

# Randomised Bayesian Least-Squares Policy Iteration

**Nikolaos Tziortziotis**

*Computer Science Laboratory (LIX)  
École Polytechnique, France*

NTZIORZI@GMAIL.COM

**Christos Dimitrakakis**

*Department of Informatics  
University of Oslo, Norway*

CHRISTOS.DIMITRAKAKIS@GMAIL.COM

**Michalis Vazirgiannis**

*LIX, École Polytechnique, France  
Athens University of Economics and Business, Greece*

MVAZIRG@LIX.POLYTECHNIQUE.FR

## Abstract

We introduce Bayesian least-squares policy iteration (BLSPI), an off-policy, model-free, policy iteration algorithm that uses the Bayesian least-squares temporal-difference (BLSTD) learning algorithm to evaluate policies. An online variant of BLSPI has been also proposed, called randomised BLSPI (RBLSPI), that improves its policy based on an incomplete policy evaluation step. In online setting, the exploration-exploitation dilemma should be addressed as we try to discover the optimal policy by using samples collected by ourselves. RBLSPI exploits the advantage of BLSTD to quantify our uncertainty about the value function. Inspired by Thompson sampling, RBLSPI first samples a value function from a posterior distribution over value functions, and then selects actions based on the sampled value function. The effectiveness and the exploration abilities of RBLSPI are demonstrated experimentally in several environments.

**Keywords:** Reinforcement learning, Bayesian reinforcement learning, Bayesian least-squares temporal-difference, Bayesian least-squares policy-iteration, exploration

## 1. Introduction

In artificial intelligence (AI), our primary objective is to design intelligent agents able to discover autonomously optimal policies (behaviors) by interacting with their (usually unknown) environment. The reinforcement learning (RL) problem (Sutton and Barto, 1998) is a special case of this general setting. Least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003) is a model-free RL algorithm that is known for its efficient use of training samples and has succeeded in many challenging control problems. It belongs to the family of policy iteration algorithms, using a variant of the least-squares temporal difference (LSTD) algorithm (Bradtko and Barto, 1996; Boyan, 2002) for policy evaluation.

In its original form, LSPI is an offline algorithm that is based on a batch of samples provided beforehand and have been collected through the interaction of the decision maker with his environment. Nevertheless, most of the problems that encountered on real world are online. An online version of LSPI has been proposed by Buşoniu et al. (2010), that performs policy improvements once every few transitions have been completed, and updates its policy based on an *incomplete* evaluation of its current policy. Efficient exploration is one of the main challenges in online learning, as at each time step we have to decide if we will exploit our current knowledge (estimation), or we will explore our world trying to gain more information about poorly understood/visited states

and actions. This is well-known as the *exploration-exploitation dilemma*. In online LSPI, authors have adopted the simple  $\epsilon$ -greedy exploration strategy (Sutton and Barto, 1998). Before Buşoniu et al. (2010), Li et al. (2009) have introduced RMAX-LSPI algorithm that incorporates the RMAX exploration technique (Brafman and Tennenholz, 2003) into LSPI. In contrast to online LSPI that discards the collected transition samples, RMAX-LSPI algorithm updates its policy based on the whole set of visited sample transitions (LSPI is fully executed at the end of an episode).

In this work, we propose the Bayesian LSPI (BLSPI) algorithm that constitutes a Bayesian version of the standard LSPI algorithm. Instead of the LSTD algorithm used by LSPI for policy evaluation, BLSPI uses the Bayesian LSTD (BLSTD) algorithm proposed recently by Tziortziotis and Dimitrakakis (2017). BLSTD adopts a probabilistic model for the empirical Bellman operator and introduces a prior distribution over the model’s parameters. This gives us the advantage of probabilistic predictions that quantify our uncertainty over the estimated value function after each policy evaluation step. We also extend the BLSPI algorithm in the online setting by introducing the Randomised Bayesian LSPI (RBLSPI) algorithm. Like online LSPI, RBLSPI updates its policies based on incomplete policy evaluations. To address the exploration problem, RBLSPI exploits the ability of BLSTD to capture our uncertainty of the value estimates by using Bayesian inference. Instead of selecting random actions, RBLSPI explores its environment by sampling randomly value functions from a posterior distribution over value functions. More specifically, RBLSPI samples a value function after each policy improvement step, that defines the *behavioral policy* of the agent. Thereafter, the *behavioral policy* is followed greedily up to the next policy improvement step that is performed once every few state transition samples.

This kind of exploration is based on the simple idea of Thompson sampling (Thompson, 1933) that has been shown to perform very well in Bayesian reinforcement learning (Strens, 2000; Ghavamzadeh et al., 2015). In model-based Bayesian RL (Osband et al., 2013; Tziortziotis et al., 2013, 2014), the agent starts by considering a prior belief over the unknown environment model. Then, a model from the posterior distribution is randomly sampled, and an optimal policy is calculated w.r.t. the sampled model. This policy is greedily followed thereafter, with the observed samples to be used to update the posterior distribution over models. Recently, Osband et al. (2016) introduced the idea of randomised value functions by proposing the randomised least-squares value iteration (RLSVI) algorithm. RLSVI uses an exploration method that is similar to the one of RBLSPI: it estimates a Bayesian linear regression model for the value function at the end of each episode, from the observations made up to this point. This means that we need to keep in memory the transitions observed through the time, in order to estimate value functions at each episode by building a “new” Bayesian regression model. Then, a sample from the posterior of the bayesian model over value functions is obtained, and the greedy policy with respect to it is followed at the next episode. Due to our different modelling choices, RBLSPI can use all of the previous episodes data history without the need to keep the observed transitions in memory, and is so more data efficient. The idea of randomised value functions has been also adopted to deep RL showing that it is an efficient exploration strategy (Osband et al., 2017; Touati et al., 2018; Azizzadenesheli et al., 2018).

## 2. Preliminaries

We formulate the underlying control problem as a discrete-time  $\gamma$ -discounted *Markov decision process* (MDP),  $\mu \in \mathcal{M}$ , defined by the tuple  $\{\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma\}$ , where  $\mathcal{S}$  is the set of states;  $\mathcal{A}$  is the set of available actions;  $P(\cdot|s, a)$  is a transition kernel, specifying the probability of transition from

state  $s \in \mathcal{S}$  to next states by taking action  $a \in \mathcal{A}$ ;  $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is a reward function and  $\gamma \in [0, 1)$  is a constant discount factor.

We assume that the agent selects its actions based on a deterministic policy,  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ , which is a mapping from states to actions;  $\pi(s) \in \mathcal{A}$  denotes the action returned by policy  $\pi$  at state  $s$ . The utility of the agent is the discounted sum of future reward,  $U \triangleq \sum_{t=0}^{\infty} \gamma^t r_t$ , where  $r_t = \mathcal{R}(s_t, a_t)$  is the reward received after executing action  $a_t$  at state  $s_t$ . Given MDP  $\mu$ , the agent's objective is to discover an optimal policy  $\pi^*$  that maximises its expected utility for each possible state  $s$ :  $V^\pi(s) \triangleq \mathbb{E}_\mu^\pi[U|s_0 = s]$ , where the expectation is getting w.r.t. the agent's policy  $\pi$  and environment  $\mu$ . In the control problem and especially in the case where the model of the environment is unknown, it is more preferable to consider the action-value function,  $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ , which given a policy  $\pi$  indicates the expected return obtained by executing action  $a$  at state  $s$ , and following the policy  $\pi$  thereafter:  $Q^\pi(s, a) \triangleq \mathbb{E}_\mu^\pi[U|s_0 = s, a_0 = a]$ .

