CS–2001–05

Model-Free Least-Squares Policy Iteration[1]

Michail G. Lagoudakis Ronald Parr

Department of Computer Science

Duke University

Durham, North Carolina 27708–0129

December 2001

[1]A shorter version of this technical report has been accepted to *NIPS*2001: Neural Information Processing Systems: Natural and Synthetic.*

Model-Free Least-Squares Policy Iteration

Michail G. Lagoudakis and **Ronald Parr**

Department of Computer Science
Duke University
Durham, NC 27708

{mgl,parr}@cs.duke.edu

Abstract

We propose a new approach to reinforcement learning which combines least squares function approximation with policy iteration. Our method is model-free and completely off policy. We are motivated by the least squares temporal difference learning algorithm (LSTD), which is known for its efficient use of sample experiences compared to pure temporal difference algorithms. LSTD is ideal for prediction problems, however it heretofore has not had a straightforward application to control problems. Moreover, approximations learned by LSTD are strongly influenced by the visitation distribution over states. Our new algorithm, Least-Squares Policy Iteration (LSPI) addresses these issues. The result is an off-policy method which can use (or reuse) data collected from any source. We test LSPI on several problems, including a bicycle simulator in which it learns to guide the bicycle to a goal efficiently by merely observing a relatively small number of completely random trials.

1 Introduction

Linear least squares function approximators offer many advantages in the context of reinforcement learning. While their ability to generalize is less powerful than black box methods such as neural networks, they have their virtues: They are easy to implement and use, and their behavior is fairly transparent, both from an analysis standpoint and from a debugging and feature engineering standpoint. When linear methods fail, it is usually relatively easy to get some insight into why the failure has occurred.

Our enthusiasm for this approach is inspired by the least squares temporal difference learning algorithm (LSTD) [4]. The LSTD algorithm makes efficient use of data and converges faster than other conventional temporal difference learning methods. Although it is initially appealing to attempt to use LSTD in the evaluation step of a policy iteration algorithm, this combination can be problematic. Koller and Parr [5] present an example where the combination of LSTD style function approximation and policy iteration oscillates between two bad policies in an MDP with just 4 states. This behavior is explained by the fact that linear approximation methods such as LSTD compute an approximation that is weighted by the state visitation frequencies of the policy under evaluation[1]. Further, even if this problem is overcome, a more serious difficulty is that the state value function that LSTD learns is of no use for policy improvement when a model of the process is not available.

This paper introduces the *Least Squares Policy Iteration (LSPI)* algorithm, which extends the benefits of LSTD to control problems. First, we introduce LSQ, an algorithm that learns least squares approximations of the *state-action* (Q) value function, thus permitting action selection and policy improvement *without* a model. Next we introduce LSPI which uses the results of LSQ to form an approximate policy iteration algorithm. This algorithm combines the policy search efficiency of policy iteration with the data efficiency of LSTD. It is completely off policy and can, in principle, use data collected from any reasonable sampling distribution. We evaluate this method on several problems, including a simulated bicycle control problem in which LSPI learns to guide the bicycle to the goal by observing a relatively small number of completely random trials.

[1] Recent work by Sutton and Precup [11] addresses the question of learning a value function for one policy while following another stochastic policy. It does not directly address the question of how the policies are generated.

2 Markov Decision Processes

We assume that the underlying control problem is a *Markov Decision Process* (MDP). An MDP is defined as a 4-tuple $(S, \mathcal{A}, P, R)$ where: S is a finite set of states; $\mathcal{A}$ is a finite set of actions; P is a *Markovian transition model* where $P(s, a, s')$ represents the probability of going from state s to state s' with action a; and R is a *reward function* $R : S \times \mathcal{A} \times S \mapsto \mathbb{R}$, such that $R(s, a, s')$ represents the reward obtained when taking action a in state s and ending up in state s'. For simplicity of notation, we use define $R : S \times \mathcal{A} \mapsto \mathbb{R}$. Our results easily extend to the more general case.

We will be assuming that the MDP has an infinite horizon and that future rewards are discounted exponentially with a discount factor $\gamma \in [0, 1)$ (assuming that all policies are proper, our results generalize to the undiscounted case as well). A stationary deterministic policy π for an MDP is a mapping $\pi : S \mapsto A$, where $\pi(s)$ is the action the agent takes at state s. The state-action value function $Q^\pi(s, a)$ is defined over all possible combinations of states and actions and indicates the expected, discounted total reward when taking action a in state s and following policy π thereafter. The exact Q-values for all state-action pairs can be found by solving the linear system of the Bellman equations:

$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s'} P(s, a, s') Q^\pi(s', \pi(s')).$$

where Q^π and $\mathcal{R}$ are vectors of size $|S||\mathcal{A}|$ and $\mathbf{P}^\pi$ is a stochastic matrix of size $(|S||\mathcal{A}| \times |S||\mathcal{A}|)$. $\mathcal{R}$ is the expected reward for state-action pairs, $\mathcal{R}(s, a) = \sum_{s'} P(s, a, s') R(s, a, s')$. In matrix format, the system becomes

$$Q^\pi = \mathcal{R} + \gamma \mathbf{P}^\pi Q^\pi,$$

where Q^π and $\mathcal{R}$ are vectors of size $|S||A|$ and $\mathbf{P}^\pi$ is a stochastic matrix of size $(|S||A| \times |S||A|)$. $\mathbf{P}^\pi$ describes the probability of transitions from pairs (s, a) to pairs $(s', \pi(s'))$.

For every MDP, there exists an optimal policy, π^*, which maximizes the expected, discounted return of every state. *Policy iteration* is a method of discovering this policy by iterating through a sequence of monotonically improving policies. Each iteration consists of two phases. *Value determination* computes the state-action values for a policy $\pi^{(t)}$ by solving the above system. *Policy improvement* defines the next policy as $\pi^{(t+1)}(s) = \arg\max_a Q^{\pi^{(t)}}(s, a)$. These steps are repeated until convergence to an optimal policy, often in a surprisingly small number of steps.

3 Least-Squares Approximation of Q Functions

Policy iteration relies upon the solution of a system of linear equations to find the Q values for the current policy. This is impractical for large state and action spaces. In such cases we may wish to approximate Q^π with a parametric function approximator and do some form of approximate policy iteration. We now address the problem of finding a set of parameters that maximizes the accuracy of our approximator. A common class of approximators is the so called *linear architectures*, where the Q values are approximated as a linear weighted combination of k basis functions (features):

$$\widehat{Q}^\pi(s,a,w) = \sum_{i=1}^{k} \phi_i(s,a) w_i = \phi(s,a)^\intercal w,$$

where w is a set of weights (parameters). In general, $k << |S||A|$ and so, the linear system above now becomes an overconstrained system over the k parameters w :

$$\begin{aligned} \mathbf{\Phi} w &\approx \mathcal{R} + \gamma \mathbf{P}^\pi \mathbf{\Phi} w \\ (\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi}) w &\approx \mathcal{R} \end{aligned}$$

where $\mathbf{\Phi}$ is a $(|S||A| \times k)$ matrix, of the form

$$\mathbf{\Phi} = \begin{pmatrix} \phi(s_1,a_1)^\intercal \\ \cdots \\ \phi(s,a)^\intercal \\ \cdots \\ \phi(s_{|S|},a_{|A|})^\intercal \end{pmatrix}.$$

Solving this system in the least-squares sense will yield a solution

$$w = ((\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi})^\intercal (\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi}))^{-1} (\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi})^\intercal \mathcal{R}$$

that minimizes the Bellman error, but the resulting value function will not necessarily be a fixed point under one application of the Bellman equation and subsequent orthogonal projection. We note that this approach was proposed as a means of estimating state value functions given a model in [13].

