NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Bachelor Thesis

Random Fourier Features Theory And Applications

Spyridon C. Pougkakiotis

Supervisor : Sergios Theodoridis -Professor

ATHENS
JULY 2016

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛ/ΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Random Fourier Features
Θεωρία και εφαρμογές

Σπυρίδων Χ. Πουγκακιώτης

Επιβλεπων : Σέργιος Θεοδωρίδης - Καθηγητης

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2016

Bachelor Thesis

Random Fourier Features Theory And Applications

Spyridon C. Pougkakiotis
R.N.: 1115201200151

Supervisor: Sergios Theodoridis -Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Random Fourier Features Θεωρια και εφαρμογες

Σπυρίδων Χ. Πουγκακιώτης
Α.Μ.: 1115201200151

Επιβλέπων : **Σέργιος Θεοδωρίδης** - Καθηγητής

ABSTRACT

As discussed in the preface, the center of the thesis , is the Random Fourier Features Technique, which in Chapter 3 is explained in more depth, and implemented in various problems. Initially, the project focuses to the problem of online learning in RKHS, and the Kernel Least Mean Squares as well as the Kernel Recursive Least Squares algorithms are presented. As mentioned in the preface, in the previous framework, as the number of input variables increases, so does the complexity of the algorithm both in terms of space and time. Hence, we adopt a dictionary so as to keep the complexity under reasonable limits. Also, we mention the problem that occurs when we are working in time-varying environments, while the algorithm is not capable of discarding the obsolete elements held in the dictionary, and thus we compare algorithms with different choices of dictionary through various experiments.

On the other hand, the Random Fourier Features technique defines a known mapping which takes the data held in the input space and transfers them to a new finite dimensional Euclidean space, where the problem is linearly separable and the inner product of the mapped data approximates the kernel function. Once the "linear" task is solved, a vector of fixed size is obtained, which defines a hyperplane in the new space, separating the data. Thus, the algorithm provided is computationally more efficient and as will be shown in Chapter 3, converges at similar speeds and to similar error floors. We have applied this algorithm to address non linear adaptive filtering tasks, however it can be extended to classification tasks too, as it is an easy to understand and generalize technique. The main reason for its simplicity, is the fact that the "linear" algorithm solving the task in the new space, is almost exactly the same with the conventional linear algorithms used to solve simple linear tasks in the initial space, and the reason is that the features in the new space, are known, in contrast with the RKHS method, where the solution is obtained using the kernel trick. We will present various experiments comparing the existing adaptive filtering algorithm with the new ones, using the Random Fourier Features method. The technique's broad applications give us the capability to apply it, even in the problem of Kalman Filters, where we have to solve a non-linear regression task, given the output of a system, and find the input data, knowing that the latter have a linear recursive dependence.

SUBJECT AREA: Machine Learning
KEYWORDS: Random Fourier Feature,RKHS,KLMS,KRLS,Kalman Filter

ΠΕΡΙΛΗΨΗ

Όπως συζητήθηκε και στον πρόλογο, το επίκτεντρο αυτής της πτυχιακής είναι τα Random Fourier Features, τα οποία συζητώνται στο κεφάλαιο 3, αλλά και εφαρμόζονται σε διάφορα προβλήματα. Αρχικά, η εργασία εστιάζει στο πρόβλημα της εκμάθησης σε πραγματικό χρόνο, σε χώρους Χιλμπερτ και παρουσιάζονται οι αλγοριθμοι KRLS, KLMS. Όπως ήδη αναφέρθηκε και στον πρόλογο, στο προηγούμενο πρόβλημα, καθώς αυξάνει ο αριθμός των μεταβλητών εισόδου, ανάλογα αυξάνει και η πολυπλοκότητα του αλγορίθμου, τόσο από άποψη χώρου, όσο και χρόνου. Συνεπώς, χρειάζεται να υιοθετήσουμε ένα λεξικό, έτσι ώστε να κρατήσουμε την πολυπλοκότητα σε χαμηλά επίπεδα. Επίσης, αναφέρουμε το πρόβλημα το οποίο προκύπτει όταν δουλεύουμε σε περιβάλλοντα που εξαρτώνται από τον χρόνο, όταν ο αλγόριθμος δεν μπορεί να αφερεί αχρείαστα στοιχεία από το λεξικό και έτσι συγκρίνουμε αλγορίθμους με διαφορετικές επιλογές λεξικών μέσω διαφόρων πειραμάτων.

Από την άλλη πλευρά, η τεχνική των Random Fourier Features ορίζει μία γνωστή συνάρτηση η οποία μετατρέπει τα δεδομένα του χώρου εισόδου σε σημεία ενώς νέου Ευκλείδιου χώρου πεπερασμένων διαστάσεων, στον οποίο, το αρχικό μή γραμμικό πρόβλημα είναι γραμμικό, ενώ το εσωτερικό γινόμενο των νέων αυτών σιμείων προσεγγίζει τη συνάρτηση πυρήνα. Όταν το γραμμικό αυτό πρόβλημα λυθεί, ένα διάνυσμα σταθερού μεγέθους επιστρέφετε από τον αλγόριθμο, το οποίο αποτελεί τη λύση του προβλήματος. Επομένως, ο προκύπτων αλγόριθμος είναι υπολογιστικά πιό αποδοτικός, και όπως θα δείξουμε στο κεφάλαιο 3, συγκλίνει σε παρόμοιους χρόνους επιτυγχάνοντας παρόμοια σφάλματα με τους κλασσικούς αλγορίθμους πυρήνα. Εφαρμόσαμε την τεχνική αυτή σε διάφορα προβλήματα παλινδρόμησης όμως μπορεί εύκολα κανείς να την επεκτείνει και σε άλλα προβλήματα. Η απλότητα της τεχνικής αυτής έγκειται κυρίως στο γεγονός ότι ο αλγόριθμος επίλυσης του γραμμικού προβλήματος στον νέο χώρο, είναι σχεδόν ίδιος με τους κλασσικούς γραμμικούς αλγόριθμους που παρουσιάζονται στο κεφάλαιο 1, και αυτό συμβαίνει γιατί τα σημεία στον καινούργιο χώρο είναι γνωστά, σε αντίθεση με την τεχνική του πυρήνα, όπου η λύσει λαμβάνεται μέσω του Kernel Trick. Θα παρουσιάσουμε διάφορα πειράματα συγκρίνοντας τους υπάρχοντες αλγορίθμους επίλυσης προβλημάτων παλινδρόμησης με τους αντίστοιχους που χρησιμοποιούν τα Random Fourier Features.Τέλος, θα επεκτείνουμε το πεδίο εφαρμογής αυτής της τεχνικής, υλοποιώντας την στο πρόβλημα των Kalman φίλτρων, όπου προσπαθούμε να λύσουμε ένα μή γραμμικό πρόβλημα παλινδρόμησης, με δεδομένη την έξοδο του συστήματος, ψάχνοντας την είσοδο που το πυροδώτησε, γνωρίζοντας ότι τα δεδομένα εισόδου έχουν μία αναδρομική εξάρτηση μεταξύ τους.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ : Μηχανική Μάθηση
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ : Random Fourier Feature,RKHS,KLMS,KRLS,Kalman Filter

To my parents,
for their support

ACKNOWLEDGEMENTS

At this point, I would like to wholeheartedly thank my supervisor and teacher, Prof. Sergios Theodoridis, not only for the time he spent with me, in order to introduce me to the highly complex concepts of Machine Learning as well as the hidden theories behind the field, but also for his warm and generous treatment throughout this year. I understood and realized all these new and advanced theories, in a quite short period, mainly because of the remarkable guidance I received. He changed my way of thinking, introduced me to his clear vision and provided me with the basic and most important tools in order to be able to continue my career.

Also, I would like to thank the outstanding, extremely helpful and friendly, Dr.Panteli Boubouli, for the hours he spent with me, helping me understand various aspects of the field, answering every question I had, and introducing me to the basics of research. His contribution was highly crucial for my thesis, as well as for my studies in general.

PREFACE

In the framework of Machine Learning, various algorithms and techniques have been developed in order to construct and set the principles of a general theory, whose goal is **learning from the data**. The problems as well as the solutions vary, depending on the linearity or not of the task, the adopted approach, i.e. parametric or non-parametric modelling, etc. It can readily be seen that most of the real-world problems can be modelled accurately with non-linear schemes rather than linear ones, and as expected, the former are harder to solve than the latter.

One of the most significant tools employed in such tasks is that of Reproducing Kernel Hilbert Spaces(RKHS). More specifically, one has to implicitly map the input data into a new space of greater dimension, such that the originally non-linear task is transformed into a linear one in the RKHS. The linear task in the RKHS, can now be solved just by performing inner product operations in a very efficient way (as we are taking into advantage the structure of these spaces). As a result, the complexity is independent of the implicit dimensionality of the RKHS, which it may be infinite.

Everything seems fine, until we face the problem of dealing with online learning in RKHS. In parametric modelling in a Euclidean space, $\Re^l$, the number of unknown parameters, remains fixed as time passes. This does not apply for the case of a function lying in an RKHS, as the number of unknown parameters grows linearly with time. As a consequence, complexity will eventually become unmanageable. There are various ways to solve this problem by applying sparsification rules, with the most common being the adoption of a dictionary, where instead of using all the training points up to time n, a subset of the most crucial ones is selected. However, such ad-hoc techniques are difficult to be defined and manipulated in a scientific manner, that is, their implementation, heavily depends on the application and is almost impossible to be generalized, as well as to be theoretically analysed.

In the context of this project, an alternative solution is discussed. Instead of employing a dictionary, we approximate the kernel function using a fixed-size randomly selected subset of its Fourier features. The advantage of this rationale is that one can recast the problem as a linear one lying on $\Re^D$ where D are the number of Fourier features used for the aforementioned approximation. The resulting algorithm does not require any form of sparcification, as the solution's size remains fixed and does not increase with time. In the sequel, this technique will be implemented to solve various non-linear problems and will be tested against the ones solved in the RKHS, such as the KLMS and the KRLS algorithms, as well as the non-linear Kalman Filter.

CONTENTS

LIST OF FIGURES

NOTATION

This section provides the details of the notation that will be adopted throughout the manuscript. That is:

1. Vectors are denoted with **boldface** letters, such as $\mathbf{x}$
2. Matrices are denoted with capital letters, for example, A.
3. The identity matrix is denoted as I.
4. The trace of a matrix is denoted as trace$\{$A$\}$.
5. The vectors are assumed to be column-vectors. That is: $\mathbf{x} = \begin{pmatrix} x(1) \\ x(2) \\ . \\ . \\ . \\ x(l) \end{pmatrix}$
6. The transpose of a vector is denoted as $\mathbf{x}^T$
7. Functions are denoted with lower case letters, e.g. f, or in terms of their arguments, e.g. $f(x)$, or sometimes as $f(\cdot)$, if no specific argument is used.

1. INTRODUCTION

1.1 General theory of Reproducing Kernel Hilbert Spaces

As mentioned before, this manuscript focuses on methods for learning non-linear models. Most of the real-world problems cannot be modelled accurately with linear tasks. For example, the problem of predicting a sequence of camera movements from videos, where the orientations of the camera are continuous and non-Euclidean, and are predicted from continuous video features, is a problem that cannot be solved with linear methods [4]. In this section, solutions with the help of Reproducing Kernel Hilbert Spaces (RKHS) are going to be discussed, i.e. the (implicit) mapping of the input data to a new space, such that the originally non-linear task, is transformed into a linear one. But what exactly are these spaces? Why are the non-linear problems transformed into linear ones? How one can apply this technique? And finally, what is the structure of these spaces that makes them so popular?

Firstly, it suffices to answer the second question, of why the non-linear problems are transformed into linear, after the implicit mapping. The answer is provided by the Cover's Theorem. Consider N vectors, $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_N$ ϵ $\Re^l$. We say that these vectors are in general position, if there is no subset of l+1 of them lying on an (l-1)-dimensional hyperplane. For example, in the two-dimensional space, any three vectors are not permitted to lie on a straight line [1].
Theorem 1.1-Cover's Theorem: *The number of groupings, denoted as O(N,l), that can be formed by (l-1)-dimensional hyperplanes to separate the N points in two classes, exploiting all possible combinations, is*

$$O(N,l) = 2\sum_{i=0}^{l} \begin{pmatrix} N-1 \\ i \end{pmatrix}$$

where :

$$\begin{pmatrix} N-1 \\ i \end{pmatrix} = \frac{(N-1)!}{(N-1-i)!i!}$$

The number of all possible combinations of N points in two groups is 2^N. However, Cover's Theorem, counts only the possible linear dichotomies. For example, if we had 4 points lying in the 2-dimensional space, in a XOR format, that is $x_1 = (-1,0),\ x_2 = (1,0),\ x_3 = (0,1),\ x_4 = (0,-1)$, the Cover's theorem would count 14 groupings, without counting the following: [(x_1, x_2), (x_3, x_4)] as well as [(x_3, x_4), (x_1, x_2)], which are NOT linearly separable.

In the case where, $N \leq l+1$ then $O(N,l) = 2^N$, that is, all possible combinations in two groups are linearly separable. So, what does that theorem says? Given N points in the l-dimensional space, the theorem computes the probability of grouping these points in two linearly separable classes:

$$P_N^l = \frac{O(N,l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^{l} \begin{pmatrix} N-1 \\ i \end{pmatrix} & N>l+1 \\ 1 & N \leq l+1 \end{cases}$$

Thus, it is readily observable that for large values of l, and provided that $N < 2(l+1)$, the probability of any grouping of the data into two classes to be linearly separable tends to unity.

Now lets use this theorem in practice. Given N feature vectors $\mathbf{x}_n$ ϵ R^l, $n = 1, ..., N$,perform a mapping :

$$\phi : \boldsymbol{x_n} \ \ \epsilon \ \ R^l \mapsto \phi(\boldsymbol{x_n}) \ \ \epsilon \ \ \Re^K \ , \ K \gg l$$

Then according to the previous theorem, the higher the value of K is, the higher the probability becomes for the images of the mappings to be linearly separable in the new space. If we suppose that K is large enough, our task is transformed into a linear one in the new space, $\Re^K$. Even after the mapping, the transformed vectors are still lying in an l-dimensional surface held in the new space $\Re^K$. However, it can readily be seen that mapping the data into an arbitrarily large space increases the computational load and it may lead to overfitting solutions. Instead we have to choose a space of a specific structure, that bypasses these problems. So it is now important to answer the next questions: What are these spaces, what is their structure, and how can we construct them?

First of all, it is important to note that Hilbert spaces, are generalization of Euclidean spaces, allowing infinite dimensions. It is an abstract vector space equipped with an inner product that allows length and angle to be measured. In the following we will use the notation $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ for the inner product and the corresponding norm $\| \cdot \|_{\mathcal{H}}$ of a Hilbert space $\mathcal{H}$.

Definition 1.1 [1]: *A Hilbert space $\mathcal{H}$ is called reproducing kernel Hilbert space(RKHS), if there exists a function*

$$\kappa : \mathcal{X} x \mathcal{X} \mapsto \Re$$

with the following properties:

$$\bullet \forall\ \boldsymbol{x} \in \mathcal{X},\ \kappa(\cdot, \boldsymbol{x})\ belongs\ to\ \mathcal{H}$$
$$\bullet \kappa(\cdot,\cdot)\ has\ the\ so-called\ reproducing\ property,\ that\ is:$$
$$f(x) = \langle f\ , \kappa(\cdot, \boldsymbol{x})\ \rangle\ ,\ \forall\ f \in \mathcal{H}\ ,\ \forall\ \boldsymbol{x} \in \mathcal{X}$$

It can easily be seen that if we set $f(\cdot) = \kappa(\cdot, \boldsymbol{y})$, $\boldsymbol{y} \in \mathcal{X}$, then:

$$\langle \kappa(\cdot, \boldsymbol{y}), \kappa(\cdot, \boldsymbol{x}) \rangle = \kappa(\boldsymbol{x}, \boldsymbol{y})$$

Definition 1.2 [1]: *Let $\mathcal{H}$ be an RKHS, associated with a kernel function $\kappa(\cdot,\cdot)$, and $\mathcal{X}$ a set of elements. Then, the mapping :*

$$\boldsymbol{x} \in \mathcal{X}\ \mapsto\ \phi(\boldsymbol{x}) := \kappa(\cdot, \boldsymbol{x}) \in \mathcal{H}$$

is known as the feature map, and the space $\mathcal{H}$ as the feature space.

