Online Regression Using Reproducing Kernel Hilbert Spaces

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Abhishek Ojha

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2019

Supervisor: Vivek S. Borkar

Certificate

This is to certify that this dissertation entitled Online Regression Using Reproducing Kernel Hilbert Spaces towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Abhishek Ojha at Indian Institute of Science Education and Research under the supervision of Vivek S. Borkar, Chair Professor, Department of Electrical Engineering, IIT Bombay , during the academic year 2018-2019.

Vivek S. Borkar

Committee:

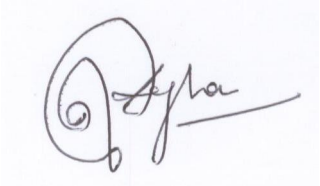
Vivek S. Borkar

Anup Biswas

This thesis is dedicated to my grandfather Shri Nagendra Nath Ojha, without whom this journey would not have been possible.

Declaration

I hereby declare that the matter embodied in the report entitled Online Regression Using Reproducing Kernel Hilbert Spaces are the results of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research, Pune, under the supervision of Prof. Vivek S. Borkar , Department of Electrical Engineering, IIT Bombay and the same has not been submitted elsewhere for any other degree.



Abhishek Ojha

Acknowledgments

I am grateful to my supervisor Prof. Vivek Borkar for introducing me to this field of research and his immense support throughout the project. I would like to thank Prof. Anup Biswas, Prof. Uttara Naik-Nimbalkar, and Prof. Anindya Goswami for their invaluable discussions whenever I needed clarifications. I would also like to thank Infosys Foundation for covering my tuition fees for the FALL 2017, SPRING 2018, and SPRING 2019 semesters.

Abstract

Suppose we have $X \subset \mathbb{R}^2$ and there exists an unknown function $F(\boldsymbol{x}) : X \to \mathbb{R}$. We will consider the Unnikrishnan -Vetterli problem [1] in which a vehicle moves on X making observations (input-output pairs) $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$ (where y_i is a noisy version of $F(\boldsymbol{x_i})$). The task is to maintain a running estimate for F using the observations. In learning literature, such a task is referred to as regression. In this thesis, we have surveyed regression methods suitable for this scenario when data arrive sequentially. The methods that have been included in this thesis consider the Reproducing Kernel Hilbert Spaces (RKHS) as their hypothesis space. Towards the end, we propose an improvement and present some results without any mathematical proofs.

Contents

Introduction

The Estimation Problem

Let us consider a random field (assume stationary for now) $F(x, \omega)$, $x \in X$ ($X = [0,1]^2 \subset \mathbb{R}^2$), sampled at $x_1, x_2, \ldots, x_t$ ($t \in \mathbb{N}$) with noisy observations given by

$$y_i = F(x_i, \omega) + \xi_i,$$

where $\xi_i, 1 \leq i \leq t$ are i.i.d zero mean random variables with $|\xi_t| \leq \eta < \infty, \ \forall \ t$. If we use a Reproducing Kernel Hilbert Space (RKHS),$\mathcal{H}_K$ (discussed in the second chapter) as our hypothesis space and we solve the following optimisation problem:

$$\min_{f \in \mathcal{H}_K} \frac{1}{t} \sum_{i=1}^{t} (f(x_i) - y_i)^2 + \gamma \|f\|^2_{\mathcal{H}_k}, \qquad \text{for some } \gamma > 0,$$

then from the representer theorem ([2], [3],[4]), we will have an estimate for $F(x, \omega)$ as:

$$\hat{F}(x, \omega) = \sum_{i=1}^{t} \alpha_i k(x_i, x), \text{where } \boldsymbol{\alpha} = (\gamma t \ I_t + \boldsymbol{K_t})^{-1} \mathbf{y}. \tag{1}$$

Next, we will consider the Unnikrishnan -Vetterli problem [1] in which a vehicle moves on X making observations (input-output pairs) $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$. We want to maintain a running estimate of $\hat{F}(x, \omega)$. Note that $\boldsymbol{\alpha}$ can be updated in online manner since the inverse can be updated ($\mathcal{O}(t^2)$) at each step using matrix-inversion lemma [5]. Even quadratic time complexity may turn out to be computationally heavy when t becomes quite large, so we will visit some methods which avoid computing the inverse for $\boldsymbol{\alpha}$ and use gradient descent which results in linear time complexity ($\mathcal{O}(t)$).

As time increases, the number of terms in expression (1) of the estimate increases and hence we will visit some methods to control the number of terms which are being included in the expression for the estimate. At each time, the estimate ($\hat{F}(x, \omega)$) will be expressed in terms of a few selected inputs which is referred to as dictionary. The growing and pruning strategies for this dictionary will be discussed.

Notations

In this section the notations are introduced which will be followed in this report unless stated otherwise.

$\{(\mathbf{x_i}, y_i)\}_{i=1}^{t}$ is the sequence of input, observed output pair till time t. D_t denotes the dictionary which contains m_t ($\leq t$) elements at time t. $D_t := \{\boldsymbol{x}_{\omega_1}, \ldots, \boldsymbol{x}_{\omega_{m_t}}\}$.

Most of the methods in the literature work with the Gaussian kernel which we will denote by $k : X \times X \to \mathbb{R}$ and $k(\boldsymbol{x_1}, \boldsymbol{x_2}) := exp(\frac{-\|\boldsymbol{x_1} - \boldsymbol{x_2}\|^2}{2\sigma^2})$.

$\mathbf{K}_t$ is the kernel matrix on input points, i.e., $[\mathbf{K}_t]_{ij} := k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ whereas $\mathbf{K}_{\omega_\mathbf{t}}$ is the kernel matrix on dictionary points, i.e., $[\mathbf{K}_{\omega_\mathbf{t}}]_{ij} := k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x}_{\omega_j})$.

Two vectors $\boldsymbol{k}_{\boldsymbol{\omega_t}}(\boldsymbol{x}) := [k(\boldsymbol{x}_{\omega_1}, \boldsymbol{x}), k(\boldsymbol{x}_{\omega_2}, \boldsymbol{x}), \ldots, k(\boldsymbol{x}_{\omega_{m_t}}, \boldsymbol{x})]^T$ and

$\boldsymbol{k_t}(\boldsymbol{x}) := [k(\boldsymbol{x_1}, \boldsymbol{x}), \ldots, k(\boldsymbol{x_t}, \boldsymbol{x})]^T$ will be helpful to make the equations concise. These two vectors have been recalled wherever they are used to avoid any confusion.

Φ denotes the feature map (discussed in chapter 2) and $\Phi(x)$ is the feature vector for $x \in X$. $\boldsymbol{\Phi_t} := [\Phi(x_1), \ldots, \Phi(x_t)]$ and $\tilde{\boldsymbol{\Phi}}_\mathbf{t} := [\Phi(\boldsymbol{x}_{\omega_1}), \ldots, \Phi(\boldsymbol{x}_{\omega_{m_t}})]$ are the matrices in which columns are feature vectors of input points and dictionary elements respectively.

And finally, $\boldsymbol{\alpha_t}$ is the coefficient vector at time t (for example see 1) and I_t is the $t \times t$ identity matrix.

The above-mentioned notations appear in nearly every method discussed below but a few notations are algorithms specific and hence will be introduced in the description of those particular algorithms.

Organization of this report

The problem and the notations has been introduced in the first and second section respectively of Introduction.
Chapter 1 consists of a review of the theory of RKHS with an example of Mercer Kernel. We will go through the definition of kernel, reproducing kernel, and RKHS and properties of RKHS. We will also visit the non-linear regression using RKHS and will state the Representer theorem and the Universal Approximation theorem to motivate the importance of using RKHSs.
Chapter 2 consists of survey of a few existing methods to solve the online kernel regression problem. Section 3.1 mentions the two approaches towards solving this problem. Section 3.2 talks about the different growing and pruning strategies for the dictionary. Section 3.3-3.9 are the algorithms with linear time complexity. Quadratic algorithms ([6], [7], [8], [9]) have been omitted due to space constraints. We were trying to tackle this problem using concepts from the theory of compressed sensing ([10], [11], [12], [13], [14],[15], [16]) but that turned out to be computationally expensive. The compressed sensing theory has been omitted due to space constraints.
Note that the problem of fitting a function based on sequential data (online kernel regression) has been studied extensively. Most of the methods developed were meant for studying the time series data. In this report we are trying to use the same techniques to provide an alternative approach to the Unnikrishnan-Vetterli problem [1]. In chapter 3, we present a local approach for updating the coefficient vector and compare it with existing algorithms (without mathematical proof).

Chapter 1

Theory of Reproducing Kernel Hilbert Spaces (RKHS)

1.1 Kernel, Reproducing kernel and RKHS

In this section, we first go through the definition of a kernel and its equivalence to positive semi-definite functions and then we visit the definitions related to a reproducing kernel, and RKHS. This section closely follows the explanations provided in [17] and [18].

Definition 1.1.1. (**Kernel**) Let X be a non-empty set. A function $k : X \times X \to \mathbb{R}$ is called kernel if $\exists$ a $\mathbb{R}$- Hilbert space H (inner product has range $\mathbb{R}$) and a map $\Phi : X \to H$ such that $\forall x, y \in X$, we have

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle_H,$$

Φ is called feature map and H is called feature space.

Note that a kernel can have multiple feature map and feature space pairs. For example, take $X = \mathbb{R}$ and $k(x, y) := xy \ \forall x, y \in \mathbb{R}$. Then this k is a kernel for $H = \mathbb{R}, \Phi(x) = x$ as well as for $H = \mathbb{R}^2, \Phi(x) = \left(\frac{x}{\sqrt{2}}, \frac{x}{\sqrt{2}}\right)$.
Given a function it maybe difficult to come up with its corresponding feature space and feature map to check whether it is a kernel or not. So, an equivalence between definition of kernels and definition of symmetric positive definite functions can be obtained. The definition of positive semi-definite functions has been revisited below.

Definition 1.1.2. A function $k : X \times X \to \mathbb{R}$ is said to be positive semi-definite if, $\forall n \in \mathbb{N}, \alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and all $x_1, \ldots, x_n \in X$, we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \geq 0.$$

Finally, k is said to be symmetric if $k(x, y) = k(y, x)\ \forall x, y \in X$

To check whether a given function is positive-semi definite is relatively easy since one has to check the positive semi-definiteness of gram matrix K ($K_{ij} = k(x_i, x_j)$). Now, the next theorem provides an equivalence between kernels and positive semi-definite symmetric functions.

Theorem 1.1.1. *A function $k : X \times X \to \mathbb{R}$ is a kernel if and only if k is a symmetric positive semi-definite function.*

After establishing an easier way of characterizing a kernel, we will move to the definitions of reproducing kernel and reproducing kernel Hilbert spaces.

Definition 1.1.3. Let X be a non-empty set and H be Hilbert function space over X.

1. A function $k : X \times X \to \mathbb{R}$ is called a **reproducing kernel** of H if

   (a) $k(\cdot, x) \in H$, $\forall x \in X$ and

   (b) $f(x) = \langle f, k(\cdot, x)\rangle_H$ holds $\forall f \in H$ and $\forall x \in X$

2. The space H is called RKHS over X if $\forall x \in X$, the Dirac functional $\delta_x : H \to \mathbb{R}$ defined by

   $$\delta_x(f) := f(x)$$

   is continuous.

The continuity of Dirac functionals imply that if a sequence of functions f_n converges to f in Hilbert norm then they converge pointwise to f as well.

$$\lim_{n\to\infty} f_n(x) = \lim_{n\to\infty} \delta_x(f_n) = \delta_x(f) = f(x).$$

The next lemma connects the definition of a reproducing kernel with that of a kernel and RKHS. It shows that every reproducing kernel is in fact a kernel. The associated feature space and the feature map will be mentioned in the lemma.

Lemma 1.1.1. *Let H be a Hilbert function space over X that has a reproducing kernel k. Then H is an RKHS and H is also a feature space of k where the feature map $\Phi : X \to H$ is given by*

$$\Phi(x) = k(\cdot, x), \forall x \in X.$$

This feature map is called ***canonical feature map.***

Now, we have established that a Hilbert space with a reproducing kernel is a RKHS and its reproducing kernel is a kernel. The next theorem shows that, conversely, every RKHS has a unique reproducing kernel and hence a kernel associated with it. This theorem also shows that this kernel can be written in terms of orthonormal basis (if it exists) of RKHS.

Theorem 1.1.2. *(**Every RKHS has a unique kernel**) Let H be a RKHS over X. Then, $k : X \times X \to \mathbb{R}$ such that*

$$k(x, y) := \langle \delta_x, \delta_y \rangle_H \quad , x, y \in X,$$

is the only reproducing kernel of H. If $\{e_i\}_{i \in \mathcal{I}}$ is orthonormal basis for H, then

$$k(x, y) = \sum_{i \in \mathcal{I}} e_i(x) e_i(y) \quad \forall x, y \in X.$$

Thus, every RKHS has a unique reproducing kernel and hence kernel associated with it. The next theorem states that given any kernel there exists unique RKHS.

Theorem 1.1.3. *(**Every kernel has a unique RKHS**) Let X be a non-empty set and k be kernel over X with feature space H_o and feature map $\Phi_o : X \to H_o$. Then*

$$H := \left\{ f : X \to \mathbb{R} \;\middle|\; \exists w \in H_o \text{ with } f(x) = \langle w, \Phi_o(x) \rangle_{H_o} \; \forall x \in X \right\},$$

equipped with norm

$$\|f\|_H := \inf_{w \in H_o} \{ \|w\|_{H_o} : w \in H_o \text{ with } f = \langle w, \Phi_o(\cdot) \rangle_{H_o} \}$$

is the only RKHS for which k is a reproducing kernel. Consequently, both the definitions are

independent of choice of H_o and Φ_o. Moreover, the operator $V : H_o \to H$ defined by

$$V(w) := \langle w, \Phi_o(\cdot)\rangle_{H_o} \quad w \in H_o$$

is a metric surjection, i.e. $V(B_{H_o}) = B_H$, where B_{H_o} and B_H are the open unit balls of H_o and H respectively. Finally, the set

$$H_{pre} := \left\{ \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) : \ n \in \mathbb{N}, \ \alpha_1, \ldots, \alpha_n \in \mathbb{R}, \ x_1, \ldots, x_n \in X \right\}$$

is dense in H and for $f := \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) \in H_{pre}$ we have

$$\|f\|_H^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(x_i, x_j).$$

There are three main points worth noting due to Theorem 1.1.3.

- As already mentioned, a given kernel can have multiple feature space, feature map pairs. But no matter which of these pairs is chosen, the RKHS remains same for this given kernel.
- The map V in the above theorem is a metric surjection from any feature space of a kernel to its RKHS which shows that the RKHS of a kernel is the smallest feature space associated with it.
- Every function in the RKHS can be expressed as $f = \langle w, \Phi(\cdot)\rangle_{H_o} = w^T\Phi$ according to the definition. So finding a function satisfying some property (such as minimising some loss function) in RKHS reduces to finding the vector w that corresponds to that property. We will refer to this w as weight vector in the feature space (which can be infinite dimensional).

At this stage it is worth mentioning that such result was first introduced by Moore and Aronszajn [18] who proved the uniqueness and existence of RKHS for a given symmetric-positive semidefinite function (not kernel!) using constructive approach. They used the set H_{pre} (defined above with k being symmetric positive semi-definite function) and showed that its completion will lead us to the RKHS H. Theorem 1.1.3 also mentions that the

completion of H_{pre} is the RKHS H but the difference is that here H_{pre} is constructed when k is a kernel. Since the equivalence between kernel and symmetric positive semi-definite functions was already discussed in Theorem 1.1.1, we can conclude that the two approaches are equivalent. Actually, the proof of Theorem 1.1.1 consists of the steps used in the proof by Moore and Aronszajn. Next, a theorem has been stated upon which Moore-Aronszajn theorem is based. Note that it talks about conditions in which the completion is possible.

Theorem 1.1.4. *Let H_o be any subspace real-valued functions on X ($\mathbb{R}^X$), on which an inner product $<,>_{H_o}$ is defined. In order that there exists a Hilbert space H such that*

1. *$H_o \subset H \subset \mathbb{R}^X$ and the topology defined by the inner product $<,>_{H_o}$ coincides with the topology induced on H_o by H,*
2. *H has a reproducing kernel k,*

   it is necessary and sufficient that

3. *the evaluation functionals are continuous on H_o,*
4. *any Cauchy sequence in H_o converging pointwise to 0 converges also to 0 in the norm sense.*

In the next section, the RKHS related to the Gaussian kernel will be explained with a few interesting properties.

1.2 Gaussian RKHS

In [19], few interesting results have been proved about the Gaussian RKHS using concepts from complex analysis. Three most important points discussed in this work are:
1) Gaussian RKHS over a set in $\mathbb{R}^d$ which has non-empty interior is same as the Gaussian RKHS over whole $\mathbb{R}^d$.
2) Gaussian RKHS doesn't contain non-trivial constant functions.
3) Gaussian RKHS corresponding to a smaller kernel-width is a strict superset of Gaussian RKHS with a larger kernel-width.
The tools used and the proof techniques are interesting. For more details see [19] and [17].

Next, we will look at a special class of kernels known as Mercer kernels. In this case, we can also characterize the RKHS with the help of eigenvectors of operator defined by these kernels.

1.3 Mercer Kernel and associated RKHS

A Mercer kernel [2] ($k : X \times X \to \mathbb{R}$) is a symmetric, positive semi-definite and continuous function i.e. a Mercer kernel is a continuous kernel. Let X be any compact metric space in $\mathbb{R}^N$ (in our case $N = 2$). Let ν be a Borel measure on X and $\mathcal{L}^2_\nu(X)$ be the Hilbert space of square integrable functions. The operator defined by the kernel is $L_k : \mathcal{L}^2_\nu(X) \to \mathcal{C}(X)$ which is given by the integral transform

$$L_k f(x) = \int k(x, y) f(y) d\nu(y).$$

We need to have the feature space and feature map for the Mercer kernel. We will construct its feature space and feature map of its operator. Since Mercer kernel K is symmetric, positive semi-definite and continuous, the operator L_k will be a self-adjoint, positive and compact operator. Then, the spectral theorem will guarantee the existence of an orthonormal basis of $\mathcal{L}^2_\nu(X)$ which consists of eigenvectors of L_k. Let $(\phi_n)_{n\in\mathbf{N}}$ be the set of eigenvectors and $(\lambda_n)_{n\in\mathbf{N}}$ be the corresponding eigenvalues. One should note that the eigenvectors with non-zero eigenvalues are continuous since L_k maps into set of continuous functions on X and $\phi_n = \frac{1}{\lambda_n} L_k(\phi_n)$.
Next, we will visit Mercer's theorem which will help us in defining the feature map for the Mercer kernel.

Theorem 1.3.1. *Let X, K, L_k, ϕ_n and λ_n be defined as above. Then for all $x, y \in X$,*

$$k(x, y) = \sum_{n=1}^{\infty} \lambda_n \phi_n(x) \phi_n(y),$$

where the convergence is absolute and uniform on $X \times X$.

Let us consider $H = l^2$ and a map $\Phi : X \to H$ given by

$$\Phi(x) = (\sqrt{\lambda_n}\phi_n(x))_{n\in\mathbf{N}}. \tag{1.1}$$

With the help of Mercer's theorem it can be shown that this map Φ is well-defined, continuous and satisfies

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle_{l^2} \qquad \textbf{(Kernel Trick)}.$$

This shows that l^2 is the feature space and Φ is the feature map of k. We already know from Theorem 1.1.3 that k will have a unique RKHS (call it $\mathcal{H}_k$) and since we have found a feature space and feature map pair, we can also state that every $f \in \mathcal{H}_k$ can be expressed as:

$$f = \langle \boldsymbol{w}, \Phi \rangle_{l^2}, \quad \text{for some } \boldsymbol{w} \in l^2. \tag{1.2}$$

An example of Mercer kernel is Gaussian kernel which is given by:

$$k_\sigma(x, y) = exp\bigg(- \frac{\|x - y\|_2^2}{2\sigma^2} \bigg) \quad \text{for } x, y \in \mathbb{R}^N.$$

In the rest of this report, the Gaussian kernel will be used and its feature space will be l^2 and feature map will be Φ as defined in equation (1.1). The expression in (1.2) will be written as $f = \boldsymbol{w}^T \Phi$ just for the sake of easier notation.
Now that we have established the properties of RKHS, let us see how these spaces are useful in non-linear regression.

1.4 Regression using RKHS

In a regression problem, we have some input-output pairs (observations). The outputs depend on the inputs via some unknown function. We assume that the unknown function can be best approximated by a function from some set of functions which is called hypothesis space, then we define a risk function and try to find the function from the hypothesis space which minimizes the risk over observations. For example, in linear regression, we assume that underlying unknown function can be well-approximated by a linear function and hence our aim in such case is to find the best (one which minimizes the squared error risk) linear approximator. The statistical learning theory [2] gives us probabilistic bounds over how different the minimizer is from the best performing function from the hypothesis space. We won't discuss that aspect in this report.
We will choose RKHS as our hypothesis space. In the case of a Gaussian kernel it can be shown that [2] its RKHS ($\mathcal{H}_k$) is isomorphic to the subspace of $\mathcal{L}^2_\nu(X)$ which is spanned by

the eigenvectors of L_k with non-zero eigenvalues. This allows us to include many different functions in our hypothesis. We will see that for the risk which we generally choose, its minimizer (or the best function fitting through the observations) turns out to be of a form which is easier to compute and has universal approximation property (stated below). Let us introduce the risk function involved and hence the optimization problem that we will solve. Given input-output pair $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^t$ ($t \in \mathbb{N}$), our aim is to solve following optimization problem (which is called regularized empirical error):

$$\min_{f \in \mathcal{H}_k} \frac{1}{t} \sum_{i=1}^{t} (f(\boldsymbol{x}_i) - y_i)^2 + \gamma \|f\|_{\mathcal{H}_k}^2, \qquad \text{for some } \gamma > 0. \tag{1.3}$$

The second term in the above expression is called regularisation term as it prevents the overfitting through the observations. By the virtue of the representer theorem ([2],[3],[4]), one can show that the minimiser of (1.3) is given by:

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{t} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}), \text{where } \boldsymbol{\alpha_t} = (\gamma t I_t + \boldsymbol{K_t})^{-1} \mathbf{y_t}, \tag{1.4}$$

$\boldsymbol{K_t}$ is the $t \times t$ kernel gram matrix with $(\boldsymbol{K_t})_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{\alpha}_t = [\alpha_1, \alpha_2, \ldots, \alpha_t]^T$, and $\boldsymbol{y}_t = [y_1, y_2, \cdots, y_t]^T$

Note that even if the regularisation term is removed, a function of form (1.4) can be minimiser but it may not be certain that every minimiser will be of this form. At this stage let us see an interesting property that the expansion in (1.4) has. Since, the kernel which we will consider (Gaussian kernel) is also a radial basis function, the following can be proved:

Theorem 1.4.1. *(**Universal Approximation Theorem** [20]) Let* $S_k := \left\{ q : \mathbb{R}^r \to \mathbb{R} \;\middle|\; q(x) = \sum_{i=1}^{M} \alpha_i k(\frac{x - z_i}{\sigma}),\ M \in \mathbb{N}, \sigma > 0,\ \alpha_i \in \mathbb{R},\ z_i \in \mathbb{R}^r,\ i = 1, \ldots, M \right\}$. *Next, suppose that* $k : \mathbb{R}^r \to \mathbb{R}$ *is an integrable, bounded function such that* K *is continuous a.e. and* $\int_{\mathbb{R}^r} k(x)dx \neq 0$. *Then the family* S_k *is dense in* $L^p(\mathbb{R}^r)$ $\forall p \in [1, \infty)$.

Thus, we see that choosing RKHS as our hypothesis space and a proper risk function leads us to a form of minimiser which can approximate a very large set. This is also one of the reasons why RKHS is so widely used. There is an interesting connection between the regularisation (for example in (1.3) and the set S_K in the Theorem 1.4.1. See [21] for details.

Chapter 2

Survey of Existing Methods

2.1 Different Methods of Dictionary Learning

As already mentioned, at time t, we want to express the estimate in terms of a few inputs (dictionary) instead of every input observed till t. Following the notation that has been introduced already, the estimate at time t will be:

$$\hat{F}(x) = \sum_{i=1}^{m_t} \alpha_i k(\boldsymbol{x}_{\omega_i}, x) = (\boldsymbol{k}_{\boldsymbol{\omega_t}}(x))^T \boldsymbol{\alpha_t}.$$

Now, when an observation is made at a new point x_{t+1}, the growing strategy will decide whether to include x_{t+1} in the dictionary. The pruning strategy decides which element(s) to remove from the dictionary. But one should note that the coefficient vector $(\boldsymbol{\alpha}_t)_{m_t \times 1}$ (which corresponds to the dictionary elements) is updated at every time.
Following are some growing strategies (novelty criteria):

1. **Coherence Criterion** [22]: In this strategy one first measures the coherence i.e.,

$$\mu = \max_{\boldsymbol{x}_{\omega_i} \in D_t} k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x}_{t+1}).$$

   If the coherence is below some threshold (μ_o) then $\boldsymbol{x}_{t+1}$ is added to the dictionary. Larger value of threshold allows more elements to be included in the dictionary. Com-

plexity of this criterion is $\mathcal{O}(m_t)$.