We are interested in a set of weights w^π that yields a fixed point in value function space, that is a value function $\widehat{Q}^\pi = \mathbf{\Phi} w^\pi$ that is invariant under one step of value determination followed by orthogonal projection to the

space spanned by the basis functions. The value function Q_{temp} that results from one step of value iteration is

$$\widehat{Q}_{\text{temp}} = \mathcal{R} + \gamma \mathbf{P}^{\pi} \mathbf{\Phi} w^{\pi}.$$

The orthogonal projection of $\widehat{Q}_{\text{temp}}$ to the space spanned by the columns of Φ (assuming that the columns of $\mathbf{\Phi}$ are independent) is

$$\widehat{Q}_{\text{proj}} = \mathbf{\Phi}(\mathbf{\Phi}^{\intercal}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\intercal}\widehat{Q}_{\text{temp}}.$$

$\widehat{Q}_{\text{proj}}$ must be equal to the original $\widehat{Q}^{\pi}$:

$$\begin{aligned}
\widehat{Q}_{\text{proj}} &= \widehat{Q}^{\pi} \\
\mathbf{\Phi}(\mathbf{\Phi}^{\intercal}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\intercal}\widehat{Q}_{\text{temp}} &= \widehat{Q}^{\pi} \\
\mathbf{\Phi}(\mathbf{\Phi}^{\intercal}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\intercal}(\mathcal{R} + \gamma \mathbf{P}^{\pi}\mathbf{\Phi} w^{\pi}) &= \mathbf{\Phi} w^{\pi} \\
\mathbf{\Phi}\left((\mathbf{\Phi}^{\intercal}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\intercal}(\mathcal{R} + \gamma \mathbf{P}^{\pi}\mathbf{\Phi} w^{\pi}) - w^{\pi}\right) &= 0 \\
(\mathbf{\Phi}^{\intercal}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\intercal}(\mathcal{R} + \gamma \mathbf{P}^{\pi}\mathbf{\Phi} w^{\pi}) - w^{\pi} &= 0 \\
(\mathbf{\Phi}^{\intercal}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\intercal}(\mathcal{R} + \gamma \mathbf{P}^{\pi}\mathbf{\Phi} w^{\pi}) &= w^{\pi} \\
\mathbf{\Phi}^{\intercal}(\mathcal{R} + \gamma \mathbf{P}^{\pi}\mathbf{\Phi} w^{\pi}) &= \mathbf{\Phi}^{\intercal}\mathbf{\Phi} w^{\pi} \\
\mathbf{\Phi}^{\intercal}(\mathbf{\Phi} - \gamma \mathbf{P}^{\pi}\mathbf{\Phi}) w^{\pi} &= \mathbf{\Phi}^{\intercal}\mathcal{R} \\
w^{\pi} &= \left(\mathbf{\Phi}^{\intercal}(\mathbf{\Phi} - \gamma \mathbf{P}^{\pi}\mathbf{\Phi})\right)^{-1}\mathbf{\Phi}^{\intercal}\mathcal{R}.
\end{aligned}$$

We note that this is the standard fixed point approximation method for linear value functions with the exception that the problem is formulated in terms of Q values instead of state values. For any $\mathbf{P}^{\pi}$, the solution is guaranteed to exist for all but finitely many values of γ [5].

4 LSQ: Learning the State-Action Value Function

In the previous section we assumed that a model $(R, \mathbf{P}^{\pi})$ of the underlying MDP is available. In many practical applications, such a model is not available and the value function or, more precisely, its parameters have to be learned from sampled data. These sampled data are tuples of the form:

$$(s, a, r, s')$$

meaning that in state s, action a was taken, a reward r was received, and the resulting state was s'. These data can be collected from actual (sequential) episodes or from queries to a generative model of the MDP. In the extreme

case, they can be experiences of other agents on the same MDP. We know that the desired set of weights can be found as the solution of the system,

$$\mathbf{A}w^{\pi} = b,$$

where

$$\mathbf{A} = \mathbf{\Phi}^{\intercal}(\mathbf{\Phi} - \gamma \mathbf{P}^{\pi}\mathbf{\Phi}) \qquad \text{and} \qquad b = \mathbf{\Phi}^{\intercal}\mathcal{R}.$$

The matrix $\mathbf{P}^{\pi}$ and the vector $\mathcal{R}$ are unknown and so, $\mathbf{A}$ and b cannot be determined *a priori*. However, $\mathbf{A}$ and b can be approximated using samples. Recall that $\mathbf{\Phi}$, $\mathbf{P}^{\pi}\mathbf{\Phi}$, and $\mathcal{R}$ are of the form

$$\mathbf{\Phi} = \begin{pmatrix} \phi(s_1, a_1)^{\intercal} \\ \dots \\ \phi(s, a)^{\intercal} \\ \dots \\ \phi(s_{|S|}, a_{|A|})^{\intercal} \end{pmatrix}$$

$$\mathbf{P}^{\pi}\mathbf{\Phi} = \begin{pmatrix} \sum_{s'} P(s_1, a_1, s')\phi(s', \pi(s'))^{\intercal} \\ \dots \\ \sum_{s'} P(s, a, s')\phi(s', \pi(s'))^{\intercal} \\ \dots \\ \sum_{s'} P(s_{|S|}, a_{|A|}, s')\phi(s', \pi(s'))^{\intercal} \end{pmatrix}$$

$$\mathcal{R} = \begin{pmatrix} \sum_{s'} P(s_1, a_1, s')R(s_1, a_1, s') \\ \dots \\ \sum_{s'} P(s, a, s')R(s, a, s') \\ \dots \\ \sum_{s'} P(s_{|S|}, a_{|A|}, s')R(s_{|S|}, a_{|A|}, s') \end{pmatrix}$$

Given a set of samples, $D = \{s_{d_i}, a_{d_i}, s'_{d_i}, r_{d_i})|\ i = 1, 2, \dots, L\}$, where the (s_{d_i}, a_{d_i}) are sampled from $S \times \mathcal{A}$ according to distribution ρ and the s'_{d_i} are sampled according to $P(s'_{d_i}|s_{d_i}, a_{d_i})$, we can construct approximate versions of $\mathbf{\Phi}$, $\mathbf{P}^{\pi}\mathbf{\Phi}$, and $\mathcal{R}$ as follows :

$$\widehat{\mathbf{\Phi}} = \begin{pmatrix} \phi(s_{d_1}, a_{d_1})^{\intercal} \\ \dots \\ \phi(s_{d_i}, a_{d_i})^{\intercal} \\ \dots \\ \phi(s_{d_L}, a_{d_L})^{\intercal} \end{pmatrix} \qquad \widehat{\mathbf{P}^{\pi}\mathbf{\Phi}} = \begin{pmatrix} \phi(s'_{d_1}, \pi\left(s'_{d_1}\right))^{\intercal} \\ \dots \\ \phi(s'_{d_i}, \pi\left(s'_{d_i}\right))^{\intercal} \\ \dots \\ \phi(s'_{d_L}, \pi\left(s'_{d_L}\right))^{\intercal} \end{pmatrix} \qquad \widehat{\mathcal{R}} = \begin{pmatrix} r_{d_1} \\ \dots \\ r_{d_i} \\ \dots \\ r_{d_L} \end{pmatrix}$$

These approximations can be thought of as first sampling rows from $\mathbf{\Phi}$ according to ρ and then, conditioned on these samples, as sampling terms from the summations in the corresponding rows of $\mathbf{P}^\pi\mathbf{\Phi}$ and $\mathcal{R}$. The sampling distribution from the summations is governed by the underlying dynamics ($P(s,a,s')$) of the process as the samples in D are taken directly from the MDP.

Given $\widehat{\mathbf{\Phi}}$, $\widehat{\mathbf{P}^\pi\mathbf{\Phi}}$, and $\widehat{\mathcal{R}}$, $\mathbf{A}$ and b can be approximated as

$$\widehat{\mathbf{A}} = \widehat{\mathbf{\Phi}}^\intercal(\widehat{\mathbf{\Phi}} - \gamma\widehat{\mathbf{P}^\pi\mathbf{\Phi}}) \qquad \text{and} \qquad \widehat{b} = \widehat{\mathbf{\Phi}}^\intercal\widehat{\mathcal{R}}$$

With L uniformly distributed samples over pairs of states and actions (s,a), the approximations $\widehat{\mathbf{A}}$ and $\widehat{b}$ are consistent approximations of the true $\mathbf{A}$ and b :

$$E(\widehat{\mathbf{A}}) = \frac{L}{|S||\mathcal{A}|}\mathbf{A} \qquad \text{and} \qquad E(\widehat{b}) = \frac{L}{|S||\mathcal{A}|}b$$

The Markov property ensures that the solution $\widehat{w}^\pi$ will converge to the true solution w^π for sufficiently large L whenever w^π exists:

$$E(\widehat{w}^\pi) = E(\widehat{\mathbf{A}}^{-1}\widehat{b}) = (\frac{L}{|S||\mathcal{A}|}\mathbf{A})^{-1}(\frac{L}{|S||\mathcal{A}|}b) = \mathbf{A}^{-1}b = w^\pi$$

In the more general case, where ρ is not uniform, we will compute a weighted projection, which minimizes the ρ weighted distance in the projection step. Thus, state s is implicitly assigned weight $\rho(s)$ and the projection minimizes the weighted sum of squared errors with respect to ρ. In LSTD, for example, ρ is the stationary distribution of $\mathbf{P}^\pi$, giving high weight to frequently visited states, and low weight to infrequently visited states.

