

# A BAYESIAN APPROACH TO TRACKING WITH KERNEL RECURSIVE LEAST-SQUARES

*Miguel Lázaro-Gredilla, Steven Van Vaerenbergh and Ignacio Santamaría*

{miguellg, steven, nacho}@gtas.dicom.unican.es  
Department of Communications Engineering  
University of Cantabria, Spain

## ABSTRACT

In this paper we introduce a kernel-based recursive least-squares (KRLS) algorithm that is able to track nonlinear, time-varying relationships in data. To this purpose we first derive the standard KRLS equations from a Bayesian perspective (including a principled approach to pruning) and then take advantage of this framework to incorporate forgetting in a consistent way, thus enabling the algorithm to perform tracking in non-stationary scenarios. In addition to this tracking ability, the resulting algorithm has a number of appealing properties: It is online, requires a fixed amount of memory and computation per time step and incorporates regularization in a natural manner. We include experimental results that support the theory as well as illustrate the efficiency of the proposed algorithm.

*Index Terms*— kernel recursive-least squares, tracking, Bayesian inference, adaptive filtering, forgetting.

## 1. INTRODUCTION

Kernel methods offer an attractive framework to deal with nonlinear signal processing problems [1]. By relying on a transformation of the data into a high-dimensional reproducing kernel Hilbert space, they are able to solve learning problems that are nonlinear in the input space as linear problems in the transformed space. Using the “kernel trick” efficient algorithms with algebraically simple expressions are obtained, such as support vector machines or kernel principal component analysis [1]. However, the functional representation of classical kernel-based algorithms grows linearly with the number of processed data, and therefore they are not directly suitable for online applications, where the complexity of each update must be limited [2].

The recursive least-squares (RLS) filter [3] is a popular algorithm that is used extensively in many engineering applications thanks to its good convergence and reasonably low complexity. In [4], a kernel recursive least-squares (KRLS) algorithm was proposed. To avoid an “evergrowing” functional representation of its solution, it features an online sparsification technique that allows to avoid redundancy in the solution’s support. In particular, the support is reduced to a sparse “dictionary” of bases and a new basis is only added if it cannot be represented by a combination of other bases that are already present in the dictionary. A number of different criteria for online sparsification have since been explored in the literature [5], including an online pruning criterion [6].

An important difference between linear adaptive filters and their kernel-based counterparts has to do with their ability to handle non-stationary data. While most linear adaptive filters allow for tracking

non-stationary data directly or through some straightforward extension of their standard form [3], such an extension is more complicated in the case of kernel-based algorithms. Most kernel adaptive filters are therefore designed to operate on stationary data only, and they converge approximately to the batch filtering solution [4, 5]. As a result, tracking is an aspect of kernel adaptive filtering that has not been satisfactorily addressed yet in the literature.

In order to perform tracking, more weight should be given to more recent data. A radical approach to this idea is found in sliding-window algorithms, whose solution depends only on the latest  $M$  observed data [7, 8], while any older data is discarded. This procedure is suboptimal for tracking time-varying data, since the quality of its solution depends on the bases present in its support, and all samples are given the same importance. A few attempts have also been made to allow tracking by extending the standard KRLS setting with a forgetting factor [5] or a simplified state-space model [9]. Nevertheless, while these algorithms theoretically allow for tracking, they are not capable of limiting dictionary growth at the same time. Furthermore, they present numerical difficulties that do not allow their practical implementation on finite-precision machines.

In this paper we explore a more principled approach to tracking. We specifically handle the uncertainty about the inferred input-output relationship, which we consider a latent function, and we study the problem of how older data should be forgotten. First, we define a probabilistic framework based on Gaussian processes (GPs) that offers an intuitive view of KRLS and allows to deal with uncertainty and regularization in a natural manner. Then, we propose a reasonable model for forgetting, which shifts the probability distribution over the input-output relationship towards the prior belief. In this manner, the solution converges to the prior belief once all data are forgotten, which is consistent with the Bayesian framework. The presented method is closely related to the work of Csató and Opper in [10], in which a GP perspective is adopted. We aim to bring their principled approach to the attention of the signal processing community, provide an (arguably) more intuitive sparse formulation, and extend it by including forgetting. The resulting KRLS algorithm is capable of tracking non-stationary data that exhibit nonlinear relationships. In order to guarantee online operation, we limit its memory and computational complexity in each step to  $\mathcal{O}(M^2)$ , where  $M$  is the number of bases allowed in memory at any given time.

## 2. A BAYESIAN PERSPECTIVE OF KRLS

In this section we provide a Bayesian derivation of previous results for KRLS, obtained within the probabilistic framework of Gaussian Processes (GPs). This unifying interpretation offers a simpler, more intuitive view of KRLS, adds specific handling of uncertainty, and will prove specially useful to operate in non-stationary scenarios.

This work was supported by MICINN (Spanish Ministry for Science and Innovation) under grants TEC2010-19545-C04-03 (COSIMA) and CONSOLIDER-INGENIO 2010 CSD2008-00010 (COMONSENS).

## 2.1. Standard KRLS (with evergrowing dictionary)

Assume a set of ordered input-output pairs  $\mathcal{D}_t \equiv \{\mathbf{x}_i, y_i\}_{i=1}^t$ , where  $\mathbf{x}_i \in \mathbb{R}^D$  are  $D$ -dimensional input vectors and  $y_i \in \mathbb{R}$  are scalar outputs. Data pairs are made available on a one-at-a-time basis, i.e.,  $(\mathbf{x}_t, y_t)$  is made available at time  $t$ . Our objective is to infer the predictive distribution of a new, unseen output  $y_{t+1}$  given the corresponding input  $\mathbf{x}_{t+1}$  and data available up to time  $t$ ,  $\mathcal{D}_t$ .