2. **ALD Criterion** [6]: In this starategy, one checks whether the feature vector $\Phi(\boldsymbol{x}_{t+1})$ of $\boldsymbol{x}_{t+1}$ is approximately linearly dependent on the feature vectors of the dictionary $\{\Phi(\boldsymbol{x}_{\omega_1}), \ldots \Phi(\boldsymbol{x}_{\omega_{m_t}})\}$. For a given threshold ν_o (> 0), x_{t+1} is added to the dictionary if following criterion is satisfied:

$$\delta_t = \min_{\boldsymbol{w}} \| \sum_{i=1}^{m_t} w_i \Phi(\boldsymbol{x}_{\omega_i}) - \Phi(\boldsymbol{x}_{t+1}) \|^2 \leq \nu_o.$$

Smaller value of ν_o allows fewer elements to be included in the dictionary. Complexity of this criterion is $\mathcal{O}(m_t^2)$.

Following are some pruning strategies:

1. **Sliding Window**: This is a naive way of removing the oldest element and hence we get the picture of sliding window.

2. **Least weight**: In this method, $\boldsymbol{x}_{\omega_i}$ corresponding to least $|\alpha_i|$ is removed. Complexity of this criterion is $\mathcal{O}(m_t)$.

3. **Least a posteriori SE**: In this method $\boldsymbol{x}_{\omega_i}$ corresponding to least $\frac{|\alpha_i|}{[\tilde{\boldsymbol{K}}_t^{-1}]_{ii}}$, where $\tilde{\boldsymbol{K}}_t$ is kernel gram matrix corresponding to dictionary elements (already mentioned in Section 1.2). This criterion is achieved when one tries to find the element which causes the least squared error upon its removal from D_t [7]. Complexity of this criterion is $\mathcal{O}(m_t^2)$.

In some cases (Section 2.7-2.9, Chapter 3), we will see that an additional constraint (for example, weighted l_1-norm on the coefficient vector $(\boldsymbol{\alpha_t})$) can be introduced to the objective function to make the small coefficients go to zero and then elements corresponding to such zero coefficients are removed.

2.2 Functional and Parametric approaches

We get these two strands when we try to tackle this problem using gradient descent methods (also see Section 2.6.3 for more information).

1. **Functional Approach:** We know from (1.2) that every $f \in \mathcal{H}_K$ can be written as $f = \boldsymbol{w}^{\boldsymbol{T}}\Phi$ so the objective (1.3) can be rewritten as (without the regularisation term):

$$\min_{f\in\mathcal{H}_K} \frac{1}{t}\sum_{i=1}^{t}(f(x_i)-y_i)^2 \sim \min_{\boldsymbol{w}\in l^2}\sum_{i=1}^{t}(\boldsymbol{w}^{\boldsymbol{T}}\Phi(\boldsymbol{x_i})-y_i)^2. \tag{2.1}$$

   So, in this approach we find the optimal weight vector from the feature space. It will be clear in the algorithms discussed below that when instantaneous gradient descent (stochastic gradient descent with i=t) is used, only the recent coefficient (the last component of $\boldsymbol{\alpha_t}$) gets updated.

2. **Parametric Approach:** In this approach, the form of minimiser from the representer theorem is directly used in the objective and the optimal coefficient vector is calculated. The objective will be as follows:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^t}\sum_{i=1}^{t}(\boldsymbol{\alpha}^{\boldsymbol{T}}\boldsymbol{k_t}(\boldsymbol{x_i})-y_i)^2, \tag{2.2}$$

   where $\boldsymbol{k_t}(\boldsymbol{x_i}) = [k(\boldsymbol{x_1},\boldsymbol{x_i}),\ldots,k(\boldsymbol{x_t},\boldsymbol{x_i})]^T$.
   It will be shown in the discussion below that if one performs instantaneous gradient descent in this case, every coefficient (whole $\boldsymbol{\alpha_t}$) gets updated.

This methods are like stochastic gradient descent methods in which $i = t$ is always selected i.e., to move according to the gradient of newest entry in the objective. Let us see few linear methods and notice the effect of the choice of objective in different methods.

2.3 Naive Online Regularized Risk Minimization Algorithm (NORMA)

It involves using gradient descent for solving the original regularised empirical error (1.3). In this method, the gradient is calculated in the functional space $\mathcal{H}_k$ only. To that end, note that [23] for $f \in \mathcal{H}_k$,

$$\partial_f \|f\|^2_{\mathcal{H}_k} = 2f \text{ and } \partial_f(f(x_i)) = \partial_f(\langle f, k_{x_i}\rangle) = k_{x_i} = k(x_i, \cdot). \tag{2.3}$$

As already mentioned, the objective considered will be the following (1.3):

$$R_{reg}[f] = \min_{f \in \mathcal{H}_k} \frac{1}{t}\sum_{i=1}^{t}(f(x_i) - y_i)^2 + \gamma\|f\|^2_{\mathcal{H}_k}, \quad \text{for some } \gamma > 0,$$

which will be approximated by the virtue of stochastic gradient descent with $i = t$ using

$$R_{stoch}[f] = \frac{1}{2}(f(x_t) - y_t)^2 + \gamma\|f\|^2_{\mathcal{H}_k}.$$

Hence,

$$f_t = f_{t-1} - \eta\partial_f R_{stoch}[f_{t-1}],$$

where η is the step size. After plugging in the gradient (evaluated using (2.3), we get:

$$f_t = f_{t-1} - \eta(f_{t-1}(x_t) - y_t)k(x_t, \cdot) - \eta\gamma f_{t-1}.$$

Rearranging the terms, we get:

$$f_t = (1 - \eta\gamma)f_{t-1} - \eta(f_{t-1}(x_t) - y_t)k(x_t, \cdot) = (1 - \eta\gamma)f_{t-1} + \eta e_n k(x_t, \cdot),$$

where, $e_n = y_t - f_{t-1}(x_t)$. Going back to the representer theorem,
if $f_{t-1}(\cdot) = \sum_{i=1}^{t-1} \boldsymbol{\alpha_{t-1}}(i)k(x_i, \cdot)$, then

$$f_t(\cdot) = \sum_{i=1}^{t-1}(1 - \eta\gamma)\boldsymbol{\alpha_{t-1}}(i)k(x_i, \cdot) + \eta e_n k(x_t, \cdot) = \sum_{i=1}^{t}\boldsymbol{\alpha_t}(i)k(x_i, \cdot). \tag{2.4}$$

Here $\boldsymbol{\alpha_{t-1}}(i)$ and $\boldsymbol{\alpha_t}(i)$ are the i^{th} components of $\boldsymbol{\alpha_{t-1}}$ and $\boldsymbol{\alpha_t}$ respectively.
In terms of the coefficient vector,

$$\boldsymbol{\alpha_t} = \begin{bmatrix} (1-\eta\gamma)\boldsymbol{\alpha_{t-1}} \\ \eta e_n \end{bmatrix}.$$

Note that this algorithm does not use any growing and pruning strategy, however the older components of the coefficient vector are getting smaller as t grows. So, it is similar to having a sliding window where the older coefficients are shrinking to zero.

2.4 Kernel Least Mean Squares (KLMS) Algorithm

It is similar to NORMA (with $\gamma = 0$) but is derived from the perspective of feature space [9] (gradient is computed w.r.t. feature weight vector) and the derivation is motivated by classical LMS algorithm [9]. In other words, the gradient is evaluated in the feature space instead of the functional space H_k and the gradient step is taken in order to find best weight vector from the feature space. The convergence analysis [24] has been done for this algorithm which also gives a condition on step size required for convergence. It has been shown in the analysis [24] that KLMS has self-regularising property too.

2.4.1 Objective and Updating Procedure

Here the functional form of the objective (2.1) w.r.t. feature weight vector is used. :

$$J_{\boldsymbol{w}} = \frac{1}{2}\sum_{i=1}^{t}(y_i - \boldsymbol{w}^T\Phi(\boldsymbol{x_i}))^2,$$

where $\boldsymbol{w} \in l^2$ (the feature space).
The instantaneous gradient ($i = t$ in stochastic gradient descent) at t is:

$$(\nabla_{\boldsymbol{w}} J)_{stoch}(\boldsymbol{w}) = -\Phi(\boldsymbol{x_t})(y_i - \boldsymbol{w}^T\Phi(\boldsymbol{x_t})).$$

Now, the gradient step will be :

$$\begin{aligned} \boldsymbol{w_t} &= \boldsymbol{w_{t-1}} - \eta(\nabla_{\boldsymbol{w}} J)_{stoch}(\boldsymbol{w_{t-1}}) \\ &= \boldsymbol{w_{t-1}} + \eta\Phi(\boldsymbol{x_t})(y_t - \boldsymbol{w}_{\boldsymbol{t-1}}^T\Phi(\boldsymbol{x_t})) \\ &= \boldsymbol{w_{t-1}} + \eta e_t\Phi(\boldsymbol{x_t}) \end{aligned} \tag{2.5}$$

where, $e_t = y_t - \boldsymbol{w}_{\boldsymbol{t-1}}^T\Phi(\boldsymbol{x_t})$.
Recursively, we obtain:

$$\begin{aligned} \boldsymbol{w_t} &= \boldsymbol{w_{t-1}} + \eta e_t\Phi(\boldsymbol{x_t}) \\ &= \boldsymbol{w_{t-2}} + \eta[e_{t-1}\Phi(\boldsymbol{x_{t-1}}) + \eta e_t\Phi(\boldsymbol{x_t})] \\ &\vdots \\ &= \boldsymbol{w_o} + \eta\Big[\sum_{i=1}^{t} e_i\Phi(\boldsymbol{x_i})\Big] \\ &= \eta\Big[\sum_{i=1}^{t} e_i\Phi(\boldsymbol{x_i})\Big], \end{aligned} \tag{2.6}$$

assuming $\boldsymbol{w_o} = 0$. Estimate at input $\boldsymbol{x}$ after time t will be:

$$\begin{aligned} \hat{f}_t(\boldsymbol{x}) &= \boldsymbol{w}_{\boldsymbol{t}}^T\Phi(\boldsymbol{x}) \\ &= \Big[\eta\sum_{i=1}^{t} e_i\Phi(\boldsymbol{x_i})^T\Big]\Phi(\boldsymbol{x}) \\ &= \eta\sum_{i=1}^{t} e_i[\Phi(\boldsymbol{x_i})^T\Phi(\boldsymbol{x})] \\ &\overset{\text{kerneltrick}}{=} \eta\sum_{i=1}^{t} e_i k(\boldsymbol{x_i}, \boldsymbol{x}). \end{aligned} \tag{2.7}$$

So, in terms of the estimate, one can write the updates as:

$$\begin{aligned} f_{t-1} &= \eta \sum_{i=1}^{t-1} e_i k(\boldsymbol{x_i}, \cdot), \\ f_{t-1}(\boldsymbol{x_t}) &= \eta \sum_{i=1}^{t-1} e_i k(\boldsymbol{x_i}, \boldsymbol{x_t}), \\ e_t &= y_i - f_{t-1}(\boldsymbol{x_t}), \\ f_t &= f_{t-1} + \eta e_t k(\boldsymbol{x_t}, \cdot). \end{aligned} \tag{2.8}$$

Finally, in the notation of coefficients, we can write the updates as:

$$\boldsymbol{\alpha_t} = \begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ \eta e_t \end{bmatrix}. \tag{2.9}$$

Again, note that like NORMA this method also doesn't use any growing and pruning strategy. If we add an extra quantisation condition [25] to control the growth of the dictionary, we will get the next algorithm called Quantised KLMS (QKLMS).
At this stage, it is worth mentioning that this algorithm can also be derived from the parametric form of objective. In that case, the whole coefficient vector will be updated. This derivation has been presented in section 10.1.

2.4.2 Step-Size parameter

As already mentioned, $\boldsymbol{\Phi_t} = [\Phi(\boldsymbol{x_1}), \ldots, \Phi(\boldsymbol{x_t})]$. Next, define

$$\boldsymbol{R_\phi} = \frac{1}{N} \boldsymbol{\Phi_t} \boldsymbol{\Phi_t^T},$$

and

$$\boldsymbol{G_\phi} = \boldsymbol{\Phi_t^T} \boldsymbol{\Phi_t}.$$

Then the step size should satisfy following for the stability of the algorithm [9]:

$$\eta < \frac{1}{\rho_{max}},$$

where ρ_{max} is the largest eigenvalue of $\boldsymbol{R_\phi}$ To get an upper bound on ρ_{max} following is used:

$$\rho_{max} < tr[\boldsymbol{R_\phi}] < tr[\boldsymbol{G_\phi}]/t.$$

Hence a conservative upper bound on step size parameter is

$$\eta < \frac{t}{tr[\boldsymbol{G_\phi}]} = \frac{t}{\sum_{i=1}^{t} k(\boldsymbol{x_i}, \boldsymbol{x_i})} \ (= 1 \text{ in our case}).$$

2.4.3 Self-regularisation

Note that the objective in KLMS didn't contain any regularisation term because it can be shown that it has self-regularization ability [9]. The details related to this property have been skipped in this report.

2.5 Quantised KLMS (QKLMS)

In KLMS algorithm, we have ever-growing dictionary but in QKLMS [25], the new input will have to satisfy an additional condition (based on quantisation, see the 'if condition' in the second step of Algorithm 1 below) to get an entry into the dictionary. If the new input satisfies this additional condition, then the update equation will be exactly same as KLMS (a new component is added to the coefficient vector) but if the new input fails to satisfy the additional condition, the update will be different (some existing component of the coefficient vector will be updated). This algorithm will be better understood if we explain the quantisation procedure first.

2.5.1 Quantisation Procedure

In the quantisation literature, the set termed codebook is same as what we have been calling as dictionary.
The following pseudocode [25] explains the quantisation procedure nicely.

Input: $\{\boldsymbol{x}_t\}$, $t = 1, 2, \ldots$
Initialisation: choose quantisation size = ϵ_q and initialize dictionary $\mathcal{D}_1 = \{\boldsymbol{x_1}\}$
Computation:
while $\boldsymbol{x_t}$ available do:

1. Distance between new input and dictionary is computed: $dis(\boldsymbol{x_t}, D_{t-1}) = \min_{1 \leq j \leq m_{t-1}} \|\boldsymbol{x_t} - \boldsymbol{x}_{\omega_j}\|$
2. ***if*** $dis(\boldsymbol{x_t}, D_{t-1}) \leq \epsilon_q$
   - $D_t = D_{t-1}$, $m_t = m_{t-1}$
   - x_t will be quantized to the closest code vector (or dictionary element) i.e., $\boldsymbol{x_t^q} = \boldsymbol{x}_{\omega_{j^*}}$ where
   $$j^* = \underset{1 \leq j \leq m_{t-1}}{\operatorname{argmin}} \|\boldsymbol{x_t} - \boldsymbol{x}_{\omega_j}\|$$
3. ***if*** $dis(\boldsymbol{x_t}, D_{t-1}) > \epsilon_q$
   - update the codebook (or dictionary) i.e., $D_t = D_{t-1} \cup \{\boldsymbol{x_t}\}$, $m_t = m_{t-1} + 1$
   - quantize $\boldsymbol{x_t}$ to itself $\boldsymbol{x_t^q} = \boldsymbol{x_t}$ $(= \boldsymbol{x}_{\omega_{m_t}})$

Algorithm 1: Pseudo code for online quantisation

2.5.2 Update and sparsification

Let us denote the quantisation in input space (which was explained above) by $\mathcal{Q}$ and the quantisation in feature space by $\boldsymbol{\mathcal{Q}}$. The quantisation is not performed in the feature space since its dimension is very high so this notation has been introduced just for the purpose of explaining the method. Revisit (2.6) update of KLMS. The new feature map will be replaced by its quantised version:

$$\begin{aligned} \boldsymbol{w_o} &= \mathbf{0}, \\ e_t &= y_t - \boldsymbol{w_{t-1}}^T \Phi(\boldsymbol{x_t}), \\ \boldsymbol{w_t} &= \boldsymbol{w_{t-1}} + \eta e_t \boldsymbol{\mathcal{Q}}[\Phi(\boldsymbol{x_t})]. \end{aligned} \tag{2.10}$$

As mentioned above, since, quantisation is preferred in input space, so the last step will be written as following:

$$\boldsymbol{w_t} = \boldsymbol{w_{t-1}} + \eta e_t \Phi(\mathcal{Q}[\boldsymbol{x_t}]). \tag{2.11}$$

We have used $\boldsymbol{x_t^q} = \mathcal{Q}[\boldsymbol{x_t}]$, let us denote $\Phi_q(\boldsymbol{x_t}) = \boldsymbol{\mathcal{Q}}[\Phi(\boldsymbol{x_t})]$ for the purpose of analysis of this algorithm. Using these notations, the estimate will now look like (From (2.8), (2.10)

and (2.11)):

$$\begin{aligned} f_o &= 0, \\ e_t &= y_i - f_{t-1}(\boldsymbol{x_t}), \\ f_t &= f_{t-1} + \eta e_t k(\boldsymbol{x_t^q}, \cdot). \end{aligned} \tag{2.12}$$

Following pseudocode (Algorithm 2) explains QKLMS nicely.

Input: $\{(\boldsymbol{x}_t, y_t)\}$, $t = 1, 2, \ldots$
Initialisation: choose step size= η, kernel width $\sigma > 0$, quantisation size = $\epsilon_q \geq 0$ and initialize dictionary $\mathcal{D}_1 = \{\boldsymbol{x_1}\}$, $m_1 = 1$ and coefficient vector $\boldsymbol{\alpha_1} = [\eta y_1]$
Computation:
while $\boldsymbol{x_t}$ ($t \geq 2$) available do:

1. compute the estimate:
$$f_{t-1}(\boldsymbol{x_t}) = \sum_{i=1}^{m_{t-1}} \boldsymbol{\alpha_{t-1}}(i) k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_t})$$
2. compute the error:
$e_t = y_t - f_{t-1}(\boldsymbol{x_t})$
3. compute the distance between new input and dictionary: $dis(\boldsymbol{x_t}, D_{t-1}) = \min_{1 \leq j \leq m_{t-1}} \|\boldsymbol{x_t} - \boldsymbol{x}_{\omega_j}\|$
4. ***if*** $dis(\boldsymbol{x_t}, D_{t-1}) \leq \epsilon_q$
   - $D_t = D_{t-1}$, $m_t = m_{t-1}$
   - x_t will be quantized to the closest code vector (or dictionary element) i.e.,
   $\boldsymbol{x_t^q} = \boldsymbol{x}_{\omega_{j^*}}$ where
   $$j^* = \underset{1 \leq j \leq m_{t-1}}{\operatorname{argmin}} \|\boldsymbol{x_t} - \boldsymbol{x}_{\omega_j}\|$$
   - $f_t = f_{t-1} + \eta e_t k(D_{t-1}(j^*), \cdot)$ from (6.3).
   - $\boldsymbol{\alpha_t}(j^*) = \boldsymbol{\alpha_{t-1}}(j^*) + \eta e_t$.
5. ***if*** $dis(\boldsymbol{x_t}, D_{t-1}) > \epsilon_q$
   - update the codebook (or dictionary) i.e.,
   $D_t = D_{t-1} \cup \{\boldsymbol{x_t}\}$, $m_t = m_{t-1} + 1$
   - quantize $\boldsymbol{x_t}$ to itself
   $\boldsymbol{x_t^q} = \boldsymbol{x_t}$
   - $f_t = f_{t-1} + \eta e_t k(\boldsymbol{x_t}, \cdot)$
   - $\boldsymbol{\alpha_t} = \begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ \eta e_t \end{bmatrix}$

Algorithm 2: Pseudo code for QKLMS

2.5.3 Mean Square Convergence Analysis for QKLMS

Recall that $\boldsymbol{x_t^q} = \mathcal{Q}[\boldsymbol{x_t}]$ and $\Phi_q(\boldsymbol{x_t}) = \boldsymbol{\mathcal{Q}}[\Phi(\boldsymbol{x_t})]$, where $\mathcal{Q}$ and $\boldsymbol{\mathcal{Q}}$ are quantisations in input and feature spaces respectively.
Let us assume the input-output pairs in the observations are related as:

$$y_t = f^*(\boldsymbol{x_t}) + \xi_t,$$

where ξ_t is some disturbance noise (iid $\mathcal{N}(0, \sigma^2)$) and f^* denotes the unknown non-linear function that we need to estimate. Suppose that this unknown function can be written as $f^*(\cdot) = \boldsymbol{w^{*T}}\Phi(\cdot)$ (by the virtue of universal approximation property [25]), then we can rewrite above relation as:

$$y_t = \boldsymbol{w^{*T}}\Phi(\boldsymbol{x_t}) + \xi_t.$$

Prediction error at time t is $e_t = y_t - w_{t-1}^T\Phi(\boldsymbol{x_t})$. Let us define $\underline{\tilde{\boldsymbol{w}}_t = \boldsymbol{w^*} - \boldsymbol{w_t}.}$ Using this we can define 'a-priori error' $(e_t(a))$ and 'a posteriori error' $e_t(p)$ as:

$$e_t(a) := \tilde{\boldsymbol{w}}_{t-1}^T\Phi(\boldsymbol{x_t}),$$

and

$$e_t(p) := \tilde{\boldsymbol{w}}_t^T\Phi(\boldsymbol{x_t}).$$

From the QKLMS update, we have following:

$$\tilde{\boldsymbol{w}}_t^T = \tilde{\boldsymbol{w}}_{t-1}^T - \eta e_t\Phi_q(\boldsymbol{x_t}).$$

We can also arrive at following relation between $e_t(a)$ and $e_t(p)$:

$$e_t(p) = e_t(a) + \big(\tilde{\boldsymbol{w}}_t^T - \tilde{\boldsymbol{w}}_{t-1}^T\big)\Phi(\boldsymbol{x_t}). \tag{2.13}$$

Let $\mathbb{F}$ denotes the feature map (In first section it was l^2 for example), then one can show that

$$\boxed{\|\tilde{\boldsymbol{w}}_t\|_{\mathbb{F}}^2 + \frac{e_t(a)^2}{k(\boldsymbol{x_t^q}, \boldsymbol{x_t})^2} = \|\tilde{\boldsymbol{w}}_{t-1}\|_{\mathbb{F}}^2 + \frac{e_t(p)^2}{k(\boldsymbol{x_t^q}, \boldsymbol{x_t})^2} + \beta_q}, \tag{2.14}$$

where

$$\beta_q = \frac{2(e_t(p) - e_t(a))\big\{\tilde{\boldsymbol{w}}_{t-1}^T\Phi_q(\boldsymbol{x_t})k(\boldsymbol{x_t^q}, \boldsymbol{x_t}) - e_t(a)\big\}}{k(\boldsymbol{x_t^q}, \boldsymbol{x_t})^2}.$$

This equation is called energy conservation equation. Note that as quantisation radius $\epsilon_q \to 0$, $\beta_q \to 0$ and this equation reduces to

$$\|\tilde{\boldsymbol{w}}_t\|_{\mathbb{F}}^2 + \frac{e_t(a)^2}{k(\boldsymbol{x}_t^q, \boldsymbol{x}_t)^2} = \|\tilde{\boldsymbol{w}}_{t-1}\|_{\mathbb{F}}^2 + \frac{e_t(p)^2}{k(\boldsymbol{x}_t^q, \boldsymbol{x}_t)^2}, \quad \text{where } \|\boldsymbol{z}\|_{\mathbb{F}}^2 = \boldsymbol{z}^T\boldsymbol{z}.$$

Next, assuming that ξ_t is independent of $e_t(a)$, substituting (2.13) in (2.14) and then taking expectations on both sides, we get:

$$E\left[\|\tilde{\boldsymbol{w}}_t\|_{\mathbb{F}}^2\right] = E\left[\|\tilde{\boldsymbol{w}}_{t-1}\|_{\mathbb{F}}^2\right] + \eta^2\left(E\left[e_t(a)^2\right] + \sigma^2\right) - 2\eta E\left[e_t(a)\tilde{\boldsymbol{w}}_{t-1}^T\Phi_q(\boldsymbol{x}_t)\right], \tag{2.15}$$

where $E\left[\|\tilde{\boldsymbol{w}}_t\|_{\mathbb{F}}^2\right]$ is called weight-error power (WEP)[25]. Next, to ensure monotonic decrease of WEP, we find a condition on step size.

$$\begin{aligned} E\left[\|\tilde{\boldsymbol{w}}_t\|_{\mathbb{F}}^2\right] \quad &\leq \quad E\left[\|\tilde{\boldsymbol{w}}_{t-1}\|_{\mathbb{F}}^2\right] \\ &\iff \quad \eta^2\left(E\left[e_t(a)^2\right] + \sigma^2\right) - 2\eta E\left[e_t(a)\tilde{\boldsymbol{w}}_{t-1}^T\Phi_q(\boldsymbol{x}_t)\right] \leq 0 \qquad (2.16) \\ &\iff \quad \eta \leq \frac{2E\left[e_t(a)\tilde{\boldsymbol{w}}_{t-1}^T\Phi_q(\boldsymbol{x}_t)\right]}{E\left[e_t(a)^2\right] + \sigma^2}. \end{aligned}$$

Since, $\eta > 0$, we get following sufficient condition for monotonic decrease of WEP and hence mean square convergence:

$$\begin{aligned} &\forall t, \\ &2E\left[e_t(a)\tilde{\boldsymbol{w}}_{t-1}^T\Phi_q(\boldsymbol{x}_t)\right] > 0, \qquad (2.17) \\ &0 < \eta \leq \frac{2E\left[e_t(a)\tilde{\boldsymbol{w}}_{t-1}^T\Phi_q(\boldsymbol{x}_t)\right]}{E\left[e_t(a)^2\right] + \sigma^2}. \qquad (2.18) \end{aligned}$$

Condition (2.17) can be further replaced by

$$E\big(\Phi(\boldsymbol{x}_t)\Phi_q(\boldsymbol{x}_t)\big) > 0,$$

using an assumption that $\Phi(\boldsymbol{x}_t)$ and $\tilde{\boldsymbol{w}}_{t-1}$ are independent.
Next, assume that the sufficient conditions hold and take limit $t \to \infty$, on both sides of (2.15). Now the algorithm reaches steady state, we have: $\lim_{t\to\infty} E\left[\|\tilde{\boldsymbol{w}}_t\|_{\mathbb{F}}^2\right] = \lim_{t\to\infty} E\left[\|\tilde{\boldsymbol{w}}_{t-1}\|_{\mathbb{F}}^2\right]$.

