Approximate Dynamic Programming Using Bellman Residual Elimination and Gaussian Process Regression

2009 American Control Conference
Hyatt Regency Riverfront, St. Louis, MO, USA
June 10-12, 2009

Approximate Dynamic Programming Using Bellman Residual Elimination and Gaussian Process Regression

Brett Bethke and Jonathan P. How

***Abstract*— This paper presents an approximate policy iteration algorithm for solving infinite-horizon, discounted Markov decision processes (MDPs) for which a model of the system is available. The algorithm is similar in spirit to Bellman residual minimization methods. However, by using Gaussian process regression with nondegenerate kernel functions as the underlying cost-to-go function approximation architecture, the algorithm is 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 algorithm can be proven to reduce to *exact* policy iteration in the limit of sampling the entire state space. Furthermore, the algorithm can automatically optimize the choice of any free kernel parameters and provide error bounds on the resulting cost-to-go solution. Computational results on a classic reinforcement learning problem indicate that the algorithm yields a high-quality policy and cost approximation.**

I. Introduction

Markov Decision Processes (MDPs) are a powerful framework for addressing problems involving sequential decision making under uncertainty [1]. Such problems occur frequently in a number of fields, including engineering, finance, and operations research. This paper considers the general class of infinite horizon, discounted, finite state MDPs. The MDP is specified by $(\mathcal{S}, \mathcal{A}, P, g)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P_{ij}(u)$ 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. We assume that the MDP specification is fully known. 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_μ over the set of admissible policies Π:

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

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 difficult or impossible to solve exactly. To overcome this challenge, a wide variety of methods for generating approximate solutions to large MDPs have been developed, giving rise to the fields of *approximate dynamic programming* and *reinforcement learning* [2], [3].

B. Bethke is a Ph.D. Candidate, Dept. of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139 bbethke@mit.edu

J. P. How is a Professor of Aeronautics and Astronautics, MIT, jhow@mit.edu

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 an updated 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. Eq. (1) is an n-dimensional, linear system of equations, where $n = |\mathcal{S}|$ is the size of the state space $\mathcal{S}$. Because n 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 approximate dynamic programming 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:

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

where $||.||_p$ is a suitably chosen p-norm. 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 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 [2, Ch. 6]:

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

It is then practical to minimize $\widetilde{BE}$ over the set of candidate functions. This approach has been investigated by several authors [4]–[6], resulting in a class of 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 [7], linear architectures [8], splines [9], and wavelets [10] have been investigated for use in reinforcement learning. Recently, motivated by the success of kernel-based methods such as support vector machines [11] and Gaussian processes [12] for pattern classification and regression, researchers have begun applying these powerful techniques in the reinforcement learning domain (see, for example, [13]–[15]).

In [16], we proposed an approximate dynamic programming algorithm based on support vector regression that is similar in spirit to traditional Bellman residual methods. Similar to traditional methods, our algorithm is 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, our algorithm explicitly constructs a solution for which $\widetilde{BE} = 0$. As an immediate consequence, our algorithm has 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 algorithm eliminates the need to perform trajectory simulations and therefore does not suffer from simulation noise effects. We refer to this 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, [16] was the first to propose Bellman residual elimination as an approach to approximate dynamic programming.

This paper extends the work of [16] in two ways. First, in many applications, the kernel function employed may have a number of free parameters. These parameters may encode, for example, characteristic length scales in the problem. In [16], these parameters were required to be set by hand; this paper shows how they may be learned automatically by the algorithm as it runs. Second, this paper explains how error bounds on the resulting cost-to-go solution can be computed. The key to achieving these goals is the use of Gaussian process regression, instead of the support vector regression formulation used in our previous work. This paper first explains the basics of Gaussian process regression, and shows how the problem of forcing the Bellman residuals to zero can be posed as a regression problem. It then develops a complete approximate policy iteration algorithm that repeatedly solves this regression problem using Gaussian process regression, and proves that the cost-to-go function learned by the algorithm indeed satisfies $\widetilde{BE} = 0$. Finally, computational results for a classic reinforcement learning problem are presented.

