To appear in the *Journal of Experimental & Theoretical Artificial Intelligence*
Vol. 00, No. 00, Month 20XX, 1–16

Approximate Policy Iteration using Regularized Bellman Residuals Minimization

G. Esposito[a*] and M. Martin[a]

[a] *Universitat Politecnica de Catalunya, Barcelona, Spain*;

(*v1.0 released October 2014*)

In this paper we present an Approximate Policy Iteration (API) method called $API-BRM_\epsilon$ using a very effective implementation of incremental Support Vector Regression (SVR) to approximate the value function able to generalize in continuous (or large) space Reinforcement Learning (RL) problems. RL ia a methodology able to solve complex and uncertain decision problem usually modeled as Markov Decision Problem (MDP). $API-BRM_\epsilon$ is formalized as a non-parametric regularization problem based on an outcome of the Bellman Residual Minimization (BRM) which is able to minimize the variance of the problem. $API-BRM_\epsilon$ is incremental and can be applied to RL using the on-line agent interaction framework. Based on non-parametric SVR $API-BRM_\epsilon$ is able to find the global solution of the problem with convergence guarantees to the optimal solution. A value function should be defined to find the optimal policy specifying the total reward that an agent might expect in its current state taking one action. Therefore the agent will use the value function to choose the action to take. Some experimental evidence and performance for well known RL benchmarks are presented.

Keywords: Reinforcement Learning, Support Vector Machine, Approximate Policy Iteration, Regularization, Regression

1. Introduction

In RL an autonomous agent interacting with the environment learns how to take correct action for every situation in order to reach its goal. RL gives a method able to solve difficult and uncertain sequential decision problems, which could be quite challenging in real-world applications. RL problem is often modeled as a MDP, which has been deeply studied in the literature. The main particularity of RL algorithms with respect to other approaches to MDPs is that the RL agent learns optimal policies from experiences without knowing the parameters of the MDP. Usually to find the optimal policy a value function should be defined to specify the total reward an agent might expect at its current state and taking one action. The agent will choose the action to take by using this value function. However if one wants to handle problems with continuous or very large state spaces, generalization should be used. Generalization property of RL algorithms is an important factor to determine prediction performance in such cases.

Main problems using generalization in approximate RL are convergence guarantees and quality of the solution. Parametric methods have been considered in the literature, showing the advantage of fast and easy learning mechanisms. However, they present the inherent problem that in some cases the solution of the problem might not be expressed with the given architecture and the number of chosen parameters. Candidates for non

*Corresponding author. Email: gesposit@lsi.upc.edu

parametric value function approximation are SVR known to have good properties over the generalization ability. SVR being a convex optimization problem, do not suffer from sub-optimality, finding always the global optimal solution to the approximation problem. As non parametric learning method, they are also able to automatically adapt to the complexity of the problem. Using non parametric methods for approximate RL allows to choose the capacity of the approximation function space by means of a fixed kernel function while the number supports can be determined as needed in order to find the required solution.

The algorithm presented in this paper is an instance of API. Like in policy iteration, the algorithm repeatedly computes an evaluation function of the policy of the previous step and then uses this evaluation function to compute the next improved policy. In order to avoid the need of learning a model, action value functions are computed, making the policy improvement step simple, like in the Least-Squares Policy Iteration (LSPI) algorithm of (Lagoudakis and Parr, 2003). LSPI relies on least squares temporal difference learning (LSTD) while we build our algorithm using BRM. Major novelty of our algorithm is based on the exact incremental SVR in the context of BRM with approximate policy iteration. The idea of using Bellman residuals in policy iteration goes back to (Baird, 1995) who proposed it for computing approximate state value functions given the model of a finite-state and action MDP using a quadratic loss function. Small Bellman errors might yield a good approximation of the policy evaluation function, which in turn may imply a good final performance. One major obstacle of using BRM when learning without a model is that the sample based approximation using a least squares loss function based on the data from a single trajectory of the behavior policy, is not an unbiased estimate of the Bellman residual. Our algorithm using $\epsilon-$insensitive loss function might solve this problem. As main condition for the convergence of our method we ask that the trajectory should be sufficiently representative and rapidly mixing. We also require that the states in the trajectory follow a stationary distribution. The mixing condition is essential for efficient learning and in particular we use the exponential $\beta-$mixing condition.

2. Background and notation

For a space Ω with a σ-algebra σ_Ω define $\mathcal{M}(\Omega)$ the set of all probability measure over σ_Ω and $\mathcal{B}(\Omega)$ the space of bounded measurable function w.r.t. σ_Ω and $\mathcal{B}(\Omega, L)$ the space of bounded measurable function with bound $0 < L < \infty$. Hence we may introduce

Definition 2.1: *(**Continuous State MDP**) A continuous state and finite action discounted **MDP** can be defined as a tuple (S, A, P, γ), with the following definitions for its contents:*

- *S is a measurable state space where $s_t \in S$ denotes the state the agent is in at time t.*
- *A is a finite set of available actions where $a_t \in A$ denotes the action the agent performs at time t.*
- *$P : S \times A \to \mathcal{M}(\mathrm{R} \times S)$ is a mapping that when evaluated at $(s, a) \in S \times A$ gives the distribution over $\mathrm{R} \times S$ denoted as $P(r, s^{'}|s, a)$.*
- *Marginals of P can be defined as $\mathcal{P}(\cdot|s, a) = \int_{\mathrm{R}} P(dr, \cdot|s, a)$ denotes the probability of ending up in state $s^{'}$ when performing action a in state s (transition probability)*
- *$\mathcal{R}(\cdot|s, a) = \int_S P(\cdot, ds^{'}|s, a)$ is a reward function denoting the expected reward when the agent transitions from state s to state $s^{'}$ after performing action a.*
- *The immediate expected reward is defined as $r(s, a) = \int_{\mathrm{R}} r\mathcal{R}(dr|s, a) = \mathrm{E}[r|s, a]$*
- *$\gamma \in [0, 1]$ is a discount factor.*

At stage t an action $a_t \in A$ is selected by the agent controlling the process and in response the pair $(r_t, s_t^{'})$ is drawn from the distribution $P(r, s^{'}|s_t, a_t)$ i.e $(r_t, s_t^{'}) \sim P(r, s^{'}|s_t, a_t)$ where r_t is the reward the agent receives and $s_t^{'}$ the next MDP state. The procedure continue leading to a random trajectory $\xi_t = \{s_1, a_1, r_1, s_2, a_2, r_2, ...\} \in \Xi$ where Ξ denotes the space of all possible trajectories.

For a MDP an agent consists mainly of an action selection policy such that $a_t = \pi(s_t)$. A stationary stochastic policy maps states to distributions over the action space $\pi_t : S \to \mathcal{M}(A)$ with $\pi_t(a|s)$ denoting the probability that the agent will select the action a to perform in state s at time t. We will use $\pi(s)$ to refer to the probability distribution or the probability mass function of the actions in state s. Stochastic policies are also called soft when they do not commit to a single action per state. $\pi(a|s)$ stands for the probability that the soft policy chooses action a in state s. An $\epsilon-$greedy policy is a soft policy which for some $0 \leq \epsilon \leq 1$ picks deterministically a particular action with probability $1 - \epsilon$ and a uniformly random action with probability ϵ. We will then use $a \sim \pi(\cdot|s)$ to indicate that action a is chosen according to the probability function in state s.