Therefore, following, must hold:

$$\eta^2\left(\lim_{t\to\infty} E\left[e_t(a)^2\right] + \sigma^2\right) - 2\eta \lim_{t\to\infty} E\left[e_t(a)\tilde{\boldsymbol{w}}_{t-1}^T \Phi_q(\boldsymbol{x_t})\right] = 0.$$

From here we get expression for the steady state excess MSE (apriori error is also referred to as excess error) as (using the expression for $e_t(a)$ in one of the steps):

$$\lim_{t\to\infty} E\left[e_t(a)^2\right] = \frac{\eta\sigma^2 - 2\lim_{t\to\infty} E\left[\tilde{\boldsymbol{w}}_{t-1}^T \Phi(\boldsymbol{x_t})\tilde{\boldsymbol{w}}_{t-1}^T(\Phi_q(\boldsymbol{x_t}) - \Phi(\boldsymbol{x_t}))\right]}{2-\eta}.$$

One can show that

$$\left|E\left[\tilde{\boldsymbol{w}}_{t-1}^T \Phi(\boldsymbol{x_t})\tilde{\boldsymbol{w}}_{t-1}^T(\Phi_q(\boldsymbol{x_t}) - \Phi(\boldsymbol{x_t}))\right]\right| \leq \sqrt{2 - 2exp(\tfrac{-\gamma^2}{2})} E\left[\|\boldsymbol{w}^*\|_{\mathbb{F}}^2\right].$$

And hence using this in above expression we get upper and lower bounds on the steady-state EMSE as:

$$\boxed{max\left\{\frac{\eta\sigma^2 - 2\zeta}{2-\eta}, 0\right\} \leq \lim_{t\to\infty} E\left[e_t(a)^2\right] \leq \frac{\eta\sigma^2 + 2\zeta}{2-\eta}},$$

where $\zeta = \sqrt{2 - 2exp(\frac{-\gamma^2}{2})} E\left[\|\boldsymbol{w}^*\|_{\mathbb{F}}^2\right]$.

2.6 Kernel Normalised Least Mean Squares (KNLMS)

This algorithm is a special case of a class of algorithms called kernel affine projection algorithms (**KAPA**)[9] combined with a growth control criteria for the dictionary. The reason why such algorithms are called affine projection algorithms will be clear in the explanation below. In this case we will use coherence based growth criterion. I will explain this procedure again but one should note that it is equivalent to quantisation mentioned in QKLMS for radial kernels (gaussian kernel for example).
Note that till now, NORMA, KLMS and QKLMS algorithms involved updating a single component of the coefficient vector $\boldsymbol{\alpha_t}$, but in this method the whole vector $\boldsymbol{\alpha_t}$ will be updated as the parametric form of the objective is considered here.
Please note that KAPA is not linear algorithm but its special case KNLMS[22] is.

2.6.1 Objective and Updating Procedure

Here the parametric form of objective is used:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^t} \|\boldsymbol{y}_t - K\boldsymbol{\alpha}\|^2.$$

In order to avoid taking inverse at each step, till now we relied on gradient descent methods to move to the solution. But unlike previous descent methods, where movement was along instantaneous direction ($i = t$), KAPA considers recent p observations and solves the following problem at t^{th} time:

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{\alpha} - \boldsymbol{\alpha_{t-1}}\|^2, \quad \text{s.t. } \boldsymbol{y_p} = \boldsymbol{K_p}\boldsymbol{\alpha}, \tag{2.19}$$

where ith row of $\boldsymbol{K_p}$ is $\boldsymbol{k_{p_{t-i+1}}} = [k(\boldsymbol{x_{t-i+1}}, \boldsymbol{x_{\omega_1}}), \ldots, k(\boldsymbol{x_{t-i+1}}, \boldsymbol{x_{\omega_{m_t}}})]^T$, and
$\boldsymbol{y_p} = [y_t, \ldots, y_{t-p+1}]^T$.
In other words, we are trying to project $\boldsymbol{\alpha_{t-1}}$ onto the intersection of following manifolds:

$$\mathcal{A}_i = \{\boldsymbol{\alpha} : \boldsymbol{k}^T_{\boldsymbol{p_{t-i+1}}}\boldsymbol{\alpha} = y_{t-i+1}\}, \quad \text{for } i = 1, \ldots, p.$$

And this is the reason why such algorithms are called affine projection algorithms.
Next, let us visit the growth criterion that will control the entry of new element to the dictionary. Dictionary at time t will be denoted by D_t and its element by $\{\boldsymbol{x_{\omega_i}}\}_{i=1}^{m_t}$ where m_t is size of D_t.
Let us revisit coherence criterion once again. At first threshold μ_o is chosen and define :

$$\mu(\boldsymbol{x}) = \max_{\boldsymbol{x_{\omega_i}}\in D_t} k(\boldsymbol{x_{\omega_i}}, x).$$

When $(\boldsymbol{x_t}, y_t)$ is observed, we calculate $\mu(\boldsymbol{x})$. If $\mu(\boldsymbol{x}) > \mu_o$, element won't be added to the dictionary otherwise it will be. In both the cases, the update equations will be explained below.

1. **Case 1:** $\mu(\boldsymbol{x_t}) > \mu_o$;

   The new element won't be added to the dictionary. The solution to (2.19) can be obtained using method of lagrange multipliers by minimising following lagrangian func-

tion:

$$\mathcal{J}(\boldsymbol{\alpha}, \boldsymbol{\lambda}) = \|\boldsymbol{\alpha} - \boldsymbol{\alpha_{t-1}}\|^2 + \boldsymbol{\lambda}^T(\boldsymbol{y_p} - \boldsymbol{K_p}\boldsymbol{\alpha}).$$

Upon differentiating w.r.t. $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ separately and substituting together we get minimiser ($\hat{\boldsymbol{\alpha}}$) must satisfy:

$$2(\hat{\boldsymbol{\alpha}} - \boldsymbol{\alpha_{t-1}}) = \boldsymbol{K_p^T}\boldsymbol{\lambda}, \tag{2.20}$$

$$\boldsymbol{K_p}\hat{\boldsymbol{\alpha}} = \boldsymbol{y_p}. \tag{2.21}$$

Substituting (2.21) in (2.20) and then eliminating $\boldsymbol{\lambda}$ we get:

$$\hat{\boldsymbol{\alpha}} = \boldsymbol{\alpha_{t-1}} + \boldsymbol{K_p^T}(\boldsymbol{K_p}\boldsymbol{K_p^T})^{-1}(\boldsymbol{y_p} - \boldsymbol{K_p}\boldsymbol{\alpha_{t-1}}).$$

Next a step size parameter and regularisation factor is added and the update equation will look like:

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha_{t-1}} + \eta\boldsymbol{K_p^T}\big(\epsilon\boldsymbol{I} + \boldsymbol{K_p}\boldsymbol{K_p^T}\big)^{-1}(\boldsymbol{y_p} - \boldsymbol{K_p}\boldsymbol{\alpha_{t-1}}). \tag{2.22}$$

2. **Case 2:** $\mu(\boldsymbol{x_t}) \leq \mu_o$;

   In this the new element will be added to the dictionary ($m_{t+1} = m_t$) i.e., $D_t = D_{t-1} \cup \{\boldsymbol{x}_{\boldsymbol{\omega}_{\boldsymbol{m_{t+1}}}}\}$, where $\boldsymbol{x_t}$ is denoted by $\boldsymbol{x}_{\boldsymbol{\omega}_{\boldsymbol{m_{t+1}}}}$ and hence $\boldsymbol{K_p}$ will be updated. A new column $[k(\boldsymbol{x_t}, \boldsymbol{x}_{\boldsymbol{\omega}_{\boldsymbol{m_{t+1}}}}), \ldots, k(\boldsymbol{x_{t-p+1}}, \boldsymbol{x}_{\boldsymbol{\omega}_{\boldsymbol{m_{t+1}}}}),]^T$ is added. Now, (2.19) will be solved with updated mtarix and m_{t+1}-dimensional $\boldsymbol{\alpha}$ will be calculated. A zero will be added to the previous coefficient vector and using same steps we get:

$$\boldsymbol{\alpha}_t = \begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ 0 \end{bmatrix} + \eta\boldsymbol{K_p^T}\big(\epsilon\boldsymbol{I} + \boldsymbol{K_p}\boldsymbol{K_p^T}\big)^{-1}\left(\boldsymbol{y_p} - \boldsymbol{K_p}\begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ 0 \end{bmatrix}\right). \tag{2.23}$$

2.6.2 Special Case $p = 1$

When we use $p = 1$ in the above algorithm, we get KNLMS. At time t, we now have to solve:

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{\alpha} - \boldsymbol{\alpha_{t-1}}\|^2, \quad \text{s.t. } y_t = \boldsymbol{k_t}^T\boldsymbol{\alpha},$$

where $\boldsymbol{k_t} = [k(\boldsymbol{x_t}, \boldsymbol{x_{\omega_1}}), \ldots, k(\boldsymbol{x_t}, \boldsymbol{x_{\omega_{m_t}}})]^T$. The solution will be obtained using same steps as above with $\boldsymbol{y_p} = y_t$ and $\boldsymbol{K_p} = \boldsymbol{k_t}^T$.
The pseudocode (Algorithm 3) will be used to explain KNLMS.

Input: $\{(\boldsymbol{x}_t, y_t)\}$, $t = 1, 2, \ldots$
Initialisation: choose step size= η, kernel width $\sigma > 0$, Choose threshold $=\mu_o \geq 0$ and initialize dictionary $\mathcal{D}_1 = \{\boldsymbol{x_1}\}$ and coefficient vector $\boldsymbol{\alpha_1} = 0$, $m_1 = 1$
Computation:
while $\boldsymbol{x_t}$ available do:

1. **if** $\mu(\boldsymbol{x_t}) > \mu_o$
   $D_t = D_{t-1}$, $m_t = m_{t-1}$
   Compute $\boldsymbol{k_t} = [k(\boldsymbol{x_t}, \boldsymbol{x_{\omega_1}}), \ldots, k(\boldsymbol{x_t}, \boldsymbol{x_{\omega_{m_t}}})]^T$
   Update step:
   $$\boldsymbol{\alpha}_t = \boldsymbol{\alpha_{t-1}} + \frac{\eta}{\epsilon + \|\boldsymbol{k_t}\|^2}(y_t - \boldsymbol{k_t^T}\boldsymbol{\alpha_{t-1}})\boldsymbol{k_t}$$
2. **if** $\mu(\boldsymbol{x_t}) \leq \mu_o$
   $D_t = D_{t-1} \cup \{\boldsymbol{x_{\omega_{m_t}}}\}$ $\boldsymbol{x_t}$ is denoted by $\boldsymbol{x_{\omega_{m_t}}}$, $m_t = m_{t-1} + 1$
   Compute $\boldsymbol{k_t} = [k(\boldsymbol{x_t}, \boldsymbol{x_{\omega_1}}), \ldots, k(\boldsymbol{x_t}, \boldsymbol{x_{\omega_{m_t}}}), k(\boldsymbol{x_t}, \boldsymbol{x_{\omega_{m_t}}})]^T$
   Update step:
   $$\boldsymbol{\alpha}_t = \begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ 0 \end{bmatrix} + \frac{\eta}{\epsilon + \|\boldsymbol{k_t}\|^2}\left(y_t - \boldsymbol{k_t^T}\begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ 0 \end{bmatrix}\right)\boldsymbol{k_t}$$

Algorithm 3: Pseudo code for KNLMS

2.6.3 Comment on the choice of space while affine projection

The choice of affine space also affects the updates just as the choice of objective affects the updates. We will again see that if the functional space is chosen for the projection then only one coefficient is updated but if the parametric space is chosen, then the whole coefficient vector is updated (as explained in the algorithm above). I am following the explanation from (Appendix A, [26]). Let

$$\Pi_t := \{\boldsymbol{z} \in \mathbb{R}^m : \boldsymbol{k_t}^T\boldsymbol{z} = y_t\},$$

and

$$\hat{\Pi}_t := \{f \in \mathcal{H}_K : f(\boldsymbol{x_t}) = \langle f, k(\boldsymbol{x_t}, \cdot)\rangle_{\mathcal{H}_K} = y_t\}.$$

Now, for a complete metric space X consider $C \subset X$. For any $x \in X$, we define operator P_C as:

$$P_C(x) = \underset{y \in C}{\operatorname{argmin}} \|y - x\|_X.$$

Now, observe that the updates in KNLMS can be written as (let $\epsilon = 0$):

$$\boldsymbol{\alpha_t} := \boldsymbol{\alpha_{t-1}} + \eta(P_{\Pi_t} - \boldsymbol{\alpha_{t-1}}).$$

When new element is added to the dictionary, $m = m_t + 1$ and when it is not added, $m = m_t$ In other words, this updating procedure relies on projecting in parameter space with corresponding Euclidean norm. Let us call it parametric approach.
Now, consider the following updating procedure:

$$f_t := f_{t-1} + \eta(P_{\hat{\Pi}_t} - f_{t-1}).$$

This is the projection in RKHS using corresponding norm. Let us call it functional approach. Actually, this was the original formulation of KAP algorithms.
In the functional approach, the solution moves according to the vector $k(\boldsymbol{x_t}, \cdot)$ (either in its positive direction or negative direction depending on prediction error) and reaches $\hat{\Pi}_t$. Notice that it is similar to KLMS (2.8) and that is the same reason why those methods concentrated their updates along one coefficient only.
On the other hand in paramatric approach, the next vector moves according to the whole kernel vector of the dictionary $[k(\boldsymbol{x}_{\omega_1}, \cdot), \ldots, k(\boldsymbol{x}_{\omega_{m_t}}, \cdot)]^T$, and that is the reason why every component was getting updated at each time.
But in both the cases, it can be shown that they satisfy monotonic decreasing property [27] i.e.,

$$\|f_t - f^*\|_{\mathcal{H}_K} \leq \|f_{t-1} - f^*\|_{\mathcal{H}_K},$$

$$\|\boldsymbol{\alpha_t} - \boldsymbol{\alpha^*}\| \leq \|\boldsymbol{\alpha_{t-1}} - \boldsymbol{\alpha^*}\|$$

where superscript * denotes the true function or coefficient which we are trying to find.

2.7 Multi-Kernel Normalised Mean Squares Algorithm (MKNLMS)

In [26], the use of multi-kernels is suggested. The motivation behind this was the lack of knowledge about choice of optimal kernel. The parametric form of the KAP algorithms will be followed here just like KNLMS. Update equations will be similar to (2.22) and (2.23) (with

coherence criterion as before) but now instead of coefficient vector, coefficient matrix will be updated. Next, an additional penalty (block soft-thresholding) will be used on coefficient matrix which ensures further sparsification.

2.7.1 Notations

For $A, B \in \mathbb{R}^{p\times q}$, define inner product as $\langle A, B\rangle = tr(A^T B)$ and $\|A\| = \sqrt{\langle A, A\rangle}$. For a closed convex set $C \subset \mathbb{R}^{p\times q}$, we define:$P_C(X) = \underset{Y\in C}{\operatorname{argmin}}\|X - Y\|$.
The set of indices of kernel will be denoted by $\mathcal{M}$ ($\mathcal{M} = \{1, 2, \ldots, m\}$ in this case), dictionary at time t will as usual be denoted by D_t and the indices of the dictionary will be denoted by the index set $\mathcal{J}_t$. So $D_t = \{\boldsymbol{x}_{\omega_j}\}_{\omega_j\in\mathcal{J}_t}$ and $|D_t| = m_t$, $\mathcal{J}_t = \{\omega_{j_1}, \ldots \omega_{j_{m_t}}\}$.
The ω_j-th ($j \in \mathcal{J}_t$) component of the coefficient vector of m-th ($m \in \mathcal{M}$) kernel at time t will be denoted by $h^{(m)}_{\omega_j,t}$ and m-th kernel will be denoted by $k_m : X \times X \to \mathbb{R}$.

2.7.2 Updating Procedure with Coherence Criterion (MLKNLMS-CS)

In this method, coherence criterion is used to control the entry of recent input into the dictionary and then update is performed similar to KNLMS. Using the notation above, the estimating function at time $t + 1$ will be:

$$\hat{f}(\boldsymbol{x}) = \sum_{m\in\mathcal{M}}\sum_{\omega_j\in\mathcal{J}_t} h^{(m)}_{\omega_j,t} k_m(\boldsymbol{x}, \boldsymbol{x}_{\omega_j})$$

For input $\boldsymbol{x_{t+1}}$., the estimate $\hat{y}_{t+1}$ will be:

$$\hat{y}_{t+1} = \sum_{\omega_j\in\mathcal{J}_t} \boldsymbol{h}^T_{\boldsymbol{\omega_j,t}} \boldsymbol{k}_{\boldsymbol{\omega_j,t}},$$

where,

$$\boldsymbol{h}_{\boldsymbol{\omega_j,t}} = [h^{(1)}_{\omega_j,t}, \ldots, h^{(M)}_{\omega_j,t}]^T \in \mathbb{R}^M,$$

$$\boldsymbol{k}_{\boldsymbol{\omega_j,t}} = [k_1(\boldsymbol{x_{t+1}}, \boldsymbol{x}_{\omega_j}), \ldots, k_M(\boldsymbol{x_{t+1}}, \boldsymbol{x}_{\omega_j})]^T \in \mathbb{R}^M.$$

This estimate can also be written as:

$$\hat{y}_{t+1} = tr(\boldsymbol{H}_{\boldsymbol{t}}^T \boldsymbol{K}_{\boldsymbol{t}}) = \langle \boldsymbol{H}_{\boldsymbol{t}}, \boldsymbol{K}_{\boldsymbol{t}} \rangle,$$

where

$$\boldsymbol{H}_{\boldsymbol{t}} = [\boldsymbol{h}_{\boldsymbol{\omega}_{j_1},t}, \ldots, \boldsymbol{h}_{\boldsymbol{\omega}_{j_{m_t}},t}] \in \mathbb{R}^{M \times m_t},$$

$$\boldsymbol{K}_{\boldsymbol{t}} = [\boldsymbol{k}_{\boldsymbol{\omega}_{j_1},t}, \ldots, \boldsymbol{k}_{\boldsymbol{\omega}_{j_{m_t}},t}] \in \mathbb{R}^{M \times m_t}.$$

We have to improve the definition of coherence since many kernels are involved. We define coherence criterion in this case as:

$$\|\boldsymbol{K}_{\boldsymbol{t+1}}\|_{max} := \max_{m \in \mathcal{M}} \max_{\omega_j \in \mathcal{J}_t} |k_m(\boldsymbol{x}_{\boldsymbol{t+1}}, \boldsymbol{x}_{\omega_j})| \leq \mu_o. \tag{2.24}$$

Following the method in KNLMS ($p = 1$, Algorithm 3) with $\boldsymbol{\alpha}_{\boldsymbol{t}}$ replaced with $\boldsymbol{H}_{\boldsymbol{t}}$ and $\boldsymbol{k}_{\boldsymbol{t}}$ replaced with $\boldsymbol{K}_{\boldsymbol{t}}$ and finally replacing μ in coherence criterion with $\|\boldsymbol{K}_{\boldsymbol{t}}\|_{max}$, we get updates as:

1. **Case 1**: When (2.24) is not satisfied.
   The new element won't be added to the dictionary and we obtain the update:

$$\boldsymbol{H}_{\boldsymbol{t+1}} = \boldsymbol{H}_{\boldsymbol{t}} + \eta \frac{y_{t+1} - \langle \boldsymbol{H}_{\boldsymbol{t}}, \boldsymbol{K}_{\boldsymbol{t}} \rangle}{\epsilon + \|\boldsymbol{K}_{\boldsymbol{t}}\|^2} \boldsymbol{K}_{\boldsymbol{t}}. \tag{2.25}$$

2. **Case 2**: When (2.24) is satisfied.
   The new element will be added to the dictionary and after few modifications in coefficient and kernel matrices we obtain the update:

$$\boldsymbol{H}_{\boldsymbol{t+1}} = \bar{\boldsymbol{H}}_{\boldsymbol{t}} + \eta \frac{y_{t+1} - \langle \bar{\boldsymbol{H}}_{\boldsymbol{t}}, \bar{\boldsymbol{K}}_{\boldsymbol{t}} \rangle}{\epsilon + \|\bar{\boldsymbol{K}}_{\boldsymbol{t}}\|^2} \bar{\boldsymbol{K}}_{\boldsymbol{t}}, \tag{2.26}$$

   where $\bar{\boldsymbol{H}}_{\boldsymbol{t}} = [\boldsymbol{H}_{\boldsymbol{t}}\ \boldsymbol{0}]$ ($\in \mathbb{R}^{M \times (m_t+1)}$) and $\bar{\boldsymbol{K}}_{\boldsymbol{t}} = [\boldsymbol{K}_{\boldsymbol{t}}\ \bar{\boldsymbol{k}}_{t+1}]$ ($\in \mathbb{R}^{M \times (m_t+1)}$),
   $\bar{\boldsymbol{k}}_{t+1} = [k_1(\boldsymbol{x}_{\boldsymbol{t+1}}, \boldsymbol{x}_{\boldsymbol{t+1}}), \ldots, k_M(\boldsymbol{x}_{\boldsymbol{t+1}}, \boldsymbol{x}_{\boldsymbol{t+1}})]^T$.

