ of the function value at $x_\star$ can be expressed as

$$\bar{f}(x_\star) = \underline{k}_\star^T \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1} \underline{y} \tag{21}$$

$$\mathbb{V}[f(x_\star)] = k(x_\star, x_\star) - \underline{k}_\star^T \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1} \underline{k}_\star. \tag{22}$$

Defining the vector $\underline{\lambda}$ as

$$\underline{\lambda} = \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1} \underline{y},$$

Eq. (21) also can be written as

$$\bar{f}(x_\star) = \sum_{i=1}^{n} \lambda_i k(x_i, x_\star). \tag{23}$$

Marginal Likelihood and Kernel Parameter Selection

In many cases, the kernel function $k(i, i')$ depends on a set of parameters $\underline{\Omega}$. The notation $k(i, i'; \underline{\Omega})$ and $\mathbb{K}(\mathcal{X}, \mathcal{S}; \underline{\Omega})$ shall be used when we wish to explicitly emphasize dependence of the kernel and its associated Gram matrix on the parameters. Each choice of $\underline{\Omega}$ defines a different model of the data, and not all models perform equally well at explaining the observed data. In Gaussian process regression, there is a simple way to quantify the performance of a given model. This quantity is called the *log marginal likelihood* and is interpreted as the probability of observing the data, given the model. The log marginal likelihood is given by (Rasmussen and Williams, 2006, 5.4)

$$\log p(\underline{y}|\mathcal{X}, \underline{\Omega}) = -\frac{1}{2} \underline{y}^T \mathbb{K}(\mathcal{X}, \mathcal{X}; \underline{\Omega})^{-1} \underline{y} - \frac{1}{2} \log |\mathbb{K}(\mathcal{X}, \mathcal{X}; \underline{\Omega})| - \frac{n}{2} \log 2\pi. \tag{24}$$

The best choice of the kernel parameters $\underline{\Omega}$ are those which give the highest probability of the data; or equivalently, those which maximize the log marginal likelihood [Eq. (24)]. The derivative of the likelihood with respect to the individual parameters Ω_j can be calculated analytically (Rasmussen and Williams, 2006, 5.4):

$$\frac{\partial \log p(\underline{y}|\mathcal{X}, \underline{\Omega})}{\partial \Omega_j} = \frac{1}{2} \mathrm{tr} \left((\underline{\lambda}\underline{\lambda}^T - \mathbb{K}(\mathcal{X}, \mathcal{X})^{-1}) \frac{\partial \mathbb{K}(\mathcal{X}, \mathcal{X})}{\partial \Omega_j} \right). \tag{25}$$

Eq. (25) allows the use of any gradient-based optimization method to select the optimal values for the parameters $\underline{\Omega}$.

2.4 Reproducing Kernel Hilbert Spaces

This section provides a brief overview of kernel functions and the associated theory of Reproducing Kernel Hilbert Spaces. The presentation of some of this material follows the more detailed discussion in (Schölkopf and Smola, 2002, 2.2).

The kernel function $k(\cdot, \cdot)$ plays an important role in any kernel-based learning method. The kernel maps any two elements from a space of input patterns $\mathcal{X}$ to the real numbers,

$$k(\cdot, \cdot) : \mathcal{X} \otimes \mathcal{X} \to \mathbb{R},$$

and can be thought of as a similarity measure on the input space. In this paper, we shall take $\mathcal{X}$ to be the state space $\mathcal{S}$ of the MDP. In the derivation of kernel methods such as support vector regression, the kernel function arises naturally as an inner (dot) product in a high-dimensional feature space. As such, the kernel must satisfy several important properties of an inner product: it must be *symmetric* ($k(x, y) = k(y, x)$) and *positive semi-definite*. The latter property is defined as positive semi-definiteness of the associated Gram matrix $\mathbb{K}$, where

$$\mathbb{K}_{ij} \equiv k(x_i, x_j),$$

for all subsets $\{x_1, \ldots, x_m\} \subset \mathcal{X}$. A kernel that satisfies these properties is said to be *admissible*. In this paper, we shall deal only with admissible kernels. Furthermore, if the associated Gram matrix $\mathbb{K}$ is strictly positive definite for all subsets $\{x_1, \ldots, x_m\} \subset \mathcal{X}$, the kernel is called *nondegenerate*. The use of nondegenerate kernels will play an important role in establishing many of the theoretical results in this paper.

Given a mapping from inputs to the feature space, the corresponding kernel function can be constructed. However, in many cases, it is desirable to avoid explicitly defining the feature mapping and instead specify the kernel function directly. Therefore, it is useful to consider a construction that proceeds in the opposite direction. That is: *given a kernel, we seek to construct a feature mapping such that the kernel can be expressed as an inner product in the corresponding feature space.*

To begin, assume that $\mathcal{X}$ is an arbitrary set of input data. (In later sections, the set $\mathcal{X}$ will be taken as $\mathcal{S}$, the set of states in the MDP). Furthermore, assume that $k(\cdot, \cdot)$ is a symmetric, positive definite kernel which maps two elements in $\mathcal{X}$ to the reals:

$$k(\cdot, \cdot) : \mathcal{X} \otimes \mathcal{X} \to \mathbb{R}$$

Now, define a mapping $\underline{\Phi}$ from $\mathcal{X}$ to the space of real-valued functions over $\mathcal{X}$ as follows:

$$\begin{aligned} \underline{\Phi} : \qquad & \mathcal{X} \to \mathbb{R}^{\mathcal{X}} \\ & \underline{\Phi}(x)(\cdot) = k(\cdot, x). \end{aligned} \tag{26}$$

Using the set of functions $\{\underline{\Phi}(x)(\cdot) \mid x \in \mathcal{X}\}$ as a basis, a real vector space $\mathcal{H}$ can be constructed by taking linear combinations of the form

$$f(\cdot) = \sum_{i=1}^{m} \lambda_i \underline{\Phi}(x_i)(\cdot) = \sum_{i=1}^{m} \lambda_i k(\cdot, x_i), \tag{27}$$

where $m \in \mathbb{N}$, $\lambda_1 \ldots \lambda_m \in \mathbb{R}$, and $x_1, \ldots, x_m \in \mathcal{X}$ are arbitrary. The vector space $\mathcal{H}$ is then given by the set of all such functions $f(\cdot)$:

$$\mathcal{H} = \{f(\cdot) \mid m \in \mathbb{N}, \lambda_1 \ldots \lambda_m \in \mathbb{R}, x_1, \ldots, x_m \in \mathcal{X}\}$$

Furthermore, if

$$g(\cdot) = \sum_{j=1}^{m'} \beta_j k(\cdot, x'_j)$$

is another function in the vector space, an inner product $\langle f(\cdot), g(\cdot)\rangle$ can be defined as

$$\langle f(\cdot), g(\cdot)\rangle = \sum_{i=1}^{m}\sum_{j=1}^{m'} \lambda_i \beta_j k(x_i, x'_j). \tag{28}$$

It is straightforward to show that $\langle \cdot, \cdot \rangle$ satisfies the necessary properties of an inner product (Schölkopf and Smola, 2002, 2.2.2).

The inner product as defined in Eq. (28) has the following important property, which follows immediately from the definition:

$$\langle f(\cdot), k(\cdot, x)\rangle = \sum_{i=1}^{m} \lambda_i k(x, x_i) = f(x). \tag{29}$$

In particular, letting $f(\cdot) = k(\cdot, x')$:

$$\langle k(\cdot, x), k(\cdot, x')\rangle = k(x, x').$$

Substituting Eq. (26),

$$\langle \underline{\Phi}(x)(\cdot), \underline{\Phi}(x')(\cdot)\rangle = k(x, x'). \tag{30}$$

$k(\cdot, \cdot)$ is therefore said to have the *reproducing property* in $\mathcal{H}$, and the mapping $\underline{\Phi}$ is called the *reproducing kernel map*. Notice that the main objective has now been accomplished: starting from the kernel function $k(\cdot, \cdot)$, a feature mapping $\underline{\Phi}(x)$ [Eq. (26)] has been constructed such that the kernel is expressible as an inner product in the feature space $\mathcal{H}$ [Eq. (30)].

An important equivalent characterization of nondegenerate kernels states that *a kernel is nondegenerate if and only if its associated features* $\{\underline{\Phi}(x_1)(\cdot), \ldots, \underline{\Phi}(x_m)(\cdot)\}$ *are linearly independent for all subsets* $\{x_1, \ldots, x_m\} \subset \mathcal{X}$. This property will be important in a number of proofs later in the paper.

RKHS Definition

The feature space $\mathcal{H}$ constructed in the previous section is an inner product (or *pre-Hilbert*) space. $\mathcal{H}$ can be turned into a real Hilbert space by endowing it with the norm $||.||$ associated with the inner product $\langle \cdot, \cdot \rangle$:

$$||f|| = \sqrt{\langle f, f\rangle}.$$

In light of the reproducing property of the kernel in $\mathcal{H}$, the resulting space is called a *Reproducing Kernel Hilbert Space*. A formal definition of an RKHS is as follows:

Definition: Reproducing Kernel Hilbert Space. Let $\mathcal{X}$ be an input set, and $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called a Reproducing Kernel Hilbert Space endowed with an inner product $\langle \cdot, \cdot \rangle$ and norm $|| \cdot ||_{\mathcal{H}}$ if there exists a kernel function $k(\cdot, \cdot) : \mathcal{X} \otimes \mathcal{X} \to \mathbb{R}$ with the following properties:

1. k has the *reproducing property* [Eq. (29)].
2. The set of functions $\{k(\cdot, x) \mid x \in \mathcal{X}\}$ spans $\mathcal{H}$.

It can be shown (Schölkopf and Smola, 2002, 2.2.3) that the RKHS uniquely defines k. The preceding section shows that k also uniquely determines the RKHS (by construction), so there is a one-to-one relationship between the kernel and its corresponding RKHS. As such, we shall sometimes write $\mathcal{H}_k$ to explicitly denote the RKHS corresponding to the kernel k. Notice that every element $f(\cdot) \in \mathcal{H}$, being a linear combination of functions $k(\cdot, x_i)$ [Eq. (27)], can be represented by its expansion coefficients $\{\lambda_i \mid i = 1, \ldots, m\}$ and input elements $\{x_i \mid i = 1, \ldots, m\}$. This representation will be important in our development of the BRE algorithms.

3. BRE Using Support Vector Regression

With the necessary background material established, this section now demonstrates how the basic support vector regression problem can be used to construct a BRE algorithm. We will refer to the resulting algorithm as BRE(SV) and show that it can efficiently solve practical reinforcement learning problems. Furthermore, the key ideas behind BRE(SV) will subsequently be used as a starting point for the development of our generalized class of BRE algorithms later in the paper.

We begin with the following problem statement: assume that an MDP specification $(\mathcal{S}, \mathcal{A}, P, g)$ is given, along with a policy μ of the MDP. Furthermore, assume that a representative set of sample states $\widetilde{\mathcal{S}}$ is known. The goal is to construct an approximate cost-to-go $\widetilde{J}_\mu$ of the policy μ, such that the Bellman residuals at the sample states are identically zero.

Following the standard functional form assumed in the support vector regression problem [Eq. (11)], we express the approximate cost-to-go (at a specified state $i \in \mathcal{S}$) as the inner product between a feature mapping $\underline{\Phi}(i)$ and a set of weights $\underline{\Theta}$:

$$\widetilde{J}_\mu(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \quad i \in \mathcal{S}. \tag{31}$$

The kernel $k(i, i')$ corresponding to the feature mapping $\underline{\Phi}(\cdot)$ is given by

$$k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle, \quad i, i' \in \mathcal{S}. \tag{32}$$

Recall that the Bellman residual at i, $BR(i)$, is defined as

$$\begin{aligned} BR(i) &\equiv \widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i) \\ &= \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \right). \end{aligned}$$

Substituting the functional form of the cost function, Eq. (31), into the expression for $BR(i)$ yields

$$BR(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle - \left(g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \langle \underline{\Theta}, \underline{\Phi}(j) \rangle \right).$$

Finally, by exploiting linearity of the inner product $\langle \cdot, \cdot \rangle$, we can express $BR(i)$ as

$$\begin{aligned} BR(i) &= -g_i^\mu + \langle \underline{\Theta}, \left(\underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \underline{\Phi}(j) \right) \rangle \\ &= -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle, \end{aligned} \tag{33}$$

where $\underline{\Psi}(i)$ is defined as

$$\underline{\Psi}(i) \equiv \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j). \tag{34}$$

$\underline{\Psi}(i)$ represents a new feature mapping that accounts for the structure of the MDP dynamics (in particular, it represents a combination of the features at i and all states j that can be reached in one step from i). The following lemma states an important property of the new feature mapping that will be important in establishing the theoretical properties of our BRE algorithms.

Lemma 1 *Assume the vectors $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$ are linearly independent. Then the vectors $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$, where $\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j)$, are also linearly independent.*

This lemma, as well as all of the following lemmas, theorems and corollaries in the paper, are proved in the appendix. The definition of $\underline{\Psi}(i)$ and the corresponding expression for the Bellman residual [Eq. (33)] now allow the basic support vector regression problem to be modified to find a cost function, of the form Eq. (31), which has small Bellman residuals at the sample states.