Definition 2.2: *(Value function) For an agent following the policy π considering the sequence of rewards $\{r_t : t \geq 1\}$ when the MDP is started in the state action $(s_1, a_1) \sim \nu(s, a) \in \mathcal{M}(S \times A)$ the action value function Q^π is defined as $Q^\pi(s, a) = \mathrm{E}\{\sum_{t=1}^{\infty} \gamma^{t-1} r_t | \ s_1 = s, a_1 = a, \pi\}$ The optimal action value $Q^*(s, a) = \sup_\pi Q^\pi(s, a) \quad \forall\, (s, a) \in S \times A$. A policy π^* is optimal whenever achieves the best value in every state and a policy $\pi = \hat{\pi}(\cdot, Q)$ is greedy w.r.t. an action value function Q if $\forall\, s \in S$ we choose $\pi(s) = \arg\max_{a \in A}\ Q(s, a)$.*

Definition 2.3: *(Bellman Operator) The Bellman operator can be defined as $T^\pi : \mathcal{B}(S \times A) \to \mathcal{B}(S \times A)$ for action value function is defined as $(T^\pi Q)(s, a) = r(s, a) + \gamma \int \mathcal{P}(ds^{'}|s, a) Q(s^{'}, \pi(s^{'})) = \mathrm{E}[r + \gamma \sum_{a^{'} \in A} \pi(a^{'}|s^{'}) Q(s^{'}, a^{'})]$ where $\mathcal{B}(\cdot)$ represents the space of bounded measurable functions. Given a policy π a fixed point of the Bellman operator is the action value function $Q^\pi = T^\pi Q^\pi$. Bellman optimality operator can be defined for action value function as $(T^* Q)(s, a) = r(s, a) + \gamma \int \mathcal{P}(ds^{'}|s, a) \max_{a^{`} \in A} Q(s^{'}, a^{`})$*

Definition 2.4: *(Reproducing Kernel Hilbert Spaces) Consider a subset of measurable functions $\mathcal{F} :\to \mathrm{R}$ and a subset of vector values measurable functions $\mathcal{F}^{|A|} : S \times A \to \mathrm{R}^{|A|}$ such that $\mathcal{F}^{|A|} = \{(Q_1, ..., Q_{|A|}) : Q_i \in \mathcal{F},\ i = 1, ..., |A|\}$ called the hypothesis space $\mathcal{H}$ which can be chosen as Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H} : S \times A \to \mathrm{R}$ defined in $S \times A$ with the inner product $\langle \cdot, \cdot \rangle$ and characterized by a symmetric positive definite function $\kappa : (S \times A) \times (S \times A) \to \mathrm{R}$ called reproducing kernel, continuous in $S \times A$ such that for each $(s, a) \in S \times A$ the following reproducing property holds: $Q(s, a) = \langle Q(\cdot), \kappa(\cdot, s, a) \rangle_{\mathcal{H}} \quad \kappa(\cdot, s, a) \in \mathcal{H}$ assuming as measurable function space $\mathcal{F}^{|A|} = \mathcal{H}$. $\mathcal{H}$ is the closure of the linear span of the set of functions $\mathbf{\Phi}_{span} = \{\Phi(s, a) = \kappa(\cdot, s, a) \quad (s, a) \in S \times A\}$ considering the map from $\Phi : S \times A \to C^{S \times A}$ which denotes the function that assigns the value $\kappa(s_t, a_t, s, a)$ to $(s_t, a_t) \in S \times A$.*

The kernel function $\kappa(\cdot, \cdot)$ plays an important role in any kernel-based learning method mapping any two elements from a space of input patterns to the real numbers taken the state space of the MDP, and can be thought of as a similarity measure on the input space. In the derivation of kernel methods, the kernel function arises naturally as an inner product in a high-dimensional features space. Hence the kernel satisfy several important properties of an inner product: it must be symmetric and positive semidefinite, meaning that the associated Gram matrix $K_{ij} = \kappa(s_i, a_i, s_j, a_j)$ must be positive semidefinite. A kernel that satisfies these properties is said to be admissible. An important aspect of any method solving RL problems is the way that data are collected and processed. The *data*

collection setting can be categorized as *online* or *offline* and the *data processing* method can be categorized as *batch* or *incremental*. The online sampling setting is when the agent chooses the action sequence $a_t \sim \pi_t(\cdot|s_t)$ and directly influences how the data stream is generated $D_n = \{(s_1, a_1, s_1^{'}, r_1), ..., (s_n, a_n, s_n^{'}, r_n)\}$ which we may assume as a stationary and in general non-i.i.d. process. The stochastic functions $\pi_t(\cdot|s)$, whenever evaluated for the states $s^{'}$ in D_n define the stochastic process $\Pi_n = \{\pi_1(\cdot|s_1^{'}), ..., \pi_n(\cdot|s_n^{'})\}$. Assuming controlled mixing condition for the non-i.i.d. process, allows for the generalization of strictly i.i.d. scenarios. We may define the ordered multisets $\{(s_1, a_1), ..., (s_n, a_n)\}$ and $\{(r_1, s_1^{'}), ..., (r_1, s_1^{'})\}$ with $a_t = \pi_{b_t}(s_t)$ and $s_t \sim \nu_s(s)$ and $\nu_s(s) \in \mathcal{M}(S)$ while $(r_t, s_t^{'}) \sim P(r, s^{'}|s_t, a_t)$. π_b is a *behavior* stationary policy producing $(s_t, a_t) \sim \nu(s, a)$ with $\nu(s, a) \in \mathcal{M}(S \times A)$ the resulting state action distribution. The collected data D_n can be used to define the empirical operators and can be thought of as the empirical approximation to the true operators.

Definition 2.5: *(Empirical Bellman operators) Given the policy π consider the D_n data set the empirical Bellman operator is defined as $(\hat{T}^{\pi}Q)(s_t, a_t) = r_t + \gamma \sum_{a' \in A} \pi(a^{'}|s_t^{'})Q(s_t^{'}, a^{'})$ while the empirical Bellman optimality operator $\hat{T}^{*} : Z_n \to \mathrm{R}^n$ is defined as $(\hat{T}^{*}Q)(s_t, a_t) = r_t + \gamma \max_{a' \in A} Q(s_t^{'}, a^{'})$. This provide an unbiased estimate of the Bellman operator where given the policy π for any fixed bounded measurable deterministic function $Q : S \times A \to \mathrm{R}$ it holds that $\mathrm{E}[\hat{T}^{\pi}Q(s_t, a_t)|s_t, a_t] = T^{\pi}Q(s_t, a_t)$ and also $\mathrm{E}[\hat{T}^{*}Q(s_t, a_t)|s_t, a_t] = T^{*}Q(s_t, a_t)$ for $1 \leq t \leq n$.*

We may also use the symbol $\mathrm{E}_n[f] = \frac{1}{n}\sum_{t=1}^{n} f(z_t)$ as a shorthand for the empirical version of the expectation operator applied over a function $f(z)$ using the data set $Z_n = \{z_1, ..., z_n\}$.

3. Approximate Policy Iteration using Bellman Residuals Minimization

Policy iteration is a method of discovering the optimal policy for any given MDP, providing an iterative procedure in the policies space. PI discovers the optimal policy by generating a sequence of monotonically improving policies. Each iteration consists of two phases: *policy evaluation* which computes the action value function Q_k of the current policy π_k by solving the linear system of the Bellman equations and *policy improvement* defining the improved greedy policy π_{k+1} over Q^{π_k} through $\pi_{k+1} = \arg\max_{a \in A} Q_k(s, a)$. The value function Q_k is typically chosen to be such that $Q_k \approx T^{\pi_k}Q_k$, i.e., it is an approximate fixed point of T^{π_k}. The policy π_{k+1} is at least as good as π_k if not better. These two steps are repeated until there is no change in the policy in which case the iteration has converged to the optimal policy, often in a surprisingly small number of iterations. The guaranteed convergence of policy iteration to the optimal policy relies heavily upon a tabular representation of the value function, exact solution of the Bellman equations, and tabular representation of each policy. Exact representations and methods are impractical for large state and action spaces. In such cases, approximation methods are used. Approximations in the policy iteration framework can be introduced in the representation of the value function or the policy. The crucial factor for a successful approximate algorithm is the choice of the approximation architecture and this form of policy iteration is known as Approximate Policy Iteration (API).

A way to implement API is using BRM and in this case API proceeds at iteration k evaluating π_k choosing Q_k such that the Bellman residuals $\epsilon_k^{BR} = |Q_k - T^{\pi_k}Q_k|$ to be small (i.e. is the approximate fixed point of T^{π_k}). API calculates $\pi_{k+1} = \hat{\pi}(\cdot, Q_k)$ producing the sequence $Q_0 \to \pi_1 \to Q_1 ....$ and for the sequence $\{Q_k\}_{k=0}^{K-1}$ the Bellman Residuals (BR) and the policy Approximation Error (AE) can be defined at each iteration

as $\epsilon_k^{BR} = Q_k - T^{\pi_k}Q_k$ and $\epsilon_k^{AE} = Q_k - Q^{\pi_k}$. BRM minimizes the Bellman Error (BE) of the Bellman residuals of Q given the distribution of the input data ν and using a given loss function ℓ defined as

$$L_{BRM_\ell}(Q,\pi) = \int \ell(Q(s,a) - T^\pi Q(s,a))d\nu(s,a)$$

Using the sample data set D_n and the policy process Π_n one tries to evaluate the empirical estimate $\hat{L}_{BRM_\ell}(Q,\Pi_n,D_n) = \mathrm{E}_n[\ell(Q(s_t,a_t) - \hat{T}^\pi Q(s_t,a_t))]$. BRM problem can be solved in the contest of an API method using a Reproducing Kernel Hilbert Space as hypothesis space by taking linear combinations of the form $\hat{Q}(s,a) = \sum_{t=1}^n \beta_t \kappa(s_t,a_t,s,a), \quad \beta_t \in \mathrm{R}$. Accordingly the regularized BRM problem can be written as

$$\hat{Q} = \arg\min_{Q\in\mathcal{H}} \{\hat{L}_{BRM_\ell}(Q,\Pi_n,D_n) + \lambda\|Q\|_{\mathcal{H}}^2\}$$

Standard BRM algorithm (Baird, 1995) uses a least squares loss function to approximate the action value function over the data. Unfortunately this approach brings to a biased estimate and to overcome this problem a common suggestion is to use uncorrelated, or double sampling data sets.