2.7.3 Updating Procedure using Block-Soft Thresholding (MKNLMS-BT)

In this method, there is no growth criterion. At each time, the recent input is added to the dictionary and then a constrained optimisation problem is solved which ensures removal of irrelevant elements from the dictionary. The regularised problem consists a part which is differentiable and a part which is not (see below). Forward-Backward splitting is used to solve this problem. The proximity operator is used which attracts small coefficient column vectors to zero. In the end, whichever coefficient column is zero, the corresponding element is excluded from the dictionary.
When a new input $\boldsymbol{x_{t+1}}$ arrives, it is added to the dictionary and following updates are performed before solving any objective:

$$\mathcal{J}_t = \mathcal{J}_t \cup \{t+1\} \text{and} \;\; m_t = m_t + 1,$$

$$\boldsymbol{H_t} = \begin{bmatrix} \boldsymbol{H_t} & \boldsymbol{0} \end{bmatrix} \quad \in \mathbb{R}^{M\times(m_t+1)},$$

$$\boldsymbol{K_t} = \begin{bmatrix} \boldsymbol{K_t} & \bar{\boldsymbol{k}}_{t+1} \end{bmatrix} \quad \in \mathbb{R}^{M\times(m_t+1)},$$

where, $\bar{\boldsymbol{k}}_{t+1} = [k_1(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}}), \ldots, k_M(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}})]^T$.
Next, following cost function is considered:

$$C_t(\boldsymbol{X}) := \underbrace{\frac{1}{2} d^2(\boldsymbol{X}, D_t)}_{C_t^{(1)}} + \underbrace{\lambda \sum_{i=1}^{m_t} w_{i,t} \|\boldsymbol{x_i}\|_1}_{C_t^{(2)}} \;\; , \boldsymbol{X} := [\boldsymbol{x_1}, \ldots, \boldsymbol{x_{m_t}}] \in \mathbb{R}^{M\times m_t}, \tag{2.27}$$

where

$$d(\boldsymbol{X}, D_t) := \min_{\boldsymbol{Y}\in D_t} \|\boldsymbol{X} - \boldsymbol{Y}\|,$$

and D_t is closed convex set:

$$D_t := \{\boldsymbol{X} \in \mathbb{R}^{M\times m_t} : |\epsilon_t(\boldsymbol{X})| \le \epsilon\}, \qquad \text{where} \;\; \epsilon_t(\boldsymbol{X}) := \langle \boldsymbol{X}, \boldsymbol{K_t}\rangle - y_{t+1}.$$

Here $\epsilon \ge 0$ is small constant.
Since, $C_t^{(2)}$ is non-differentiable, proximal-forward backward splitting method [28] is used:

$$\tilde{\boldsymbol{H}}_{t+1} := \text{prox}_{\mu C_t^{(2)}}\big(\boldsymbol{H_t} - \mu \nabla C_t^{(1)}(\boldsymbol{H_t})\big). \tag{2.28}$$

It can be shown that:

$$\nabla C_t^{(1)}(\boldsymbol{H_t}) = \boldsymbol{H_t} - P_{D_t}(\boldsymbol{H_t}),$$

and

$$P_{D_t}(\boldsymbol{H_t}) = \begin{cases} \boldsymbol{H_t} - \dfrac{\epsilon_t(\boldsymbol{H_t}) - \epsilon}{\|\boldsymbol{K_t}\|^2}\boldsymbol{K_t} & \text{if } \epsilon_t(\boldsymbol{H_t}) > 0 \\ \boldsymbol{H_t} - \dfrac{\epsilon_t(\boldsymbol{H_t}) + \epsilon}{\|\boldsymbol{K_t}\|^2}\boldsymbol{K_t} & \text{if } \epsilon_t(\boldsymbol{H_t}) > 0 \\ \boldsymbol{H_t} & \text{otherwise} \end{cases}.$$

The operator $\text{prox}_{\mu C_t^{(2)}} : \mathbb{R}^{M\times m_t} \to \mathbb{R}^{M\times m_t}$ is the proximal operator of index μ. It can be shown ([28], (Appendix, [26])) that :

$$\begin{aligned} \text{prox}_{\mu C_t^{(2)}}(\boldsymbol{X}) &:= \underset{\boldsymbol{Y}\in\mathbb{R}^{M\times m_t}}{\text{argmin}}\ C_t^2 + \frac{1}{2\mu}\|\boldsymbol{X} - \boldsymbol{Y}\| \\ &= \sum_{i=1}^{m_t} max\left\{1 - \frac{\lambda\mu w_{i,t}}{\|\boldsymbol{x_i}\|}, 0\right\}\boldsymbol{x_i}\boldsymbol{e}_{i,t}^T. \end{aligned} \tag{2.29}$$

$\boldsymbol{x_i}$'s are rows of $\boldsymbol{X}$ in 2.27 and $\boldsymbol{e}_{i,t} \in \mathbb{R}^{m_t}$ is the unit vector (non-zero at i-th place). Using (2.29) in (2.28), we understand that the columns of $\bar{\boldsymbol{H}}_{t+1}$ are attracted to zero. Next, whichever columns become zero, the corresponding indices are removed from the dictionary. Algorithm 4 and Algorithm 5 contain pseudocode for both the approaches MKNLMS-CS and MKNLMS-BT respectively.

Input: $\{(\boldsymbol{x}_t, y_t)\}$, $t = 1, 2, \ldots$
Initialisation: choose step size= η, set of kernels = $\{k_1, \ldots, k_M\}$, Choose threshold =$\mu_o \geq 0$ and initialize dictionary $\mathcal{D}_1 = \{\boldsymbol{x_1}\}$ and coefficient matrix $\boldsymbol{H_1} = \boldsymbol{0}$, $m_1 = 1$
Computation:
while $\boldsymbol{x_{t+1}}$ available do:

1. **if** (2.24) holds:
   $D_{t+1} = D_t$, $m_{t+1} = m_t$
   Update according to (2.25).

2. **if** (2.24) doesn't hold:
   $D_{t+1} = D_t \cup \{\boldsymbol{x_t}\}$, $m_{t+1} = m_t + 1$
   Compute $\bar{\boldsymbol{H}}_{\boldsymbol{t}} = [\boldsymbol{H_t}\ \boldsymbol{0}]$ $(\in \mathbb{R}^{M\times(m_t+1)})$ and $\bar{\boldsymbol{K}}_{\boldsymbol{t}} = [\boldsymbol{K_t}\ \bar{\boldsymbol{k}}_{t+1}]$ $(\in \mathbb{R}^{M\times(m_t+1)})$,
   $\bar{\boldsymbol{k}}_{t+1} = [k_1(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}}), \ldots, k_M(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}})]^T$.
   Update according to (2.26).

Algorithm 4: Pseudo code for MKNLMS-CS

Input: $\{(\boldsymbol{x}_t, y_t)\}$, $t = 1, 2, \ldots$
Initialisation: choose proximity index= μ, set of kernels = $\{k_1, \ldots, k_M\}$, regularising parameter $\lambda \geq 0$ and initialize dictionary $\mathcal{D}_1 = \{\boldsymbol{x_1}\}$, coefficient matrix $\boldsymbol{H_1} = \boldsymbol{0}$, $m_1 = 1$ and weight vector (can be adaptive).
Computation:
while $\boldsymbol{x_{t+1}}$ available do:

$$\mathcal{J}_t = \mathcal{J}_t \cup \{t+1\} \text{and} \quad r_t = r_t + 1$$

$$\boldsymbol{H_t} = \begin{bmatrix} \boldsymbol{H_t} & \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{M \times (m_t+1)}$$

$$\boldsymbol{K_t} = \begin{bmatrix} \boldsymbol{K_t} & \boldsymbol{k_{t+1}} \end{bmatrix} \in \mathbb{R}^{M \times (m_t+1)}$$

where, $\boldsymbol{k}_{t+1} = [k_1(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}}), \ldots, k_M(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}})]^T$.
Using (2.29), compute: $\bar{\boldsymbol{H}}_{t+1} := \text{prox}_{\mu C_t^{(2)}}\left(\boldsymbol{H_t} - \mu \nabla C_t^{(1)}(\boldsymbol{H_t})\right)$
I_t =indices corresponding to zero columns on $\bar{\boldsymbol{H}}_{t+1}$
$\boldsymbol{H}_{t+1} = \bar{\boldsymbol{H}}_{t+1}|_{\mathcal{J}_t - I_t}$, where $A|_J$ denotes matrix A with all columns removed except those with indices J
$\mathcal{J}_{t+1} = \mathcal{J}_t - I_t$

Algorithm 5: Pseudo code for MKNLMS-BT

2.8 KLMS algorithm with forward-backward splitting (KLMS-L1)

The parametric perspective of KLMS has been used in this algorithm so its description has been provided first.

2.8.1 KLMS in parametric approach

In the original paper of KLMS [29], following objective has been considered ($t \in \mathbb{N}$):

$$J(\boldsymbol{w}) = \min_{\boldsymbol{w} \in l^2} \frac{1}{2} \sum_{i=1}^{t} (y_i - \boldsymbol{w}^T \Phi(\boldsymbol{x_i}))^2,$$

where the fact that every function (f) of RKHS can be written as $f(\cdot) = \boldsymbol{w}^T \Phi(\cdot)$ has been used. In this objective $\boldsymbol{w} \in l^2$ is the feature space and the map Φ is the feature map and both are defined in Chapter 2. KLMS with its original formulation has already been discussed in Section 3.6.
Using the representer theorem, a new formulation can be introduced for the same objective. The representer theorem states that the function represented as $f(\cdot) = \boldsymbol{k_t}^T \boldsymbol{\alpha}$ where $\boldsymbol{k_t}(\cdot) =$

$[k(\boldsymbol{x_1}, \cdot), \ldots, k(\boldsymbol{x_t}, \cdot)]^T$ and $\boldsymbol{\alpha} \in \mathbb{R}^t$ can act as a minimiser of

$$\min_{f \in \mathcal{H}_K} \sum_{i=1}^{t} (y_i - f(\boldsymbol{x_i}))^2,$$

where, $\mathcal{H}_K$ is the RKHS being considered. Thus, the objective can be rewritten as (input domain is $X \subset \mathbb{R}^d$):

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^t} \frac{1}{2} \sum_{i=1}^{t} (y_i - \boldsymbol{k_t}(\boldsymbol{x_i})^T \boldsymbol{\alpha})^2.$$

As usual, one doesn't want to express the minimiser in terms of every input observed till t so instead a dictionary is maintained at each time and the minimiser is expressed in terms of elements of the dictionary. Let the D_{t-1} denote the dictionary at time $t-1$ and m_{t-1} denote its size. We will denote the elements of the dictionary using $\boldsymbol{x}_{\omega_i}$ i.e, $D_{t-1} = \{\boldsymbol{x}_{\omega_i}\}_{i=1}^{m_{t-1}}$. Thus, the objective now will be to find the optimal coefficient vector in terms of the elements of the dictionary. And thus, the objective can be rewritten as

$$J(\boldsymbol{\alpha}) = \min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{t} (y_i - \boldsymbol{k_{\omega_t}}(\boldsymbol{x_i})^T \boldsymbol{\alpha})^2,$$

where $\boldsymbol{k_{\omega_t}}(\cdot) = [k(\boldsymbol{x}_{\omega_1}, \cdot), \ldots, k(\boldsymbol{x}_{\omega_{m_{t-1}}}, \cdot)]^T$. Note that in this objective, the dimension of $\boldsymbol{\alpha}$ has not been mentioned because it depends on whether the new element enters into the dictionary or not.

First, let us see the criteria which controls the entry of the new elements to the dictionary. Out of the growing strategies mentioned in Section 3.1, coherence based growth has been utilised here which provides a quantitative value for the similarity between an element and the dictionary. Let us revisit the coherence based rule. We define coherence for the input $\boldsymbol{x_t}$ at time t (μ_t) as:

$$\mu_t(\boldsymbol{x_t}) := \max_{\boldsymbol{x}_{\omega_i} \in D_{t-1}} k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_t}).$$

Next, fix some threshold parameter (μ_o) which decides the level of similarity that we want. According to the coherence-based growth:

- if $\mu_t(\boldsymbol{x_t}) > \mu_o$: The new element ($\boldsymbol{x_t}$) is similar to at least one of the dictionary elements. So, it is not added to the dictionary and thus $D_t = D_{t-1}$, $m_t = m_{t-1}$ and we search $\boldsymbol{\alpha}$ in $\mathbb{R}^{m_t} = \mathbb{R}^{m_{t-1}}$

- if $\mu_t(\boldsymbol{x_t}) \leq \mu_o$: The new element ($\boldsymbol{x_t}$) is not similar to any of the dictionary elements. So, it is added to the dictionary ($\boldsymbol{k_{\omega_t}}$ is updated as will be shown below in 2.31), thus $D_t = D_{t-1} \cup \{\boldsymbol{x_t}\}$, $m_t = m_{t-1} + 1$ and we search $\boldsymbol{\alpha}$ in $\mathbb{R}^{m_t} = \mathbb{R}^{m_{t-1}+1}$.

It is clear from the definition of coherence that the smaller value of μ_o will allow more elements to be included in the dictionary.
In order to minimise $J(\boldsymbol{\alpha})$, the instantaneous gradient descent method is used (case of stochastic gradient descent in which $i = t$ is always chosen). Thus,

$$\nabla_t J(\boldsymbol{\alpha}) = -(y_t - \boldsymbol{k_{\omega_t}}(\boldsymbol{x_t})^T \boldsymbol{\alpha}) \boldsymbol{k_{\omega_t}}(\boldsymbol{x_t}).$$

Next, the updates for KLMS in paramteric form with coherence criterion are performed as (which is movement opposite to the gradient evaluated at coefficient vector at previous time):

1. **Case 1:** $\mu_t > \mu_o$:
   The new element $\boldsymbol{x_{t+1}}$ won't be added to the dictionary. $D_t = D_{t-1}$ and $m_t = m_{t-1}$,

   $$\boldsymbol{\alpha_t} = \boldsymbol{\alpha_{t-1}} + \eta e_t \boldsymbol{k_{\omega_t}}, \tag{2.30}$$

   where $e_t = y_t - \boldsymbol{k_{\omega_t}^T \alpha_{t-1}}$ and $\boldsymbol{k_{\omega_t}} = [k(\boldsymbol{x}_{\omega_1}, \boldsymbol{x_t}), \ldots, k(\boldsymbol{x}_{\omega_{m_t}}, \boldsymbol{x_t})]^T$.

2. **Case 2:** $\mu_t \leq \mu_o$:
   The new element $\boldsymbol{x_t}$ will be added to the dictionary and the corresponding coefficient will be initialised as 0 i.e. coefficient vector becomes $\begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ 0 \end{bmatrix}$, $m_t = m_{t-1} + 1$ and $D_t = D_{t-1} \cup \{\boldsymbol{x}_{\omega_{m_t}}\}$ ($\boldsymbol{x_t}$ is denoted by $\boldsymbol{x}_{\omega_{m_t}}$).

   $$\boldsymbol{\alpha_t} = \begin{bmatrix} \boldsymbol{\alpha_{t-1}} \\ 0 \end{bmatrix} + \eta e_t \bar{\boldsymbol{k}}_{\omega_t}, \tag{2.31}$$

   where $e_t = y_t - \boldsymbol{\alpha_{t-1}^T} \bar{\boldsymbol{k}}_{\omega_t}$ and $\bar{\boldsymbol{k}}_{\omega_t} = [k(\boldsymbol{x}_{\omega_1}, \boldsymbol{x_t}), \ldots, k(\boldsymbol{x}_{\omega_{m_{t-1}}}, \boldsymbol{x_t}), k(\boldsymbol{x}_{\omega_{m_t}}, \boldsymbol{x_t})]^T$.

The convergence analysis of this algorithm has been provided in [24].
Note that in this algorithm, the extra growth condition only decides whether to add a new element to the dictionary or not. There is no privilege of removing elements which are added to the dictionary. And that's why the use of an extra penalty function along with

$J(\boldsymbol{\alpha})$ has been proposed in [30]. The use of penalty attracts the smaller coefficients of the dictionary to zero and hence the corresponding elements are removed from the dictionary. In the convergence analysis of this algorithm, it has been shown that its solution converges in mean to the solution of KLMS in parametric form. Let us visit the updating procedure of the KLMS-L1 algorithm. Since, the penalty function being used is non-differentiable, the theory of proximal gradient descent has been used.

2.8.2 Objective and updating procedure

Here the cost function being considered is as follows(following the notation in [30]) ($\lambda \geq 0$):

$$Q(\boldsymbol{\alpha}) = J(\boldsymbol{\alpha}) + \lambda\Omega(\boldsymbol{\alpha}),$$

here $\Omega(\boldsymbol{\alpha})$ is a convex function and $J(\cdot)$ is convex with $\nabla J(\cdot)$ being lipschitz (constant $\frac{1}{\mu}$). At a given point $\boldsymbol{\alpha_n} \in \mathbb{R}^d$, the proximal operator of index μ for above cost will be defined as:

$$\text{prox}_{\lambda\mu\Omega(\cdot)} : \mathbb{R}^d \to \mathbb{R}^d$$

$$\text{prox}_{\lambda\mu\Omega(\cdot)}(\hat{\boldsymbol{\alpha}}_n) = \underset{\boldsymbol{\alpha}\in\mathbb{R}^d}{\text{argmin}}\left\{\lambda\Omega(\boldsymbol{\alpha}) + \frac{1}{2}\|\boldsymbol{\alpha} - \hat{\boldsymbol{\alpha}}_n\|^2\right\} \text{ ,where } \ \hat{\boldsymbol{\alpha}}_n = \boldsymbol{\alpha_n} - \eta\nabla J(\boldsymbol{\alpha_n}).$$

The term inside argmin is obtained using quadratic approximation of $J(\boldsymbol{\alpha})$ at $\boldsymbol{\alpha_n}$(theory of proximal gradient descent) [30].

Two choices of $\Omega(\boldsymbol{\alpha})$ have been suggested in [30]. One is l_1 norm of $\boldsymbol{\alpha}$ and other is its weighted l_1 norm in which larger coefficients get smaller weights and smaller coefficients get larger weights. The second penalty is better since it attracts smaller coefficients more to zero and its effect on larger coefficients is less. From now on, the weighted l_1 norm will be considered in this report. The adaptive weights will be considered as explained in the pseudocode (Algorithm 6).

So, $\Omega(\boldsymbol{\alpha}) = \sum_j w_j|\alpha_j|$, where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_d]^T$. One can show that (in our case $\mu = 1$):

$$\text{prox}_{\lambda\Omega(\cdot)}(\hat{\boldsymbol{\alpha}})(j) = \text{sign}\{\hat{\alpha}_j\}\max\big\{|\hat{\alpha}_j| - \lambda w_j, 0\big\}., \tag{2.32}$$

So, the coefficients satisfying $|\hat{\alpha}_j| \leq \lambda w_j$ will become zero once proximal operator is applied. The algorithm consists of two steps:

- Update coefficient vector according to parametric KLMS using coherence based growth criterion,

- Apply proximal operator to the solution obtained in previous step and then remove those elements which corresponding coefficients become zero.

Suppose $(\boldsymbol{x_{t+1}}, y_{t+1})$ has been observed at $t+1$. Let us revisit KLMS updates in paramteric form with coherence criterion. As before define coherence at time $t+1$ as $\mu_{t+1} = \max_{\boldsymbol{x}_{\omega_i} \in D_t} k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_{t+1}})$. Choose some threshold coherence μ_o. Dictionary at time t is D_t, its size is m_t and its elements are denoted by $\boldsymbol{x}_{\omega_i}$. Then KLMS updates will be as :

1. **Case 1:** $\mu_{t+1} > \mu_o$:
   The new element $\boldsymbol{x_{t+1}}$ won't be added to the dictionary. $D_{t+1} = D_t$ and $m_{t+1} = m_t$

$$\boldsymbol{\alpha_{t+1}} = \boldsymbol{\alpha_t} + \eta e_{t+1} \boldsymbol{k_{\omega_{t+1}}}, \tag{2.33}$$

   where $e_{t+1} = y_{t+1} - \boldsymbol{\alpha_t^T k_{\omega_{t+1}}}$ and $\boldsymbol{k_{\omega_{t+1}}} = [k(\boldsymbol{x}_{\omega_1}, \boldsymbol{x_{t+1}}), \ldots, k(\boldsymbol{x}_{\omega_{m_t}}, \boldsymbol{x_{t+1}})]^T$.

2. **Case 2:** $\mu_{t+1} \leq \mu_o$:
   The new element $\boldsymbol{x_{t+1}}$ will be added to the dictionary, $m_{t+1} = m_t + 1$ and $D_t = D_t \cup \{\boldsymbol{x}_{\omega_{m_{t+1}}}\}$ ($\boldsymbol{x_{t+1}}$ is denoted by $\boldsymbol{x}_{\omega_{m_{t+1}}}$).

$$\boldsymbol{\alpha_{t+1}} = \begin{bmatrix} \boldsymbol{\alpha_t} \\ 0 \end{bmatrix} + \eta e_{t+1} \bar{\boldsymbol{k}}_{\omega_{t+1}}, \tag{2.34}$$

   where $e_{t+1} = y_{t+1} - \boldsymbol{\alpha_t^T} \bar{\boldsymbol{k}}_{\omega_{t+1}}$, and
   $\bar{\boldsymbol{k}}_{\omega_{t+1}} = [k(\boldsymbol{x}_{\omega_1}, \boldsymbol{x_{t+1}}), \ldots, k(\boldsymbol{x}_{\omega_{m_t}}, \boldsymbol{x_{t+1}}), k(\boldsymbol{x}_{\omega_{m_{t+1}}}, \boldsymbol{x_{t+1}})]^T$.

Next, according to the algorithm defined in [30] prox operator (2.32) will be applied to $\boldsymbol{\alpha}_{t+1}$. Then those indices for which coefficients become zero, the corresponding elements are removed from the dictionary. Algorithm 6 presents the pseudo-code for this algorithm. We use adaptive weights at each time.

Input: $\{(\boldsymbol{x}_t, y_t)\}$, $t = 1, 2, \ldots$
Initialisation: choose step size= η, kernel width $\sigma > 0$, Choose threshold $=\mu_o \geq 0$ and initialize dictionary $\mathcal{D}_1 = \{\boldsymbol{x_1}\}$ and coefficient vector $\boldsymbol{\alpha_1} = 0$, $m_1 = 1$
Computation:
while $\boldsymbol{x_{t+1}}$ available do: Compute $\mu_{t+1}(\boldsymbol{x_{t+1}}) = \max_{\boldsymbol{x}_{\omega_i} \in D_t} k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_{t+1}})$

- item **if** $\mu_{t+1}(\boldsymbol{x_{t+1}}) > \mu_o$
  $D_{t+1} = D_t$, $m_{t+1} = m_t$
  Update $\boldsymbol{\alpha_{t+1}}$ according to (2.33).
- **if** $\mu_{t+1}(\boldsymbol{x_t}) \leq \mu_o$
  $D_{t+1} = D_t \cup \{\boldsymbol{x}_{\boldsymbol{\omega}_{\boldsymbol{m_{t+1}}}}\}$, $\boldsymbol{x_t}$ is denoted by $\boldsymbol{x}_{\boldsymbol{\omega}_{\boldsymbol{m_{t+1}}}}$, $m_{t+1} = m_t + 1$
  Update $\boldsymbol{\alpha_{t+1}}$ according to (2.34).

1. Using (9.1) with $w_j = \frac{1}{|\boldsymbol{\alpha_t}(j)| + \epsilon_a}$, (where ϵ_a is small positive number) compute:
   $\boldsymbol{\alpha_{t+1}} = \text{prox}_{\lambda\Omega(\cdot)}(\boldsymbol{\alpha_{t+1}}) = [\boldsymbol{\alpha_{t+1}}(1), \ldots, \boldsymbol{\alpha_{t+1}}(m_{t+1})]^T$
2. $\mathcal{I}_{t+1} = \{j : \boldsymbol{\alpha_{t+1}}(j) = 0\}$
3. $D_{t+1} = D_{t+1} \setminus \{\boldsymbol{x}_{\boldsymbol{\omega_j}}\}_{\omega_j \in \mathcal{I}_{t+1}}$ and $m_{t+1} = m_{t+1} - |\mathcal{I}_{t+1}|$

Algorithm 6: Pseudo code for KLMS with forward-backward splitting (KLMS-L1)

2.8.3 Convergence Analysis

Overall update after applying proximal operator at time $t + 1$ can be written as:

$$\boldsymbol{\alpha_{t+1}} = \boldsymbol{\alpha_t} + \eta e_{t+1} \boldsymbol{k}_{\boldsymbol{\omega_{t+1}}} - \boldsymbol{f_t}, \tag{2.35}$$

where $\boldsymbol{\alpha_t}$ and $\boldsymbol{k}_{\boldsymbol{\omega_{t+1}}}$ correspond to the modifications in 2.34 when required and

$$\boldsymbol{f_t}(j) = \begin{cases} \lambda_t(\text{sign}(\hat{\boldsymbol{\alpha}}(j)) & \text{if } |\hat{\boldsymbol{\alpha}}(j)| \geq \lambda_t \\ \hat{\boldsymbol{\alpha}}(j) & \text{otherwise} \end{cases}$$

where $\hat{\boldsymbol{\alpha}} = \boldsymbol{\alpha_t} + \eta e_{t+1} \boldsymbol{k}_{\boldsymbol{\omega_{t+1}}}$. Let $\boldsymbol{v_t} = \boldsymbol{\alpha_t^0} - \boldsymbol{\alpha_t}$ where $\boldsymbol{\alpha_t^0}$ is the solution to non-regularised problem (KLMS solution). Next, one can prove theorem 2.8.1.