It is well-known that the action-value function of a policy  $\pi$  is the unique fixed-point of the *Bellman operator* (Puterman, 2005), i.e.  $Q^\pi = T^\pi Q^\pi$ , with the operator  $T^\pi : (\mathcal{S} \times \mathcal{A}) \rightarrow (\mathcal{S} \times \mathcal{A})$  to be defined as:

$$T^\pi Q(s, a) \triangleq r(s, a) + \gamma \int_{\mathcal{S}} Q(s', \pi(s')) dP(s'|s, a), \quad (1)$$

or in a vector form as:  $T^\pi Q \triangleq \mathcal{R} + \gamma P^\pi Q$ , where  $\mathcal{R} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$  is the reward vector, and  $P^\pi \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|}$  is the transition matrix induced by the selection of an action and policy  $\pi$  right after.

The optimal action-value function is defined as  $Q^*(s, a) \triangleq \sup_\pi Q^\pi(s, a)$ , for each  $(s, a) \in \mathcal{S} \times \mathcal{A}$ . Given  $Q$ , a policy is called greedy when  $\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a), \forall s \in \mathcal{S}$ . The greedy policy w.r.t. the optimal action-value function  $Q^*$  is optimal, and is denoted by  $\pi^*$ . Thus, we need to determine the optimal action-value function in order to find an optimal policy.

**Policy iteration (Howard, 1960)** Policy iteration is an iterative procedure able to discover the optimal solution for a given MDP. By starting from an arbitrary policy  $\pi_0$  (i.e., randomly selected), it generates a sequence of monotonically improving policies along with their corresponding approximate action-value functions:  $Q^{\pi_0} \rightarrow \pi_1 \rightarrow Q^{\pi_1} \rightarrow \pi_2 \rightarrow \dots$ . Each iteration  $k$  consists of two successive steps: *policy evaluation* and *policy improvement*. In *policy evaluation* step the action-value function  $Q^{\pi_k}$  of current policy  $\pi_k$  is computed by solving the linear system of the Bellman equations:  $Q^\pi = (\mathbf{I} - \gamma P^\pi)^{-1} \mathcal{R}$ . Afterwards, *policy improvement* step generates an improved greedy policy with respect to the action-value function  $Q^{\pi_k}$ , i.e.,  $\pi_{k+1} = \arg \max_{a \in \mathcal{A}} Q^{\pi_k}(s, a)$ . The whole procedure terminates when no further improvements is possible (i.e.,  $\pi_k = \pi_{k-1}$ ), with the policy iteration algorithm to have converge to an optimal policy,  $\pi^* = \pi_k$ . In general, policy iteration converges after a small number of iterations.

**Approximate policy iteration (API)** Despite its merits, policy iteration requires the exact computation and representation of the action-value function. Nevertheless, in general the state and/or the action spaces is large or infinite (e.g., continuous spaces), making the explicit representation of the action-value function infeasible. To tackle this problem, a function approximation scheme is usually employed in the policy evaluation step. This kind of policy iteration is widely known as *approximate policy iteration* (API) (see Bertsekas (2011) for a survey). In RL, it is common to approximate the action-value function  $Q$  by considering linear approximation architectures, i.e. a linear combination of basis functions:  $Q_\theta^\pi(s, a) = \phi(s, a)^\top \theta$ , where  $\theta \in \mathbb{R}^k$  is a parameter vector and  $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^k$  is a function that maps state-action pairs to a feature vector of  $k$  components,  $\phi(\cdot) = (\phi_1(\cdot), \dots, \phi_k(\cdot))^\top$ . In that way, the number of parameters ( $k \ll |\mathcal{S}||\mathcal{A}|$ ) that need to be

estimated is much less compared to these required in the case of an exact representation. We also denote by  $\mathcal{F}$  the linear function space spanned by the features  $\phi_i$ , i.e.,  $\mathcal{F} = \{f_\theta | f_\theta(\cdot) = \phi(\cdot)^\top \theta\}$ . Actually,  $\mathcal{F}$  contains all the action-value functions that can be represented by the features.

## Least Squares Policy Iteration

One of the most well-known approximate policy iteration algorithms is that of least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003). It is a model-free, off-policy algorithm that uses the least-squares temporal difference (LSTD) (Bradtko and Barto, 1996) algorithm at the policy evaluation phase. In practice, LSTD returns the parameters vector  $\theta$  that minimises the *mean-squared projected bellman error* (MSPBE):

$$\text{MSPBE}(\theta) \triangleq \|Q_\theta^\pi - \Pi T^\pi Q_\theta^\pi\|_{\Xi}^2,$$

where  $\Xi$  is a diagonal matrix whose diagonal elements indicate the distribution over  $\mathcal{S} \times \mathcal{A}$ , and  $\Pi$  is the projection operator onto feature space  $\mathcal{F}$ . Actually, operator  $\Pi$  projects any value function  $u$  to its nearest value function onto  $\mathcal{F}$ , such that  $\Pi u = Q_\theta^\pi$  where the corresponding parameters are the solution on the next least-squares problem:  $\theta = \arg \min_\theta \|Q_\theta^\pi - u\|_{\Xi}^2$  (Sutton et al., 2009). As the parameterisation is linear, we can show that the projection operator is linear and independent of the parameters  $\theta$  and given by:  $\Pi = \Phi C^{-1} \Phi^\top \Xi$ , where  $\Phi \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times k}$  is a matrix whose rows contain the feature vector  $\phi(s, a)^\top, \forall (s, a) \in \mathcal{S} \times \mathcal{A}$  and  $C = \Phi^\top \Xi \Phi$  is the Gram matrix.

In practice, the dynamics of the environment  $\mu$  are unknown and the full feature matrices  $\Phi$  cannot be formed explicitly in continuous environments. For that purpose, LSTD relies on a batch of transition samples, which is assuming to be available at our disposal and have been collected through the interaction of the agent with the generative model:  $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ , where  $s'_i \sim P(s_i, a_i)$ . Given dictionary  $\mathcal{D}$ , let us define as  $\tilde{\Phi} = [\phi(s_1, a_1)^\top; \dots; \phi(s_N, a_N)^\top]$  and  $\tilde{\Phi}' = [\phi(s'_1, \pi(s'_1))^\top; \dots; \phi(s'_N, \pi(s'_N))^\top]$  the sampled feature matrices, and as  $\mathcal{R} = [r_i, \dots, r_N]^\top$  the sampled reward vector. Therefore, the empirical MSPBE can be expressed as a standard weighted least-squares problem:

$$E_{\mathcal{D}}(\theta) \triangleq \|A\theta - b\|_{\tilde{C}^{-1}}^2,$$

where  $A \triangleq \tilde{\Phi}^\top (\tilde{\Phi} - \gamma \tilde{\Phi}')$ ,  $b \triangleq \tilde{\Phi}^\top \mathcal{R}$ , and  $\tilde{C} \triangleq \tilde{\Phi}^\top \tilde{\Phi}$ . Minimising the empirical MSPBE we get that the optimal solution is given as:

$$\theta^* = A^{-1}b. \quad (2)$$

It has been shown (Bradtko and Barto, 1996; Lazaric et al., 2010; Nedić and Bertsekas, 2003) that the LSTD solution  $\tilde{\Phi}\theta^*$  converges to the fixed-point of  $\hat{\Pi}T^\pi$  as  $N \rightarrow \infty$ . For the rest of the paper, we denote as  $\hat{\Pi}$  the sample based feature space projector, called empirical projection. A variant of the LSTD algorithm that can be considered to have some resemblances to BLSTD (Tziortziotis and Dimitrakakis, 2017) adopted in our work is the sLSTD proposed by Geist and Pietquin (2010). The sLSTD algorithm is a statistical linearisation-based generalisation of LSTD that allows considering nonlinear parameterisations, i.e., neural networks.