3.1 Error Function Substitution

Recall that the support vector regression problem seeks to minimize the absolute value of an ϵ-insensitive loss function, encoded by the constraints Eqs. (13) and (14). In the nominal problem, the error is

$$y_i - \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle,$$

which is just the difference between the observed function value y_i and the predicted value

$$f(x_i) = \langle \underline{\Theta}, \underline{\Phi}(x_i) \rangle.$$

In order to perform BRE over the set of sample states $\widetilde{\mathcal{S}}$, the Bellman residual is first substituted for the error function in the nominal problem:

$$\begin{aligned}
\min_{\underline{\Theta}, \underline{\xi}, \underline{\xi}^{\star}} \quad & \frac{1}{2}||\underline{\Theta}||^2 \quad + \quad c \sum_{i \in \widetilde{\mathcal{S}}} (\xi_i + \xi_i^{\star}) \\
s.t. \quad & BR(i) \quad \leq \quad \epsilon + \xi_i & (35) \\
& -BR(i) \quad \leq \quad \epsilon + \xi_i^{\star} & (36) \\
& \xi_i, \xi_i^{\star} \quad \geq \quad 0 \qquad \forall i \in \widetilde{\mathcal{S}}.
\end{aligned}$$

By introducing the Bellman residual as the error function to be minimized, the optimization process will seek to find a solution such that the residuals are small at the sample states, as desired. If the optimization problem can be solved, the solution yields the values of the weights $\underline{\Theta}$, which in turn uniquely determine the cost function $\widetilde{J}_{\mu}$ through Eq. (31). However, it is not yet clear if this optimization problem still fits the general form of the support vector regression problem [Eqs. (12)-(15)]. To see that it does, the expression for the Bellman residual [Eq. (33)] is substituted in the constraints Eqs. (35)-(36), and the

following optimization problem is obtained:

$$\begin{aligned} \min_{\underline{\Theta},\underline{\xi},\underline{\xi}^\star} \quad & \frac{1}{2}||\underline{\Theta}||^2 \quad + \quad c\sum_{i\in\widetilde{\mathcal{S}}}(\xi_i+\xi_i^\star) && (37)\\ s.t. \quad & -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i)\rangle \quad \leq \quad \epsilon+\xi_i && (38)\\ & g_i^\mu - \langle \underline{\Theta}, \underline{\Psi}(i)\rangle \quad \leq \quad \epsilon+\xi_i^\star && (39)\\ & \xi_i,\xi_i^\star \quad \geq \quad 0 \qquad \forall i\in\widetilde{\mathcal{S}}. && (40) \end{aligned}$$

Eqs. (37)-(40) are referred to as the *Bellman residual minimization problem*. This problem is identical to the basic support vector regression problem [Eqs. (12)-(15)]. The original feature mapping $\underline{\Phi}(i)$ has been replaced by the new feature mapping $\underline{\Psi}(i)$; this is only a notational change and does not alter the structure of the basic problem in any way. Furthermore, the one-stage costs g_i^μ have replaced the generic observations y_i, but again, this is only a notational change. Therefore, the dual problem is given by the normal support vector regression dual [Eqs. (16)-(17)], with the substitutions $\underline{\Phi}(i)\rightarrow\underline{\Psi}(i)$ and $y_i\rightarrow g_i^\mu$:

$$\max_{\underline{\lambda},\underline{\lambda}^\star} \quad -\frac{1}{2}\sum_{i,i'\in\widetilde{\mathcal{S}}}(\lambda_i^\star-\lambda_i)(\lambda_{i'}^\star-\lambda_{i'})\langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle - \epsilon\sum_{i\in\widetilde{\mathcal{S}}}(\lambda_i^\star+\lambda_i)+\sum_{i\in\widetilde{\mathcal{S}}}g_i^\mu(\lambda_i^\star-\lambda_i) \quad (41)$$

$$s.t. \quad 0\leq\lambda_i,\lambda_i^\star\leq c \quad \forall i\in\widetilde{\mathcal{S}}. \quad (42)$$

Notice that the dual of the Bellman residual minimization problem contains a new kernel function, denoted by $\mathcal{K}$:

$$\mathcal{K}(i,i') = \langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle.$$

This kernel is related to the base kernel k [Eq. (32)] through the feature mapping $\underline{\Psi}$ [Eq. (34)]:

$$\begin{aligned} \mathcal{K}(i,i') &= \langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle\\ &= \langle\ \underline{\Phi}(i)-\alpha\sum_{j\in\mathcal{S}}P_{ij}^\mu\underline{\Phi}(j)\ ,\ \underline{\Phi}(i')-\alpha\sum_{j\in\mathcal{S}}P_{i'j}^\mu\underline{\Phi}(j)\ \rangle\\ &= \langle\underline{\Phi}(i),\underline{\Phi}(i')\rangle-\alpha\sum_{j\in\mathcal{S}}\Big(P_{i'j}^\mu\langle\underline{\Phi}(i),\underline{\Phi}(j)\rangle+P_{ij}^\mu\langle\underline{\Phi}(i'),\underline{\Phi}(j)\rangle\Big)+\alpha^2\sum_{j,j'\in\mathcal{S}}P_{ij}^\mu P_{i'j'}^\mu\langle\underline{\Phi}(j),\underline{\Phi}(j')\rangle\\ &= k(i,i')-\alpha\sum_{j\in\mathcal{S}}\Big(P_{i'j}^\mu k(i,j)+P_{ij}^\mu k(i',j)\Big)+\alpha^2\sum_{j,j'\in\mathcal{S}}P_{ij}^\mu P_{i'j'}^\mu k(j,j'). \end{aligned} \quad (43)$$

We shall refer to $\mathcal{K}$ as the *Bellman kernel associated with* k. The following theorem establishes an important property of this kernel.

Theorem 2 *Assume that the kernel* $k(i,i')=\langle\underline{\Phi}(i),\underline{\Phi}(i')\rangle$ *is nondegenerate. Then the associated Bellman kernel defined by* $\mathcal{K}(i,i')=\langle\underline{\Psi}(i),\underline{\Psi}(i')\rangle$, *where* $\underline{\Psi}(i)=\underline{\Phi}(i)-\alpha\sum_{j\in\mathcal{S}}P_{ij}^\mu\underline{\Phi}(j)$, *is also nondegenerate.*

In order to specify and solve the dual problem [Eqs. (41)-(42)], the stage cost values g_i^μ and the kernel values $\mathcal{K}(i,i')$ must be computed. Since by assumption the MDP model is known, the state transition probabilities P_{ij}^μ [Eq. (9)] are available, and thus Eq. (43)

allows direct computation of the kernel values $\mathcal{K}(i,i')$. Furthermore, the g_i^μ values are also available [Eq. (8)]. Thus, all information necessary to formulate and solve the dual problem is known.

Once the kernel values $\mathcal{K}(i,i')$ and the cost values g_i^μ are computed, the dual problem is completely specified and can be solved using, for example, a standard SVM solving package such as libSVM (Chang and Lin, 2001). The solution yields the dual variables $\underline{\lambda}, \underline{\lambda}^\star$. Again, in direct analogy with the normal SV regression problem, the primal variables $\underline{\Theta}$ are given by the support vector expansion [Eq. (18)], with the substitution $\underline{\Phi} \to \underline{\Psi}$:

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} (\lambda_i - \lambda_i^\star) \underline{\Psi}(i).$$

Finally, in order to find the cost $\widetilde{J}_\mu(i)$, the expressions for $\underline{\Theta}$ and $\underline{\Psi}$ are substituted into Eq. (31):

$$\begin{aligned}
\widetilde{J}_\mu(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \langle \underline{\Psi}(i'), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \langle \left(\underline{\Phi}(i') - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu \underline{\Phi}(j) \right), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \left(\langle \underline{\Phi}(i'), \underline{\Phi}(i) \rangle - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu \langle \underline{\Phi}(j), \underline{\Phi}(i) \rangle \right) \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} (\lambda_{i'} - \lambda_{i'}^\star) \left(k(i',i) - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu k(j,i) \right). \qquad (44)
\end{aligned}$$

Thus, once the dual variables are solved for, Eq. (44) can be used to calculate the primary object of interest, the cost-to-go function $\widetilde{J}_\mu(i)$.

This section has shown how the basic support vector regression problem can be modified to find a cost-to-go function whose Bellman residuals are small at the sample states $\widetilde{\mathcal{S}}$. A summary of the procedure is given in Algorithm 1. While this algorithm is directly related to the basic support vector regression problem, it does not yet achieve the goal of explicitly forcing all of the Bellman residuals to zero. Intuitively, this is because the model complexity penalty (controlled by the parameter c) may prevent the optimal solution from achieving the lowest possible value of the Bellman residuals. The next section will explain how to modify Algorithm 1 in order to explicitly force the Bellman residuals to zero.

3.2 Forcing the Bellman Residuals to Zero

The basic support vector regression problem is designed to deal with noisy observations of the true function values. With noisy observations, the problem of overfitting becomes important, and choosing a regression function that matches the observations exactly may not be desirable. The use of the ϵ-insensitive loss function helps to alleviate the overfitting problem, allowing observation points to lie within a distance ϵ of the regression function without

Algorithm 1 Support vector policy evaluation, Bellman residuals not explicitly forced to zero

Input: (μ, $\widetilde{\mathcal{S}}$, k, ϵ, c)
μ: policy to be evaluated
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel function defined on $\mathcal{S} \times \mathcal{S}$
ϵ: width of ϵ-insensitive loss function
c: model complexity parameter
Begin
$\forall i, i' \in \widetilde{\mathcal{S}}$, compute $\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$ using Eq. (43)
$\forall i \in \widetilde{\mathcal{S}}$, compute g_i^μ using Eq. (8)
Using $\mathcal{K}(i, i')$ and g_i^μ values, solve the dual optimization problem Eqs. (41)-(42) for the dual variables $\underline{\lambda}, \underline{\lambda}^\star$
Using the dual solution variables $\underline{\lambda}, \underline{\lambda}^\star$, compute the cost-to-go $\widetilde{J}_\mu(i)$ using Eq. (44)
End

causing the objective function to increase. This loss function leads to good performance with noisy data sets.

However, in Algorithm 1, the "observations" g_i^μ are *exact* values dictated by the structure of the MDP. Therefore, overfitting is not a concern, since we are working directly with the Bellman residuals, which are exact mathematical relationships between the cost-to-go values $J_\mu(i)$ for all states $i \in \mathcal{S}$. Indeed, in order to force the Bellman residuals to zero as desired, the regression function $\langle \underline{\Theta}, \underline{\Psi}(i) \rangle$ must match the "observation" g_i^μ exactly:

$$BR(i) = -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = 0 \quad \Longleftrightarrow \quad \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = g_i^\mu.$$

Fortunately, the support vector regression problem can be modified to perform exact regression, where the learned regression function passes exactly through each of the observation points. This is accomplished by setting $\epsilon = 0$ and $c = \infty$; intuitively, this causes the objective function to be unbounded whenever the regression function does not match an observation. Having fixed c and ϵ, the Bellman residual minimization problem [Eqs. (37)-(40)] can be recast in a simpler form. With $c = \infty$, any feasible solution must have $\xi_i, \xi_i^\star = 0$ for all i, since otherwise the objective function will be unbounded. Furthermore, if ϵ is also zero, then the constraints [Eqs. (38) and (39)] become equalities. With these modifications, the primal problem reduces to

$$\begin{aligned} \min_{\underline{\Theta}} \quad & \frac{1}{2}||\underline{\Theta}||^2 \\ s.t. \quad & g_i^\mu - \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = 0 \qquad \forall i \in \widetilde{\mathcal{S}}, \end{aligned} \tag{45}$$

where $\underline{\Psi}(i)$ is given by Eq. (34). The Lagrangian dual of this optimization problem can be calculated as follows. The Lagrangian is

$$\mathcal{L}(\underline{\Theta}, \underline{\lambda}) = \frac{1}{2}||\underline{\Theta}||^2 + \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \left(g_i^\mu - \langle \underline{\Theta}, \underline{\Psi}(i) \rangle \right). \tag{46}$$

Maximizing $\mathcal{L}(\underline{\Theta}, \underline{\lambda})$ with respect to $\underline{\Theta}$ is accomplished by setting the corresponding partial derivative to zero:

$$\frac{\partial \mathcal{L}}{\partial \underline{\Theta}} = \underline{\Theta} - \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Psi}(i) = 0,$$

and therefore

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Psi}(i). \tag{47}$$

As a result, the cost function $\widetilde{J}_\mu(i)$ [Eq. (31)] is given by

$$\begin{aligned} \widetilde{J}_\mu(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\ &= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \langle \underline{\Psi}(i'), \underline{\Phi}(i) \, \rangle \\ &= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(k(i', i) - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} k(j, i) \right). \end{aligned} \tag{48}$$

Finally, substituting Eq. (47) into Eq. (46) and maximizing with respect to $\underline{\lambda}$ gives the dual problem:

$$\max_{\underline{\lambda}} \; -\frac{1}{2} \sum_{i, i' \in \widetilde{\mathcal{S}}} \lambda_i \lambda_{i'} \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle + \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i g^\mu_i.$$

This can also be written in vector form as

$$\max_{\underline{\lambda}} \; -\frac{1}{2} \underline{\lambda}^T \mathbb{K} \underline{\lambda} + \underline{\lambda}^T \underline{g}^\mu, \tag{49}$$

where

$$\mathbb{K}_{ii'} = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle = \mathcal{K}(i, i')$$

is the Gram matrix of the kernel $\mathcal{K}$ (this matrix is calculated using Eq. (43)). Note that the problem is just an unconstrained maximization of a quadratic form. Since by assumption the base kernel k is nondegenerate, Theorem 2 implies that the associated Bellman kernel $\mathcal{K}$ is nondegenerate also. Therefore, the Gram matrix $\mathbb{K}$ is full-rank and strictly positive definite, so the quadratic form has a unique maximum. The solution is found analytically by setting the derivative of the objective with respect to $\underline{\lambda}$ to zero, which yields

$$\mathbb{K} \underline{\lambda} = \underline{g}^\mu. \tag{50}$$

Note the important fact that the dimension of this linear system is $n_s = |\widetilde{\mathcal{S}}|$, the number of sampled states.

