From Supervised to Reinforcement Learning: a Kernel-based Bayesian Filtering Framework

Matthieu Geist, Olivier Pietquin, Gabriel Fricout

From Supervised to Reinforcement Learning: a Kernel-based Bayesian Filtering Framework

Matthieu GEIST[1,2,3], Olivier PIETQUIN[1] and Gabriel FRICOUT[2]
[1]IMS Research Group, Supélec, Metz, France
[2]MC Cluster, ArcelorMittal Research, Maizières-lès-Metz, France
[3]CORIDA Project-Team, INRIA Nancy - Grand Est, France
{matthieu.geist,olivier.pietquin}@supelec.fr

***Abstract*—In a large number of applications, engineers have to estimate a function linked to the state of a dynamic system. To do so, a sequence of samples drawn from this unknown function is observed while the system is transiting from state to state and the problem is to generalize these observations to unvisited states. Several solutions can be envisioned among which regressing a family of parameterized functions so as to make it fit at best to the observed samples. This is the first problem addressed with the proposed kernel-based Bayesian filtering approach, which also allows quantifying uncertainty reduction occurring when acquiring more samples. Classical methods cannot handle the case where actual samples are not directly observable but only a non linear mapping of them is available, which happens when a special sensor has to be used or when solving the Bellman equation in order to control the system. However the approach proposed in this paper can be extended to this tricky case. Moreover, an application of this indirect function approximation scheme to reinforcement learning is presented. A set of experiments is also proposed in order to demonstrate the efficiency of this kernel-based Bayesian approach.**

***Index Terms*—supervised learning; reinforcement learning; Bayesian filtering; kernel methods**

I. INTRODUCTION

In a large number of applications, engineers have to estimate values of an unknown function given some observed samples. For example, in order to have a map of wifi (wireless fidelity) coverage in a building, one solution would be to simulate the wave propagation in the building according to Maxwell equations, which would be intractable in practice. An other solution is to measure the electromagnetic field magnitude in some specific locations, and to interpolate between theses observations in order to build a field map which covers the whole building. This task is referred to as function approximation or as generalization. One way to solve the problem is to regress a family of parameterized functions so as to make it fit at best the observed samples. Lots of existing regression methods can be found in the literature for a wide range of function families. Artificial Neural Networks (ANN) [2] or kernel machines [3], [4] are popular methods. Yet, usually batch methods are used (gradient descent for ANN or Support Vector Regression for kernel machines); that is all the observed samples have to be known before regression is done. A new observed sample requires running again the regression algorithm using every sample.

Online regression describes a set of methods able to incrementally improve the regression results as new samples are observed by recursively updating previously computed parameters. There exists online regression algorithms using ANN or kernel machines, yet the uncertainty reduction occurring when acquiring more samples (thus more information) is usually not quantified, as well as with the batch methods. Bayesian methods are such recursive techniques able to quantify uncertainty about the computed parameters. They have already been applied to ANN [5], [6] and, to some extent, to kernel machines [7], [8]. In this paper is proposed a method based on the Bayesian filtering framework [9] for recursively regressing a nonlinear function from noisy samples. In this framework a hidden state (here the regression parameter vector) is recursively estimated from observations (here the samples), while maintaining a probability distribution over parameters (uncertainty estimation).

Several problems are not usually handled by standard techniques. For instance, actual samples are sometimes not directly observable but only a non linear mapping of them is available. This is the case when a special sensor has to be used (*e.g.*, measuring a temperature using a spectrometer or a thermocouple). This is also the case when solving the Bellman equation in a Markovian decision process with unknown deterministic transitions [10]. This is important for (asynchronous and online) dynamic programming, and more generally for control theory. The proposed approach is extended to online regression of nonlinear mapping of observations. First a quite general formulation of the problem is described in order to appeal a broader audience. Indeed the technique introduced below handles well nonlinearities in a derivative free way and it can be useful in other fields. Nevertheless, an application of this framework to reinforcement learning [11] is also described. The general outline of the proposed method is as follows.

The parametric function approximation problem as well as its extension to nonlinear mapped observations case mainly breaks down in two parts. First, a representation for the approximated function must be chosen. For example, it can be an ANN. Notice that this also involves to choose a specific structure, *e.g.*, number of hidden layers, number of neurons, synaptic connections, *etc*. A kernel representation is chosen in this paper, because of its expressiveness given by the Mercer theorem [4]. Moreover a dictionary method [12] allows quasi-

automatizing the choice of the associated structure (that is number and positions of kernel basis). Second, an algorithm to learn the parameters is necessary. For this, the regression problem is cast in a Bayesian tracking problem [9]. As it will be shown, it allows handling well nonlinearities, uncertainty and even non-stationarity.

The next section presents some necessary background: the dictionary method and Bayesian filtering. The following sections describe the proposed algorithm for function approximation [1], its extension to regression from nonlinear mapping of observations [13] and the application of this general algorithm to reinforcement learning [14]. All these algorithms are experimented and compared to the state-of-the-art, and the last section concludes.

II. Background

Before introducing the proposed approach, some backbone methods are presented. The first one is a dictionary method [12] based on mathematical signification of kernels and basic linear algebra which allows automatizing the choice of the structure (number and position of kernels). The second one, Bayesian filtering and more precisely Sigma Point Kalman filtering, is used as the learning part of the proposed algorithms. First, kernel-based regression is briefly introduced.

A. Kernel-based Regression

A kernel-based regression is used, namely the approximation is of the form $\hat{f}_\theta(x) = \sum_{i=1}^{p} \alpha_i K(x, x_i)$ where x belongs to a compact set $\mathcal{X}$ of $\mathbb{R}^n$ (all the work is done in $\mathcal{X}$) and K is a kernel, that is a continuous, symmetric and positive semi-definite function. The parameter vector θ contains the weights $(\alpha_i)_i$, and possibly the centers $(x_i)_i$ and some parameters of the kernel (*e.g.*, the variance for Gaussian kernels). These methods rely on the Mercer theorem [4] which states that each kernel is a dot product in a higher dimensional space. More precisely, for each kernel K, there exists a mapping $\varphi : \mathcal{X} \to \mathcal{F}$ ($\mathcal{F}$ being called the feature space) such that $\forall x, y \in \mathcal{X}$, $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$. Thus, any linear regression algorithm which only uses dot products can be cast by this *kernel trick* into a nonlinear one by implicitly mapping the original space $\mathcal{X}$ to a higher dimensional one. Many approaches to kernel regression can be found in the literature, the most classical being the Support Vector Machines (SVM) framework [4]. There are fewer Bayesian approaches, nonetheless the reader can refer to [7] or [8] for interesting examples. To our knowledge, none of them is designed to handle the second regression problem described in this paper (when observations are nonlinearly mapped).

B. Dictionary

A first problem is to choose the number p of kernel functions required for the regression task and the prior kernel centers. A variety of methods can be contemplated, the simplest one being to choose equally spaced kernel fonctions. However the method described below rests on the mathematical signification of kernels and basic algebra, and is thus well motivated. By observing that although the feature space $\mathcal{F}$ is a (very) higher dimensional space, $\varphi(\mathcal{X})$ can be a quite smaller embedding, the objective is to find a set of p points in $\mathcal{X}$ such that

$$\varphi(\mathcal{X}) \simeq \mathrm{Span}\left\{\varphi(\tilde{x}_1), \ldots, \varphi(\tilde{x}_p)\right\} \tag{1}$$

This method is iterative. Suppose that samples $x_1, x_2, \ldots$ are sequentially generated. At time k, a set

$$\mathcal{D}_{k-1} = (\tilde{x}_j)_{j=1}^{m_{k-1}} \subset (x_j)_{j=1}^{k-1} \tag{2}$$

of m_{k-1} elements is available where by construction feature vectors $\varphi(\tilde{x}_j)$ are approximately linearly independent in $\mathcal{F}$. A sample x_k is then uniformly sampled from $\mathcal{X}$, and is added to the dictionary if $\varphi(x_k)$ is linearly independent on $\mathcal{D}_{k-1}$. To test this, weights $a = (a_1, \ldots, a_{m_{k-1}})^T$ have to be computed so as to verify

$$\delta_k = \min_{a \in \mathbb{R}^{m_{k-1}}} \left\| \sum_{j=1}^{m_{k-1}} a_j \varphi(\tilde{x}_j) - \varphi(x_k) \right\|^2 \tag{3}$$

Formally, if $\delta_k = 0$ then the feature vectors are linearly dependent, otherwise not. Practically an approximate dependence is allowed, and δ_k is compared to a predefined threshold ν determining the quality of the approximation (and consequently the sparsity of the dictionary). Thus the feature vectors will be considered as approximately linearly dependent if $\delta_k \leq \nu$.

By using the kernel trick and the bilinearity of dot products, equation (3) can be rewritten as

$$\delta_k = \min_{a \in \mathbb{R}^{m_{k-1}}} \left\{ a^T \tilde{K}_{k-1} a - 2a^T \tilde{k}_{k-1}(x_t) + K(x_k, x_k) \right\} \tag{4}$$

where

$$\left(\tilde{K}_{k-1}\right)_{i,j} = K(\tilde{x}_i, \tilde{x}_j) \tag{5}$$

is a $m_{k-1} \times m_{k-1}$ matrix and

$$\left(\tilde{k}_{k-1}(x)\right)_i = K(x, \tilde{x}_i) \tag{6}$$

is a $m_{k-1} \times 1$ vector. If $\delta_k > \nu$, $x_k = \tilde{x}_{m_k}$ is added to the dictionary, otherwise not. Equation (4) admits the following analytical solution

$$\begin{cases} a_k = \tilde{K}_{k-1}^{-1} \tilde{k}_{k-1}(x_k) \\ \delta_k = K(x_k, x_k) - \tilde{k}_{k-1}(x_k)^T a_k \end{cases} \tag{7}$$

Notice that the matrix $\tilde{K}_k^{-1}$ can be computed efficiently. If $\delta_k \leq \nu$ no point is added to the dictionary, and thus $\tilde{K}_k^{-1} = \tilde{K}_{k-1}^{-1}$. If x_k is added to the dictionary, one can write the matrix $\tilde{K}_k$ by blocs:

$$\tilde{K}_k = \begin{pmatrix} \tilde{K}_{k-1} & \tilde{k}_{k-1}(x_k) \\ \tilde{k}_{k-1}(x_k)^T & K(x_k, x_k) \end{pmatrix} \tag{8}$$

By using the partitioned matrix inversion formula, its inverse is incrementally computed:

$$\tilde{K}_k^{-1} = \frac{1}{\delta_k} \begin{pmatrix} \delta_k \tilde{K}_{k-1}^{-1} + a_k a_k^T & -a_k \\ -a_k^T & 1 \end{pmatrix} \tag{9}$$

where a_k and δ_k are the given analytical solution to problem (4). This bounds the computational cost for the k^{th} sample by $O(m_k^2)$. Thus this approach allows computing sequentially and incrementally an approximate basis of $\varphi(\mathcal{X})$. The dictionary method is briefly sketched in Algorithm 1, nevertheless see [12] for more details and theoretical analysis of the properties of this approach.