Theorem 2.8.1. *Assuming $\boldsymbol{k}_{\boldsymbol{\omega_t}} \boldsymbol{k}_{\boldsymbol{\omega_t}}^T$ is independent of $\boldsymbol{v_t}$, the regularised KLMS asymptotically converges in mean for any intial condition $\boldsymbol{\alpha_o}$ given step size η is chosen such that*

$$0 < \eta < \frac{2}{\lambda_{max}(\boldsymbol{R_{kk}})},$$

where $\boldsymbol{R_{kk}} = E[\boldsymbol{k}_{\boldsymbol{\omega_t}} \boldsymbol{k}_{\boldsymbol{\omega_t}}^T]$.

2.9 Functional Affine Projection Algorithms (APA)

In section 2.8, KAPA (Kernel APA) algorithms were presented. In that section, the projections were taken along the parametric space and in section 2.8.3, the effect of choice of space on the coefficients update was discussed. In this section, improvements along the line of functional approach ([31], [32]) will be presented along with an interesting result that this functional projection is faster than the parametric one ([33]). This comparison will be discussed using an equivalence between the paramteric and functional approaches ([33], [32]). The explanation will be brief and concise and for detailed version, one should visit the cited references. These algorithms use proof techniques from Projection On Convex Sets (POCS) ([34]) theory to provide theoretical guarantees. First, the PHYPASS algorithm (in papers [31] it has been called $\Phi-$PASS) is presented.

2.9.1 PHYPASS Algorithm

From the discussion in section 2.8.3, in case of functional approaches, only the most recent coefficient gets updated or in other words we move along $k(\cdot, \boldsymbol{x}_{t+1})$ if $\boldsymbol{x}_{t+1}$ is the most recent input. That means if we want to include dictionary growth criterion such as coherence criterion then we will face an inconsistency problem. Because if it excludes the most recent input from the dictionary and we are still following the same update rules then we will move out of the span of dictionary elements as $k(\cdot, \boldsymbol{x}_{t+1})$ is no longer part of the dictionary. This algorithm was developed to account for this problem and later it added a few more improvements. At this point recall a notation used for set of all the functions in our RKHS which output the observed output at t in section 2.8.3:

$$\hat{\Pi}_t := \{f \in \mathcal{H}_K : f(\boldsymbol{x_t}) = \langle f, k(\boldsymbol{x_t}, \cdot)\rangle_{\mathcal{H}_K} = y_t\}.$$

PHYPASS is based on following three considerations (suppose $(\boldsymbol{x}_{t+1}, y_{t+1})$ is the most recent observation pair):

1. **Projection Along Hyperplanes:** Instead of moving along $k(\cdot, \boldsymbol{x}_{t+1})$ (equivalent to moving to $\hat{\Pi}_t$ since $k(\cdot, \boldsymbol{x}_{t+1})$ is its normal), it suggests that we should take projection of $k(\cdot, \boldsymbol{x}_{t+1})$ along the span of the dictionary (denote by $Sp(D_t)$) and then move along this projection (equivalent to moving to $\hat{\Pi}_t \cap Sp(D_t)$).

2. **Data Reusing (parallel projections):** This is functional analog of KAPA with $p > 1$ in section 2.8, i.e, instead of just satisfying the most recent output, we also want to consider recent p outputs. In other words, instead of just moving to $\hat{\Pi}_{t+1}$, we want to move to $\{\hat{\Pi}_i\}_{i=t+2-p}^{t+1}$ in parallel and then take a convex combination of all the p movements to decide the net movement.

3. **Selective Updates:** While taking the projection along the span of dictionary in point 1, we will have to find inverse of a $m_t \times m_t$ matrix (m_t is size of dictionary at t) which is computationally expensive as dictionary grows. So, a few elements (number is fixed and small) are selected from the dictionary which are close to $\boldsymbol{x}_{t+1}$ and the projection is taken along only those bases.

Suppose we want to consider recent p outputs to be satisfied and have decided to project only along s bases from the dictionary. Then at each iteration we do: 1) For each $\{\boldsymbol{x}_i\}_{i=t+2-p}^{t+1}$, find s elements from the dictionary closest to them. Denote each set by $\mathcal{S}_i$ which is set of s dictionary elements closest to $\boldsymbol{x_i}$. 2) Find projection of $k(\cdot, \boldsymbol{x}_{t+1})$ along the $\{\mathcal{S}_i\}_{i=t+2-p}^{t+1}$ separately and move along the projections in parallel to $\{\hat{\Pi}_i\}_{i=t+2-p}^{t+1}$. 3) Take convex combination of the movements and decide the next step. Note that step 2 is same as projecting the current estimate onto intersection of $Sp(\mathcal{S}_i)$ and $\hat{\Pi}_i$. See [31] regarding how to evaluate these projections. Finally, the update equation looks like (f_t is the estimate at iteration t):

$$f_{t+1} = f_t + \lambda_t \Bigg(\sum_{i=t+2-p}^{t+1} w_i P_{Sp(\mathcal{S}_i) \cap \hat{\Pi}_i}(f_t) - f_t \Bigg), \quad P \text{ is the projection operator.} \tag{2.36}$$

There are convergence results (with assumptions on step sizes) for this algorithm using Adaptive Projected Subgradient Method (APSM) (see [31] for details) without the selective strategy. But once the selective strategy is used, the results don't hold.

2.9.2 Equivalence and comparison between functional KAPA and parametric KAPA

First an equivalence between the two approaches will be presented. For both the APA algorithms we will consider the case $p = 1$ (when we only care about the most recent observation pair). For now, the dictionary growth criterion won't be considered which means

the issue of consistency mentioned above won't arise and in this case we will just need to move to the subspace of functions which satisfy the most recent observation. Let us recall the notations for these subspaces again:

$$\Pi_t := \{\boldsymbol{z} \in \mathbb{R}^{m_t} : \boldsymbol{k}_{\boldsymbol{\omega_t}}^T \boldsymbol{z} = y_t\},$$

and

$$\hat{\Pi}_t := \{f \in \mathcal{H}_K : f(\boldsymbol{x_t}) = \langle f, k(\boldsymbol{x_t}, \cdot)\rangle_{\mathcal{H}_K} = y_t\}.$$

The parametric update reads as:

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \lambda_t \bigg(P_{\Pi_{t+1}}(\boldsymbol{\alpha}_t) - \boldsymbol{\alpha}_t \bigg). \tag{2.37}$$

The functional update reads as:

$$f_{t+1} = f_t + \lambda_t \bigg(P_{\hat{\Pi}_{t+1}}(f_t) - f_t \bigg). \tag{2.38}$$

Equation 2.37 is the KNLMS algorithm (note that $\boldsymbol{\alpha}_t$ contains appended zero to match the dimensions, see KNLMS for more details) and its convergence rate is governed by the spread of the eigenvalues of the auto-correlation matrix of its kernelised inputs. Let's denote this auto-correlation matrix by $\boldsymbol{R}$, then at iteration $t+1$, one can show ([33]) that (recall the notations about dictionary and kernel gram matrix)

$$\boldsymbol{R} = E[\boldsymbol{k}_{\boldsymbol{\omega_{t+1}}} \boldsymbol{k}_{\boldsymbol{\omega_{t+1}}}^{\boldsymbol{T}}] \approx \frac{1}{m_{t+1}} \boldsymbol{K}_{\boldsymbol{\omega_{t+1}}}^2. \tag{2.39}$$

Since, the span of dictionary at each stage is finite dimensional, one can arrive at the following lemma ([32]):

Lemma 2.9.1. *$\bigg(Sp(D_t), \langle\cdot,\cdot\rangle_{\mathcal{H}_K}\bigg)$ and $\bigg(\mathbb{R}^{m_t}, \langle\cdot,\cdot\rangle_{\boldsymbol{K}_{\boldsymbol{\omega_t}}}\bigg)$ are isomorphic Hilbert spaces under the correspondence:*

$$Sp(D_t) \ni f := \sum_{i=1}^{m_t} \alpha_i k(\cdot, \boldsymbol{x}_i) \leftrightarrow [\alpha_1, \dots \alpha_{m_t}]^T =: \boldsymbol{\alpha} \in \mathbb{R}^{m_t},$$

where, $\langle \boldsymbol{x}, \boldsymbol{y}\rangle_{\boldsymbol{K}_{\boldsymbol{\omega_t}}} = \boldsymbol{x}^T \boldsymbol{K}_{\boldsymbol{\omega_t}} \boldsymbol{y}$ ($\boldsymbol{K}_{\boldsymbol{\omega_t}}$ is the kernel gram matrix on dictionary elements)

Using Lemma 2.9.1, we can write equation 2.38 as (at iteration $t+1$):

$$\tilde{\boldsymbol{\alpha}}_{t+1} = \tilde{\boldsymbol{\alpha}}_t + \lambda_t \Bigg(P_{\Pi_{t+1}^{new}}(\tilde{\boldsymbol{\alpha}}_t) - \tilde{\boldsymbol{\alpha}}_t \Bigg), \tag{2.40}$$

where $\tilde{\boldsymbol{\alpha}}_t = \boldsymbol{K}_{\boldsymbol{t+1}}^{1/2} \tilde{\boldsymbol{\alpha}}_t$, $\Pi_{t+1}^{new} := \{\tilde{\boldsymbol{\alpha}} \in \mathbb{R}^{m_{t+1}} : \tilde{\boldsymbol{k}}_{\boldsymbol{\omega_{t+1}}}^T \tilde{\boldsymbol{\alpha}} = y_{t+1}\}$ and $\tilde{\boldsymbol{k}}_{\boldsymbol{\omega_{t+1}}} = \boldsymbol{K}_{\boldsymbol{\omega_{t+1}}}^{1/2} \boldsymbol{k}_{\boldsymbol{\omega_{t+1}}}$. Thus, the convergence rate of functional approach will be governed by the eigenvalue spread of its corresponding auto-correlation matrix. Let us denote by $\tilde{\boldsymbol{R}}$, then as before,

$$\boldsymbol{R} = E[\tilde{\boldsymbol{k}}_{\boldsymbol{\omega_{t+1}}} \tilde{\boldsymbol{k}}_{\boldsymbol{\omega_{t+1}}}^{\boldsymbol{T}}] = \boldsymbol{K}_{\boldsymbol{\omega_{t+1}}}^{-1/2} \boldsymbol{R} \boldsymbol{K}_{\boldsymbol{\omega_{t+1}}}^{-1/2} \approx \frac{1}{m_{t+1}} \boldsymbol{K}_{\boldsymbol{\omega_{t+1}}}. \tag{2.41}$$

Comparing equation 2.39 and 2.41, it can be concluded that:

$$\text{condition-number}(\tilde{\boldsymbol{R}}) \approx (\text{condition-number}(\boldsymbol{R}))^{1/2}.$$

It has been shown ([35]) that larger condition number (more eigenvalue spread) leads to slower convergence.

2.9.3 DR-PHYPASS Algorithm

As the name suggests it is Dictionary-Refinement PHYPASS algorithm ([32]). The coefficients are updated according to the parametric version (equation 2.40) with an extra penalty term like the one used in section 2.10 (KLMS-L1). This penalty forces the small coefficients to go to zero.

Chapter 3

Proposed Modification with Simulations and Results

3.1 KLMS-L1 with window approach (KLMS-L1w)

Note that in the KLMS-L1 algorithm every component of coefficient vector is updated at each time (c.f. (2.33) and (2.34)). Our idea is to update only those components for which the corresponding dictionary elements are close to the new input i.e., to consider a window around $\boldsymbol{x_{t+1}}$ and update only those coefficients which corresponding elements lie in this window. Let us denote window at time $t+1$ by w_{t+1}. This window is defined as (for $\epsilon_{t_w} \geq 0$ training window size):

$$\begin{aligned} w_{t+1}(\boldsymbol{x_{t+1}}) &= \{\boldsymbol{x_{\omega_i}} \in D_t : \text{dist}(\boldsymbol{x_{t+1}}, \boldsymbol{x_{\omega_i}}) \leq \epsilon_{t_w}\} && (3.1) \\ &= \Big\{\boldsymbol{x_{\omega_i}} \in D_t : k(\boldsymbol{x_{\omega_i}}, \boldsymbol{x_{t+1}}) \geq \exp\left(\frac{-\epsilon_{t_w}^2}{2\sigma^2}\right) = \epsilon_t\Big\}. && (3.2) \end{aligned}$$

Since, we are already calculating the kernel vector at each time (for coherence), so, we will use (3.2) in our algorithm.

We can also apply penalty in a window only. Let us denote this window by w_p. It is defined

in similar manner as before for (penalty window size $\epsilon_{p_w} \geq 0$):

$$\begin{aligned} w_p(\boldsymbol{x_{t+1}}) &= \{\boldsymbol{x_{\omega_i}} \in D_t : \text{dist}(\boldsymbol{x_{t+1}}, \boldsymbol{x_{\omega_i}}) \leq \epsilon_{p_w}\} && (3.3) \\ &= \Big\{\boldsymbol{x_{\omega_i}} \in D_t : k(\boldsymbol{x_{\omega_i}}, \boldsymbol{x_{t+1}}) \geq \exp\left(\frac{-\epsilon_{p_w}^2}{2\sigma^2}\right) = \epsilon_p\Big\}. && (3.4) \end{aligned}$$

Again due to same reason as mentioned above, we will use (3.4) in our algorithm.
Please note the difference between ϵ_{t_w} and ϵ_t, and the difference between ϵ_{p_w} and ϵ_p. For the explanation of the algorithm, ϵ_t and ϵ_p have been used whereas for the plotting purposes, ϵ_{t_w} and ϵ_{p_w} have been used.

3.1.1 Updating Procedure

The objective is same as KLMS-L1. We will just modify the update step (2.33) and (2.34).
For a vector $\boldsymbol{v}$ and an index set I, $\boldsymbol{v}|_I$ stands for a vector of all the components of $\boldsymbol{v}$ marked by I.
$A \setminus B = A - A \cap B$.
After time t, let the dictionary be D_t and its index set be $\mathcal{J}_t (= \{\omega_1, \ldots, \omega_{m_t}\})$. ϵ_t and ϵ_p correspond to training and penalty window sizes respectively (see (3.2) and (3.4)).

1. **Case 1:** $\mu > \mu_o$:
   The new element $\boldsymbol{x_{t+1}}$ won't be added to the dictionary.
   $e_{t+1} = y_{t+1} - \boldsymbol{\alpha_t^T k_{\omega_{t+1}}}$ and $\boldsymbol{k_{\omega_{t+1}}} = [k(\boldsymbol{x}_{\omega_1}, \boldsymbol{x_{t+1}}), \ldots, k(\boldsymbol{x}_{\omega_{m_t}}, \boldsymbol{x_{t+1}})]^T$
   $T_{t+1} = \{\omega_i \in \mathcal{J}_t : k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_{t+1}}) \geq \epsilon_t\}$ (Training window).
   $P_{t+1} = \{\omega_i \in \mathcal{J}_t : k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_{t+1}}) \geq \epsilon_p\}$ (Penalty window).

$$\begin{aligned} \hat{\boldsymbol{\alpha}}_{t+1}\big|_{T_{t+1}} &= \boldsymbol{\alpha_t}|_{T_{t+1}} + \eta e_{t+1} \boldsymbol{k_{\omega_{t+1}}}\big|_{T_{t+1}} \text{ (components in nbd are updated)}, \\ \hat{\boldsymbol{\alpha}}_{t+1}\big|_{\mathcal{J}_t \setminus T_{t+1}} &= \boldsymbol{\alpha_t}\big|_{\mathcal{J}_t \setminus T_{t+1}} \text{ (remaining components remain same).} \qquad (3.5) \end{aligned}$$

2. **Case 2:** $\mu \leq \mu_o$:
   The new element $\boldsymbol{x_{t+1}}$ will be added to the dictionary, $\bar{m}_t = m_t + 1$ (temporary index included) and
   $\mathcal{J}_t = \mathcal{J}_t \cup \{t+1 = \omega_{\bar{m}_t}\}$, $D_t = D_t \cup \{\boldsymbol{x}_{\omega_{\bar{m}_t}}\}$ ($\boldsymbol{x_{t+1}}$ is denoted by $\boldsymbol{x}_{\omega_{\bar{m}_t}}$).

$e_{t+1} = y_{t+1} - \boldsymbol{\alpha_t^T k}_{\omega_{t+1}}$ and $\boldsymbol{k}_{\omega_{t+1}} = [k(\boldsymbol{x}_{\omega_1}, \boldsymbol{x_{t+1}}), \ldots, k(\boldsymbol{x}_{\omega_{m_t}}, \boldsymbol{x_{t+1}}), k(\boldsymbol{x}_{\omega_{\bar{m}_t}}, \boldsymbol{x_{t+1}})]^T$.
$T_{t+1} = \{\omega_i \in \mathcal{J}_t : k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_{t+1}}) \geq \epsilon_t\}$ (Training window),
$P_{t+1} = \{\omega_i \in \mathcal{J}_t : k(\boldsymbol{x}_{\omega_i}, \boldsymbol{x_{t+1}}) \geq \epsilon_p\}$ (Penalty window), $\boldsymbol{\alpha_t} = \begin{bmatrix} \boldsymbol{\alpha_t} \\ 0 \end{bmatrix}$

$$\begin{aligned} \hat{\boldsymbol{\alpha}}_{t+1}\big|_{T_{t+1}} &= \boldsymbol{\alpha_t}_{T_{t+1}} + \eta e_{t+1} \boldsymbol{k}_{\boldsymbol{\omega_{t+1}}}\big|_{T_{t+1}} \text{ (components in nbd are updated)}, \\ \hat{\boldsymbol{\alpha}}|_{t+1}\big|_{\mathcal{J}_t \setminus T_{t+1}} &= \boldsymbol{\alpha_t}\big|_{\mathcal{J}_t \setminus T_{t+1}} \text{ (remaining components remain same).} \end{aligned} \tag{3.6}$$

3. Proximal operator is now applied in window w_p and finally we get (prox can be calculated using 2.32):

$$\begin{aligned} \tilde{\boldsymbol{\alpha}}_{\boldsymbol{t+1}}|_{P_{t+1}} &= \text{prox}_{\lambda\Omega(\cdot)}(\hat{\boldsymbol{\alpha}}_{\boldsymbol{t+1}}|_{P_{t+1}}) \text{ (prox operator in the nbd only)}, \\ \tilde{\boldsymbol{\alpha}}_{t+1}\big|_{\mathcal{J}_t \setminus P_{t+1}} &= \hat{\boldsymbol{\alpha}}_{t+1}\big|_{\mathcal{J}_t \setminus P_{t+1}}. \end{aligned} \tag{3.7}$$

4. Next find the components which are zero and remove the corresponding elements:
$Z_{t+1} = \{\omega_i \in P_{t+1} : \tilde{\boldsymbol{\alpha}}_i = 0\}$,
$\mathcal{J}_{t+1} = \mathcal{J}_t \setminus Z_{t+1}$, $D_{t+1} = D_t \setminus \{\boldsymbol{x}_{\boldsymbol{\omega_i}}\}_{\omega_i \in Z_{t+1}}$, and $m_{t+1} = |\mathcal{J}_{t+1}|$

$$\boldsymbol{\alpha_{t+1}} = \tilde{\boldsymbol{\alpha}}_{\boldsymbol{t+1}}|_{J_{t+1}} \text{ vector at only non-zero positions.}$$

3.2 Experiments and Results

A comparison between following algorithms (approaches) will be reported in this section:

1. **Approach 1**: KLMS-L1-(org) : This the original method [30] in which training and penalty is performed overall i.e., $\epsilon_t = \epsilon_p = 0$. We will refer to it as approach 1. In the plots, it has been depicted as 'org'.

2. **Approach 2**: KLMS-L1w-(tt-pw): In this approach, update is performed overall i.e., $\epsilon_t = 0$ and penalty is applied in a window i.e., $\epsilon_p > 0$. We will refer to it as approach 2. In the plots, it has been depicted as 'tt-pw'.

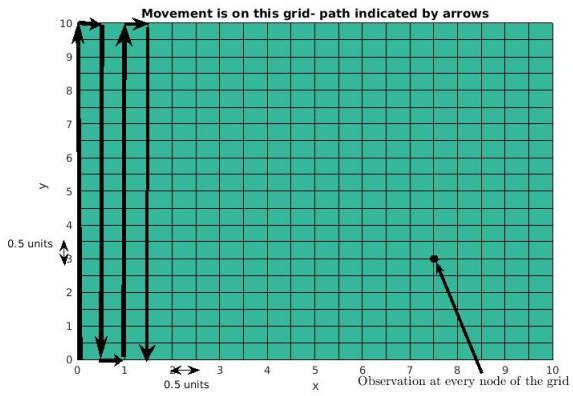


Figure 3.1: Training Grid

3. **Approach 3**: KLMS-L1w-(tw-po): In this approach, update is performed in a window i.e., $\epsilon_t > 0$ and penalty is applied overall i.e., $\epsilon_p = 0$. We will refer to it as approach 3. In the plots, it has been depicted as 'tw-po'.

4. **Approach 4**: KLMS-L1w-(ww): In this approach both training and penalty are applied within a window i.e., $\epsilon_t > 0$ and $\epsilon_p > 0$. We will refer to it as approach 4. In the plots, it has been depicted as 'ww'.

3.2.1 Simulation

The true function being used is $f(x, y) = 10sin(x) + 5cos(y)$. The input space cosidered here is $[0, 10] \times [0, 10] \subset \mathbb{R}^2$. This domain has been divided into grids. The sensor starts at origin and moves along the path indicated in the Figure 3.1, taking observations at every 0.5 units. This grid has total 441 points.

After the sensor reached (10,10), we obtained the estimate ($\hat{f}$) for the whole region and then we found the Total mean squares error(TMSE) on a finer grids(FGD) which has 0.25 units separation between nodes. This finer grid has total 1681 points.

$$\text{TMSE} = log_{10}\Big(\frac{1}{1681}\sum_{i=1}^{1681}(\hat{f}\big(\boldsymbol{x_i}\big) - y_i\big)^2\Big).$$

I have compared TMSE for different approaches for three different values of kernel width (0.5, 0.8 and 1.8). Every method will return $\hat{f}(\cdot)$ which is expansion in terms of dictionary elements. We have compared dictionary sizes for different approaches also for different values

of kernel width (0.5, 0.8 and 1.8). In subsequent subsections, I have plotted the comparisons for different kernel widths separately.

3.2.2 Results for $\sigma = 0.5$

In this subsection Fig 3.2 - Fig 3.7 represent the training comparisons have been plotted for kernel width 0.5 units.
TMSE for approach 1 and approach 3 vs the training window size (ϵ_{t_w}) has been plotted in Fig 3.2.
TMSE for approach 2 for different training (ϵ_{t_w}) and penalty window (ϵ_{t_p}) sizes have been plotted in Fig 3.3. Note that this approach is independent of ϵ_{t_w} because the training is done using the entire data and only penalty is applied within a window.
TMSE for approach 4 for different training and penalty window sizes have been plotted in Fig 3.4.
Dictionary sizes for approach 1 and approach 3 vs the training window size (ϵ_{t_w}) has been plotted in Fig 3.5.
Dictionary sizes for approach 2 for different training (ϵ_{t_w}) and penalty window (ϵ_{t_p}) sizes have been plotted in Fig 3.6. Note that this approach is independent of ϵ_{t_w} because the training is done using the entire data and only penalty is applied within a window.
Dictionary sizes for approach 4 for different training and penalty window sizes have been plotted in Fig 3.7.