The development of the policy evaluation procedure is now complete and is summarized in Algorithm 2. The key computational step in the algorithm is solving the linear system Eq. (50). Recall that the original problem of solving the full, fixed-policy Bellman equation [Eq. (7)] involves solving an n-dimensional linear system, where $n = |\mathcal{S}|$ is the size of the entire state space. After developing a support vector regression-based method for approximating the cost-to-go and reformulating it to force Bellman residuals to zero, the

Algorithm 2 Support vector policy evaluation, Bellman residuals explicitly forced to zero

Input: $(\mu, \widetilde{\mathcal{S}}, k)$
μ: policy to be evaluated
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel function defined on $\mathcal{S} \times \mathcal{S}$
Begin
Construct the Gram matrix $\mathbb{K}$, where $\mathbb{K}_{ii'} = \mathcal{K}(i, i') \quad \forall i, i' \in \widetilde{\mathcal{S}}$, using Eq. (43)
$\forall i \in \widetilde{\mathcal{S}}$, compute g_i^μ using Eq. (8)
Using $\mathcal{K}(i, i')$ and g_i^μ values, solve the linear system Eq. (50) for $\underline{\lambda}$
Using solution $\underline{\lambda}$, compute the cost-to-go $\widetilde{J}_\mu(i)$ using Eq. (48)
End

approximate policy evaluation problem has been reduced to the problem of solving another linear system [Eq. (50)]. However, the dimensionality of this system is only $n_s = |\widetilde{\mathcal{S}}|$, where $n_s \ll n$. Furthermore, since the designer is in control of $\widetilde{\mathcal{S}}$, he can select an appropriate number of sample states based on the computational resources available.

The following theorem establishes the important claim that the Bellman residuals are exactly zero at the sampled states.

Theorem 3 *Assume that the kernel $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by Algorithm 2 satisfies*

$$\widetilde{J}_\mu(i) = g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \qquad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ are identically equal to zero at every state $i \in \widetilde{\mathcal{S}}$.

An immediate corollary of Theorem 3 follows:

Corollary 4 *Assume that the kernel $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by Algorithm 2 satisfies*

$$\widetilde{J}_\mu(i) = J_\mu(i) \qquad \forall i \in \mathcal{S}.$$

That is, the cost function $\widetilde{J}_\mu(i)$ is exact.

Recovering the Bellman equation

Corollary 4 claims that Algorithm 2 yields the exact cost-to-go when the entire state space is sampled. Recall that the exact cost-to-go can be computed by solving the Bellman equation

$$\underline{J}_\mu = (I - \alpha P^\mu)^{-1} \underline{g}^\mu.$$

Intuitively, therefore, the Bellman equation must be somehow "embedded" into the algorithm. The following calculation shows that this is indeed the case: the Bellman equation can be recovered from Algorithm 2 under a suitable choice of kernel. We begin by taking the kernel function $k(i, i')$ to be the Kronecker delta function:

$$k(i, i') = \delta_{ii'}.$$

This kernel is nondegenerate, since its Gram matrix is always the identity matrix, which is clearly positive definite. Furthermore, we take $\widetilde{\mathcal{S}} = \mathcal{S}$, so that both conditions of Corollary 4 are met. Now, notice that with this choice of kernel, the associated Bellman kernel [Eq. (43)] is

$$\begin{aligned}\mathcal{K}(i,i') &= \delta_{ii'} - \alpha \sum_{j \in \mathcal{S}} \left(P^\mu_{i'j}\delta_{ij} + P^\mu_{ij}\delta_{i'j} \right) + \alpha^2 \sum_{j,j' \in \mathcal{S}} P^\mu_{ij} P^\mu_{i'j'} \delta_{jj'} \\ &= \delta_{ii'} - \alpha \left(P^\mu_{i'i} + P^\mu_{ii'} \right) + \alpha^2 \sum_{j \in \mathcal{S}} P^\mu_{ij} P^\mu_{i'j}. \end{aligned} \tag{51}$$

Line 6 in Algorithm 2 computes the Gram matrix $\mathbb{K}$ of the associated Bellman kernel. A straightforward calculation shows that, with the associated Bellman kernel given by Eq. (51), the Gram matrix is

$$\mathbb{K} = (I - \alpha P^\mu)^2.$$

Line 7 simply constructs the vector $\underline{g}^\mu$, and Line 8 computes $\underline{\lambda}$, which is given by

$$\underline{\lambda} = \mathbb{K}^{-1}\underline{g}^\mu = (I - \alpha P^\mu)^{-2}\underline{g}^\mu.$$

Once $\underline{\lambda}$ is known, the cost function $\widetilde{J}_\mu(i)$ is computed in Line 9:

$$\begin{aligned}\widetilde{J}_\mu(i) &= \sum_{i' \in \mathcal{S}} \lambda_{i'} \left(\delta_{i'i} - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j}\delta_{ji} \right) \\ &= \lambda_i - \alpha \sum_{i' \in \mathcal{S}} \lambda_{i'} P^\mu_{i'i}. \end{aligned}$$

Expressed as a vector, this cost function can now be written as

$$\begin{aligned}\underline{\widetilde{J}}_\mu &= \underline{\lambda} - \alpha P^\mu \underline{\lambda} \\ &= (I - \alpha P^\mu)\underline{\lambda} \\ &= (I - \alpha P^\mu)(I - \alpha P^\mu)^{-2}\underline{g}^\mu \\ &= (I - \alpha P^\mu)^{-1}\underline{g}^\mu \\ &= \underline{J}_\mu. \end{aligned}$$

Thus, we see that for the choice of kernel $k(i,i') = \delta_{ii'}$ and sample states $\widetilde{\mathcal{S}} = \mathcal{S}$, Algorithm 2 effectively reduces to solving the Bellman equation directly. Of course, if the Bellman equation could be solved directly for a particular problem, it would make little sense to use an approximation algorithm of any kind. Furthermore, the Kronecker delta is a poor practical choice for the kernel, since the kernel is zero everywhere except at $i = i'$; we use it here only to illustrate the properties of the algorithm. The real value of the algorithm arises because the main results - namely, that the Bellman residuals are identically zero at the sample states - are valid for *any* nondegenerate kernel function and *any* set of sample states $\widetilde{\mathcal{S}} \subseteq \mathcal{S}$. This permits the use of "smoothing" kernels that enable the computed cost function to generalize to states that were not sampled, which is a major goal of any reinforcement learning algorithm. Thus, in the more realistic case where solving the Bellman equation

Algorithm 3 BRE(SV)

Input: $(\mu_0, \widetilde{\mathcal{S}}, k)$
μ_0: initial policy
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel function defined on $\mathcal{S} \times \mathcal{S}$
Begin
$\mu \leftarrow \mu_0$
loop
  $\widetilde{J}_\mu(i) \leftarrow$ `ALGORITHM_2`$(\mu, \widetilde{\mathcal{S}}, k)$ {Policy evaluation}
  $\mu(i) \leftarrow \arg\min_u \sum_{j \in \mathcal{S}} P_{ij}(u) \left(g(i,u) + \alpha \widetilde{J}_\mu(j) \right)$ {Policy improvement}
end loop
End

directly is not practical, Algorithm 2 (with a smaller set of sample states and a suitable kernel function) becomes a way of effectively reducing the dimensionality of the Bellman equation to a tractable size while preserving the character of the original equation.

The associated Bellman kernel [Eq. (43)] plays a central role in the work developed thus far; it is this kernel and its associated feature mapping $\underline{\Psi}$ [Eq. (34)] that effectively allow the Bellman equation to be embedded into the algorithms, and consequently to ensure that the Bellman residuals at the sampled states are zero. This idea will be developed further in Section 4.

3.3 BRE(SV), A Full Policy Iteration Algorithm

Algorithm 2 shows how to construct a cost-to-go approximation $\widetilde{J}_\mu(\cdot)$ of a fixed policy μ such that the Bellman residuals are exactly zero at the sample states $\widetilde{\mathcal{S}}$. This section now presents BRE(SV) (Algorithm 3), a full policy iteration algorithm that uses Algorithm 2 in the policy evaluation step.

The following corollary to Theorem 3 establishes that BRE(SV) reduces to exact policy iteration when the entire state space is sampled:

Corollary 5 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then BRE(SV) is equivalent to exact policy iteration.*

This result is encouraging, since it is well known that exact policy iteration converges to the optimal policy in a finite number of steps (Bertsekas, 2007).

3.4 Computational Results

BRE(SV) was implemented on the well-known "mountain car problem" (Sutton and Barto, 1998; Rasmussen and Kuss, 2004) to evaluate its performance. In this problem, a unit mass, frictionless car moves along a hilly landscape whose height $H(x)$ is described by

$$H(x) = \begin{cases} x^2 + x & \text{if } x < 0 \\ \frac{x}{\sqrt{1+5x^2}} & \text{if } x \geq 0 \end{cases}.$$

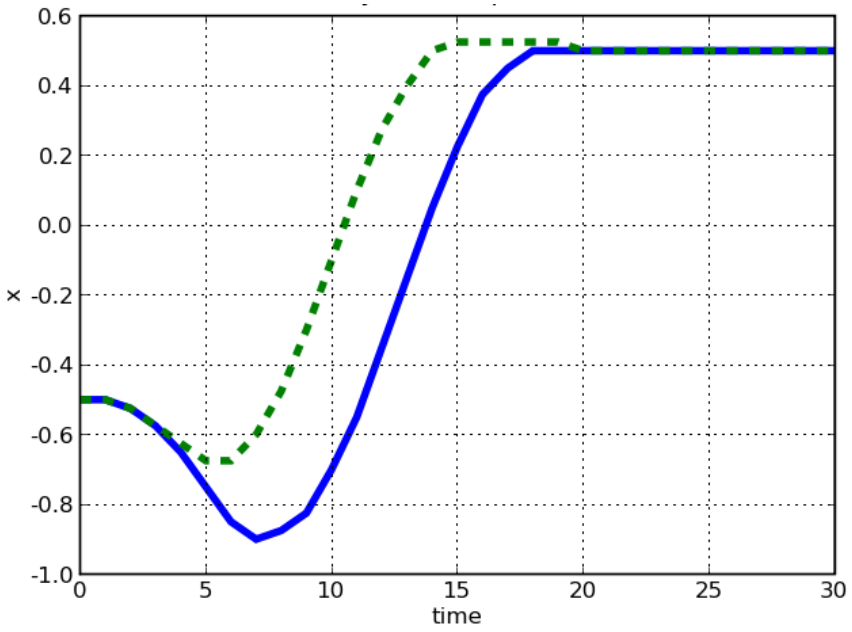

Figure 1: System response under the optimal policy (dashed line) and the policy computed by BRE(SV) after 3 iterations (solid line).

The system state is given by $(x, \dot{x})$ (the horizontal position and velocity of the car). A horizontal control force $-4 \leq u \leq 4$ can be applied to the car, and the goal is to drive the car from its starting location $x = -0.5$ to the "parking area" $0.5 \leq x \leq 0.7$ as quickly as possible. Since the car is also acted upon by gravitational acceleration $g = 9.8$, the total horizontal acceleration of the car, $\ddot{x}$, is given by

$$\ddot{x} = \frac{u - gH'(x)}{1 + H'(x)^2},$$

where $H'(x)$ denotes the derivative $\frac{dH(x)}{dx}$. The problem is challenging because the car is underpowered: it cannot simply drive up the steep slope. Rather, it must use the features of the landscape to build momentum and eventually escape the steep valley centered at $x = -0.5$. The system response under the optimal policy (computed using value iteration) is shown as the dashed line in Figure 1; notice that the car initially moves *away* from the parking area before reaching it at time $t = 14$.

In order to apply the support vector policy iteration algorithm, an evenly spaced 9x9 grid of sample states,

$$\widetilde{\mathcal{S}} = \{(x, \dot{x}) \mid x = -1.0, -0.75, \ldots, 0.75, 1.0 \\ \dot{x} = -2.0, -1.5, \ldots, 1.5, 2.0\}$$

was chosen. Furthermore, a radial basis function kernel, with differing length-scales for the x and $\dot{x}$ axes in the state space, was used:

$$k((x_1, \dot{x}_1), (x_2, \dot{x}_2)) = \exp\left(-(x_1 - x_2)^2/(0.25)^2 - (\dot{x}_1 - \dot{x}_2)^2/(0.40)^2\right).$$

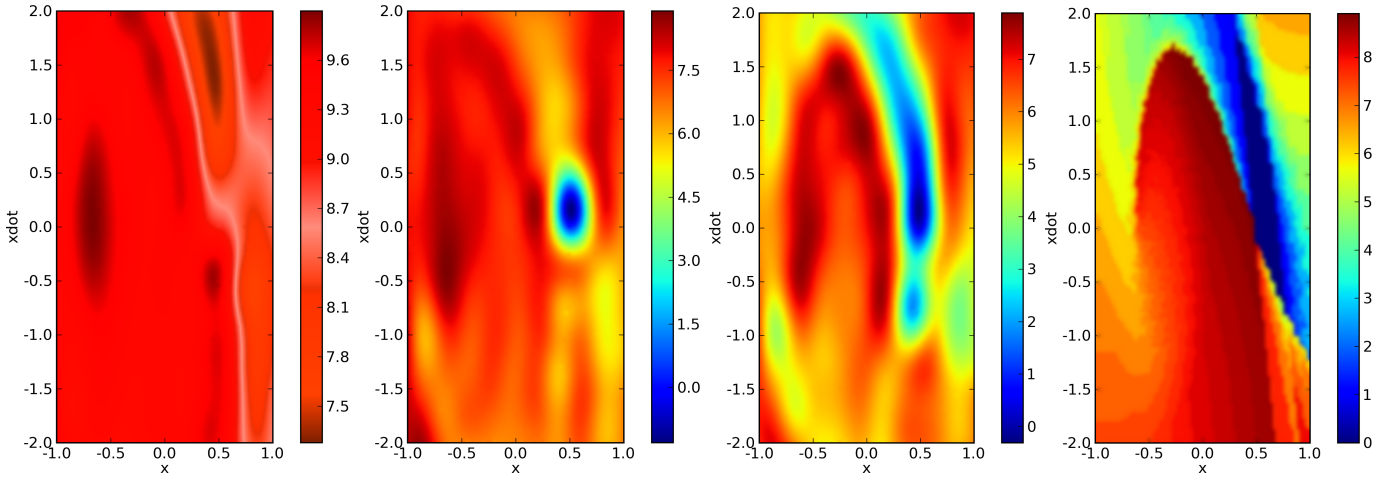

Figure 2: From left to right: Approximate cost-to-go computed by BRE(SV) for iterations 1, 2 , and 3; exact cost-to-go computed using value iteration.