4. Bellman Residuals Minimization with SVR

Hereafter we introduce our method aiming to solve the BRM problem using SVR. Consider the Bellman residuals $BR(s,a) = Q - T^\pi Q = Q_r^\pi(s,a) - r(s,a)$ where the approximating function is

$$Q_r^\pi(s,a) = Q(s,a) - \gamma \int \mathcal{P}(ds^{'}|s,a) \sum_{a^{'}\in A} \pi(a^{'}|s^{'}) Q(s^{'},a^{'})$$
$$= \mathrm{E}[\, Q(s,a) - \gamma \sum_{a^{'}\in A} \pi(a^{'}|s^{'}) Q(s^{'},a^{'})|s,a] = \mathrm{E}[\, \hat{Q}_r^\pi(s,a,s^{'})|s,a]$$

while $r(s,a) = \mathrm{E}[\hat{r}|s,a]$ and the BRM using the $\epsilon-$insensitive loss function $\ell_\epsilon(Q_r^\pi - r) = \max(0, |Q_r^\pi - r| - \epsilon)$ can be written as $L_{BRM_\epsilon}(Q,\pi) = \mathrm{E}[\ell_\epsilon(Q_r^\pi - r)] = \mathrm{E}[\ell_\epsilon(Q - T^\pi Q)]$. Using the data sets Π_n, D_n the empirical estimate becomes: $\hat{L}_{BRM_\epsilon}(Q,\Pi_n,D_n) = \mathrm{E}_n[\ell_\epsilon(\hat{Q}_r^\pi - \hat{r})] = \mathrm{E}_n[\ell_\epsilon(Q - \hat{T}^\pi Q)]$ with $\hat{Q}_r^\pi(s_t,a_t,s_t^{'}) = Q(s_t,a_t) - \gamma \sum_{a_t^{'}\in A} \pi_t(a_t^{'}|s_t^{'}) Q(s_t^{'},a_t^{'})$. Hence the BRM_ϵ optimization problem becomes $\hat{Q} = \arg\min_{Q\in\mathcal{H}}\{\, \hat{L}_{BRM_\epsilon}(Q,\Pi_n,D_n) + \lambda\|Q\|_{\mathcal{H}}^2 \,\}$ where the regularization term uses the norm in the Hilbert space $\mathcal{H}$. BRM_ϵ shows a remarkable sparsity property in the solution which essentially relies on the training support vectors. $\hat{L}_{BRM_\epsilon}(Q,\Pi_n,D_n)$ is an unbiased estimator of $L_{BRM_\epsilon}(Q,\pi)$ as results evaluating the expectation over the empirical losses $\mathrm{E}[\hat{L}_{BRM_\epsilon}(Q,\Pi_n,D_n)|\Pi_n,D_n] = \mathrm{E}[\mathrm{E}_n[\ell_\epsilon(\hat{Q}_r^\pi - \hat{r})]|\Pi_n,D_n] = \mathrm{E}[\mathrm{E}_n[\ell_\epsilon(Q - \hat{T}^\pi Q)]|\Pi_n,D_n] \geq \mathrm{E}[\, \ell_\epsilon(Q^{\pi_r} - r)] = L_{BRM_\epsilon}(Q,\pi)$ where we used the Jensen's inequality $\ell(\mathrm{E}[\, X]) \leq \mathrm{E}[\, \ell(X)]$ holding for any convex function ℓ and $Q_r^\pi(s,a) = \mathrm{E}[\, Q_r^\pi(s,a,s^{'})|s,a] \quad r(s,a) = \mathrm{E}[\hat{r}|s,a] \quad T^\pi Q = \mathrm{E}[\hat{T}^\pi Q]$. In practice the empirical estimate can be biased whenever slacks are presents i.e. the errors on the regression function are above the fixed threshold ϵ. It is unbiased when the error is contained in the resolution tube of the SVR. Nevertheless the choice of the SVR parameters C and ϵ gives a way to control this effect.

5. $API - BRM_\epsilon$ Dual Batch Solution

Consider the subset of observed samples D_n and express the approximation of the value function using a linear architecture as $Q(s,a) = \langle \Phi(s,a), \mathbf{w}\rangle + b$ where $\mathbf{w} = (w_1,...,w_d)^T$

is the weight vector and $\Phi(s,a) = (\phi_1(s,a), ..., \phi_d(s,a))^T$ the features vector of the point (s,a) from which we may build the kernel function $\kappa(s_t, a_t, s, a) = \langle \Phi(s_t, a_t), \Phi(s,a) \rangle$. The action value function belongs the Hilbert space $Q \in \mathcal{H}$ so it does the weight vector $\mathbf{w} \in \mathcal{H}$. Using the Representer theorem we also know that the function in $\mathcal{H}$ can be expressed as linear combination of the elements in the span $\Phi_{span} = \{\Phi(s,a) = \kappa(\cdot, s, a) \quad (s,a) \in S \times A\}$ which can be expressed as $Q(s,a) = \sum_t \alpha_t \kappa(s, a, s_t, a_t)$. Using the definition of the Bellman operator $T^\pi Q$ the Bellman residuals at each training point for a fixed policy π we may write: $BR(s_t, a_t) = Q(s_t, a_t) - T^\pi Q(s_t, a_t) = Q_r^\pi(s_t, a_t) - r(s_t, a_t)$
$= Q(s_t, a_t) - \gamma \int \mathcal{P}(ds^{'}|s_t, a_t) \cdot \sum_{a^{'} \in A} \pi(a^{'}|s^{'}) \cdot Q(s^{'}, a^{'}) - r(s_t, a_t)$ and substituting the functional form of Q yields $BR(s_t, a_t) = \langle \Phi(s_t, a_t), \mathbf{w} \rangle - \gamma \int \mathcal{P}(ds^{'}|s_t, a_t) \cdot \sum_{a^{'} \in A} \pi(a^{'}|s^{'}) \cdot \langle \Phi(s^{'}, a^{'}), \mathbf{w} \rangle + (1-\gamma) b - r(s_t, a_t)$ expressing the Bellman residuals using the weight $\mathbf{w}$ and the features mapping $\Phi(\cdot)$. Here the policy and the MDP dynamic are not included into the Hilbert space $\mathcal{H}$. However an alternative way to express the Bellman residuals is trough the combination of the two terms is a Bellman feature mapping $\Psi^\pi(s_t, a_t) = \Phi(s_t, a_t) - \gamma \int \mathcal{P}(ds^{'}|s_t, a_t) \sum_{a^{'} \in A} \pi(a^{'}|s^{'}) \Phi(s^{'}, a^{'})$ which takes into account the structure of the MDP dynamics. The Bellman residuals are now expressed as $BR(s_t, a_t) = \langle \Psi^\pi(s_t, a_t), \mathbf{w} \rangle + (1-\gamma) b - r(s_t, a_t)$ and with the Bellman feature vector $\Psi^\pi(s_t, a_t)$ we may build the Bellman kernel $\tilde{\kappa}^\pi(s_t, a_t, s, a) = \langle \Psi^\pi(s_t, a_t), \Psi^\pi(s, a) \rangle$. The function Q_r^π and weight vector $\mathbf{w}$ belong to the Bellman Hilbert space $\mathcal{H}_{\Psi^\pi}$. Using the Representer Theorem the function in $\mathcal{H}_{\Psi^\pi}$ can be expressed as linear combination of the elements in the span $\Psi_{span}^\pi = \{\Psi^\pi(s,a) = \tilde{\kappa}^\pi(\cdot, s, a) \quad (s,a) \in S \times A\}$ which can be expressed as $Q_r^\pi(s,a) = \sum_t \beta_t \tilde{\kappa}^\pi(s, a, s_t, a_t)$. With this choice the policy as well as the MDP dynamic are directly incorporated into the Hilbert space $\mathcal{H}_{\Psi^\pi}$.