Algorithm 1: Dictionary computation

inputs : a set of N samples randomly selected from $\mathcal{X}$, sparsification parameter ν
outputs: a dictionary $\mathcal{D}$

Initialization;
$\mathcal{D}_1 = \{x_1\}$;

Dictionary computation;
for $k = 1, 2, \ldots N$ **do**
  Observe sample x_k;
  Compute approximate dependence:

$$\delta_k = \min_{a \in \mathbb{R}^{m_{k-1}}} \left\| \sum_{j=1}^{m_{k-1}} a_j \varphi(\tilde{x}_j) - \varphi(x_k) \right\|^2$$

  if $\delta_k > \nu$ **then**
    Add x_k to the dictionary: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \{x_k\}$
  else
    Let the dictionary unchanged: $\mathcal{D}_k = \mathcal{D}_{k-1}$

Thus, by choosing a prior on the kernel to be used, and by applying this algorithm to a set of points $(x_1, \ldots, x_N)$ randomly sampled from $\mathcal{X}$, a sparse set of good candidates to the kernel regression problem is obtained. This method is theorically well founded, easy to implement, computationally efficient and it does not depend on kernels nor space topology. Notice that, despite the fact that this algorithm is naturally online, this dictionary cannot be built (straightforwardly) while estimating the parameters, since the hyper-parameters of the chosen kernels (such as mean and deviation for Gaussian kernels) will be parameterized as well (which leads to a nonlinear parameterization). If the samples used for regression are known beforehand, they can be used to construct the dictionary. However, for online regression, samples are generally not known beforehand. Knowing bounds on $\mathcal{X}$ is sufficient to compute a dictionary.

C. Bayesian Filtering

Bayesian filtering was originally designed to track the state of a stochastic dynamic system from observations (*e.g.*, tracking the position of a plane from radar measures). When used as a learner for function approximation, the parameter vector is the hidden state to be tracked. As it will be shown below, this parameter vector is modeled as a random variable, whereas it is generally deterministic. However, this allows handling non-stationary regression problems, and even if stationary, it helps avoiding local minima. Indeed, the process noise (to be defined below) then plays a role quite similar to simulated annealing. A comprehensive survey of Bayesian filtering is given in [9].

1) Paradigm: The problem of Bayesian filtering can be expressed in its state-space formulation; suppose that the dynamic of a system and the associated generated observations are driven by the following equations:

$$\begin{cases} s_{k+1} = f_k(s_k) + v_k \\ y_k = g_k(s_k) + n_k \end{cases} \tag{10}$$

The objective is to sequentially infer the hidden state s_k given the observations $y_1, \ldots, y_k = y_{1:k}$. The state evolution is driven by the possibly nonlinear mapping f_k and the process noise v_k (centered and of variance P_{v_k}). The observation y_k is a function of the state s_k, corrupted by an observation noise n_k (centered and of variance P_{n_k}). To do so, the posterior density (of state over past observations) is recursively updated as new observations arrive by making use of the Bayes rule and of the dynamic state-space model of the system (10).

Such a Bayesian filtering approach can be decomposed in two steps, which crudely consists in predicting the new observation generated by the system given the current approximated model, and then correcting this model according to the accuracy of this prediction, given the new observation. The first stage is the prediction step. It consists in computing the following distribution:

$$p(S_k|Y_{1:k-1}) = \int_{\mathcal{S}} p(S_k|S_{k-1}) p(S_{k-1}|Y_{1:k-1}) dS_{k-1} \tag{11}$$

It is the prior distribution of current state conditioned on past observations up to time $k-1$. It is a projection forward in time of the posterior at time $k-1$, $p(S_{k-1}|Y_{1:k-1})$ by using the process model represented by $p(S_k|S_{k-1})$ which depends on f_k. For example, if the evolution function is linear and the noise is Gaussian, the distribution of $S_k|S_{k-1}$ is Gaussian of mean $f_{k-1}(S_{k-1})$ and of variance $P_{v_{k-1}}$:

$$S_k|S_{k-1} \sim \mathcal{N}\left(f_{k-1}(S_{k-1}), P_{v_{k-1}}\right) \tag{12}$$

Second, the noisy measurement is incorporated using the observation likelihood and is combined with the prior to update the posterior. This is the correction step:

$$p(S_k|Y_{1:k}) = \frac{p(Y_k|S_k) p(S_k|Y_{1:k-1})}{\int_{\mathcal{S}} p(Y_k|S_k) p(S_k|Y_{1:k-1}) dS_k} \tag{13}$$

The likelihood $Y_k|S_k$ is linked to the observation function g_k. For exemple, if this function is linear and if the noise is Gaussian, the likelihood is also Gaussian:

$$Y_k|S_k \sim \mathcal{N}\left(g_k(S_k), P_{n_k}\right) \tag{14}$$

If the mappings are linear and if the noises n_k and v_k are Gaussian, prior and posterior distributions are analytically computable and the optimal solution is given by the Kalman filter [15]: quantities of interest are random variables, and inference (that is prediction of these quantities and correction of them given a new observation) is done online by propagating sufficient statistics through linear transformations. If the

mappings are nonlinear (but the noises are still Gaussian), a first solution is to linearize them around the state: it is the principle of the Extended Kalman Filter (EKF), and sufficient statistics are still propagated through linear transformations. Another approach is the Sigma Point Kalman Filter (SPKF) framework [6]. The basic idea is that it is easier to approximate a probability distribution than an arbitrary nonlinear function. It is based on the unscented transform [16], which is now described (the sigma-point designation can be seen as a generalization of the unscented transform).

2) The Unscented Transform: The problem of approximating Bayesian filtering when evolution and observation equations are not linear can be expressed as follows: given first and second order moment of a random variable, compute first and second order moments of a nonlinear mapping of this random variable. The unscented transform addresses this issue by deterministically sampling the distribution using its mean and variance.

Let's abstract from previous sections and their notations. Let X be a random vector, and let Y be a mapping of X. The problem is to compute mean and covariance of Y knowing the mapping and first and second order moments of X. If the mapping is linear, the relation between X and Y can be written as $Y = AX$ where A is a matrix of *ad hoc* dimension. In this case, required mean and covariance can be analytically computed: $E[Y] = AE[X]$ and $E[YY^T] = AE[XX^T]A^T$.

If the mapping is nonlinear, the relation between X and Y can be generically written as:

$$Y = f(X) \tag{15}$$

A first solution would be to approximate the nonlinear mapping, that is to linearize it around the mean of the random vector X. This leads to the following approximations of the mean and covariance of Y:

$$E[Y] \approx f\left(E[X]\right) \tag{16}$$

$$E[YY^T] \approx \left(\nabla f\left(E[X]\right)\right) E[XX^T] \left(\nabla f\left(E[X]\right)\right)^T \tag{17}$$

This approach is the basis of Extended Kalman Filtering (EKF) [17], which has been extensively studied and used in past decades. However it has some limitations. First it cannot handle non-derivable nonlinearities. It requires to compute the gradient of the mapping f, which can be quite difficult even if possible. It also supposes that the nonlinear mapping is locally linearizable, which is unfortunately not always the case and can lead to quite bad approximations, as exemplified in [16].

The basic idea of the unscented transform is that it is easier to approximate an arbitrary random vector than an arbitrary nonlinear function. Its principle is to sample *deterministically* a set of so-called sigma-points from the expectation and the covariance of X. The images of these points through the nonlinear mapping f are then computed, and they are used to approximate statistics of interest. It shares similarities with Monte-Carlo methods, however here the sampling is deterministic and requires less samples to be drawn, nonetheless allowing a given accuracy [16].

The original unscented transform is now described more formally (some variants have been introduced since, but the basic principle is the same). Let n be the dimension of X. A set of $2n+1$ sigma-points is computed as follows:

$$x_0 = \bar{X} \qquad j = 0 \tag{18}$$

$$x_j = \bar{X} + \left(\sqrt{(n+\kappa)P_X}\right)_j \qquad 1 \leq j \leq n \tag{19}$$

$$x_j = \bar{X} - \left(\sqrt{(n+\kappa)P_X}\right)_{n-j} \qquad n+1 \leq j \leq 2n \tag{20}$$

as well as associated weights:

$$w_0 = \frac{\kappa}{n+\kappa} \text{ and } w_j = \frac{1}{2\left(n+\kappa\right)} \forall j > 0 \tag{21}$$

where $\bar{X}$ is the mean of X, P_X is its variance matrix, κ is a scaling factor which controls the accuracy of the unscented transform [16], and $(\sqrt{(n+\kappa)P_X})_j$ is the j^{th} column of the Cholesky decomposition of the matrix $(n+\kappa)P_X$. Then the image through the mapping f is computed for each of these sigma-points:

$$y_j = f(x_j), \quad 0 \leq j \leq 2n \tag{22}$$

The set of sigma-points and their images can finally be used to compute first and second order moments of Y, and even P_{XY}, the covariance matrix between X and Y:

$$\bar{Y} \approx \bar{y} = \sum_{j=0}^{2n} w_j y_j \tag{23}$$

$$P_Y \approx \sum_{j=0}^{2n} w_j \left(y_j - \bar{y}\right)\left(y_j - \bar{y}\right)^T \tag{24}$$

$$P_{XY} \approx \sum_{j=0}^{2n} w_j \left(x_j - \bar{x}\right)\left(y_j - \bar{y}\right)^T \tag{25}$$

where $\bar{x} = x_0 = \bar{X}$.

3) Sigma Point Kalman Filtering: The unscented transform having been presented, the Sigma-Point Kalman Filtering, which is an approximation of Bayesian filtering for nonlinear mapping based on unscented and similar transforms (*e.g.*, central differences transform, see [6] for details), is shortly described.

SPKF and classical Kalman equations are very similar. The major change is how to compute sufficient statistics (directly for Kalman, through sigma points for SPKF). Algorithm 2 sketches a SPKF update based on the state-space formulation (10), and using the standard Kalman notations: $s_{k|k-1}$ denotes a prediction, $s_{k|k}$ an estimate (or correction), $P_{s,y}$ a covariance matrix, $\bar{n}_k$ a mean and k is the discrete time index. The principle of each update is as follow. First, the prediction step consists in predicting the current mean and covariance for the hidden state given previous estimates and using the evolution equation. From this and using the observation equation, the current observation is predicted. This implies to use the unscented transform if the mappings are nonlinear. Then mean and covariance of the hidden state are corrected using the current observation and some system

statistics (the better the prediction, the lesser the correction). Update (or equivalently correction step) is made according to the so-called Kalman gain K_k which depends on statistics computed thanks to the unscented transform and using system dynamics. Notice also that being online this algorithm must be initialized with some priors $\bar{s}_{0|0}$ and $P_{0|0}$.

The reader can refer to [6] for details. More precise algorithms will be given later, when developing specific approaches. Generally speaking, the computational complexity of such an update is $O(|S|^3)$, where $|S|$ is the dimension of the state space. However, it will be shown that a square computational complexity is possible for the approach developed in next sections.

Algorithm 2: SPKF Update

inputs : $\bar{s}_{k-1|k-1}$, $P_{k-1|k-1}$
outputs: $\bar{s}_{k|k}$, $P_{k|k}$

Sigma-points computation;
Compute deterministically sigma-point set $S_{k-1|k-1}$ from $\bar{s}_{k-1|k-1}$ and $P_{k-1|k-1}$;

Prediction Step;
Compute sigma-point set $S_{k|k-1}$ from $f_k(S_{k-1|k-1}, \bar{n}_k)$ and process noise covariance;
Compute $\bar{s}_{k|k-1}$ and $P_{k|k-1}$ from $S_{k|k-1}$;

Correction Step;
Observe y_k;
Compute sigma-point set $Y_{k|k-1} = g_k(S_{k|k-1}, \bar{v}_k)$;
Compute $\bar{y}_{k|k-1}$, P_{y_k} and P_{sy_k} from $S_{k|k-1}$, $Y_{k|k-1}$ and observation noise covariance;
$K_k = P_{sy_k} P_{y_k}^{-1}$;
$\bar{s}_{k|k} = \bar{s}_{k|k-1} + K_k(y_k - \bar{y}_{k|k-1})$;
$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;

III. Supervised Learning

The approach presented in this section addresses the problem of nonlinear function approximation. The aim here is to approximate a nonlinear function $f(x)$, $x \in \mathcal{X}$, where $\mathcal{X}$ is a compact set of $\mathbb{R}^n$, from noisy samples

$$(x_k, y_k = f(x_k) + n_k)_k \tag{26}$$

where k is the time index and n_k is the observation random noise, by a function $\hat{f}_\theta(x)$ parameterized by the vector θ. The rest of this paper is written for a scalar output, nevertheless Bayesian filtering paradigm allows an easy extension to the vectorial output case.