II. Gaussian Process Regression Basics

In this section, Gaussian process regression [12] is reviewed. Gaussian process regression attempts to solve the following problem: given a set of data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, find a function $f(x)$ that provides a good approximation to the training 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 an 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 function $k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]$. (This covariance function is also commonly referred to as a *kernel*). 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 inputs $\{x_1, \ldots, x_n\}$, the distribution over the corresponding output variables $\{y_1, \ldots, y_n\}$ is given by a standard Gaussian distribution: $(y_1, \ldots, y_n)^T \sim \mathcal{N}(\underline{\mu}, \mathbb{K})$, where the mean $\underline{\mu}$ and covariance $\mathbb{K}$ of the distribution are obtained by sampling the mean $m(x)$ and covariance $k(x, x')$ functions of the Gaussian process at the points $\{x_1, \ldots, x_n\}$:

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

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. In particular, the mean $\bar{f}(x_\star)$ and variance $\mathbb{V}[f(x_\star)]$ of the posterior distribution at an arbitrary point $x_\star$ are given by

$$\begin{aligned} \bar{f}(x_\star) &= \underline{\lambda}^T \underline{k}_\star && (4) \\ \mathbb{V}[f(x_\star)] &= k(x_\star, x_\star) - \underline{k}_\star^T \mathbb{K}^{-1} \underline{k}_\star && (5) \end{aligned}$$

where $\underline{\lambda} = \mathbb{K}^{-1}\underline{y}$ and $\underline{k}_\star = (k(x_\star, x_1), \dots, k(x_\star, x_n))^T$.

As long as the Gram matrix $\mathbb{K}$ is strictly positive definite, a solution for $\underline{\lambda}$ can always be found, and the resulting function $\bar{f}(x_\star)$ satisfies $\bar{f}(x_i) = y_i \quad \forall i = 1, \dots, n$; that is, the learned regression function matches the training data, as desired. Finally, drawing on results from the theory of Reproducing Kernel Hilbert Spaces (RKHSs) [17], we can equivalently express the result as

$$\bar{f}(x_\star) = \langle \underline{\Theta}, \underline{\Phi}(x_\star) \rangle, \tag{6}$$

where $\underline{\Phi}(x_\star) = k(\cdot, x_\star)$ is the so-called reproducing kernel map of the kernel k, $\underline{\Theta} = \sum_{i=1}^n \lambda_i \underline{\Phi}(x_i)$ is a weighting factor learned by carrying out Gaussian process regression, and $\langle \cdot, \cdot \rangle$ denotes the inner product in the RKHS of k. The representation given by Eq. (6) will be important for constructing our Bellman residual elimination (BRE) algorithm.

Marginal Likelihood and Kernel Parameter Selection

In many cases, the covariance (kernel) function $k(i, i')$ depends on a set of parameters $\underline{\Omega}$. 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 [12, 5.4]

$$\log p(\underline{y}|\underline{\Omega}) = -\frac{1}{2}\underline{y}^T \mathbb{K}^{-1} \underline{y} - \frac{1}{2}\log|\mathbb{K}| - \frac{n}{2}\log 2\pi. \tag{7}$$

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. (7)]. The derivative of the likelihood with respect to the individual parameters Ω_j can be calculated analytically [12, 5.4]:

$$\frac{\partial \log p(\underline{y}|\underline{\Omega})}{\partial \Omega_j} = \frac{1}{2}\mathrm{tr}\left((\underline{\lambda}\underline{\lambda}^T - \mathbb{K}^{-1}) \frac{\partial \mathbb{K}}{\partial \Omega_j} \right). \tag{8}$$

Eq. (8) allows the use of any gradient-based optimization method to select the optimal values for the parameters $\underline{\Omega}$.

III. Formulating BRE as a Regression Problem

This section explains how policy evaluation can be performed by formulating BRE as a regression problem, which can be subsequently solved using Gaussian process regression. The problem statement is as follows: assume that an MDP specification $(\mathcal{S}, \mathcal{A}, P, g)$ is given, along with a fixed policy μ of the MDP. Furthermore, assume that a representative set of sample states $\widetilde{\mathcal{S}}$ is given. The goal is to construct an approximate cost-to-go $\widetilde{J}_\mu$ of the policy μ, such that the Bellman residuals at all of the sample states are identically zero.