If we consider the empirical Bellman operator $\hat{T}^\pi Q$ the Bellman residuals can be written as $\hat{BR}(s_t, a_t, s_t^{'}) = Q(s_t, a_t) - \hat{T}^\pi Q(s_t, a_t) = \hat{Q}_r^\pi(s_t, a_t, s_t^{'}) - \hat{r}_t$
$= Q(s_t, a_t) - \gamma \sum_{a_t^{'} \in A} \pi(a_t^{'}|s_t^{'}) \cdot Q(s_t^{'}, a_t^{'}) - \hat{r}_t$ and substituting the functional form of Q yields $hat BR(s_t, a_t, s_t^{'}) = \langle \Phi(s_t, a_t), \mathbf{w} \rangle - \gamma \sum_{a_t^{'} \in A} \pi(a_t^{'}|s_t^{'}) \cdot \langle \Phi(s_t^{'}, a_t^{'}), \mathbf{w} \rangle + (1-\gamma) b - \hat{r}_t$ expressing the Bellman residuals using the weight $\mathbf{w}$ and the features mapping $\Phi(\cdot)$. The empirical Bellman features mapping are $\hat{\Psi}^\pi(s_t, a_t, s_t^{'}) = \Phi(s_t, a_t) - \gamma \sum_{a_t^{'} \in A} \pi(a_t^{'}|s_t^{'}) \Phi(s_t^{'}, a_t^{'})$ and the empirical Bellman residuals $\hat{BR}(s_t, a_t, s_t^{'}) = \langle \hat{\Psi}^\pi(s_t, a_t, s_t^{'}), \mathbf{w} \rangle + (1-\gamma) b - \hat{r}_t$ with the empirical Bellman feature vector $\hat{\Psi}^\pi(s_t, a_t, s_t^{'})$ we may build the Bellman kernel $\tilde{\kappa}^\pi(s_t, a_t, s_t^{'}, s, a, s^{'}) = \langle \hat{\Psi}^\pi(s_t, a_t, s_t^{'}), \hat{\Psi}^\pi(s, a, s^{'}) \rangle$.

Using the Bellman kernel we may find the weighting vector $\mathbf{w}$ using the features mapping $\Psi^\pi(s,a)$ and searching for a solution of the regression function $Q_r^\pi(s,a) = \langle \Psi^\pi(s,a), \mathbf{w} \rangle + b$ with $Q_r^\pi \in \mathcal{H}_{\Psi^\pi}$ solving the SVR problem

$$\begin{aligned} \min_{\mathbf{w}, b, \xi, \xi^*} \quad & \tfrac{1}{2} \|\mathbf{w}\|^2_{\mathcal{H}_{\Psi^\pi}} + C \sum_{t=1}^n (\xi_t + \xi_t^*) \\ s.t. \quad & r(s_t, a_t) - \langle \Psi^\pi(s_t, a_t), \mathbf{w} \rangle - (1-\gamma) b \leq \epsilon + \xi_t \\ & -r(s_t, a_t) + \langle \Psi^\pi(s_t, a_t), \mathbf{w} \rangle + (1-\gamma) b \leq \epsilon + \xi_t^* \quad \xi_t, \xi_t^* \geq 0 \end{aligned} \tag{1}$$

Once the Bellman kernel $\tilde{\kappa}^\pi(s_t, a_t, s, a)$ and the rewards $r(s_t, a_t)$ are provided it can be solved in principle using any standard SVM package.

6. $API - BRM_\epsilon$ Computational Complexity

SVR are powerful tools, but their compute and storage requirements increase rapidly with the number of training vectors. Solving SVR relies on Quadratic Programming (QP) optimization, which is able to separate support vectors from the rest of the train-

ing data. For our SVR implementation test having an optimal solution involves $O(n^2)$ dot products, while solving the QP problem directly involves inverting the kernel matrix, which has complexity $O(n^3)$ where n is the size of the training set. Hence the computational complexity of a K-iteration $API-BRM_\epsilon$ algorithm is dominated by three factors: first computing Gram matrix of the Bellman kernel $\tilde{\kappa}^\pi(z_i, z_j)$ for each pair $z_i = (s_i, a_i)$, $z_j = (s_j, a_j)$, second solving the regularized regression problem and third the number of policy iteration steps used by the algorithm. Computing $\tilde{\kappa}^\pi$ for each pair involves enumerating each successor state of both s_i and s_j and evaluating the base kernel $\kappa(\cdot, \cdot)$ for each pair of successor states. Empirically me may define the average branching factor of a finite state MDP $\hat{\beta}$ as the average number of possible successor states for any state $s \in S$ which in the limit of continuous state space can be assumed $\hat{\beta} = 1$. Equivalently given the dataset D_n and an estimation of the transition kernel $P^{s'}_{s,a}$, the branching factor $\hat{\beta}$ is the average number of terms $P^{s'}_{s,a}$ that are non zero given (s, a). Assuming we have a n samples data set with an MDP that empirically presents an average branching factor $\hat{\beta}$ each state, computing a single element of the Gram matrix requires $O(\hat{\beta}^2)$ operations. Since the Gram matrix has dimension $n \times n$ but is symmetric there are $n(n+1)/2$ unique elements that must be computed. Therefore, the total number of operations to compute the full Gram matrix is $O(kernel) = O(\hat{\beta}^2 n(n+1)/2)$. Once the Gram matrix is constructed in principle its inverse must be evaluated. Since the Gram matrix is positive definite and symmetric, Cholesky decomposition might be used, resulting in a total complexity of $O(n^3)$.

As a result, the total complexity of the BRM_ϵ algorithm is $O(n^3 + \hat{\beta}^2 n(n+1)/2)$ and with K policy iteration steps we assume a complexity $O(K(n^3 + \hat{\beta}^2 n(n+1)/2))$. This is clearly a pessimistic result and thanks to the sparsity property of SVR, assuming an average number of support vectors $n_{sv} \ll n$ one may obtain a posterior complexity $O(K(n_{sv}^3 + \hat{\beta}^2 n_{sv}(n_{sv}+1)/2))$. For the incremental implementation of $API-BRM_\epsilon$ if we consider the worst case, to add a new sample we need $O(n^3) \cdot O(kernel)$ when all the training samples are support vectors (Martin, 2002). On average the algorithm has complexity $O(n^2)$ and for the incremental $API-BRM_\epsilon$ we may assume $O(K(n^2 + \hat{\beta}^2 n(n+1)/2)))$ as complexity bound with an optimist posterior bound $O(K(\hat{n}_{sv}^2 + \hat{\beta}^2 \hat{n}_{sv}(\hat{n}_{sv}+1)/2)))$ with $\hat{n}_{sv}$ the average number of support vectors in K policy iteration.