After having approximated the action-value function  $Q_\theta^{\pi_{k-1}}$  of policy  $\pi_{k-1}$  (*policy evaluation* phase), an improved greedy policy  $\pi_k$  is generated (*policy improvement* phase). These two steps are repeated successively at each iteration, until we converge at an optimal policy.

### 3. Bayesian Least-Squares Policy Iteration

In this section, we introduce a Bayesian variant of LSPI algorithm, called BLSPI. Like LSPI, BLSPI is an offline, model-free and off-policy algorithm that belongs to the family of API algorithms and can be applied to control problems. In contrast to LSPI, that uses the standard LSTD algorithm at the policy evaluation step, BLSPI evaluates policies by using a Bayesian version of the LSTD algorithm, called BLSTD (Tziortziotis and Dimitrakakis, 2017). BLSTD has the advantage of probabilistic predictions as it quantifies our uncertainty about the value function instead of having only a point estimate over the unknown value function parameters like in the case of LSTD. In this section, we present a modified version of the BLSTD algorithm that learns the approximate action-value function  $Q^\pi$  of a given policy  $\pi$  instead of the state value function,  $V$ . In this way, we are able to select actions without the knowledge of the environment model.

Similarly to Tziortziotis and Dimitrakakis (2017), we consider the *empirical Bellman operator* that is given by the standard Bellman operator (1) plus some additive noise  $\epsilon = \mathcal{N}(0, \beta^{-1})$ :

$$\hat{T}^\pi Q_\theta^\pi = \mathcal{R} + \gamma P^\pi Q_\theta^\pi + N, \quad N \sim \mathcal{N}(0, \beta^{-1} \mathbf{I}).$$

Given a set of transitions  $\mathcal{D}$ , BLSTD seeks the action-value function parameters  $\theta$  that are invariant w.r.t. the composed operator  $\hat{\Pi}\hat{T}^\pi$ :

$$Q_\theta^\pi = \hat{\Pi}\hat{T}^\pi Q_\theta^\pi \Leftrightarrow \tilde{\Phi}^\top \mathcal{R} = \tilde{\Phi}^\top (\tilde{\Phi} - \gamma \tilde{\Phi}')\theta + \tilde{\Phi}^\top N.$$

It allows us to treat the empirical MSPBE as a linear regression model:

$$\mathbf{b} = A\theta + \tilde{\Phi}^\top N,$$

with its **log likelihood function** to be:

$$\ln p(\mathbf{b}|\theta, \beta) = \frac{k}{2} \ln(\beta) - \frac{1}{2} \ln(|\tilde{C}|) - \frac{k}{2} \ln(2\pi) - \frac{\beta}{2} E_{\mathcal{D}}(\theta).$$

It can be easily verified that by setting the gradient of the log likelihood with respect to model's parameters  $\theta$  equal to zero, we get the batch LSTD solution (2) (*maximum likelihood* solution).

A zero-mean isotropic Gaussian conjugate **prior distribution** over the model parameters  $\theta$  has been also considered, to model the *parametric* uncertainty (Mannor et al., 2004):  $p(\theta|\alpha) = \mathcal{N}(\theta|0, \alpha^{-1} \mathbf{I})$ , with the log posterior distribution to given as  $\ln p(\theta|\mathcal{D}) \propto -\frac{\beta}{2} E_{\mathcal{D}}(\theta) - \frac{\alpha}{2} \theta^\top \theta$ .

Therefore, maximising the posterior distribution w.r.t.  $\theta$  is equivalent to the minimisation of the MSPBE plus an  $\ell_2$ -penalty ( $\lambda = \alpha/\beta$ ). Thus, if we set hyperparameter  $\alpha$  to a large value, the total squared length of the parameter vector  $\theta$  will be encouraged to be small. Completing the squares of the log of the posterior distribution, we get that the **posterior distribution** is also Gaussian,

$$p(\theta|\mathcal{D}) = \mathcal{N}\left(\theta|\mathbf{m} \triangleq \beta S A^\top \tilde{C}^{-1} \mathbf{b}, S \triangleq (\alpha \mathbf{I} + \beta A^\top \tilde{C}^{-1} A)^{-1}\right), \quad (3)$$

where matrix  $\Sigma \triangleq A^\top \tilde{C}^{-1} A$  is always positive definite. The **predictive distribution** of the action-value function over a new state-action pair,  $(s^*, a^*)$ , is again Gaussian and it is given by:

$$p(Q_\theta^\pi(s^*, a^*)|s^*, a^*, \mathcal{D}) = \mathcal{N}(Q_\theta^\pi(s^*, a^*)|\phi(s^*, a^*)^\top \mathbf{m}, \phi(s^*, a^*)^\top S \phi(s^*, a^*)).$$

Finally, the generic bound of approximate policy iteration (Bertsekas and Tsitsiklis, 1996) holds also for the proposed BLSPI algorithm like in LSPI. Therefore, the performance of the sequence of policies produced by BLSPI is at most a constant multiple of  $\delta$  (positive scalar that bounds the policy evaluation errors over all iterations) away from the optimal one.

**Algorithm 1:** Randomised BLSPI (RBLSPi)

---

**Input:** Basis  $\phi, \gamma, \alpha, \beta, K$   
**Initialization:**  $A \leftarrow \mathbf{0}_{k,k}, \tilde{C} \leftarrow \mathbf{0}_{k,k}, b \leftarrow \mathbf{0}_k, m \sim \mathcal{N}(\mathbf{0}_k, I), \tilde{\theta} = m$

**begin**

Observe  $s_0$

**for**  $t = 0, \dots$  **do**

$a_t \in \arg \max_{a \in \mathcal{A}} \phi(s_t, a) \tilde{\theta}$  // Take action

Observe  $r_t, s_{t+1}$

$A \leftarrow A + \phi(s_t, a_t)(\phi(s_t, a_t) - \gamma \phi(s_{t+1}, \arg \max_{a \in \mathcal{A}} \phi(s_{t+1}, a)m))^\top$

$\tilde{C} \leftarrow \tilde{C} + \phi(s_t, a_t)\phi(s_t, a_t)^\top$

$b \leftarrow b + \phi(s_t, a_t)r_t$

**if** ( $t \bmod K == 0$ ) **then**

$S = (\alpha I + \beta A^\top \tilde{C}^{-1} A)^{-1}$

$m = \beta S A^\top \tilde{C}^{-1} b$

$\tilde{\theta} \sim \mathcal{N}(m, S)$  // Sample Gaussian posterior (Eq. 3)