### 2.1.1. Bayesian model

In a Bayesian setting, we need a model that describes the observations, and priors on the parameters of such model. Following the standard setup of GP regression, we can describe observations as the sum of an unobservable *latent function* of the inputs plus zero-mean Gaussian noise

$$y_i = f(\mathbf{x}_i) + \varepsilon_i. \quad (1)$$

In order to perform Bayesian inference, we also need a prior over the latent function, which is taken to be a zero-mean GP with covariance function  $k(\mathbf{x}, \mathbf{x}')$ . The use of a GP prior has a simple meaning: It implies that the prior joint distribution of vector  $\mathbf{f}_t = [f_1, \dots, f_t]^\top$  (where  $f_i = f(\mathbf{x}_i)$ ) is a zero-mean multivariate Gaussian with covariance matrix  $\mathbf{K}_t$ , with elements  $[\mathbf{K}_t]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ . In line with the previous literature on KRLS, we will refer to  $k(\mathbf{x}, \mathbf{x}')$  as the *kernel function*, and to  $\mathbf{K}_t$  as the *kernel matrix* (which in this example would correspond to inputs  $\{\mathbf{x}_1, \dots, \mathbf{x}_t\}$ ). For the sake of clarity, in the following we will omit conditioning on the inputs  $\{\mathbf{x}_i\}_{i=1}^t$  or the parameters  $\theta$  that parameterize the kernel function.

### 2.1.2. Bayesian recursive update

The adopted setup corresponds to standard GP regression, which is thoroughly described in [11] for the batch setting. Here we consider the online setting in which new observations are incorporated sequentially. Therefore, a new posterior  $p(\mathbf{f}_t|\mathcal{D}_t)$  including the most recent observation  $t$  must be computed at each time instant. We will compute the joint posterior only at the locations at which data is observed, instead of the full GP posterior. This is because  $p(\mathbf{f}_t|\mathcal{D}_t)$  implicitly defines the posterior full GP distribution  $p(f(\mathbf{x})|\mathcal{D}_t)$

$$p(f(\mathbf{x})|\mathcal{D}_t) = \int p(f(\mathbf{x})|\mathbf{f}_t)p(\mathbf{f}_t|\mathcal{D}_t)d\mathbf{f}_t \quad (2)$$

and therefore, both posterior distributions hold the same information.

Instead of recomputing  $p(\mathbf{f}_t|\mathcal{D}_t)$  from scratch at every time instant, we can obtain a simple recursive update as follows:

$$p(\mathbf{f}_{t+1}|\mathcal{D}_{t+1}) = p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_t, y_{t+1}) \quad (3a)$$

$$= \frac{p(y_{t+1}|f_{t+1})p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_t)}{p(y_{t+1}|\mathcal{D}_t)} \quad (3b)$$

$$= \frac{p(y_{t+1}|f_{t+1})p(f_{t+1}|\mathbf{f}_t)}{p(y_{t+1}|\mathcal{D}_t)} \times p(\mathbf{f}_t|\mathcal{D}_t). \quad (3c)$$

Eq. (3b) follows from (3a) by direct application of Bayes rule. Eq. (3c) includes an additional expansion due to  $f_{t+1}$  being conditionally independent of  $\mathcal{D}_t$  given  $\mathbf{f}_t$ .

If the posterior at time  $t$  is a known Gaussian  $p(\mathbf{f}_t|\mathcal{D}_t) = \mathcal{N}(\mathbf{f}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ , then we can use (3) to update it to include a new observation  $(\mathbf{x}_{t+1}, y_{t+1})$ . All the expression in (3) can be easily inferred from the stated assumptions as follows.

First, introducing the new quantities  $\mathbf{Q}_t = \mathbf{K}_t^{-1}$ ,  $\mathbf{q}_{t+1} = \mathbf{Q}_t \mathbf{k}_{t+1}$  and  $\gamma_{t+1}^2 = k_{t+1} - \mathbf{k}_{t+1}^\top \mathbf{Q}_t \mathbf{k}_{t+1}$ , where  $[\mathbf{k}_{t+1}]_i = k(x_i, x_{t+1})$  and  $k_{t+1} = k(x_{t+1}, x_{t+1})$ , we can express the density

of the latent function at the new input given its value at previous inputs as

$$p(f_{t+1}|\mathbf{f}_t) = \mathcal{N}(f_{t+1}|\mathbf{q}_{t+1}^\top \mathbf{f}_t, \gamma_{t+1}^2). \quad (4)$$

This result follows directly from the known prior probability  $p(f_{t+1}, \mathbf{f}_t)$  and conditioning on  $\mathbf{f}_t$ . The inverse of the kernel matrix,  $\mathbf{Q}_t$ , has been defined because we will be computing and storing it instead of  $\mathbf{K}_t$ , which will never be directly used.

Then, the denominator of Eq. (3), which corresponds to the marginal likelihood (also known as evidence), provides the predictive distribution of a new observation  $y_{t+1}$  given past data, and can be computed by integration of the numerator

$$\begin{aligned} p(y_{t+1}|\mathcal{D}_t) &= \int p(y_{t+1}|f_{t+1})p(f_{t+1}|\mathbf{f}_t)p(\mathbf{f}_t|\mathcal{D}_t)d\mathbf{f}_t df_{t+1} \\ &= \mathcal{N}(y_{t+1}|\hat{y}_{t+1}, \hat{\sigma}_{y_{t+1}}^2) \end{aligned} \quad (5)$$

with the mean and the variance of this Gaussian being

$$\hat{y}_{t+1} = \mathbf{q}_{t+1}^\top \boldsymbol{\mu}_t \quad \text{and} \quad \hat{\sigma}_{y_{t+1}}^2 = \sigma_n^2 + \hat{\sigma}_{f_{t+1}}^2,$$

respectively. We have also introduced the predictive variance of the latent function at the new input

$$\hat{\sigma}_{f_{t+1}}^2 = k_{t+1} + \mathbf{k}_{t+1}^\top (\mathbf{Q}_t \boldsymbol{\Sigma}_t \mathbf{Q}_t - \mathbf{Q}_t) \mathbf{k}_{t+1} = \gamma_{t+1}^2 + \mathbf{q}_{t+1}^\top \mathbf{h}_{t+1}$$

with  $\mathbf{h}_{t+1} = \boldsymbol{\Sigma}_t \mathbf{q}_{t+1}$ .