BRE(SV) was executed, resulting in a sequence of policies (and associated cost functions) that converged after three iterations. The sequence of cost functions is shown in Figure 2 along with the optimal cost function for comparison. Of course, the main objective is to learn a policy that is similar to the optimal one. The solid line in Figure 1 shows the system response under the approximate policy generated by the algorithm after 3 iterations. Notice that the qualitative behavior is the same as the optimal policy; that is, the car first accelerates away from the parking area to gain momentum. The approximate policy arrives at the parking area at $t = 17$, only 3 time steps slower than the optimal policy.

BRE(SV) was derived starting from the viewpoint of the standard support vector regression framework. The theorems of this section have established several advantageous theoretical properties of the approach, and furthermore, the results from the mountain car example problem demonstrate that BRE(SV) produces a high-quality cost approximation and policy. The example problem highlights two interesting questions:

1. The scaling parameters (0.25 and 0.40) used in the squared exponential kernel were found, in this case, by carrying out numerical experiments to see which values fit the data best. Is it possible to design a BRE algorithm that can learn these values automatically as it runs?

2. Is it possible to provide error bounds on the learned cost-to-go function for the non-sampled states?

Fortunately, the answer to both of these questions is yes. In the next section, we show how the key ideas in BRE(SV) can be extended to a more general framework, which allows BRE to be performed using any kernel-based regression technique. This analysis will result in a deeper understanding of the approach and also enable the development of a BRE algorithm based on Gaussian process regression, BRE(GP), that preserves all the advantages of BRE(SV) while addressing these two questions.

4. A General BRE Framework

Building on the ideas and intuition presented in the development of BRE(SV), we now seek to understand how that algorithm can be generalized. To begin, it is useful to view kernel-based regression techniques, such as support vector regression and Gaussian process regression, as algorithms that search over a reproducing kernel Hilbert space for a regression function solution. In particular, given a kernel k and a set of training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, these kernel-based regression techniques find a solution of the standard form

$$f(x) = \langle \underline{\Theta}, \underline{\Phi}(x) \rangle,$$

where $\underline{\Phi}$ is the feature mapping of the kernel k. The solution f belongs to the RKHS $\mathcal{H}_k$ of the kernel k:

$$f(\cdot) \in \mathcal{H}_k.$$

Broadly speaking, the various kernel-based regression techniques are differentiated by how they choose the weighting element $\underline{\Theta}$, which in turn uniquely specifies the solution f from the space of functions $\mathcal{H}_k$.

In the BRE(SV) algorithm developed in the previous section, the kernel k and its associated Bellman kernel $\mathcal{K}$ play a central role. Recall the relationship between these two kernels:

$$k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle \tag{52}$$

$$\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P^{\mu}_{ij} \underline{\Phi}(j) \tag{53}$$

$$\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle. \tag{54}$$

By the uniqueness property of Section 2.4, each of the kernels is associated with its own, unique RHKS, denoted by $\mathcal{H}_k$ and $\mathcal{H}_{\mathcal{K}}$, respectively. In this section, we explore some properties of these RKHSs, and use the properties to develop a general BRE framework. In particular, it is shown that $\mathcal{H}_k$ corresponds to the space of possible cost-to-go functions, while $\mathcal{H}_{\mathcal{K}}$ corresponds to the space of Bellman residual functions. We explain how the problem of performing BRE is equivalent to solving a simple regression problem in $\mathcal{H}_{\mathcal{K}}$ in order to find a Bellman residual function with the desired property of being zero at the sampled states $\widetilde{\mathcal{S}}$. This regression problem can be solved using any kernel-based regression technique. Furthermore, an invertible linear mapping between elements of $\mathcal{H}_k$ and $\mathcal{H}_{\mathcal{K}}$ is constructed. This mapping can be used to find the corresponding cost-to-go function once the Bellman residual function is found.

4.1 The Cost Space $\mathcal{H}_k$

The goal of any approximate policy iteration algorithm, including our BRE methods, is to learn an approximation $\widetilde{J}_{\mu}(\cdot)$ to the true cost-to-go function $J_{\mu}(\cdot)$. As discussed, if a kernel-based regression algorithm, with kernel $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$, is used to find the approximate cost function $\widetilde{J}_{\mu}(\cdot)$, then this function is an element of $\mathcal{H}_k$ and takes the standard form

$$\widetilde{J}_{\mu}(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle.$$

1. Using the reproducing kernel map,

$$\underline{\Phi}(i)(\cdot) = k(\cdot, i),$$

the kernel $k(\cdot, \cdot)$ is represented as

$$k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle \tag{55}$$

2. An arbitrary element $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ is represented in one of two equivalent forms:

$$\widetilde{J}_\mu(\cdot) = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i k(\cdot, i) \tag{56}$$

$$= \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle, \tag{57}$$

where

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Phi}(i).$$

In either form, $\widetilde{J}_\mu(\cdot)$ is uniquely specified by the expansion coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ and sample states $\{i \mid i \in \widetilde{\mathcal{S}}\}$.

Figure 3: Structure of the cost space $\mathcal{H}_k$

This form was the starting point for the development of the BRE(SV) algorithm [Eq. (31)]. It is important to note that the weighting element $\underline{\Theta}$ is itself an element of the RKHS $\mathcal{H}_k$, and therefore can be expressed in terms of the basis functions $\{\underline{\Phi}(i) \mid i \in \widetilde{\mathcal{S}}\}$, which span $\mathcal{H}_k$:

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Phi}(i).$$

The structure of the kernel k and cost functions $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ are summarized in Figure 3. Note that there are two equivalent forms for representing $\widetilde{J}_\mu(\cdot)$ [Eqs. (56) and (57)]. Eq. (56) is important from a computational standpoint because the output of a general kernel-based regression method is explicitly of this compact form (that is, the input to the regression method is a set of training data $\mathcal{D}$, and the output are the coefficients λ_i). Eq. (57) is important because it helps establish the mapping from $\mathcal{H}_k$ to $\mathcal{H}_\mathcal{K}$, as will be shown in the next section.

4.2 The Bellman Residual Space $\mathcal{H}_\mathcal{K}$

The previous section showed that the RKHS $\mathcal{H}_k$ contains the possible set of cost-to-go functions $\widetilde{J}_\mu(\cdot)$. This section will show that the RKHS $\mathcal{H}_\mathcal{K}$ of the associated Bellman kernel contains functions which are closely related to the set of Bellman residual functions $BR(\cdot)$ associated with the cost functions $\widetilde{J}_\mu(\cdot)$. The structure of $\mathcal{H}_\mathcal{K}$ is summarized in Figure 4;

1. Using the reproducing kernel map,

$$\underline{\Psi}(i)(\cdot) = \mathcal{K}(\cdot, i),$$

the kernel $\mathcal{K}(\cdot,\cdot)$ is represented as

$$\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle \tag{60}$$

2. An arbitrary element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ is represented in one of two equivalent forms:

$$\widetilde{W}_\mu(\cdot) = \sum_{i \in \widetilde{\mathcal{S}}} \beta_i \mathcal{K}(\cdot, i) \tag{61}$$

$$= \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle, \tag{62}$$

where

$$\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \beta_i \underline{\Psi}(i). \tag{63}$$

In either form, $\widetilde{W}_\mu(\cdot)$ is uniquely specified by the expansion coefficients $\{\beta_i \mid i \in \widetilde{\mathcal{S}}\}$ and sample states $\{i \mid i \in \widetilde{\mathcal{S}}\}$.

Figure 4: Structure of the Bellman residual space $\mathcal{H}_\mathcal{K}$

in particular, note that every element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ can be expressed as

$$\widetilde{W}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle. \tag{58}$$

Recall that, given any cost function $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$, the associated Bellman residual function $BR(\cdot)$ is defined by

$$BR(i) = \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \right).$$

The derivation leading to Eq. (33) showed that this Bellman residual function can be expressed as

$$BR(i) = -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle.$$

Finally, identifying $\langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle$ as an element $\widetilde{W}_\mu(\cdot)$ of $\mathcal{H}_\mathcal{K}$ allows the Bellman residual to be written as

$$BR(i) = -g_i^\mu + \widetilde{W}_\mu(i). \tag{59}$$

By the preceding construction, every cost function $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ has a corresponding function $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$, and this function $\widetilde{W}_\mu(\cdot)$ is equal to the Bellman residual function up to the known factor $-g_i^\mu$. In other words, a mapping $\mathcal{H}_k \to \mathcal{H}_\mathcal{K}$ has been constructed which, given a cost function $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$, allows us to find its corresponding residual function $\widetilde{W}_\mu$ in $\mathcal{H}_\mathcal{K}$. However, we are interested in algorithms which first compute the residual function

and then find the corresponding cost function. Therefore, the mapping must be shown to be invertible. The following theorem establishes this important property:

Theorem 6 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate. Then there exists a linear, invertible mapping $\hat{A}$ from $\mathcal{H}_k$, the RKHS of k, to $\mathcal{H}_\mathcal{K}$, the RKHS of the associated Bellman kernel $\mathcal{K}$:*

$$\begin{aligned} \hat{A}: \quad & \mathcal{H}_k \to \mathcal{H}_\mathcal{K} \\ & \hat{A}\widetilde{J}_\mu(\cdot) = \widetilde{W}_\mu(\cdot). \end{aligned}$$

Furthermore, if an element $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ is given by

$$\widetilde{J}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle,$$

then the corresponding element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ is given by

$$\widetilde{W}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle,$$

and vice versa:

$$\begin{array}{rcl} \widetilde{J}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ & \Updownarrow & \\ \widetilde{W}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle. \end{array} \tag{64}$$

The mapping $\hat{A}$ of Theorem 6 allows us to design BRE algorithms that find the weight element $\underline{\Theta}$ of the desired residual function $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ and then map the result back to $\mathcal{H}_k$ in order to find the associated cost function $\widetilde{J}_\mu(\cdot)$ (using Eq. (64)).

The objective of the BRE algorithms is to force the Bellman residuals $BR(i)$ to zero at the sample states $\widetilde{\mathcal{S}}$:

$$BR(i) = 0 \quad \forall i \in \widetilde{\mathcal{S}}. \tag{65}$$

Now, using Eq. (59), we see that Eq. (65) is equivalent to

$$\widetilde{W}_\mu(i) = g_i^\mu \quad \forall i \in \widetilde{\mathcal{S}}. \tag{66}$$

We have now succeeded in reducing the BRE problem to a straightforward regression problem in $\mathcal{H}_\mathcal{K}$: the task is to find a function $\widetilde{W}_\mu(i) \in \mathcal{H}_\mathcal{K}$ that satisfies Eq. (66). The training data of the regression problem is given by

$$\mathcal{D} = \{(i, g_i^\mu) \mid i \in \widetilde{\mathcal{S}}\}.$$

Notice that the training data is readily available, since by assumption the sample states $\widetilde{\mathcal{S}}$ are given, and the stage costs g_i^μ are obtained directly from the MDP specification.

4.3 A Family of Kernel-Based BRE Algorithms

With the invertible mapping between the RKHSs $\mathcal{H}_k$ and $\mathcal{H}_\mathcal{K}$ established, a generalized family of algorithms for kernel-based BRE can now be presented. The family is summarized in Algorithm 4. An important input to the algorithms is a generic kernel-based regression method (denoted by $\mathcal{R}$), which takes the set of training data $\mathcal{D} = \{(i, g_i^\mu) \mid i \in \widetilde{\mathcal{S}}\}$ and the associated Bellman kernel $\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$, and outputs the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ such that the regression function solution $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ is given by

$$\widetilde{W}_\mu(i) = \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \mathcal{K}(i, i').$$

The corresponding weight element $\underline{\Theta}$ is given by

$$\underline{\Theta} = \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \underline{\Psi}(i'). \tag{67}$$

For notational convenience, the process of solving the regression problem using the regression method $\mathcal{R}$ to obtain the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ is denoted by

$$\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\} = \mathcal{R}(\mathcal{D}, \mathcal{K}).$$

By choosing different kernel-based regression techniques as the input $\mathcal{R}$ to Algorithm 4, a family of BRE algorithms is obtained.