The hint is to cast this regression problem in a state-space formulation. The parameter vector is considered as the hidden state to be infered. It is thus modeled as a random variable. As a process model, a random walk is chosen, which is generally a good choice if no more information is available. First it allows handling non-stationarity, but it can also help to avoid local minima, as discussed before. The observation equation links the observations (noisy samples from the function of interest) to the parameterized function. Notice that even if the function of interest is not noisy, it does not necessarily exist in the function space spanned by the parameters, so the observation noise is also structural. This gives the following state-space formulation :

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ y_k = \hat{f}_{\theta_k}(x_k) + n_k \end{cases} \tag{27}$$

As announced in Section I, the principle of the propose approach is to choose a (nonlinear) kernel representation of the value function, to use the dictionary method to automate the choice of the structure, and to use a SPKF to track the parameters.

A. Parameterization

The approximation is of the form

$$\hat{f}_\theta(x) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i}(x, x_i) \tag{28}$$

$$\text{where } \theta = [(\alpha_i)_{i=1}^p,\ (x_i)_{i=1}^p,\ (\sigma_i)_{i=1}^p]^T$$

$$\text{and } K_{\sigma_i}(x, x_i) = \exp(-\frac{\|x - x_i\|^2}{2\sigma_i^2})$$

The problem is to find the optimal number of kernels and a good initialisation for the parameters (extension to other types of kernel is quite straightforward). As a preprocessing step, the dictionary method of Section II-B is used. First a prior σ_0 is put on the Gaussian width (or alternatively on *ad hoc* parameters if another kernel is considered, *e.g.*, degree and bias for a polynomial kernel). Then N random points are sampled uniformly from $\mathcal{X}$ and used to compute the dictionary. Thus a set of p points $\mathcal{D} = \{x_1, \ldots, x_p\}$ is obtained such that

$$\varphi_{\sigma_0}(\mathcal{X}) \simeq \text{Span}\,\{\varphi_{\sigma_0}(x_1), \ldots, \varphi_{\sigma_0}(x_p)\} \tag{29}$$

where φ_{σ_0} is the mapping corresponding to the kernel K_{σ_0}. Notice that even if the sparsification procedure is offline, the algorithm (the regression part) is online. Moreover, no training sample is needed for this preprocessing step, but only classical prior which is anyway needed for the Bayesian filter (σ_0), one sparsification parameter ν and bounds for $\mathcal{X}$. These requirements are not too restrictive.

Let q be the number of parameters. Given the chosen parameterization of equation (28), there is p parameters for the weights, p parameters for Gaussian standard deviations and np parameters for Gaussian centers: $q = (2 + n)p$.

B. Prior

As for any Bayesian approach (and more generally any online method) a prior (an initialization) has to be put on the (supposed Gaussian) parameter distribution:

$$\theta_0 \sim \mathcal{N}(\bar{\theta}_0, \Sigma_{\theta_0}) \tag{30}$$

where mean and covariance matrix are defined as

$$\begin{cases} \bar{\theta}_0 = [\alpha_0, \ldots, \alpha_0, \mathcal{D}, \sigma_0, \ldots, \sigma_0]^T \\ \Sigma_{\theta_0} = \text{diag}([\sigma_{\alpha_0}^2, \ldots, \sigma_{\alpha_0}^2, \sigma_{\mu_0}^2, \ldots, \sigma_{\mu_0}^2, \sigma_{\sigma_0}^2, \ldots, \sigma_{\sigma_0}^2]) \end{cases} \tag{31}$$

The operator diag applied to a column vector gives a diagonal matrix. In these expressions, α_0 is the prior mean on kernel weights, $\mathcal{D}$ is the dictionary computed in the preprocessing step, σ_0 is the prior mean on kernel deviation, and $\sigma^2_{\alpha_0}$, $\sigma^2_{\mu_0}$, $\sigma^2_{\sigma_0}$ are respectively the prior variance on kernel weights, centers and deviations. All these parameters (except the dictionary) have to be set up beforehand. Notice that $\bar{\theta}_0 \in \mathbb{R}^q$ and $\Sigma_{\theta_0} \in \mathbb{R}^{q\times q}$. A prior has also to be put on noises: $v_0 \sim \mathcal{N}(0, R_{v_0})$ where $R_{v_0} = \sigma^2_{v_0} I_q$, I_q being the identity matrix, and $n_0 \sim \mathcal{N}(0, R_{n_0})$, where $R_{n_0} = \sigma^2_{n_0}$ is a scalar.

C. Artificial Process Noise

Another issue is to choose the artificial process noise. Formally, since the target function is stationary, there is no process noise. However introducing an artificial process noise can strengthen convergence and robustness properties of the filter. Choosing this noise is still an open research problem. Following [6] two types of artificial process noise are considered (the observation noise is chosen to be constant).

The first one is a Robbins-Monro stochastic approximation scheme for estimating innovation. That is the process noise covariance is set to

$$R_{v_k} = (1-\alpha_{RM})R_{v_{k-1}} + \alpha_{RM} K_k (y_k - \hat{f}_{\bar{\theta}_{k|k}}(x_k))^2 K_k^T \quad (32)$$

Here α_{RM} is a forgetting factor set by the user, and K_k is the Kalman gain obtained during the Bayesian filter update.

The second type of noise provides for an approximate exponentially decaying weighting past data. Its covariance is set to a fraction of the parameters covariance, that is

$$R_{v_k} = (\lambda^{-1} - 1) P_{k|k} \quad (33)$$

where $\lambda \in]0, 1]$ $(1 - \lambda \ll 1)$ is a forgetting factor similar to the one from the recursive least-squares (RLS) algorithm.

D. Tracking Parameters

A Sigma Point Kalman filter update (which updates first and second order moments of the random parameter vector) is applied at each time step, as a new training sample (x_k, y_k) is available. The evolution equation being linear, the update algorithm does not involve the computation of a sigma-point set in the prediction step as in Algorithm 2. The proposed approach is fully described in Algorithm 3.

First, the dictionary has to be computed and priors have to be chosen. Then, as each input-output couple (x_k, y_k) is observed, the parameter vector mean and covariance are updated. First is the prediction phase. As the parameter vector evolution is modeled as a random walk, the prediction of the mean at time k is equal to the estimate of this mean at time $k-1$ (that is $\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$), and the parameters covariance is updated thanks to the process noise covariance. Then, the set of $2q+1$ sigma-points $\Theta_{k|k-1}$ corresponding to the parameters distribution must be computed using $\bar{\theta}_{k|k-1}$ and $P_{k|k-1}$, as well as associated weights $\mathcal{W}$ (see Section II-C2):

$$\Theta_{k|k-1} = \left\{ \theta^{(j)}_{k|k-1}, \quad 0 \le j \le 2q \right\} \quad (34)$$

$$\mathcal{W} = \{ w_j, \quad 0 \le j \le 2q \quad \} \quad (35)$$

Notice that each of these sigma-points corresponds to a particular parameterized function. Each of these functions is evaluated in x_k, the current observed input, which forms the set of images of the sigma-points:

$$\mathcal{Y}_{k|k-1} = \left\{ y^{(j)}_{k|k-1} = \hat{f}_{\theta^{(j)}_{k|k-1}}(x_k),\ 0 \le j \le 2q \right\} \quad (36)$$

It can be seen as an approximated sampled prior distribution over observations. This sigma-point set and its image are then used to compute the prediction of the observation as well as some statistics necessary to the computation of the Kalman gain:

$$\bar{y}_{k|k-1} = \sum_{j=0}^{2q} w_j y^{(j)}_{k|k-1} \quad (37)$$

$$P_{\theta y_k} = \sum_{j=0}^{2q} w_j (\theta^{(j)}_{k|k-1} - \bar{\theta}_{k|k-1})(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1}) \quad (38)$$

$$P_{y_i} = \sum_{j=0}^{2q} w_j \left(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1} \right)^2 + P_{n_k} \quad (39)$$

Finally, the Kalman gain can be computed, and the estimated mean and covariance can be updated:

$$K_k = P_{\theta y_k} P^{-1}_{y_k} \quad (40)$$

$$\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(y_k - \bar{y}_{k|k-1} \right) \quad (41)$$

$$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T \quad (42)$$

If necessary, the artificial process noise is updated following one of the two proposed schemes. A constant or null process noise can also be considered.

Notice that practically a square-root implementation of this algorithm is used, which principally avoids a full Cholesky decomposition at each time-step. This approach is computationally cheaper: the complexity per iteration is $O(q^2)$, where it is recalled that $q = |\theta|$ is the size of the parameter vector ($O(q^3)$ in the general case). See [6] for details concerning this square-root implementation of SPKF.

E. Experiments

In this section, the aim is to experiment Algorithm 3 by regressing a cardinal sine ($\mathrm{sinc}(x) = \frac{\sin(x)}{x}$) on $\mathcal{X} = [-10, 10]$. It is an easy problem, but a common benchmark too which allows comparison to state-of-the-art algorithms. The uncertainty about function approximation which can be computed from this framework is also illustrated.

1) Problem Statement and Settings: At each time step k samples $x_k \sim \mathcal{U}_\mathcal{X}$ (x_k uniformly sampled from $\mathcal{X}$) and $y_k = \mathrm{sinc}(x_k) + w_k$ where $w_k \sim \mathcal{N}(0, \sigma^2_w)$ are observed. Notice that the true observation covariance noise σ^2_w and the prior σ^2_v are distinguished. For those experiments the first type of artificial process noise presented in Section III-C is used. The true noise is set to $\sigma_w = 0.1$. The algorithm parameters are set to $N = 100$, $\sigma_0 = 1.6$, $\nu = 0.1$, $\alpha_0 = 0$, $\sigma_v = 0.5$, $\alpha_{RM} = 0.7$, and all variances ($\sigma^2_{\alpha_0}$, $\sigma^2_{\mu_0}$, $\sigma^2_{\sigma_0}$, $\sigma^2_{n_0}$) to 0.1. Notice that this parameters were not finely tuned (only orders of magnitude seem to be important).