Finally, the likelihood follows from the model definition (1):

$$p(y_{t+1}|f_{t+1}) = \mathcal{N}(y_{t+1}|f_{t+1}, \sigma_n^2). \quad (6)$$

Thus, all involved distributions (4) and (5), (6) are univariate normal with simple, known expressions. Using those, (3) can be evaluated and the posterior updates emerge

$$p(\mathbf{f}_{t+1}|\mathcal{D}_{t+1}) = \mathcal{N}(\mathbf{f}_{t+1}|\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \quad (7a)$$

$$\boldsymbol{\mu}_{t+1} = \begin{bmatrix} \boldsymbol{\mu}_t \\ \hat{y}_{t+1} \end{bmatrix} + \frac{y_{t+1} - \hat{y}_{t+1}}{\hat{\sigma}_{y_{t+1}}^2} \begin{bmatrix} \mathbf{h}_{t+1} \\ \hat{\sigma}_{f_{t+1}}^2 \end{bmatrix} \quad (7b)$$

$$\boldsymbol{\Sigma}_{t+1} = \begin{bmatrix} \boldsymbol{\Sigma}_t & \mathbf{h}_{t+1} \\ \mathbf{h}_{t+1}^\top & \hat{\sigma}_{f_{t+1}}^2 \end{bmatrix} - \frac{1}{\hat{\sigma}_{y_{t+1}}^2} \begin{bmatrix} \mathbf{h}_{t+1} \\ \hat{\sigma}_{f_{t+1}}^2 \end{bmatrix} \begin{bmatrix} \mathbf{h}_{t+1}^\top \\ \hat{\sigma}_{f_{t+1}}^2 \end{bmatrix}^\top. \quad (7c)$$

The inverse of the kernel matrix including the new input can also be easily computed using the corresponding low-rank update

$$\mathbf{K}_{t+1}^{-1} = \mathbf{Q}_{t+1} = \begin{bmatrix} \mathbf{Q}_t & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{\gamma_{t+1}^2} \begin{bmatrix} \mathbf{q}_{t+1} \\ -1 \end{bmatrix} \begin{bmatrix} \mathbf{q}_{t+1}^\top \\ -1 \end{bmatrix}^\top. \quad (8)$$

This illustrates both how probabilistic predictions for new observations can be made (using Eq. (5), which does not require knowledge of  $y_{t+1}$ ), and how these new observations can be included in the posterior once they are available. All computations for a given update can be made in  $\mathcal{O}(t^2)$  time, as is obvious from the update formulas. Only  $\boldsymbol{\mu}_{t+1}$ ,  $\boldsymbol{\Sigma}_{t+1}$  and  $\mathbf{Q}_{t+1}$  will be reused in the next iteration, and the remaining quantities will be computed on demand.

The recursion updates can be initialized by setting

$$\boldsymbol{\mu}_1 = \frac{y_1 k(\mathbf{x}_1, \mathbf{x}_1)}{\sigma_n^2 + k(\mathbf{x}_1, \mathbf{x}_1)} \quad (9)$$

$$\boldsymbol{\Sigma}_1 = k(\mathbf{x}_1, \mathbf{x}_1) - \frac{k(\mathbf{x}_1, \mathbf{x}_1)^2}{\sigma_n^2 + k(\mathbf{x}_1, \mathbf{x}_1)} \quad (10)$$

$$\mathbf{Q}_1 = \frac{1}{k(\mathbf{x}_1, \mathbf{x}_1)}, \quad (11)$$

which corresponds to inference according to the proposed model for a single data point.

Since the model is exactly that of GP regression and all provided formulas are exact, probabilistic predictions made at time  $t$  for observation  $t + 1$  are exactly the same as those obtained using a standard GP in the batch setting. Using the batch formulation from [11], we can equivalently express the predictive mean and variance from (5) as

$$\hat{y}_{t+1} = \mathbf{k}_{t+1}^\top (\mathbf{K}_t + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}_t \quad (12a)$$

$$\hat{\sigma}_{y_{t+1}}^2 = k_{t+1} - \mathbf{k}_{t+1}^\top (\mathbf{K}_t + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_{t+1}. \quad (12b)$$

Direct application of (12a) and (12b) involves a higher,  $\mathcal{O}(t^3)$  cost, so the recursive procedure of the previous section is preferred. However, these equations are useful to illustrate the form of the predictive distribution after several iterations, which is somewhat obscured in the recursive formulation.

In the standard KRLS setting, the predictive mean is often expressed as  $\hat{y}_{t+1} = \mathbf{k}_{t+1}^\top \boldsymbol{\alpha}_t$ , where  $\boldsymbol{\alpha}_t$  weights each kernel. When the batch formulation is used, these weights can be obtained as  $\boldsymbol{\alpha}_t = (\mathbf{K}_t + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}_t$ , whereas in our recursive formulation the same result can be obtained at each step  $t$  by computing  $\boldsymbol{\alpha}_t = \mathbf{K}_t^{-1} \boldsymbol{\mu}_t = \mathbf{Q}_t \boldsymbol{\mu}_t$ . Observe the resemblance between the batch and recursive formulations: In the batch formulation we are using *noisy* observations  $\mathbf{y}_t$ , so the kernel matrix includes a regularization term  $\sigma_n^2 \mathbf{I}$ . In the recursive formulation we use  $\boldsymbol{\mu}_t$ , which are the values of the *noiseless* function evaluated at the inputs, so no noise term is added. Obviously, the same value for  $\boldsymbol{\alpha}_t$  is obtained with both formulations.