After the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ are determined, the cost function $\widetilde{J}_\mu(i)$ can be computed using Equations (64), (67), and (53):

$$\begin{aligned}
\widetilde{W}_\mu(i) &= \langle \underline{\Theta}, \underline{\Psi}(i) \rangle \\
&\Updownarrow \\
\widetilde{J}_\mu(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\
&= \langle \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \underline{\Psi}(i') \ , \ \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \langle \underline{\Psi}(i'), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(\langle \underline{\Phi}(i'), \underline{\Phi}(i) \rangle - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} \langle \underline{\Phi}(j), \underline{\Phi}(i) \rangle \right) \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(k(i', i) - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} k(j, i) \right)
\end{aligned} \tag{68}$$

Once $\widetilde{J}_\mu(\cdot)$ is found, policy evaluation is complete.

Note that Eq. (68) is identical to the functional form for $\widetilde{J}_\mu(\cdot)$ that was found in deriving BRE(SV) [Eq. (48)]. Although that derivation was carried out in a different way (by computing a dual optimization problem instead of explicitly constructing a map between

Algorithm 4 Generalized approximate policy iteration using kernel-based BRE

Input: $(\mu_0, \widetilde{\mathcal{S}}, k, \mathcal{R})$
μ_0: initial policy
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel function defined on $\mathcal{S} \times \mathcal{S}$, $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$
$\mathcal{R}$: generic kernel-based regression method
Begin
Define $\mathcal{K}(i,i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$ using Eq. (43) {Define the associated Bellman kernel}
$\mu \leftarrow \mu_0$
loop
  Set $\mathcal{D} = \{(i, g_i^\mu) \mid i \in \widetilde{\mathcal{S}}\}$. {Construct the training data set}
  Compute $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\} = \mathcal{R}(\mathcal{D}, \mathcal{K})$. {Solve the regression problem, using the regression method $\mathcal{R}$ and the associated Bellman kernel $\mathcal{K}$}
  Using the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$, compute the cost function $\widetilde{J}_\mu(i)$ using Eq. (68). {Policy evaluation step complete}
  $\mu(i) \leftarrow \arg\min_u \sum_{j \in \mathcal{S}} P_{ij}(u) \left(g(i,u) + \alpha \widetilde{J}_\mu(j) \right)$ {Policy improvement}
end loop
End

$\mathcal{H}_k$ and $\mathcal{H}_\mathcal{K}$), the resulting cost functions have the same functional form. The more general family of BRE algorithms developed in this section provides a deeper insight into how the BRE(SV) algorithm works.

To summarize, the generalized family of BRE algorithms allows any kernel-based regression method to be utilized to perform BRE and generate an approximate cost-to-go function. If the learned regression function $\widetilde{W}_\mu(\cdot)$ satisfies $\widetilde{W}_\mu(i) = g_i$ at some state i, then the Bellman residual of the corresponding cost function $\widetilde{J}_\mu(i)$, calulated using Eq. (68), is identically zero (as long as a nondegenerate kernel is used, as shown by Theorem 6).

5. BRE Using Gaussian Process Regression

The previous section showed how the problem of eliminating the Bellman residuals at the set of sample states $\widetilde{\mathcal{S}}$ is equivalent to a simple regression problem in $\mathcal{H}_\mathcal{K}$, and how to compute the corresponding cost function once the regression problem is solved. We now present BRE(GP), a BRE algorithm that uses Gaussian process regression to compute the solution to the regression problem in $\mathcal{H}_\mathcal{K}$. Like BRE(SV), BRE(GP) produces a cost-to-go solution whose Bellman residuals are zero at the sample states, and therefore reduces to exact policy iteration in the limit of sampling the entire state space (this will be proved later in the section). In addition, BRE(GP) can automatically select any free kernel parameters and provide natural error bounds on the cost-to-go solution, and therefore directly addresses the two important questions posed at the end of Section 3. As we will see, these desirable features of BRE(GP) arise because Gaussian process regression provides a tractable way to compute the posterior covariance and log marginal likelihood.

Pseudocode for BRE(GP) is shown in Algorithm 5. Similar to BRE(SV), BRE(GP) takes an initial policy μ_0, a set of sample states $\widetilde{\mathcal{S}}$, and a kernel k as input. However, it also

Algorithm 5 BRE(GP)

Input: $(\mu_0, \widetilde{\mathcal{S}}, k, \underline{\Omega})$
μ_0: initial policy
$\widetilde{\mathcal{S}}$: set of sample states
k: kernel (covariance) function defined on $\mathcal{S} \times \mathcal{S} \times \mathbf{\Omega}$, $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$
$\underline{\Omega}$: initial set of kernel parameters
Begin
Define $\mathcal{K}(i, i'; \underline{\Omega}) = \langle \underline{\Psi}(i; \underline{\Omega}), \underline{\Psi}(i'; \underline{\Omega}) \rangle$ {Define the associated Bellman kernel}
$\mu \leftarrow \mu_0$
loop
  Construct $\underline{g}^\mu$, the vector of stage costs $g_i^\mu \quad \forall i \in \widetilde{\mathcal{S}}$
  repeat
    Construct the Gram matrix $\mathbb{K}(\underline{\Omega})$, where $\mathbb{K}(\underline{\Omega})_{ii'} = \mathcal{K}(i, i'; \underline{\Omega}) \quad \forall i, i' \in \widetilde{\mathcal{S}}$, using Eq. (43)
    Solve $\underline{\lambda} = \mathbb{K}(\underline{\Omega})^{-1} \underline{g}^\mu$
    Calculate the gradient of the log marginal likelihood, $\nabla_{\underline{\Omega}} \log p(\underline{g}^\mu | \widetilde{\mathcal{S}}, \underline{\Omega})$, where

$$\frac{\partial \log p(\underline{g}^\mu | \widetilde{\mathcal{S}}, \underline{\Omega})}{\partial \Omega_j} = \frac{1}{2} \mathrm{tr} \left((\underline{\lambda} \underline{\lambda}^T - \mathbb{K}(\underline{\Omega})^{-1}) \frac{\partial \mathbb{K}(\underline{\Omega})}{\partial \Omega_j} \right).$$

    Update the kernel parameters using any gradient-based optimization rule:

$$\underline{\Omega} \leftarrow \underline{\Omega} + \gamma \nabla_{\underline{\Omega}} \log p(\underline{g}^\mu | \widetilde{\mathcal{S}}, \underline{\Omega}),$$

    where γ is an appropriately selected step size
  until stopping condition for gradient-based optimization rule is met
  Using the coefficients $\{\lambda_i \mid i \in \widetilde{\mathcal{S}}\}$ and kernel parameters $\underline{\Omega}$, compute the cost function

$$\widetilde{J}_\mu(i) = \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(k(i', i; \underline{\Omega}) - \alpha \sum_{j \in \mathcal{S}} P_{i'j}^\mu k(j, i; \underline{\Omega}) \right)$$

  {Policy evaluation step complete}
  Compute $E(i)$, the 1-σ error bound on the Bellman residual function

$$E(i) = \sqrt{\mathcal{K}(i, i; \underline{\Omega}) - \underline{h}^T \mathbb{K}(\underline{\Omega})^{-1} \underline{h}}$$

  where $h_j \equiv \mathcal{K}(i, j; \underline{\Omega}), \quad j \in \widetilde{\mathcal{S}}$
  $\mu(i) \leftarrow \arg\min_u \sum_{j \in \mathcal{S}} P_{ij}(u) \left(g(i, u) + \alpha \widetilde{J}_\mu(j) \right)$ {Policy improvement}
end loop
End

takes a set of initial kernel parameters $\underline{\Omega} \in \mathbf{\Omega}$ (in BRE(SV), these parameters were assumed to be fixed). The kernel k, as well as its associated Bellman kernel $\mathcal{K}$ and the Gram matrix

$\mathbb{K}$ all depend on these parameters, and to emphasize this dependence they are written as $k(i, i'; \underline{\Omega})$, $\mathcal{K}(i, i'; \underline{\Omega})$, and $\mathbb{K}(\underline{\Omega})$, respectively.

The algorithm consists of two nested loops. The inner loop (lines 11-16) is responsible for repeatedly solving the regression problem in $\mathcal{H}_{\mathcal{K}}$ (notice that the target values of the regression problem, the one-stage costs $\underline{g}^{\mu}$, are computed in line 10) and adjusting the kernel parameters using gradient-based optimization. This process is carried out with the policy fixed, so the kernel parameters are tuned to each policy prior to the policy improvement stage being carried out. Line 13 computes the $\underline{\lambda}$ values necessary to compute the mean function of the posterior process [Eq. (23)]. Lines 14 and 15 then compute the gradient of the log likelihood function, using Eq. (25), and use this gradient information to update the kernel parameters. This process continues until a maximum of the log likelihood function has been found.

Once the kernel parameters are optimally adjusted for the current policy μ, the main body of the outer loop (lines 17-19) performs three important tasks. First, it computes the cost-to-go solution $\widetilde{J}_{\mu}(i)$ using Eq. (68) (rewritten on line 17 to emphasize dependence on $\underline{\Omega}$). Second, on line 18, it computes the posterior standard deviation $E(i)$ of the Bellman residual function. This quantity is computed using the standard result from Gaussian process regression for computing the posterior variance [Eq. (22)], and it gives a Bayesian error bound on the magnitude of the Bellman residual $BR(i)$. This error bound is useful, of course, because the goal is to achieve small Bellman residuals at as many states as possible. Finally, the algorithm carries out a policy improvement step in Line 19.

Note that the log likelihood function is not necessarily guaranteed to be convex, so it may be possible for the gradient optimization carried out in the inner loop (lines 11-16) to converge to a local (but not global) maximum. However, by initializing the optimization process at several starting points, it may be possible to decrease the probability of ending up in a local maximum (Rasmussen and Williams, 2006, 5.4).

The following theorems establish the same important properties of BRE(GP) that were proved earlier for BRE(SV):

Theorem 7 *Assume that the kernel $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ is nondegenerate. Then the cost-to-go functions $\widetilde{J}_{\mu}(i)$ computed by BRE(GP) (on line 17) satisfy*

$$\widetilde{J}_{\mu}(i) = g_i^{\mu} + \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \widetilde{J}_{\mu}(j) \quad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ are identically zero at every state $i \in \widetilde{\mathcal{S}}$.

An immediate corollary of Theorem 7 follows:

Corollary 8 *Assume that the kernel $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then the cost-to-go function $\widetilde{J}_{\mu}(i)$ produced by BRE(GP) satisfies*

$$\widetilde{J}_{\mu}(i) = J_{\mu}(i) \quad \forall i \in \mathcal{S}.$$

That is, the cost function $\widetilde{J}_{\mu}(i)$ is exact.

Corollary 9 *Assume that the kernel $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then BRE(GP) is equivalent to exact policy iteration.*

5.1 Computational Complexity

The computational complexity of running BRE(GP) is dominated by steps 12 (constructing the Gram matrix) and 13 (inverting the Gram matrix). To obtain an expression for the computational complexity of the algorithm, first define

$$n_s \equiv |\widetilde{\mathcal{S}}|$$

as the number of sample states. Furthermore, define the average branching factor of the MDP, β, as the average number of possible successor states for any state $i \in \mathcal{S}$, or equivalently, as the average number of terms P_{ij}^{μ} that are nonzero for fixed i and μ. Finally, recall that since $\mathcal{K}(i, i')$ is nondegenerate (as proved earlier), the Gram matrix is positive definite and symmetric.

Now, each entry of the Gram matrix $\mathbb{K}(\underline{\Omega})_{ii'}$ is computed using Eq. (43). Using the average branching factor defined above, computing a single element of the Gram matrix using Eq. (43) therefore requires $\mathcal{O}(\beta^2)$ operations. Since the Gram matrix is symmetric and of dimension $n_s \times n_s$, there are $n_s(n_s+1)/2$ unique elements that must be computed. Therefore, the total number of operations to compute the full Gram matrix is

$$\mathcal{O}(\beta^2 n_s(n_s+1)/2).$$

Once the Gram matrix is constructed, its inverse must be computed in line 13. Since the Gram matrix is positive definite and symmetric, Cholesky decomposition can be used, resulting in a total complexity of

$$\mathcal{O}(n_s^3)$$

for line 13. (The results of the Cholesky decomposition computed in line 13 can and should be saved to speed up the calculation of the log marginal likelihood in line 14, which also requires the inverse of the Gram matrix). As a result, the total complexity of the BRE(GP) algorithm is

$$\mathcal{O}(n_s^3 + \beta^2 n_s(n_s+1)/2). \tag{69}$$

This complexity can be compared with the complexity of exact policy iteration. Recall that exact policy iteration requires inverting the matrix $(I - \alpha P^{\mu})$ in order to evaluate the policy μ:

$$\underline{J}_{\mu} = (I - \alpha P^{\mu})^{-1} \underline{g}^{\mu}.$$

$(I - \alpha P^{\mu})$ is of dimension $n \times n$, where

$$n = |\mathcal{S}|$$

is the size of the state space. Therefore, the complexity of exact policy iteration is

$$\mathcal{O}(n^3). \tag{70}$$

Comparing Eqs. (69) and (70), notice that both expressions involve cubic terms in n_s and n, respectively. By assumption, the number of sample states is taken to be much smaller than the total size of the state space ($n_s \ll n$). Furthermore, the average branching factor β is typically also much smaller than n (and can certainly be no larger than n). Therefore, the computational complexity of BRE(GP) is significantly less than the computational complexity of exact policy iteration.