To begin, according to the representation given in the previous section [Eq. (6)], the approximate cost-to-go function $\widetilde{J}_\mu(i)$ is expressed as an inner product between a feature mapping $\underline{\Phi}(i)$ and a set of weights $\underline{\Theta}$:

$$\widetilde{J}_\mu(i) = \langle \underline{\Theta}, \underline{\Phi}(i) \rangle, \qquad i \in \mathcal{S}. \tag{9}$$

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

$$k(i, i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle, \quad i, i' \in \mathcal{S}. \tag{10}$$

Now, recall that the Bellman residual at state i, denoted by $BR(i)$, is defined as

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

where $g_i^\mu \equiv g(i, \mu(i))$ and $P_{ij}^\mu \equiv P_{ij}(\mu(i))$. Substituting the functional form of the cost function, Eq. (9), 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

$$\begin{aligned} BR(i) &= -g_i^\mu + \langle \underline{\Theta}, \left(\underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \underline{\Phi}(j) \right) \rangle \\ &= -g_i^\mu + \langle \underline{\Theta}, \underline{\Psi}(i) \rangle, && (11) \end{aligned}$$

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

$$\underline{\Psi}(i) \equiv \underline{\Phi}(i) - \alpha \sum_{j \in \mathcal{S}} P_{ij}^\mu \underline{\Phi}(j). \tag{12}$$

Intuitively, $\underline{\Psi}(i)$ represents a new reproducing kernel map 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 under the policy μ). The definition of $\underline{\Psi}(i)$ and the corresponding expression for the Bellman residual [Eq. (11)] now allow the BRE problem to be formulated as a regression problem. To specify the regression problem, the set of training data $\mathcal{D}$ must be specified, along with the kernel function to use. Notice that, from Eq. (11), the desired condition $BR(i) = 0$ is equivalent to

$$\widetilde{W}_\mu(i) \equiv \langle \underline{\Theta}, \underline{\Psi}(i) \rangle = g_i^\mu. \tag{13}$$

Therefore, the regression problem is to find a function $\widetilde{W}_\mu(i)$ that satisfies $\widetilde{W}_\mu(i) = g_i^\mu \quad \forall i \in \widetilde{\mathcal{S}}$, and the training data of the problem is given by $\mathcal{D} = \{\underbrace{(i, g_i^\mu)} \mid i \in \widetilde{\mathcal{S}}\}$. Furthermore, given the functional form of $\widetilde{W}_\mu(i)$, the kernel function in the regression problem is given by

$$\mathcal{K}(i,i') = \langle \underline{\Psi}(i), \underline{\Psi}(i') \rangle. \tag{14}$$

We shall refer to $\mathcal{K}(i,i')$ as the *Bellman kernel associated with* $k(i,i')$. By substituting Eq. (12) into Eq. (14) and recalling that the original kernel k can be expressed as $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$, we obtain the following expression for $\mathcal{K}(i,i')$:

$$\mathcal{K}(i,i') = k(i,i') - \alpha \sum_{j \in \mathcal{S}} \Big(P^\mu_{i'j} k(i,j) + P^\mu_{ij} k(i',j) \Big) + \alpha^2 \sum_{j,j' \in \mathcal{S}} P^\mu_{ij} P^\mu_{i'j'} k(j,j'). \tag{15}$$

With the associated Bellman kernel $\mathcal{K}(i,i')$ and the training data $\mathcal{D}$ specified, the BRE regression problem is completely defined. In order to solve the problem using Gaussian process regression, the Gram matrix $\mathbb{K}$ of the associated Bellman kernel is calculated (where $\mathbb{K}_{ii'} = \mathcal{K}(i,i')$), and the λ_i values are found by solving $\underline{\lambda} = \mathbb{K}^{-1} \underline{g}^\mu$. Once $\underline{\lambda}$ is known, the corresponding weight element $\underline{\Theta}$ is given by $\underline{\Theta} = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \underline{\Psi}(i)$. Note that $\underline{\Theta}$ is a linear combination of $\underline{\Psi}(i)$ vectors (instead of $\underline{\Phi}(i)$ vectors), since the regression problem is being solved using the associated Bellman kernel $\mathcal{K}(i,i')$ instead of the original kernel $k(i,i')$. Substituting $\underline{\Theta}$ into the expression for $\widetilde{W}_\mu(i)$ [Eq. (13)] and using Eq. (14), we obtain $\widetilde{W}_\mu(i) = \sum_{i \in \widetilde{\mathcal{S}}} \lambda_i \mathcal{K}(i,i')$. Finally, the corresponding cost function $\widetilde{J}_\mu(i)$ is found using $\underline{\Theta}$ and the expression for $\underline{\Psi}(i)$ [Eq. (12)]:

$$\begin{aligned}
\widetilde{J}_\mu(i) &= \langle \underline{\Theta}, \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \langle \underline{\Psi}(i'), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \langle \left(\underline{\Phi}(i') - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} \underline{\Phi}(j) \right), \underline{\Phi}(i) \rangle \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(\langle \underline{\Phi}(i'), \underline{\Phi}(i) \rangle - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} \langle \underline{\Phi}(j), \underline{\Phi}(i) \rangle \right) \\
&= \sum_{i' \in \widetilde{\mathcal{S}}} \lambda_{i'} \left(k(i',i) - \alpha \sum_{j \in \mathcal{S}} P^\mu_{i'j} k(j,i) \right).
\end{aligned} \tag{16}$$

IV. Approximate Policy Iteration Algorithm Using BRE

The previous section showed how the problem of eliminating the Bellman residuals at the set of sample states $\widetilde{\mathcal{S}}$ is equivalent to solving a regression problem, and how to compute the corresponding cost function once the regression problem is solved. We now present the main policy iteration algorithm, denoted by BRE(GP), which carries out BRE using Gaussian process regression. 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 error bounds on the cost-to-go solution.

Pseudocode for BRE(GP) is shown in Algorithm 1. The algorithm takes an initial policy μ_0, a set of sample states $\widetilde{\mathcal{S}}$, and a kernel $k(i,i';\underline{\Omega})$ as input. In addition, it takes a set of initial kernel parameters $\underline{\Omega} \in \mathbf{\Omega}$. 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 main 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 $\underline{g}$, are computed in line 10) and adjusting the kernel parameters using a gradient-based approach. 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 inverts the Gram matrix $\mathbb{K}(\Omega)$ to find $\underline{\lambda}$. Lines 14 and 15 then compute the gradient of the log likelihood function, using Eq. (8), 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, 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. (16) (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 directly from Eq. (5), and it gives a Bayesian error bound on the quality of the produced cost-to-go function: for states where $E(i)$ is small, the Bellman residual $BR(i)$ is likely to be small as well. Finally, it carries out a policy improvement step.

Theoretical Properties

In this section, we prove that the cost functions $\widetilde{J}_\mu(i)$ generated by BRE(GP) have zero Bellman residuals at the sample states, and as a result, BRE(GP) reduces to exact policy iteration when the entire state space is sampled. The concept of a *nondegenerate kernel* is important in the proofs to follow, and is stated here:

Definition A *nondegenerate kernel* $k(i,i') = \langle \underline{\Phi}(i), \underline{\Phi}(i') \rangle$ is defined as a kernel for which the Gram matrix $\mathbb{K}$, where $\mathbb{K}_{ii'} = k(i,i'), \quad i,i' \in \widetilde{\mathcal{S}}$, is strictly positive definite for every set $\widetilde{\mathcal{S}}$. Equivalently, a nondegenerate kernel is one for which the feature vectors $\{\underline{\Phi}(i) \mid i \in \widetilde{\mathcal{S}}\}$ are always linearly independent.

To begin, the following lemma states an important property of the mapping $\underline{\Psi}(i)$ that will be used in later 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 $\hat{A}$ that maps $\underline{\Phi}(i)$ to $\underline{\Psi}(i)$ can be defined: $\underline{\Psi}(i) \equiv \hat{A}\underline{\Phi}(i) = (I - \alpha P^{\mu})\underline{\Phi}(i)$. Here, 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, $\hat{A} = I - \alpha P^{\mu}$ is full rank and $dim(ker(\hat{A})) = 0$. Therefore, $dim(\{\underline{\Psi}(i) \mid i \in \mathcal{S}\}) = dim(\{\underline{\Phi}(i) \mid i \in \mathcal{S}\}) = n_s$, 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, by definition 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 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: 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 thus invertible. It follows that, in Line 13 of the algorithm, there is a unique solution for $\underline{\lambda}$. Therefore, the regression solution $\widetilde{W}_{\mu}(i)$ computed using Gaussian process regression matches the training data: $\widetilde{W}_{\mu}(i) = g_i^{\mu} \quad \forall i \in \widetilde{\mathcal{S}}$. Eq. (13) shows that this is equivalent to $BR(i) = 0 \quad \forall i \in \widetilde{\mathcal{S}}$, as desired. ■