So, why do we choose these spaces? The vectors in the initial space e.g. **x**, are mapped in the feature space and are denoted as $\phi(\mathbf{x})$. Let **x**,**y** $\in \mathcal{X} \subseteq \Re^l$. Then, the inner product of the respective mapping images is written as:

$$\langle \phi\ (\boldsymbol{x}),\ \phi(\boldsymbol{y})\ \rangle = \kappa(\boldsymbol{x}, \boldsymbol{y}) \quad :\ Kernel\ Trick$$

This is the so-called Kernel Trick. Now, if we manage to express our evaluation function in the format of inner products, then by employing the kernel trick one does not have to know the form of the feature mapping $\phi(\cdot)$ and as a consequence the complexity becomes independent of the dimensionality of the Hilbert space, which may be infinite. This is why the mapping of the vectors is not explicit but **implicit**, because in fact, we don't even know how to perform it.

Now, lets move forward and explain the theoretical way to perform this technique[1]:

1. *Map (implicitly) the input data to an RKHS.*

$$\boldsymbol{x}_n\ \mapsto\ \phi(\boldsymbol{x}_n) \in \mathcal{H}\ ,\ n = 1, 2, ..., N$$

2. *Solve a linear estimation task in $\mathcal{H}$, using the images $\phi(\mathbf{x}_n)$, n=1,2,...,N.*

3. *Cast the algorithm, in terms of inner products (this procedure will be explained in more depth in the next subsection where the Representer's Theorem as well as the technique using Lagrange Multipliers is explained), such that the only way the maps $\phi(\cdot)$ are involved is as follows:*

$$\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{y}_j) \rangle \ , \ i, j = 1, 2, ..., N$$

4. *Replace each inner product by a kernel evaluation, that is:*

$$\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{y}_j) \rangle = \kappa(\boldsymbol{x}_i, \boldsymbol{y}_j) \quad : \ Kernel\ Trick$$

The form of $\kappa(\cdot, \cdot)$ does not concern the analysis, and our possible choices for the kernel function will be presented later. Different choices of kernel function, lead to different results. One can use the cross validation technique to converge to the right kernel decision.

In the sequel, we summarize some properties of the kernel functions[1], [5]. First of all, it is important to note that we can write the kernel in a matrix form over the data samples $\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}$, such that its entries are given by:

$$\mathcal{K}_{ij} = \langle \phi(\boldsymbol{x_i}), \phi(\boldsymbol{x_j}) \rangle = \kappa(\boldsymbol{x_i}, \boldsymbol{x_j})$$

It is known as the Gram matrix.

Lemma 1.1 [1]: *The reproducing kernel, associated with an RKHS, $\mathcal{H}$, is a positive definite kernel, i.e. :*

$$\boldsymbol{a}^T \mathcal{K} \boldsymbol{a} \ \geq \ 0 \quad \forall a \in \Re^N \ and \ x_1, x_2, \ldots, x_N \in \Re^d$$

Also, given that a Hilbert space $\mathcal{H}$, is produced by a kernel function $\kappa(\cdot, \cdot)$, then $\mathcal{H}$, can be fully generated from the knowledge of $\kappa(\cdot, \cdot)$. There are infinitely many possible selections of kernel functions, as we are also able to construct our own ones, so that they satisfy the properties mentioned before. Now we present some of the most common used kernel functions:

• The Gaussian kernel function(it is a shift-invariant kernel, used in the Random Fourier Features method):

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = exp(-\frac{\parallel \boldsymbol{x} - \boldsymbol{y} \parallel^2}{2\sigma^2})$$

with $\sigma > 0$ being a parameter. The dimension of the RKHS produced by this kernel is infinite.

• The homogeneous polynomial kernel:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y})^r$$

where r is a parameter.

• The inhomogeneous polynomial kernel:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + c)^r$$

where $c \geq 0$ and r are parameters. Polynomial kernels are associated with an RKHS of finite dimension.

• The Laplacian kernel(also a shift-invariant kernel):

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = exp(-t \parallel \boldsymbol{x} - \boldsymbol{y} \parallel)$$

where t is a parameter. The associated RKHS is of infinite dimension

Generally, there are countless kernels for every different kind of application. As mentioned before, one can also construct his/her own kernel. This is possible with the following properties:

If

$$\kappa_1(\boldsymbol{x}, \boldsymbol{y}) \ : \ \mathcal{X} x \mathcal{X} \ \mapsto \ \Re$$
$$\kappa_2(\boldsymbol{x}, \boldsymbol{y}) \ : \ \mathcal{X} x \mathcal{X} \ \mapsto \ \Re$$

are kernels, then

$$\begin{aligned}\kappa(\boldsymbol{x},\boldsymbol{y}) &= \kappa_1(\boldsymbol{x},\boldsymbol{y}) + \kappa_2(\boldsymbol{x},\boldsymbol{y}) \\ \kappa(\boldsymbol{x},\boldsymbol{y}) &= a\kappa_1(\boldsymbol{x},\boldsymbol{y}) \ , \ a > 0 \\ \kappa(\boldsymbol{x},\boldsymbol{y}) &= \kappa_1(\boldsymbol{x},\boldsymbol{y})\kappa_2(\boldsymbol{x},\boldsymbol{y})\end{aligned}$$

are also kernels.

1.2 Learning in Reproducing Kernel Hilbert Spaces

Now that we covered the basic theory that underlies kernel spaces, we may proceed to more practical issues. Remember the four steps mentioned before, concerning the procedure in order to solve a problem with the help of RKHS.

1. Map (Implicitly) the input training data to an RKHS.
2. Solve the linear estimation task in this space, involving the images
3. Cast the algorithm that solves for the unknown parameters in terms of inner product operations
4. Replace the inner products by a kernel evaluation

In order to exploit the kernel trick, we first have to find a way of writing the algorithm, so that none of the images $\phi(x)$ are involved as individuals, but only in terms of inner products. This step, is the most crucial in order for the procedure to be accomplished and there are various ways of dealing with it. We will present the Representer Theorem and the solution it provides in several problems. Another possible solution is the use of the Support Vector Machine method, which is not presented in this project.

1.2.1 Representer Theorem

Intuitively, the Representer Theorem, allows a minimizer f^* of a regularized empirical risk function defined over an RKHS to be presented as a finite linear combination of kernel products evaluated on the input data.
Theorem 1.2 [1]: *Let $\Omega[0,+\infty] \mapsto \Re$ be an arbitary strictly monotonic increasing function. Let also $\mathcal{L} : \Re^2 \mapsto \Re \cup \infty$ be an arbitary loss function. Then each minimizer, $f \ \epsilon \ \mathcal{H}$, of the regularized minimization task*

$$min_{f\epsilon\mathcal{H}} J(f) := \sum_{n=1}^{N} \mathcal{L}(y_n, f(\boldsymbol{x_n})) + \lambda\Omega(\parallel f \parallel^2)$$

admits a representation of the form:

$$f(\cdot) = \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x_n})$$

The necessity of the regularizer is obvious, since we are working in very high dimensional spaces, and without it the solution would suffer from overfitting since we are using only a finite number of training data samples. It is also common to use a bias term, which turns out to improve the performance. However, despite these obstacles, we managed to minimize the cost function with respect to f, using a finite set of parameters. In [1] a theorem that provides the representation of f while using a bias term is presented. We are now ready to give an example(a non-linear task), where the Representer theorem is applied, and the whole procedure of working in the RKHS is presented.

1.2.2 Kernel Ridge Regression

Denote as f the estimate of the unknown g($\cdot$). Furthermore, assume that f lies in a space $\mathcal{H}$ associated with the kernel:

$$\kappa \ : \ \Re^l x \Re^l \ \mapsto \ \Re$$

Also, assume that the data are generated by the following scheme:

$$y_n = g(\boldsymbol{x_n}) + \boldsymbol{v_n} \ , \ n = 1, 2, ..., N \ , \ (y_n, \boldsymbol{x_n}) \ \epsilon \ \Re x \Re^l,$$

where u_n is noise.
Using the Representer Theorem, it is easily observed that:

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \boldsymbol{\theta_n} \kappa(\boldsymbol{x}, \boldsymbol{x_n})$$

Then the coefficients that we are looking for, are optimizing the following expression:

$$\boldsymbol{\theta}^{'} = argmin_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$J(\boldsymbol{\theta}) := \sum_{n=1}^{N} \left(y_n - \sum_{m=1}^{N} \theta_m \kappa(\boldsymbol{x_n}, \boldsymbol{x_m}) \right)^2 + C\langle f, f \rangle,$$

where C is the regularizer. Now let $\mathcal{K}$ be the following matrix:

$$\mathcal{K} = \begin{pmatrix} \kappa(\boldsymbol{x_1}, \boldsymbol{x_1}) & ... & \kappa(\boldsymbol{x_1}, \boldsymbol{x_N}) \\ ... & ... & ... \\ \kappa(\boldsymbol{x_N}, \boldsymbol{x_1}) & ... & \kappa(\boldsymbol{x_N}, \boldsymbol{x_N}) \end{pmatrix}$$

Then the cost function can be rewritten as:

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta})^T (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta}) + C\boldsymbol{\theta}^T \mathcal{K}^T \boldsymbol{\theta},$$

where $\boldsymbol{y} = [y_1, ..., y_N]^T$ and $\boldsymbol{\theta} = [\theta_1, ..., \theta_N]^T$, which is completely specified by the training points and the kernel function. Hence we are ready to perform the optimization by taking the gradient of J($\boldsymbol{\theta}$) with respect to $\boldsymbol{\theta}$ and then equate it to 0, which leads to:

$$(\mathcal{K} + CI)\boldsymbol{\theta}^{'} = \boldsymbol{y}$$

and since $\mathcal{K}$ is invertible, we are able to find $\boldsymbol{\theta}$. After that, since we now know the value of $\boldsymbol{\theta}$, we can use the form of f given by the representer theorem in order to find the prediction value $\boldsymbol{y}^{'}$.

2. ADAPTIVE FILTERING ALGORITHMS

2.1 The Least Mean Squares Algorithm

In this chapter, we are going to provide various algorithms that can learn the statistics, of linear or non-linear tasks, iteratively via the training set. We are going to present some adaptive filtering algorithms, such as the Least Mean Squares Algorithm (LMS), which is capable of solving various linear regression tasks. The LMS algorithm, adjusts the filter coefficients to minimize the cost function. This gradient descent scheme, requires a step size, which may be fixed or not, and in order for the algorithm to converge, has to satisfy to following properties:

$$\sum_n \mu_n^2 < \infty, \quad \sum_n \mu_n \longrightarrow \infty \quad Convergence \;\; Conditions$$

Once the algorithm converges, it produces the solution of the problem. Unfortunately, in case where the statistics of the input data change, the algorithm is not capable of tracking these changes. This is because the error term will significantly increase and since after convergence, μ_n is very small, the increased value of the error will not lead to changes of the estimate at time instant n. However, if one chooses a fixed step size, μ, the latter is not a problem anymore, and the algorithm will be able to track changes in the underlying statistics, and hence to change the estimate in which it has converged. Since this holds, one has to study its performance both in stationary and non-stationary environments.

Now, lets provide the LMS algorithm, which is a simple, online, gradient descent method.
Algorithm 2.1-The LMS algorithm [1]:

- Initialize

  $\boldsymbol{\theta_{-1}} = \mathbf{0} \;\epsilon\; \Re^l$

  Select a value for the step size μ

- For n=0,1,..., Do

  $e_n = y_n - \boldsymbol{\theta_{n-1}^T x_n}$

  $\boldsymbol{\theta_n} = \boldsymbol{\theta_{n-1}} + \mu e_n \boldsymbol{x_n}$

- End For

2.2 The Quantized Kernel Least Mean Squares Algorithm

In this section, we are going to present the elegant KLMS algorithm, used for online learning in an RKHS. It is an extension of the previous algorithm, provided in section 2.1, capable of solving non-linear regression tasks. Despite the ease and the power of this algorithm, the number of unknown parameters, grows linearly with time, and hence, the complexity increases proportionally with time iterations until it becomes unmanageable both in time and memory. This problem has been discussed multiple times in this manuscript, and now, we will present some methods to overtake it, methods we called before as ad-hoc techniques.

As stated in Chapter 1, when we are working in an RKHS, we consider $\boldsymbol{x} \;\epsilon\; \Re^l$ and an implicit mapping:

$$\boldsymbol{x} \;\mapsto\; \phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x}) \;\epsilon\; \mathcal{H}$$

The task is to estimate a function $f \;\epsilon\; \mathcal{H}$, such that the expected risk is minimized, that is [1]:

$$J(f) = \frac{1}{2} E[|y - \langle f, \phi(\boldsymbol{x})\rangle|^2] \qquad (3)$$

Since our variable here is a function, while the space $\mathcal{H}$ is a space of functions, in order to optimize equation (3), we have to differentiate with respect to f. The latter is possible, if we adopt the Frechet derivatives, where: $\nabla_f \langle f, g\rangle = g$, for $f, g \;\epsilon\; \mathcal{H}$. After differentiating, it turns out that:

$$\nabla_f J(f) = -E[\phi(\boldsymbol{x})(y - \langle f, \phi(\boldsymbol{x})\rangle)]$$

However, in online learning, we can't work with random variables, and hence we have to replace them with our given observations, which increase in every time step. As a consequence, we have to adopt an error that is updated in every time instant, as well as an estimate, that is:

$$e_n = y_n - \langle f_{n-1}, \phi(\boldsymbol{x_n}) \rangle,$$

so that a stochastic gradient is adopted:

$$f_n = f_{n-1} + \mu_n e_n \phi(\boldsymbol{x_n}) = f_{n-1} + \mu_n e_n \kappa(\cdot, \boldsymbol{x_n}),$$

where $(y_n, \boldsymbol{x_n})$, n=1,2,... are the received observations, and μ being a step size. If we set $f_0 = 0$ as our initial prediction for the value of f, and use a fixed step size, $\mu_n = \mu$, then it is readily observed that :

$$f_n = \mu \sum_{i=1}^{n} e_i \kappa(\cdot, \boldsymbol{x_i}),$$

and the prediction takes place with the help of the kernel trick, such that:

$$y'_n = \langle f_{n-1}, \kappa(\cdot, \boldsymbol{x_n}) \rangle = \mu \sum_{i=1}^{n-1} e_i \kappa(\boldsymbol{x_n}, \boldsymbol{x_i}).$$

As stated before, the memory grows unbounded, since in every time instant, in order to predict y' as well as evaluate f, we are using all the given training samples. So, a sparsification rule has to be adopted. In [1], various solutions are proposed, that employ either regularization or a growing dictionary. In the latter, instead of using all the training samples up to time n, one can select a sufficient subset of them. This subset as well as the properties which a point has to satisfy in order to be included in it, forms a dictionary. There are various strategies used to decide whether a point has to be included in the dictionary or not. We will present only two at this point, called the *Novelty Criterion* and *The Coherence Criterion* respectively.