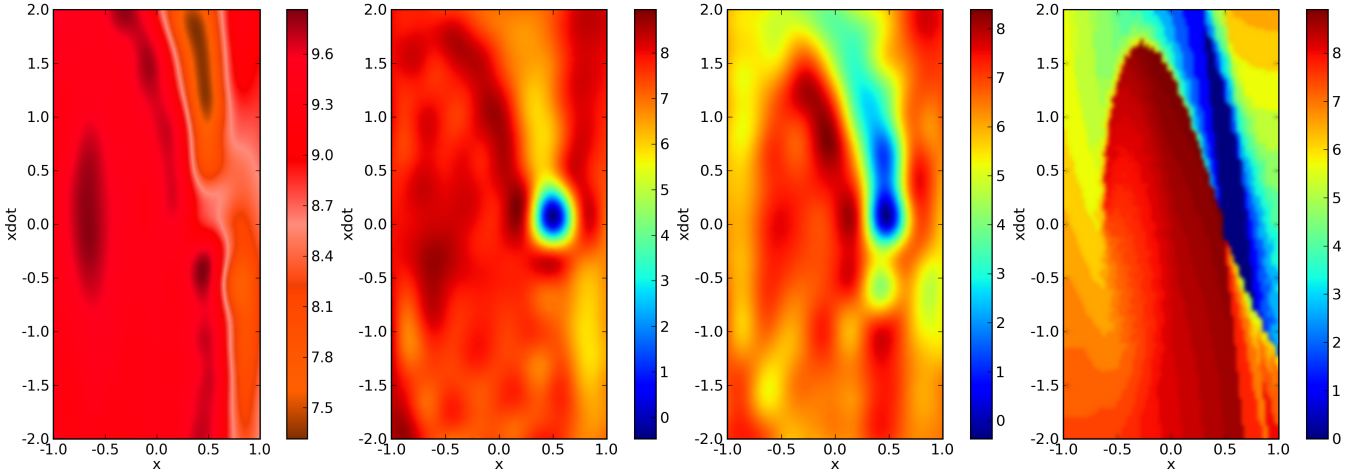

Figure 5: From left to right: Approximate cost-to-go computed by BRE(GP) for iterations 1, 2 , and 3; exact cost-to-go computed using value iteration.

5.2 Computational Results

BRE(GP) was implemented on the same mountain car problem examined earlier for BRE(SV). The kernel function employed for these experiments was

$$k((x_1, \dot{x}_1), (x_2, \dot{x}_2); \underline{\Omega}) = \exp\left(-(x_1 - x_2)^2/\Omega_1^2 - (\dot{x}_1 - \dot{x}_2)^2/\Omega_2^2\right).$$

This is the same functional form of the kernel that was used for BRE(SV). However, the length-scales Ω_1 and Ω_2 were left as free kernel parameters to be automatically learned by the BRE(GP) algorithm, in contrast with the BRE(SV) tests, where these values had to be specified by hand. The parameters were initially set at $\Omega_1 = \Omega_2 = 10$. These values were deliberately chosen to be far from the empirically chosen values used in the BRE(SV) tests, which were $\Omega_1 = 0.25$ and $\Omega_2 = 0.40$. Therefore, the test reflected a realistic situation in which the parameter values were not initially well-known.

Initially, the same 9x9 grid of sample states as used in the BRE(SV) tests was employed to compare the performance of the two algorithms:

$$\widetilde{\mathcal{S}} = \{(x, \dot{x}) \mid x = -1.0, -0.75, \ldots, 0.75, 1.0 \\ \dot{x} = -2.0, -1.5, \ldots, 1.5, 2.0\}.$$

BRE(GP) was then executed. Like the BRE(SV) test, the algorithm converged after 3 policy updates. The associated cost functions (after each round of kernel parameter optimization) are shown in Figure 5 along with the true optimal cost for comparison. The final parameter values found by BRE(GP) were $\Omega_1 = 0.253$ and $\Omega_2 = 0.572$. These values are similar to those found empirically in the BRE(SV) tests but are actually better values, as evidenced by the system response under the policy found by BRE(GP), which is shown in Figure 6. The BRE(GP) policy arrives in the parking area at time $t = 15$, 2 time steps faster than the BRE(SV) policy and only 1 step step slower than the optimal policy. This result demonstrates that BRE(GP) is not only able to learn appropriate values for initially

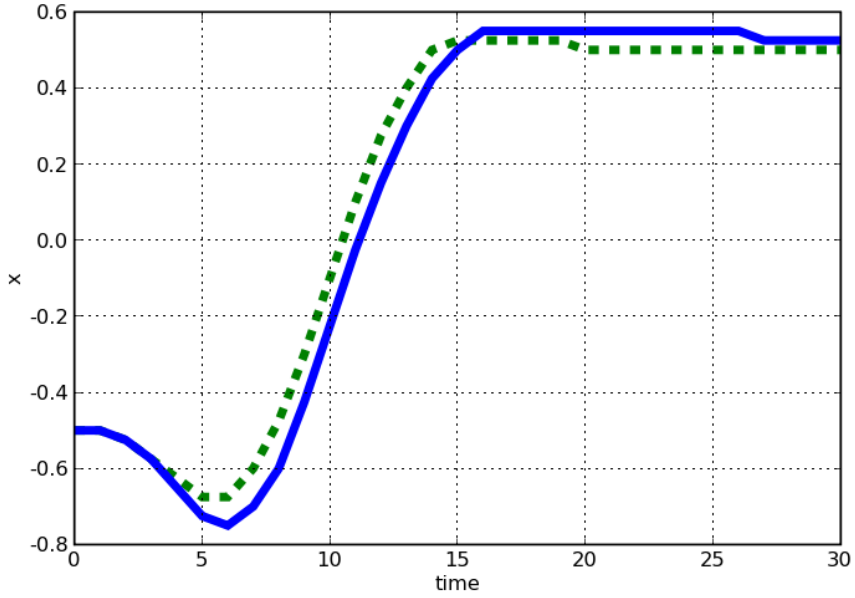

Figure 6: System response under the optimal policy (dashed line) and the policy computed by BRE(GP) after 3 iterations (solid line).

unknown kernel parameters, but also that the resulting policy can be of higher quality than could be obtained using time-consuming empirical testing.

As discussed, in addition to automatically learning the kernel parameters, BRE(GP) also computes error bounds on the cost-to-go solution. Figure 7 plots the Bellman residuals $BR(i)$, as well as the 2σ error bounds $2E(i)$ computed in line 18 of the BRE(GP) algorithm. For clarity, the plot is given as a function of x only; $\dot{x}$ was fixed to zero. The sampled states

$$\{(x, 0) \mid x = -1.0, -0.75, \ldots, 0.75, 1.0\}$$

are represented by red dots in the figure. Notice that, as expected, the Bellman residuals at the sampled states are exactly zero. Furthermore, the error bounds computed by BRE(GP), shown as the dashed green lines, do a good job of bounding the Bellman residuals at non-sampled states. Recall that these bounds represent the posterior variance of the Bellman residuals. Thus, for the 2σ case shown in the figure, the bounds represent a 95% confidence interval, so we expect that 95% of the Bellman residuals should lie within the bounds. There are two cases where the residuals lie outside the bounds (at $x = 0.45$ and $x = 0.70$), and there are a total of 81 points in the graph. This leads to an empirical confidence interval of

$$\frac{81 - 2}{81} = 97.53\%,$$

which is in excellent agreement with the bounds computed by BRE(GP).

A further series of tests were conducted to explore the quality of the policies produced by BRE(GP) as a function of the number of sample states. In these tests, a uniform

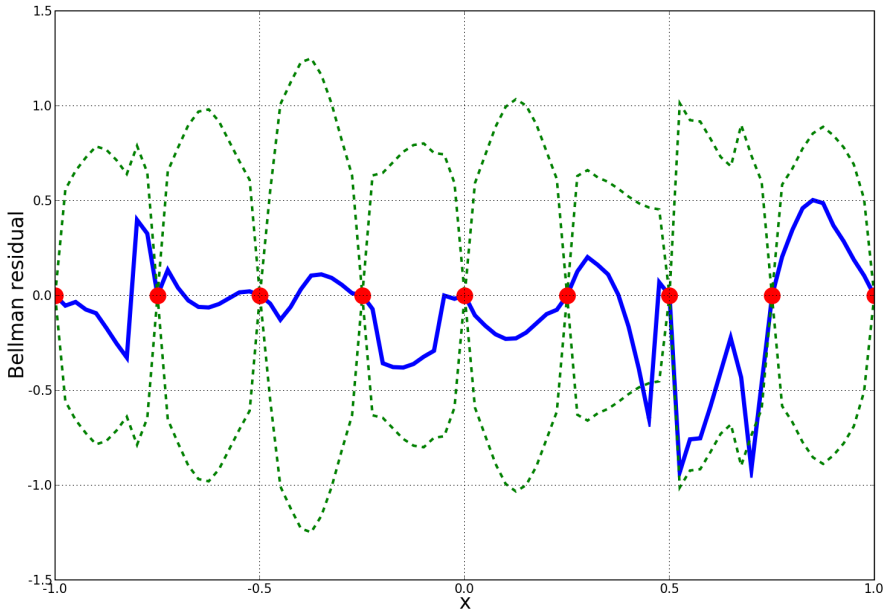

Figure 7: Bellman residuals (solid blue line) as a function of x ($\dot{x}$ is fixed to zero in this plot). Sample states are represented by the red dots; notice that the Bellman residuals at the sample states are exactly zero as expected. 2σ error bounds computed by BRE(GP) are shown by the dashed green lines.

sampling of the two-dimensional state space was used, and tests were conducted for as few as 8 sample states and as many as 13,041 (which was the total size of the state space, so the entire space was sampled in this test). For each test, the cost of the resulting policy, starting at the initial condition ($x = -0.5, \dot{x} = 0.0$), was compared with the cost of the optimal policy. Figure 8 plots the difference between these two costs (this value is called the *policy loss*) as a function of n_s, the number of sample states. Note that for this example problem, Corollary 9 guarantees that the policy loss must go to zero as n_s goes to 13,041, since in this limit the entire state space is sampled and BRE(GP) reduces to exact policy iteration, which always converges to the optimal policy. Figure 8 confirms this prediction, and in fact demonstrates a stronger result: the policy loss goes to zero well *before* the entire space is sampled. While this is behavior is not guaranteed to always occur, it does suggest that there may be a "critical number" of sample states, above which BRE(GP) yields the optimal policy.

6. Kernel Considerations and Extensions

The kernel $k(i, i')$ plays an important role in the BRE algorithms presented in this paper. Since the algorithms learn a cost function $\widetilde{J}_\mu(i)$ which is an element of the RKHS $\mathcal{H}_k$,

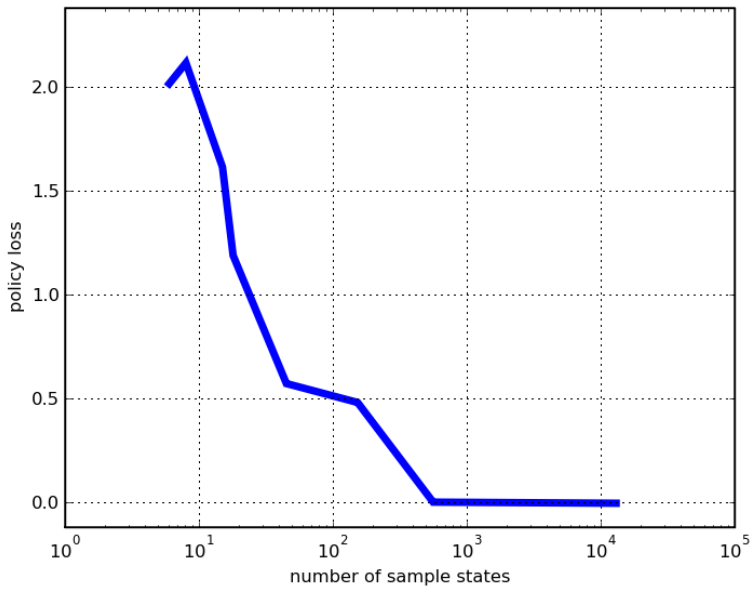

Figure 8: Policy loss vs. number of sampled states for the BRE(GP) algorithm. The total size of the state space is 13,041 (1.3041×10^4). Note that BRE(GP) finds the optimal policy (i.e. zero policy loss) well before the entire state space is sampled.

the kernel $k(i, i')$ implicitly determines the structure of the cost function that is ultimately produced by the algorithms. Furthermore, in order to guarantee that the Bellman residuals are identically zero at the sample states, it is important that $k(i, i')$ be nondegenerate. Functional forms for a variety of nondegenerate kernels are known; Table 6 lists several of these. Note that all of these kernels contain a number of free parameters, such as the positive semidefinite scaling matrix Σ, that can be automatically optimized by the BRE(GP) algorithm. Furthermore, it is well-known that new, "composite" kernels can be constructed by taking linear combinations (Rasmussen and Williams, 2006, 5.4): if $k_1(i, i')$ and $k_2(i, i')$ are kernels, then

$$k_3(i, i') = \Omega_1 k_1(i, i') + \Omega_2 k_2(i, i')$$

is also a valid kernel. Again, BRE(GP) can be used to learn the parameters Ω_1 and Ω_2, automatically determining an appropriate weighting for the problem at hand.

All of the kernels presented in Table 6 can be described as "geometric" in the sense that they compute a dot product - either of the form $(i - i')^T \Sigma (i - i')$ or $i^T \Sigma i'$ - in the state space. This dot product encodes a notion of distance or proximity between states in the state space. Thus, these geometric kernels are appropriate for many problems, such as the mountain car problem used as the example in this paper, where a natural distance metric can be defined on the state space.