## 2.2. Fixed-budget KRLS

The set of locations at which the joint posterior is stored is usually referred to as *set of bases* or *dictionary*. The recursive procedure grows this dictionary unboundedly. A simple strategy to limit resource usage is to remove one basis from the dictionary whenever it grows larger than a predefined budget  $M$ . In order to accomplish this, we need to know *how* to remove a basis from the dictionary, and a criterion to select *which* basis should be removed. Both are described in the following. Note that though we have been using  $\mathbf{f}_t = [f_1, \dots, f_t]^\top$  so far, as we start adding and pruning bases  $\mathbf{f}_t$  no longer maintains that structure.

### 2.2.1. How to optimally remove a basis

After the inclusion of several observations, we are left with a posterior of the form  $p(\mathbf{f}_t, f_{t+1} | \mathcal{D}_{t+1}) = \mathcal{N}(\mathbf{f}_{t+1} | \boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1})$ . Without lack of generality we will assume that we want to remove the basis corresponding to  $f_{t+1}$ . To this end we can approximate  $p(\mathbf{f}_t, f_{t+1} | \mathcal{D}_{t+1})$  with the product of  $p(f_{t+1} | \mathbf{f}_t)$  (independent of data) times some distribution  $q(\mathbf{f}_t)$  which does not depend on the removed basis. The optimal form of  $q(\mathbf{f}_t)$  is then derived by minimizing the Kullback-Leibler (KL) divergence between the exact and approximate posteriors  $\text{KL}(p(\mathbf{f}_{t+1} | \mathcal{D}_{t+1}) || p(f_{t+1} | \mathbf{f}_t) q(\mathbf{f}_t))$ , which yields  $q(\mathbf{f}_t) = p(\mathbf{f}_t | \mathcal{D}_{t+1})$ . Unsurprisingly, the optimal way to remove a basis from the posterior within this Bayesian framework is simply to marginalize it out.

Marginalizing out a variable in a joint Gaussian distribution implies removing the corresponding row and column from its mean vector and covariance matrix, so the removal equations become  $\boldsymbol{\mu}_{t+1} \leftarrow [\boldsymbol{\mu}_{t+1}]_{-i}$  and  $\boldsymbol{\Sigma}_{t+1} \leftarrow [\boldsymbol{\Sigma}_{t+1}]_{-i,-i}$ , where the notation  $[\cdot]_{-i}$  refers to a vector in which the  $i$ -th row has been removed, and  $[\cdot]_{-i,-i}$  to matrix in which the  $i$ -th row and column have been

removed. Following this notation, we will use  $[\cdot]_{-i,i}$  to refer to the  $i$ -th column of a matrix, excluding the element in the  $i$ -th row.

The  $i$ -th basis can be removed from  $\mathbf{Q}_{t+1}$  using

$$\mathbf{Q}_{t+1} \leftarrow [\mathbf{Q}_{t+1}]_{-i,-i} - \frac{[\mathbf{Q}_{t+1}]_{-i,i} [\mathbf{Q}_{t+1}]_{-i,i}^\top}{[\mathbf{Q}_{t+1}]_{i,i}}. \quad (13)$$

Additionally, it can be proved that whenever  $\gamma_{t+1}^2$  is (numerically) zero, the above KL divergence is also zero, i.e., discarding the last basis produces no information loss. In such cases, after updating the posterior with Eq. (7), we can immediately prune the last row and column without incurring any information loss. This is sometimes known in the literature as *reduced update* and is specially useful since update (8), which would be ill-defined, is avoided.

### 2.2.2. Selecting which basis should be removed

We will now address the selection of a basis for removal. Probably, the most sensible option would be to remove the basis  $i$  that minimizes the KL divergence between the exact and approximate posteriors  $\text{KL}(p(\mathbf{f}_{t+1} | \mathcal{D}_{t+1}) || p([\mathbf{f}_{t+1}]_i | [\mathbf{f}_{t+1}]_{-i}) p([\mathbf{f}_{t+1}]_{-i} | \mathcal{D}_{t+1}))$ , i.e., a measure of the information loss due to basis removal. This is exactly the same cost function used in the previous section. Unfortunately, this is computationally too expensive. A more practical option is to minimize the squared error (SE) induced by the approximation in the mean of the approximate posterior. In our experiments, this simpler cost function delivered almost identical performance compared to the KL-based criterion.

The SE can be computed by subtracting the means of the exact and approximate distributions after removing the  $i$ -th basis, which is  $[0, \dots, [\mathbf{Q}_{t+1} \boldsymbol{\mu}_{t+1}]_i / [\mathbf{Q}_{t+1}]_{i,i}, \dots, 0]^\top$ ; and then computing the Euclidean norm of this vector. Observe that only the posterior mean corresponding to the removed basis is distorted. With this result, we can easily check the SE of all basis in  $\mathcal{O}(M^2)$  time and flag for removal the basis that incurs in the least error. This is a well-known pruning criterion, used for instance in [10, 12].

Thus, flagging a basis for removal and then effectively removing it from the posterior are operations that have the same cost as including new bases, rendering each complete KRLS iteration  $\mathcal{O}(M^2)$ .

## 3. THE KRLS TRACKER

In the previous section we offered a Bayesian interpretation of fixed-budget KRLS. Data was assumed to be stationary, i.e.,  $f(\mathbf{x})$  did not change over time. However, in a time-varying scenario, only recent samples have relevant information, whereas the information contained in older samples is actually misleading. In such a case, we would be interested in having a KRLS *tracker* that is able to forget past information and track changes in the target latent function.