As with LSTD, it's easy to see that approximations $(\widehat{\mathbf{A}}_1, \widehat{b}_1, \widehat{\mathbf{A}}_2, \widehat{b}_2)$ derived from different sets of samples (D_1, D_2) can be combined additively to yield a better approximation $(\widehat{\mathbf{A}}, \widehat{b})$ that corresponds to the combined set of samples $(D = D_1 \cup D_2)$:

$$\begin{aligned}
\widehat{\mathbf{A}} &= \widehat{\mathbf{\Phi}}^\intercal(\widehat{\mathbf{\Phi}} - \gamma\widehat{\mathbf{P}^\pi\mathbf{\Phi}}) \\
&= \begin{pmatrix} \widehat{\mathbf{\Phi}}_1 \\ \widehat{\mathbf{\Phi}}_2 \end{pmatrix}^\intercal \left(\begin{pmatrix} \widehat{\mathbf{\Phi}}_1 \\ \widehat{\mathbf{\Phi}}_2 \end{pmatrix} - \gamma \begin{pmatrix} \widehat{\mathbf{P}^\pi\mathbf{\Phi}}_1 \\ \widehat{\mathbf{P}^\pi\mathbf{\Phi}}_2 \end{pmatrix} \right) \\
&= \begin{pmatrix} \widehat{\mathbf{\Phi}}_1^\intercal & \widehat{\mathbf{\Phi}}_2^\intercal \end{pmatrix} \begin{pmatrix} \widehat{\mathbf{\Phi}}_1 - \gamma\widehat{\mathbf{P}^\pi\mathbf{\Phi}}_1 \\ \widehat{\mathbf{\Phi}}_2 - \gamma\widehat{\mathbf{P}^\pi\mathbf{\Phi}}_2 \end{pmatrix} \\
&= \widehat{\mathbf{\Phi}}_1^\intercal(\widehat{\mathbf{\Phi}}_1 - \gamma\widehat{\mathbf{P}^\pi\mathbf{\Phi}}_1) + \widehat{\mathbf{\Phi}}_2^\intercal(\widehat{\mathbf{\Phi}}_2 - \gamma\widehat{\mathbf{P}^\pi\mathbf{\Phi}}_2) \\
&= \widehat{\mathbf{A}}_1 + \widehat{\mathbf{A}}_2
\end{aligned}$$

$$\widehat{b} = \mathbf{\Phi}^\intercal \widehat{\mathcal{R}} = \begin{pmatrix} \widehat{\mathbf{\Phi}}_1 \\ \widehat{\mathbf{\Phi}}_2 \end{pmatrix}^\intercal \begin{pmatrix} \widehat{\mathcal{R}}_1 \\ \widehat{\mathcal{R}}_2 \end{pmatrix} = \widehat{\mathbf{\Phi}}_1^\intercal \widehat{\mathcal{R}}_1 + \widehat{\mathbf{\Phi}}_2^\intercal \widehat{\mathcal{R}}_2 = \widehat{b}_1 + \widehat{b}_2$$

This observation leads to an incremental update rule for $\widehat{\mathbf{A}}$ and $\widehat{b}$. Assume that initially $\widehat{\mathbf{A}} = 0$ and $\widehat{b} = 0$. For a fixed policy, a new sample (s, a, r, s') contributes to the approximation according to the following update equation:

$$\widehat{\mathbf{A}} \leftarrow \widehat{\mathbf{A}} + \phi(s,a)\Big(\phi(s,a) - \gamma\phi(s', \pi(s'))\Big)^\intercal$$

$$\widehat{b} \leftarrow \widehat{b} + \phi(s,a)r$$

We call this new algorithm LSQ (summarized in Figure 1) due to its similarity to LSTD. However, unlike LSTD, LSQ computes Q functions and does not expect the data to come from any particular Markov chain. It is a feature of this algorithm that it can use the same set of samples to compute Q values for any policy representation, as long as that representation offers an action choice for each s' in the set. The policy merely determines which $\phi(s', \pi(s')$ is added to $\widehat{\mathbf{A}}$ for each sample. Thus, LSQ can use every single sample available to it no matter what policy is under evaluation. We note that if a particular set of projection weights are desired, it is straightforward to reweight the samples as they are added to $\widehat{\mathbf{A}}$.

Notice that apart from storing the samples, LSQ requires only $O(k^2)$ space independently of the size of the state and the action space. For each sample in D, LSQ incurs a cost of $O(k^2)$ to update the matrices $\widehat{\mathbf{A}}$ and $\widehat{b}$ and a one time cost of $O(k^3)$ to solve the system and find the weights. Singular value decomposition (SVD) can be used for robust inversion of $\widehat{\mathbf{A}}$ as it is not always a full rank matrix.

One can think of the LSQ algorithm as the LSTD algorithm applied on a Markov chain with states (s, a) in which transitions are influenced by both the dynamics of the original MDP and the policy π being evaluated. Further, LSQ includes LSTD as a special case, where there is only one action available.

It is also possible to extend LSQ to LSQ(λ) in a way that resembles closely LSTD(λ) [3], but in that case it is necessary that the sample set consists of complete episodes generated using the policy under evaluation. Unfortunately, this requirement is very restrictive, raises again the question

LSQ $(D,\ k,\ \phi,\ \gamma,\ \pi)$

// D : Set of sample data
// k : Number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// π : Policy whose value function is sought

$\widehat{\mathbf{A}} = 0$ // $(k \times k)$ matrix
$\widehat{b} = 0$ // $(k \times 1)$ vector

for all $(s, a, r, s') \in D$
$\widehat{\mathbf{A}} \leftarrow \widehat{\mathbf{A}} + \phi(s,a)\Big(\phi(s,a) - \gamma\phi(s', \pi(s'))\Big)^\intercal$
$\widehat{b} \leftarrow \widehat{b} + \phi(s,a)r$

$\widehat{w}^\pi = \widehat{\mathbf{A}}^{-1}\widehat{b}$

return $\widehat{w}^\pi$

Figure 1: The LSQ algorithm.

of bias due to sampling distribution, and prevents the reusability of samples which is a very crusial property in the context of policy iteration. The only gain of using a value of λ other than 0 is a (possible) faster convergence to the same result.

LSQ is also applicable in the case of infinite and continuous state and/or action spaces with no modification. States and actions are reflected only through the basis functions of the linear approximation and the resulting value function can cover the entire state-action space with the appropriate set of continuous basis functions.

If there is a (compact) model of the MDP available, LSQ can benefit from it by using it to compute (instead of sampling) the summations from the rows of $\mathbf{P}^\pi\mathbf{\Phi}$ and $\mathcal{R}$:

$$\widehat{\mathbf{A}} \leftarrow \widehat{\mathbf{A}} + \phi(s,a)\Big(\phi(s,a) - \gamma\sum_{s'} P(s,a,s')\phi(s', \pi(s'))\Big)^\intercal$$

$$\widehat{b} \leftarrow \widehat{b} + \phi(s,a)\sum_{s'} P(s,a,s')R(s,a,s')$$

This is practical only if the set of possible next states s' is fairly small for each (s,a) pair. Alternatively, only a few dominant terms of the summation

can be computed. In any case, this will result in better approximations of $\mathbf{P}^{\pi}\mathbf{\Phi}$ and $\mathcal{R}$. A generative model of the MDP can be also useful in generating samples uniformly distributed over pairs of states and actions.

5 LSPI: Least-Squares Policy Iteration

The LSQ algorithm provides a means of learning an approximate state-action value function, $\widehat{Q}^{\pi}(s,a)$, for any fixed policy π. We now integrate LSQ into an approximate policy iteration algorithm. Clearly, LSQ is a candidate for the value determination step. The key insight is that we can achieve the policy improvement step without ever explicitly representing our policy and without any sort of model. Recall that in policy improvement, the improved policy $\pi^{(t+1)}$ will pick the action a that maximizes $\widehat{Q}^{\pi^{(t)}}(s,a)$. Since LSQ computes Q functions directly, we do not need a model to determine our improved policy; all the information we need is contained implicitly in the weights parameterizing our Q functions:

$$\pi^{(t+1)}(s,w) = \arg\max_a \widehat{Q}^{\pi^{(t)}}(s,a) = \arg\max_a \phi(s,a)^{\intercal} w^{\pi^{(t)}}.$$

In this sense the improved (greedy) policy is represented implicitly through a finite set of parameters and can be determined on demand for any given state as shown above.

We close the loop simply by requiring that LSQ performs this maximization for each s' when constructing the $\widehat{\mathbf{A}}$ matrix for a policy. For very large or continuous action spaces, explicit maximization over a may be impractical. In such cases, some sort of global nonlinear optimization may be required required to determine the optimal action. However, depending on the exact form of $\widehat{Q}(s,a)$ as a function of a, a closed form solution could possibly possibly be derived.

Since LSPI uses LSQ to compute approximate Q functions, it can use any data source for samples. A single set of samples may be used for the entire optimization, or additional samples may be acquired, either through trajectories or some other scheme, for each iteration of policy iteration. The LSPI algorithm is summarized in Figure 2.