---

**4. Randomised Bayesian Least-Squares Policy Iteration**

One of the most challenging tasks in reinforcement learning is the development of an agent able to adapt its behavior online by interacting with the environment. In this section, we introduce an online variant of Bayesian LSPI, called randomised BLSPI (RBLSPi). RBLSPi updates the agent's policy once every few transition samples based on the samples collected by itself up to this point. Similarly to online LSPI introduced by Buşoniu et al. (2010), RBLSPi carries out policy updates based on an incomplete evaluation of the current policy. This is a variation of policy iteration named as *optimistic policy iteration* (Tsitsiklis, 2003). In contrast to the standard offline BLSPI algorithm, in the case of RBLSPi algorithm two critical parameters should be considered: i) the number  $K \geq 1$  of collected transitions between successive policy improvements, and ii) the exploration strategy that should be followed. In the case where the parameter  $K$  has been selected to be too large, a bad policy is highly possible to be used for a long period that could affect the general performance of the algorithm. On the other hand, if policy is improved after each transition ( $K = 1$ ), the running time of the RBLSPi will be increased making that non applicable in some cases. An extensive analysis about the impact of the free parameter  $K$  in the performance of RBLSPi is presented in appendix.

The design of an efficient exploration strategy constitutes the core of an online agent that updates its belief based on samples collected by itself. In practice, the agent has to make sure that the collected samples cover the state and action spaces sufficiently. In any other case, the value function of the unvisited state-action pairs will be poorly estimated leading to imprecise policy improvements. One of the simplest exploration strategies is that of  $\epsilon$ -greedy: the agent selects a random action with probability  $\epsilon \in [0, 1]$  and the greedy action w.r.t. to its value function otherwise. Nevertheless, it has been shown that it could lead to highly inefficient learning (Osband et al., 2016).

Motivated by Thompson sampling and taking advantage of the BLSTD algorithm that considers our uncertainty over the estimated action-value function, RBLSPi algorithm explores the environment by sampling the value function randomly instead of selecting random actions. More specifically, a value function is sampled right after each policy evaluation step and actions are selected greedily based on the sampled value function thereafter. RBLSPi samples a value function by sam-

pling the posterior distribution (given by Eq. (3)) over model’s parameters. Therefore, RBL SPI tends to explore more non frequently selected actions with a highly uncertain value. As learning process evolves through time, our knowledge about the environment will be enriched as the number of visited state-action pairs will be increased. Therefore, our confidence about the estimated value functions will be increased, driving us to act more greedily with the passing of time. The main advantage of the specific exploration strategy over other strategies, i.e.,  $\epsilon$ -greedy, boltzmann, etc., is its ability to identify if a region has sufficiently explored by acting randomly mainly only on the areas that have not explored adequately yet. Finally, it should be noted that in RBL SPI the free parameter  $K$  has a dual role, as it determines how often we update our policy and select a new exploration policy. Randomised BLSPI is presented in detail in Algorithm 1.

## 5. Experiments

In this section, we formally present the results about the performance of the online RBL SPI algorithm. Through our analysis, we examine the exploration efficiency of the proposed Randomised Bayesian LSPI (RBL SPI) algorithm in four well-known continuous state, discrete-action, episodic domains. Due to space limitations, only a part of our empirical results are presented here. Specifically, we present only results about the mountain car environment and a sparse reward version of it that requires *deep exploration*. The full set of our results along with the experimental set-up is presented in detail in appendix.

In mountain car (Moore, 1990; Sutton and Barto, 1998), our objective is to drive an underpowered car up a steep road to the right hilltop ( $p \geq 0.5$ ) within at most 500 steps. The car state in this domain is described by two continuous variables, its position ( $p$ ) and its velocity ( $v$ ). At each time step, the agent can select between three possible actions: forward, reverse and zero throttle. In our experiments we have considered two different reward signals:

- In standard mountain car the immediate reward at each step is equal to  $-1$  except in the case where the agent reaches the goal (zero reward).
- The immediate reward is equal to  $1$  when the episode terminates and zero otherwise (Tang and Agrawal, 2018). We call this version sparse Mountain car.

In both cases, the car is positioned motionless ( $v_0 = 0$ ) to a uniformly randomly selected position close to the valley ( $-0.6 \leq p_0 \leq -0.4$ ) at the beginning of each episode. The discount factor is set to 0.99. An equidistant  $8 \times 8$  grid of RBFs over the state space plus a constant term is selected. This set of basis functions is replicated for each action, giving a total of 195 RBFs. In the task of mountain car the usage of an efficient exploration strategy is of high importance due to the fact that the goal cannot be easily reached by the agent. The necessity of an effective (deep) exploration scheme becomes more crucial in the case of the sparse reward mountain car environment. The empirical results of the RBL SPI algorithm on the the standard and sparse mountain car environments are presented in Figs 1a and 1b, respectively. Comparisons have been conducted with an onpolicy variant of online LSPI algorithm that uses the simple  $\epsilon$ -greedy exploration scheme. In the case of RBL SPI, we start by using a randomly selected policy (see Alg. 1) at the beginning of each run. For each experiment, we report the mean performance (average number of steps over 100 episodes) across 100 independent runs. For each average, we also plot the 95% confidence interval for the accuracy of the mean estimate with error bars. Additionally, we show the 90% percentile region of the runs. In the case of RBL SPI algorithm we set precision  $\beta$  equal to 0.1 and 1000 for the mountain car and sparse mountain car, respectively, and vary the precision hyperparameter  $\alpha$ .

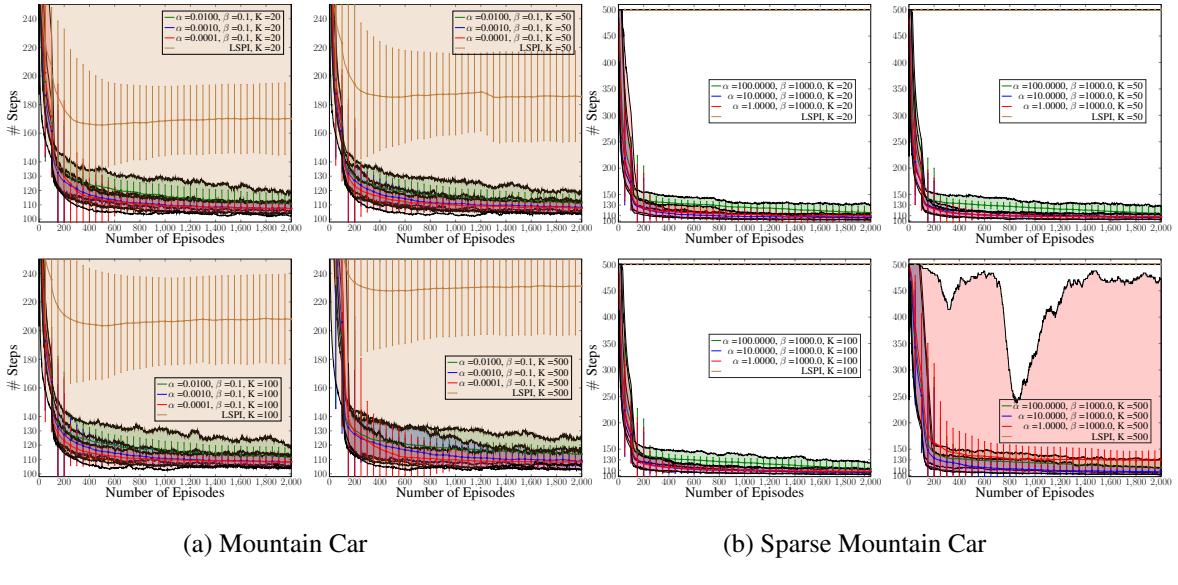


Figure 1: Performance of RBLSPF and online LSPF on the environments of Mountain Car and Sparse Mountain Car for varying parameter  $K$ .