In this section we are interested in developing a forgetting strategy and assess its effect throughout the whole input space, so we will work with complete GPs. We briefly remind the reader the GP notation: GPs are stochastic processes that are defined through a mean function  $m(\mathbf{x})$  and a covariance function  $c(\mathbf{x}, \mathbf{x}')$ . To denote that  $f(\mathbf{x})$  is a stochastic function drawn from a GP, we will use the notation  $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), c(\mathbf{x}, \mathbf{x}'))$ . Loosely speaking, one can think of a GP as a Gaussian distribution over an infinite set of points (evaluating  $f(\mathbf{x})$  at every possible  $\mathbf{x}$  and thus building an infinitely long vector), with a corresponding infinitely long mean (given by the evaluation of  $m(\mathbf{x})$  at every possible point) and an infinitely big covariance matrix (given by evaluating  $c(\mathbf{x}, \mathbf{x}')$  at each possible pair of points). A complete background on GPs can be found in [11].

### 3.1. A general forgetting setup

After several inclusion-deletion steps, all information available up to time  $t$  has (approximately) been stored in the posterior density over the dictionary bases  $\mathbf{f}_t|\mathcal{D}_t \sim \mathcal{N}(\mathbf{f}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ . Inserting this  $p(\mathbf{f}_t|\mathcal{D}_t)$  in Eq. (2) and solving the integral, we can obtain the implied posterior GP over the whole input space,

$$f(\mathbf{x})|\mathcal{D}_t \sim \mathcal{GP}(\mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \boldsymbol{\mu}_t, k(\mathbf{x}, \mathbf{x}') + \mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t (\boldsymbol{\Sigma}_t - \mathbf{K}_t) \mathbf{Q}_t \mathbf{k}_t(\mathbf{x}')), \quad (14)$$

where  $\mathbf{k}_t(\mathbf{x})$  is the vector of covariances between  $\mathbf{x}$  and all the bases in the dictionary at time  $t$ . Observe that (14) has the same form as the prediction equation (5), but extended to the whole input space.

In order to make KRLS able to adapt to non-stationary environments, we should make it able to “forget” past samples, i.e., to intentionally force the posterior  $p(f(\mathbf{x})|\mathcal{D}_t)$  to lose some information. A very general approach to this is to linearly combine  $f(\mathbf{x})|\mathcal{D}_t$  with another independent GP  $n(\mathbf{x})$  that holds no information about data. Since this new posterior after forgetting will be a linear combination of two GPs, it will also be a GP, and we will denote it as

$$\check{f}(\mathbf{x})|\mathcal{D}_t = \alpha f(\mathbf{x})|\mathcal{D}_t + \beta n(\mathbf{x}), \quad (15)$$

where  $\alpha, \beta > 0$  are used to control the trade-off between the informative GP  $f(\mathbf{x})|\mathcal{D}_t$  and the uninformative “forgetting noise”  $n(\mathbf{x})$ .

The posterior GP after forgetting,  $p(\check{f}(\mathbf{x})|\mathcal{D}_t)$ , should be expressible in terms of a distribution over the latent points in the dictionary (to avoid a budget increase). We will refer to this distribution as  $\mathcal{N}(\check{\boldsymbol{\mu}}_t, \check{\boldsymbol{\Sigma}}_t)$ . Using Eq. (2) again, the posterior after forgetting in terms of  $\check{\boldsymbol{\mu}}_t$  and  $\check{\boldsymbol{\Sigma}}_t$  is

$$\check{f}(\mathbf{x})|\mathcal{D}_t \sim \mathcal{GP}(\mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \check{\boldsymbol{\mu}}_t, k(\mathbf{x}, \mathbf{x}') + \mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t (\check{\boldsymbol{\Sigma}}_t - \mathbf{K}_t) \mathbf{Q}_t \mathbf{k}_t(\mathbf{x}')). \quad (16)$$

Different definitions for  $\alpha, \beta$  and  $n(\mathbf{x})$  will result in different types of forgetting. One reasonable approach is discussed next.

### 3.2. Back-to-the-prior forgetting

First, we select the form of the GP  $n(\mathbf{x})$  which acts as noise. This GP holds no information about the data and it is independent of  $f(\mathbf{x})|\mathcal{D}_t$ . Assume for a moment that we want to forget all past data. In this case we must set  $\alpha = 0$  to completely remove the informative GP. Then, our posterior GP would be  $\beta n(\mathbf{x})$ . The distribution of the posterior when no data has been observed, should, by definition, be equal to the prior. Therefore  $n(\mathbf{x})$  must be a scaled version of the GP prior. Without lack of generality, we can choose this scale to be 1, so that the noise GP becomes  $n(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$ . Obviously, with this choice, setting  $\alpha = 0$  should imply  $\beta = 1$ , which as we will see later, is the case. Observe that  $n(\mathbf{x})$  corresponds to colored noise, using the same coloring as the prior.

Once  $n(\mathbf{x})$  has been defined, the distribution of  $\check{f}(\mathbf{x})|\mathcal{D}_t$  can be obtained from its definition (15). Since both GPs are independent, their linear combination is distributed as

$$\check{f}(\mathbf{x})|\mathcal{D}_t \sim \mathcal{GP}(\alpha \mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \boldsymbol{\mu}_t, (\alpha^2 + \beta^2)k(\mathbf{x}, \mathbf{x}') + \mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t (\alpha^2 \boldsymbol{\Sigma}_t - \alpha^2 \mathbf{K}_t) \mathbf{Q}_t \mathbf{k}_t(\mathbf{x}')). \quad (17)$$