Algorithm 3: Direct regression algorithm

Compute dictionary;
$\forall i \in \{1 \dots N\},\ x_i \sim \mathcal{U}_{\mathcal{X}}$;
Set $X = \{x_1, \dots, x_N\}$;
$\mathcal{D}$ =`Compute-Dictionary` (X,ν,σ_0) ;

Initialisation;
Initialise $\bar{\theta}_0$, $P_{0|0}$, R_{n_0}, R_{v_0};

for $k = 1, 2, \dots$ **do**
  Observe $(\mathbf{x_k}, \mathbf{y_k})$;
  SPKF update;
  Prediction Step;
  $\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$;
  $P_{k|k-1} = P_{k-1|k-1} + P_{v_{k-1}}$;
  Sigma-points computation ;
  $\Theta_{k|k-1} = \left\{ \theta^{(j)}_{k|k-1}, \quad 0 \leq j \leq 2q \right\}$;
  $\mathcal{W} = \{ w_j, \quad 0 \leq j \leq 2q \quad \}$;
  $\mathcal{Y}_{\mathbf{k|k-1}} = \left\{ \mathbf{y}^{(\mathbf{j})}_{\mathbf{k|k-1}} = \hat{\mathbf{f}}_{\theta^{(\mathbf{j})}_{\mathbf{k|k-1}}}(\mathbf{x_k}),\ \mathbf{0 \leq j \leq 2q} \right\}$;
  Compute statistics of interest;
  $\bar{y}_{k|k-1} = \sum_{j=0}^{2q} w_j y^{(j)}_{k|k-1}$;
  $P_{\theta y_k} = \sum_{j=0}^{2q} w_j (\theta^{(j)}_{k|k-1} - \bar{\theta}_{k|k-1})(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1})$;
  $P_{y_i} = \sum_{j=0}^{2q} w_j \left(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1} \right)^2 + P_{n_k}$;
  Correction step;
  $K_k = P_{\theta y_k} P^{-1}_{y_k}$;
  $\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(y_k - \bar{y}_{k|k-1} \right)$;
  $P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;
  Artificial process noise update;
  Robbins-Monro covariance;
  $R_{v_k} =$
  $(1 - \alpha_{RM}) R_{v_{k-1}} + \alpha_{RM} K_k (y_k - \hat{f}_{\bar{\theta}_{k|k}}(x_k))^2 K_k^T$;
  or;
  Forgetting covariance;
  $R_{v_k} = (\lambda^{-1} - 1) P_{k|k}$;

2) Quality of Regression: The quality of regression is measured with the RMSE (Root Mean Square Error) $\|f - \hat{f}_\theta\|$ (where $\|.\|$ is the usual L_2-norm), computed over 200 equally-spaced points. Averaged over 100 runs of Algorithm 3, a RMSE of 0.0676 ± 0.0176 is obtained for 50 samples, of 0.0452 ± 0.0092 for 100 samples, and of 0.0397 ± 0.0065 for 200 samples, for an average of 9.6 kernels. This is illustrated on Figure 1. A typical result is given in Figure 2 (50 observed samples). The dotted line, solid line and the crosses represent respectively the cardinale sine, the regression and the observations.

The proposed algorithm compares favorably with state-of-the-art batch and online algorithms. Table I shows the performance of the proposed algorithm and of other methods (some being online and other batch, some handling uncertainty information and other not), for which results are reproduced

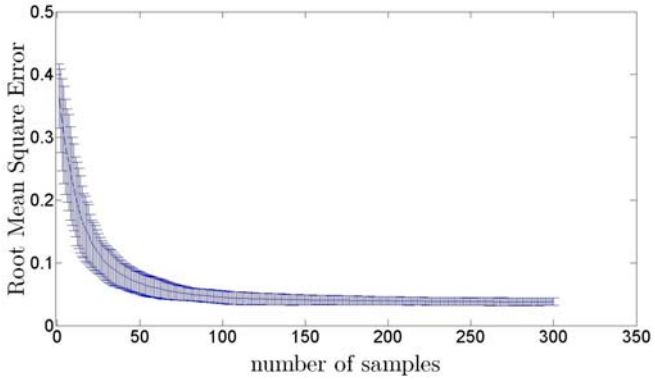


Fig. 1. RMSE (mean ± deviation).

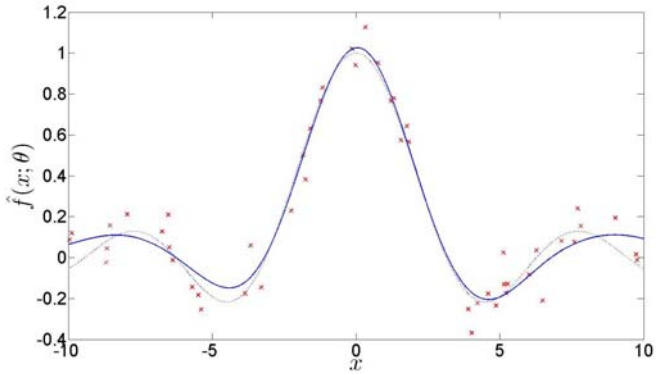


Fig. 2. Typical regression.

from [8] (see this paper and references therein for details about these algorithms). For each method is given the RMSE as well as the number of kernel functions, and the associated variations, the SVM (Support Vector Machine) [4] being the baseline. The proposed method achieves the best RMSE with slightly more kernels than other approaches. However the sparsification parameter ν of the dictionary can be tuned in order to address the trade-off between number of kernels and quality of approximation. Moreover, recall that the parameters were not finely tuned.

3) Uncertainty of Generalization: Through the sigma point approach, it is also possible to derive a confidence interval over $\mathcal{X}$. This allows to quantify the uncertainty of the regression at any point (and not a global upper bound as it is often computed

Method	test error	# kernels
Proposed algorithm	**0.0385 (-25.8%)**	**9.6 (-65.7%)**
Figueiredo	0.0455 (-12.3%)	7.0 (-75%)
SVM	**0.0519 (-0.0%)**	**28.0 (-0%)**
RVM	0.0494 (-4.8%)	6.9 (-75.3%)
VRVM	0.0494 (-4.8%)	7.4 (-73.5%)
MCMC	0.0468 (-9.83%)	6.5 (-76.8%)
Sequential RVM	0.0591 (+13.8%)	4.5 (-83.9%)

TABLE I
COMPARATIVE RESULTS.

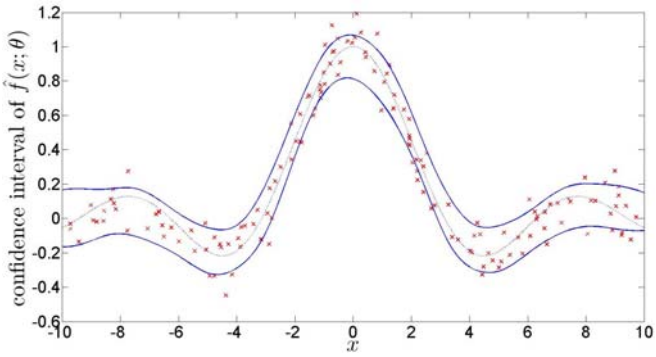


Fig. 3. Confidence (uniform distribution).

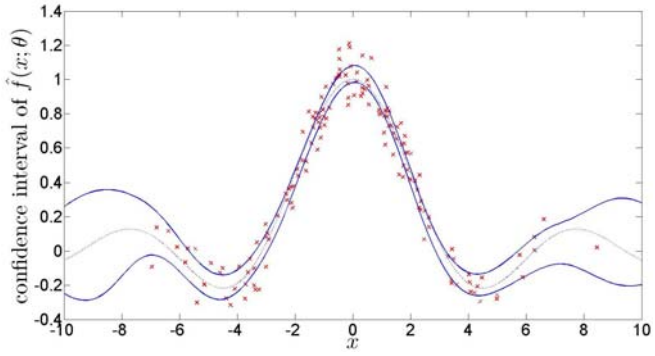


Fig. 4. Confidence (normal distribution).

in kernel-based regression methods). A typical confidence interval is illustrated on Figure 3. The dotted line, solid line and the crosses represent respectively the cardinale sine, the confidence interval (indeed the standard deviation, which corresponds to a confidence interval if Gaussian noise is assumed) and the observations. It can be particularly useful when the regression is used in a control framework, where this confidence approach can be used to take more cautious decisions (see [18] for example). It can be also useful for active learning, which aims at selecting costly samples such as gaining the more possible new information.

In Figure 3, the samples used to feed the regressor are sampled uniformly from $\mathcal{X}$, and thus the associated confidence interval has an approximately constant width. However, a regressor which handles uncertainty should do it locally. This is indeed the case for the proposed algorithm. In Figure 4, the distribution of samples is Gaussian zero-centered. It can be seen that the confidence interval is much larger where samples are less frequent (close to the bounds). So the computed confidence intervals make sens.

IV. Extension to the Case of Nonlinearly Mapped Observations

The problem addressed here is slightly different from the one in Section III. The function of interest is not directly observed, but only a nonlinear (and possibly non-derivable) mapping of it is available, the mapping being known analytically. Moreover, some of the involved nonlinearities can be just observed. This is the case when a special sensor has to be used (*e.g.*, measuring a temperature using a spectrometer or a thermocouple). This is also the case when solving the Bellman equation in a Markovian decision process with unknown deterministic transitions, which will be developped in Section V.

More formally, let $x = (x^1, x^2) \in \mathcal{X} = \mathcal{X}^1 \times \mathcal{X}^2$ where $\mathcal{X}^1$ (resp. $\mathcal{X}^2$) is a compact set of $\mathbb{R}^n$ (resp. $\mathbb{R}^m$). Let $t : \mathcal{X}^1 \times \mathcal{X}^2 \to \mathcal{X}^1$ be a nonlinear transformation (transitions in case of dynamic systems) which will be observed. Let g be a nonlinear mapping such that $g : f \in \mathbb{R}^{\mathcal{X}} \to g_f \in \mathbb{R}^{\mathcal{X}\times\mathcal{X}^1}$. The aim here is to approximate sequentially the nonlinear function $f : x \in \mathcal{X} \to f(x) = f(x^1, x^2) \in \mathbb{R}$ from samples

$$\left(x_k, t_k = t(x_k^1, x_k^2), y_k = g_f(x_k^1, x_k^2, t_k)\right)_k \tag{43}$$

by a function $\hat{f}_\theta(x) = \hat{f}_\theta(x^1, x^2)$ parameterized by the vector θ. Here the output is scalar, however the proposed method can be straightforwardly extended to vectorial outputs. Notice that as

$$\left(\mathbb{R}^{\mathcal{X}}\right)^{\mathbb{R}^{\mathcal{X}}} \subset \left\{ g \in \left(\mathbb{R}^{\mathcal{X}\times\mathcal{X}^1}\right)^{\mathbb{R}^{\mathcal{X}}} | g_f : x \in \mathcal{X} \to g_f(x, t(x)) \right\} \tag{44}$$

this problem statement is quite general. Thus the work presented in this section can be considered with $g : f \in \mathbb{R}^{\mathcal{X}} \to g_f \in \mathbb{R}^{\mathcal{X}}$ (g being known analytically), this case being more specific. The interest of this particular formulation is that a part of the nonlinearities has to be known analytically (the mapping g) whereas the other part can be just observed (the t function).

As before, this regression problem is stated in the corresponding state-space formulation:

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ y_k = g_{\hat{f}_{\theta_k}}(x_k^1, x_k^2, t_k) + n_k \end{cases} \tag{45}$$

Recall that the algorithm proposed in Section III is derivative free and handle nonlinearities. Thus it is quite simple to adapt it to this new state-space formulation. A kernel-based representation is chosen as before, and its structure is automatized thanks to the dictionary method. A prior is put, and then a SPKF is used to learn the parameters.