Let us first to examine the impact of free parameter  $K$  in the performances of both algorithmic schemes. It should be reminded that parameter  $K$  defines the number of transitions that should be conducted between successive policy improvement steps (see Algorithm 1 for details). The following values have been examined for  $K$ : 20, 50, 100 and 500 (end of episode). As it was expected, the performance of both algorithms is better when  $K$  is set to a low value, i.e.,  $K = 20$ . It should be also mentioned that even if we set  $K$  equal to 500, RBL SPI is able to reach the goal in both environments. We have also examined the impact of precision parameter  $\alpha$  on the performance of RBL SPI. Our experiments show that it is more preferable to be set to a lower value if the parameter  $K$  is not equal to 500 (policy updates are executed only at the end of each episode). Nevertheless, it can be easily verified that RBL SPI achieves to discover good policies independent of the selection of the value of the precision  $\alpha$ . Our results also validates our claims about the efficiency of the exploration mechanism of RBL SPI. Actually, RBL SPI outperforms online LSPI in both environments. More specifically, RBL SPI discovers a near-optimal policy in both environments as our RBL SPI agent is able to reach the goal in less than 110 steps on average. On the other hand, the online LSPI stuck in the valley in the case of sparse mountain car, and it doesn't reach the goal in none of the runs.

## 6. Conclusion

In the present work, we have introduced a fully Bayesian version of the widely used least-squares policy iteration, called BLSPI. It is achieved by adopted BLSTD in the policy evaluation step of policy iteration. We further extended BLSPI on an online setting by proposing the RBLSPI algorithm. Taking advantage of the BLSTD algorithm to estimate our uncertainty over the value function estimations, RBLSPI explores efficiently its environment by sampling randomly value functions instead of selecting random actions. The efficiency of the proposed exploration strategy has been demonstrated experimentally in four continuous state-space environments, where comparisons have been conducted with the online LSPI algorithm that uses the simple  $\epsilon$ -greedy exploration strategy.

## References

- Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *CoRR*, abs/1802.04412, 2018.
- Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, (5):834–846, 1983.
- Dimitri P. Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011.
- Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.
- Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, 1996.
- Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003. ISSN 1532-4435.
- Lucian Buşoniu, Damien Ernst, Bart De Schutter, and Robert Babuška. Online least-squares policy iteration for reinforcement learning control. In *Proceedings of the 2010 American Control Conference*, pages 486–491, 2010.
- Matthieu Geist and Olivier Pietquin. Statistically linearized least-squares temporal differences. In *International Congress on Ultra Modern Telecommunications and Control Systems*, pages 450–457, 2010.
- Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Found. Trends Mach. Learn.*, 8(5-6):359–483, 2015.
- Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
- Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.
- Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Finite-sample analysis of LSTD. In *International Conference on Machine Learning*, pages 615–622, 2010.
- Lihong Li, Michael L. Littman, and Christopher R. Mansley. Online exploration in least-squares policy iteration. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 733–739, 2009.
- Shie Mannor, Duncan Simester, Peng Sun, and John N. Tsitsiklis. Bias and variance in value function estimation. In *International Conference on Machine Learning*, 2004.
- Andrew W. Moore. Efficient memory-based learning for robot control. Technical report, 1990.

- A. Nedić and D. P. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1):79–110, 2003.
- Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems 26*, pages 3003–3011, 2013.
- Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2377–2386, 2016.
- Ian Osband, Daniel Russo, Zheng Wen, and Benjamin Van Roy. Deep exploration via randomized value functions. *CoRR*, 2017.
- Martin L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New Jersey, US, 2005.
- Malcolm J. A. Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML, pages 943–950, 2000.
- Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1038–1044, 1996.
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT, 1998.
- Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning*, pages 993–1000, 2009.
- Yunhao Tang and Shipra Agrawal. Exploration by distributional reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2710–2716, 2018.
- William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.
- Ahmed Touati, Harsh Satija, Joshua Romoff, Joelle Pineau, and Pascal Vincent. Randomized value functions via multiplicative normalizing flows. *CoRR*, 2018.
- John N. Tsitsiklis. On the convergence of optimistic policy iteration. *The Journal of Machine Learning Research*, 3:59–72, 2003.
- Nikolaos Tziortziotis and Christos Dimitrakakis. Bayesian inference for least squares temporal difference regularization. In *Machine Learning and Knowledge Discovery in Databases - European Conference*, pages 126–141, 2017.
- Nikolaos Tziortziotis, Christos Dimitrakakis, and Konstantinos Blekas. Linear bayesian reinforcement learning. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 1721–1728, 2013.

Nikolaos Tziortziotis, Christos Dimitrakakis, and Konstantinos Blekas. Cover tree bayesian reinforcement learning. *Journal of Machine Learning Research*, 15(1):2313–2335, 2014. ISSN 1532-4435.

## Appendix: Empirical Analysis

In this section, we present the full set of our empirical results about the two proposed RL algorithmic schemes introduced in our manuscript, BLSPI (Sec. 3) and RBLSPPI (Sec. 4). Actually, our empirical analysis is divided in two main parts:

- First, we examine the ability of the Bayesian LSPI algorithm to discover the same (or a close) policy returned by the original LSPI algorithm. For this purpose, we consider the simple chain walk domain proposed by Lagoudakis and Parr (2003).
- Second, we examine the exploration efficiency of the proposed Randomised Bayesian LSPI (RBLSPPI) algorithm on four well-known continuous state, discrete-action, episodic domains. Comparisons have been conducted with the online LSPI algorithm that follows the simple  $\epsilon$ -greedy strategy (Buşoniu et al., 2010).

### Bayesian LSPI Performance

As aforementioned, the simple chain walk (Lagoudakis and Parr, 2003) environment has been considered to examine how close the policies returned by the Bayesian LSPI algorithm are to those returned by the vanilla LSPI algorithm. This is a discrete 20-state chain<sup>1</sup> with its boundaries to be dead-ends. There are two noisy available actions, i.e., “left” (L) and “right” (R). The probability of an action to fail is equal to 0.1, moving the agent in the opposite direction. A reward of +1 is given only at the boundaries of the chain (states 1 and 20). The discount factor is set to 0.9. Similarly to Lagoudakis and Parr (2003), we have used a polynomial of degree 4 for approximating the value function for each one of the two actions (10 basis functions in total have been used).