- *Novelty Criterion*
  Denote $\mathcal{D}_{n-1}$ as the current dictionary and M_{n-1} as its cardinality, where $\boldsymbol{u_k}$, k=1,2,...,M_{n-1}, are its elements. Once a new observation arrives, we have to decide whether it is informative or not, so we take its distance from all the point held in the current dictionary, that is:

  $$d(\boldsymbol{x_n}, \mathcal{D}_{n-1}) = min_{\boldsymbol{u_k} \epsilon \mathcal{D}_{n-1}} \|\boldsymbol{x_n} - \boldsymbol{u_k}\|$$

  If this distance is smaller than a threshold δ_1 then the point is ignored and the dictionary remains the same. If not, an error is computed, such that:

  $$e_n = y_n - y'_n = y_n - \mu \sum_{i=1}^{M_{n-1}} e_i \kappa(\boldsymbol{x_n}, \boldsymbol{u_i})$$

  Now, if $|e_n| < \delta_s$, where δ_s is a threshold, then again the point is ignored. If not the point is inserted in the dictionary.

- *The Coherence Criterion*
  In this case, a new input point $\boldsymbol{x_n}$ is added to the dictionary if its coherence is above a given threshold ϵ_0, that is :

  $$max_{\boldsymbol{u_k} \epsilon \mathcal{D}_{n-1}} \{|\kappa(\boldsymbol{x_n}, \boldsymbol{u_k})|\} > \epsilon_0$$

  It can be proven that, by following this rule, the cardinality of the dictionary, $\mathcal{D}_n$ remains fixed as $n \longrightarrow \infty$.

Both these techniques have a common drawback. Once a data point is inserted into the dictionary, it remains there forever, something that affects the performance of the algorithm in the case of non- stationary environments. The following algorithm, uses a different technique, which changes the respective weights of the points in the dictionary, constructing the quantized KLMS algorithm. Now, lets present the algorithm, where the quantization technique is used :

Algorithm 2.2-The quantized kernel LMS [1]:

- Initialization:
  $\mathcal{D} = \emptyset$,M=0
  choose μ and the quantization level δ.
  d(**x**,$\emptyset$) $:= \Delta > \delta$,$\forall$ **x**

- For n=1,2,..

```
    if n=1 then
        $y'_n = 0$
    else
        $y'_n = \mu \sum_{i=1}^{M} e_i \kappa(\boldsymbol{x_n}, \boldsymbol{u_i})$
    End if
    $e_n = y_n - y'_n$
    $d(\boldsymbol{x_n}, \mathcal{D}) = min_{\boldsymbol{u_k} \epsilon \mathcal{D}} \|\boldsymbol{x_n} - \boldsymbol{u_k}\|$
    if $d(\boldsymbol{x_n}, \mathcal{D}) > \delta$ then
        $\boldsymbol{\theta} = \mu e_n$
        $M = M + 1$
        $\boldsymbol{u_M} = \boldsymbol{x_n}$
        $D = D \cup \{\boldsymbol{x_M}\}$
        $\boldsymbol{\theta} = [\boldsymbol{\theta}, \theta_n]$, the dimension of $\boldsymbol{\theta}$ is increased
    else
        $\theta_{l_0} = \theta_{l_0} + \mu e_n$, Update the weight of the nearest point
    End if
```

- End for

The algorithm returns the vector $\boldsymbol{\theta}$ as well as the dictionary.

2.3 The FOBOS Kernel Least Mean Squares Algorithm

In the previous section, we presented the Quantized KLMS algorithm, that employs a dictionary whose elements are inserted by checking the Novelty criterion and their weights are changed, depending on the input data. However, in many situations, the statistics of the input data vary over time. Hence, it seems a good idea to update the dictionary in an online way, by discarding the obsolete elements and adding appropriate ones. Such a solution, is presented and analysed in [6], where the FOBOS-KLMS algorithm is presented and tested against other versions of the KLMS algorithm. Generally, most adaptive filtering techniques with kernels, comprise two steps at each iteration: the update of the dictionary and a parameter update step. Most of the existing strategies are only able to insert new elements to the dictionary and some of them to forget the old ones, using a forgetting factor. Hence, they cannot discard obsolete kernel functions, which may be a serious drawback in the case of time-varying environments. In [6], an analytical study of the convergence behavior of the Gaussian least-mean square algorithm in the case where the statistics of the dictionary elements only partially match the statistics of the input data is presented. The latter property is quite useful in the case of non-stationary environments, and that is confirmed by various experiments held in paper [6].

Firstly, lets provide some general knowledge about forward backward splitting method[6] (the technique which the FOBOS KLMS algorithm is using). Consider the following optimization problem:

$$\boldsymbol{a}* = argmin_{\boldsymbol{a}\epsilon\Re^N}\{Q(\boldsymbol{a}) = J(\boldsymbol{a}) + \lambda\Omega(\boldsymbol{a})\}, \qquad (4)$$

where $J(\cdot)$ is a convex empirical loss function with Lipschitz continuous gradient and Lipschitz constant $\frac{1}{\eta_0}$. $\Omega(\cdot)$ is a convex regularizer and λ is the regularization constant. Our goal is to minimize the quadratic approximation $Q(\boldsymbol{a})$ at a given point, iteratively, which since $Q(\boldsymbol{a}) \leq Q_\eta(\boldsymbol{a}, \boldsymbol{a_n})$, is:

$$Q_\eta(\boldsymbol{a}, \boldsymbol{a_n}) = J(\boldsymbol{a_n}) + \nabla J(\boldsymbol{a_n})^T(\boldsymbol{a} - \boldsymbol{a_n}) + \frac{1}{2\eta}\|\boldsymbol{a} - \boldsymbol{a_n}\|_2^2 + \lambda\Omega(\boldsymbol{a}) \qquad (5)$$

As stated in [6], (5) admits a unique minimizer $\boldsymbol{a_{n+1}}$, such that:

$$\boldsymbol{a_{n+1}} = argmin_{\boldsymbol{a}\epsilon\Re^N}\left\{\lambda\Omega(\boldsymbol{a}) + \frac{1}{2\eta}\|\boldsymbol{a} - \boldsymbol{a'_n}\|_2^2\right\} \qquad (6)$$

where $\boldsymbol{a'_n} = \boldsymbol{a_n} - \eta\nabla J(\boldsymbol{a_n})$. Now, (6), is called the proximity operator for the regularizer $\Omega(\cdot)$, and is denoted as $Prox_{\lambda\eta\Omega(\cdot)}(\cdot)$. This scheme, inserts exact sparsity at every time instant. We obtain a global minimum over the last equation, if $\frac{1}{\eta}$ is a Lipschitz constant of the gradient $\nabla J(\boldsymbol{a})$[6].

Forward-Backward splitting is an efficient method for optimization over convex cost functions, using a sparse regularization. In the sequel, we will present FOBOS-KLMS, an algorithm using the method presented before, for stochastic optimization. Lets transform the problem in order to apply this method for the KLMS algorithm. Consider the following problem:

$$\boldsymbol{a}* = argmin_{\boldsymbol{a}\epsilon\Re^N}\{Q(\boldsymbol{a}) = \|\boldsymbol{d} - \boldsymbol{K}\boldsymbol{a}\|_2^2 + \lambda\Omega(\boldsymbol{a})\}, \qquad (7)$$

where $\boldsymbol{K}$ is the gram matrix. Problem (7), can be solved using the forward-backward splitting technique. In [6], two regularization techniques are presented but it suffices to adopt only one here, that is, the l_1- norm function, $\Omega(\boldsymbol{a}) = \sum_m |a(m)|$. Its proximity operator, for each of its entries, can be expressed as:

$$(Prox_{\lambda\eta\|\cdot\|_1}(\boldsymbol{a}))(m) = sign\{a(m)\}max\{|a(m)| - \lambda\eta, 0\} \qquad (8)$$

The latter regularization is known as soft thresholding operator. Also, recall the Coherence Criterion mentioned in section 2.2, concerning the entrance of an element into a dictionary. In the following, the FOBOS KLMS algorithm is given in details. The algorithm employs sparsity promoting regularization and in addition, it has the ability to discard obsolete elements from the dictionary, while using the Coherence Criterion in order to insert useful ones.

Algorithm 2.3-FOBOS-KLMS [6]:

```
• Initialization:
  Select the step size $\eta$
  Select the parameters of the kernel(e.g. Gaussian : select $\sigma$)
  Insert $\kappa(\cdot, \boldsymbol{u_1})$ ,into the dictionary
  $\boldsymbol{a_1} = 0$
• for $n = 1, 2, ...,$ do
      if $(max_{m=1,...,M}|\kappa(\boldsymbol{u_n}, \boldsymbol{u_{\omega_m}})| > \mu_0)$ - Coherence Criterion
          Compute : $\boldsymbol{\kappa}_{\omega,n} = [\kappa(\boldsymbol{u_n}, \boldsymbol{u_{\omega_1}}), ..., \ \kappa(\boldsymbol{u_n}, \boldsymbol{u_{\omega_M}})]^T$
          $\boldsymbol{a'_{n+1}} = \boldsymbol{a'_n} + \eta e_n \boldsymbol{\kappa}_{\omega,n}$
      else if $(max_{m=1,...,M}|\kappa(\boldsymbol{u_n}, \boldsymbol{u_{\omega_m}})| \leq \mu_0)$
          M = M + 1;
          Insert $\kappa(\cdot, \boldsymbol{u_n})$ into the dictionary
          Compute : $\boldsymbol{\kappa}_{\omega,n} = [\kappa(\boldsymbol{u_n}, \boldsymbol{u_{\omega_1}}), ..., \ \kappa(\boldsymbol{u_n}, \boldsymbol{u_{\omega_M}})]^T$
          $\boldsymbol{a'_{n+1}} = \begin{pmatrix} \boldsymbol{a'_n} \\ 0 \end{pmatrix} + \eta e_n \boldsymbol{\kappa}_{\omega,n}$
      end if
      for $i = 1, ..., M$
          $Prox_{\lambda\eta\|\cdot\|_1}(\boldsymbol{a})(m) = sign\{a(m)\}max\{|a(m)| - \lambda\eta, 0\}$
          if $(a_n(m) = 0$ )
              Remove $\kappa(\cdot, \boldsymbol{u_{\omega_m}})$ from the dictionary (M = M - 1)
          end if
      end for
      The solution is given as:
          $$\psi(\boldsymbol{u_n}) = \sum_{m=1}^{M} a_m \kappa(\boldsymbol{u_n}, \boldsymbol{u_{\omega_m}})$$
• end for
```

In the sequel, we are going to provide an example of chaotic series, where the variance of the input data is changed throughout the learning phase! Then, we are going to compare the FOBOS-KLMS, and the QKLMS algorithm, over the mentioned series. It is expected that both algorithms achieve similar error floors, while the former maintains a smaller dictionary, without being affected by the change in the underlying statistics in the input data. The following example was taken from [6], as well as the parameters of the FOBOS-KLMS algorithm. The value of the variance of the Gaussian Kernel, is :$\sigma = 0.02$. Consider the following problem:

• **Example 2.1:**

$$y(n) = \frac{y(n-1)}{1 + y^2(n-1)} + u^3(n-1),$$

while,

$$d(n) = y(n) + z(n),$$

where the output signal $y(n)$ is corrupted by a zero-mean i.i.d. Gaussian noise $z(n)$, with variance $\sigma_z^2 = 10^{-4}$. The input sequence $u(n)$ was randomly generated from a zero-mean Gaussian distribution with a standard deviation equal to : $\sigma_u = 0.15$ for the first half samples, while $\sigma_u = 0.35$ for the rest (this is the change in the underlying statistics of the input data). The length of the sequence was set to: $N = 4000$. The Gaussian kernel's (which was used in both algorithms) bandwidth, σ, was set to 0.02 for the FOBOS while 2 for the QKLMS. The parameters of the FOBOS-KLMS were: step size $\eta = 0.01$, coherence criterion threshold $\mu_0 = 0.01$, the regularization constant $\lambda = 0.000001$. The parameters of the QKLMS were: step size $\mu = 1$, quantization size $q = 1$. One can observe in figure 2.1 that the obtained MSE of the QKLMS, is quite better than the one achieved by the Quantized-KLMS

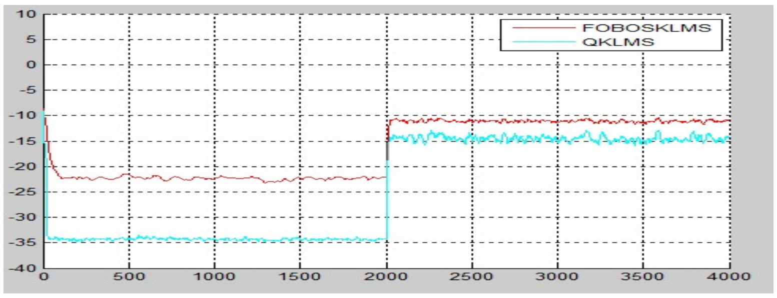

Figure 2.1: Simulations of the FOBOS-KLMS versus the Quantized-KLMS algorithm. The input data follow the Example 2.1. The results are averaged over 200 runs.The horizontal line in the figure, represents the approximation of the steady-state MSE.

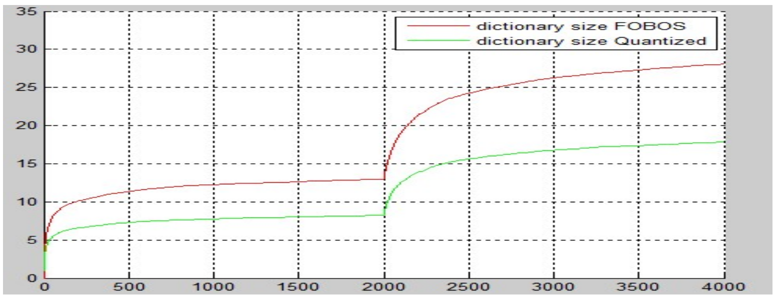

Figure 2.2: Simulations of the FOBOS-KLMS versus the Quantized-KLMS algorithm. The input data follow the Example 2.1. The results are averaged over 200 runs. The horizontal line in the figure, represents the mean number of elements held in the dictionary at the respective time instant.

algorithm. Concerning the convergence properties, both algorithms achieve similar scores. However, the most interesting fact can be seen in figure 2.2, where the size of the dictionary of the respective algorithms is plotted, and it is averaged over 200 runs. One can observe, that the FOBOS-KLMS has more elements in the dictionary compared to the QKLMS, despite the fact that it discards the obsolete elements. Hence, it is readily observed that the regularization taking place in the FOBOS is not that important.

2.4 The standard RLS algorithm

Since our discussion in this Chapter concerns the adaptive filtering algorithms, we will include one of the most popular linear algorithms of this type, known as Recursive Least Squares Algorithm, which will be revisited in Chapter 3, in order to transform it into a non-linear one, using the Random Fourier Features technique. This algorithm minimizes a linear least squares cost function related to the input signal, and does not aim in reducing the mean square error, such as in the LMS algorithm. It is worth mentioning that RLS exhibits extremely fast convergence, however with high computational complexity.

• The Recursive Least-Squares Algorithm

The RLS is a recursive online algorithm. Its main goal is solving the LS problem in a more efficient way than other general purpose solvers. This is possible, since the algorithm takes into advantage the special structure of its covariance matrix. The algorithm, is also able to deal with time varying environments, something that will also be presented here. We will assume that time starts at instant n=0, and the received observations are $(y_n, \boldsymbol{x_n}), n = 0, 1, 2, ...$. So the input data can also be presented through an input matrix, that is: $\boldsymbol{X}^T = [\boldsymbol{x_0}, \boldsymbol{x_1}, ..., \boldsymbol{x_n}]$. In order for the algorithm to be able to deal with time-varying environments, we have to include a forgetting factor, $0 < \beta \leq 1$. The latter, makes it possible for the algorithm to slowly forget past data samples, by reducing their weight, so that the algorithm is able to track changes in the underlying statistics. Initially, where $n < l - 1$, the input matrix is not invertible, since its rank is less that l. If $n > l - 1$, it can become of full rank. All the arguments stated here, lead to the following least-squares scheme:

$$\boldsymbol{\theta_n} = arg\ min_\theta \Bigg(\sum_{i=0}^{N} \beta^{n-1} (y_i - \boldsymbol{\theta}^T \boldsymbol{x_i})^2 + \lambda \beta^{n+1} \|\boldsymbol{\theta}\|^2 \Bigg), \qquad (9)$$

where β is a user defined parameter. The regularization parameter is also time-varying, since for large values of n, the ipnut matrix is invertible and hence no regularization is required.