An alternative approach may be taken for problems that do not admit a natural distance metric. An example of such a problem is the two-room robot navigation problem shown in Figure 9. In this problem, a robot must navigate to the goal location, shown in green,

Table 1: Nondegenerate kernel function examples (Rasmussen and Williams, 2006, 4.2.3)

Kernel Name	Functional Form
Radial basis function	$k(i,i') = \exp\left(-(i-i')^T\Sigma(i-i')\right)$
Exponential	$k(i,i') = \exp\left(-\sqrt{(i-i')^T\Sigma(i-i')}\right)$
γ-exponential	$k(i,i') = \exp\left(-((i-i')^T\Sigma(i-i'))^\gamma\right)$
Matérn	$k(i,i') = \frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{l}(i-i')^T\Sigma(i-i')\right)^\nu K_\nu\left(\frac{\sqrt{2\nu}}{l}(i-i')^T\Sigma(i-i')\right)$
Rational quadratic	$k(i,i') = \left(1+(i-i')^T\Sigma(i-i')/(2\alpha)\right)^{-\alpha}$
Neural network	$k(i,i') = \sin^{-1}\left(\frac{2i^T\Sigma i'}{\sqrt{(1+2i^T\Sigma i')(1+2i'^T\Sigma i)}}\right)$

by moving between adjacent points on the two-dimensional grid. The robot cannot move through the vertical wall in the center of the grid. Because of the presence of the wall, a kernel based on a normal Euclidean distance metric of the form $\sqrt{(i-i')^T\Sigma(i-i')}$ (where i and i' are two-dimensional position vectors) would work poorly, since it would tend to predict high correlation in the cost-to-go between two states on opposite sides of the wall (shown in red). However, since all feasible paths between these two highlighted states are long, the cost-to-go values of these two states are in reality very weakly correlated. To solve this problem, a new type of kernel function can be constructed that relies on exploiting the graph structure of the state space. More precisely, for a fixed policy, the state transition dynamics of the MDP are described by a Markov chain, where each state i is represented as a node in the graph and is connected to states that are reachable from i in one step with positive probability. Instead of using a distance metric based on the geometry of the problem, one can define a distance metric based on the graph which measures the number of steps necessary to move from one state to another. A number of authors have shown that these graph-based methods give rise to kernel functions that naturally capture the structure of the cost-to-go function and have demonstrated good results in applications like the two-room robot navigation problem (Sugiyama et al., 2006, 2007; Mahadevan, 2005; Mahadevan and Maggioni, 2005; Belkin and Niyogi, 2005, 2004; Smart, 2004). The BRE algorithms developed in this paper can make use of either geometric or graph-based kernels (or even linear combinations of both kinds) depending which type of kernel is most appropriate for the application.

6.1 Generalized n-stage Associated Bellman Kernels

The associated Bellman kernel $\mathcal{K}(i,i')$ plays a central role in the BRE algorithms developed in this paper. An interesting extension to these algorithms is possible by deriving a generalized form of the associated Bellman kernel. Recall that this kernel was derived starting from the fixed policy Bellman equation,

$$T_\mu J_\mu = J_\mu.$$

A more general expression for the associated Bellman kernel can be derived by starting with other, equivalent forms of the Bellman equation, generated by successively applying

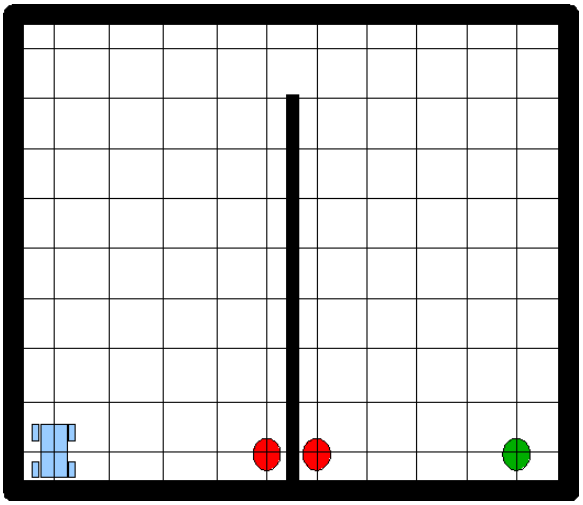

Figure 9: Two-room robot navigation problem.

the dynamic programming operator T_μ:

$$\begin{aligned} T_\mu^2 J_\mu &= T_\mu J_\mu = J_\mu \\ &\vdots \\ T_\mu^n J_\mu &= J_\mu. \end{aligned} \tag{71}$$

Eq. (71) is an "n-stage" equivalent version of the Bellman equation. If the n-stage Bellman residual $BR^n(i)$ is defined as the difference between the right and left hand sides of Eq. (71), we obtain the following:

$$\begin{aligned} BR^n(i) &\equiv \widetilde{J}_\mu(i) - T_\mu^n \widetilde{J}_\mu(i) \\ &= \widetilde{J}_\mu(i) - T_\mu \left(T_\mu^{n-1} \widetilde{J}_\mu(i) \right) \\ &= \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu T_\mu \left(T_\mu^{n-2} \widetilde{J}_\mu(j_1) \right) \right) \\ &\vdots \\ &= \widetilde{J}_\mu(i) - \left(g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu \left(g_{j_1}^\mu + \alpha \sum_{j_2 \in \mathcal{S}} P_{j_1 j_2}^\mu \left(g_{j_2}^\mu + \cdots + \alpha \sum_{j_n \in \mathcal{S}} P_{j_{n-1} j_n}^\mu \widetilde{J}_\mu(j_n) \right) \right) \right). \end{aligned}$$

Substituting the standard functional form $\widetilde{J}_\mu(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle$ into the expression for $BR^n(i)$ and using linearity of the inner product gives

$$\begin{aligned} BR^n(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\ &\quad - \left(g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu \left(g_{j_1}^\mu + \alpha \sum_{j_2 \in \mathcal{S}} P_{j_1 j_2}^\mu \left(g_{j_2}^\mu + \cdots + \alpha \sum_{j_n \in \mathcal{S}} P_{j_{n-1} j_n}^\mu \langle \underline{\Theta}, \underline{\Phi}(j_n) \rangle \right) \right) \right) \\ &= \langle \underline{\Theta}, \left(\underline{\Phi}(i) - \alpha^n \sum_{j_1, \ldots, j_n \in \mathcal{S}} \left(P_{ij_1}^\mu \ldots P_{j_{n-1} j_n}^\mu \right) \underline{\Phi}(j_n) \right) \rangle \\ &\quad - \left(g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu \left(g_{j_1}^\mu + \alpha \sum_{j_2 \in \mathcal{S}} P_{j_1 j_2}^\mu \left(g_{j_2}^\mu + \cdots + \alpha \sum_{j_{n-1} \in \mathcal{S}} P_{j_{n-2} j_{n-1}}^\mu g_{j_{n-1}}^\mu \right) \right) \right). \end{aligned}$$

Finally, a new n-stage feature vector $\underline{\Psi}^n(i)$ and n-stage cost $g_i^{n,\mu}$ can be defined as

$$\underline{\Psi}^n(i) \equiv \underline{\Phi}(i) - \alpha^n \sum_{j_1, \ldots, j_n \in \mathcal{S}} \left(P_{ij_1}^\mu \ldots P_{j_{n-1} j_n}^\mu \right) \underline{\Phi}(j_n)$$

and

$$g_i^{n,\mu} \equiv g_i^\mu + \alpha \sum_{j_1 \in \mathcal{S}} P_{ij_1}^\mu \left(g_{j_1}^\mu + \alpha \sum_{j_2 \in \mathcal{S}} P_{j_1 j_2}^\mu \left(g_{j_2}^\mu + \cdots + \alpha \sum_{j_{n-1} \in \mathcal{S}} P_{j_{n-2} j_{n-1}}^\mu g_{j_{n-1}}^\mu \right) \right),$$

respectively. Using these expressions, the n-stage Bellman residual can be expressed as

$$BR^n(i) = \langle \underline{\Theta}, \underline{\Psi}^n(i) \rangle - g_i^{n,\mu}.$$

Therefore, in direct analogy with the derivation of the earlier BRE algorithms, solving the regression problem

$$\widetilde{W}_\mu^n(i) = \langle \underline{\Theta}, \underline{\Psi}^n(i) \rangle = g_i^{n,\mu} \quad \forall i \in \widetilde{\mathcal{S}} \tag{72}$$

using the n-stage associated Bellman kernel given by

$$\mathcal{K}^n(i, i') = \langle \underline{\Psi}^n(i), \underline{\Psi}^n(i') \rangle$$

is equivalent to finding a cost-to-go function of the form $\widetilde{J}_\mu(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle$ satisfying

$$T_\mu^n(i) \widetilde{J}_\mu(i) = \widetilde{J}_\mu(i) \quad \forall i \in \widetilde{\mathcal{S}}.$$

Again, any kernel-based regression method can be used to solve the regression problem specified by Eq. (72), resulting in a general class of n-stage Bellman residual elimination algorithms. It is possible to prove that these n-stage BRE algorithms possess all of the desirable theoretical properties we have shown in this paper for single stage BRE. In particular, if the kernel $k(i, i')$ is nondegenerate, then the n-stage associated Bellman kernel $\mathcal{K}^n(i, i')$ is nondegenerate, the n-stage Bellman residuals $BR^n(i)$ are identically zero at every sample state, and the algorithms reduce to exact policy iteration in the limit of sampling the entire state space.

The advantage obtained by working with the n-stage Bellman equation is that the algorithm is forced to consider and adjust the relative values of more states in the state space, resulting in a more accurate cost-to-go solution. Indeed, in the n-stage problem, the algorithm must consider the values of not only every state reachable in 1 step from each sample state $i \in \widetilde{\mathcal{S}}$, but every state reachable in n steps. This increased solution accuracy comes at a higher computational cost. If the average braching factor of the MDP (that is, the average number of edges per node in the state transition Markov chain) is m, then computing the n-stage associated Bellman kernel $\mathcal{K}^n(i, i')$ involves $\mathcal{O}(m^n)$ operations. For some problems, however, the branching factor may be small enough to allow practical computation of $\mathcal{K}^n(i, i')$ for $n > 1$. (In particular, n-stage BRE is especially appropriate for deterministic problems, where the branching factor is 1 for every state). In these cases, n-stage BRE is practical to carry out and may increase the performance of the resulting policy. Future work will examine the performance of these n-stage BRE algorithms applied to benchmark reinforcement learning problems.

7. Conclusion

This paper has demonstrated how kernel-based techniques can be used to design a new class of reinforcement learning algorithms that carry out Bellman residual elimination (BRE). These BRE algorithms learn the structure of the cost-to-go function by forcing the Bellman residuals to zero at a chosen set of representative sample states. Essentially, our approach reduces the dimensionality of the fixed-policy Bellman equation to a much smaller linear system that is practical to solve, and uses the solution to the smaller system in order to infer the cost everywhere in the state space. The BRE algorithms have the desirable theoretical property of reducing to exact policy iteration when the entire state space is sampled, since in that case the algorithms effectively compute the solution to the full fixed-policy Bellman equation. Furthermore, by exploiting knowledge of the underlying MDP model, the algorithms avoid the need to carry out trajectory simulations and thus do not suffer from simulation noise effects.

Application of our BRE algorithms to a classic reinforcement learning problem indicates that they yield nearly optimal policies while requiring relatively few sample states (and therefore, can be computed efficiently). The kernel parameter learning mechanism employed by BRE(GP) was demonstrated to be effective at finding appropriate values of the kernel parameters. This is an important feature of the BRE(GP) algorithm, especially in higher-dimensional and more complex problems where suitable values of the kernel parameters may not be known a priori. Furthermore, the error bounds computed by BRE(GP) can be used not only for predicting the quality of the computed cost function, but also for actively choosing which regions of the state space to sample (i.e. by selecting more sample states in regions where the error is large).

Based on both the theoretical results (in particular, that our BRE algorithms are guaranteed to yield the optimal policy when all states are sampled) and experimental validation presented in this paper, we believe that our Bellman residual elimination algorithms offer a useful approach for efficiently solving large-scale reinforcement learning problems.

Acknowledgments

Research supported by the Hertz Foundation, the American Society for Engineering Education, and the Boeing Company.