Figure 2 illustrates the policies returned after a single run of BLSPI and LSPI on the chain walk problem, respectively. In both cases, the initial policy was selected to be the policy that chooses left action at each state. Furthermore, both algorithms use the same single set of samples (5000 in total) collected from a single episode with the actions to be chosen uniformly at random. The optimal policy in this domain is to go left in states 1 – 10 and right in states 11 – 20. Therefore, it can be easily verified that both algorithms need only 6 iterations to discover the optimal policy. Additionally, this example shows that the performance of BLSPI is quite close to that of LSPI.

### Randomised Bayesian LSPI Performance

Let us now present our results for the online RBLSPPI algorithm that constitutes the main contribution of this work. The main advantage of RBLSPPI is its efficient exploration scheme that is based on the idea of the randomised value function (Osband et al., 2016). To examine the exploration efficiency of the proposed Randomised Bayesian LSPI (RBLSPPI) algorithm, we have considered four well-known continuous state, discrete-action, episodic domains: i) (sparse) Mountain Car, ii) Inverted Pendulum, iii) Cart Pole, and iv) Puddle World. Comparisons have been conducted with an onpolicy variant of online LSPI algorithm that uses the simple  $\epsilon$ -greedy exploration scheme. For each experiment, we report the mean performance (average return or number of steps over 100 episodes) across 100 independent runs. For each average, we also plot the 95% confidence interval for the accuracy of the mean estimate with error bars. Additionally, we show the 90% percentile region of the runs.

---

1. In chain walk domain, we have used the LSPI code which is freely available at: <https://www2.cs.duke.edu/research/AI/LSPI/>.

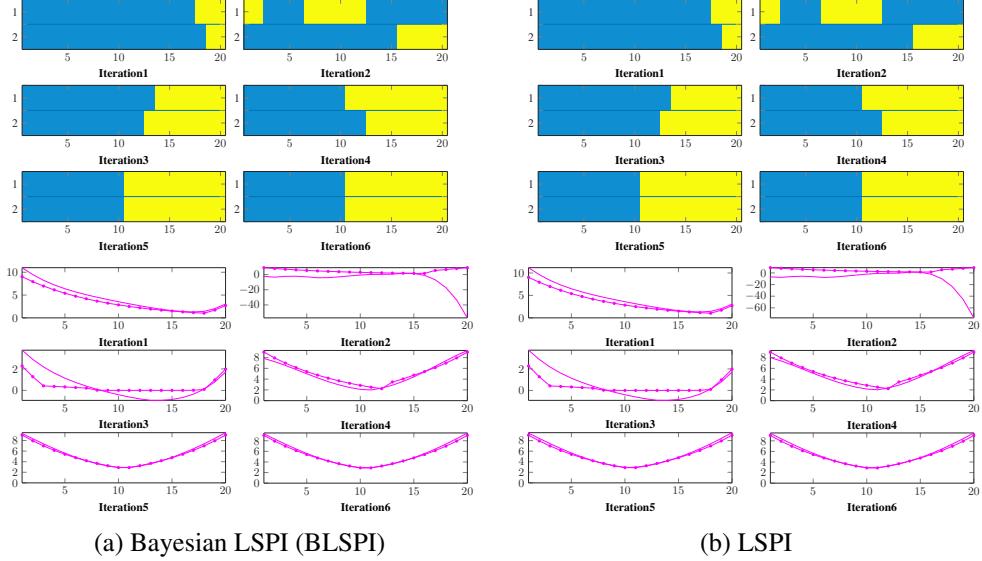


Figure 2: BLSPI and LSPI iterations on the 20-state chain problem. Top: Improved policy after each iteration of LSPI and Bayesian LSPI, respectively (R action - yellow shade; L action - blue shade; BLSPI/LSPI - top stripe; exact - bottom stripe). Bottom: State value function  $V^\pi(s)$  of the policy being evaluated at each iteration (BLSPI/LSPI approximation - solid line; exact values - dotted line)

**Mountain Car (Moore, 1990; Sutton and Barto, 1998)** The goal in this task is to drive an underpowered car up a steep road to the right hilltop ( $p \geq 0.5$ ) within at most 500 steps. The car state in this domain is described by two continuous variables, its position ( $p$ ) and its velocity ( $v$ ). At each time step, the agent can select between three possible actions: forward, reverse and zero throttle. In our experiments we have considered two different reward signals:

- In standard mountain car the immediate reward at each time step is equal to  $-1$  except in the case where the agent reaches the goal (zero reward).
- The immediate reward is equal to  $1$  when the episode terminates and zero otherwise (Tang and Agrawal, 2018). We call this version sparse Mountain car.

In both cases, the car is positioned motionless ( $v_0 = 0$ ) to a uniformly randomly selected position close to the valley ( $-0.6 \leq p_0 \leq -0.4$ ) at the beginning of each episode. The discount factor is set to  $0.99$ . An equidistant  $8 \times 8$  grid of RBFs over the state space plus a constant term is selected. This set of basis functions is replicated for each action, giving a total of 195 RBFs.

In the task of mountain car the usage of an efficient exploration strategy is of high importance due to the fact that the goal cannot be easily reached by the agent. The necessity of an effective (deep) exploration scheme becomes more crucial in the case of the sparse reward mountain car environment. The empirical results of the RBL SPI and online LSPI algorithms on the the standard and sparse mountain car environments are presented in Figures 3 and 4, respectively. In both versions of mountain car environment, we have examined the effects in the performance of the RBL SPI algorithm of varying the precision parameters  $\alpha$  and  $\beta$ . Let us first to examine the impact

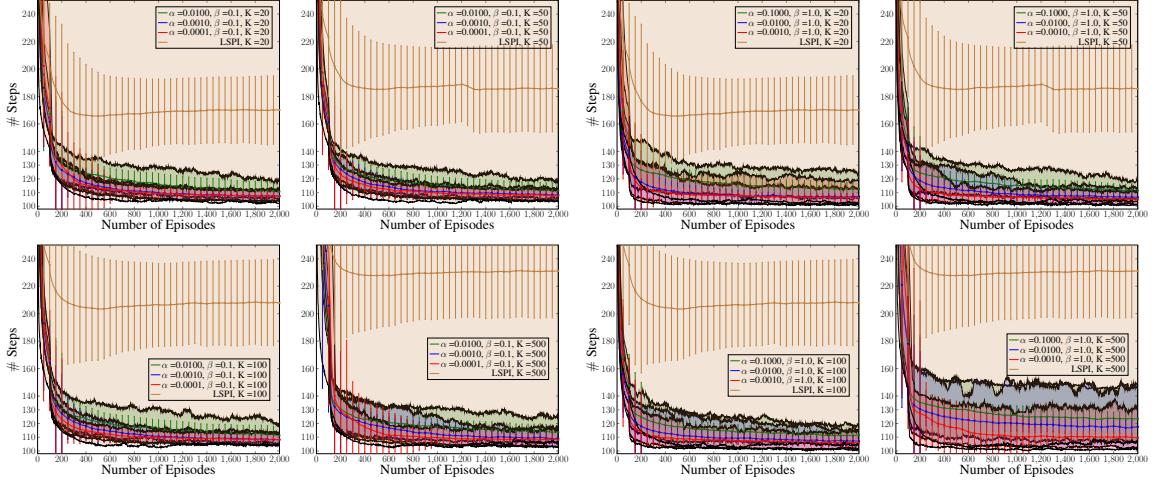


Figure 3: Performance of RBLSPI and online LSPI on the **mountain car** environment for varying parameter  $K$ . The performance of the RBLSPI algorithm is also presented for varying hyperparameters  $\alpha$  and  $\beta$ .