By minimizing (9), with respect to $\boldsymbol{\theta}$ we end up in [1]:

$$\Phi_n \boldsymbol{\theta_n} = \boldsymbol{p_n},$$

where

$$\Phi_n = \sum_{i=0}^{n} \beta^{n-i} \boldsymbol{x_i} \boldsymbol{x_i}^T + \lambda \beta^{n+1} I$$

and

$$\boldsymbol{p_n} = \sum_{i=0}^{n} \beta^{n-i} \boldsymbol{x_i} y_i$$

After some algebra, one reaches the following relations:

$$\Phi_n^{-1} = \beta^{-1} \Phi_{n-1}^{-1} - \beta^{-1} K_n \boldsymbol{x_n}^T \Phi_{n-1}^{-1}$$

$$K_n = \frac{\beta^{-1} \Phi_{n-1}^{-1} \boldsymbol{x_n}}{1 + \beta^{-1} \boldsymbol{x_n}^T \Phi_{n-1}^{-1} \boldsymbol{x_n}},$$

where K_n is known as the Kalman Gain. Let $P_n = \Phi_n^{-1}$, then after some operations provided in [1]:

$$K_n = P_n \boldsymbol{x_n}$$

Now the time updating of $\boldsymbol{\theta_n}$ turns out to be:

$$\boldsymbol{\theta_n} = \boldsymbol{\theta_{n-1}} + K_n e_n,$$

where

$$e_n := y_n - \boldsymbol{\theta_{n-1}} \boldsymbol{x_n}$$

We are now ready to present the algorithm:
Algorithm 2.4-The Standard RLS [1]:

```
• Initialization
  $\boldsymbol{\theta_{-1}} = \mathbf{0}$
  $P_{-1} = \lambda^{-1} I$, where $\lambda > 0$ is a user -defined parameter
  Choose $\beta$

• For $n = 0, 1, 2 ... Do$
      $e_n = y_n - \boldsymbol{\theta_{n-1}^T} \boldsymbol{x_n}$
      $z_n = P_{n-1} \boldsymbol{x_n}$
      $K_n = \frac{z_n}{\beta + \boldsymbol{x_n}^T z_n}$
      $\boldsymbol{\theta_n} = \boldsymbol{\theta_{n-1}} + K_n e_n$
      $P_n = \beta^{-1} P_{n-1} - \beta^{-1} K_n z_n^T$

• End For
```

(A convergence analysis concerning the standard RLS can be found in [8])

2.5 The Kernel RLS algorithm

Now lets perform the analysis held in the previous section, in the framework of the Reproducing Kernel Hilbert Spaces, presented in Chapter 1. The standard RLS algorithm is used to train a linear regression model. Now in the case of a non-linear one, we have to minimize the following cost function:

$$\mathcal{L}(\boldsymbol{w}) = \sum_{i=1}^{n} \left(f'(\boldsymbol{x_i}) - y_i \right)^2 = \| \boldsymbol{\Phi_n^T w} - \boldsymbol{y_n} \|^2, \qquad (10)$$

where $\boldsymbol{y_n} = (y_1, ..., y_n)^T$ and $\boldsymbol{\Phi_n} = (\phi(\boldsymbol{x_1}), ..., \phi(\boldsymbol{x_n}))^T$, are the feature maps (in the RKHS) of the input data. The first thought is to minimize (10) with respect to $\boldsymbol{w}$, as we did in the case of the standard RLS, using the pseudo-inverse of $\boldsymbol{\Phi_n}$, however the feature space may be of infinite dimension, making the matrix manipulation prohibitive. Despite this difficulty, we are able to express $\boldsymbol{w}$ as a linear combination of some centres in the RKHS, with the feature maps obtained from the training samples, that is:

$$\boldsymbol{w_n} = \sum_{i=1}^{n} a_i \phi(\boldsymbol{x_i}) = \boldsymbol{\Phi_n a}, \qquad (11)$$

where $\boldsymbol{a} = (a_1, ..., a_n)^T$. By substituting (11) in (10), one ends up in:

$$\mathcal{L}(\boldsymbol{a}) = \|\boldsymbol{K_n a} - \boldsymbol{y_n}\|^2$$

One could say that the latter can easily produce the minimizer, $\boldsymbol{a_n} = \boldsymbol{K_n^{-1} y_n}$, but in fact this is extremely difficult for large datasets, due to the enormous time and space complexity of storing $\boldsymbol{K}$, estimating $\boldsymbol{a}$ and evaluating the obtained points. Also, overfitting is a possible problem, as well as $\boldsymbol{K}$ is numerically unstable [8]. Hence, as we did in the case of the KLMS, a sparcification technique has to be adopted, in order for the algorithm to work properly. The idea is the same as in the KLMS case, since we use a dictionary holding information from certain training points and not all of them. Technically, we use the following property provided in [7]:

$$\boldsymbol{w_n} = \boldsymbol{\Phi_n a_n} \approx \boldsymbol{\Phi'_n A_n^T a_n} = \boldsymbol{\Phi'_n a'_n},$$

where $\boldsymbol{a'_n} = \boldsymbol{A_n^T a_n}$ is a vector of m "reduced" coefficients. Then the loss function becomes:

$$\mathcal{L}(\boldsymbol{a'}) = \|\boldsymbol{\Phi_n^T \Phi'_n a'} - \boldsymbol{y_n}\|^2 = \|\boldsymbol{A_n K'_n a'} - \boldsymbol{y_n}\|^2$$

Now lets describe the use of the dictionary. First we have to use the user-defined parameter v. Then:

- If $\delta_n \leq v$ then $\mathcal{D}_n = \mathcal{D}_{n-1}$, $m_n = m_{n-1}$(the cardinality of the dictionary) and $\boldsymbol{K'_n} = \boldsymbol{K'_{n-1}}$.
  Update Equations In this case :Only the matrix $\boldsymbol{A}$ changes: $\boldsymbol{A_n^T} = [\boldsymbol{A_{n-1}}, \boldsymbol{a_n}]^T$. Defining $P_n = (\boldsymbol{A_n^T A_n})^{-1}$ we end up with[7]:
  $$P_n = P_{n-1} - \frac{P_n \boldsymbol{a_n a_n^T} P_{n-1}}{1 + \boldsymbol{a_n^T} P_{n-1} a_n}$$
  and the recursive update for $\boldsymbol{a'}$ becomes:
  $$\boldsymbol{a'_n} = \boldsymbol{a'_{n-1}} + (K'_n)^{-1} q_n \left(y_n - \boldsymbol{k'_{n-1}}(\boldsymbol{x_n})^T \boldsymbol{a'_{n-1}} \right),$$
  where $q_n = \frac{P_{n-1}\boldsymbol{a_n}}{1+\boldsymbol{a_n^T} P_{n-1} \boldsymbol{a_n}}$.
- If $\delta_n > v$, $\boldsymbol{x_n}$ is added to the dictionary ($\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{\boldsymbol{x_n}\}$).
  Update Equations In this case:
  $$\boldsymbol{K'_n} = \begin{pmatrix} \boldsymbol{K'_{n-1}} & \boldsymbol{k'_{n-1}}(\boldsymbol{x_n}) \\ \boldsymbol{k'_{n-1}}(\boldsymbol{x_n})^T & k_{nn} \end{pmatrix}$$
  $$\Rightarrow$$
  $$\boldsymbol{K'^{-1}_n} = \frac{1}{\delta_n} \begin{pmatrix} \delta_n \boldsymbol{K'^{-1}_{n-1}} + \boldsymbol{a'_n a'^T_n} & -\boldsymbol{a'_n} \\ -\boldsymbol{a'^T_n} & 1 \end{pmatrix}$$
  Also:
  $$\boldsymbol{A_n} = \begin{pmatrix} \boldsymbol{A_{n-1}} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{pmatrix}$$

and

$$\boldsymbol{P_n} = \begin{pmatrix} \boldsymbol{P_{n-1}} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

Finally, the update rule for $\boldsymbol{a'}$ will be:

$$\boldsymbol{a'_n} = \begin{pmatrix} \boldsymbol{a'_{n-1}} - \frac{a'_n}{\delta_n}(y_n - \boldsymbol{k'_{n-1}}(\boldsymbol{x_n})^T \boldsymbol{a'_{n-1}}) \\ \frac{1}{\delta_n}(y_n - \boldsymbol{k_{n-1}}(\boldsymbol{x_n})^T \boldsymbol{a'_{n-1}}) \end{pmatrix},$$

where $\boldsymbol{a'^T_n} \boldsymbol{K'_{n-1}} = \boldsymbol{k'_{n-1}}(\boldsymbol{x_n})^T$. We are now ready to present the KRLS algorithm. (For a more detailed view in the latter, one can look at [7], where the analysis was taken from)

Algorithm 2.5-The Kernel RLS [7]:

```
• Initialization
  $\boldsymbol{K'_1} = k_{11}, \boldsymbol{K'^{-1}_1} = [\frac{1}{k_{11}}]$
  $\boldsymbol{a_1} = \frac{y_1}{k_{11}}, \boldsymbol{P_1} = [1], m = 1$
  Choose parameter $v$
• For $n = 2, 3, ...$ Do
      Compute $\boldsymbol{k'_{n-1}}(\boldsymbol{x_n})$
      $\boldsymbol{a_n} = \boldsymbol{K'^{-1}_{n-1}} \boldsymbol{k'_{n-1}}(\boldsymbol{x_n})$
      $\delta_n = k_{nn} - \boldsymbol{k'_{n-1}}(\boldsymbol{x_n})^T \boldsymbol{a_n}$
      If ($\delta_n > v$) -add $\boldsymbol{x_n}$ to the dictionary
          $\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{\boldsymbol{x_n}\}$
          Compute $\boldsymbol{K'^{-1}_n}$ with $\boldsymbol{a'_n} = \boldsymbol{a_n}$
          Compute $\boldsymbol{P_n}$, $\boldsymbol{a_n}$
          $m = m + 1$
      else -dictionary unchanged
          $\mathcal{D}_n = \mathcal{D}_{n-1}$
          $q_n = \frac{P_{n-1}\boldsymbol{a_n}}{1 + \boldsymbol{a^T_n} P_{n-1} \boldsymbol{a_n}}$
          Compute $\boldsymbol{P_n}, \boldsymbol{a_n}$
• End For
```

The algorithm returns the vector $\boldsymbol{a_n}$ as well as the dictionary.

Example 2.2:

In this example, the following scheme is adopted (which is a non-linear regression task):

$$y_n = \boldsymbol{w^T_0} \boldsymbol{x_n} + 0.1(\boldsymbol{w^T_1} \boldsymbol{x_n})^2 + \boldsymbol{\eta_n},$$

where $\boldsymbol{\eta_n}$ are zero-mean i.i.d. Gaussian noise samples with $\sigma_n = 0.05$, and the coefficients of the vectors $\boldsymbol{w_0}, \boldsymbol{w_1}$ ϵ $\Re^5$ are i.i.d. samples drawn from $\mathcal{N}(0,1)$. We are going to compare the results of the Engel Kernel RLS and the Quantized KLMS. For the former, the following parameters were used: the sparsification parameter was set to 0.0005, and the Gaussian Bandwidth σ was set to 5, while for the latter, the Gaussian Bandwidth was also set to 5, while the quantization threshold q was set to 0.01. One can see in **Figure 2.3**, that KRLS achieves much better MSE as well as faster convergence, however in the expense of a more complicated algorithm.

• **Expample 2.3:**

We adopt the following chaotic series model :

$$d_n = u_n + 0.5u_n - 0.2d_{n-1} + 0.35d_{n-1}$$

$$\phi(d_n) = \begin{cases} \frac{d_n}{3(0.1+0.9d_n^2)^{1/2}} & d_n > 0 \\ \frac{-d_n^2(1-exp(0.7d_n))}{3} & d_n < 0 \end{cases}$$

$$y_n = \phi(d_n) + \eta_n,$$

where η_n is zero mean i.i.d. Gaussian noise with $\sigma_n = 0.001$, u_n is also zero-mean i.i.d. Gaussian with $\sigma^2_u = 0.0156$ and $u_n = 0.5u_n + \eta'_n$, where η'_n is also i.i.d Gaussian noise with $\sigma^2 = 0.0156$. The kernel parameter σ was set to 0.05 for the QKLMS while 1 for the KRLS. The step size μ was set to 1. Finally the sparsification parameter for the KRLS was set to 0.00001 while the quantization parameter for the QKLMS was set to $q = 0.01$. **Figure 2.4** shows the MSE achieved by the algorithms over 100 runs. It is readily observable that the KRLS achieves better MSE while both algorithms achieve similar convergence speed.

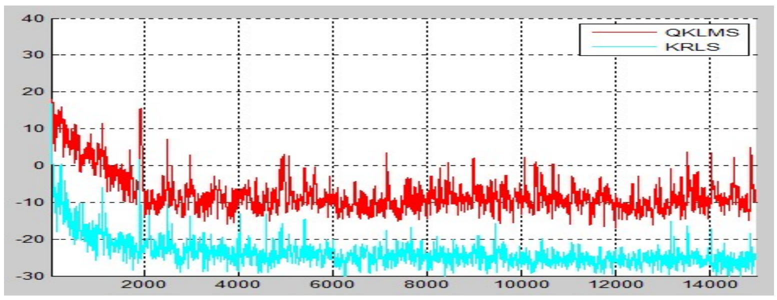

Figure 2.3: Simulations of the KRLS versus the Quantized-KLMS algorithm. The input data follow the Example 2.2 over 15000 samples. The orthogonal line in the figure, represents the MSE. In this image, one can observe, that KRLS achieves much better error floor as well as faster convergence, in the expense of a more complicated algorithm.

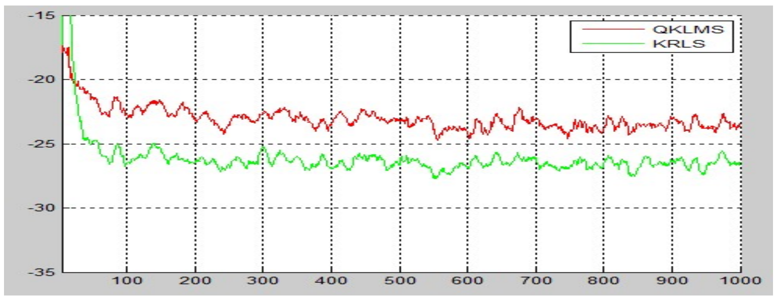

Figure 2.4: Simulations of the Engel's Kernel RLS versus the Quantized-KLMS algorithm. The input data follow the Example 2.3 over 1000 samples. The orthogonal line in the figure, represents the MSE. In this image, one can observe, that KRLS achieves much better error floor as well as similar convergence speed, in the expense of a more complicated algorithm.