When the entire state space is sampled ($\widetilde{\mathcal{S}} = \mathcal{S}$), two immediate and important corollaries of Theorem 3 follow:

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 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: This follows immediately from the fact that $BR(i) = 0 \quad \forall i \in \mathcal{S}$. ■

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(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. ■

Algorithm 1 BRE(GP)

```
1: **Input:** $(\mu_0, \widetilde{\mathcal{S}}, k, \underline{\Omega})$
2: $\mu_0$: initial policy
3: $\widetilde{\mathcal{S}}$: set of sample states
4: $k$: kernel (covariance) function defined on $\mathcal{S} \times \mathcal{S} \times \mathbf{\Omega}$,
   $k(i, i'; \underline{\Omega}) = \langle \underline{\Phi}(i; \underline{\Omega}), \underline{\Phi}(i'; \underline{\Omega}) \rangle$
5: $\underline{\Omega}$: initial set of kernel parameters
6: **Begin**
7: Define $\mathcal{K}(i, i'; \underline{\Omega}) = \langle \underline{\Psi}(i; \underline{\Omega}), \underline{\Psi}(i'; \underline{\Omega}) \rangle$ {Define the
   associated Bellman kernel}
8: $\mu \leftarrow \mu_0$
9: **loop**
10:    Construct $\underline{g}$, the vector of stage costs $g_i^{\mu} \quad \forall i \in \widetilde{\mathcal{S}}$
11:    **repeat**
12:       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. (15)
13:       Solve $\underline{\lambda} = \mathbb{K}(\underline{\Omega})^{-1}\underline{g}$
14:       Calculate the gradient of the log marginal likeli-
          hood, $\nabla_{\underline{\Omega}} \log p(\underline{g}|\widetilde{\mathcal{S}}, \underline{\Omega})$, where
          $$\frac{\partial \log p(\underline{g}|\widetilde{\mathcal{S}}, \underline{\Omega})}{\partial \Omega_j} = \frac{1}{2}\mathrm{tr}\left( (\underline{\lambda}\underline{\lambda}^T - \mathbb{K}(\underline{\Omega})^{-1}) \frac{\partial \mathbb{K}(\underline{\Omega})}{\partial \Omega_j} \right).$$
15:       Update the kernel parameters using any gradient-
          based optimization rule:
          $$\underline{\Omega} \leftarrow \underline{\Omega} + \gamma \nabla_{\underline{\Omega}} \log p(\underline{g}|\widetilde{\mathcal{S}}, \underline{\Omega}),$$
          where $\gamma$ is an appropriately selected step size
16:    **until** stopping condition for gradient-based optimiza-
       tion rule is met
17:    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}
18:    Compute $E(i)$, the 1-$\sigma$ 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 $h_j \equiv \mathcal{K}(i, j; \underline{\Omega})$
19:    $\mu(i) \leftarrow \arg\min_u \sum_{j \in \mathcal{S}} P_{ij}(u) \left( g(i, u) + \alpha \widetilde{J}_{\mu}(j) \right)$
       {Policy improvement}
20: **end loop**
21: **End**
```

V. Computational Results

BRE(GP) was implemented on the well-known "mountain car problem" [3], [14] to evaluate its performance. In this problem, a unit mass, frictionless car moves along a hilly landscape whose height $H(x)$ is described by

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

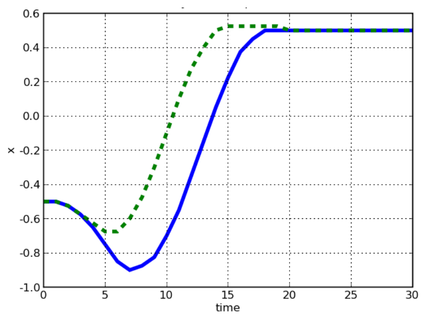

Fig. 1: System response under the optimal policy (dashed line) and the policy learned by the support vector policy iteration algorithm (solid line).