A. Parameterization

Recall that the objective here is to sequentially approximate a nonlinear function, as samples $(x_k, t_k = t(x_k^1, x_k^2), y_k)$ are available, with a weighted sum of kernel functions. This parameter estimation problem can be expressed as the state-space problem (45). Notice that here f does not depend on time, however this approach can be easily extended to nonstationary function approximation. In this section, the approximation is

of the form

$$\hat{f}_\theta(x) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i^1}(x^1, x_i^1) K_{\sigma_i^2}(x^2, x_i^2), \text{ with} \quad (46)$$

$$\theta = [(\alpha_i)_{i=1}^p, \left((x_i^1)^T\right)_{i=1}^p, (\sigma_i^1)_{i=1}^p, \left((x_i^2)^T\right)_{i=1}^p, (\sigma_i^2)_{i=1}^p]^T$$

$$\text{and } K_{\sigma_i^j}(x^j, x_i^j) = \exp(-\frac{\|x^j - x_i^j\|^2}{2(\sigma_i^j)^2}),\ j = 1, 2$$

Notice that $K_{\sigma_i}(x, x_i) = K_{(\sigma_i^1, \sigma_i^2)}\left((x^1, x^2), (x_i^1, x_i^2)\right) = K_{\sigma_i^1}(x^1, x_i^1) K_{\sigma_i^2}(x^2, x_i^2)$ is a product of kernels, thus it is a kernel. Again, extension to other kernels is straightforward. The optimal number of kernels and a good initialisation for the parameters have to be determined first.

To tackle this initial difficulty, the dictionary method is used in a preprocessing step. A prior $\sigma_0 = (\sigma_0^1, \sigma_0^2)$ on the Gaussian width is first put (the same for each kernel). Then a set of N random points is sampled uniformly from $\mathcal{X}$ and used to compute the dictionary. A set of p points $\mathcal{D} = \{x_1, \ldots, x_p\}$ is thus obtained such that $\varphi(\mathcal{X}) \simeq \text{Span}\{\varphi_{\sigma_0}(x_1), \ldots, \varphi_{\sigma_0}(x_p)\}$ where φ_{σ_0} is the mapping corresponding to the kernel K_{σ_0}. As before, even though this sparsification procedure is offline, the proposed algorithm (the regression part) is online.

B. Prior, Learning and Artificial Noises

A SPKF is then used to estimate the parameters. As for any Bayesian approach an initial prior on the parameter distribution has to be put. The initial parameter vector follows a Gaussian law, that is $\theta_0 \sim \mathcal{N}(\bar{\theta}_0, \Sigma_{\theta_0})$, where

$$\begin{cases} \bar{\theta}_0 = [\alpha_0, \ldots, \alpha_0, \mathcal{D}^1, \sigma_0^1, \ldots, \sigma_0^1, \mathcal{D}^2, \sigma_0^2, \ldots, \sigma_0^2]^T \\ \Sigma_{\theta_0} = \text{diag}(\sigma_{\alpha_0}^2, \ldots, \sigma_{\mu_0^1}^2, \ldots, \sigma_{\sigma_0^1}^2, \ldots, \sigma_{\mu_0^2}^2, \ldots, \sigma_{\sigma_0^2}^2, \ldots) \end{cases} \quad (47)$$

In these equations, α_0 is the prior mean on kernel weights, $\mathcal{D}^j = [(x_1^j)^T, \ldots, (x_p^j)^T]$ is derived from the dictionary computed in the preprocessing step, σ_0^j is the prior mean on kernel deviation, and $\sigma_{\alpha_0}^2$, $\sigma_{\mu_0^j}^2$ (which is a row vector with the variance for each component of x_i^j, $\forall i$) and $\sigma_{\sigma_0}^2$ are respectively the prior variances on kernel weights, centers and deviations. All these parameters (except the dictionary) have to be set up beforehand. Let $|\theta| = q = (n + m + 3)p$. Notice that $\bar{\theta}_0 \in \mathbb{R}^q$ and $\Sigma_{\theta_0} \in \mathbb{R}^{q \times q}$. A prior on noises has to be put too: more precisely, $v_0 \sim \mathcal{N}(0, R_{v_0})$ where $R_{v_0} = \sigma_{v_0}^2 I_q$ and $n \sim \mathcal{N}(0, R_n)$ where $R_n = \sigma_n^2$. Notice that here the observation noise is also structural. In other words, it models the ability of the parameterization $\hat{f}_\theta$ to approximate the true function of interest f. Then, a SPKF update is simply applied at each time step, as a new training sample (x_k, t_k, y_k) is available, as summarized in Algorithm 4. The only difference between this algorithm and Algorithm 3 is what is observed, as well as how to compute the set of images of sigma-points, directly using the approximated function $\hat{f}_\theta$ for the first one, and using its nonlinear mapping $g_{\hat{f}_\theta}$ for the second one.

A last issue is to choose the artificial process noise. The same noises as in Section III-C are considered, namely the Robbins-Monro stochastic approximation scheme for estimating innovation and the approximate exponentially decaying weighting past data. Alternatively a constant or null process noise can be considered. The observation noise is chosen constant.

Algorithm 4: Indirect regression algorithm

Compute dictionary;
$\forall i \in \{1 \ldots N\},\ x_i \sim \mathcal{U}_\mathcal{X}$;
Set $X = \{x_1, \ldots, x_N\}$;
$\mathcal{D}$ =`Compute-Dictionary` (X,ν,σ_0) ;

Initialisation;
Initialise $\bar{\theta}_0$, $P_{0|0}$, R_{n_0}, R_{v_0};

for $k = 1, 2, \ldots$ **do**

Observe $(\mathbf{x_k}, \mathbf{t_k}, \mathbf{y_k})$;

SPKF update;

Prediction Step;
$\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$;
$P_{k|k-1} = P_{k-1|k-1} + P_{v_{k-1}}$;

Sigma-points computation ;
$\Theta_{k|k-1} = \left\{\theta_{k|k-1}^{(j)}, \quad 0 \le j \le 2q\right\}$;
$\mathcal{W} = \{w_j, \quad 0 \le j \le 2q \quad\}$;
$\mathcal{Y}_{\mathbf{k|k-1}} = \left\{\mathbf{y_{k|k-1}^{(j)}} = \mathbf{g_{\hat{f}_{\theta_{k|k-1}^{(j)}}}}(\mathbf{x_k}, \mathbf{t_k}),\ \mathbf{0 \le j \le 2q}\right\}$;

Compute statistics of interest;
$\bar{y}_{k|k-1} = \sum_{j=0}^{2q} w_j y_{k|k-1}^{(j)}$;
$P_{\theta y_k} = \sum_{j=0}^{2q} w_j (\theta_{k|k-1}^{(j)} - \bar{\theta}_{k|k-1})(y_{k|k-1}^{(j)} - \bar{y}_{k|k-1})$;
$P_{y_i} = \sum_{j=0}^{2q} w_j \left(y_{k|k-1}^{(j)} - \bar{y}_{k|k-1}\right)^2 + P_{n_k}$;

Correction step;
$K_k = P_{\theta y_k} P_{y_k}^{-1}$;
$\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(y_k - \bar{y}_{k|k-1}\right)$;
$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;

Artificial process noise update;
Robbins-Monro covariance;
$R_{v_k} =$
$(1 - \alpha_{RM}) R_{v_{k-1}} + \alpha_{RM} K_k (y_k - g_{\hat{f}_{\bar{\theta}_{k|k}}}(x_k, t_k))^2 K_k^T$;
or;
Forgetting covariance;
$R_{v_k} = (\lambda^{-1} - 1) P_{k|k}$;

C. Experiments

This section aims at illustrating Algorithm 4. The proposed artificial problem is quite arbitrary, it has been chosen for its interesting nonlinearities so as to emphasize on the potential benefits of the proposed approach. Let $\mathcal{X} = [-10, 10]^2$ and f be a 2d nonlinear cardinal sine:

$$f(x^1, x^2) = \frac{\sin(x^1)}{x^1} + \frac{x^1 x^2}{100} \quad (48)$$

Let the observed nonlinear transformation be

$$t(x^1, x^2) = 10 \tanh(\frac{x^1}{7}) + \sin(x^2) \quad (49)$$

Recall that this analytical expression is only used to generate observations in this experiment but it is not used in the regression algorithm. Let the nonlinear mapping g be

$$g_f\left(x^1, x^2, t(x^1, x^2)\right) = f(x^1, x^2) - \gamma \max_{y \in \mathcal{X}^2} f\left(t(x^1, x^2), y\right) \tag{50}$$

with $\gamma \in [0, 1[$ a predefined constant. This is a specific form of the Bellman equation [10] with the transformation t being a deterministic transition function. Recall that this equation is of special interest in optimal control theory. The associated state-space formulation is thus

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ y_k = \hat{f}_{\theta_k}(x_k^1, x_k^2) - \gamma \max_{x^2 \in \mathcal{X}^2} \hat{f}_{\theta_k}\left(t(x_k^1, x_k^2), x^2\right) + n_k \end{cases} \tag{51}$$

1) Problem Statement and Settings: At each time step k the regressor observes $x_k \in \mathcal{U}_\mathcal{X}$ (x_k uniformly sampled from $\mathcal{X}$), the transformation $t_k = t(x_k^1, x_k^2)$ and $y_k = f(x_k) - \gamma \max_{x^2 \in \mathcal{X}^2} f(t_k, x^2)$. Once again the regressor does not have to know the function t analytically, it just has to observe it. The function f is shown on Figure 5(a) and its nonlinear mapping on Figure 5(b). For these experiments the first type of artificial process noise presented in Section III-C is used. The algorithm parameters were set to $N = 1000$, $\sigma_0^j = 4.4$, $\nu = 0.1$, $\alpha_0 = 0$, $\sigma_n = 0.05$, $\alpha_{RM} = 0.7$, and all variances ($\sigma_{\alpha_0}^2$, $\sigma_{\mu_0^j}^2$, $\sigma_{\sigma_0^j}^2$, $\sigma_{n_0}^2$) to 0.1, for $j = 1, 2$. The factor γ was set to 0.9. Notice that this empirical parameters were not finely tuned.

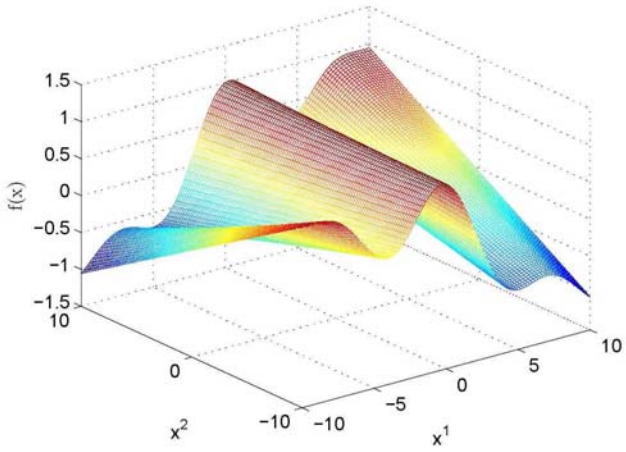
2) Quality of Regression: Because of the special form of g_f, any constant bias added to f still gives the same observations and it may exist other invariances: the root mean square error (RMSE) between f and $\hat{f}$ should not be used to measure the quality of regression. Instead the nonlinear mapping of f is computed from its estimate $\hat{f}$ and is then used to calculate the RMSE. The quality of regression is thus measured with the following RMSE:

$$\sqrt{\int_\mathcal{X} \left(g_f(x, t(x)) - g_{\hat{f}_\theta}(x, t(x))\right)^2 dx} \tag{52}$$

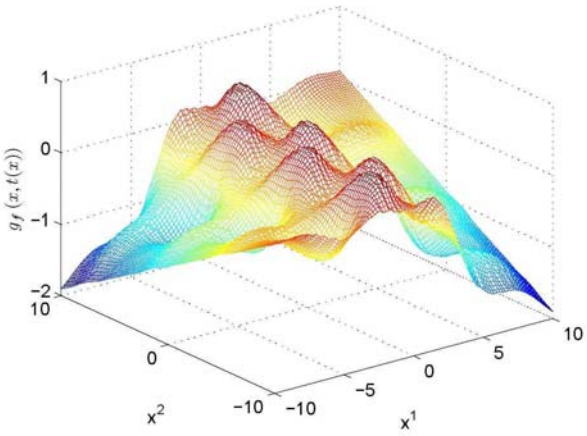
As it is computed from $\hat{f}$, it really measures the quality of regression, and as it uses the associated nonlinear mapping, it will not take into account the bias. Practically it is computed on 10^4 equally spaced points. The function g_f is what is observed (Figure 5(b)) and it is used by the SPKF to approximate f (Figure 5(a)) by $\hat{f}_\theta$ (Figure 6(a)). The nonlinear mapping of the approximated function $g_{\hat{f}_\theta}$ is computed from $\hat{f}_\theta$ (Figure 6(b)) and it is used to compute the RMSE.