3. RANDOM FOURIER FEATURES

3.1 General theory of Random Fourier Features

In the previous discussion about the Reproducing Kernel Hilbert Spaces, we mentioned the computational obstacle that occurs in real-time problems, where in every time instant, new input data are introduced to the algorithm and hence, the complexity of the solution is constantly increasing. Various solutions to this problem have been proposed, mainly comprising ad-hoc techniques. In this chapter, we are going to present an efficient and elegant solution, which not only overtakes this problem, but it is also easy to understand and to extend to other situations. But what exactly are the Random Fourier Features? Intuitively, the main idea behind this technique is that we map the input data into a randomized low-dimensional feature space, in which the initial non-linear problem, is linear and we can apply the existing fast linear methods [9]. But, what is the difference with the implicit mapping into an RKHS? Well, in this case, we apply an explicit mapping to the new space, the new space is low-dimensional and the randomized features are designed so that the inner products of the mapped data are approximately equal to those in the feature space defined by a shift-invariant kernel. Finally, is the problem of the computational complexity solved? In fact, the latter is true, because the new low-dimensional space, has a fixed, finite dimension. Hence, the solution, has a fixed size, which is independent of the number of input data.

As mentioned before, methods that operate on the kernel matrix of the data scale poorly with the size of the training dataset. On the contrary, algorithms for linear SVR's operate on the covariance matrix rather than in the kernel matrix of the training data, which produces a complexity proportional to the square of dimensionality of the space in which we are working. The following method, combines the advantages both of non-linear and linear approaches. This method competes with various state-of-the-art kernel-based classification and regression algorithms[9]. More specifically, we apply an explicit mapping of the input data to a new low-dimensional Euclidean inner-product space, using a randomized feature map: $z : \Re^d \mapsto \Re^D$, so that the inner product between a pair of mapped data points approximates their kernel evaluation, that is

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle \approx z(\boldsymbol{x})^T z(\boldsymbol{y})$$

The first question that has to be addressed, is the needed dimension of the new low-dimensional space, so that the non-linear problem is transformed into a linear one. In [9], an analysis is provided showing that a feature space that uniformly approximates popular shift-invariant kernels $\kappa(\boldsymbol{x} - \boldsymbol{y})$ to within ϵ has a dimension :

$$D = O(d\epsilon^{-2} log(\frac{1}{\epsilon^2})),$$

providing a good performance even with a lower dimension than the lower bound provided previously. Also, another advantage of this method, is that it evaluates the machine in $O(D * d)$ operations. That holds true, because, once we learn the hyperplane **w**, the linear machine can be evaluated simply by an inner product operation, that is: $f(\boldsymbol{x}) = \mathbf{w}^T z(\boldsymbol{x})$, in contrast to the case of the RKHS, where one evaluates the machine, with the help of the Representer Theorem and the kernel trick, i.e. $f(\boldsymbol{x}) = \sum_{i=1}^{N} c_i \kappa(\boldsymbol{x_i}, \boldsymbol{x})$, which requires $O(Nd)$ operations in the case of non-sparse machines.

In [9], two randomized feature maps are provided for approximating shift invariant kernels. In this manuscript, only the first one will be presented, which consists of sinusoids randomly drawn from the Fourier transform of the kernel function we want to approximate.

• The Random Fourier Features Method

The theorem in which this method relies follows:

Theorem 3.1-Bochner's Theorem [9]: *A continuous kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{x} - \boldsymbol{y})$ on $\Re^d$ is positive definite if and only if $\kappa(\delta)$ is the Fourier transform of a non-negative measure.*

What the previous theorem guarantees, is that if $\kappa(\delta)$ is properly scaled, its Fourier transform $p(\boldsymbol{\omega})$ is a proper **probability distribution**.

The set of random features consists of random Fourier bases, $cos(\boldsymbol{\omega}^T\boldsymbol{x} + b)$, where $\boldsymbol{\omega} \epsilon \Re^d$ and $b \epsilon \Re$. What we are doing, is that we project data points into a random chosen line and then we pass the resulting scalar through a sinusoidal function. Now we have to address each component of this projection. Lets start with $\boldsymbol{\omega}$. Assume a given, shift invariant kernel, which we want to approximate, $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{x} - \boldsymbol{y})$. Finally we have **x**,**y** $\epsilon \Re^D$. Now, $p(\boldsymbol{\omega})$ is the Fourier transform of $\kappa(\delta)$, that is:

$$p(\boldsymbol{\omega}) = \frac{1}{2 * \pi^D} \int\limits_{\Re^D} \kappa(\delta) e^{-i\boldsymbol{\omega}^T\delta} d\delta \qquad : D - dimensional\ Fourier\ transform$$

and

$$\kappa(\boldsymbol{x} - \boldsymbol{y}) = k(\delta) = \int\limits_{\Re^D} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^T(\boldsymbol{x}-\boldsymbol{y})} d\boldsymbol{\omega} \qquad : (12) Inverse\ D - dimensional\ Fourier\ transform$$

Now assume that, $\zeta_{\boldsymbol{\omega}}(\boldsymbol{x}) = e^{i\boldsymbol{\omega}^T\boldsymbol{x}}$, then:

$$\kappa(\boldsymbol{x} - \boldsymbol{y}) = E_{\boldsymbol{\omega}}[\zeta_{\boldsymbol{\omega}}(\boldsymbol{x})\zeta^*_{\boldsymbol{\omega}}(\boldsymbol{y})] \quad (13)\ (since\ p(\boldsymbol{\omega})\ is\ a\ probability\ distribution,)$$

where the random variable $\boldsymbol{\omega}$ follows the pdf $p(\boldsymbol{\omega})$.
Since, $\kappa(\delta)$ and $p(\boldsymbol{\omega})$ are both real, the integral (12) converges when the complex exponentials are replaced with cosines.Therefore , we expect to obtain a real-valued mapping that satisfies the condition (13). So, we set:

$$Z_{\boldsymbol{\omega}}(\boldsymbol{x}) = \sqrt{2} cos(\boldsymbol{\omega}^T\boldsymbol{x} + b),$$

where $\boldsymbol{\omega}$ is drawn from $p(\boldsymbol{\omega})$ and b is drawn uniformly from $[0, 2\pi]$. Then,

$$\kappa(\boldsymbol{x} - \boldsymbol{y}) = E_{\boldsymbol{\omega},b}[Z_{\boldsymbol{\omega}}(\boldsymbol{x})Z_{\boldsymbol{\omega}}(\boldsymbol{y})]$$

Proof:

$$\kappa(\boldsymbol{x} - \boldsymbol{y}) = \int\limits_{\Re^D} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^T(\boldsymbol{x}-\boldsymbol{y})} d\boldsymbol{\omega} = \int\limits_{\Re^D} p(\boldsymbol{\omega}) cos(\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{y})) d\boldsymbol{\omega} + i \int\limits_{\Re^D} p(\boldsymbol{\omega}) sin(\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{y})) d\boldsymbol{\omega}$$

The second term of the last equation, is equal to zero for real kernels. Also:

$$Z_{\boldsymbol{\omega}}(\boldsymbol{x})Z_{\boldsymbol{\omega}}(\boldsymbol{y}) = 2cos(\boldsymbol{\omega}^T\boldsymbol{x} + b)cos(\boldsymbol{\omega}^T\boldsymbol{y} + b) = cos(\boldsymbol{\omega}^T(\boldsymbol{x} + \boldsymbol{y}) + 2b) + cos(\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{y}))$$

Hence,

$$E_{\boldsymbol{\omega},b}[Z_{\boldsymbol{\omega}}(\boldsymbol{x})Z_{\boldsymbol{\omega}}(\boldsymbol{y})] = E_{\boldsymbol{\omega},b}[cos(\boldsymbol{\omega}^T(\boldsymbol{x} + \boldsymbol{y}) + 2b)] + E_{\boldsymbol{\omega},b}[cos(\boldsymbol{\omega}^T(\boldsymbol{x} - \boldsymbol{y}))] =$$

$$\int\limits_0^{2\pi} \int\limits_{\Re^D} \frac{1}{2\pi} p(\boldsymbol{\omega}) cos(\boldsymbol{\omega}^T(\boldsymbol{x} + \boldsymbol{y}) + 2b) d\boldsymbol{\omega} db + \kappa(\boldsymbol{x} - \boldsymbol{y}) =$$

$$= \int\limits_0^{2\pi} \frac{1}{2\pi} p(\boldsymbol{\omega}) \int\limits_{\Re^D} cos(\boldsymbol{\omega}^T(\boldsymbol{x} + \boldsymbol{y}) + 2b) d\boldsymbol{\omega} db + \kappa(\boldsymbol{x} - \boldsymbol{y}) = \kappa(\boldsymbol{x}, \boldsymbol{y})$$

Now that all the theoretical parts are given, we are ready to produce a method, which one has to apply, in order to use this technique in various problems.

Method 3.1-Random Fourier Features:

- Select a shift invariant kernel (e.g. Gaussian Kernel) and the respective parameters.
- Calculate the Fourier transform $p(\boldsymbol{\omega})$ of $\kappa(\cdot,\cdot)$.
- Choose D random elements, $\boldsymbol{\omega_1, \omega_2, ..., \omega_D}$ from $p(\boldsymbol{\omega})$
- Choose D random elements, $b_1, b_2, ..., b_D$ uniformly from $[0, 2\pi]$
- Map input data with the following mapping:

$$\boldsymbol{x} \mapsto Z_{\boldsymbol{\omega}}(\boldsymbol{x}) = \frac{\sqrt{2}}{\sqrt{D}} \begin{pmatrix} cos(\boldsymbol{\omega_1^T x} + b_1) \\ cos(\boldsymbol{\omega_2^T x} + b_2) \\ . \\ . \\ . \\ cos(\boldsymbol{\omega_D^T x} + b_D) \end{pmatrix}$$

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) \approx Z_{\boldsymbol{\omega}}(\boldsymbol{x}) Z_{\boldsymbol{\omega}}(\boldsymbol{y})$$

3.2 The Random Fourier Features LMS

In this section, we are going to provide a linearized from of the KLMS algorithm, based on the Random Fourier Features method [10], given in section **3.1**. Although the sparsification techniques mentioned in the previous section, concerning the KLMS, KRLS, etc. algorithms reduce the size of the solution significantly, they require many computational resources, since at every time instant, a sequential search over all the elements of the dictionary has to take place, so that the new center $\boldsymbol{x_n}$ is added or not to the latter. Hence, it makes the algorithm slower, something that becomes a problem when the initial input space is of large dimension. Furthermore, as indicated and before, such techniques are theoretically poor, as they are built around ad-hoc arguments, which make the theoretical analysis difficult. Finally, another worth to be mentioned argument, is that distributed learning with the previous algorithm gets quite difficult as the dictionary has to be exchanged among the network nodes.

As mentioned in section **3.1**, we first have to define the following mapping:

$$Z_{\boldsymbol{\omega}} : \Re^d \mapsto \Re^D$$

, such that:

$$Z_{\boldsymbol{\omega}} = \sqrt{\frac{2}{D}} \begin{pmatrix} cos(\boldsymbol{\omega_1^T x} + b_1) \\ cos(\boldsymbol{\omega_2^T x} + b_2) \\ . \\ . \\ . \\ cos(\boldsymbol{\omega_D^T x} + b_D) \end{pmatrix},$$

where $\boldsymbol{\omega_1, \omega_2, ..., \omega_D}$ are drawn randomly from $p(\boldsymbol{\omega})$ and $b_1, b_2, ..., b_D$ are drawn uniformly from $[0, 2\pi]$. Also, we will adopt the Gaussian kernel (which is a shift-invariant kernel), that is:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}}$$

The Fourier transform of the Gaussian kernel is:

$$p(\boldsymbol{\omega}) = \left(\frac{\sigma}{\sqrt{2\pi}}\right)^D e^{-\frac{\sigma^2\|\boldsymbol{\omega}\|}{2}}$$

In fact, the RFFLMS, is like the linear LMS, but with different training data, i.e. $\{(Z_{\boldsymbol{\omega}}(\boldsymbol{x_n}), \boldsymbol{y_n})$, n=1,2,...$\}$. Once we compute, the fixed sized hyperplane $\boldsymbol{\theta}$, the prediction is held easily just by a simple inner product operation, that is:

$$y'_n = \boldsymbol{\theta}^T Z_{\boldsymbol{\omega}}(\boldsymbol{x}_n),\ \forall\ \boldsymbol{x}_n,$$

with the error term being computed as :

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^T Z_{\boldsymbol{\omega}}(\boldsymbol{x}_n),$$

while the gradient scheme adopted is:

$$\boldsymbol{\theta_n} = \boldsymbol{\theta_{n-1}} + \mu e_n Z_{\boldsymbol{\omega}}(\boldsymbol{x}_n)$$

as in order to find $\boldsymbol{\theta}\ \epsilon\ \Re^D$, we have to minimize the MSE: $J_n = E[e_n^2]$ at each time instant n. Since we have described the Random Fourier Features in previous section, we are now ready to provide the algorithm:
Algorithm 3.1-The RFFLMS [10]:

- Initialization
  Set $\boldsymbol{\theta} = 0$
  Select the step size μ
  Select the dimension of the new Euclidean space,D.
  Select the kernel parameters(σ)
  Draw D samples from $p(\boldsymbol{\omega})$ and D scalars uniformly for $[0, 2\pi]$
- For n=1,2,...

  $Z_{\boldsymbol{\omega}}(\mathbf{x}_n) = \sqrt{(\frac{2}{D})} cos(\boldsymbol{\omega}^T \mathbf{x}_n + b)$-Apply the explicit map

  $e_n = y_n - \boldsymbol{\theta}_{n-1}^T Z_{\boldsymbol{\omega}}(\mathbf{x}_n)$-Compute the error

  $\boldsymbol{\theta_n} = \boldsymbol{\theta_{n-1}} + \mu e_n Z_{\boldsymbol{\omega}}(\mathbf{x}_n)$
- End For

It can easily be seen, that after n-1 steps, the algorithm will give a solution,

$$\boldsymbol{\theta} = \mu \sum_{i=1}^{n-1} e_i Z_{\boldsymbol{\omega}}(\boldsymbol{x}_i)$$

which is approximately equal to the output of the standard KLMS, since

$$y'_n = \mu \sum_{i=1}^{n-1} e_i Z_{\boldsymbol{\omega}}(\boldsymbol{x}_i) Z_{\boldsymbol{\omega}}(\boldsymbol{x}_n) \approx \mu \sum_{i=1}^{n-1} e_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_n)$$

As a consequence, in order to study the convergence properties, one has to assume that the data pairs are generated as a linear combination of some kernel centres in the respective RKHS, that is:

$$y_n = \sum_{m=1}^{M} a_m \kappa(\boldsymbol{c_m}, \boldsymbol{x}_n) + \boldsymbol{\eta_n},$$

where $\boldsymbol{c_1, ..., c_M}$ are fixed centres, $\mathbf{x}_n$ are zero-mean i.i.d. samples drawn from a Gaussian with covariance matrix $\sigma_x^2 \mathbf{I}_d$, and $\boldsymbol{\eta_n}$ are i.i.d. noise samples from $\mathcal{N}(0, \sigma_n^2)$[10]. Then[10], it can be proved that the optimal solution is given by:

$$\boldsymbol{\theta}_{opt} = argminE[e_n^2] = Z_C \boldsymbol{a} + R_{zz}^{-1} E[\boldsymbol{\eta'_n} Z_{\boldsymbol{\omega}}(\boldsymbol{x}_n)],$$

where $Z_C = (Z_{\boldsymbol{\omega}}(\boldsymbol{c_1}), ..., Z_{\boldsymbol{\omega}}(\boldsymbol{c_M}))^T$, $\boldsymbol{a} = (a_1, ..., a_M)^T$, $R_{zz} = E[Z_{\boldsymbol{\omega}}(\mathbf{x}_n) Z_{\boldsymbol{\omega}}(\mathbf{x}_n)^T]$ and $\boldsymbol{\eta'_n}$ is the approximation error between the noise-free component of y_n and its approximation, using the Random Fourier Features. It is worth mentioning that the latter error, can be made very small, if the value of D (the dimension of the new

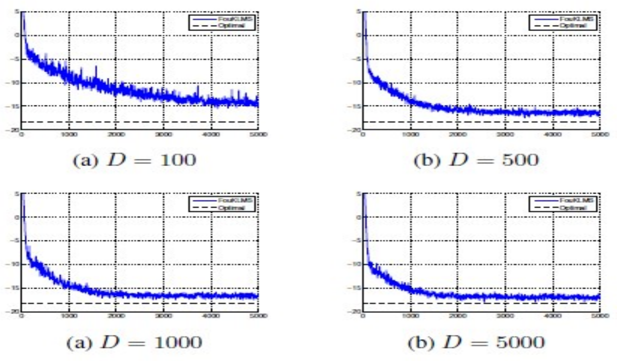

Figure 3.1: Graph taken from [10]. Simulations of the RFFLMS for various values of D, applied in data pairs generated by the linear combination of some kernel centres. The results are averaged over 100 runs. The horizontal line in the figure, represents the approximation of the steady-state MSE given in theorem 3.2.