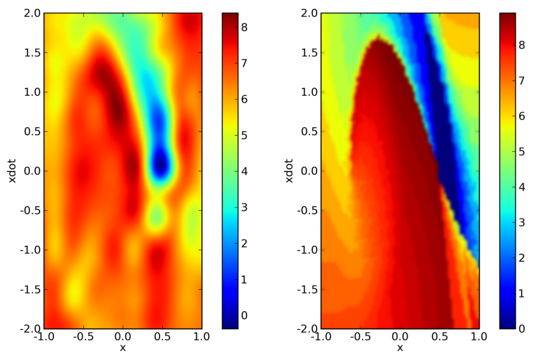

Fig. 2: Approximate cost-to-go produced by BRE(GP) (left); exact cost-to-go (right).

The system state is given by $(x, \dot{x})$ (the position and speed of the car). A horizontal control force $-4 \le u \le 4$ 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" $0.5 \le x \le 0.7$ as quickly as possible. 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 BRE(GP), an evenly spaced 9x9 grid of sample states was chosen. Furthermore, a squared exponential kernel $k((x_1, \dot{x}_1), (x_2, \dot{x}_2); \underline{\Omega}) = \exp\left(-(x_1 - x_2)^2/\Omega_1^2 - (\dot{x}_1 - \dot{x}_2)^2/\Omega_2^2\right)$ was used; here the parameters Ω_1 and Ω_2 represent characteristic length-scales in each of the two dimensions of the state space. BRE(GP) 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 (computed using value iteration) for comparison. The cost functions are shown after the kernel parameters were optimally adjusted for each policy; the final kernel parameters were $\Omega_1 = 0.253$ and $\Omega_2 = 0.572$. 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.

VI. Conclusion

This paper has extended the work presented in [16] by designing a Bellman residual elimination algorithm, BRE(GP), that automatically optimizes the choice of kernel parameters and provides error bounds on the resulting cost-to-go solution. This is made possible by using Gaussian process regression to solve the BRE regression problem. The BRE(GP) algorithm has a number of desirable theoretical properties, including being provably exact in the limit of sampling the entire state space. Application to a classic reinforcement learning problem indicate the algorithm yields a high-quality policy and cost approximation.

VII. Acknowledgments

The first author is sponsored by the Hertz Foundation and the American Society for Engineering Education. The research has also been supported by the Boeing Company and by AFOSR grant FA9550-08-1-0086.

References

[1] D. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 2007.

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

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

[4] P. Schweitzer and A. Seidman, "Generalized polynomial approximation in Markovian decision processes," *Journal of mathematical analysis and applications*, vol. 110, pp. 568–582, 1985.

[5] L. C. Baird, "Residual algorithms: Reinforcement learning with function approximation." in *ICML*, 1995, pp. 30–37.

[6] 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*, vol. 71, no. 1, pp. 89–129, 2008.

[7] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[8] M. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[9] M. A. Trick and S. E. Zin, "Spline Approximations to Value Functions," *Macroeconomic Dynamics*, vol. 1, pp. 255?–277, January 1997.

[10] M. Maggioni and S. Mahadevan, "Fast direct policy evaluation using multiscale analysis of markov diffusion processes." in *ICML*, ser. ACM International Conference Proceeding Series, W. W. Cohen and A. Moore, Eds., vol. 148. ACM, 2006, pp. 601–608.

[11] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," in *Knowledge Discovery and Data Mining*, no. 2, 1998.

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

[13] T. Dietterich and X. Wang, "Batch value function approximation via support vectors." in *NIPS*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2001, pp. 1491–1498.

[14] C. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 16, pp. 751–759, 2004.

[15] Y. Engel, "Algorithms and representations for reinforcement learning," Ph.D. dissertation, Hebrew University, 2005.

[16] B. Bethke, J. How, and A. Ozdaglar, "Approximate Dynamic Programming Using Support Vector Regression," in *Proceedings of the 2008 IEEE Conference on Decision and Control*, Cancun, Mexico, 2008.

[17] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, vol. 68, pp. 337–404, 1950.