Appendix A. Proofs

Lemma 1 *Assume the vectors* $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$ *are linearly independent. Then the vectors* $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$*, where* $\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j)$*, are also linearly independent.*
Proof Consider the real vector space $\mathcal{V}$ spanned by the vectors $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$. It is clear that $\underline{\Psi}(i)$ is a linear combination of vectors in $\mathcal{V}$, so a linear operator $\hat{B}$ that maps $\underline{\Phi}(i)$ to $\underline{\Psi}(i)$ can be defined:

$$\underline{\Psi}(i) = \hat{B}\underline{\Phi}(i).$$

Since

$$\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j),$$

the matrix of $\hat{B}$ is

$$(I - \alpha P^{\mu}),$$

where I is the identity matrix and P^{μ} is the probability transition matrix for the policy μ. Since P^{μ} is a stochastic matrix, its largest eigenvalue is 1 and all other eigenvalues have absolute value less than 1; hence all eigenvalues of αP^{μ} have absolute value less than or equal to $\alpha < 1$. Since all eigenvalues of I are equal to 1, $(I - \alpha P^{\mu})$ is full rank and $dim(ker(\hat{B})) = 0$. Therefore,

$$dim(\{\underline{\Psi}(i) \,|\, i \in \mathcal{S}\}) = dim(\{\underline{\Phi}(i) \,|\, i \in \mathcal{S}\}) = n_s$$

so the vectors $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$ are linearly independent. ■

Theorem 2 *Assume that the kernel* $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ *is nondegenerate. Then the associated Bellman kernel defined by* $\mathcal{K}(i, i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle$*, where* $\underline{\Psi}(i) = \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \underline{\Phi}(j)$*, is also nondegenerate.*
Proof Since $k(i, i')$ is nondegenerate, the vectors $\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}$ are linearly independent. Therefore, Lemma 1 applies, and the vectors $\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}$ are linearly independent, immediately implying that $\mathcal{K}(i, i')$ is nondegenerate. ■

Theorem 3 *Assume that the kernel* $k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ *is nondegenerate. Then the cost-to-go function* $\widetilde{J}_{\mu}(i)$ *produced by Algorithm 2 satisfies*

$$\widetilde{J}_{\mu}(i) = g_i^{\mu} + \alpha \sum_{j \in \mathcal{S}} P_{ij}^{\mu} \widetilde{J}_{\mu}(j) \qquad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ *are identically zero at every state* $i \in \widetilde{\mathcal{S}}$*.*
Proof Since $k(i, i')$ is nondegenerate, Theorem 2 applies and $\mathcal{K}(i, i')$ is also nondegenerate. Therefore, the Gram matrix $\mathbb{K}$ of $\mathcal{K}(i, i')$ is positive definite and invertible. It follows that,

in Line 8 of Algorithm 2, there is a unique solution for $\underline{\lambda}$. Having found a feasible solution $\underline{\lambda}$ for the dual problem given by [Eq. (49)], Slater's condition is satisfied and strong duality holds. Therefore, the primal problem [Eq. (45)] is feasible, and its optimal solution is given by Eq. (47). Feasibility of this solution implies

$$BR(i) = -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = 0 \quad \forall i \in \widetilde{\mathcal{S}}$$

as claimed. ■

Corollary 4 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then the cost-to-go function $\widetilde{J}_\mu(i)$ produced by Algorithm 2 satisfies*

$$\widetilde{J}_\mu(i) = J_\mu(i) \quad \forall i \in \mathcal{S}.$$

That is, the cost function $\widetilde{J}_\mu(i)$ is exact.

Proof Applying Theorem 3 with $\widetilde{\mathcal{S}} = \mathcal{S}$, we have

$$BR(i) = 0 \quad \forall i \in \mathcal{S}.$$

Using the definition of the Bellman residual,

$$BR(i) \equiv \widetilde{J}_\mu(i) - T_\mu \widetilde{J}_\mu(i),$$

immediately implies that

$$\widetilde{J}_\mu(i) = T_\mu \widetilde{J}_\mu(i) \quad \forall i \in \mathcal{S}.$$

Therefore, $\widetilde{J}_\mu(i)$ satisfies the fixed-policy Bellman equation, so

$$\widetilde{J}_\mu(i) = J_\mu(i) \quad \forall i \in \mathcal{S}$$

as claimed. ■

Corollary 5 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then BRE(SV) (given in Algorithm 3) is equivalent to exact policy iteration.*

Proof By Corollary 4, the cost function $\widetilde{J}_\mu(i)$ produced by BRE(SV) is equal to the exact cost $J_\mu(i)$, at every state $i \in \mathcal{S}$. Since the policy improvement step (Line 9) is also exact, BRE(SV) carries out exact policy iteration by definition, and converges in a finite number of steps to the optimal policy. ■

Theorem 6 *Assume that the kernel $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is nondegenerate. Then there exists a linear, invertible mapping $\hat{A}$ from $\mathcal{H}_k$, the RKHS of k, to $\mathcal{H}_\mathcal{K}$, the RKHS of the associated Bellman kernel $\mathcal{K}$:*

$$\begin{aligned} \hat{A}: \quad & \mathcal{H}_k \to \mathcal{H}_\mathcal{K} \\ & \hat{A}\widetilde{J}_\mu(\cdot) = \widetilde{W}_\mu(\cdot). \end{aligned}$$

Furthermore, if an element $\widetilde{J}_\mu(\cdot) \in \mathcal{H}_k$ *is given by*

$$\widetilde{J}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle,$$

then the corresponding element $\widetilde{W}_\mu(\cdot) \in \mathcal{H}_\mathcal{K}$ *is given by*

$$\widetilde{W}_\mu(\cdot) = \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle, \tag{73}$$

and vice versa:

$$\begin{array}{rcl} \widetilde{J}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ & \Updownarrow & \\ \widetilde{W}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle. \end{array}$$

Proof Recall from the proof of Lemma 1 that the relation between $\underline{\Phi}(i)$ and $\underline{\Psi}(i)$ [Eq. (53)] is defined by the invertible linear operator $\hat{B}$:

$$\underline{\Psi}(i) = \hat{B}\underline{\Phi}(i)$$

Using the definition of $\hat{B}$ and linearity of the inner product $\langle \cdot, \cdot \rangle$, Eq. (73) can be rewritten as

$$\begin{array}{rcl} \widetilde{W}_\mu(\cdot) & = & \langle \underline{\Theta}, \hat{B}\underline{\Phi}(\cdot) \rangle \\ & = & \hat{B}\langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ & = & \hat{B}\widetilde{J}_\mu(\cdot), \end{array}$$

which establishes the invertible linear mapping between $\mathcal{H}_k$ and $\mathcal{H}_\mathcal{K}$, as claimed. Therefore, the linear mapping $\hat{A}$ in the statement of the theorem exists and is equal to the linear mapping $\hat{B}$ from Lemma 1:

$$\hat{A} = \hat{B}.$$

Comparing Eqs. (57) and (62), notice that we can convert between a cost function $\widetilde{J}_\mu(\cdot)$ and its corresponding function $\widetilde{W}_\mu(\cdot)$ by replacing $\underline{\Phi}(\cdot)$ with $\underline{\Psi}(\cdot)$ in the inner product:

$$\begin{array}{rcl} \widetilde{J}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Phi}(\cdot) \rangle \\ & \Updownarrow & \\ \widetilde{W}_\mu(\cdot) & = & \langle \underline{\Theta}, \underline{\Psi}(\cdot) \rangle \end{array}$$

■

Theorem 7 *Assume that the kernel* $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ *is nondegenerate. Then the cost-to-go functions* $\widetilde{J}_\mu(i)$ *computed by BRE(GP) (on line 17) satisfy*

$$\widetilde{J}_\mu(i) = g_i^\mu + \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \widetilde{J}_\mu(j) \quad \forall i \in \widetilde{\mathcal{S}}.$$

That is, the Bellman residuals $BR(i)$ are identically zero at every state $i \in \widetilde{\mathcal{S}}$.
Proof Note that line 13 of the BRE(GP) algorithm computes the weights $\underline{\lambda}$ as

$$\underline{\lambda} = \mathbb{K}(\underline{\Omega})^{-1}\underline{g}^{\mu}.$$

Aside from the purely notational difference emphasizing the dependence of the Gram matrix $\mathbb{K}(\underline{\Omega})$ on the kernel parameters $\underline{\Omega}$, the weights $\underline{\lambda}$ and the cost function $\widetilde{J}_{\mu}(i)$ computed by BRE(GP) are identical to those computed by BRE(SV). Therefore, Theorem 3 applies directly, and the Bellman residuals are identically zero at every state $i \in \widetilde{\mathcal{S}}$. ■

Corollary 8 *Assume that the kernel $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then the cost-to-go function $\widetilde{J}_{\mu}(i)$ produced by BRE(GP) satisfies*

$$\widetilde{J}_{\mu}(i) = J_{\mu}(i) \quad \forall i \in \mathcal{S}.$$

That is, the cost function $\widetilde{J}_{\mu}(i)$ is exact.
Proof Applying Theorem 7 with $\widetilde{\mathcal{S}} = \mathcal{S}$, we have

$$BR(i) = 0 \ \ \forall i \in \mathcal{S}.$$

Using the definition of the Bellman residual,

$$BR(i) \equiv \widetilde{J}_{\mu}(i) - T_{\mu}\widetilde{J}_{\mu}(i),$$

immediately implies that

$$\widetilde{J}_{\mu}(i) = T_{\mu}\widetilde{J}_{\mu}(i) \ \ \forall i \in \mathcal{S}.$$

Therefore, $\widetilde{J}_{\mu}(i)$ satisfies the fixed-policy Bellman equation, so

$$\widetilde{J}_{\mu}(i) = J_{\mu}(i) \ \ \forall i \in \mathcal{S}$$

as claimed. ■

Corollary 9 *Assume that the kernel $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$ is nondegenerate, and that $\widetilde{\mathcal{S}} = \mathcal{S}$. Then BRE(GP) is equivalent to exact policy iteration.*
Proof By Corollary 4, we have that the cost produced by BRE(GP), $\widetilde{J}_{\mu}(i)$ is equal to the exact cost $J_{\mu}(i)$, at every state $i \in \mathcal{S}$. Since the policy improvement step (Line 19) is also exact, the algorithm carries out exact policy iteration by definition, and converges in a finite number of steps to the optimal policy. ■

References

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

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *ICML*, pages 30–37, 1995.

M. Belkin and P. Niyogi. Towards a theoretical foundation for laplacian-based manifold methods. In Peter Auer and Ron Meir, editors, *COLT*, volume 3559 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 2005. ISBN 3-540-26556-2.

M. Belkin and P. Niyogi. Semi-supervised learning on riemannian manifolds. *Machine Learning*, 56(1-3):209–239, 2004.

C. Berg, J. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer-Verlag, New York, 1984.

D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2007.

D. Bertsekas and S. Ioffe. Temporal Differences-Based Policy Iteration and Applications in Neuro-Dynamic Programming. http://web.mit.edu/people/dimitrib/Tempdif.pdf, 1996.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. In *Knowledge Discovery and Data Mining*, number 2, 1998.

J. Candela and C. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

N. Christianini and J. Shawe-Taylor. *Support Vector Machines and Other kernel-based Learning Methods*. Cambridge University Press, 2000.

DP de Farias and B. Van Roy. The Linear Programming Approach to Approximate Dynamic Programming. *Operations Research*, 51(6):850–865, 2003.

M. Deisenroth, J. Peters, and C. Rasmussen. Approximate dynamic programming with gaussian processes. In *Proceedings of the American Control Conference*, 2008.

T. Dietterich and X. Wang. Batch value function approximation via support vectors. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *NIPS*, pages 1491–1498. MIT Press, 2001.

Y. Engel. *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University, 2005.

Y. Engel, S. Mannor, and Ron Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 154–161. AAAI Press, 2003. ISBN 1-57735-189-4.

Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with gaussian processes. In Luc De Raedt and Stefan Wrobel, editors, *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 201–208. ACM, 2005. ISBN 1-59593-180-5.

F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.

S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. http://citeseer.ist.psu.edu/244558.html, 1999.

M. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003. ISSN 1533-7928.

M. Maggioni and S. Mahadevan. Fast direct policy evaluation using multiscale analysis of markov diffusion processes. In William W. Cohen and Andrew Moore, editors, *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 601–608. ACM, 2006. ISBN 1-59593-383-2.

S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *International Conference on Machine Learning*, 2005.

S. Mahadevan and M. Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. In *NIPS*, 2005.

R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 1:815–857, 2008.

D. Ormoneit and Ś. Sen. Kernel-Based Reinforcement Learning. *Machine Learning*, 49(2): 161–178, 2002.

J. Platt. Using sparseness and analytic QP to speed training of support vector machines. In *Advances in Neural Information Processing Systems*, pages 557–563, 1999.

M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

C. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems*, 16:751–759, 2004.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

J. Reisinger, P. Stone, and R. Miikkulainen. Online kernel selection for bayesian reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

S. Saitoh. *Theory of Reproducing Kernels and its Applications*. Longman Scientific and Technical, Harlow, England, 1988.

B. Schölkopf. Support Vector Learning. Available from http://www.kyb.tuebingen.mpg.de/-bs, 1997.

B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

P. Schweitzer and A. Seidman. Generalized polynomial approximation in Markovian decision processes. *Journal of mathematical analysis and applications*, 110:568–582, 1985.

A. Barto W. Powell J. Si and D. Wunsch. *Learning and Approximate Dynamic Programming*. IEEE Press, NY, 2004. URL `http://citeseer.ist.psu.edu/651143.html`.

W. Smart. Explicit manifold representations for value-function approximation in reinforcement learning. In *AMAI*, 2004.

A. Smola and B. Schölkopf. A Tutorial on Support Vector Regression. *Statistics and Computing*, 14:199–222, 2004.

M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar. Geodesic gaussian kernels for value function approximation. In *Workshop on Information-Based Induction Sciences*, 2006.

M. Sugiyama, H. Hachiya, C. Towell, and S. Vijayakumar. Value function approximation on non-linear manifolds for robot motor control. In *Proc. of the IEEE International Conference on Robotics and Automation*, 2007.

R. Sutton and A. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

G. Tesauro. Temporal difference learning and TD-Gammon. *Commun. ACM*, 38(3):58–68, 1995. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/203330.203343.

J. Tobias and P. Daniel. Least squares svm for least squares td learning. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI*, pages 499–503. IOS Press, 2006. ISBN 1-58603-642-4.

M. A. Trick and S. E. Zin. Spline Approximations to Value Functions. *Macroeconomic Dynamics*, 1:255?–277, January 1997.

M. Valenti. *Approximate Dynamic Programming with Applications in Multi-Agent Systems*. PhD thesis, Massachusetts Institute of Technology, 2007.

G. Wahba. Spline Models for Observational Data. In *Vol. 59 of CBMS-NSF Regional Conference Series in Applied Mathematics*, SIAM, Philidelphia, 1990.