Euclidean product space) is sufficiently large [9]. Now, lets provide a Lemma, giving the sufficient condition in order for R_{zz} to be positive definite, and hence invertible. This analysis is provided in [10].

Lemma 3.1 [10]: *Consider a selection of sammples, $\boldsymbol{\omega_1, ..., \omega_D}$, drawn from the Fourier Transform of the Gaussian Kernel, such that $\boldsymbol{\omega_i} \neq \boldsymbol{\omega_j}$, for any $i \neq j$. Then the matrix $R_{zz} = E[Z_{\boldsymbol{\omega}}(\boldsymbol{x_n})Z_{\boldsymbol{\omega}}(\boldsymbol{x_n})^T]$ is strictly positive definite.*

The eigenvalues of R_{zz} are very important in the convergence study of the algorithm. Since it is strictly positive definite, its eigenvalues satisfy: $0 < \lambda_1 \leq ... \leq \lambda_D$. Applying an analysis similar to the one in the standard LMS[10], the following theorem can be proved:

Theorem 3.2 [10]: *For datasets generated by the linear combinations of some kernel centres, we have:*

- *If the step update parameter satisfies $0 < \mu < \frac{2}{\lambda_D}$, then RFFLMS converges in the mean, i.e. $E[\boldsymbol{\theta}_n - \boldsymbol{\theta}_{opt}] \to 0$*

- *The optimal MSE (which is achieved when one replaces $\boldsymbol{\theta}_n$ with $\boldsymbol{\theta}_{opt}$)is given by:*

$$J_n^{opt} = \sigma_n^2 + E[\boldsymbol{\eta_n'}] - E[\boldsymbol{\eta_n'} Z_{\boldsymbol{\omega}}(\boldsymbol{x_n})] R_{zz}^{-1} E[\boldsymbol{\eta_n'} Z_{\boldsymbol{\omega}}(\boldsymbol{x_n})^T].$$

  For large enough D, we have $J_n^{opt} \approx \sigma_n^2$

- *The excess MSE is given by $J_n^{ex} = J_n - J_n^{opt} = trace(R_{zz}A_n)$, where $A_n = E[(\boldsymbol{\theta}_n - \boldsymbol{\theta}_{opt})(\boldsymbol{\theta}_n - \boldsymbol{\theta}_{opt})^T]$*

- *If the step update parameter satisfies $0 < \mu < \frac{1}{\lambda_D}$, the A_n converges. For large enough n and D, we can approximate A_n's evolution as $A_{n+1} \approx A_n - \mu(R_{zz}A_n + A_n R_{zz}) + \mu^2 \sigma_n^2 R_{zz}$. Using this model, we can approximate the steady-state MSE*

Now, recall the Quantized KLMS algorithm, given in section **2.2**. We are going to present some examples that compare the RFFLMS algorithm with the QKLMS as well as the FOBOS-KLMS, given that the same kernel (Gaussian) is applied, with the same parameter (σ), as well as the same fixed step size μ. The q in the QKLMS (which highly controls the achieved MSE) was chosen so that it has a value close to the optimal, in [10], while the parameter D of RFFLMS, which is also highly correlated to the performance of the algorithm was chosen so that the lowest steady-state MSE is achieved.

• Example 3.1:
The first example is illustrated in **Figure 3.1**, and it is worth mentioning that the attained MSE is close to the approximation given in **Theorem 3.2**! The input vectors as well as the noise are i.i.d. Gaussian samples, drawn from $\mathcal{N}(0, I)$ and $\mathcal{N}(0, 0.1I)$ respectively, so as the parameters of the expansion, which were drawn from $\mathcal{N}(0, 25)$.

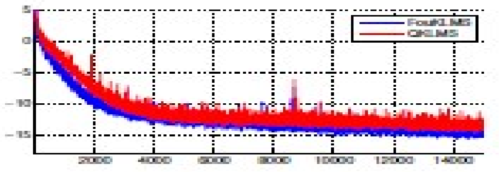

Figure 3.2: Graph taken from [10]. The specifics of the problem are presented in the **Example 3.2**. MSE (in Decibel) of the QKLMS is denoted with the red colour, while RFFLMS's with the blue. The results are averaged over 1000 runs.

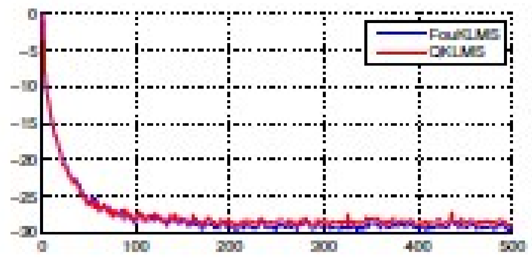

Figure 3.3: Graph taken from [10]. The specifics of the problem are presented in the **Example 3.3**. MSE (in Decibel) of the QKLMS is denoted with the red colour, while RFFLMS's with the blue. The results are averaged over 1000 runs.

The kernel parameter was, $\sigma = 5$, while the step size was $\mu = 1$. The Figure shows the evolution of the MSE for 100 realizations of the experiment.

• **Example 3.2:**
In the second example, the following scheme is adopted (which is a non-linear regression task):

$$y_n = \boldsymbol{w_0}^T \boldsymbol{x_n} + 0.1(\boldsymbol{w_1}^T \boldsymbol{x_n})^2 + \boldsymbol{\eta_n},$$

where $\boldsymbol{\eta_n}$ are zero-mean i.i.d. Gaussian noise samples with $\sigma_n = 0.05$, and the coefficients of the vectors $\boldsymbol{w_0}, \boldsymbol{w_1}$ ϵ $\Re^5$ are i.i.d. samples drawn from $\mathcal{N}(0, 1)$. The kernel parameters as well as the step size are the same as in Example 3.1. The quantization parameters ϵ for the QKLMS is set to $\epsilon = 5$ (average dictionary size M=100). Finally the dimension of the new Euclidean product space used by the RFFLMS is D=300. **Figure 3.2** shows the evolution of the MSE for both QKLMS and RFFLMS running 1000 experiments over 15000 samples.

• **Example 3.3:**
Finally, we adopt the following chaotic series:

$$d_n = \frac{d_{n-1}}{1 + d_{n-1}^2} + u_{n-1}^3, \qquad y_n = d_n + \eta_n,$$

with η_n and u_n being zero-mean i.i.d. Gaussian with $\sigma_n = 0.01$ and $\sigma_u = 0.15$ respectively. The kernel parameter is set to $\sigma = 0.05$ while the step size is $\mu = 1$. Initialization was $d_1 = 1$. In **Figure 3.3** the resulted MSE for both QKLMS and RFFLMS is presented over 1000 experiments with 500 samples. The $\epsilon = 0.01$ for the QKLMS, while the dimension of the space used for the explicit mapping in the RFFLMS was set to $D = 100$ (number of random Fourier coefficients).

As it is easily observed in the examples, the proposed algorithm exhibits similar convergence performance to the standard QKLMS, as well as MSE, while as can be seen in **Algorithm 3.1**, it has a much easier implementation (due to its simplicity) while its "linear" characteristics make its generalization, to other settings, quite easy. Recall now the **Example 2.1** provided in **section 2.3**. One can observe that the RFFLMS, exhibits better error floors as well as convergence speed than the FOBOS-KLMS, while, it is way easier to implement, since it has a fixed solution! **Figure 3.4** compares FOBOS-KLMS,QKLMS and RFFLMS.

3.3 The RFF RLS algorithm

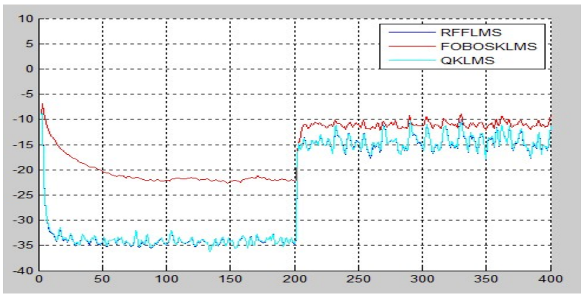

Figure 3.4: The specifics of the problem are presented in the **Example 2.1**. MSE(in Decibel) of the FOBOS-KLMS is denoted with the red colour, QKLMS's with Cyan while RFFLMS's with the blue. The results are averaged over 200 runs, while the input data were 400.

We are now ready to extend the method of the Random Fourier Features to the online RLS algorithm. It can readily be seen that the KRLS algorithm provided in section **2.5**, is highly complex to implement while having the same problem that occurs in the KLMS algorithm, when the input dataset is large. Due to the latter, as we saw and before, a sparcification technique was introduced to the algorithm (e.g. a dictionary). One can compare the the standard RLS algorithm with the KRLS one. The former is way much easier to understand and implement, while its analysis is held with more simple arguments. Hence, it is now obvious why we present the "linearized" KRLS algorithm with the help of the Random Fourier Features, which takes into advantage the simplicity of the standard RLS, as well as the strengths of the non-linear methods.

Once again, we are going to use the **Method 3.1**, provided in the section **3.1**. So we have to define the following mapping:

$$Z_\omega \ : \ \Re^d \mapsto \Re^D,$$

such that:

$$Z_{\boldsymbol{\omega}} = \sqrt{\frac{2}{D}} \begin{pmatrix} cos(\boldsymbol{\omega_1}^T \boldsymbol{x} + b_1) \\ cos(\boldsymbol{\omega_2}^T \boldsymbol{x} + b_2) \\ . \\ . \\ . \\ cos(\boldsymbol{\omega_D}^T \boldsymbol{x} + b_D) \end{pmatrix},$$

where $\boldsymbol{\omega_1}, ..., \boldsymbol{\omega_D}$ are drawn randomly from $p(\boldsymbol{\omega})$(with $p(\boldsymbol{\omega})$ being the Fourier transform of the Kernel function we chose) and $b_1, ..., b_D$ are drawn uniformly from $[0, 2\pi]$. In our paradigm, we are going to use the Gaussian Kernel, i.e.:

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}}$$

Its Fourier Transform is:

$$p(\boldsymbol{\omega}) = (\frac{\sigma}{\sqrt{2\pi}})^D e^{-\frac{\sigma^2\|\boldsymbol{\omega}\|}{2}}$$

In this case, as it was in the case of the RFFLMS algorithm, the RFFRLS, is exactly like the linear RLS, but with different training data, i.e. $\{(Z_\omega(\boldsymbol{x_n}), y_n), n = 1, 2...\}$. Once we compute the fixed sized hyperplane $\boldsymbol{\theta}$, the prediction is once again held easily just by a simple inner product operation, that is:

$$y'_n = \boldsymbol{\theta}^T Z_\omega(\boldsymbol{x_n}), \ \forall \ \boldsymbol{x_n}$$

Now, recall the analysis of the standard RLS.As we stated before, the RFFRLS algorithm, is exactly the same, but with different training data. So we can just apply the new dataset, into the already exported equations from the analysis of the standard RLS. Then the error term is being computed as:

$$e_n = y_n - \boldsymbol{\theta_{n-1}^T} Z_\omega(\boldsymbol{x_n})$$

while the Kalman Gain, is:

$$K_n = \frac{z_n}{\beta + Z_\omega(\boldsymbol{x_n})^T z_n},$$

where

$$z_n = P_n Z_\omega(\boldsymbol{x_n})$$

Now the update of $\boldsymbol{\theta}$ will become:

$$\boldsymbol{\theta_n} = \boldsymbol{\theta_{n-1}} + K_n e_n$$

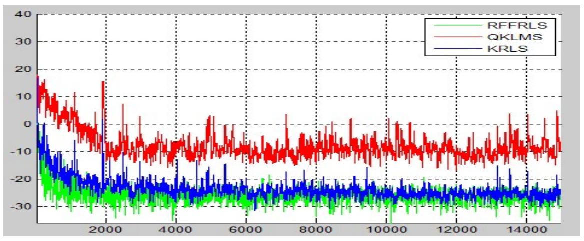

Figure 3.5: Simulations on data pairs generated as described in (14). The RFFRLS is denoted with the green colour, the KRLS provided in [11], with black while the QKLMS with red. One can easily observe that the former converges after 2000 data samples, while the KRLS after about 4000. The horizontal line in the figure, represents the approximation of the steady-state MSE.

while the update of P_n is:

$$P_n = \beta^{-1}P_{n-1} - \beta^{-1}K_n z_n^T$$

And, that's all we had to do. Now we are ready to derive the algorithm !
Algorithm 3.2-The RFFRLS:

- Initialization
  Set $\boldsymbol{\theta} = \mathbf{0}$
  Select a value for the forgetting factor β-A good value is 0.999
  Select a value for the regularization parameter λ-A good value is 0.0001
  Select the dimension of the new Euclidean space, D
  Select the kernel parameters(σ)
  Draw D samples for $p(\boldsymbol{\omega})$ and D scalars uniformly from $[0, 2\pi]$
  Set P_0 to the zero matrix of appropriate dimensions

- For n=1,2,...

$$Z_\omega(\boldsymbol{x_n}) = \sqrt{\tfrac{2}{D}}cos(\boldsymbol{\omega}^T\boldsymbol{x_n} + b)$$
$$e_n = (y_n - \boldsymbol{\theta}^T Z_\omega(\boldsymbol{x_n}))$$
$$z_n = P_{n-1}z_n$$
$$K_n = \frac{z_n}{\beta + Z_\omega(\boldsymbol{x_n})^T z_n}$$
$$\boldsymbol{\theta} = \boldsymbol{\theta} + K_n e_n$$
$$P_n = \beta^{-1}P_{n-1} - \beta^{-1}K_n z_n^T$$

- End For

It is worth mentioning that the RFFRLS, is way much easier to understand than the KRLS provided in section **2.5**. The only thing we have to do, is to replace the input data, in the equations of the standard RLS, with their Random Fourier Features. Now, recall the KRLS algorithm provided in [11]. We are going to present an example, comparing the KRLS, the QKLMS and the RFFRLS algorithm, given the same kernel parameter (σ) as well as the same kernel (Gaussian). The parameter v, for the sparsification in the KRLS was set to $v = 0.0005$ while concerning the RFFRLS: the forgetting factor β was set $\beta = 0.9995$, the dimension of the new Euclidean product space D, was set to $D = 300$ while the regularization parameter λ was set to $\lambda = 0.0001$. As can be seen in **Figure 3.5**, the RFFRLS performs as good as the original KRLS tested on samples generated by:

$$y_n = \boldsymbol{w_0^T x_n} + 0.1(\boldsymbol{w_1^T x_n})^2 + \eta_n \qquad (14)$$

however, it converges almost at half time!
• Expample 3.4:
Finally, we adopt the following chaotic series model [10].

$$d_n = u_n + 0.5u_n - 0.2d_{n-1} + 0.35d_{n-1}$$

$$\phi(d_n) = \begin{cases} \frac{d_n}{3(0.1+0.9d_n^2)^{1/2}} & d_n > 0 \\ \frac{-d_n^2(1-exp(0.7d_n))}{3} & d_n < 0 \end{cases}$$

$$y_n = \phi(d_n) + \eta_n,$$

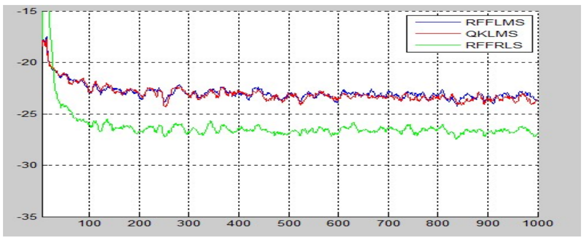

Figure 3.6: Simulations on data pairs generated as described in Example 3.4. It is observed that the RFFRLS exhibits better MSE however it has slower convergence speed.

where η_n is zero mean i.i.d. Gaussian noise with $\sigma_n = 0.001$, u_n is also zero-mean i.i.d. Gaussian with $\sigma_u^2 = 0.0156$ and $u_n = 0.5u_n + \eta_n'$, where η_n' is also i.i.d Gaussian noise with $\sigma^2 = 0.0156$. The kernel parameter σ was set to 0.05 for the RFFLMS and the QKLMS while 1 for the RFFRLS. The step size μ was set to 1, while the beta for the RFFRLS was set to 0.9995. Finally the regularization parameter λ for the RFFRLS was set to 0.00001, the quantization parameter for the QKLMS was set to $q = 0.01$, while the number of random fourier coefficients for the RFFLMS and the RFFRLS was set to D=200. **Figure 3.6** shows the MSE achieved by those three algorithms over 100 runs.