As with any approximate policy iteration algorithm, the convergence of LSPI is not guaranteed. Approximate policy iteration variants are typically analyzed in terms of a value function approximation error and an action selection error [2]. LSPI does not require an approximate policy representation, e.g., a policy function or "actor" architecture, removing one source of error. Moreover, the direct computation of linear Q functions from any data

LSPI $(k,\ \phi,\ \gamma,\ \epsilon,\ \pi_0,\ D_0)$

// k : Number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// ϵ : Stopping criterion
// π_0 : Initial policy, given as w_0, $\pi_0 = \pi(s, w_0)$ (default: $w_0 = 0$)
// D_0 : Initial set of samples, possibly empty

$D = D_0$
$\pi' = \pi_0$ // In essence, $w' = w_0$

repeat
  Update D (optional) // Add/remove samples, or leave unchanged
  $\pi = \pi'$ // $w = w'$
  $\pi' =$ **LSQ** $(D,\ k,\ \phi,\ \gamma,\ \pi)$ // $w' =$ **LSQ** $(D,\ k,\ \phi,\ \gamma,\ \pi(s, w))$
until $(\pi \approx \pi')$ // that is, $(||w - w'|| < \epsilon)$

return π // **return** w

Figure 2: The LSPI algorithm.

source, including stored data, allows the use of *all* available data to evaluate a policy, making the problem of minimizing value function approximation error more manageable.

6 Results

We initially tested LSPI on the problematic MDP in [5] which consists of a chain with 4 states (numbered from 1 to 4). There are two actions available, "left" (L) and "right" (R). The actions succeed with probability 0.9 and have the opposite effect with probability 0.1. The reward vector is defined as $(0, +1, +1, 0)$ and the discount factor is set to 0.9. It is clear that the optimal policy is RRLL. Koller and Parr [5] showed that, starting with the policy RRRR, a combination of LSTD style function approximation and policy iteration oscillates between the suboptimal policies RRRR and LLLL. The approximator used to represent the state value function was a linear combination of the following three basis functions:

$$\left(1,\ s,\ s^2\right),$$

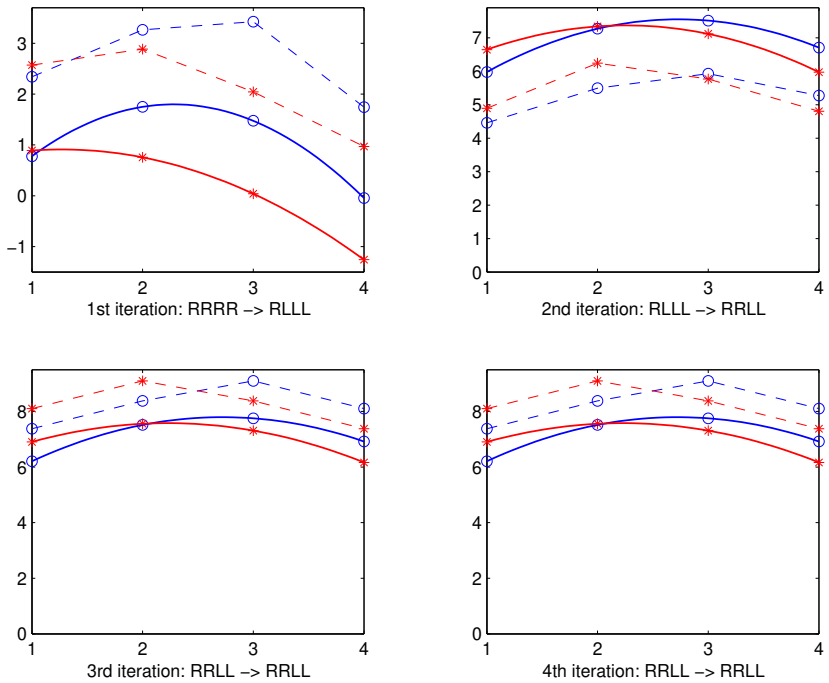


Figure 3: LSPI applied on the problematic MDP. States are on the horizontal axis and Q values on the vertical axis. The approximations are shown with solid lines, whereas the exact values are connected with dashed lines. The values for action L are marked with $\circ$ and for action R with $*$.

where s is the state number. LSPI was applied on the same problem using the same basis functions repeated for each of the two actions:

$$\big(\delta(a = L) \times 1,\ \delta(a = L) \times s,\ \delta(a = L) \times s^2,\ \delta(a = R) \times 1,\ \delta(a = R) \times s,\ \delta(a = R) \times s^2\big).$$

LSPI finds the optimal policy in 4 or 5 iterations using samples collected by acting randomly for about 25 steps or more. Figure 3 shows the iterations of LSPI on a training set of 50 samples. The optimal policy is found by the 2nd iteration, at which point the weights were approximately equal. Notice that the approximations capture the qualitative structure of the value function, although the quantitative error is fairly big. The state visitation distribution for this training set was $(0.24,\ 0.14,\ 0.28,\ 0.34)$. Although it was not exactly uniform it was "flat" enough to prevent bias in the approximation.

LSPI was also tested on variants of the above chain with more states and different reward schemes. This class of problems has no significant practical importance, but is particularly instructive as it is possible to compare the

approximations learned by LSPI to the true underlying value functions, and therefore understand better the way the algorithm works. Figure 4 shows a run of LSPI on a 20-state chain with the same dynamics as above and reward of +1 given only at the two boundaries (states 1 and 20). The optimal policy in this case is to go left in states 1-10 and right in states 11-20. LSPI converged to the optimal policy after 8 iterations using a training set of samples collected from a single "random-walk" episode of 5000 steps. A polynomial of degree 4 was used for approximating the value function for each of the two actions, giving a block of 5 basis functions per action, or a total of 10 basis functions. The initial policy in the iteration was that of going left (L) in all states. LSPI gradually discovered the optimal policy, although in iteration 2 it temporarily failed to get a good approximation and an improved policy.

Figure 5 shows an example where LSPI fails to discover the optimal policy. In this 50-state chain, reward is given only in states 10 and 41 and therefore due to the symmetry the optimal policy is to go right in states 1-9 and 26-41 and left in states 10-25 and 42-50. Again a polynomial of degree 4 was used for each of the actions. Using a training set of 10000 samples (all from a single trajectory), LSPI converged after 5 iterations to a suboptimal policy that nevertheless resembles the optimal policy to some extent. This failure can be explained as an inability of the polynomial approximator to capture the precise structure of the underlying value function.

Figure 6 shows the very same example with a radial basis function approximator (everything else is identical). This approximator uses 10 guassians with means spread uniformly over the state space ($\sigma = 4$ for all) plus a constant term for each of the 2 actions, resulting in a total of 22 basis functions. In this case, using the same training set, LSPI easily converged to the optimal policy after 7 iterations.

LSPI calls LSQ to determine the best approximation of the value function given a set of data. LSQ, in turn, solves for the fixed point of the Bellman equation in a least-squares sense. However, as discussed in Section 3, the direct least-squares solution that minimizes the Bellman error would be a potentially viable alternative. In particular, for this modified version of LSQ (and LSPI) the update equations would be:

$$\widehat{\mathbf{A}} \leftarrow \widehat{\mathbf{A}} + \Big(\phi(s,a) - \gamma\phi(s',\pi(s'))\Big)\Big(\phi(s,a) - \gamma\phi(s',\pi(s'))\Big)^{\mathsf{T}}$$

and

$$\widehat{b} \leftarrow \widehat{b} + \Big(\phi(s,a) - \gamma\phi(s',\pi(s'))\Big)r$$