As the proposed algorithm is stochastic, the results have been averaged over 100 runs. Figure 7 shows an errorbar plot, that is mean $\pm$ standard deviation of RMSE. The average number of kernels was 26.55 ± 1.17. This results can be compared with the RMSE directly computed from $\hat{f}$, that is

$$\sqrt{\int_{x \in \mathcal{X}} (f(x) - \hat{f}_\theta(x))^2 dx} \tag{53}$$



(a) 2-dimensional nonlinear cardinal sine.



(b) Nonlinear mapping observed by the regressor.

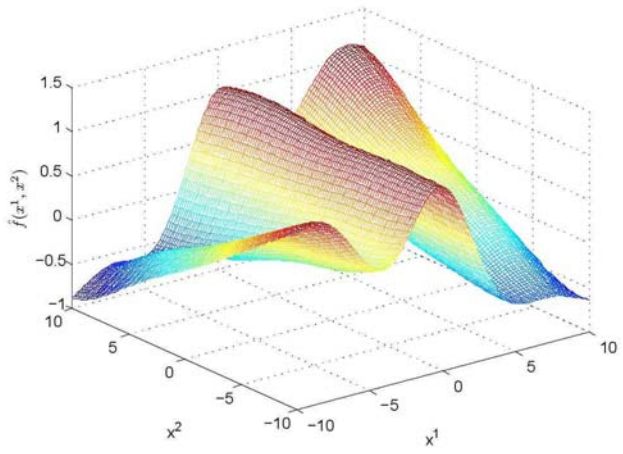
Fig. 5. Original functions.

	500	1000	1500
$g_{\hat{f}_\theta}$	0.072 ± 0.022	0.031 ± 0.005	0.024 ± 0.003
$\hat{f}_\theta$	0.296 ± 0.149	0.108 ± 0.059	0.066 ± 0.035

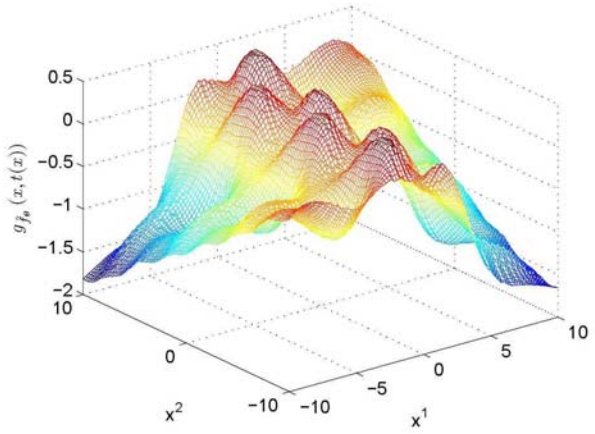
TABLE II
RMSE (MEAN $\pm$ DEVIATION).

which is illustrated on Table II (RMSE as a function of number of samples). There is more variance and bias in these results because of the possible invariances of the considered nonlinear mapping. However one can observe on Figure 6(a) that a good approximation is obtained.

Thus the RMSE computed from g_f is a better quality measure. As far as we know, no other method to handle such a problem has been published so far, thus comparisons with other approaches is made difficult. However the order of magnitude for the RMSE obtained from g_f is comparable with the state-of-the-art regression approaches when the function of interest is directly observed. This is demonstrated on Table III. Here the problem is to regress a linear 2d cardinal sine $f(x^1, x^2) = \frac{\sin(x^1)}{x^1} + \frac{x^2}{10}$. For the proposed algorithm, the



(a) Approximation of $f(x)$.



(b) Nonlinear mapping calculated from $\hat{f}_\theta(x)$.

Fig. 6. Approximated functions.

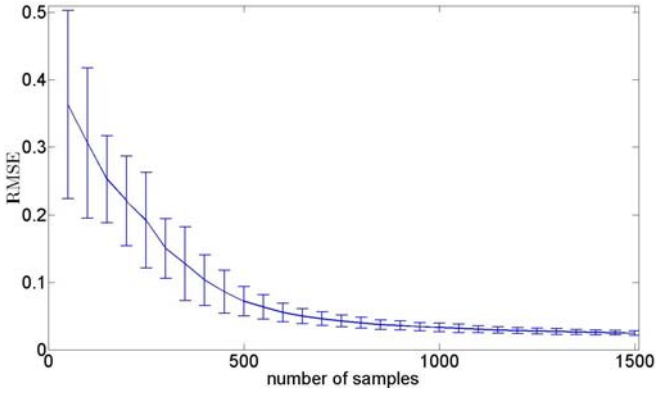


Fig. 7. RMSE (nonlinear mapping).

Method	test error $\hat{f}$	test error $g_{\hat{f}}$	%DV/SVs
Proposed Alg.	2.45×10^{-1}	4.5×10^{-2}	2.6%
KRLS	1.5×10^{-2}		7%
SVR	5.5×10^{-2}		60%

TABLE III
COMPARATIVE RESULTS.

nonlinear mapping is the same as considered before. RMSE results of Algorithm 4 are compared to Kernel Recursive Least Squares (KRLS) and Support Vector Regression (SVR). For the proposed approach, the approximation is computed from nonlinear mapping of observations. For KLRS and SVR, it is computed from direct observations. Notice that SVR is a batch method, that is it needs the whole set of samples in advance. The target of the regressor is indeed g_f, and the same order of magnitude is obtained (however much nonlinearities are introduced for this method and the representation is more sparse). The RMSE on $\hat{f}$ is slightly higher for this contribution, but this can be mostly explained by the invariances (as bias invariance) induced by the nonlinear mapping.

V. APPLICATION TO REINFORCEMENT LEARNING

The work presented in this section is a rather direct application of the general algorithm proposed in Section IV. However, before presenting the so-called *Bayesian Reward Filter*, the reinforcement learning paradigm is briefly introduced. Notice that the notion of state used in this section is different from the one used in Section II-C, despite same name and notation. Here the state is the state of a dynamic system to be controlled and not a parameter vector. Moreover it is directly observable and there is no need to infer it.

A. Reinforcement Learning Paradigm

Reinforcement learning (RL) [11] is a general paradigm in which an agent learns to control a dynamic system only through interactions. A feedback signal is provided to this agent as a reward information, which is a local hint about the quality of the control. Markov Decision Processes (MDP) are a common framework to solve this problem. A MDP is fully described by the state space that can be explored, the action set that can be chosen by the agent, a family of transition probabilities between states conditioned by the actions and a set of expected rewards associated to transitions. This is further explained in Section V-B. In this framework, at each time step k, the system adopts a state s_k. According to this, the agent can chose an action a_k, which leads to a transition to state s_{k+1} and to the obtention of a reward r_k, the agent objective being to maximize the future expected cumulative rewards. Here the knowledge of the environment is modelled as a Q-function which maps state action pairs to the expected cumulative rewards when following a given associated policy after the first transition. The proposed approach is model-free, no model of transitions and reward distributions is (directly or explicitly) learnt or known.

Solutions exist for the RL problem with discrete state and action spaces. However they generally do not scale up very well and cannot handle continuous state and/or action spaces. A wide variety of function approximation schemes have thus been applied to reinforcement learning (see [11] as a starting point). This is known as the generalization problem, and it is proposed here to handle it with a Bayesian filtering approach. The idea to use Bayesian filtering for reinforcement learning is not novel, but it has been surprisingly little studied. In [19] a modification of the linear quadratic Gaussian Kalman filter model is proposed, which allows the on-line estimation of optimal control (which is off-line for the classical one). In [20] Gaussian processes are used for reinforcement learning. This method can be understood as an extension of the Kalman filter to an infinite dimensional hidden state (the Gaussian process), but it can only handle (optimistic) policy-iteration-like update rules (because of the necessary linearity of the observation equation), contrarily to the proposed contribution, which can be seen as a nonlinear extension of the parametric case developed in [20] to a nonlinear and value-iteration-like scheme. In [21] a Kalman filter bank is used to find the parameters of a piecewise linear approximation of the value function.

B. Problem Statement

A Markov Decision Process (MDP) consists of a state space S, an action space A, a Markovian transition probability $p : S \times A \to \mathcal{P}(S)$ and a bounded reward function $r : S \times A \times S \to \mathbb{R}$. A policy is a mapping from state to action space: $\pi : S \to A$. At each time step k, the system is in a state s_k, the agent chooses an action $a_k = \pi(s_k)$, and the system is then driven in a state s_{k+1} following the conditional probability distribution $p(.|s_k, a_k)$. The agent receives the associated reward $r_k = r(s_k, a_k, s_{k+1})$. Its goal is to find the policy which maximizes the expected cumulative rewards, that is the quantity $E_\pi[\sum_{k\in\mathbb{N}} \gamma^k r(S_k, A_k, S_{k+1})|S_0 = s_0]$ for every possible starting state s_0, the expectation being over the state transitions taken upon executing π, where $\gamma \in [0, 1[$ is a discount factor.

A classical approach to solve this optimization problem is to introduce the Q-function defined as:

$$Q^\pi(s,a) = \int_S p(z|s,a)\Big(r(s,a,z) + \gamma Q^\pi(z,\pi(z))\Big)dz \quad (54)$$

It is the expected cumulative rewards by taking action a in state s and then following the policy π. The optimality criterion is to find the policy π^* (and associated Q^*) such that for every state s and for every policy π, $\max_{a\in A} Q^*(s,a) \geq \max_{a\in A} Q^\pi(s,a)$. The optimal Q-function Q^* satisfies the Bellman's equation:

$$Q^*(s,a) = \int_S p(z|s,a)\left(r(s,a,z) + \gamma \max_{b\in A} Q^*(z,b)\right)dz \quad (55)$$

In the case of discrete and finite action and state spaces, the Q-learning algorithm provides a solution to this problem. Its principle is to update a tabular approximation of the optimal Q-function after each transition (s, a, r, s'):

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\left(r + \gamma \max_{b\in A} \hat{Q}(s',b) - \hat{Q}(s,a)\right) \quad (56)$$

where α is a learning rate. An interesting fact is that the Q-learning is an off-policy algorithm, that is it allows to learn the optimal policy (from the learned optimal Q-function) while following a suboptimal one, given that it is sufficiently explorative. The proposed contribution can be seen as an extension of this algorithm to a Bayesian filtering framework (however with other advantages). See [11] for a comprehensive introduction to reinforcement learning, or [22] for a more formal treatment.