of free parameter  $K$  in the performances of both algorithmic schemes. It should be reminded that parameter  $K$  defines the number of transitions that should be conducted between successive policy improvement steps (see Algorithm 1 for details). For this purpose, the following values have been examined for  $K$ : 20, 50, 100 and 500 (end of episode). As it was expected, the performance of both algorithms is better when  $K$  is set equal to a low value, i.e.,  $K = 20$ . It should be also mentioned that even if we set  $K$  equal to 500, the RBLSPI is able to reach the goal in both environments. We have also examined the impact of the precision parameter  $\alpha$  on the performance of RBLSPI. Our experiments show that it is more preferable to be set to a lower value if the parameter  $K$  is not equal to 500 (policy updates are executed only at the end of each episode). Nevertheless, it can be easily seen that RBLSPI achieves to discover good policies independent of the selection of the value of precision parameter  $\alpha$ . Regarding the precision parameter  $\beta$ , it impacts a lot the performance of the RBLSPI. More specifically, the setting of the precision parameter  $\beta$  depends heavily on the reward signal of the environment. For instance, on sparse mountain car we should set  $\beta$  to a high value in contrast to the standard mountain car where the best performance is achieved when  $\beta$  is set to 0.1. Our results also validate our claims about the efficiency of the exploration mechanism of RBLSPI. Actually, RBLSPI outperforms online LSPI in both environments. More specifically, RBLSPI discovers a near-optimal policy in both environments as our RBLSPI agent is able to reach the goal in less than 110 steps on average. On the other hand, the online LSPI stuck in the valley in the case of sparse mountain car, and it doesn't reach the goal in none of the runs.

**Inverted Pendulum (Lagoudakis and Parr, 2003)** In this domain our target is to keep a pendulum balanced for 3000 steps by applying forces of a fixed magnitude (50 Newtons). The state space is described by two continuous variables, the vertical angle ( $\theta$ ) and the angular velocity ( $\dot{\theta}$ ) of the pendulum. There are three noisy actions where agent can apply to keep the pendulum balanced: no force, left force or right force. A uniform noise in  $[-10, 10]$  is applied to the chosen action. A zero

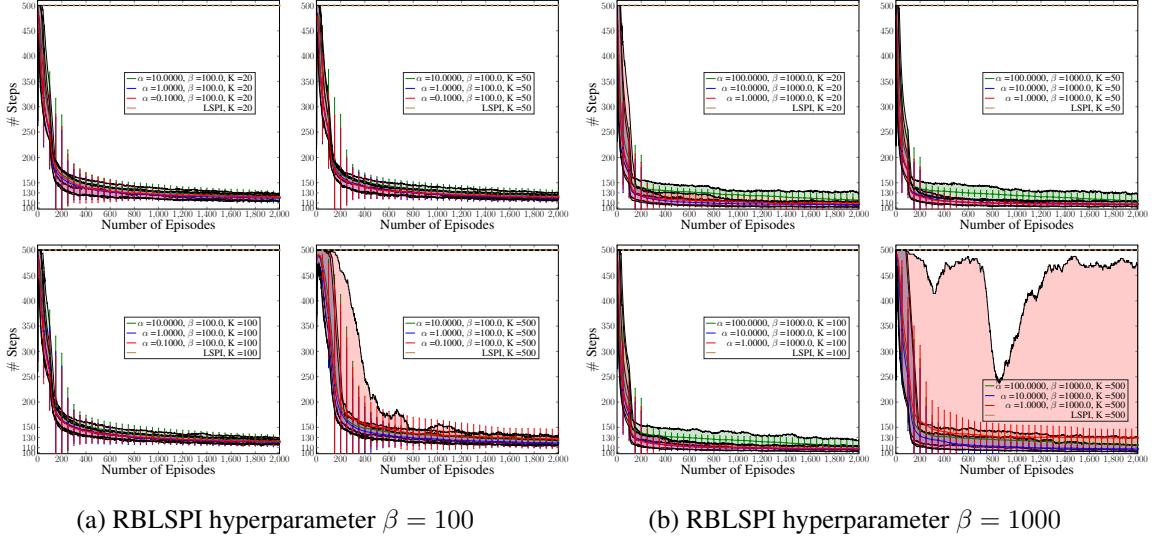


Figure 4: Performance of RBL SPI and online LSPI on the **sparse reward mountain car** environment for varying parameter  $K$ . The performance of the RBL SPI algorithm is also presented for varying hyperparameters  $\alpha$  and  $\beta$ .

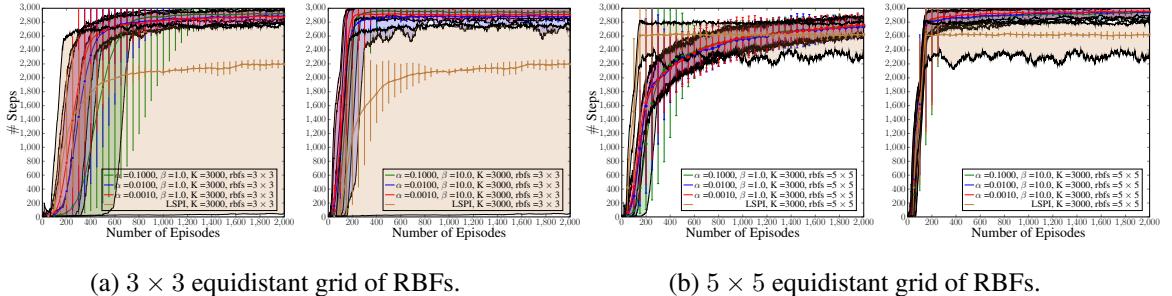


Figure 5: Performance of RBL SPI and online LSPI on the **inverted pendulum** environment on two different grids of RBFs. The performance of the RBL SPI algorithm is also presented for varying hyperparameters  $\alpha$  and  $\beta$ .

reward is received at each time step except in the case where the pendulum falls ( $|\theta| \leq \pi/2$ ). If pendulum falls ( $|\theta| \geq \pi/2$ ) the episode ends and a penalty ( $-1$ ) is received. The discount factor of the process is set equal to 0.95. Two different grids of RBFs over the state space have been considered in our analysis, a  $3 \times 3$  and a  $5 \times 5$  equidistant grid. Therefore, the total number of basis functions at each case is equal to 30 and 78, respectively.