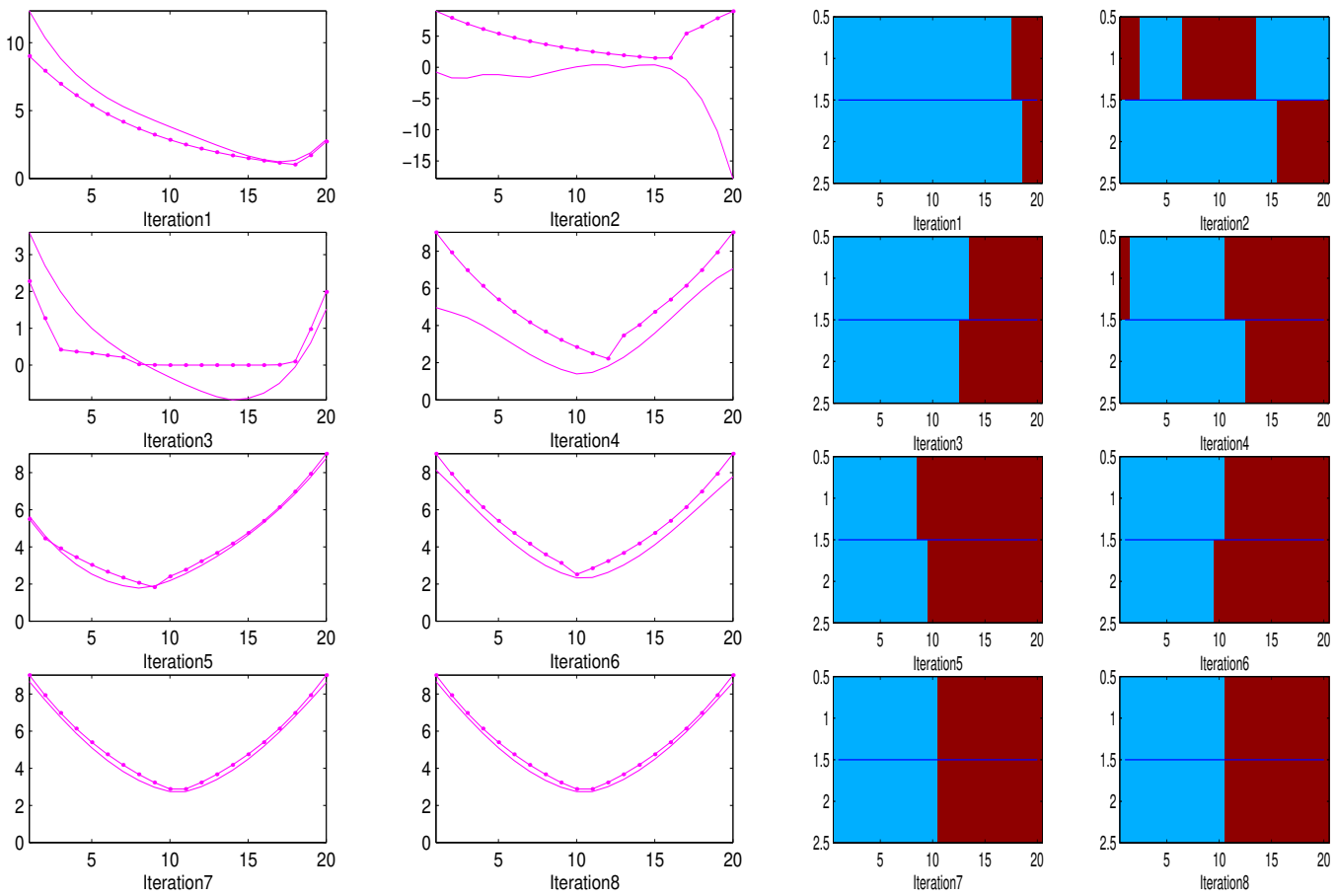
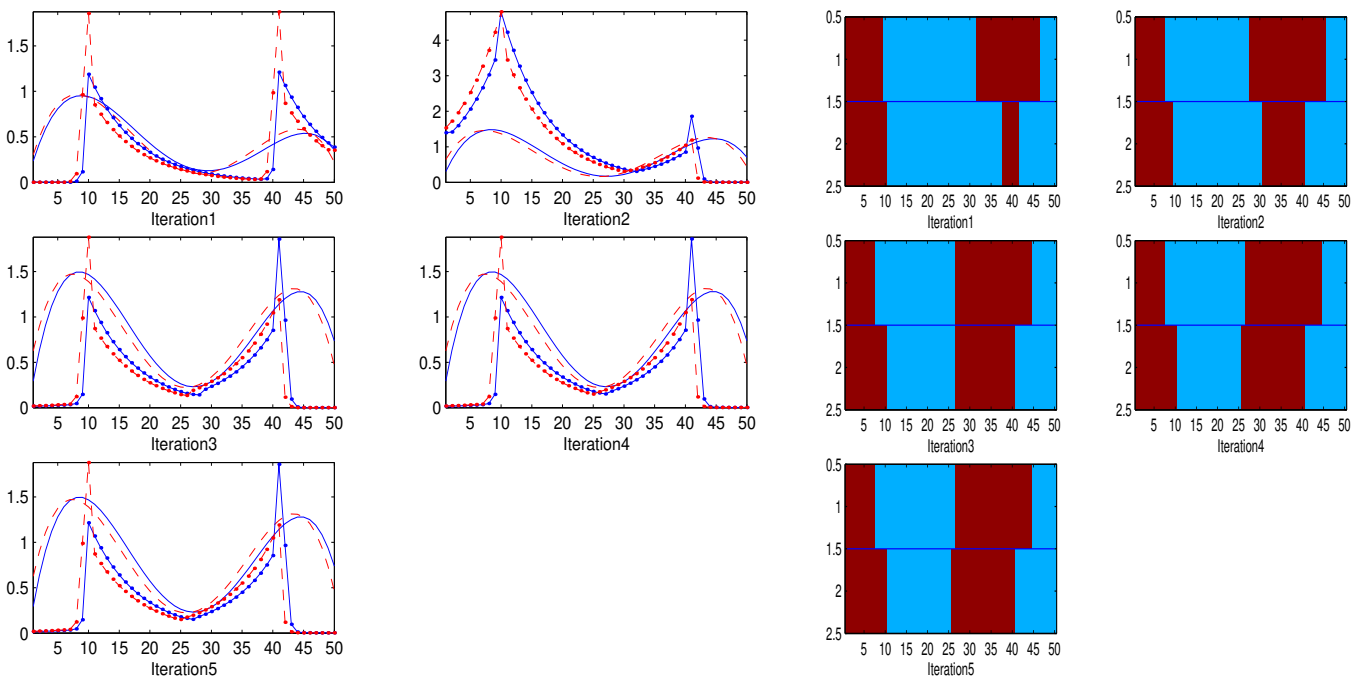


Figure 4: LSPI applied on a 20-state chain. (a) The state value function $V^{\pi}(s) = Q(s, \pi(s))$ of the policy being evaluated in each iteration. The LSPI approximation is shown with a solid line and the exact value with a dotted line. (b) The improved policy after each iteration (dark shade indicates the R action and light shade the L action). The top row is the improved policy that results from the LSPI approximation and the bottom row the improved policy from the exact value function.



(a) (b)

Figure 5: LSPI applied on a 50-state chain with a polynomial approximator. (a) The state-action value function of the policy being evaluated in each iteration. The exact values are shown with dotted lines. The curves for the R action are shown with dashed lines. (b) The improved policy after each iteration (dark shade indicates the R action and light shade the L action).

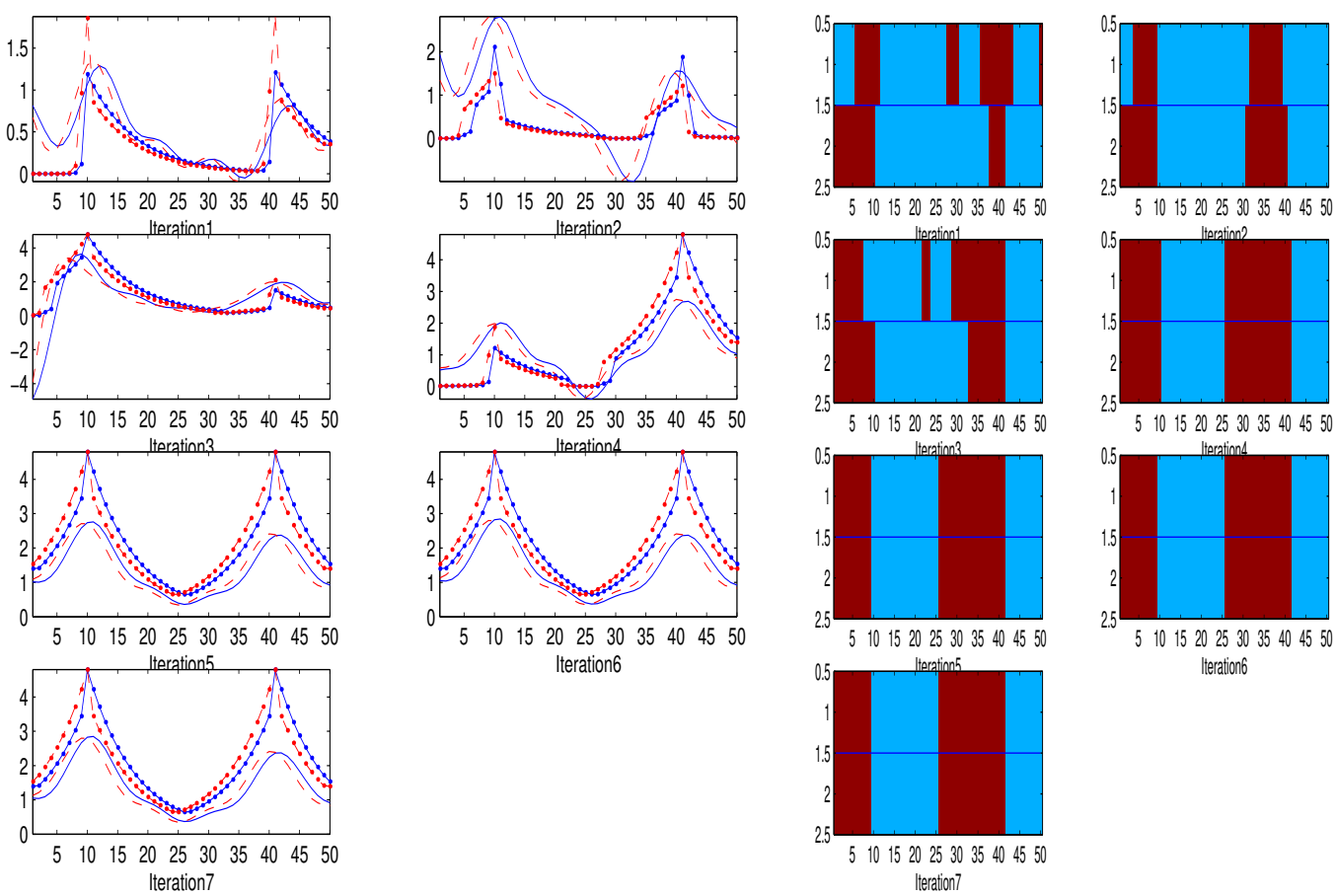


Figure 6: LSPI applied on a 50-state chain with a radial basis function approximator. (a) The state-action value function of the policy being evaluated in each iteration. The exact values are shown with dotted lines. The curves for the R action are shown with dashed lines. (b) The improved policy after each iteration (dark shade indicates the R action and light shade the L action).