3.2.3 Results for $\sigma = 0.8$

In this subsection Fig 3.8 - Fig 3.13 represent the training comparisons have been plotted for kernel width 0.8 units.
TMSE for approach 1 and approach 3 vs the training window size (ϵ_{t_w}) has been plotted in Fig 3.8.
TMSE for approach 2 for different training (ϵ_{t_w}) and penalty window (ϵ_{t_p}) sizes have been plotted in Fig 3.9. Note that this approach is independent of ϵ_{t_w} because the training is performed overall and only penalty is applied within a window.
TMSE for approach 4 for different training and penalty window sizes have been plotted in

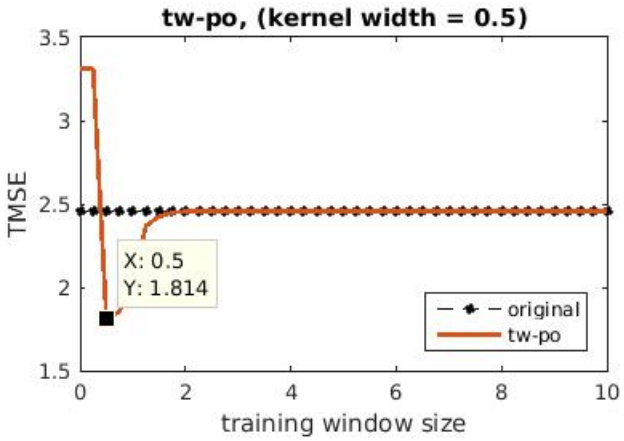


Figure 3.2: TMSE vs training window size for original and approach 3 and original

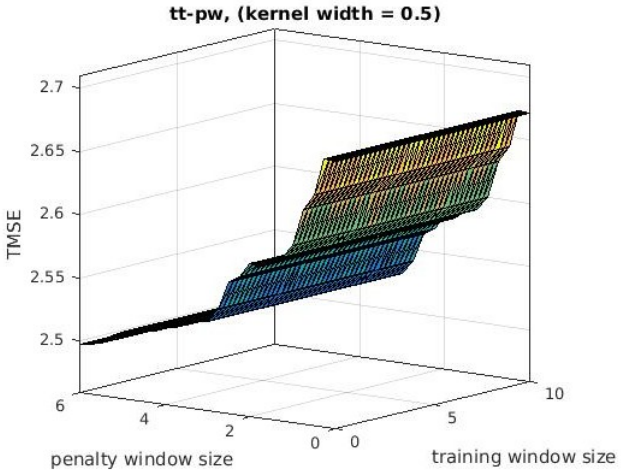


Figure 3.3: TMSE vs training and penalty window sizes for approach 2

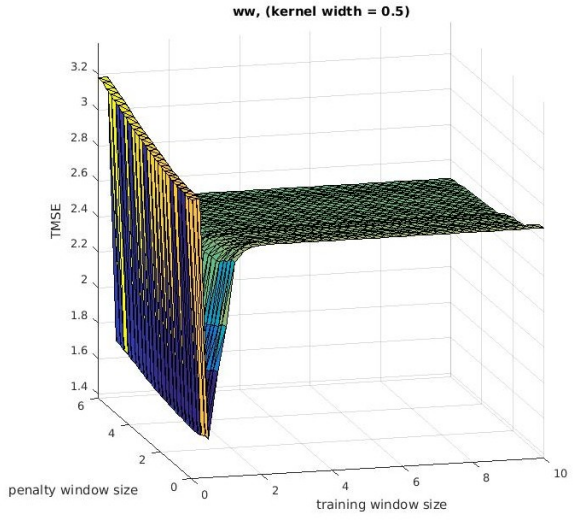


Figure 3.4: TMSE vs training and penalty window sizes for approach 4

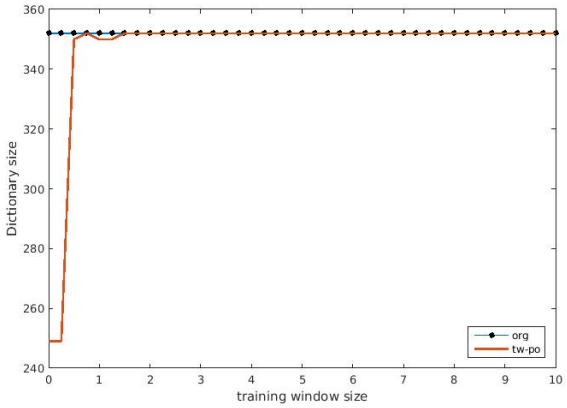


Figure 3.5: dictionary size vs training window size for approach 3 and original

Fig 3.10.

Dictionary sizes for approach 1 and approach 3 vs the training window size (ϵ_{t_w}) has been plotted in Fig 3.11.

Dictionary sizes for approach 2 for different training (ϵ_{t_w}) and penalty window (ϵ_{t_p}) sizes have been plotted in Fig 3.12. Note that this approach is independent of ϵ_{t_w} because the training is performed overall and only penalty is applied within a window.

Dictionary sizes for approach 4 for different training and penalty window sizes have been

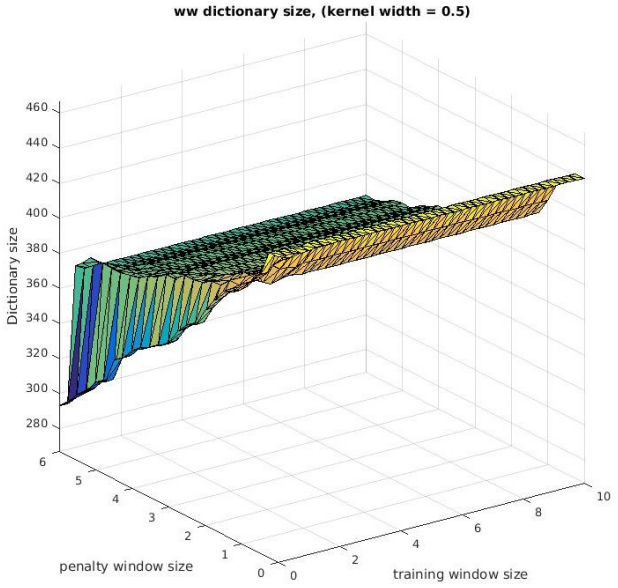


Figure 3.6: Dictionary size vs training and penalty window sizes in approach 2

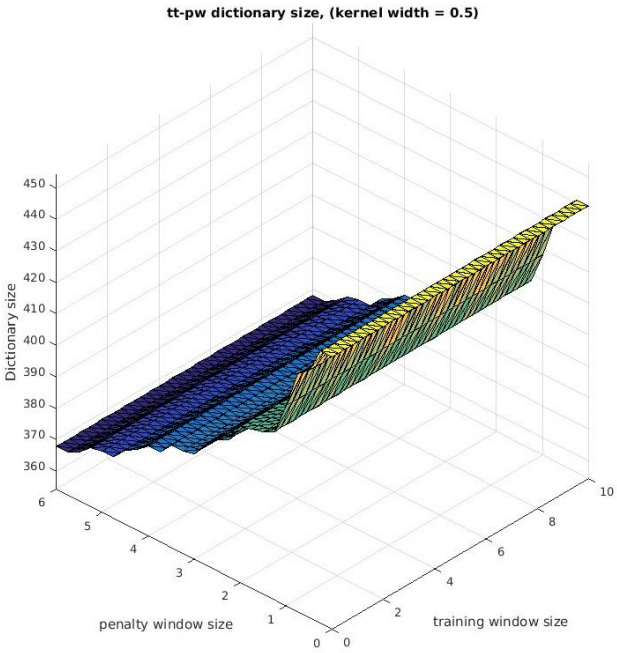


Figure 3.7: Dictionary size vs training and penalty window sizes in approach 4

plotted in Fig 3.13.

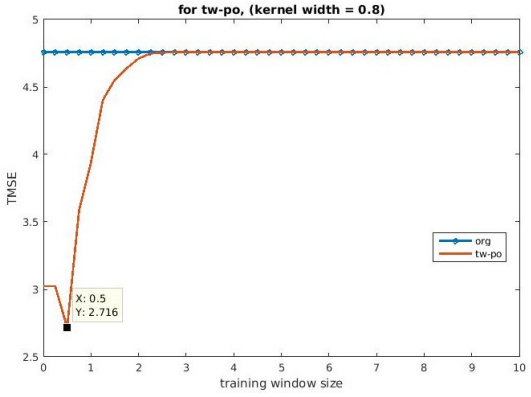


Figure 3.8: TMSE vs training window size for original and approach 3 and original

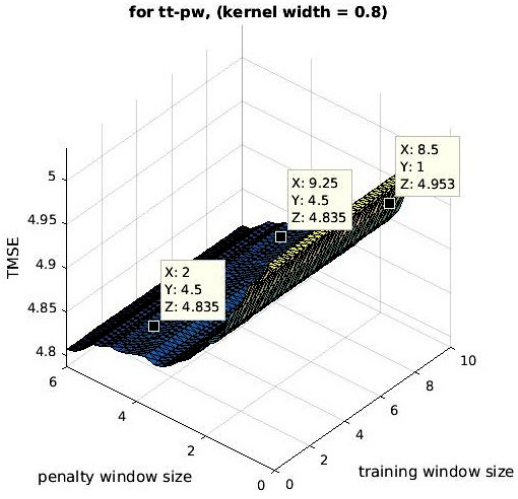


Figure 3.9: TMSE vs training and penalty window sizes for approach 2

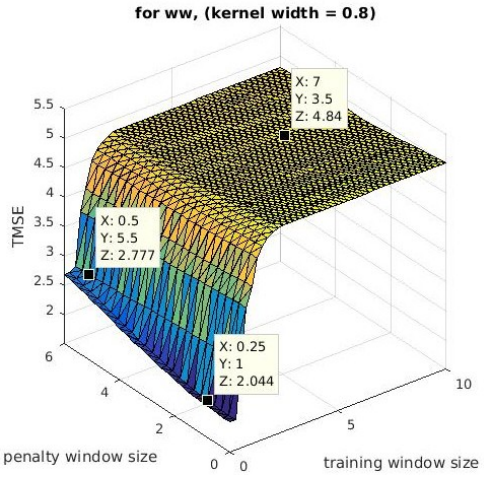


Figure 3.10: TMSE vs training and penalty window sizes for approach 4

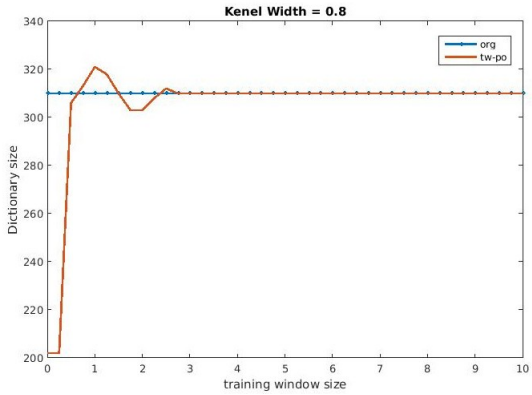


Figure 3.11: Dictionary size vs training window size in approach 3 and original

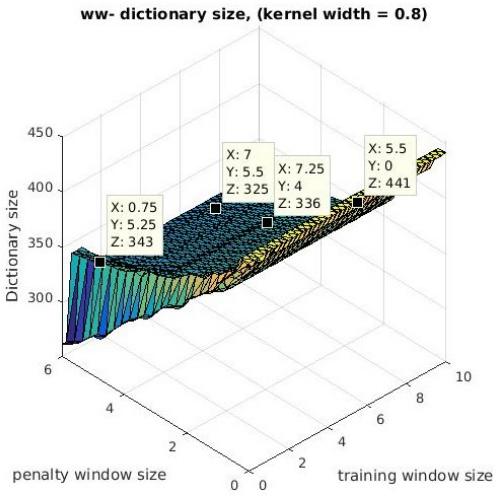


Figure 3.12: Dictionary size vs training and penalty window sizes in approach 2

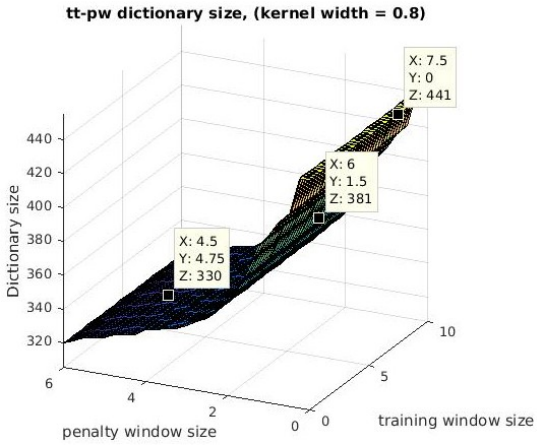


Figure 3.13: Dictionary size vs training and penalty window sizes in approach 4

3.2.4 Results for $\sigma = 1.8$

In this subsection Fig 3.14 - Fig 3.19 represent the training comparisons have been plotted for kernel width 1.8 units.

TMSE for approach 1 and approach 3 vs the training window size (ϵ_{t_w}) has been plotted in Fig 3.14.

TMSE for approach 2 for different training (ϵ_{t_w}) and penalty window (ϵ_{t_p}) sizes have been plotted in Fig 3.15. Note that this approach is independent of ϵ_{t_w} because the training is

performed overall and only penalty is applied within a window.
TMSE for approach 4 for different training and penalty window sizes have been plotted in Fig 3.16.
Dictionary sizes for approach 1 and approach 3 vs the training window size (ϵ_{t_w}) has been plotted in Fig 3.17.
Dictionary sizes for approach 2 for different training (ϵ_{t_w}) and penalty window (ϵ_{t_p}) sizes have been plotted in Fig 3.18. Note that this approach is independent of ϵ_{t_w} because the training is performed overall and only penalty is applied within a window.
Dictionary sizes for approach 4 for different training and penalty window sizes have been plotted in Fig 3.19.

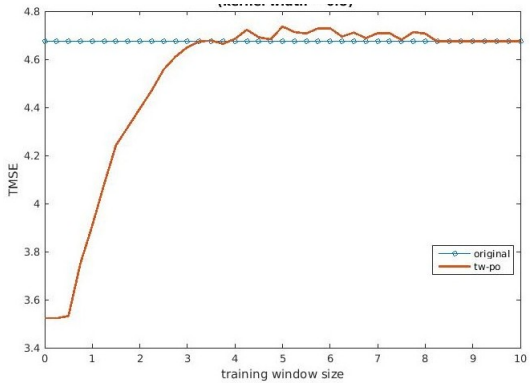


Figure 3.14: TMSE vs training window size for original and approach 3 and original

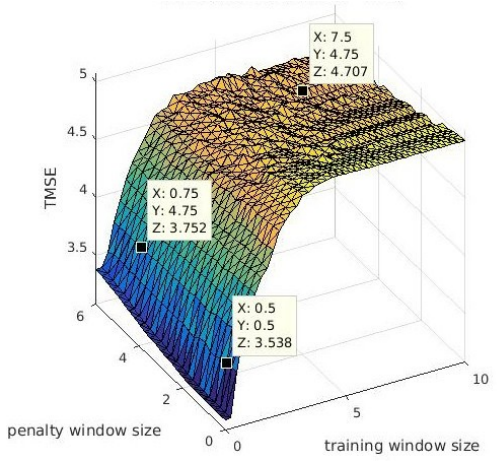


Figure 3.15: TMSE vs training and penalty window sizes for approach 2

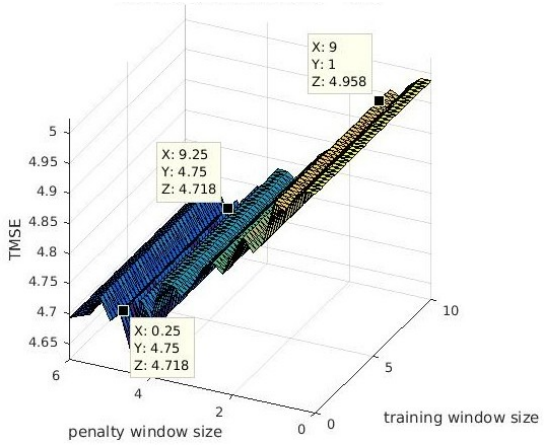


Figure 3.16: TMSE vs training and penalty window sizes for approach 4

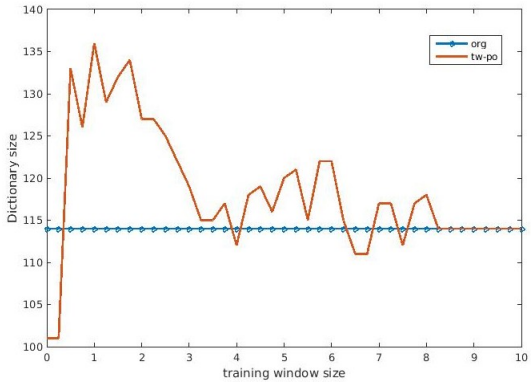


Figure 3.17: Dictionary size vs training window size in approach 3 and original

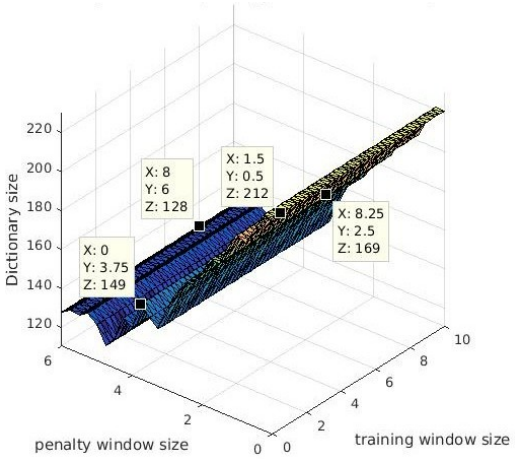


Figure 3.18: Dictionary size vs training and penalty window sizes in approach 2

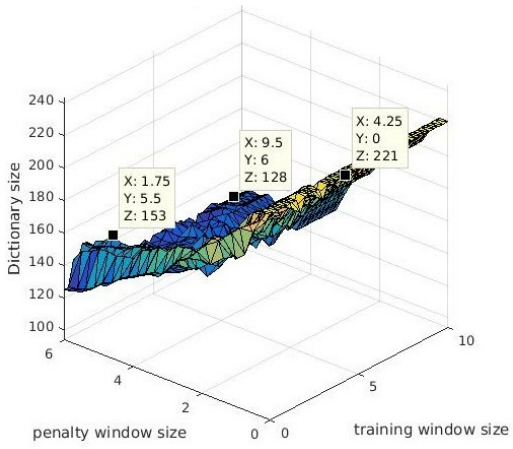


Figure 3.19: Dictionary size vs training and penalty window sizes in approach 4

3.2.5 Conclusion

- For each of the kernel widths, approach 3 (training in window and applying penalty overall) is working better than original approach both in terms of attaining minimum TMSE and minimum dictionary size for some values of training window size.
- Approach 4 (training and penalty both in windows) is attaining better TMSE than approach 3 and the original method but the sparsification is worse.
- Since, approach 3 and approach 4 are attaining TMSE better than original approach, it may be attributed to the training within a window strategy.

- More sparsification in 3 than 4 is can be explained by the fact that in every time, in approach 3 we are allowing the small coefficients throughout the dictionary to go to zero and hence removing them but in approach 4, we are allowing only those small coefficients which are within a window to go to zero and hence some small coefficients still remain in the dictionary.

- For every kernel width, the performances of original (approach 1) and approach 2 are same after a while. The reason behind such similarity is that the kernel itself acts like a window. The lesser the kernel width, lesser will be the affect a dictionary element on its neighboring elements as gaussian kernel decays with distance. So note that for smaller kernel width the similarity is attained quickly but for larger kernel widths, the similarity is attained after a while.

3.3 MKNLMS with window approach and modified dictionary construction

In the previous section, we observed that the window approach works better. But in the case of single kernel methods, it was not clear how to select proper kernel width. The multi-kernel approach (section 2.9) provides a way to accommodate multiple kernels. The growing step of the dictionary construction in every algorithm discussed so far takes only the input observations into consideration. Since output values are prone to error, they had no role in the growth of the dictionary. The coherence criterion used in multi-kernel approach (equation 2.24) is always dominated by the largest kernel width and this results in poor dictionary construction. Suppose the coherence of the new input w.r.t. an element from the current dictionary is greater than the threshold using the largest kernel i.e, the new input is close to the corresponding element w.r.t. the largest kernel then it is not added to the dictionary. Next, consider a case in which the output at the new input varied significantly from the output at the corresponding closest element from the dictionary, then discarding the new entry will lead to loss of significant information. So, in the case when fluctuation is more, we should decide coherence using a smaller kernel width. In this section, this idea has been exploited i.e, instead of taking maximum over all the kernels in equation (2.24), we choose a kernel width according to the level of fluctuation and evaluate coherence using that particular choice of kernel only. Along with this modified version of coherence criterion, we

use window approach to update coefficients within a window only. In section 3.3, the two methods of multi-kernel methods were mentioned which are MKNLMS-CS and MKNLMS-BT. MKNLMS-CS uses the coherence criterion to control the growth of dictionary without any pruning strategy. On the other hand, MKNLMS-BT doesn't have any growth control condition but it does pruning of existing dictionary using extra penalty function. In this section, we have combined the three ideas: to have control over growth using modified coherence, control over pruning using same penalty function and update coefficients only within a window. In the subsequent subsections, explanation for the algorithm and the results have been provided along with the discussion.

3.3.1 Updating Procedure

The cost function which is needed to be minimised is same as the one used in MKNLMS-BT (see equation 2.27). At every iteration the algorithm follows three steps: 1) Control the entry of new input into the dictionary via the modified coherence criterion. 2) Update the coefficients locally around the new input using gradient descent. 3) Apply prox operator (2.29) to make the insignificant coefficients zero and remove the corresponding elements from the dictionary.

At first, the modified coherence criterion will be explained. Our target at iteration $t+1$ is to consider a small (or large) kernel-width when the fluctuation is more (or less). In order to carry out such a selection, we have absolute value of slope (denoted by $|s_{t+1}|$) evaluated between the observation pairs $(\boldsymbol{x}_{t+1}, y_{t+1})$ and $(\boldsymbol{x}_t, y_t)$ which is given by

$$|s_t| = \left| \frac{y_{t+1} - y_t}{\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_2} \right|. \tag{3.8}$$

So, the basic idea is that a large (or small) value of slope which corresponds to a large (or small) fluctuation should lead to the selection of a smaller (or larger) kernel-width. We considered the Gaussian function centred around origin given by:

$$f(\boldsymbol{x}) = exp\bigg(- \frac{\|\boldsymbol{x}\|^2}{2\sigma^2} \bigg), \quad \text{where } \boldsymbol{x} \in \mathbb{R}^2.$$

Then the absolute value of the gradient at $\|\boldsymbol{x}\| = \sigma$, is $\frac{exp(-1/2)}{\sigma}$.

If we place the Gaussian kernel (with kernel-width σ) centered at the new input then we

will get the same expression for the absolute value of gradient at distance σ away from the new input. We will find the value of σ for which the absolute value of the slope m evaluated between the observation pairs $(\boldsymbol{x}_{t+1}, y_{t+1})$ and $(\boldsymbol{x}_t, y_t)$ becomes same as $\frac{exp(-1/2)}{\sigma}$. This desired equality leads to a value of kernel-with which is given by

$$\sigma = \frac{exp(-1/2)}{|s_{t+1}|}. \tag{3.9}$$

At the same time, we wanted to accommodate very few kernels (which is similar to saying very few kernel-widths) in our list of kernels and that's why we divided slopes into a few intervals and assigned a kernel-width for each interval of slope values. Using equation 3.9, we have following table:

values of $\lvert s_{t+1}\rvert$	2.02	1.01	0.36	0.17	0.09
choice of σ	0.3	0.6	1.7	3.5	7
rounded off θ_{t+1} in degrees	65	45	20	10	5

Note that larger θ_{t+1} is equivalent to large fluctuation at $t+1$ iteration. Our kernel-width selection criteria is as follows:

Calculate the absolute value of slope $|s_{t+1}|$ using equation 3.8 and find the corresponding θ_{t+1} in degrees. Let us denote the chosen kernel-width by σ^*. Then,

$$\sigma^*_{t+1} = \begin{cases} 0.3, & \text{if } \theta_{t+1} \geq 65 \\ 0.6, & \text{if } 65 > \theta_{t+1} \geq 45 \\ 1.7, & \text{if } 45 > \theta_{t+1} \geq 20 \\ 3.5, & \text{if } 20 > \theta_{t+1} \geq 10 \\ 7, & \text{if } \theta_{t+1} < 10 \end{cases} \tag{3.10}$$

So far in this section we have used three new notations namely $|s_{t+1}|$, θ_{t+1} and σ^*_{t+1} for slope, for corresponding angle in degrees and chosen kernel width respectively at iteration $t+1$. For the remaining part, we will use the notations used in section 2.9. At iteration $t+1$, after finding σ^*_{t+1}, we define the modified coherence condition (for a user-defined threshold μ_o) as:

$$\|\boldsymbol{K_{t+1}}\|_{max} := \max_{\omega_j \in \mathcal{J}_t} |k_{\sigma^*_{t+1}}(\boldsymbol{x_{t+1}}, \boldsymbol{x}_{\omega_j})| \leq \mu_o. \tag{3.11}$$

Thus, if the modified coherence criterion is satisfied, the most recent input is not added to the dictionary and the coefficient matrix doesn't change at this stage. But when this criterion

is not satisfied, the new element is added to the dictionary and a column of zeros is added to the coefficient matrix. Then the gradient step will be taken. As already mentioned, we will define a local neighborhood around the most recent input and update coefficients of the dictionary which correspond to the chosen kernel and lie within the defined neighborhood. Note that in the case of multi-kernel approaches we have a coefficient matrix instead of a coefficient vector and each column corresponds to each kernel (or kernel-width in our case) in the list. So, we will update the column corresponding to chosen kernel only i.e, only a few (see below) elements of $(\sigma^*_{t+1})^{th}$ column of coefficient matrix $\boldsymbol{H}_t$ will be updated. Next, we will define the window i.e, a local neighborhood around $\boldsymbol{x}_{t+1}$ similarly as we did it in section 3.1 (see equation 3.2). So, the window at $\boldsymbol{x}_{t+1}$ for a window size ϵ is given by:

$$
\begin{aligned}
w_{t+1}(\boldsymbol{x_{t+1}}) &= \{\boldsymbol{x}_{\boldsymbol{\omega_i}} \in D_t : \text{dist}(\boldsymbol{x_{t+1}}, \boldsymbol{x}_{\boldsymbol{\omega_i}}) \leq \epsilon\} && (3.12)\\
&= \Big\{\boldsymbol{x}_{\boldsymbol{\omega_i}} \in D_t : k_{\sigma^*_{t+1}}(\boldsymbol{x}_{\boldsymbol{\omega_i}}, \boldsymbol{x_{t+1}}) \geq \exp\left(\frac{-\epsilon^2}{2\sigma^2}\right)\Big\}. && (3.13)
\end{aligned}
$$

As already mentioned in the section 3.1, note that we don't have to do any extra computation to find out w_{t+1} since we have already computed the elements of $\boldsymbol{k}_{\sigma^*_{t+1}}$ while evaluating the modified coherence criterion in equation 3.11. So, at every iteration we find the elements from the dictionary which lie in the window that we defined in equation 3.13 and the corresponding coefficients from the $(\sigma^*_{t+1})^{th}$ column of coefficient matrix will be updated. Next, the prox operator is applied to the updated coefficient matrix which forces the rows with small l_1-norm to become zero and eventually the corresponding elements are discarded from the dictionary. A pseudo-code with relevant equations is presented in remaining of this subsection. It will closely follow the terms and notations used in section 2.9.