7. $API - BRM_\epsilon$ Algorithm Implementation

Speed of learning depends mostly on the number of support vectors, influencing significatively the performance. This is the first reason to implement an incremental version of the $API-BRM_\epsilon$ algorithm. Another reason comes from the fact that this version of the algorithm can be easily implemented in an online setting whenever the agent may interact with the environment. After each incremental step (or at least after some of them) which allows to implement a policy evaluation updating the approximation of the value function, one might perform the policy improvement updating the policy. In fact, differently from the offline case where only the final performance matters, in online learning the performance should improve once every few transition samples. Policy iteration can take this requirement into account by performing policy improvements once every few transition samples, before an accurate evaluation of the current policy can be completed. In the practical implementation of online $API - BRM_\epsilon$, by using the current behavior policy the algorithm collects its own samples interacting with the system. As a consequence some exploration has to be added to the policy which becomes soft. As already mention an $\epsilon-$greedy policy is a soft policy which for some $0 \leq \epsilon \leq 1$ picks deterministically a particular action with probability $1 - \epsilon$ and a uniformly random action with probability

Algorithm 1 Online $API - BRM_\epsilon$ with $\epsilon-$greedy exploration

Require: $(\kappa, \lambda, \gamma, K_P, \epsilon_k\ function)$
  $l \leftarrow 0$ initialize $\hat{Q}_0(s,a)$ $(\hat{\pi}_0)$
  solve initial SVR:
  $Q_1 \leftarrow API - BRM_\epsilon(\Pi_0, D_0, \kappa, \lambda)$ (policy evaluation)
  store initial next state policy Π_0
  measure initial state s_0
  for all time step $k > 0$ **do**
    update exploration factor ϵ_k
    choose action: $a_k = \{\pi_k(\cdot)\ w.p.\ 1 - \epsilon_k\ \vee$ random action $w.p.\ \epsilon_k\}$
    apply a_k and measure next state s_{k+1} and reward r_{k+1}
    update training sample set:
    $D_k \leftarrow D_{k-1} \cup (s_k, a_k, r_{k+1}, s_{k+1})$
    update next state policy $\Pi_k \leftarrow \Pi_{k-1} \cup \pi_k(\cdot, s_{k+1})$
    solve incremental SVR:
    $\hat{Q}_k \leftarrow Incremental API - BRM_\epsilon(\Pi_k, D_k, \kappa, \lambda)$
    if $k = (l+1)K_P$ **then**
      update policy $\pi_l(\cdot) \leftarrow \hat{\pi}(\cdot, \hat{Q}_{k-1})$
    end if
    $l \leftarrow l + 1$
  end for
  return

ϵ. Hence policy improvements have to be implemented without waiting for the action value function estimates to get close to their asymptotic values for the current policy. Henceforth these estimates have to be updated continuously without reset after some policy changes as this in practice corresponds to having multiple policies. In principle one may assume that value function estimates remain similar for subsequent policies or al least do not change too much. Another possibility would be to rebuild the action value function estimates from the scratch before every update (some sort of purge for the SVR). Unfortunately this alternative can be computationally costly and might not be necessary in practice. The number of transitions between consecutive policy improvements is a crucial parameter of the algorithm and should not be too large, to avoid potentially bad policies from being used too long. Hereafter we make the assumption that the policy π induces a stationary $\beta-$mixing process on the MDP with a stationary distribution ν. The $\beta-$mixing process starts from an arbitrary initial distribution and following the current policy, the state probability distribution rapidly tends to the stationary distribution of the Markov chain. To guarantee the convergence of the online version of $API - BRM_\epsilon$ we additionally require that the samples follow the stationary distribution over state-action pairs induced by the policy considered ρ. Intuitively this means that the weight of each state-action pair (s,a) is equal to the steady-state probability of this pair along an infinitely-long trajectory generated with the policy π. Moreover assuming that the distribution ρ is stationary the resulting Bellman equation has a unique solution as the Bellman operator is a contraction and it admits one unique fixed point. Consider now the incremental online $API - BRM_\epsilon$ algorithm, the performance guarantees rely on small policy evaluation errors assuming that the policy is improved before an accurate value function is available. In practice this means that the policy evaluation error can be very large and this might affects the performance. Nevertheless, the algorithm works fine in practice for several standard RL benchmarks.

8. $API - BRM_\epsilon$ Experiments

$API - BRM_\epsilon$ algorithm was implemented using a combination of Matlab and C routines and was tested on the following standard RL benchmarks: inverted pendulum, bicycle balancing and riding. The simulation is implemented using a generative model capable of simulating the environment while the learning agent is represented by the $API - BRM_\epsilon$ algorithm. The domains are standard benchmark in RL literature featuring continuous state spaces and non-linear dynamics. Moreover in our experiments, we compare performance of $API - BRM_\epsilon$ algorithm with other learning methods such as Q-learning or the parametric linear approximation architecture of LSPI algorithm implementations for offline (Lagoudakis and Parr, 2003) and online (Busoniu, Lazaric, Ghavamzadeh, Munos, Babuka, and Schutter, 2012). To rank performance it is necessary to introduce some metrics measuring the quality of the solutions. In each benchmark a specific goal is foreseen and a performance measure is represented by the fulfillment of a given task. For example for the inverted pendulum, and the bicycle domains we may assess the quality of a policy through its ability to avoid crashing during a certain period of time. To measure the quality of a solution we can use the stationary policy it produces, compute the expected return and say that the higher this expected return is, the better the RL algorithm performs. This can be done defining a set of initial states S_0 where we compute the average expected return of the stationary policy chosen independently from the set of tuples D_n. Such kind of metric is called as the score of a policy (see (Daniel, Pierre, and Luis, 2005) for more details). Given the learned policy $\hat{\pi}$ its score is defined by $Score_{\hat{\pi}} = \frac{\sum_{s_0 \in S_0} \hat{R}^{\hat{\pi}}(s_0)}{|S_0|}$ where $\hat{R}^{\hat{\pi}}(s_0)$ is the empirical estimate of $R^{\hat{\pi}}(s) = \mathrm{E}[\sum_{t=0}^{n-1} \gamma^t r(s_t, \hat{\pi}(s_t))|s_0 = s]$ the average return. In order to evaluate the score one has to estimate the average empirical return for every initial state $s_0 \in S_0$ by Monte-Carlo simulations. As we consider all the benchmarks non deterministic the average of the score in more than 10 different simulations. Another important aspect to keep in mind is the rather large flexibility of our method which is based upon the generalization ability of SVR and the use of incrementality. Generalization relies on the statistical properties of the structural risk minimization of SVR and the use of a suitable kernel function. In all our experiments for any pair of $z_i = (s_i, a_i)$ and $z_j = (s_j, a_j)$ we use the RBF kernel $\kappa(z_i, z_j) = e^{-\frac{1}{2}(z_i - z_j)^T \Sigma^2 (z_i - z_j)}$ where Σ is a diagonal matrix specifying the weight for any state-action vector component. Using this kernel also allows to manage possible variant of the problems where the action space may be considered continuous or eventually noisy. Even though we studied the statistical properties using finite action spaces, in practice the algorithm may also works fine using a continuous action space. A part from the matrix Σ we also have to define the SVR parameters (C, ϵ). We performed an grid search to find the appropriate set of parameters (Σ, C, ϵ) looking at the resulting performance of the learning system. In fact, using different set of parameters might help finding near-optimal policies whose performance can be measured using the score or their ability to reach the goal. Finally, another important aspect which may affect the performance of the algorithm is represented by the way we collect data and therefore how we manage the compromise between the need of exploration and exploitation of the learned policy. Thanks to the flexibility of our method, we may run experiments using two different methods:

• **Method-1** (online $API - BRM_\epsilon$): some data are generated offline using a random behavior policy which produces a set D_0 of tuples i.i.d. using eventually a set of different initial states S_0 or a fixed one s_0. (This step can be also avoided setting directly $Q = 0$ in each state). This data set is only used to initialize the algorithm solving the BRM_ϵ and providing an initial approximation of the value function. Hence $API - BRM_\epsilon$ algorithm proceed incrementally adding new experiences and improving the policy any K_P steps (with K_P a tunable parameter) using an $\epsilon-$greedy policy. The exploration partly relies

Parameter	Description	Value	UM
g	gravity constant	9.8	m/s^2
m	pole mass	2.0	Kg
M	cart mass	8.0	Kg
l	pole length	0.5	m
α	1/(m+M)	0.1	Kg^{-1}
dt	simulation step	0.1	s
r	reward	0/-1	
γ	discount factor	0.95	

Table 1. Parameters used in the simulation for the inverted pendulum control problem

on the initial data set D_0 and partly depends on the way we manage the exploration ϵ. Generally one may foresee an exponential decay for the ϵ factor starting from some value $\epsilon_0 \leq 1$ and when no further exploration is required fix a minimum value $\epsilon_\infty = 0.1$. We assume that the process underlying the collected data D_n follows an unknown $\beta-$mixing distribution.