For comparison purposes, this modified version of LSPI was applied to the 50-state problem above (same parameters, same training set). Figure 7 and Figure 8 show the results for the polynomial and the radial basis function approximators respectively. In both cases the approximations of the value function seem to be "close" to the true value functions. However, in both cases LSPI exhibits an oscillatory behavior. With the polynomial approximator, the iteration bounces back and forth between two "bad" policies that are far from being optimal or near-optimal. With the radial basis function approximator, the iteration proceeds toward better policies, but it gets trapped in a cycle after iteration 9. Nevertheless, the two alternating policies are not that far from the optimal policy.

It is not clear why the fixed point solution works better than the pure least-squares solution. We conjecture that the fixed point solution preserves the "shape" of the value function to some extent rather than trying to fit the absolute values. In return, the improved policy from the approximate value function is "closer" to the improved policy from the corresponding exact value function, and therefore policy iteration is guided to the "right" direction. Of course this point needs further investigation, but our experience with both methods suggests that the fixed point solution is the right choice in practice.

We also tested LSPI on the *inverted pendulum* problem, where the task is to balance a pendulum of unknown length and mass at the upright position by moving the cart to which it is attached. The state space in this problem is continuous and consists of the vertical angle and the angular velocity of the pendulum. The (nonlinear) dynamics of the system are described by the following state equations [15]:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{g\sin(x_1) - aml x_2^2 \sin(2x_1)/2 - a\cos(x_1)u}{4l/3 - aml\cos^2(x_1)} \end{aligned}$$

where the variables and parameters are as follows:

- x_1 : Angle
- x_2 : Angular Velocity
- g : Gravity constant ($g = 9.8m/s^2$)
- m : Mass of the pendulum ($m = 2.0$ kg)
- M : Mass of the cart ($M = 8.0$ kg)

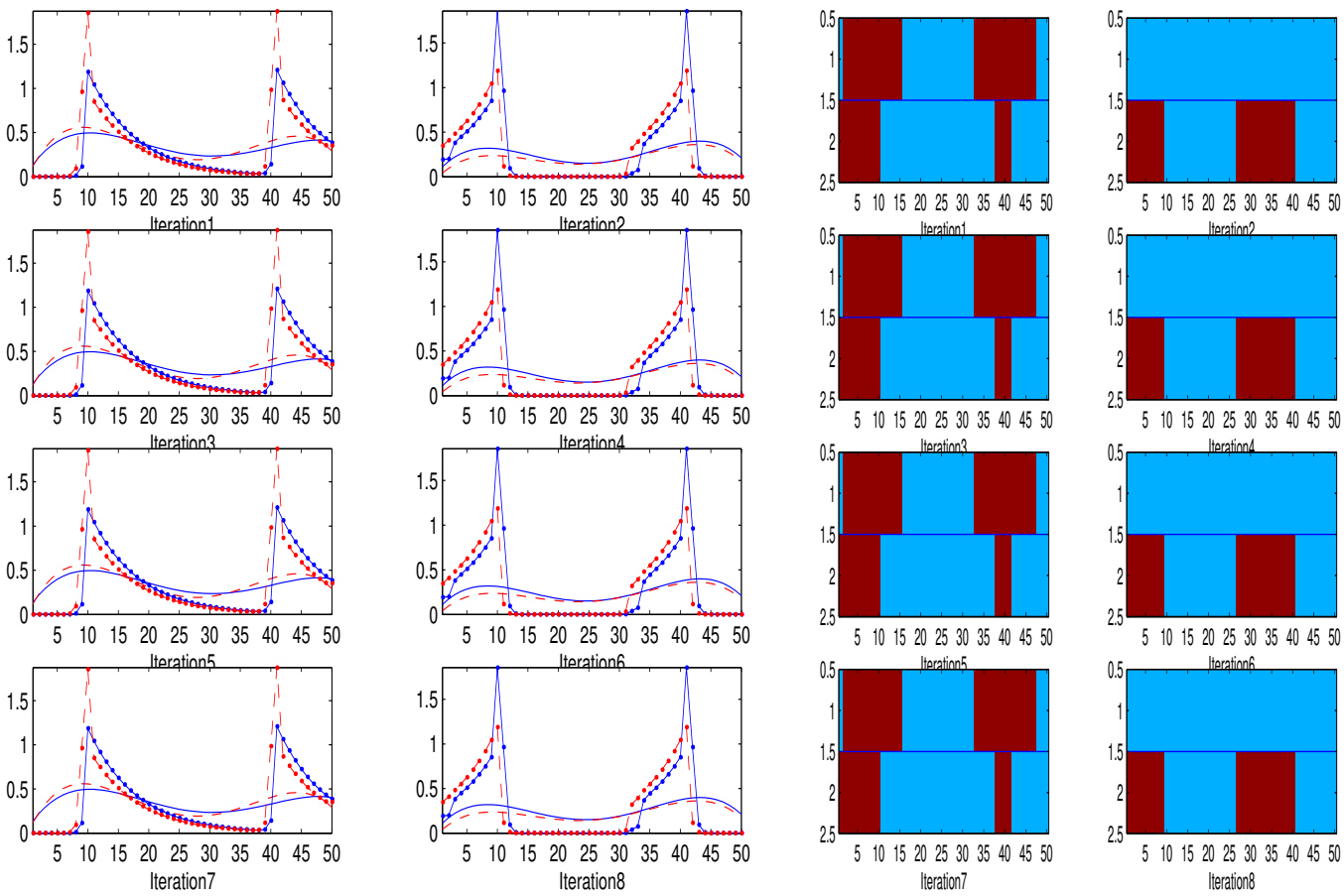
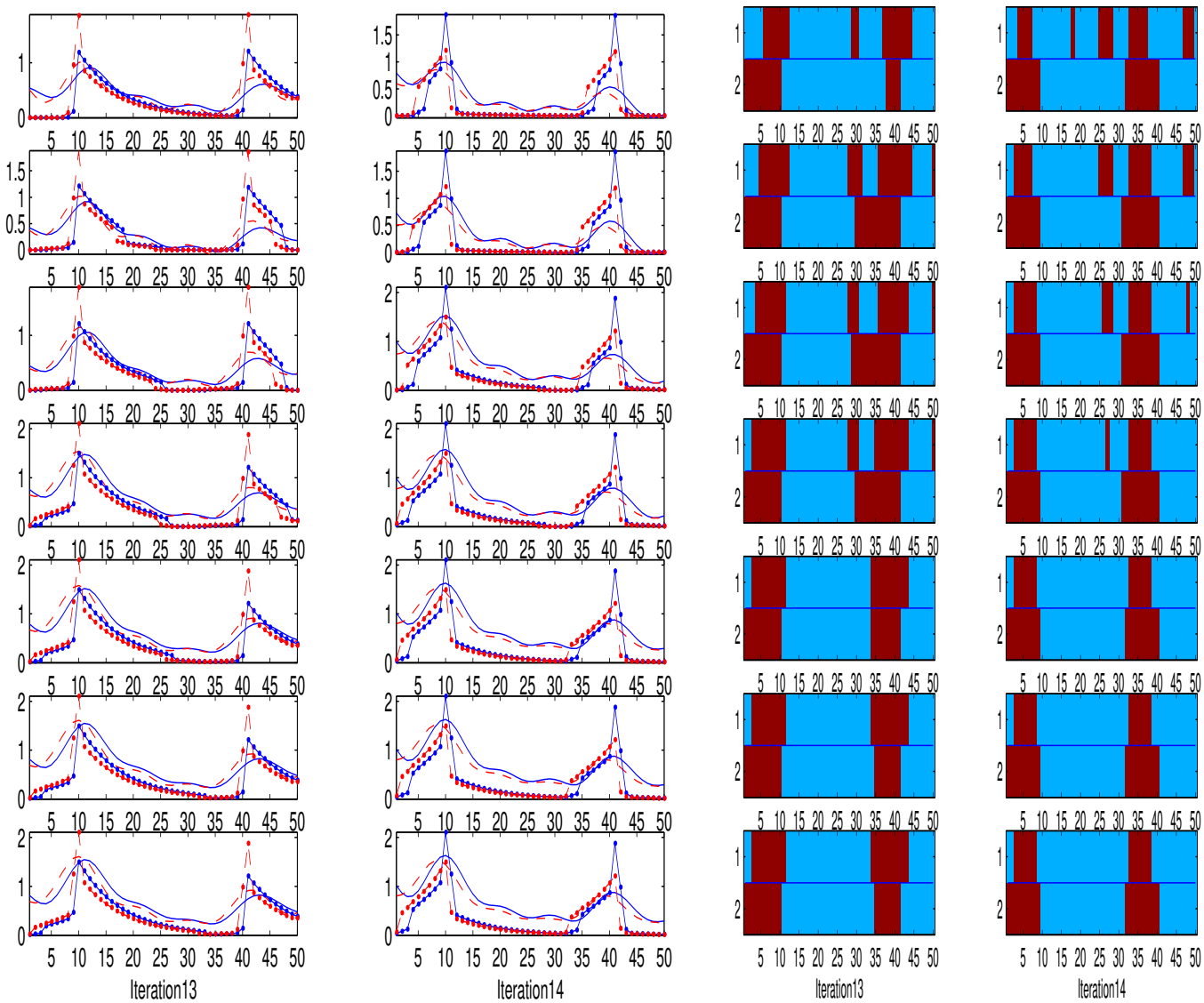


Figure 7: Modified LSPI applied on the 50-state chain with a polynomial approximator.