In the initialization, choose step size η which will be used as proximity index (μ in equation 2.29), set of kernels (kernels-widths) $\mathcal{M} = \{k_1, \ldots, k_M\}(\sim \{0.3, 0.6, 1.7, 3.5, 7\})$, regularising parameter $\lambda \geq 0$, a small coefficient $\rho > 0$ (this prevents the denominator from vanishing in the gradient step), a window size $\epsilon \geq 0$, a threshold $\mu_o \geq 0$, weight vector for prox operator $\boldsymbol{w_{-1}} = [\,]$ (empty vector at the beginning which will be adaptively updated).

For $t = 0, 1, 2, ...$, when $(\boldsymbol{x}_{t+1}, y_{t+1})$ is observed (please see the comment after step 3 for $t = 0$ case), we calculate the slope $|s_{t+1}|$ using equation 3.8 and we select the kernel width σ^*_{t+1} using equation 3.10. We also need the weight vector for prox operator which will define at each iteration as: $\boldsymbol{w_t} \in \mathbb{R}^{1\times m_t}$ given by

$$
\boldsymbol{w_t}(i) = \frac{1}{\rho + \|\boldsymbol{H_t}(i^{th}\ column)\|_1}.
$$

Next, we check the modified coherence condition (rewriting equation 3.11):

$$\|\boldsymbol{K_{t+1}}\|_{max} := \max_{\omega_j \in \mathcal{J}_t} |k_{\sigma^*_{t+1}}(\boldsymbol{x_{t+1}}, \boldsymbol{x}_{\omega_j})| \leq \mu_o.$$

1. At this stage one of the two cases will happen:

   - **Case 1:** If the above inequality holds (the modified coherence condition holds), $\boldsymbol{x}_{t+1}$ is not added to the dictionary. Next, we find the indices of elements from the dictionary which lie in the ϵ window (see 3.13).
   $T_{t+1} = \left\{ i \in \mathcal{J}_t : k_{\sigma^*_{t+1}}(\boldsymbol{x_{\omega_i}}, \boldsymbol{x_{t+1}}) \geq \exp\left(\frac{-\epsilon^2}{2\sigma^2}\right) \right\}$. **For a matrix A, let $A(I, J)$ denote the submatrix with all the rows indexed by I of all the columns indexed by J.** With a little abuse of notations, assume that σ^* also denotes its corresponding index in set of kernel widths M. Then,

   $$\bar{\boldsymbol{H}}_{\boldsymbol{t}}(T_{t+1}, \sigma^*_{t+1}) = \boldsymbol{H_t}(T_{t+1}, \sigma^*_{t+1}) + \eta \frac{y_t - \langle \boldsymbol{H_t}, \boldsymbol{K_t} \rangle}{\rho + \|\boldsymbol{K_t}\|^2} \boldsymbol{K_t}(T_{t+1}, \sigma^*_{t+1}),$$

   $$\bar{\boldsymbol{H}}_{\boldsymbol{t}}(\mathcal{J}_t \setminus T_{t+1}, M \setminus \sigma^*_{t+1}) = \boldsymbol{H_t}(\mathcal{J}_t \setminus T_{t+1}, M \setminus \sigma^*_{t+1}).$$

   - **Case 2:** The modified coherence criterion doesn't hold. Then, $\boldsymbol{x}_{t+1}$ is added to the dictionary for now. Note that in algorithmic notations $a = a_{changed}$ means a will be replaced with $a_{changed}$.

   $$\mathcal{J}_t = \mathcal{J}_t \cup \{t+1\} \text{ and } \; m_t = m_t + 1,$$

   $$\boldsymbol{H_t} = \begin{bmatrix} \boldsymbol{H_t} & \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{M \times (m_t+1)},$$

   $$\boldsymbol{K_t} = \begin{bmatrix} \boldsymbol{K_t} & \boldsymbol{k_{t+1}} \end{bmatrix} \in \mathbb{R}^{M \times (m_t+1)},$$

   $$\boldsymbol{w_t} = \begin{bmatrix} \boldsymbol{w_t} & 1 \end{bmatrix} \in \mathbb{R}^{1 \times (m_t+1)},$$

   where, $\boldsymbol{k}_{t+1} = [k_1(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}}), \ldots, k_M(\boldsymbol{x_{t+1}}, \boldsymbol{x_{t+1}})]^T$.
   Then, we again find the elements within the ϵ window and in this case $\boldsymbol{x}_{t+1}$ will also be considered.
   $T_{t+1} = \left\{ i \in \mathcal{J}_t : k_{\sigma^*_{t+1}}(\boldsymbol{x_{\omega_i}}, \boldsymbol{x_{t+1}}) \geq \exp\left(\frac{-\epsilon^2}{2\sigma^2}\right) \right\}$.

   $$\bar{\boldsymbol{H}}_{\boldsymbol{t}}(T_{t+1}, \sigma^*_{t+1}) = \boldsymbol{H_t}(T_{t+1}, \sigma^*_{t+1}) + \eta \frac{y_t - \langle \boldsymbol{H_t}, \boldsymbol{K_t} \rangle}{\rho + \|\boldsymbol{K_t}\|^2} \boldsymbol{K_t}(T_{t+1}, \sigma^*_{t+1}),$$

$$\bar{\boldsymbol{H}}_{\boldsymbol{t}}(\mathcal{J}_t \setminus T_{t+1}, M \setminus \sigma^*_{t+1}) = \boldsymbol{H}_{\boldsymbol{t}}(\mathcal{J}_t \setminus T_{t+1}, M \setminus \sigma^*_{t+1}).$$

2. Next, we apply prox operator to $\bar{\boldsymbol{H}}_{\boldsymbol{t}}$ (see 2.29 for definition). The parameters required for this operator are λ, $\mu(=\eta)$ and $\boldsymbol{w_t}$ which have already been defined. So,

$$\hat{\boldsymbol{H}}_{t+1} = \text{prox}(\bar{\boldsymbol{H}}_{\boldsymbol{t}}).$$

3. Next, we remove those elements from the dictionary which corresponding column in the coefficient matrix ($\hat{\boldsymbol{H}}_{t+1}$) have l_1-norm zero.
$Z_{t+1} = \left\{ i \in \mathcal{J}_t : \|\hat{\boldsymbol{H}}_{t+1}(i^{th}column)\|_1 = 0 \right\}$,
$\mathcal{J}_{t+1} = \mathcal{J}_t \setminus Z_{t+1}$, $D_{t+1} = D_t \setminus \{\boldsymbol{x_{\omega_i}}\}_{\omega_i \in Z_{t+1}}$, and $m_{t+1} = |\mathcal{J}_{t+1}|$ And finally,

$$\boldsymbol{H_{t+1}} = \hat{\boldsymbol{H}}_{t+1}(M, \mathcal{J}_{t+1}) \in \mathbb{R}^{M \times m_{t+1}}.$$

Remark 1: One should note that when the first observation pair arrives, we directly go to Case 2 and select first kernel-width and the corresponding weight for prox operator is just 1.
Remark 2: The effect of coherence threshold μ_o on the dictionary size has been discussed multiple times in earlier chapter.
Remark 3: If the regularizing parameter λ is increased, the dictionary size will decrease. The value of this parameter is based on prior knowledge and was taken to be very small in our experiment.
Remark 4: The step size η must lie in $(0, 2]$ for single kernel least mean squares algorithm to ensure the convergence. For the multi-kernel approach, we are not aware of such a guarantee. However, we have studied the effect of its different values which will provide insight into the observation that the window approach works better than MKNLMS-CS and MKNLMS-BT.

3.3.2 Simulations and Results

The input domain and the vehicle movement will be same as described in section 3.2.1. For completion purposes the description will be repeated here.
The input space cosidered here is $[0, 10] \times [0, 10] \subset \mathbb{R}^2$. This domain has been divided into grids. The sensor starts at origin and moves along the path indicated in the Figure 3.20, taking observations at every 0.5 units. This grid has total 441 points. After the sensor

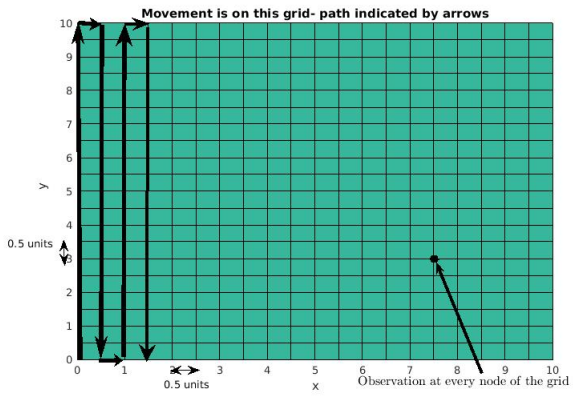


Figure 3.20: Training Grid

reached (10,10), we obtained the estimate ($\hat{f}$) for the whole region and then we found the Total Mean Squares Error (TMSE) on a finer grids (FGD) which has 0.25 units separation between nodes. This finer grid has total 1681 points.

$$\text{TMSE} = log_{10}\Big(\frac{1}{1681}\sum_{i=1}^{1681}(\hat{f}\big(\boldsymbol{x_i}\big) - y_i\big)^2\Big)$$

We will compare the window approach which we call MKNLMS-new with the existing multi-kernel approaches MKNLMS-CS and MKNLMS-BT. We have compared both the TMSE (defined above) and the dictionary sizes. The results related to three functions will be presented next. In case of each function, the results will be divided into three stages:

- In the first stage we keep the same parameters in the algorithm for each of the three methods. We present the comparison of TMSE and Dictionary sizes w.r.t. to different window sizes.

- Step size plays a major role in the way we have explained the results. So, in the second stage we will present results w.r.t. step sizes. For each step-size, minimum TMSE and the corresponding dictionary size and the window size will be compared.

- From the second stage, we will choose a good value for the step-size for MKNLMS-new and the best value for the step-size for MKNLMS-CS and MKNLMS-BT. Then, again TMSE and dictionary sizes will be compared for different window sizes using the chosen values of step size.

Algorithm	step-size (η)	regularisation parameter(λ)	coherence threshold (μ_o)	ρ
MKNLMS-CS	0.1	5×10^{-4}	0.95	10^{-5}
MKNLMS-BT	0.1	5×10^{-4}	0.95	10^{-5}
MKNLMS-new	0.1	5×10^{-4}	0.95	10^{-5}

Table 3.1: Values of the parameters for Fig 3.21 and Fig 3.22

At each stage, a table containing values of the parameters used will be presented. At all the stages, we have kept the kernel set same as described in the updating procedure above. The discussion section will contain the reason for the above-mentioned three stages and the explanation of the results.

Results for Function 1:

The first function we consider is a Gaussian type function. This is inspired by the example of function used in [32].
Let $\boldsymbol{x} \in [0, 10] \times [0, 10]$ be the input. Let $c_1 = [4,\ 8.5]^T$, $c_2 = [6,\ 3.5]^T$, $c_3 = [8,\ 6]^T$, $c_4 = [3.5,\ 1]^T$ be vectors which lie in our domain. The function is defined as:

$$\begin{aligned} f_1(\boldsymbol{x}) = 10 + exp\bigg(-\frac{\|\boldsymbol{x} - c_1\|_2^2}{2 * 3^2}\bigg) + 1.5exp\bigg(-\frac{\|\boldsymbol{x} - c_2\|_2^2}{2 * 3^2}\bigg) \\ -0.5exp\bigg(-\frac{\|\boldsymbol{x} - c_3\|_2^2}{2 * 3^2}\bigg) - 1.25exp\bigg(-\frac{\|\boldsymbol{x} - c_4\|_2^2}{2 * 3^2}\bigg) \end{aligned}. \tag{3.14}$$

As mentioned earlier, in the first stage we will use same value of the parameters for all three methods. The values of the parameters at this stage have been picked up from the previous papers i.e., the values which are considered standard in this field. At the same time, one should note that these values correspond to the prior belief/knowledge about the system in some sense. Table 3.1 depicts the chosen values. Fig 3.21 and Fig 3.22 are the plots corresponding to the values from Table 3.1. The title of each plot depicts the amount of error involved in measurement of the outputs. In all the cases $max(f_1)$ and $min(f_1)$ are evaluated only over the input domain which we have considered. Fig 3.21 is the comparison of dictionary sizes. Fig 3.22 is the comparison of TMSEs. In Fig 3.22, the line graph corresponding to 'max' corresponds to the case when our estimate is a zero function (which should be viewed as case when nothing has been learned from the data).

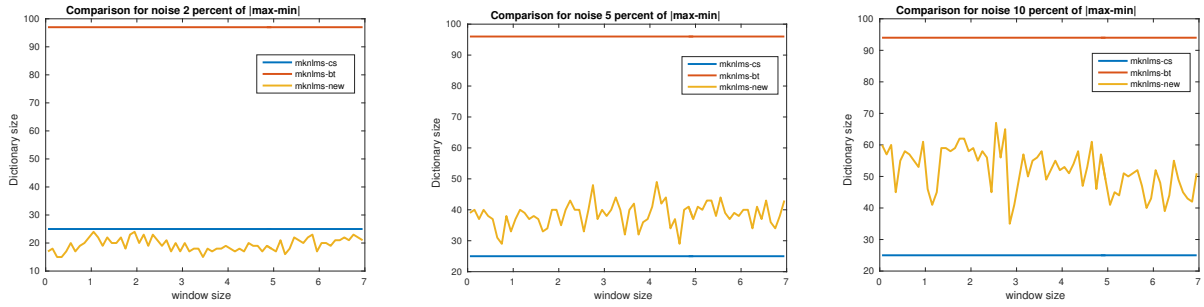


Figure 3.21: Comparisons of Dictionary Sizes with same values of parameters

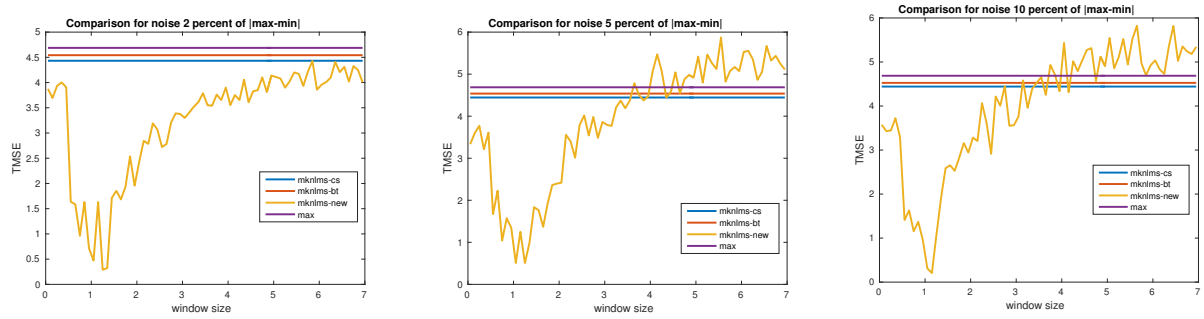


Figure 3.22: Comparisons of TMSE with same values of parameters

From the results of stage 1 (Fig 3.21 and Fig 3.22), it appears that for a few window sizes MKNLMS-new outperforms the previous approaches in terms of TMSE with nearly same dictionary sizes. In the discussion, we will present a plausible explanation for this observation. Since the step-size value plays an important role in our explanation, we wanted to see its effect on TMSE, dictionary sizes. We also wanted to see its affect on the values of window sizes which provide improved TMSE and dictionary sizes for MKNLMS-new (for example, in Fig 3.21 and Fig 3.22, this region was centred around window size of 2 units). So, we wanted to see how this outperforming region (of window sizes) changes with respect to the step-sizes.

Thus, in the stage 2 (Fig 3.23 - Fig 3.25) we have compared the three methods against different step sizes. In case of MKNLMS-CS and MKNLMS-BT, we found out the TMSE and the dictionary size for multiple step-size values. In case of MKNLMS-new, for every step-size value, we have considered multiple window sizes (from 0.01 to 7 units with gap of 0.1 units) and found out the minimum TMSE out of TMSEs of all the window sizes and the corresponding dictionary size. At the same time, we also kept track of the window size which yielded the minimum TMSE for every step-size value. Each plot in Fig 3.23- 3.25 has two axes. The axes on the right in each plot corresponds to the window size and the

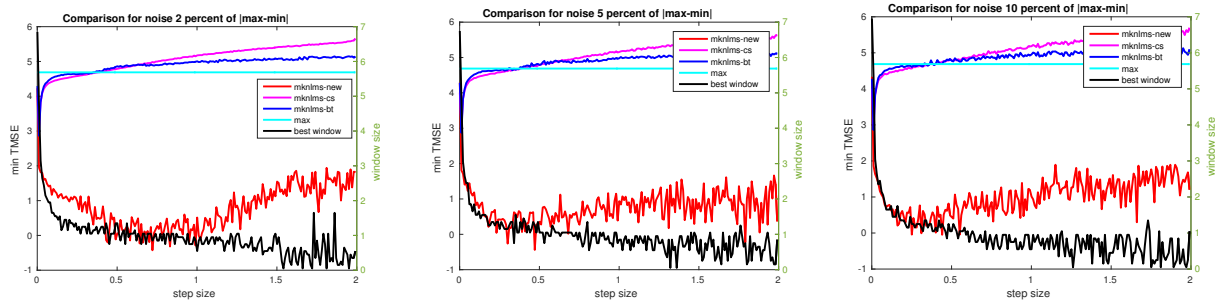


Figure 3.23: Comparisons of min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis) against step-size values (x-axis) till 2 units

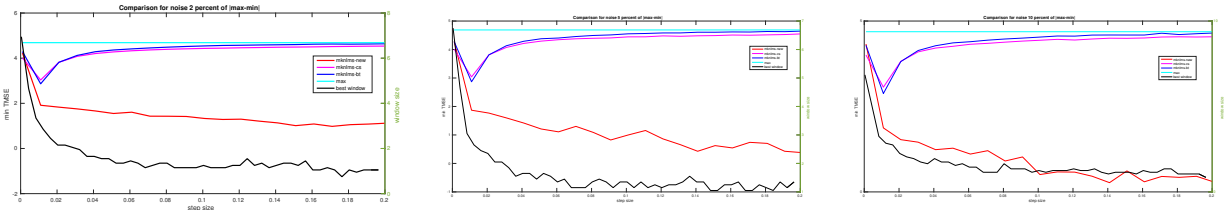


Figure 3.24: Copy of Fig 3.23 reported till 0.2 units on x-axis for better clarification

black line-graph in each plot depicts the behavior of best window size which yielded minimum TMSE for MKNLMS-new against step-size values. Fig 3.23 and Fig 3.24 are comparisons for TMSE while Fig 3.25 is comparison for dictionary size. Note that Fig 3.24 is just a zoomed in version of Fig 3.23 because we wanted to find the step-size value for which MKNLMS-CS and MKNLMS-BT yield the minimum TMSE. In Fig 3.24, one should note that MKNLMS-new outperforms the previous algorithms even at the step-size values where they yield their best result (around 0.01 units). The title of each plot depicts the error involved in observing the output values.

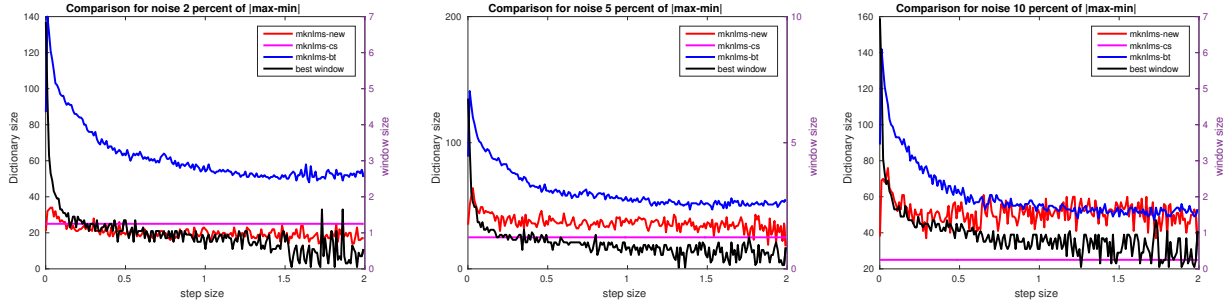


Figure 3.25: Comparisons of dictionary sizes corresponding to min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis)against step-size values (x-axis)

Algorithm	step-size (η)	regularisation parameter(λ)	coherence threshold (μ_o)	ρ
MKNLMS-CS	0.01	5×10^{-4}	0.95	10^{-5}
MKNLMS-BT	0.01	5×10^{-4}	0.95	10^{-5}
MKNLMS-new	0.5	5×10^{-4}	0.95	10^{-5}

Table 3.2: Values of the parameters for Fig 3.26 and Fig 3.27

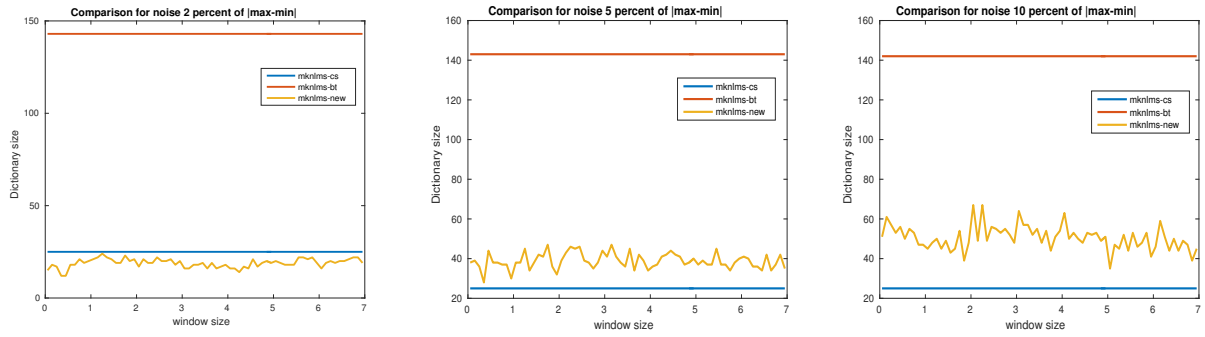


Figure 3.26: Comparisons of Dictionary Sizes with chosen values of parameters (Table 3.2)

In Fig 3.23, one should observe that the best window size (the window size which yields minimum TMSE at each step-size) tends to decrease (more or less) with increase in step-size value. From Fig 3.23 and Fig 3.24, one can deduce that MKNLMS-CS and MKNLMS-BT attain their best TMSE around step-size of 0.01 units. But even at that point, MKNLMS-new outperforms them. In case of MKNLMS-new, step-size of 0.5 units seems to be a good value for attaining small TMSE according to Fig 3.23.
In next stage, we will compare the three approaches as we did at the first stage but now the step sizes used for them will different. We will select step-sizes which yielded more or less minimum TMSE (these values have already been mentioned above). For completion purposes, Table 3.2 depicts the values of parameters used to plot Fig 3.26 and Fig 3.27. The title of each plot depicts the error involved in observing the output values.