• **Method-2** (online-growth $API - BRM_\epsilon$): another possibility consists to alternate explorative samples using a random behavior policy with exploitative samples every K_e steps (with K_e a tunable parameter) and using the $\epsilon-$greedy policy learned with a small exploration ϵ . This can be considered an online variant of the batch-growth method. We assume that the process underlying the collected data D_n follow an unknown $\beta-$mixing distribution. Algorithm 6 illustrates the online variants of $API-BRM_\epsilon$ using an $\epsilon-$greedy exploration policy. The algorithm allows the definition of two parameters which are not present into the offline version: the number of transition $K_P \in N_0$ between consecutive policy improvements and the exploration schedule K_e. Policy is fully optimistic whenever $K_P = 1$ and the policy is updated after every sample while while is partially optimistic with $1 < K_P \leq K_{max}$ where in experiments we choose $K_{max} = 10$. Extensive study about how this parameters affects the quality of the solution for the online LSPI can be found in (Busoniu, Ernst, De Shutter, and Babuska, 2010) which in principle might apply to our method. The exploration schedule can be controlled by the parameter K_e and the decay factor ϵ_{decay} which should be chosen not too large while a significant amount of exploration is necessary. However several alternatives are possible and we also implemented an online variant of the batch-growth method. In this way we perform learning alternating exploration trials using some exploration factor ϵ_f with exploitation trials using a ϵ_0. In practice this method works well and allows to easily found local optimal solution starting from a fixed initial state s_0 . Nevertheless if we need to find a global optimal valid for any initial state, it becomes necessary to explore trough the set of initial states and Method 1 or Method 2 must be applied while learning becomes slower.

9. The Inverted Pendulum Control Problem

For the inverted pendulum benchmark, the control problem consists in balancing at the upright position a pendulum of unknown length and mass. This can be done by applying a force on the cart where the pendulum cart is attached to (Wang, Tanaka, and Griffin, 1996). Due to its simplicity but still challenging control task, this benchmark is widely used to test the performance of state of the art methods for function approximation in RL. In this version of the problem we only have one degree of freedom which can be obtained by fixing the pole to an axis of rotation. The state space $S \setminus S_T = \{(\theta, \dot{\theta}) \in \mathrm{R}^2\}$ is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the inverted pendulum and a terminal state S_T described later. Three actions are allowed $A = \{-a_m, 0, a_m\}$ where $a_m = 50N$ and some uniform noise in $\sigma_a \in [-10, 10]$ might be added to the chosen action. The transitions are governed by the non-linear dynamics of

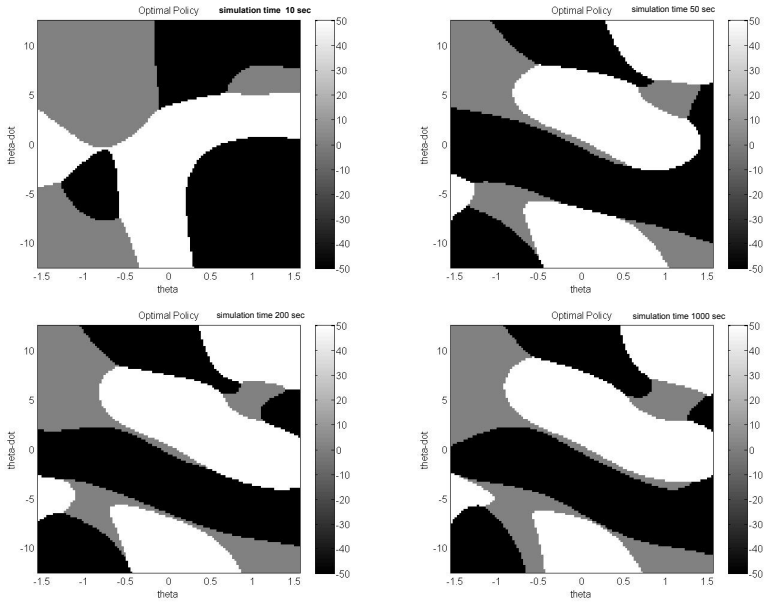

Figure 1. Inverted pendulum: representative subsequences of policy found by online $API-BRM_\epsilon$ using Method-1 (Actions are discretized and only three grey levels show up)

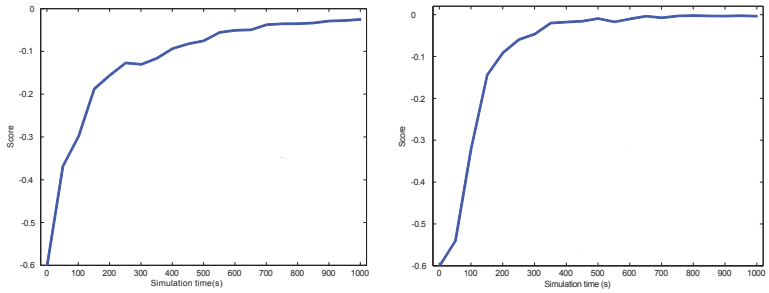

Figure 2. Inverted pendulum: average score for offline LSPI (left) and Q-learning with experience replay (right) from adapted from (Lagoudakis and Parr, 2003)

the system as:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2 \sin(2\theta)/2 - \alpha\cos(\theta)u}{4/3l - m\alpha\cos(\theta)^2} \tag{2}$$

where θ is the angular position, m the mass of the pole, l the length of the pole, M the mass of the cart, $u = a + \sigma_a$ the control action with noise consisting in the acceleration applied to the cart, g the gravity constant. The parameters of the model used in the simulation are reported in Table 1. The angular velocity $\dot{\theta}$ is restricted to $[-4\pi, 4\pi] rads^{-1}$ using saturation. The discrete-time dynamics is obtained by discretizing the time between t and $t+1$ chosen with $dt = 0.1s$. If θ_{t+1} is such that $|\theta_{t+1}| > \theta_m$ a terminal state $S_T = \{|\theta| > \theta_m\}$ is reached where we fixed $\theta_m = \pi/2$. The reward function $r(s_t, a_t)$ is defined as $r(s_t, a_t) = -1$ if $|\theta| > \theta_m$ while is zero otherwise. The discount factor γ has been chosen equal to 0.95. The dynamical system is integrated by using an Euler method with a $0.001s$ integration time step. To generate data samples we may consider episodes starting from the same initial state $s_0 = (\theta_0, \dot{\theta}_0)$ or using a random initial state and stopping when the pole leaves the region represented by $S \setminus S_T$ meaning enter in a terminal states S_T. In (Lagoudakis and Parr, 2003) an analysis of the same benchmark is reported comparing performance with offline LSPI and Q-learning. In this case simulation runs for $1000s$

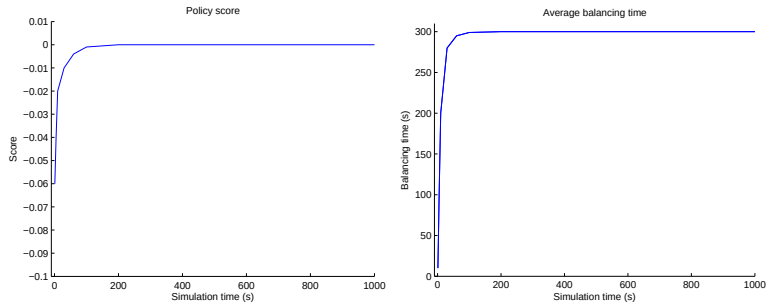

Figure 3. Inverted pendulum: (left) average score of online $API - BRM_\epsilon$ with $K_P = 10$ over a grid of initial states; (right) average balancing time over the same grid of initial states using Method-1

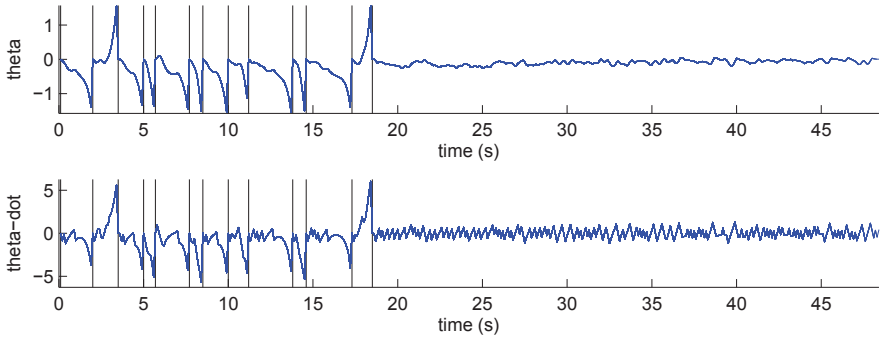

Figure 4. Inverted pendulum: States and actions in representative subsequences of learning trials. Each trial lasts 30s max considered as the minimum balancing to reach. Using Method-2 (online growth) with a fixed initial state $S_0 = (0, 0)$ $API - BRM_\epsilon$ learns a local optimal policy in a few episodes ($20s$ of simulation time).