(a) (b)

Figure 8: Modified LSPI applied on the 50-state chain with a radial basis function approximator.

- l : Length of the pendulum ($l = 0.5$ m)
- u : Force (in Newtons)
- a : Constant $a = 1/(m + M)$

The action space is discrete and consists of three actions: positive force (+50 Newtons), negative force (-50 Newtons), and zero force, so $A = \{-50,\ 0,\ +50\}$. The actions are noisy, that is, if a is one of the three actions in A, the actual input u to the system is

$$u = a + 10n,$$

where n is a gaussian noise term with zero mean, variance one, and standard deviation one. The simulation step is set to 0.1 seconds. The control input is given at a rate of 10 Hz, at the beginning of each time step, and is kept constant during any time step.

The controlling agent receives zero reward as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value. An angle greater than $\pi/2$ signals the end of the episode and a penalty of -1. The discount factor of the process is set to 0.9.

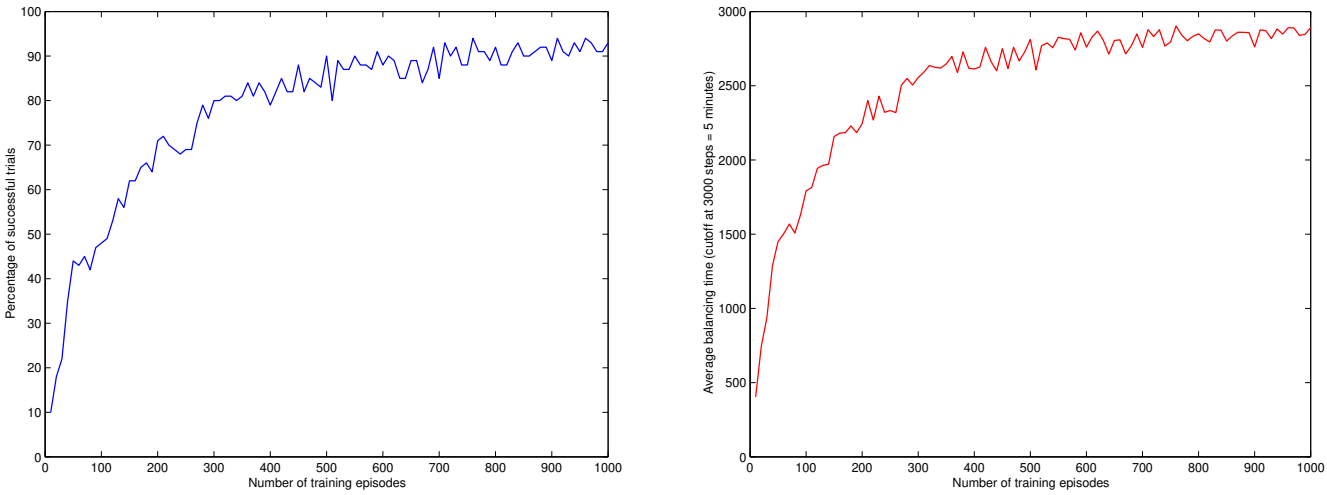
We have used a set of 10 basis functions for each of the 3 actions, thus a total of 30 basis functions, to approximate the value function. These 10 basis functions include a constant term and 9 radial basis functions (gaussians) arranged in a 3×3 grid over the 2-dimensional state space. In particular, for some state s the block of basis functions for each action is the following

$$\left(\ 1,\ e^{-\frac{||s-\mu_1||^2}{2\sigma^2}},\ e^{-\frac{||s-\mu_2||^2}{2\sigma^2}},\ e^{-\frac{||s-\mu_3||^2}{2\sigma^2}},\ \ldots,\ e^{-\frac{||s-\mu_9||^2}{2\sigma^2}}\ \right),$$

where the μ_i's are the 9 points of the grid $\{-\pi/4,\ 0,\ +\pi/4\} \times \{-1,\ 0,\ +1\}$ and $\sigma^2 = 1$.

Training samples were collected from "random episodes", i.e. starting in a random state close to the equilibrium state (angle=0, angular velocity=0) and following a purely random policy. The average length of such episodes is about 9 steps, thus each one contributes about 9 samples. LSPI is able to find better and better policies as the number of training episodes grows.

Figure 9 shows the performance of the learned controllers as a function of the number of training episodes. Each (successful) episode is allowed to run for a maximum of 3000 steps, that is, 5 minutes of continuous balancing. LSPI returns excellent policies in almost all cases given a few hundred training episodes. Failures come mostly as a result of the distribution of the



(a) (b)

Figure 9: The inverted pendulum problem: (a) Percentage of successful trials; (b) Average balancing time (cutoff at 3000 steps = 5 minutes); both averaged over 100 runs of LSPI for each training set size.

samples not being "uniform", but that fades out as the number of samples grows.

Finally, we tried a bicycle balancing problem [12] in which the goal is to learn to balance and ride a bicycle to a target position located 1 km away from the starting location. Initially, the bicycle's orientation is at an angle of 90° to the goal. The state description is a six-dimensional real-valued vector $(\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega}, \psi)$, where θ is the angle of the handlebar, ω is the vertical angle of the bicycle, and ψ is the angle of the bicycle to the goal. The actions are the torque τ applied to the handlebar (discretized to $\{-2, 0, +2\}$) and the displacement of the rider υ (discretized to $\{-0.02, 0, +0.02\}$). In our experiments, actions are restricted to be either τ or υ (or nothing) giving a total of 5 actions[2]. The noise in the system is a uniformly distributed term in $[-0.02, +0.02]$ added to the displacement component of the action. The dynamics of the bicycle are based on the model described in [12] and the time step of the simulation is set to 0.01 seconds.

The state-action value function $Q(s, a)$ for a fixed action a is approximated by a linear combination of 20 basis functions:

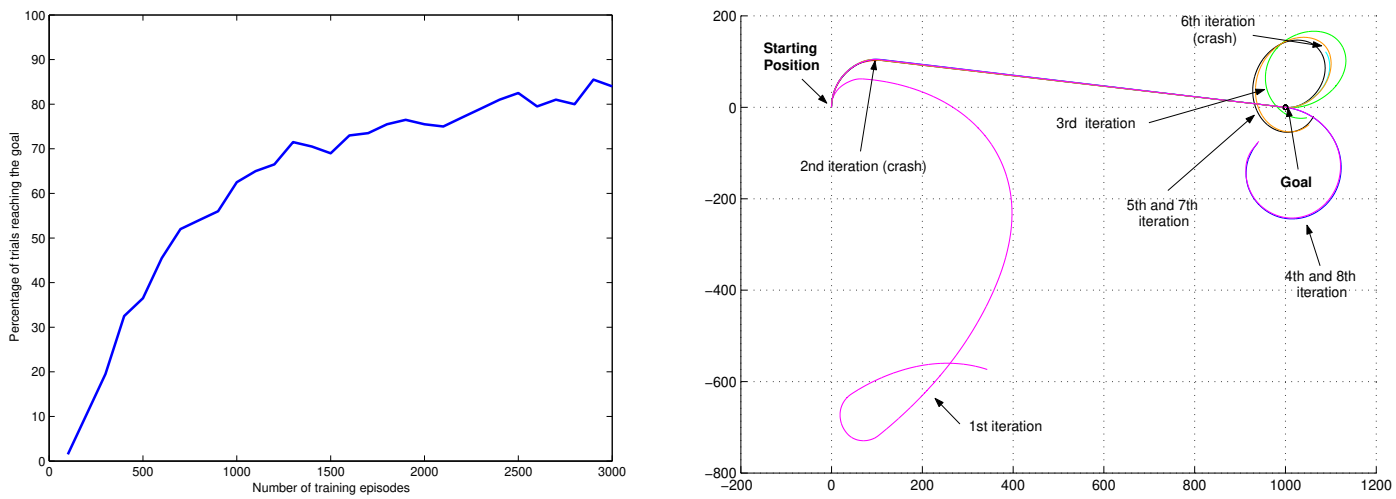
$$(\ 1,\ \omega,\ \dot{\omega},\ \omega^2,\ \dot{\omega}^2, \omega\dot{\omega},\ \theta,\ \dot{\theta},\ \theta^2,\ \dot{\theta}^2,\ \theta\dot{\theta},\ \omega\theta,\ \omega\theta^2,\ \omega^2\theta,\ \psi,\ \psi^2,\ \psi\theta,\ \bar{\psi},\ \bar{\psi}^2,\ \bar{\psi}\theta\ ),$$