For other functions, we will just present the results of stage 2 i.e., the effect of step-size on the minimum TMSE, dictionary size and best window size. Results at stage 1 and stage 3 for other functions are may have different values of TMSE and dictionary sizes but the behavior is same as that of f_1. So, we will omit those graphs (from stage 1 and stage 3) due to space constraints. From now on (starting from Fig 3.28), every graph will be based on Table 3.1 apart from the step-size value.

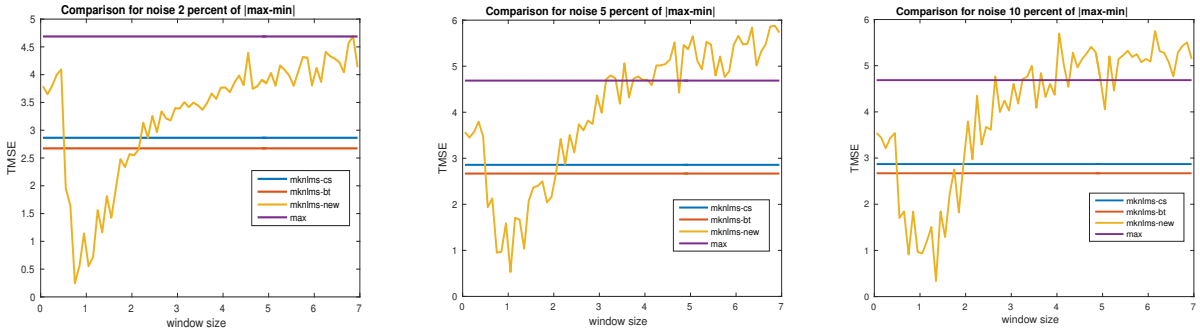


Figure 3.27: Comparisons of TMSE with chosen values of parameters (Table 3.2)

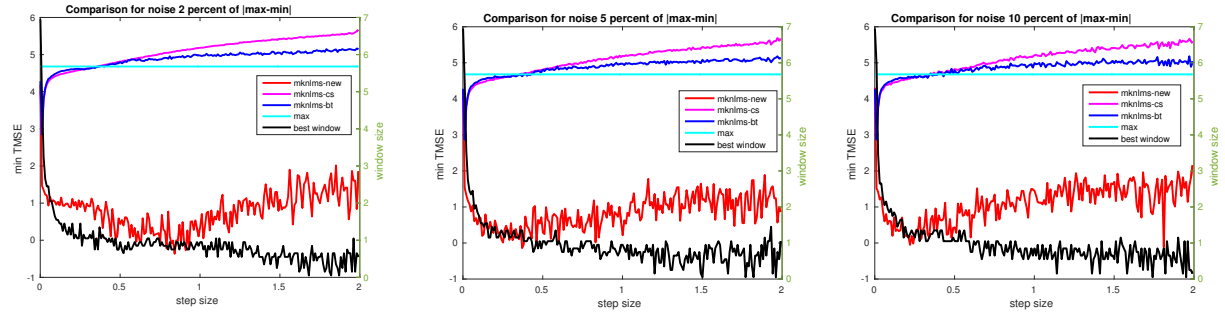


Figure 3.28: Comparisons of min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis) against step-size values (x-axis) till 2 units

Results for Function 2

Next, we changed the kernel-width of the bases from function 1 (equation 3.14). Let $\boldsymbol{x} \in [0, 10] \times [0, 10]$ be the input. Let $c_1 = [4,\ 8.5]^T$, $c_2 = [6,\ 3.5]^T$, $c_3 = [8,\ 6]^T$, $c_4 = [3.5,\ 1]^T$ be vectors which lie in our domain. The function is defined as:

$$f_2(\boldsymbol{x}) = 10 + exp\left(-\frac{\|\boldsymbol{x} - c_1\|_2^2}{2 * 3^2}\right) + 1.5exp\left(-\frac{\|\boldsymbol{x} - c_2\|_2^2}{2 * 5^2}\right) - 0.5exp\left(-\frac{\|\boldsymbol{x} - c_3\|_2^2}{2 * 2^2}\right) - 1.25exp\left(-\frac{\|\boldsymbol{x} - c_4\|_2^2}{2 * 7^2}\right). \tag{3.15}$$

Fig 3.28 - Fig 3.30 show the effect of step size on minimum TMSE, dictionary size and best window size for function 2 (f_2).

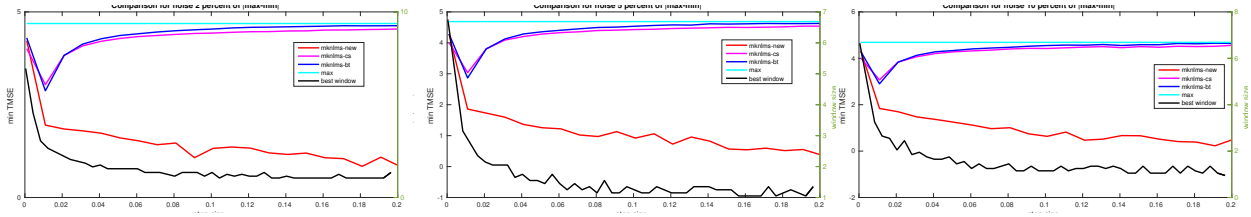


Figure 3.29: Copy of Fig 3.28 reported till 0.2 units on x-axis for better clarification

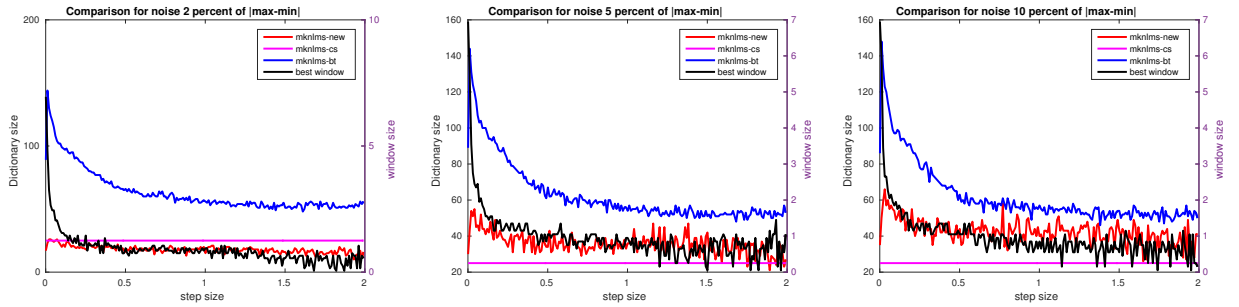


Figure 3.30: Comparisons of dictionary sizes corresponding to min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis)against step-size values (x-axis)

Results for Function 3

Next, we changed the max value attained by function 2 (equation 3.15) by removing the extra constant term (see 3.15, an extra 10 is present). Let $\boldsymbol{x} \in [0, 10] \times [0, 10]$ be the input. Let $c_1 = [4,\ 8.5]^T$, $c_2 = [6,\ 3.5]^T$, $c_3 = [8,\ 6]^T$, $c_4 = [3.5,\ 1]^T$ be vectors which lie in our domain. The function is defined as:

$$\begin{aligned} f_3(\boldsymbol{x}) =& exp\bigg(-\frac{\|\boldsymbol{x}-c_1\|_2^2}{2*3^2}\bigg) + 1.5exp\bigg(-\frac{\|\boldsymbol{x}-c_2\|_2^2}{2*5^2}\bigg) \\ & -0.5exp\bigg(-\frac{\|\boldsymbol{x}-c_3\|_2^2}{2*2^2}\bigg) - 1.25exp\bigg(-\frac{\|\boldsymbol{x}-c_4\|_2^2}{2*7^2}\bigg). \end{aligned} \tag{3.16}$$

Fig 3.31 - Fig 3.32 show the effect of step size on minimum TMSE, dictionary size and best window size for function 3 (f_3).

Results for Function 4

Next, we consider a non-Gaussian function. This is quite fluctuating function and in real life we generally don't encounter such a function but we still present the results related to

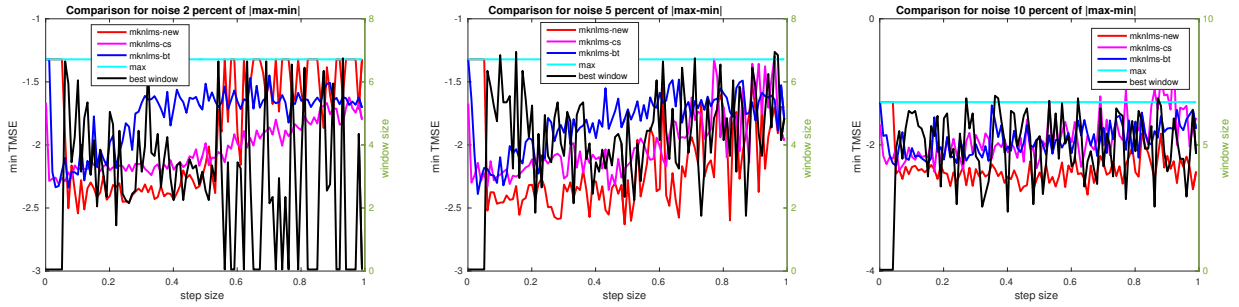


Figure 3.31: Comparisons of min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis) against step-size values (x-axis) till 2 units

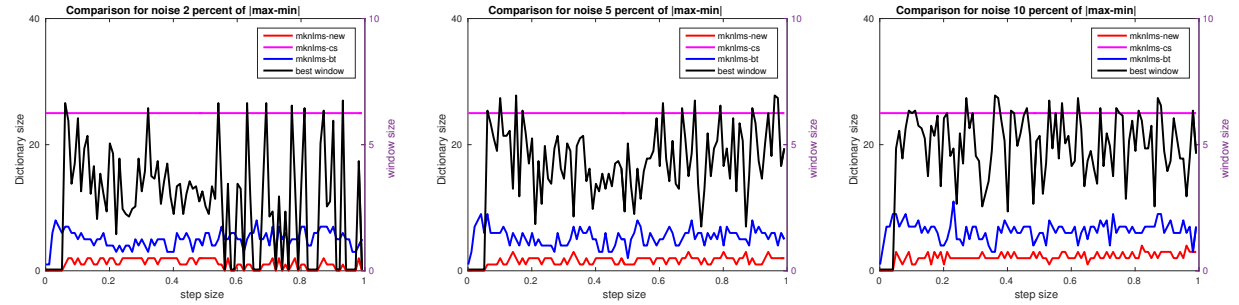


Figure 3.32: Comparisons of dictionary sizes corresponding to min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis)against step-size values (x-axis)

this function. Let $\boldsymbol{x} = [x_1, x_2]^T$ be the input vector. The function is defined as:

$$f_4(\boldsymbol{x}) = sin((x_1 + x_2)/1.5) + cos((x_1 - x_2)/2). \tag{3.17}$$

Fig 3.33 - Fig 3.34 show the effect of step size on minimum TMSE, dictionary size and best window size for function 4 (f_4).

3.3.3 Discussion

In order to explain why the MKNLMS-new is yielding better results, we need to revisit the update equation in its generic form. At each step, MKNLMS-CS and MKNLMS-BT update their coefficient matrix via (ignoring the small details of dictionary growth and pruning):

$$\boldsymbol{H_{t+1}} = \boldsymbol{H_t} + \eta \frac{y_{t+1} - \langle \boldsymbol{H_t}, \boldsymbol{K_t} \rangle}{\rho + \|\boldsymbol{K_t}\|^2} \boldsymbol{K_t}.$$

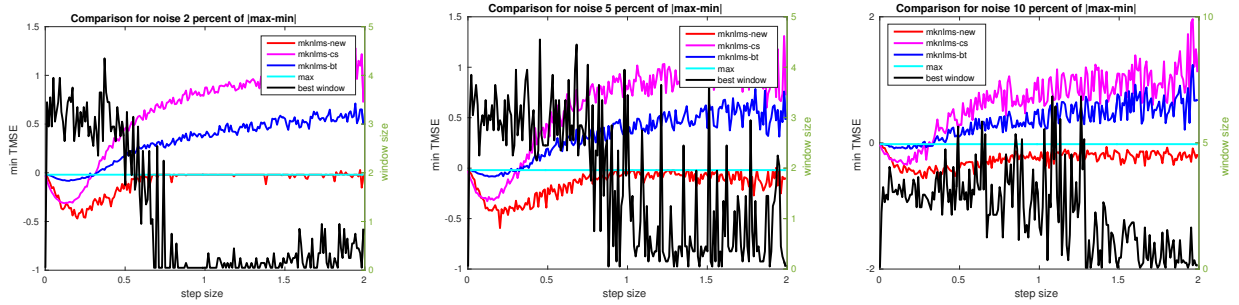


Figure 3.33: Comparisons of min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis) against step-size values (x-axis) till 2 units

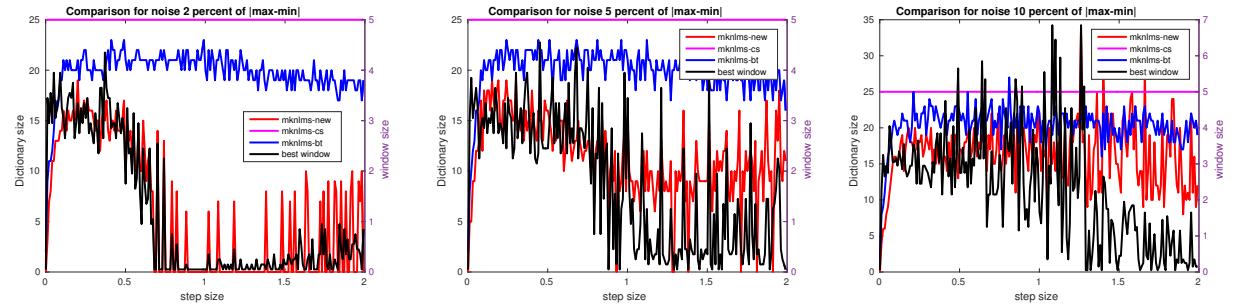


Figure 3.34: Comparisons of dictionary sizes corresponding to min TMSE attained (left y-axis) and best window sizes for MKNLMS-new (right y-axis)against step-size values (x-axis)

At each iteration, we are trying to update our coefficients in such a way that we move our estimate to its original value (notice the difference term between y_{t+1} which is original value and $\langle \boldsymbol{H_t}, \boldsymbol{K_t} \rangle$ which is its estimate). For preventing over-fitting, we don't commit fully to the original value, rather we take a fraction of step to it which is governed by the step-size parameter η. While doing so, one can note that every element of the coefficient matrix changes at each iteration. The extent of change in coefficients at other places (which correspond in some sense to the movement to their corresponding original output values) is controlled by the kernel matrix. So, larger the kernel-width, it will change more coefficients every time. This results in the movement of the estimates at those other bases away from their original output values because of the fact that at each iteration we only care about balancing out estimate at the most recent input with its original output and thus it suffers larger TMSE. By the same token, please note that in fact, we do need the new bases to push the estimates of previous bases to their corresponding previous outputs because we had only committed a fraction (η to be precise) to those original values. So, without any push from the recent bases, they will suffer large error as well. But when we have considered larger kernel-widths,

they end up over-pushing the coefficients and that leads to bad performance. By considering small values of kernel-widths we could have improved results for MKNLMS-CS and MKNLMS-BT but our dictionary size will increase (which should be clear from the definition of coherence criterion). The MKNLMS-new can accommodate larger kernel-widths, can update the previous coefficients in a proper manner and can have small dictionary size as well. But note that MKNLMS-new outperforms the previous algorithms (attaining less TMSE with similar dictionary sizes) for only few values of window sizes because of the same reason that is mentioned above. For very small windows, the other coefficients don't receive enough push to move to their original output value at each iteration (under-pushing) and on the other hand for large windows, the problem becomes similar to the case of MKNLMS-CS and MKNLMS-BT (over-pushing). In the stage 1, we compared the three methods with same values of every parameter and the explanation presented above can be accounted for better performance of MKNLMS-new for a few window sizes. It is still not clear how one can prescribe the value for window-size apriori. But some insights into this will be presented later. Following the arguments mentioned above, one can argue that if the step-size is reduced, then the problem of over-pushing can be avoided but if at the same time step-size is very small then it can lead to under-pushing. Thus, in order to understand the behavior of TMSE with changing step sizes, we plotted TMSEs of MKNLMS-CS, MKNLMS-BT and MKNLMS-new against step-size parameter. We have described earlier multiple times that in case of MKNLMS-new we consider many window sizes for each step-size value and report the minimum out of them. In other words, for every step-size we focus on the better performing window size for MKNLMS-new. In every plot of stage 2, we observe that indeed there is a smaller value of step-size (smaller than the one taken in stage 1) at which MKNLMS-CS and MKNLMS-BT achieve their best error or we should say better than the stage 1. But even at those step-sizes, MKNLMS-new outperforms them possibly because of the modified coherence criterion (which considers the fluctuations while building the dictionary).

Comment on the Relation between step-size and proper window size:

Based on the results from stage 2 (see Fig 3.23, Fig 3.28 and Fig 3.33), one can observe that the value of proper window-size more or less decreases as the step-size increases (except in case of Fig 3.31 where it more or less remains same). Again, this can be explained using the argument of under-pushing and over-pushing mentioned above. If the step-size is larger, our estimates have already moved more to the original value and so, a very little push is required which can be provided by fewer neighboring bases which leads to smaller size of proper window size. On the other hand, if the step-size is smaller, our estimates have moved

lesser to the original value and so, more little push is required which need to be provided by more neighboring bases which leads to larger size of proper window size.

Comment on the modified coherence criterion:

Since modified coherence criterion grows dictionary based on fluctuations in output as well, it gets affected by noise involved in observing output values. When there is more noise, the fluctuation is more and hence the modified coherence criterion chooses smaller kernel-width to evaluate the coherence of the new input w.r.t the dictionary which leads to more addition of elements to the dictionary. That's why with increase in noise, one can observe that the dictionary size tends to increase (see Fig 3.21, Fig 3.26, Fig 3.25, Fig 3.30, Fig 3.32, Fig 3.34).

Remark on the definition of TMSE:

We would like to clarify that the way we have defined TMSE is just for comparison purposes. It doesn't provide how good a method is while fitting. We have considered too many test points which is not generally the case in reporting the performance of learning algorithms. However, it serves well for comparing performances among different algorithms.

Remark on large fluctuation in graph of best window size

Because error is involved, the best window (where min TMSE is attained) may vary frequently. One should look at the behavior as whole. Instead of taking just the minimum TMSE, if we allow least 6 (can be any small positive integer) TMSE and take median of corresponding window sizes and plot that median, then the behavior might be captured without too much fluctuation.

Remark on the choice of path followed by the sensor

Since the modified coherence criterion takes in to account the output values as well, the path followed by the sensor becomes important. The path on which there is more noise will lead to construction of larger dictionary. How one can find an optimal path is still unclear.

3.3.4 Future Works

It is still not clear how to construct the kernel set, and how to select an optimal step-size and window-size apriori. The relation between these quantities can be exploited and some bounds can be presented based on some assumptions about the original function. In the window strategy we haven't provided any mathematical proof for its convergence or even its stability. There have been improvements regarding selection of kernels in [36] where an extra

penalty function is considered in the cost function which makes those elements of coefficient matrix go to zero which corresponding kernels are insignificant (see the reference for more details). We haven't presented a comparison between this method and our method because it involves lots of unknown parameters which make the comparison difficult.

Bibliography

[1] J. Unnikrishnan and M. Vetterli. Sampling and reconstruction of spatial fields using mobile sensors. *IEEE Transactions on Signal Processing*, 61(9):2328–2340, May 2013.

[2] Felipe Cucker and Steve Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39:1–49, 2002.

[3] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory*, COLT '01/EuroCOLT '01, pages 416–426, London, UK, UK, 2001. Springer-Verlag.

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

[5] Matthew J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

[6] Yaakov Engel, Shie Mannor, and Ron Meir. The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, 52:2275–2285, 2003.

[7] B. J. de Kruif and T. J. A. de Vries. Pruning error minimization in least squares support vector machines. *IEEE Transactions on Neural Networks*, 14(3):696–702, May 2003.

[8] S. Van Vaerenbergh, M. Lazaro-Gredilla, and I. Santamaria. Kernel recursive least-squares tracker for time-varying regression. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1313–1326, Aug 2012.

[9] Weifeng Liu, Jose C. Principe, and Simon Haykin. *Kernel Adaptive Filtering: A Comprehensive Introduction*. Wiley Publishing, 1st edition, 2010.

[10] Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Birkhäuser Basel, 2013.

[11] Massimo Fornasier and Holger Rauhut. Compressive sensing, 2010.

[12] A. Y. Yang, Z. Zhou, A. G. Balasubramanian, S. S. Sastry, and Y. Ma. Fast ℓ_1-minimization algorithms for robust face recognition. *IEEE Transactions on Image Processing*, 22(8):3234–3246, Aug 2013.

[13] Ingrid Daubechies, Ronald Devore, Massimo Fornasier, and C. Sinan Gntrk. Iteratively reweighted least squares minimization for sparse recovery. *Comm. Pure Appl. Math.*

[14] D. M. Malioutov, S. R. Sanghavi, and A. S. Willsky. Sequential compressed sensing, April 2010.

[15] Pierre J. Garrigues and Laurent El Ghaoui. Ghaoui, an homotopy algorithm for the lasso with online observations. In *in Neural Information Processing Systems (NIPS*, 2008.

[16] M. S. Asif and J. Romberg. Streaming measurements in compressive sensing: 1filtering. In *2008 42nd Asilomar Conference on Signals, Systems and Computers*, pages 1051–1058, Oct 2008.

[17] Andreas Christmann and Ingo Steinwart. *Support Vector Machines*. 01 2008.

[18] A Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Boston, Mass. : Kluwer Academic, 2004. Includes bibliographical references (p. 327-343) and index.

[19] I. Steinwart, D. Hush, and C. Scovel. An explicit description of the reproducing kernel hilbert spaces of gaussian rbf kernels. *IEEE Transactions on Information Theory*, 52(10):4635–4643, Oct 2006.

[20] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, June 1991.

[21] Simon Haykin. *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2007.

[22] Cdric Richard, Jos Carlos M. Bermudez, and Paul Honeine. Online prediction of time series data with kernels. *IEEE TRANS. SIGNAL PROCESSING*, 57(3), 2009.

[23] Jyrki Kivinen, Alex J. Smola, and Robert C. Williamson. Online learning with kernels. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 785–792, Cambridge, MA, USA, 2001. MIT Press.

[24] W. D. Parreira, J. C. M. Bermudez, C. Richard, and J. Tourneret. Stochastic behavior analysis of the gaussian kernel least-mean-square algorithm. *IEEE Transactions on Signal Processing*, 60(5):2208–2222, May 2012.

[25] B. Chen, S. Zhao, P. Zhu, and J. C. Principe. Quantized kernel least mean square algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 23(1):22–32, Jan 2012.

[26] M. Yukawa. Multikernel adaptive filtering. *IEEE Transactions on Signal Processing*, 60(9):4672–4682, Sept 2012.

[27] I. Yamada, K. Slavakis, and K. Yamada. An efficient robust adaptive filtering algorithm based on parallel subgradient projection techniques. *IEEE Transactions on Signal Processing*, 50(5):1091–1101, May 2002.

[28] P. Combettes and V. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

[29] Weifeng Liu, P.P. Pokharel, and J.C. Principe. The kernel least-mean-square algorithm. *Trans. Sig. Proc.*, 56(2):543–554, February 2008.

[30] W. Gao, J. Chen, C. Richard, J. Huang, and R. Flamary. Kernel lms algorithm with forward-backward splitting for dictionary learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5735–5739, May 2013.

[31] M. Takizawa and M. Yukawa. Adaptive nonlinear estimation based on parallel projection along affine subspaces in reproducing kernel hilbert space. *IEEE Transactions on Signal Processing*, 63(16):4257–4269, Aug 2015.

[32] M. Takizawa and M. Yukawa. Efficient dictionary-refining kernel adaptive filter with fundamental insights. *IEEE Transactions on Signal Processing*, 64(16):4337–4350, Aug 2016.

[33] M. Yukawa and K. Mller. Why does a hilbertian metric work efficiently in online learning with kernels? *IEEE Signal Processing Letters*, 23(10):1424–1428, Oct 2016.

[34] S. Theodoridis, K. Slavakis, and I. Yamada. Adaptive learning in a world of projections. *IEEE Signal Processing Magazine*, 28(1):97–123, Jan 2011.

[35] Simon Haykin. *Adaptive Filter Theory (3rd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[36] Osamu Toda and Masahiro Yukawa. Online model-selection and learning for nonlinear estimation based on multikernel adaptive filtering. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E100A(1):236–250, 1 2017.

[37] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[38] Badong Chen, Junli Liang, Nanning Zheng, and Jos C. Prncipe. Kernel least mean square with adaptive kernel size. *Neurocomput.*, 191(C):95–106, May 2016.