(10000 samples) separated in 1000 trials of max $1s$ (10 samples) stopping eventually when reaching a terminal state. Offline LSPI uses a linear approximation architecture with a set of 10 Basis Functions (BFs) for each one of the 3 actions, thus a total of 30 basis functions, to approximate the value function. These 10 BFs included a constant term and 9 RBF functions arranged in a 3×3 grid over the $2-$dimensional state space with $BF_i(s) = e^{-\|s-\mu_i\|^2/(2\sigma^2)}$ where μ_i are the 9 points of the grid $\{-\pi/4, 0, +\pi/4\} \times \{-1, 0, +1\}$ and $\sigma = 1$. Training samples were collected starting in a randomly perturbed state very close to the equilibrium state $s_0 = (0, 0)$ and following a policy that selected actions uniformly at random. Results are shown in Figure 2 showing the performance in terms of balancing steps from the analysis detailed in (Lagoudakis and Parr, 2003). Each episode was allowed to run for a maximum of $300s$ (3000 steps) of continuous balancing. A run that balanced for this period of time was considered to be successful. The optimal policy found by LSPI is good enough using the initial state $s_0 = (0, 0)$ while is much worse with other initial states. In our simulation of the same benchmark using online $API - BRM_\epsilon$ we run for $1000s$ of simulated time collecting around 10000 samples. Run was split into separate learning episodes initiated at random initial states and stopping when a terminal state has been reached or otherwise after $30s$ (300 steps). Policy improvement were performed once every $K_P = 10$ steps ($0.1s$) using an $\epsilon-$greedy policy with $\epsilon_0 = 1$ and reaching a value of $\epsilon_\infty = 0.1$ after $350s$.

We also used an RBF kernel with parameters $\Sigma = I_3\sigma$ with $\sigma = 0.5$ and the regression parameters where chosen as $C = 10$ and $\epsilon = 0.01$ selected using an grid search. Figure 1 shows a subsequence of policies found during representative run taken after simulation times $t = 10s, 50s, 200s, 1000s$. Clearly the generalization ability of the SVR makes possible to capture the structure of the approximated policy only after $50s$ of simulation time which closely resembles the final policy obtained after $1000s$ of simulation time.

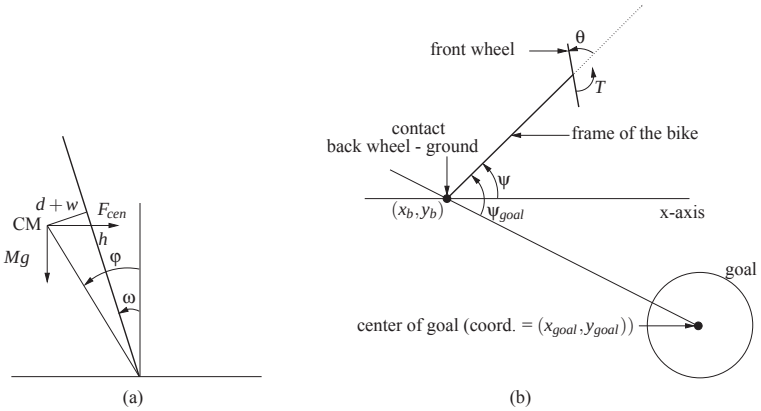

Figure 5. The bike control problem: Figure (a) represents the bicycle seen from behind where the thick line represents the bicycle. The center of mass of the bicycle+cyclist CM with height h from the ground, ω the angle from vertical to bicycle while ϕ represents the total angle of tilt of CM. Action d is agent displacement and w is some noise to simulate imperfect balance. Figure (b) represents the bicycle seen from above. θ is the angle the handlebars are displaced from normal, ψ the angle formed by the bicycle frame and the x axis and ψ_{goal} the angle between the bicycle frame and the line joining the back-wheel ground contact and the center of the goal. T is the torque applied by the cyclist to the handlebars. (x_b, y_b) is the contact point of the back-wheel with the ground (from (Daniel et al., 2005))

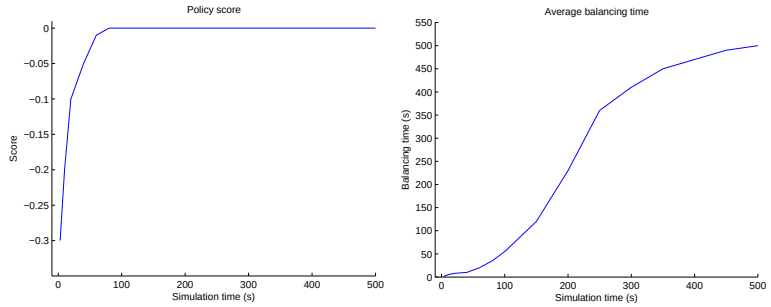

Figure 6. Bike balancing: performance of online $API - BRM_\epsilon$ with $K_P = 10$ using Method-1

Figure 1 shows representative subsequences of policy found by online $API-BRM_\epsilon$ using Method-1. Figure 3 shows the performance of the final policy found by online $API - BRM_\epsilon$ along the online learning process. The performance was measured evaluating the score over a grid of initial states simulating balancing up to $300s$ (3000 steps). Using Q-learning requires state discretization which seriously influences the performance and it cannot estimate the state action value properly until the state is visited which slows the learning. On the contrary the generalization property of SVR our algorithm can estimate state values of unvisited states reasonably using the experience gained with the other states. Moreover being a non parametric regression it can easily adapt to different situation while in general parametric approximation may work well but in a specific contest without the possibility to eventually adapt to changes. In Figure 4 we report states and actions in subsequences of learning trials. Each trial lasts 30s max considered as the minimum balancing to reach. Using Method-2 (online growth) with a fixed initial state $S_0 = (0, 0)$ an optimal local approximation can be found in less then $30s$ of simulation time. Finally the number of support vectors necessary to represents the approximate action value function with the set of parameters used in the approximation usually stays below 5% of the total number of collected samples which is also a indication of the quality of the approximation.

10. The Bike Balancing Control Problem

We consider the control problems related to a bicycle (Randlov and Alstrom, 1998) moving at constant speed on a horizontal plane (Figure 5). For the bicycle balancing the agent has to learn how to balance the bicycle. The system dynamics is composed of seven state variables $S \setminus S_T = \{(\omega, \dot{\omega}, \theta, \dot{\theta}, \psi) \in \mathrm{R}^5 \mid \quad \omega \in [-\omega_m, \omega_m] \quad \theta \in [-\theta_m, \theta_m] \quad \omega_m = \frac{\pi}{15} rad \quad \theta_m = \frac{\pi}{2.25} rad\}$ plus a terminal state S_T. Four states are related to the bicycle itself and three to the position of the bicycle on the plane. The state variables related to the bicycle are $\omega, \dot{\omega}$ (the angle and radial speed from vertical to the bicycle), $\theta, \dot{\theta}$ (the angle and radial speed the handlebars are displaced from normal). If $|\omega| > \omega_m$ the bicycle has fallen down reaching a terminal state S_T. The state variables related to the position of the bicycle on the plane are the coordinates (x_b, y_b) of the contact point of the back tire with the horizontal plane and the angle Ψ formed by the bicycle with the x-axis. The actions space $A = \{(u, T) \in \{-0.02, 0, 0.02\} \times \{-2, 0, 2\}\}$ is composed of 9 elements and depends on the the torque T applied to the handlebars and the displacement d of the rider. The noise in the system is a uniformly distributed term $\sigma_{d_t} = [-0.02, 0.02]$ added to action d. The system has a continuous time dynamics described by the following differential equations described in (Randlov and Alstrom, 1998) Details of the dynamic can be found in (Randlov and Alstrom, 1998) and the various parameter with meanings are also reported in Figure 5, The dynamic holds valid if $|\omega_{t+1}| \leq \omega_m$ while if $|\omega_{t+1}| > \omega_m$ the bicycle is supposed to have fallen down reaching a terminal state S_T. We suppose that the state variables (x_b, y_b) cannot be observed. Since these two state variables do not intervene in the dynamics of the other state variables nor in the reward functions considered. Hence they may be considered no relevant variables which does not make the control problem partially observable. The reward function for the bicycle balancing is such that zero rewards are always observed, except when the bicycle has fallen down and in that case the reward is equal to -1. The value of the discount factor γ has been chosen for both problems equal to 0.98. The dynamical system is integrated by using an Euler method with a $0.001s$ integration time step. To generate data samples we may consider episodes starting from the same initial state corresponding to the bicycle standing and going in straight line with $s_0 = (\omega_0, \dot{\omega}_0, \theta_0, \dot{\theta}_0, \psi_0) = (0, 0, 0, 0, \Psi_0)$ with a fixed value of Ψ or chosen at random $\Psi_0 \in [-\pi, \pi]$ and stopping when the bicycle leaves the region represented by $S \setminus S_T$ meaning a terminal state S_T. In our simulation of this benchmark using online $API - BRM_\epsilon$ we run for $500s$ of simulated time collecting around 50000 samples. Run was split into separate learning episodes initiated at random initial states $s_0 = (0, 0, 0, 0, \Psi_0)$ with $\Psi_0 \in [-\pi, \pi]$ and stopping when a terminal state has been reached or otherwise after $1s$ (100 steps). Policy improvement were performed once every $K_P = 10$ steps $(0.1s)$ using an $\epsilon-$greedy policy with $\epsilon_0 = 1$ and reaching a value of $\epsilon_\infty = 0.1$ after $200s$. We also used an RBF kernel with parameters $\Sigma = I_7\sigma$ with $\sigma = 1.5$ and the regression parameters where chosen as $C = 10$ and $\epsilon = 0.01$ selected using an grid search. Figure 6 shows the performance of the final policy found by online $API - BRM_\epsilon$ along the online learning process. The performance was measured evaluating the score over a grid of initial states $S_0 = \{(0, 0, 0, 0, \Psi_0\}$ with $\Psi_0 \in [-\pi, \pi]$. In Figure 7 we report states and actions in subsequences of learning trials. Each trial lasts 50s max (5000 steps) considered as the minimum balancing to reach the goal. Using Method-2 (online growth) with a fixed initial state $S_0 = (0, 0, 0, 0, \Psi_0)$ an optimal local approximation can be found in less then $50s$ of simulation time. In the lower part of Figure 7 we also show some of the trajectories during the learning process as well as the final one. Finally the number of support vectors necessary to represents the approximate action value function with the set of parameters used in the approximation usually stays below 5% of the total number of collected samples which is also a indication of the quality of the approximation.