Figure 5 illustrates the performance of the evolution of the policies discovered by RBL SPI and online LSPI algorithms in the environment of inverted pendulum. More specifically, Figs. 5a and 5a show the results of our analysis when a  $3 \times 3$  and a  $5 \times 5$  equidistant grid of RBFs is considered, respectively. It seems that RBL SPI achieves to discover optimal or near-optimal policies independent of the hyperparameters setting. In the case where we set hyperparameter  $\beta$  equal to 10.0, our agent performs much better than in the case of  $\beta = 1.0$ . Also, our analysis indicates that the agent's behavior is better by setting  $\alpha$  to a high value, i.e.  $\alpha = 0.1$ . It worths also to be noted

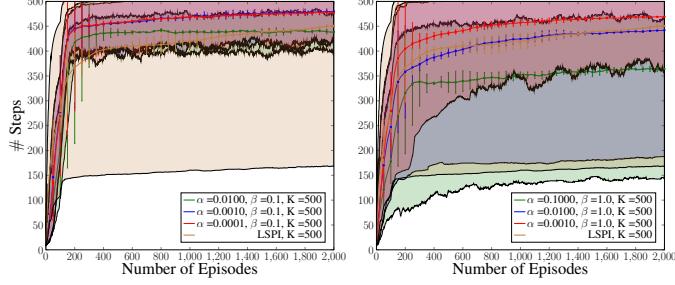


Figure 6: Performance of RBLSPI and online LSPI on the **cart pole** environment. The performance of the RBLSPI algorithm is also presented for varying hyperparameters  $\alpha$  and  $\beta$ .

that the RBLSPI outperforms the online LSPI algorithm in both cases ( $3 \times 3$  grid (Fig. 5a) and  $5 \times 5$  grid (Fig. 5b)). More specifically, the online LSPI doesn't achieve to discover the optimal policy even after 2000 episodes. On the contrary, the RBLSPI algorithm discover an optimal policy after around 200 episodes if we set the precision parameter  $\beta$  equal to 10.0.

**Cart Pole (Barto et al., 1983; Sutton and Barto, 1998)** Our objective in this task is to keep a pole hinged to a cart moving along a track from falling over. The agent controls the system by applying a force of  $+1$  or  $-1$  to the cart. The state space is described by four continuous variables: cart position ( $p$ ), cart velocity ( $v$ ), pole angle ( $\theta$ ) from vertical, and pole angular velocity ( $\dot{\theta}$ ). A failure occurs even if the pole falls ( $|\theta| \geq \pi/6$ ) or if the cart runs out of the track ( $|p| \geq 2.4$ ). At each episode, the starting state of the agent is selected uniformly over  $[-0.05, 0.05]^4$ . The reward is equal to  $+1$  for every time step where the pole remains upright and the cart is kept inside the track. The discount factor is 0.99 and the maximum number of steps per episode is equal to 500. An equidistant  $3 \times 3$  grid of RBFs over the state space plus a constant term is selected (164 basis functions in total).

Figure 6 illustrates the performance of the RBLSPI and the online LSPI algorithms in cart pole environment. Actually, our experiments show that RBLSPI is able to discover near optimal policies as it is able to balance the pole for more than 480 steps and to keep the cart on track. Moreover, the performance of the RBLSPI is much better (especially if we set  $\beta$  to 0.1) compared to the one of the online LSPI. It should be also noted that the performances of both the RBLSPI and the online LSPI are highly correlated to the number of used RBFs. In this way, we intend that the RBLSPI algorithm will be able to discover the optimal solution in the case where we increase the resolution of our feature space (i.e., by using a  $5 \times 5$  equidistant grid of RBFs).

**PuddleWorld (Sutton, 1996)** The goal for the agent in this domain is to reach an area located at the upper right corner, avoiding two puddles. Actually, the agent is located in a continuous 2-dimensional terrain ( $[0, 1]^2$ ) that contains two oval puddles. There are four discrete actions: up, down, left, and right. By selecting an action, the agent moves by 0.05 in the corresponding direction, up to the limits of the area. A random white gaussian noise with  $std = 0.01$  is also added to the motion along each dimension. A negative reward ( $-1$ ) is received at each time step plus a penalty between 0 and  $-40$  (depending on the proximity to the middle of the puddle) in a puddle is entered by the agent. A new episode starts even if the goal is reached or after 500 time steps. Starting states are chosen uniformly over  $[0, 1]^2$  and are considered to be out of a puddle. The discount factor is

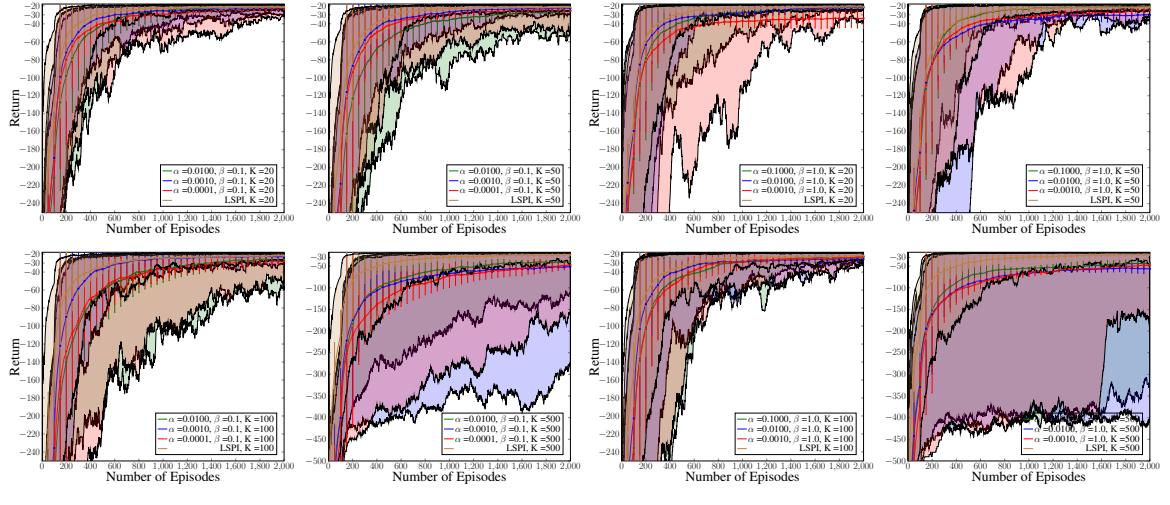


Figure 7: Performance of RBLSPI and online LSPI on the **puddle world** environment for varying parameter  $K$ . The performance of the RBLSPI algorithm is also presented for varying hyperparameters  $\alpha$  and  $\beta$ .

$\gamma = 0.99$ . An equidistant  $8 \times 8$  grid of RBFs over the state space plus a constant term is selected (260 basis functions are used in total).

In the puddle world environment, we consider the average undiscounted return received by our agent at each episode, in contrast to the previous domains where we considered the total number of steps per episode. Our results for puddle world are presented in Figure 7. Actually, Figure 7a shows the performance of RBLSPI in the case where  $\beta = 0.1$ , while Figure 7b corresponds to  $\beta = 1.0$ . Our results show that frequent improvement of our policies increase the performance of the proposed RBLSPI algorithm. This happens due to the fact that after each policy improvement step, we sample a new exploration policy (followed to select actions thereafter) though the sampling of posterior distribution over model's parameters (see Eq.3). In this way, we reinforce the exploration behavior of our agent. It worths also to be mentioned that the performance of the online LSPI is quite good in contrast to its performance on the other domains that have been examined previously. Actually, the onpolicy variant of online LSPI algorithm performs quite close or even better compared to RBLSPI algorithm. We should also mention that the performance of the original offpolicy (online) LSPI algorithm (Buşoniu et al., 2010) is poor and it cannot discover a good policy even after a huge number of episodes. The main conclusion of our empirical analysis is that deep exploration is necessary especially in the case where the reward signal is sparse or the target cannot be easily reached by the agent.