Comparing (16) and (17) and identifying terms, we obtain

$$\check{\boldsymbol{\mu}}_t = \alpha \boldsymbol{\mu}_t; \quad \check{\boldsymbol{\Sigma}}_t = \alpha^2 \boldsymbol{\Sigma}_t + (1 - \alpha^2) \mathbf{K}_t; \quad \alpha^2 + \beta^2 = 1$$

which provides the relationship between the posterior distribution before and after forgetting occurs. Forgetting depends on a single positive parameter,  $\alpha$ , and one can find the corresponding  $\beta = \sqrt{1 - \alpha^2}$ . This latter equation implies that  $\alpha$  cannot be bigger than 1. Its values are therefore in the range from 0 (all past data is forgotten and we arrive back at the prior) to 1 (no forgetting occurs and we are left with the original, unmodified posterior). Reparameterizing  $\alpha^2 = \lambda$  for convenience, the forgetting updates are finally:

$$\boldsymbol{\Sigma}_t \leftarrow \lambda \boldsymbol{\Sigma}_t + (1 - \lambda) \mathbf{K}_t \quad (18a)$$

$$\boldsymbol{\mu}_t \leftarrow \sqrt{\lambda} \boldsymbol{\mu}_t \quad (18b)$$

where we denote  $\lambda \in (0, 1]$  as the *forgetting factor*. The smaller the value of  $\lambda$ , the faster the algorithm can track changes (and the less it is able to learn, since information is quickly discarded). Usually, only values in the  $[0.95, 1]$  range are sensible. We call this technique “back-to-the-prior” forgetting.

### 3.3. The KRLS-T algorithm

The KRLS Tracker (KRLS-T) algorithm is summarized in Algorithm 1. A Matlab implementation of KRLS-T can be obtained at <http://www.tsc.uc3m.es/~miguel>

---

#### Algorithm 1 Kernel Recursive Least-Squares Tracker

---

**Parameters:** Forgetting factor  $\lambda$ , regularization  $\sigma_n^2$ , kernel function  $k(\mathbf{x}, \mathbf{x}')$ , including its parameters  $\boldsymbol{\theta}$ , budget  $M$ .

Observe  $(\mathbf{x}_1, y_1)$ .

Initialize  $\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \mathbf{Q}_1$  as per Eq. (11).

Add  $\mathbf{x}_1$  to the dictionary.

**for** time instant  $t = 1, 2, \dots$  **do**

Forget using (18).

Observe new input  $\mathbf{x}_{t+1}$ .

Compute  $\mathbf{k}_{t+1}$  (the kernel between  $\mathbf{x}_{t+1}$  and dictionary bases),

$k_{t+1} = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})$ ,  $\mathbf{q}_{t+1} = \mathbf{Q}_t \mathbf{k}_{t+1}$ ,  $\mathbf{h}_{t+1} = \boldsymbol{\Sigma}_t \mathbf{q}_{t+1}$ .

Compute projection uncertainty  $\gamma_{t+1}^2 = k_{t+1} - \mathbf{k}_{t+1}^\top \mathbf{q}_{t+1}$ .

Compute noiseless pred. var.  $\hat{\sigma}_{f_{t+1}}^2 = \gamma_{t+1}^2 + \mathbf{q}_{t+1}^\top \mathbf{h}_{t+1}$ .

Output predictive mean  $\hat{y}_{t+1} = \mathbf{q}_{t+1}^\top \boldsymbol{\mu}_t$ .

Output predictive variance  $\hat{\sigma}_{y_{t+1}}^2 = \sigma_n^2 + \hat{\sigma}_{f_{t+1}}^2$ .

Observe actual output  $y_{t+1}$ .

Compute  $\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}$  as per Eq. (7).

**if**  $\gamma_{t+1}^2 < \epsilon$  (for some  $\epsilon > 0$  close to machine precision) **then**

Remove basis  $t + 1$  (introduces no error):

$\boldsymbol{\mu}_{t+1} \leftarrow [\boldsymbol{\mu}_{t+1}]_{-(t+1)}$ ,  $\boldsymbol{\Sigma}_{t+1} \leftarrow [\boldsymbol{\Sigma}_{t+1}]_{-(t+1), -(t+1)}$ .

**else**

Compute  $\mathbf{Q}_{t+1}$  as per Eq. (8).

Add basis  $\mathbf{x}_{t+1}$  to the dictionary.

**if** Number of bases in the dictionary  $> M$  **then**

Compute squared errors  $([\mathbf{Q}_{t+1} \boldsymbol{\mu}_{t+1}]_i / [\mathbf{Q}_{t+1}]_{i,i})^2$ .

Remove basis  $i$  that introduces minimum error:

$\boldsymbol{\mu}_{t+1} \leftarrow [\boldsymbol{\mu}_{t+1}]_{-i}$ ,  $\boldsymbol{\Sigma}_{t+1} \leftarrow [\boldsymbol{\Sigma}_{t+1}]_{-i, -i}$ .

Remove basis  $i$  from  $\mathbf{Q}_{t+1}$  as per Eq. (13).

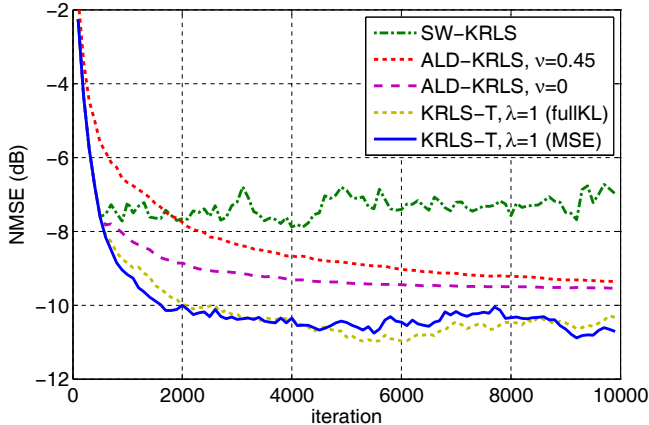
Remove basis  $\mathbf{x}_i$  from the dictionary.

**end if**

**end if**

**end for**

---



**Fig. 1.** NMSE performance of KRLS algorithms on the KIN40K regression problem, each using a dictionary of  $M = 500$  bases.