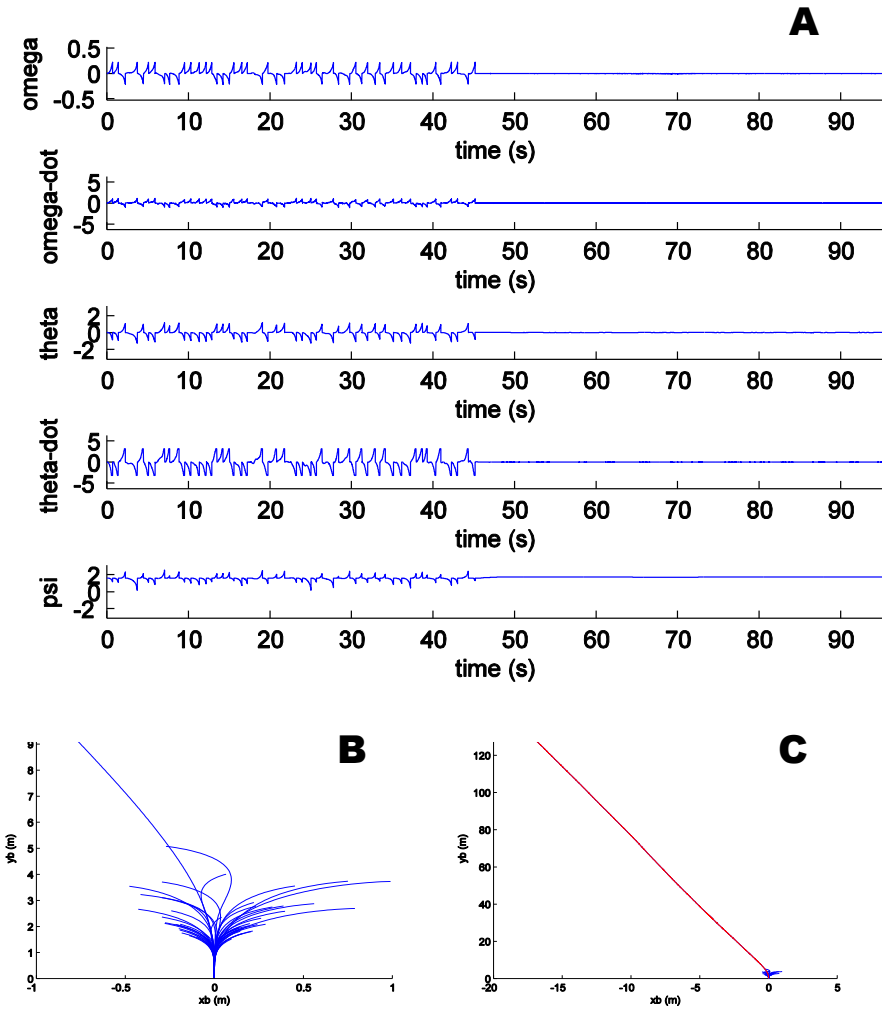

Figure 7. Bike balancing: (Upper A) States and actions in representative subsequences of learning trials. Each trial lasts 50s max (5000 steps) considered sufficient reach the goal. Using Method-2 (online-growth) with small perturbations of a fixed initial state $S_0 = (0, 0, 0, 0, \pi/2)$ $API - BRM_\epsilon$ may learn a local optimal policy in a few episodes ($50s$ of simulation time). (Lower) sketch of the trajectory (B zoom, C overall) in the time interval $(0, 500s)$ for the bicycle on the (x_b, y_b) plane controlled by the final policy of $API - BRM_\epsilon$

11. Conclusions

We developed a model free BRM approach called $API - BRM_\epsilon$ able to find the optimal policy in continuous state RL problems and studied practical implementation issues. In particular, we demonstrated how the problem of finding the optimal policy minimizing the Bellman residuals can be cast as a regression problem using SVR and an appropriate RKHS. The main contribution of this work is the experimental analysis of a non parametric approximation algorithm for the generalization problem in RL using policy iteration and kernel methods. The algorithm eventually converges to the optimal policy using $\beta-$mixing distributed data samples. Some interesting properties of $API - BRM_\epsilon$ algorithm are: $API - BRM_\epsilon$ is quite efficient for problems where sampled experience is sparse. The algorithm is based on approximate policy iteration, a very powerful framework met with success mostly among planning problems. It also open new research directions for the use of kernel based approximate policy iteration in the context of learning. $API - BRM_\epsilon$ is an API algorithm making a good use of function approximation implic-

itly constructing an approximate model using kernels. $API - BRM_\epsilon$ uses incremental SVR which allows for the estimation and approximation of state action value functions in RL. The policy iteration can be done implicitly any time a new experience is obtained. $API - BRM_\epsilon$ complexity strongly depends on the cost one has to pay in order to solve the SVR which is essentially a quadratic problem optimization. SVR can be solved in batch mode when the whole set of training sample are at disposal to the learning agents or in incremental mode enabling the addition or removal of training samples effectively. Finally it comes with a theoretical bound on his performance and statistical convergence guarantee.

Acknowledgment

This work was partially supported by the FI-DGR programme of AGAUR ECO/1551/2012.

References

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.

L. Busoniu, D. Ernst, B. De Shutter, and R. Babuska. Online least-squares policy iteration for reinforcement learning control. In US Baltimore, editor, *In Proceedings of American Control Conference ACC-10*, pages 486–491, 2010.

Lucian Busoniu, Alessandro Lazaric, Mohammad Ghavamzadeh, Remi Munos, Robert Babuka, and Bart Schutter. Least-squares methods for policy iteration. In Marco Wiering and Martijn Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 75–109. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-27644-6. . URL `http://dx.doi.org/10.1007/978-3-642-27645-3_3`.

Ernst Daniel, Geurts Pierre, and Whenkel Luis. Tree based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003. URL `http://dblp.uni-trier.de/db/journals/jmlr/jmlr4.html#LagoudakisP03`.

Mario Martin. On-line support vector machine regression. In *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, pages 282–294, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44036-4. URL `http://dl.acm.org/citation.cfm?id=645329.650050`.

Jette Randlov and Paul Alstrom. Learning to drive a bycicle using reinforcement learning an shaping. In *Proceeding of the fifth International Conference on Machine Learning*, pages 463–471, 1998.

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