4. KALMAN FILTERS

4.1 Linear Kalman Filter

This chapter focuses on solving estimation tasks, given that the we are working in a time-varying environment, that is, the statistical properties of the input random variables may change as time moves forward. Such problems, can be solved with the help of the so-called Kalman Filters, proposed by Rudolf Kalman [1], and will be presented in this Chapter. Firstly, it suffices to present the linear Kalman Filters. Then we will be able to extend this technique, in order to solve non linear tasks of this type, using the Reproducing Kernel Hilbert Spaces(RKHS). Finally, we will propose an algorithm, using the Random Fourier Features technique, solving a non-linear estimation task with time-varying statistics. The latter, amplifies the argument that the Random Fourier Features technique, can be easily extended to a wide variety of problems, making the solution easier to implement as well as to understand, using both the advantages of the linear and the non-linear methods.

A Kalman Filter, is an online (and hence a recursive) optimal estimator, minimizing the mean square error of the estimated parameters, given that all noise is Gaussian. If the noise is not Gaussian, the Kalman Filter is the best linear estimator. In recent years, Kalman Filters gain more and more attention due to their optimality in a wide variety of problems as well as their ease to be understood and implemented. Now, lets formulate the problem. Given two known matrices $F_n \epsilon \Re^{lxl}$, $H_n \epsilon \Re^{kxk}$ and the following system:

$$\boldsymbol{x_n} = F_n\boldsymbol{x_{n-1}} + \boldsymbol{\eta_n} \qquad State\ Equation \qquad (15)$$

$$\boldsymbol{y_n} = H_n\boldsymbol{x_n} + \boldsymbol{v_n}, \qquad Output\ Equation \qquad (16)$$

where $\boldsymbol{\eta_n}, \boldsymbol{x_n} \epsilon \Re^l$ and $\boldsymbol{v_n}, \boldsymbol{y_n} \epsilon \Re^k$, the task is to estimate the values of the states $\boldsymbol{x_n}$ given the output vectors of the system $\boldsymbol{y_n}$. Also, $\boldsymbol{\eta_n}, \boldsymbol{v_n}$ are noise vectors known as process noise and measurement noise, respectively. As mentioned in the previous paragraphs, the problem has time-varying statistics, which can readily be observed in the State Equation, which provides the information related to the time-varying dynamics [1] of the system. The previous model is known as the state-space model of $\boldsymbol{y_n}$. In order to find a time varying estimator, $\boldsymbol{x'_n}$, in the framework presented before, we have to adopt the following assumptions [1]:

$$E[\boldsymbol{\eta_n}\boldsymbol{\eta_n^T}] = Q_n,\ E[\boldsymbol{\eta_n}\boldsymbol{\eta_m^T}] = O\ for\ n \neq m$$

$$E[\boldsymbol{v_n}\boldsymbol{v_n^T}] = R_n,\ E[\boldsymbol{v_n}\boldsymbol{v_m^T}] = O\ for\ n \neq m$$

$$E[\boldsymbol{\eta_n}\boldsymbol{v_m^T}] = O,\ \forall\ n, m$$

$$E[\boldsymbol{\eta_n}] = E[\boldsymbol{v_n}] = O,\ \forall\ n$$

where O is the matrix with zero elements. The latter indicates that the measurement and the process noise are uncorrelated while noise vectors at different time instants are also uncorrelated. Finally, we assume that matrices R_n, Q_n are known. Now, define the prior estimator as:

$$\boldsymbol{x'_{n|n-1}}$$

which is the estimate of the state variable, given the observations up to time instant $n-1$. Also, define the posterior estimator as:

$$\boldsymbol{x'_{n|n}}$$

which is the update of the prior estimator, after the observation y_n is obtained. Furthermore, define the respective error covariance matrix, for each time instant, that is:

$$P_{n-1|n-1} = E[\boldsymbol{e_{n-1|n-1}}\boldsymbol{e^T_{n-1|n-1}}],$$

where $\boldsymbol{e_{n-1|n-1}} = \boldsymbol{x_{n-1}} - \boldsymbol{x'_{n-1|n-1}}$.

•$\boldsymbol{Step\ 1}$: $Predict\ \boldsymbol{x'_{n|n-1}}$
Now, recall the state equation (15), and exclude the noise. Then, at each time instant we can predict the prior estimator, using the previous posterior one, that is:

$$\boldsymbol{x'_{n|n-1}} = F_n\boldsymbol{x'_{n-1|n-1}} \qquad (17)$$

$\bullet$***Step 2*** : *Estimate* $P_{n|n-1}$
Now, obtain the respective error covariance matrix,

$$P_{n|n-1} = E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})^T] \quad (18)$$

It is obvious, that $\boldsymbol{x_n}$ is not known, hence one can use (17) and (15) to overtake this problem, since:

$$\boldsymbol{e_{n|n-1}} := \boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}} = F_n\boldsymbol{x_{n-1}} + \boldsymbol{\eta_n} - F_n\boldsymbol{x'_{n-1|n-1}} = F_n\boldsymbol{e_{n-1|n-1}} + \boldsymbol{\eta_n} \quad (19)$$

Hence,

$$P_{n|n-1} = E[F_n\boldsymbol{e_{n-1|n-1}}\boldsymbol{e^T_{n-1|n-1}}F_n^T] + E[F_n\boldsymbol{e_{n-1|n-1}}\boldsymbol{\eta_n^T}] + E[\boldsymbol{\eta_n}\boldsymbol{e^T_{n-1|n-1}}F_n^T] + E[\boldsymbol{\eta_n}\boldsymbol{\eta_n^T}] \Rightarrow$$

$$P_{n|n-1} = F_nP_{n-1|n-1}F_n^T + Q_n \quad (20)$$

$\bullet$***Step 3*** : *Estimate* $\boldsymbol{x'_{n|n}}$
At this point, we will adopt the following recursion:

$$\boldsymbol{x'_{n|n}} = \boldsymbol{x'_{n|n-1}} + K_n\boldsymbol{e_n}, \quad (21)$$

where

$$\boldsymbol{e_n} := \boldsymbol{y_n} - H_n\boldsymbol{x'_{n|n-1}}$$

while matrix K_n is known as the Kalman gain. The latter, controls the amount of correction and it is computed so that:

$$J(K_n) = E[\boldsymbol{e^T_{n|n}}\boldsymbol{e_{n|n}}] = trace(P_{n|n}) = trace(E[\boldsymbol{e_{n|n}}\boldsymbol{e^T_{n|n}}])$$

is minimized. However,

$$\boldsymbol{e_{n|n}} = \boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}} = \boldsymbol{x_n} - \boldsymbol{x'_{n-1|n-1}} - K_n\boldsymbol{e_n} \Rightarrow$$

$$\boldsymbol{e_{n|n}} = \boldsymbol{e_{n|n-1}} - K_n\boldsymbol{e_n} \quad (a)$$

Thus,

$$J(K_n) = trace(E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_{n|n-1}}] - E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_n}K_n^T] - E[K_n\boldsymbol{e_n}\boldsymbol{e^T_{n|n-1}}] + E[K_n\boldsymbol{e_n}\boldsymbol{e^T_n}K_n^T]) \Rightarrow$$

$$J(K_n) = trace(P_{n|n-1}) - trace(E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_n}K_n^T]) - trace(E[K_n\boldsymbol{e_n}\boldsymbol{e^T_{n|n-1}}]) + trace(E[K_n\boldsymbol{e_n}\boldsymbol{e^T_n}K_n^T])$$

Now, we are taking the gradient with respect to matrix K:

$$\nabla_K J(K_n) = -E[\boldsymbol{e_n}\boldsymbol{e^T_{n|n-1}}] - E[\boldsymbol{e_n}\boldsymbol{e^T_{n|n-1}}] + E[(\boldsymbol{e_n}\boldsymbol{e^T_n} + \boldsymbol{e_n}\boldsymbol{e^T_n})K_n^T],$$

and equate it to zero:

$$\nabla_K J(K_n) = 0 \Leftrightarrow E[\boldsymbol{e_n}\boldsymbol{e^T_{n|n-1}}] = E[\boldsymbol{e_n}\boldsymbol{e^T_n}]K_n^T \Rightarrow$$

$$K_nE[\boldsymbol{e_n}\boldsymbol{e^T_n}] = E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_n}] \quad (b)$$

Furthermore,

$$\boldsymbol{e_n} = \boldsymbol{y_n} - H_n\boldsymbol{x'_{n|n-1}} = H_n\boldsymbol{x_n} + \boldsymbol{v_n} - H_n\boldsymbol{x'_{n|n-1}} = H_n(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}}) + \boldsymbol{v_n} \Rightarrow$$

$$\boldsymbol{e_n} = H_n\boldsymbol{e_{n|n-1}} + v_n \quad (c)$$

Thus,

$$S_n = E[\boldsymbol{e_n}\boldsymbol{e^T_n}] = E[H_n\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_{n|n-1}}H_n^T] + E[H_n\boldsymbol{e_{n|n-1}}\boldsymbol{v^T_n}] + E[\boldsymbol{v_n}\boldsymbol{e^T_{n|n-1}}H_n^T] + E[\boldsymbol{v_n}\boldsymbol{v^T_n}]$$

$$S_n = H_nP_{n|n-1}H_n^T + R_n \quad (d)$$

Finally,

$$E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_n} = E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_{n|n-1}}H_n^T] + E[\boldsymbol{e_{n|n-1}}\boldsymbol{v^T_n}] = P_{n|n-1}H_n^T$$

So, from (b) $\Rightarrow$

$$K_n = E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_n}](E[\boldsymbol{e_n}\boldsymbol{e^T_n}])^{-1} \Rightarrow$$

$$K_n = P_{n|n-1}H_n^TS_n^{-1} \quad (22a)$$

where

$$S_n = H_nP_{n|n-1}H_n^T + R_n \quad (22b)$$

$\bullet$***Step 4*** : $\textit{Estimate } P_{n|n}$(will be used in the next iteration)

$$P_{n|n} = E[\boldsymbol{e_{n|n}}\boldsymbol{e^T_{n|n}}] =^{(a)} E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_{n|n-1}}] - E[\boldsymbol{e_{n|n-1}}\boldsymbol{e^T_n}K^T_n] - E[K_n\boldsymbol{e_n}\boldsymbol{e^T_{n|n-1}}] + E[K_n\boldsymbol{e_n}\boldsymbol{e^T_n}K^T_n] \Rightarrow$$

$$P_{n|n} = P_{n|n-1} - K_n H_n P^T_{n|n-1} \qquad (23)$$

Now that all the equations are exported, lets provide the algorithm. In [1], some initial conditions are provided that will also be adopted here, that is:

$$\boldsymbol{x'_{1|0}} = E[x_1]$$

$$P_{1|0} = E[(\boldsymbol{x_1} - \boldsymbol{x'_{1|0}})(\boldsymbol{x_1} - \boldsymbol{x'_{1|0}})^T] = \Pi_0,$$

for some initial guess Π_0.

Algorithm 4.1-Linear Kalman Filtering:

- Initialization

$$\boldsymbol{x'_{1|0}} = E[\boldsymbol{x_1}]$$
$$P_{1|0} = \Pi_0$$

- For $n = 1, 2, \ldots$,Do

$$S_n = H_n P_{n|n-1} H^T_n + R_n$$
$$K_n = P_{n|n-1} H^T_n S^{-1}_n$$
$$\boldsymbol{x'_{n|n}} = \boldsymbol{x'_{n|n-1}} + K_n(\boldsymbol{y_n} - H_n\boldsymbol{x'_{n|n-1}})$$
$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}$$
$$\boldsymbol{x'_{n+1|n}} = F_{n+1}\boldsymbol{x'_{n|n}}$$
$$P_{n+1|n} = F_{n+1} P_{n|n} F^T_{n+1} + Q_{n+1}$$

- End For

It can be observed that the Kalman filtering is the generalization of the optimal mean-square linear filtering, and it converges to the latter in its steady-state, if the involved processes are stationary.