#### 4. NUMERICAL EXPERIMENTS

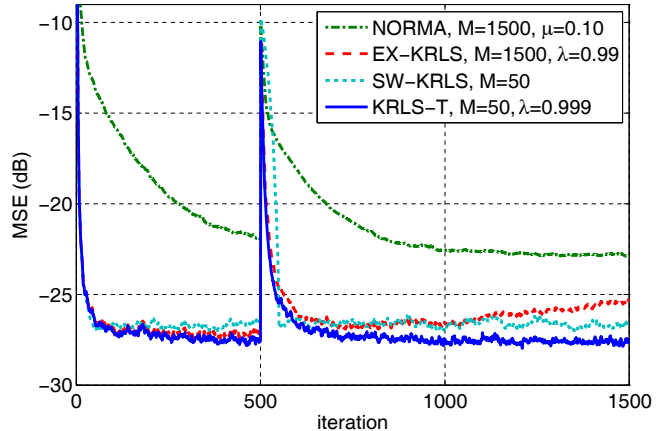
We proceed to demonstrate the performance of the KRLS-T algorithm, first on a stationary benchmark and then on a tracking problem with non-stationary data. We include several other relevant algorithms in the comparison, in particular Approximate Linear Dependency KRLS (ALD-KRLS) and Sliding-Window KRLS (SW-KRLS). The ALD-KRLS algorithm iteratively constructs the solution to a stationary batch regression problem, and hence it is not suitable for tracking. To slow down dictionary growth it uses an approximate linear dependency criterion (see [4] for details). A notable characteristic of ALD-KRLS is that it does not intrinsically handle forgetting or regularization, but rather achieves its regularization by constructing a sparse basis. The SW-KRLS algorithm [7, 13] is, to the best of our knowledge, the only relevant KRLS algorithm capable of tracking, apart from the proposed algorithm. Some additional algorithms will be mentioned briefly throughout the experiments.

##### 4.1. Online regression on stationary data

In the first experiment we apply the algorithms to perform online regression of the KIN40K data set<sup>1</sup>. This set contains 40,000 examples, each consisting of an 8-dimensional input vector and a scalar output, representing the forward kinematics of an 8-link all-revolute robot arm. We randomly selected 10,000 data points for training and used the remaining 30,000 points for testing the regression.

An anisotropic Gaussian kernel was used, in which the hyperparameters were determined off-line by standard GP regression. In particular, the noise-to-signal ratio (regularization) was  $\sigma_n^2/\sigma_0^2 = 0.0021$ . The first algorithm was SW-KRLS with a window of 500 data points. The second algorithm was ALD-KRLS with sensitivity  $\nu = 0.45$ . For this parameter value, the final dictionary contained exactly 500 bases. We also ran ALD-KRLS with  $\nu = 0$  and stopped its dictionary expansion once it contained 500 bases. While after this point the dictionary is left unchanged, ALD-KRLS continues adaptation by performing reduced updates of its other parameters. The last algorithm is the proposed KRLS-T algorithm, with a dictionary size of 500 bases and  $\lambda = 1$ . To prune the dictionary, we used the

<sup>1</sup>From the DELVE archive <http://www.cs.toronto.edu/~delve/data/datasets.html>



**Fig. 2.** MSE performance of different tracking algorithms on a communications channel that shows an abrupt change at iteration 500.

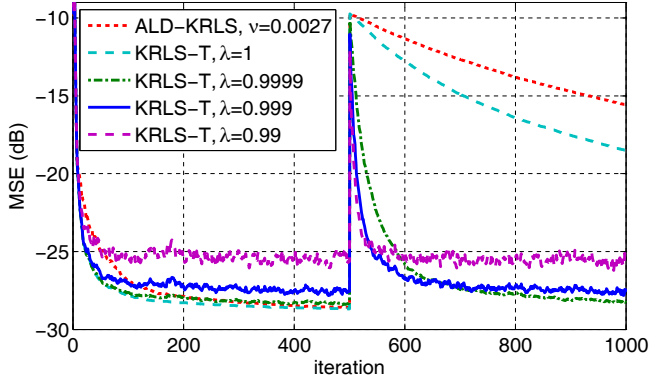
slower criterion that minimizes KL-divergence (“fullKL”) in one test and the simpler MSE-based criterion in another test.

Each algorithm performed a single run over the data. The performance was measured by calculating the normalized mean-square error (NMSE) on the test set, at different points throughout the training run. The results are displayed in Fig. 1. During the first 500 iterations, four of the algorithms show identical performance, since they accept every observed data point into their dictionaries. The fifth algorithm, ALD-KRLS with  $\nu = 0.45$ , has a slower dictionary growth and an initially larger error. From iteration 501 onwards, SW-KRLS maintains its performance, since it performs regression only on the 500 most recent samples. ALD-KRLS selects the most relevant bases (as a growing set), and therefore it converges to a better NMSE. While its sensitivity level  $\nu = 0.45$  results in a slower convergence, it achieves similar results as  $\nu = 0$ . The KRLS-T algorithm outperforms all others, since it is able to properly weight all samples and to trade weaker bases in the dictionary for more relevant ones during the entire training process. Interestingly, it obtains similar results with the simple MSE-based pruning criterion and the computationally more expensive KL-based criterion.

##### 4.2. Adaptive identification of a time-varying channel

We now evaluate the tracking capabilities of the different algorithms. For this experiment we use the setup described in [7]. Specifically, we consider the problem of online identification of a communication channel in which an abrupt change (switch) is triggered at some point. Here, a signal  $x_t \in \mathcal{N}(0, 1)$  is fed into a nonlinear channel that consists of a linear finite impulse response (FIR) channel followed by the nonlinearity  $y = \tanh(z)$ , where  $z$  is the output of the linear channel. During the first 500 iterations the impulse response of the linear channel is chosen as  $\mathbf{h}_1 = [1, 0.0668, -0.4764, 0.8070]$ , and at iteration 501 it is switched to  $\mathbf{h}_2 = [1, -0.4326, -0.6656, 0.7153]$ . Finally, 20dB of Gaussian white noise is added to the channel output.