The reward is what is observed, the Q function is what is searched, and both are linked by the Bellman equation. Suppose that the Q-function is parameterized (either linearly or nonlinearly) by a vector θ. The aim is to find a good approximation $\hat{Q}_\theta$ of the optimal Q-function Q^* by observing transitions (s, a, r, s'). This reward regression problem is cast into a state-space representation. For an observed transition (s_k, a_k, r_k, s'_k), it is written as:

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ r_k = \hat{Q}_{\theta_k}(s_k,a_k) - \gamma \max_{a\in A} \hat{Q}_{\theta_k}(s'_k,a) + n_k. \end{cases} \quad (57)$$

Here v_k is the artificial process noise and n_k the centered observation noise including all the stochasticity of the MDP. The framework is thus posed, but is far from being solved. The observation equation is nonlinear and even non-derivable (because of the $\max$ operator), that is why classical methods such as the standard Kalman filter cannot be used. Formally, the process noise is null, nevertheless introducing an artificial noise can improve the stability and convergence performances of the filter, as discussed before. This can be seen as a special case of the algorithm of the previous section, which is developed next.

C. Algorithm

As before Gaussian kernels are chosen, and their mean and deviation are considered as parameters:

$$\hat{Q}_\theta(s,a) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i^s}(s,s_i) K_{\sigma_i^a}(a,a_i) \quad (58)$$

$$\text{with } K_{\sigma_i^x}(x,x_i) = \exp\left(-(x-x_i)^T(\Sigma_i^x)^{-1}(x-x_i)\right),$$

$$\text{where } x = s,a,\ \Sigma_i^x = \mathrm{diag}(\sigma_i^x)^2,$$

$$\text{and } \theta = [(\alpha_i)_{i=1}^p, (s_i^T)_{i=1}^p, (a_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p, ((\sigma_i^a)^T)_{i=1}^p]^T$$

Once again, the dictionary method is used to automatize the choice of the structure for this kernel parameterization and the *ad hoc* prior is chosen. Once the parameters are initialized, the parameter vector has still to be updated as new observations (s_k, a_k, r_k, s'_k) are available. A SPKF is used, and this is a special case of the theory developed in Section IV, the

functions t and g being given by:

$$t : S \times A \to S \qquad s, a \mapsto s' \tag{59}$$

and

$$g : \mathbb{R}^{S\times A} \to \mathbb{R}^{S\times A\times S} \qquad Q \mapsto \left(s, a, s' \mapsto Q(s,a) - \gamma \max_{b\in A} Q(s',b) \right) \tag{60}$$

Thus Algorithm 4 can be specialized for reinforcement learning, which gives the Bayesian Reward Filter summarized in Algorithm 5.

Algorithm 5: Bayesian Reward Filter

Compute dictionary;
$\forall i \in \{1 \dots N\},\ [s_i, a_i]^T \sim \mathcal{U}_{S\times A}$;
Set $X = \{[s_1, a_1]^T, \dots, [s_N, a_N]^T\}$;
$\mathcal{D}$ =`Compute-Dictionary` $(S \times A, \nu, \sigma_0)$;

Initialisation;
Initialise $\bar{\theta}_0$, $P_{0|0}$, R_{n_0}, R_{v_0};

for $k = 1, 2, \dots$ **do**
Observe (s_k, a_k, s'_k, r_k);

SPKF update;

Prediction Step;
$\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$;
$P_{k|k-1} = P_{k-1|k-1} + P_{v_{k-1}}$;

Sigma-points computation ;
$\Theta_{k|k-1} = \left\{ \theta^{(j)}_{k|k-1}, \quad 0 \le j \le 2q \right\}$;
$\mathcal{W} = \{ w_j, \quad 0 \le j \le 2q \quad \}$;
$\mathcal{R}_{k|k-1} = \Big\{ r^{(j)}_{k|k-1} = \hat{Q}_{\theta^{(j)}_{k|k-1}}(s_k, a_k) -$
$\gamma \max_{b\in A} \hat{Q}_{\theta^{(j)}_{k|k-1}}(s'_k, b),\ 0 \le j \le 2q \Big\}$;

Compute statistics of interest;
$\bar{r}_{k|k-1} = \sum_{j=0}^{2q} w_j r^{(j)}_{k|k-1}$;
$P_{\theta r_k} = \sum_{j=0}^{2q} w_j (\theta^{(j)}_{k|k-1} - \bar{\theta}_{k|k-1})(r^{(j)}_{k|k-1} - \bar{r}_{k|k-1})$;
$P_{r_i} = \sum_{j=0}^{2q} w_j \left(r^{(j)}_{k|k-1} - \bar{r}_{k|k-1} \right)^2 + P_{n_k}$;

Correction step;
$K_k = P_{\theta y_k} P_{y_k}^{-1}$;
$\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(r_k - \bar{r}_{k|k-1} \right)$;
$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;

Artificial process noise update;
Robbins-Monro covariance;
$R_{v_k} = (1 - \alpha_{RM}) R_{v_{k-1}} + \alpha_{RM} K_k (r_k +$
$\gamma \max_b \hat{Q}_{\bar{\theta}_{k|k}}(s'_k, b) - \hat{Q}_{\bar{\theta}_{k|k}}(s_k, a_k))^2 K_k^T$;
or;
Forgetting covariance;
$R_{v_k} = (\lambda^{-1} - 1) P_{k|k}$;

D. Maximum over action space

Notice that a technical difficulty can be to compute the maximum over the actions for the parameterized Q-function. This computation is necessary for the filter update. If the action space is discrete and finite, computation of the $\max$ is easy. Otherwise it is an optimization problem. A first solution could be to sample the action space and to compute the maximum over the obtained samples. However this is especially computationally inefficient. The used method is close to one proposed in [23].

The maximum over action kernel centers is computed: $\mu^a = \operatorname{argmax}_{a_i} \hat{Q}_\theta(s, a_i)$. It serves then as the initialization for the Newton-Raphson method used to find the maximum :

$$a_m \leftarrow a_m - \left((\nabla_a \nabla_a^T) \hat{Q}_\theta(s,a) \right)^{-1}_{a=a_m} \nabla_a \hat{Q}_\theta(s,a) \Big|_{a=a_m}$$

If the Hessian matrix is singular, a gradient ascent/fixed point scheme is used:

$$a_m \leftarrow a_m + \nabla_a \hat{Q}_\theta(s,a) \Big|_{a=a_m}$$

The obtained action a_m is considered as the action which maximizes the parameterized Q-function.

E. Experiments

The proposed Bayesian Reward Filter is demonstrated on two problems. First, the "wet-chicken" task is a continuous state and action space and stochastic problem. Second, the "mountain car" problem is a continuous state, discrete action and deterministic problem. The latter one requires an hybrid parametrization. Let's first discuss the choice of parameters. They are given for reproducibility, nevertheless the reader can skip this without hurting understanding.

1) Choice of Parameters: For both tasks the reinforcement learning discount factor is set to $\gamma = 0.9$. The dictionary sparsity factor is set to $\nu = 0.9$. The second type of artificial process noise presented in Section III-C is used, and similarly to the recursive least-squares algorithm, the adaptive process noise covariance is set to a high value, such that $\lambda^{-1} - 1 \simeq 10^{-6}$.

The initial choice of kernel deviations is problem dependant. However a practical good choice seems to take a fraction of the quantity $x(j)_{\max} - x(j)_{\min}$ for the kernel deviation associated to $x(j)$, the j^{th} component of the column vector x, $x(j)_{\max}$ and $x(j)_{\min}$ being the bounds on the values taken by the variable $x(j)$. The prior kernel weights are supposed to be centered, and the associated standard deviations are set to a little fraction of the theoretical bound on Q-function, that is $\frac{r_{\max}}{1-\gamma}$. Because of geometry of Gaussian distributions, centers provided by the dictionary are supposed to be approximately uniformly distributed, and the prior deviation of the j^{th} component of the vector x is set to a little fraction of $(x(j)_{\max} - x(j)_{\min}) p^{-\frac{1}{ns+na}}$, with the convention that for discrete spaces $n = 0$. Finally the deviation of the prior kernel deviations is set to a little fraction of them. Otherwise speaking, $\sigma_{\sigma_0^{x(j)}}$ is set to a little fraction of $\sigma_0^{x(j)}$ for the j^{th} component of x.

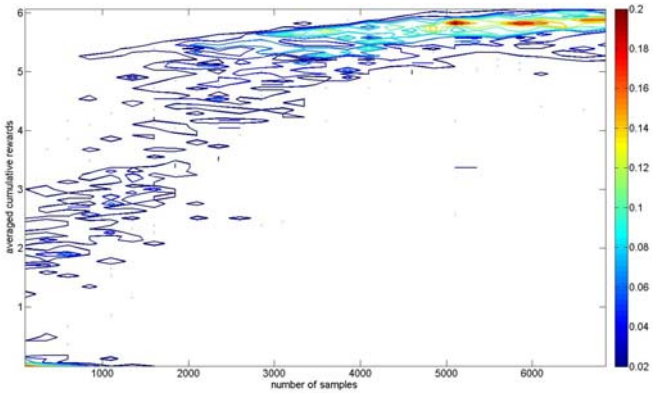


Fig. 8. Wet-chicken.

To sum up, the Gaussian prior on parameterization is chosen such that:

$$\begin{cases} \sigma_0^{x(j)} \propto x(j)_{\max} - x(j)_{\min} \\ \mu_{\alpha_0} = 0 \text{ and } \sigma_{\alpha_0} \propto \frac{r_{\max}}{1-\gamma} \\ \sigma_{\mu_0^{x(j)}} \propto \frac{x(j)_{\max} - x(j)_{\min}}{(n_s+n_a)\sqrt{p}} \\ \sigma_{\sigma_0^{x(j)}} \propto \sigma_0^{x(j)} \end{cases} \tag{61}$$

2) Wet-Chicken: In the wet-chicken problem (inspired by [24]), a canoeist has to paddle on a river until reaching a waterfall. It restarts if it falls down. Rewards increase linearly with the proximity of the waterfall, and drop off for falling. Turbulences make the transition probabilistic. More formally, the state space is $S = [0, 10]$ (10 being the waterfall position), the action space $A = [-1, 1]$ (continuously from full backward padding to full forward padding), the transition is $s' = s+a+c$ with $c \sim \mathcal{N}(0, \sigma_c)$, $\sigma_c = 0.3$ and the associated reward is equal to $r = \frac{s'}{10}$. If $s' \geq 10$ the canoeist falls, the associated reward is $r = -1$ and the episode ends.

To test the proposed framework, random transitions are uniformly sampled and used to feed the filter: at each time step k, a state s_k and an action a_k are uniformly sampled from $S \times A$, and used to generate a (random) transition to s'_k, with associated reward r_k, and the transition (s_k, a_k, s'_k, r_k) is the input of the algorithm. The results are shown on Figure 8. For each run of the algorithm and every 250 samples, the expected cumulative rewards for the current policy has been computed as an average of cumulative rewards over 1000 episodes which were randomly (uniform distribution) initiated (thus the average is done over starting states and stochasticity of transitions). Notice that the lifetime of the agent (the duration of an episode) was bounded to 100 interactions with its environment. Then a two dimensional histogram of those averaged cumulative rewards is computed over 100 different runs of the algorithm. In other words, the distribution of cumulative rewards over different runs of the algorithm as a function of the number of observed transitions is shown. The bar on the right shows the percentages associated to the histogram.