[2]Results are similar for the full 9-action case, but required more training data.

where $\bar{\psi} = \pi - \psi$ for $\psi > 0$ and $\bar{\psi} = -\pi - \psi$ for $\psi < 0$. Note that the state variable $\ddot{\omega}$ is completely ignored. This block of basis functions is repeated for each of the 5 actions, giving a total of 100 basis functions and weights. Training data were collected by initializing the bicycle to a random state around the equilibrium position and running small episodes of 20 steps each using a purely random policy. LSPI was applied on training sets of different sizes and the average performance is shown in Figure 10(a). We used the same data set for each run of policy iteration and usually obtained convergence in 6 or 7 iterations. Successful policies usually reached the goal in approximately 1 km total, near optimal performance. We also show an annotated set of trajectories to demonstrate the performance improvement over multiple steps of policy iteration in Figure 10(b).

A number of design decisions influenced the performance of LSPI on the bicycle rider problem. As is typical with this problem, we used a shaping reward [9] for the distance to the goal. In this case, we used 0.01 of the net change (in meters) in the distance to the goal. We found that when using full random trajectories, most of our sample points were not very useful; they occurred after the bicycle had already entered into a "death spiral" from which recovery was impossible. This complicated our learning efforts by biasing the samples towards hopeless parts of the space, so we decided to cut off trajectories after 20 steps. This created an additional problem because there was no terminating reward signal to indicate failure. We approximated this with an additional shaping reward, which was proportional to the net change in the square of the vertical angle. This roughly approximated the likeliness of falling at the end of a truncated trajectory. Finally, we used a discount of 0.80, which seemed to yield more robust performance.

We admit to some slight unease about the amount of shaping and adjusting of parameters that was required to obtain good results on this problem. To verify that we had not eliminated the learning problem entirely through shaping, we reran some experiments using a discount of 0. In this case LSQ simply projects the immediate reward function into the column space of the basis functions. If the problem were tweaked too much, acting to maximize the projected immediate reward would be sufficient to obtain good performance. On the contrary, these runs always produced immediate crashes in trials.



(a) (b)

Figure 10: The bicycle problem: (a) Percentage of final policies that reach the goal, averaged over 200 runs of LSPI for each training set size; (b) A sample run of LSPI based on 2500 training trials. This run converged in 8 iterations. Note that iterations 5 and 7 had different Q-values but very similar policies. This was true of iterations 4 and 8 as well. The weights of the ninth differed from the eighth by less than 10^{-10} in $\|\cdot\|_2$, indicating convergence.

7 Future Work

LSPI raises several issues and interesting questions that need further investigation. The distribution of the samples in the training set can have a significant impact on the performance of the algorithm. There are (at least) two issues here. The first is the fact that the approximation can be biased in favor of state-action pairs that are sampled more frequently and against state-action pairs that are sampled infrequently. For this problem, we plan to use density estimation to techniques to reweight the samples so that we may achieve a desired set of projection weights regardless of the distribution of samples. The second and more serious problem is that our initial batch of samples may not contain a representative set of samples, i.e., some important states may never be visited. We note that this problem appears in one form or another in all reinforcement learning methods. For example, Q-learning agents often must reach the goal state on a random walk before any meaningful learning can occur. For LSPI, we would like to develop a more general framework for sample collection that views the problem of acquiring new samples as an *active learning* problem. This would encompass both determining the need for new samples and the region of state space from which these samples are required.

In all experimental results of this report we have constructed the linear architecture by repeating the same block of basis functions (that depend only on the state) for each of the actions, essentially allowing for separate weights for each action. However, this is not the only way to go and is definitely impractical for big action spaces. Alternatively, it can be constructed as one set of basis functions that depend on both the state and the action and there is one set of weights for all actions.

The choice of basis functions is a fundamental problem in itself. In this report, we explored only polynomial and radial basis functions, but many other options exist. Answering this question seems to be very domain-specific and there is no universal choice that works in all domains. Further, it can be a serious constraining factor in the success of LSPI; LSPI relies on a good choice of basis function, and cannot really compensate for a bad choice.

In this report LSPI has been used as an off-line method; learning takes place after the training samples have been collected. An online version of LSPI would be possible by maintaining a window of the most recent experiences and using that for training. This point is yet to be explored, but it's likely that will compensate for slowly changing environments.

Other issues that we would like to explore include domains with contin-

uous actions, alternative projections in value function space, manipulation of the training samples, use of a model in LSPI, and other approximation architectures.

8 Discussion and Conclusions

We have presented a new, model-free approximate policy iteration algorithm called LSPI, which is inspired by the LSTD algorithm. This algorithm is able to use either a stored repository of samples or samples generated dynamically from trajectories. It performs action selection and approximate policy iteration entirely in value function space, without any need for model. In contrast to other approaches to approximate policy iteration, it does not require any sort of approximate policy function.

It is natural to compare our algorithm to other memory based methods as well as the recent flourish of policy search methods, [8, 7, 1, 14, 6]. In comparison to the memory based approach of [10], our method makes stronger use of function approximation. Rather than using our samples to implicitly construct an approximate model using kernels, we operate entirely in value function space and use our samples directly in the value function projection step. As noted in [3] the $\mathbf{A}$ matrix used by both LSTD and LSPI can be viewed as a form of approximate, compressed model. This is most compelling if the columns of Φ are orthonormal. While this provides some intuitions, a proper transition function cannot be reconstructed directly from $\mathbf{A}$, making a possible interpretation of LSPI as a model based method an area for future research.

In comparison to direct policy search methods, we offer the strength of policy iteration. Policy search methods typically make a large number of relatively small steps of gradient-based policy updates to a parameterized policy function. Our use of policy iteration generally results in a small number of very large steps directly in policy space.

Our experimental results demonstrate the potential of our method. We achieved good performance on the bicycle task using a very small number of randomly generated samples that were reused across multiple steps of policy iteration. Achieving this level of performance with just a linear value function architecture did require some tweaking, but the transparency of the linear architecture made the relevant issues much more salient than would be the case with any "black box" approach. We believe that the direct approach to function approximation and data reuse taken by LSPI will make the algorithm an intuitive and easy to use first choice for many

reinforcement learning tasks. In future work, we plan to investigate the application of our method to multi-agent systems and the use of density estimation to control the projection weights in our function approximator.

Acknowledgments

We would like to thank J. Randløv and P. Alstrøm for making the bicycle simulator from [12] available. We also thank C. Guestrin, D. Koller, U. Lerner and M. Littman for helpful discussions. The first author would like to thank the Lilian-Boudouri Foundation in Greece for partial financial support.

References

[1] J. Baxter and P.Bartlett. Reinforcement learning in POMDP's via direct gradient ascent. In *Proc. 17th International Conf. on Machine Learning*, pages 41–48. Morgan Kaufmann, San Francisco, CA, 2000.

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

[3] Justin A. Boyan. Least-squares temporal difference learning. In I. Bratko and S. Dzeroski, editors, *Machine Learning: Proceedings of the Sixteenth International Conference*, pages 49–56. Morgan Kaufmann, San Francisco, CA, 1999.

[4] S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1/2/3):33–57, 1996.

[5] D. Koller and R. Parr. Policy iteration for factored mdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00).* Morgan Kaufmann, 2000.

[6] V. Konda and J. Tsitsiklis. Actor-critic algorithms. In NIPS 2000 editors, editor, *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*. MIT Press, 2000.

[7] A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00).* Morgan Kaufmann, 2000.

[8] A. Ng, R. Parr, and D. Koller. Policy search via density estimation. In *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*. MIT Press, 2000.

[9] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, pages 278–287. Morgan Kaufmann, San Francisco, CA, 1999.

[10] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. To appear, *Machine Learning*, 2001.

[11] D. Precup and R. Sutton. Off-policy temporal difference learning with function approximation. In *The Eighteenth International Conference on Machine Learning*, Williamstown, Massachusetts, June 2001. To appear.

[12] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *The Fifteenth International Conference on Machine Learning*, Madison, Wisconsin, July 1998. Morgan Kaufmann.

[13] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110:568 – 582, 1985.

[14] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*, Denver, Colorado, 2000. MIT Press.

[15] K. Wang, H. Tanaka and M. Griffin. An approach to fuzzy control of nonlinear systems: Stability and design issues. *IEEE Transactions on Fuzzy Systems*, 4(1):14–23, 1996.