We perform an online identification experiment, in which the algorithms are given one input datum (with a time-embedding of 4 taps) and one output sample at each time instant. At each step, the MSE performance is measured on a set of 100 data points that are generated with the current channel model. In this comparison we include results for Naive Online  $R_{reg}$  Minimization Algorithm



**Fig. 3.** MSE performance of KRLS-T and ALD-KRLS on a communications channel that shows an abrupt change at iteration 500.

(NORMA), which is a kernel-based implementation of leaky LMS [2], and extended KRLS (EX-KRLS) from [9], which is a straightforward kernelized version of classic extended RLS [3].

An RBF kernel  $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/2\ell)$  is used in all algorithms, with a length-scale  $\ell = 1$ . The regularization is set to match the true value of the noise-to-signal ratio, 0.01. Regarding memory, SW-KRLS and KRLS-T are given a dictionary size of  $M = 50$ . NORMA and EX-KRLS are not imposed any memory limit (i.e.  $M = 1500$ ), given that the former would perform very weakly with only 50 bases (being LMS-based), and the latter can only be applied with an evergrowing dictionary when tracking. The adaptation rates are chosen as follows: NORMA uses learning rate  $\eta = 0.1$ ; EX-KRLS has parameters  $\alpha = 0.999$ ,  $q = 0.1$  and forgetting factor  $\lambda = 0.99$ ; and KRLS-T uses  $\lambda = 0.999$ . Note that the same value of  $\lambda$  does not necessarily correspond to the same convergence rate in different algorithms.

The identification results, averaged out over 25 simulations, can be found in Fig. 2. EX-KRLS initially obtains acceptable tracking results, but later starts to diverge due to numerical problems. SW-KRLS obtains very reasonable results, but, since it gives the same importance to all samples in its window, its speed of convergence is limited by its window size  $M$ . The best performance, both in terms of convergence rate and final MSE, is obtained by the proposed KRLS-T algorithm, which gives more importance to more recent data. The influence of its forgetting factor is illustrated in Fig. 3. In the limiting case  $\lambda = 1$ , KRLS-T does not perform tracking, and then it is fair to compare its performance to ALD-KRLS (which is not a tracker). We applied ALD-KRLS with  $\nu = 0.003$ , which leads to a final dictionary of  $M = 50$  bases (while we verified also that the performance was hardly affected by changing  $\nu$ ). Similar to the previous example, KRLS-T obtains superior results.

## 5. CONCLUSIONS

In this paper we have introduced a Bayesian framework that unifies existing KRLS theory and provides additional insight by explicitly handling uncertainty. This approach allows to define the concept of “forgetting” in a natural manner in KRLS, and it rigorously introduces regularization into KRLS. Finally, we have combined these ideas into a concrete algorithm, the KRLS Tracker, which works with fixed memory and computational requirements per time step, and allows for simple, practical implementation and usage.

We included different numerical experiments that show how the

proposed algorithm outperforms existing online kernel methods not only in the non-stationary scenarios for which it was designed, but also in stationary scenarios (by setting its forgetting factor to  $\lambda = 1$ ) due to its more rigorous approach to regularization.

The described Bayesian framework opens the door to many interesting future research lines: New forgetting strategies can be developed using the general setup of Section 3.1. These strategies can be combined with a linear kernel to produce new linear adaptive filtering algorithms and to gain insight into existing ones. For instance, the KRLS Tracker with forgetting updates  $\Sigma_t \leftarrow \Sigma_t/\lambda$ ,  $\mu_t \leftarrow \mu_t$  yields exactly the exponentially weighted RLS when a linear kernel is used. Furthermore, it would be interesting to study how the proposed KRLS tracker algorithm can be extended to a full kernel Kalman filter.

## 6. REFERENCES

- [1] Bernhard Schölkopf and Alexander J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, The MIT Press, December 2001.
- [2] Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson, “Online learning with kernels,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, Aug. 2004.
- [3] Simon Haykin, *Adaptive Filter Theory (4th Edition)*, Prentice Hall, September 2001.
- [4] Yaakov Engel, Shie Mannor, and Ron Meir, “The kernel recursive least squares algorithm,” *IEEE Transactions on Signal Processing*, , no. 8, pp. 2275–2285, Aug. 2004.
- [5] Weifeng Liu, José C. Príncipe, and Simon Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*, Wiley, 2010.
- [6] Steven Van Vaerenbergh, Ignacio Santamaría, Weifeng Liu, and José C. Príncipe, “Fixed-budget kernel recursive least-squares,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Dallas, USA, April 2010.
- [7] Steven Van Vaerenbergh, Javier Vía, and Ignacio Santamaría, “A sliding-window kernel RLS algorithm and its application to nonlinear channel identification,” in *IEEE Int. Conf. on Acoustics, Speech, and Sig. Proc.*, Toulouse, France, May 2006.
- [8] Konstantinos Slavakis and Sergios Theodoridis, “Sliding window generalized kernel affine projection algorithm using projection mappings,” *EURASIP Journal on Advances in Signal Processing*, vol. 16, 2008.
- [9] Weifeng Liu, Il Park, Yiwen Wang, and José C. Príncipe, “Extended kernel recursive least squares algorithm,” *IEEE Trans. on Sig. Proc.*, vol. 57, no. 10, pp. 3801–3814, Oct. 2009.
- [10] Lehel Csató and Manfred Opper, “Sparse representation for gaussian process models,” in *Advances in neural information processing systems 13*, pp. 444–450. MIT Press, 2001.
- [11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, MIT Press, 2006.
- [12] B.J. De Kruif and T.J.A. De Vries, “Pruning error minimization in least squares support vector machines,” *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 696–702, 2003.
- [13] Steven Van Vaerenbergh, J. Vía, and Ignacio Santamaría, “Nonlinear system identification using a new sliding-window kernel RLS algorithm,” *Journal of Communications*, vol. 2, no. 3, pp. 1–8, May 2007.