The optimal policy has an averaged cumulative rewards of 6. It can be seen on Figure 8 that the proposed algorithm can learn near optimal policies. After 1000 samples some of policies can achieve a score of 5 (84% of the optimal policy), which is achieved by a majority of the policies after 3000 samples. After 7000, very close to optimal policies were found in almost all runs of the algorithm (the mode of the associated distribution is at 5.85, that is 98% of the optimal policy). To represent the approximated Q-function, 7.7 ± 0.7 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

Two remarks of interest have to be made on this benchmark. First, the observation noise is input-dependant, as it models the stochasticity of the MDP. Recall that here a constant observation noise has been chosen. Secondly, the noise can be far to Gaussianity. For example, in the proximity of the waterfall it is bimodal because of the shift of reward. Recall that the proposed filter assumes Gaussianity of noises. Thus it can be concluded that the proposed approach is quite robust, and that it achieves good performance considering that the observations were totally random (off-policy aspect).

3) Mountain Car: The second problem is the mountain-car task. A underpowered car has to go up a steep mountain road. The state is 2-dimensional (position and velocity) and continuous, and there are 3 actions (forward, backward and zero throttle). The problem full description is given in [11]. A null reward is given at each time step, and $r = 1$ is given when the agent reaches the goal.

A first problem is to find a parameterization for this task. The proposed one is adapted for continuous problems, not hybrid ones. But this approach can be easily extended to continuous states and discrete actions tasks. A simple solution consists in having a parameterization for each discrete action, that is a parametrization of the form $\theta = [\theta^{a_1}, \theta^{a_2}, \theta^{a_3}]$ and an associated Q-function $Q_\theta(s, a) = Q_{\theta^a}(s)$. But it can be noticed that for a fixed state and different actions the Q-values will be very close. In other words $Q^*(s, a_1)$, $Q^*(s, a_2)$ and $Q^*(s, a_3)$ will have similar shapes, as functions over the state space. Thus consider that the weights will be specific to each action, but the kernel centers and deviations will be shared over actions. More formally the parameter vector is

$$\theta = [(\alpha_i^{a_1})_{i=1}^p, (\alpha_i^{a_2})_{i=1}^p, (\alpha_i^{a_3})_{i=1}^p, (s_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p]^T \tag{62}$$

the notation being the same as in the previous sections.

As for the wet-chicken problem, the filter has been fed with random transitions. Results are shown on Figure 9, which is a two-dimensional histogram similar to the previous one. The slight difference is that the performance measure is now the "cost-to-go" (the number of steps needed to reach the goal). It can be linked directly to the averaged cumulative rewards, however it is more meaningful here. For each run of the algorithm and every 250 samples, the expected cost-to-go for the current policy has been computed as an average of 1000 episodes which were randomly initiated (average is only done over starting states here, as transitions are deterministic). The

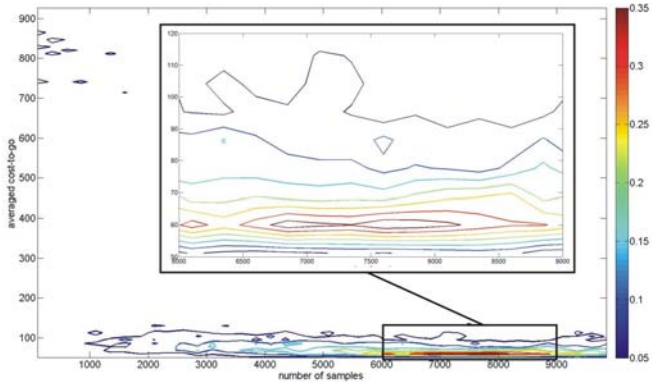


Fig. 9. Mountain car.

lifetime of the agent was bounded to 1000 interactions with its environment. The histogram is computed over 100 runs.

The optimal policy has an averaged cost-to-go of 55. It can be seen on Figure 9 that the proposed algorithm can find near optimal policies. After 1500 samples most of policies achieve a cost-to-go smaller than 120. After 6000 samples, policies very close to the optimal one were found in almost all runs of the algorithm (the mode of the associated distribution is at 60). To represent the approximated Q-function, 7.5 ± 0.8 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

This problem is not stochastic, however informative rewards are very sparse (which can cause the Kalman gain to converge too quickly), transitions are nonlinear and rewards (that is observations) are binary. Despite this, the proposed filter exhibits good convergence. Once again it can be concluded that the proposed approach achieves good results considering the task at hand.

4) Comparison to Other Methods: For now the proposed algorithm treats the different control tasks as regression problems (learning from random transitions), thus it is ill comparable to state-of-the-art reinforcement learning algorithms which learns from trajectories. Nevertheless it is argued that the quality of learned policies is comparable to state-of-the art methods. Measuring this quality depends on the problem settings and on the measure of performance, however the Bayesian reward filter finds very close to optimal policies. See [24] for example.

In most approaches, the system is controlled while learning, or for batch methods observed samples come from a suboptimal policy. In the proposed experiments, totally random transitions are observed. However for the mountain-car problem it is often reported that at least a few hundreds of episodes are required to learn a near-optimal policy (see for example [11]), and each episode may contain from a few tens to hundreds steps (this depends on the quality of the current control). In the proposed approach a few thousands of steps have to be observed in order to obtain a near optimal policy. This is roughly the same order of magnitude for convergence speed.

VI. Conclusion

A Bayesian approach to online nonlinear kernel regression with a preprocessing sparse structure automatization procedure has been proposed. This method has proven to be effective (from a RMSE point of view) on a simple cardinale sine regression problem. This example demonstrated that the proposed approach compares favorably with the state-of-the-art methods and it illustrated how the uncertainty of generalization is quantified.

An approach allowing to regress a function of interest f from observations which are obtained through a nonlinear mapping of it has also been proposed as an extension. The regression is still online, kernel-based, nonlinear and Bayesian. This method has proven to be effective on an artificial problem and reaches good performance although much more nonlinearities are introduced.

Finally the proposed approach has been specialized into a general Bayesian filtering framework for reinforcement learning. By observing rewards (and associated transitions) the filter is able to infer a near-optimal policy (through the parameterized Q-function). It has been tested on two reinforcement learning benchmarks, each one exhibiting specific difficulties for the algorithm. This off-policy Bayesian reward filter has been shown to be efficient on these two continuous tasks.

However, this paper did not demonstrated all the potentialities of the proposed framework. The Bayesian filtering approach allows to derive uncertainty information over estimated Q-function which can be used to handle the so-called exploration-exploitation dilemma, in the spirit of [25] or [26]. This could allow to speed-up and to enhance learning. This uncertainty information can also be useful from a regression perspective if used for active learning. Moreover, the partial observability problem (the issue of non-directly observable state in RL) can be quite naturally embedded in such a Bayesian filtering framework, as the Q-function can be considered as a function over probability densities. Finally, the observation noise arising in the Bayesian Reward Filter is not purely white if the Markovian decision process has stochastic transition probabilities. The whiteness of this noise being a necessary assumption for the derivation of the sigma-point Kalman filter, which is a baseline of the proposed framework, this aspect should be investigated further.

Acknowledgment

Olivier Pietquin thanks the Région Lorraine and the European Community (CLASSiC project, FP7/2007-2013, grant agreement 216594) for financial support.

References

[1] M. Geist, O. Pietquin, and G. Fricout, "A Sparse Nonlinear Bayesian Online Kernel Regression," in *Proceedings of the Second IEEE International Conference on Advanced Engineering Computing and Applications in Sciences (AdvComp 2008)*, vol. I, Valencia (Spain), October 2008, pp. 199–204.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, 1995.

[3] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[4] V. N. Vapnik, *Statisical Learning Theory*. John Wiley & Sons, Inc., 1998.

[5] L. Feldkamp and G. Puskorius, "A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering, and classification," in *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2259–2277.

[6] R. van der Merwe, "Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models," Ph.D. dissertation, OGI School of Science & Engineering, Oregon Health & Science University, April 2004.

[7] C. M. Bishop and M. E. Tipping, "Bayesian Regression and Classification," in *Advances in Learning Theory: Methods, Models and Applications*, vol. 190. OS Press, NATO Science Series III: Computer and Systems Sciences, 2003, pp. 267–285.

[8] J. Vermaak, S. J. Godsill, and A. Doucet, "Sequential Bayesian Kernel Regression," in *Advances in Neural Information Processing Systems 16*. MIT Press, 2003.

[9] Z. Chen, "Bayesian Filtering : From Kalman Filters to Particle Filters, and Beyond," Adaptive Systems Lab, McMaster University, Tech. Rep., 2003.

[10] R. Bellman, *Dynamic Programming*, 6th ed. Dover Publications, 1957.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, 3rd ed. The MIT Press, March 1998. [Online]. Available: http://www.cs.ualberta.ca/ sutton/book/the-book.html

[12] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, pp. 2275–2285, 2004.

[13] M. Geist, O. Pietquin, and G. Fricout, "Online Bayesian Kernel Regression from Nonlinear Mapping of Observations," in *Proceedings of the 18th IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2008)*, no. a53, Cancun (Mexico), October 2008, pp. 309–314.

[14] ——, "Bayesian Reward Filtering," in *Proceedings of the European Workshop on Reinforcement Learning (EWRL 2008)*, ser. Lecture Notes in Artificial Intelligence, S. G. et al., Ed. Lille (France): Springer Verlag, June 2008, vol. 5323, pp. 96–109.

[15] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[16] S. J. Julier and J. K. Uhlmann, "Unscented Filtering and Nonlinear Estimation," in *Proceedings of the IEEE*, vol. 92, no. 3, March 2004, pp. 401–422.

[17] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*, 1st ed. Wiley & Sons, August 2006.

[18] A. L. Strehl, L. Li, and M. L. Littman, "Incremental model-based learners with formal learning-time guarantees," in *22nd Conference on Uncertainty in Artificial Intelligence*, 2006, pp. 485–493.

[19] I. Szita and A. Lőrincz, "Kalman Filter Control Embedded into the Reinforcement Learning Framework," *Neural Comput.*, vol. 16, no. 3, pp. 491–499, 2004.

[20] Y. Engel, "Algorithms and Representations for Reinforcement Learning," Ph.D. dissertation, Hebrew University, April 2005.

[21] C. W. Phua and R. Fitch, "Tracking Value Function Dynamics to Improve Reinforcement Learning with Piecewise Linear Function Approximation," in *ICML 07*, 2007.

[22] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 1995.

[23] M. A. Carreira-Perpinan, "Mode-Finding for Mixtures of Gaussian Distributions," *IEEE Transsactions on Pattern Analalysis and Machine Intelligence*, vol. 22, no. 11, pp. 1318–1323, 2000.

[24] D. Schneegass, S. Udluft, and T. Martinetz, "Kernel Rewards Regression: an Information Efficient Batch Policy Iteration Approach," in *AIA'06: Proceedings of the 24th IASTED international conference on Artificial intelligence and applications*. Anaheim, CA, USA: ACTA Press, 2006, pp. 428–433.

[25] R. Dearden, N. Friedman, and S. J. Russell, "Bayesian Q-learning," in *Fifteenth National Conference on Artificial Intelligence*, 1998, pp. 761–768. [Online]. Available: http://www.cs.bham.ac.uk/ rwd/

[26] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC Model-Free Reinforcement Learning," in *23rd International Conference on Machine Learning (ICML 2006)*, Pittsburgh, PA, USA, 2006, pp. 881–888. [Online]. Available: http://paul.rutgers.edu/ strehl/