4.2 Non-Linear Kalman Filter

In the non-linear case of the Kalman filter, it is obvious that there exist various models that one can choose. For example, one can assume that the transition function in the state equation is non-linear, or that the transition function is linear, while the measurement function is not. Also, one could adopt the problem where both these functions are non-linear, etc. In this manuscript, we deal with the problem where the transition function is linear and known, while the measurement function is non-linear and may be known or not. Hence, we end up in the following model:

$$\boldsymbol{x_{n+1}} = F_n\boldsymbol{x_n} + \boldsymbol{w_n} \qquad (24a)$$

$$\boldsymbol{y_n} = h_n(\boldsymbol{x_n}) + \boldsymbol{v_n} \qquad (24b),$$

where F is the known transition matrix, of appropriate dimensions, $\boldsymbol{w_n}$ is the process noise, while $\boldsymbol{v_n}$ is the measurement noise. Finally, $h_n(\cdot)$ is an unknown non-linear function lying in an RKHS. In this chapter we will analyse the algorithm derived from [13], which solves the aforementioned problem. Once one maps the hidden states $\boldsymbol{x_n}$ into an RKHS $\mathcal{H}$, model (24) is transformed as follows:

$$\boldsymbol{x_{n+1}} = F_n\boldsymbol{x_n} + \boldsymbol{w_n} \qquad (25a)$$

$$\boldsymbol{y_n} = \langle h_n, \phi(\boldsymbol{x_n})\rangle_{\mathcal{H}} + \boldsymbol{v_n} \qquad (25b),$$

where F_n is known, while the non-linear function $h_n(\cdot)$ is unknown. The assumptions concerning the noise are those adopted in the previous section. Recall the 4 steps mentioned in the analysis of the linear Kalman filter. Since the state model in problem (25), is the same as the one in the linear Kalman filter, the only equations that change concern the Kalman gain as well as the error covariance updates [13]. The following analysis, is provided in [13] in more detail. Also, recall the KRLS algorithm, provided in **section 2.5**. Since function $h(\cdot)$ is not known, in [13], a hybrid algorithm is adopted, which comprises the KRLS algorithm.
Now, lets provide the new equations of the Kalman filter. In [14] and [13], is given that:

$$K_n = E[\boldsymbol{x_{n+1}}\boldsymbol{e_n^T}]R_{e,n}^{-1}, \qquad (26)$$

where $\boldsymbol{e_n} = \boldsymbol{y_n} - \boldsymbol{y_n'}$ and $R_{e,n} = E[\boldsymbol{e_n}\boldsymbol{e_n^T}]$. Then according to the orthogonality principle:

$$\begin{aligned}R_{e,n} = E[\boldsymbol{e_n}\boldsymbol{e_n^T}] &= E[(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x_{n|n-1}}) + \boldsymbol{v_n})(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x_{n|n-1}}) + \boldsymbol{v_n})^T] \\ &= E[(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x_{n|n-1}}))(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x_{n|n-1}}))^T] + E[\boldsymbol{v_n}\boldsymbol{v_n^T}] = \\ &E[(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x_{n|n-1}}))(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x_{n|n-1}}))^T] + R_n \qquad (27)\end{aligned}$$

Also,

$$E[\boldsymbol{x_{n+1}}\boldsymbol{e_n^T}] = F_nE[\boldsymbol{x_n}\boldsymbol{e_n^T}] + E[\boldsymbol{w_n}\boldsymbol{e_n^T}], \qquad (28)$$

where

$$\begin{aligned}E[\boldsymbol{x_n}\boldsymbol{e_n^T}] &= E[(\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}} + \boldsymbol{x_{n|n-1}})\boldsymbol{e_n^T}] = E[(\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}})\boldsymbol{e_n^T}] \\ &= E[(\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}})(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x_{n|n-1}}))^T] \qquad (29)\end{aligned}$$

since, $(\boldsymbol{x_{n|n-1}} \perp \boldsymbol{e_n^T})$ and $((\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}}) \perp \boldsymbol{v_n})$. In addition, $E[\boldsymbol{w_n}\boldsymbol{e_n^T}] = \boldsymbol{0}$. Furthermore, we know that $P_{n|n-1} = E[(\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}})(\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}})^T]$. By taking the Taylor approximation of $h_n(\boldsymbol{x_n})$, around $\boldsymbol{x_{n|n-1}}$, that is:

$$h_n(\boldsymbol{x_n}) = h_n(\boldsymbol{x_{n|n-1}} + (\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}})) \approx h_n(\boldsymbol{x_{n|n-1}}) + \frac{\partial h_n}{\partial \boldsymbol{x_{n|n-1}}}(\boldsymbol{x_n} - \boldsymbol{x_{n|n-1}}) \qquad (30)$$

and perform some algebra using (26),(27),(28) and (29), one ends up in[13]:

$$K_n \approx P_{n|n-1}H_n'^T[H_n'P_{n|n-1}H_n'^T + R_n]^{-1}, \qquad (31)$$

where $H_n' = \frac{\partial h_n}{\partial \boldsymbol{x_{n|n-1}}}$.
In every time instant, we use the KRLS algorithm (which is embedded in the this algorithm) in order to approximate $h_n(\cdot)$, based on the predictions, i.e. $\boldsymbol{x_n'}$. We are now ready to provide the algorithm:

Algorithm 4.2-EKF-KRLS algorithm [13]:

- Initialization

  Set $\boldsymbol{x'_0} = 0, \Phi_0 = [\kappa(\boldsymbol{x'_0}, \cdot)]$
  $P_0 = E[(\boldsymbol{x_0} - E[\boldsymbol{x_0}])(\boldsymbol{x_0} - E[\boldsymbol{x_0}])^T]$
  $Q(0) = (\lambda + \kappa(\boldsymbol{x'_0}, \boldsymbol{x'_0})^{-1}), \boldsymbol{a}(0) = \boldsymbol{1_d^T}$

- For $n = 1, 2, ...$

  Computation for the Kalman Filter

  $$\begin{aligned}
  &\boldsymbol{x_{n|n-1}} = F_{n-1}\boldsymbol{x_{n-1|n-1}} \\
  &\boldsymbol{y'_n} = \boldsymbol{a}(n-1)^T\Phi_{n-1}^T\phi(\boldsymbol{x_{n|n-1}}) \\
  &P_{n|n-1} = F_{n-1}P_{n-1}F_{n-1}^T + S_{n-1} \\
  &H'_n = \left(\frac{\partial\Phi_{n-1}}{\partial\boldsymbol{x_{n|n-1}}}\boldsymbol{a}(n-1)\right)^T \\
  &K_n = P_{n|n-1}H_n'^T[H'_nP_{n|n-1}H_n'^T + R_n]^{-1} \\
  &\boldsymbol{x_{n|n}} = \boldsymbol{x_{n|n-1}} + K_n(\boldsymbol{y_n} - \boldsymbol{y'_n}) \\
  &P_{n|n} = (I - K_nH'_n)P_{n|n-1}
  \end{aligned}$$

  For the KRLS Filter

  $$\begin{aligned}
  &\Phi_n = [\zeta\Phi_{n-1}, \kappa(\boldsymbol{x_{n|n}}, \cdot)] \\
  &h(n) = \Phi_{n-1}^T\phi(\boldsymbol{x_{n|n}}) \\
  &z(n) = Q(n-1)h(n) \\
  &r(n) = \lambda + \kappa(\boldsymbol{x_{n|n}}, \boldsymbol{x_{n|n}}) - z(n)^Th(n) \\
  &Q(n) = r(n)^{-1}\begin{pmatrix} Q(n-1)r(n) + z(n)z(n)^T & -z(n) \\ -z(n)^T & 1 \end{pmatrix} \\
  &\boldsymbol{e}(n) = \boldsymbol{y_n} - \boldsymbol{a}(n-1)^Th(n) \\
  &\boldsymbol{a}(n) = \begin{pmatrix} \boldsymbol{a}(n-1) - r(n)^{-1}z(n)e(n)^T \\ r(n)^{-1}e(n)^T \end{pmatrix}
  \end{aligned}$$

- End For

4.3 Random Fourier Features Kalman Filter

Recall the problem discussed in section 4.2. We are going to present an algorithm, using the Random Fourier Features technique solving the aforementioned problem. All the assumptions mentioned in section 4.2 hold here too, as well as the assumption that the non linear function $h(\cdot)$ is known and is lying in an RKH space. It suffices to solve this problem, since it can then easily be extended in the case where the function $h(\cdot)$ is not known. The first step deals with the prediction of the prior estimate $\boldsymbol{x'_{n|n-1}}$, using the posterior estimate $\boldsymbol{x'_{n-1|n-1}}$. Then it can easily be seen that:

$$\boldsymbol{x'_{n|n-1}} = F_n \boldsymbol{x'_{n-1|n-1}} \qquad (32)$$

In the sequel, we want to estimate $P_{n|n-1}$.

$$P_{n|n-1} = E[\boldsymbol{e_{n|n-1}} \boldsymbol{e^T_{n|n-1}}]$$

However

$$\boldsymbol{e_{n|n-1}} = \boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}} = F_n \boldsymbol{x_{n-1}} + \boldsymbol{\eta}_n - F_n \boldsymbol{x'_{n-1|n-1}} = F_n(\boldsymbol{e_{n-1|n-1}}) + \boldsymbol{\eta}_n \Rightarrow$$

$$P_{n|n-1} = E[F_n \boldsymbol{e_{n-1|n-1}} \boldsymbol{e^T_{n-1|n-1}} F_n^T] + E[F_n \boldsymbol{e_{n-1|n-1}} \boldsymbol{\eta}_n] + E[\boldsymbol{\eta}_n \boldsymbol{e^T_{n-1|n-1}} F_n^T] + E[\boldsymbol{\eta}_n \boldsymbol{\eta}_n^T] \Rightarrow$$

$$P_{n|n-1} = F_n P_{n-1|n-1} F_n^T + Q_n \qquad (33)$$

Furthermore, we need to estimate $\boldsymbol{x'_{n|n}}$. To do so, we have to adopt the following recursion:

$$\boldsymbol{x'_{n|n}} = \boldsymbol{x'_{n|n-1}} + K_n \boldsymbol{e_n},$$

where $\boldsymbol{e_n} = y_n - W_n Z_\omega(\boldsymbol{x'_{n|n-1}})$. In this case, a Gaussian Kernel is used (which is a shift- invariant kernel) such that:

$$K(\boldsymbol{x}, \boldsymbol{y}) = Z_\omega(\boldsymbol{x})^T Z_\omega(\boldsymbol{y})$$

while

$$Z_\omega(\boldsymbol{x}) = \frac{\sqrt{2}}{\sqrt{N}} \begin{pmatrix} cos(\boldsymbol{\omega}_1^T \boldsymbol{x} + b_1) \\ ... \\ cos(\boldsymbol{\omega}_N^T \boldsymbol{x} + b_N) \end{pmatrix},$$

where $\boldsymbol{\omega}_1, ...\boldsymbol{\omega}_N$ are drawn from $P(\boldsymbol{\omega})$, where $P(\boldsymbol{\omega}) = \frac{1}{2\pi^D} \int_{\Re^D} K(\delta) e^{-i\boldsymbol{\omega}^T \delta} d\delta$, while $b_1, ...b_N$ are uniformly drawn from $[0, 2\pi]$. Also, as mentioned in previous chapters:

$$K(\boldsymbol{x} - \boldsymbol{y}) = E_{\boldsymbol{\omega},b}[Z_\omega(\boldsymbol{x}) Z_\omega(\boldsymbol{y})]$$

Then, in our case, the known function $h(\cdot)$ takes the following form:

$$\vec{h}(\vec{\boldsymbol{x_n}}) = \begin{pmatrix} ... \\ \sum_{m=1}^M K(\vec{\boldsymbol{c_m}}, \vec{\boldsymbol{x_n}}) \\ ... \end{pmatrix} \quad \epsilon \ \Re^{KxD},$$

as $\vec{h} \ \epsilon \ \mathcal{H}^K$ and $\vec{\boldsymbol{c_m}}$ are the given centers describing $\vec{h}$ in the Hilbert space. Hence, we are now able to define a matrix W, such that:

$$W = \begin{pmatrix} \boldsymbol{a_1} Z_\omega(\boldsymbol{c_1}) \\ ... \\ \boldsymbol{a_K} Z_\omega(\boldsymbol{c_K}) \end{pmatrix}$$

and

$$W_n Z_\omega(\boldsymbol{x'_{n|n-1}}) = h(\boldsymbol{x'_{n|n-1}})$$

Now, since the hidden state model is the same, what needs to be considered is an alternative form of the Kalman Gain as well as the error covariance, following the same logic as in section 4.2, that is:

$$K_n = E[\boldsymbol{x_{n+1}} \boldsymbol{e_n^T}] R_{e,n}^{-1} \qquad (34),$$

where $R_{e,n} = E[\boldsymbol{e_n} \boldsymbol{e_n^T}]$. Then:

$$R_{e,n} = E[\boldsymbol{e_n} \boldsymbol{e_n^T}] = E[(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x'_{n|n-1}}) + \boldsymbol{v_n})(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x'_{n|n-1}}) + \boldsymbol{v_n})^T] =$$

$$E[(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x'_{n|n-1}}))(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x'_{n|n-1}}))^T] + R_n \qquad (35)$$

Also,

$$E[\boldsymbol{x_{n+1}}\boldsymbol{e_n^T}] = F_n E[\boldsymbol{x_n}\boldsymbol{e_n}] + E[\boldsymbol{\eta}_n \boldsymbol{e_n^T}],$$

where $E[\boldsymbol{\eta}_n \boldsymbol{e_n^T}] = 0$ and

$$E[\boldsymbol{x_n}\boldsymbol{e_n}] = E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}} + \boldsymbol{x'_{n|n-1}})\boldsymbol{e_n^T}] =$$

$$E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})\boldsymbol{e_n^T}] = E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x'_{n|n-1}}))^T] \quad (36)$$

Now, by using the Taylor approximation around $\boldsymbol{x'_{n|n-1}}$, we approximate $h(\cdot)$ as follows:

$$h_n(\boldsymbol{x_n}) = h_n(\boldsymbol{x'_{n|n-1}} + (\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}}))$$

$$\approx h_n(\boldsymbol{x'_{n|n-1}}) + \frac{\partial h_n(\boldsymbol{x'_{n|n-1}})}{\partial \boldsymbol{x'_{n|n-1}}}(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})$$

At this stage, we define $H'_n = \frac{\partial h_n(\boldsymbol{x'_{n|n-1}})}{\partial \boldsymbol{x'_{n|n-1}}}$. By combining (34),(35) and (36) we end up in:

$$K_n = E[\boldsymbol{x_n}(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x'_{n|n-1}})^T] R_{e,n}^{-1} \Rightarrow$$

$$K_n = E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})(h_n(\boldsymbol{x_n}) - h_n(\boldsymbol{x'_{n|n-1}})^T] R_{e,n}^{-1} \Rightarrow$$

$$K_n = E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})(\frac{\partial h_n(\boldsymbol{x'_{n|n-1}})}{\partial \boldsymbol{x'_{n|n-1}}}(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}}))^T] *$$

$$*(E[(\frac{\partial h_n(\boldsymbol{x'_{n|n-1}})}{\partial \boldsymbol{x'_{n|n-1}}}(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}}))(\frac{\partial h_n(\boldsymbol{x'_{n|n-1}})}{\partial \boldsymbol{x'_{n|n-1}}}(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}}))^T] + R_n)^{-1} \Rightarrow$$

$$K_n = P_{n|n-1} H_n'^T [H'_n P_{n|n-1} H_n'^T + R_n]^{-1} \quad (37)$$

We finally, want to estimate $P_{n|n}$. That is,

$$P_{n|n} = E[\boldsymbol{e_{n|n}}\boldsymbol{e_{n|n}^T}] = E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})^T] -$$

$$-E[(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})(K_n\boldsymbol{e_n})^T] - E[(K_n\boldsymbol{e_n})(\boldsymbol{x_n} - \boldsymbol{x'_{n|n-1}})^T] + E[K_n\boldsymbol{e_n}(K_n\boldsymbol{e_n})^T] \Rightarrow$$

$$P_{n|n} = P_{n|n-1} - K_n(P_{n|n-1}H_n'^T)^T \Rightarrow$$

$$P_{n|n} = P_{n|n-1} - K_n H'_n P_{n|n-1}^T \quad (38)$$

Now that all the equations are exported, the algorithm is going to be presented.
Algorithm 4.3-The RFFKF algorithm:

- Initialization

  $\boldsymbol{x_{1|0}} = E[\boldsymbol{x_1}]$
  $P_{1|0} = \Pi_0$

- For $n = 1, 2, ...$ Do

  $K_n = P_{n|n-1} H_n'^T [H'_n P_{n|n-1} H_n'^T + R_n]^{-1}$
  $\boldsymbol{x'_{n|n}} = \boldsymbol{x'_{n|n-1}} + K_n(y_n - W Z_\omega(\boldsymbol{x'_{n|n-1}}))$
  $P_{n|n} = P_{n|n-1} - K_n H'_n P_{n|n-1}$
  $\boldsymbol{x'_{n+1|n}} = F_{n+1}\boldsymbol{x'_{n|n}}$
  $P_{n+1|n} = F_{n+1} P_{n|n} F_{n+1}^T + Q_{n+1}$

- End For

At this point, two examples of this algorithm are going to be presented.
• **Example 4.1**: In this example, the samples are generated by the following scheme:

$$y_n = \boldsymbol{x_n^T}\boldsymbol{w_0} + 0.1 * (\boldsymbol{x_n^T}\boldsymbol{w_1})^2 + \boldsymbol{\eta}_n,$$

where, $\boldsymbol{\eta}_n$ are zero-mean i.i.d. Gaussian noise samples with $\sigma_\eta = 0.05$ and the coefficient vectors $\boldsymbol{w_0}, \boldsymbol{w_1} \in \Re^5$, are i.i.d. samples drawn for $\mathcal{N}(0, 1)$. The σ in the Gaussian kernel (used in the RFFRLS which provides the known function $h(\cdot)$) was chosen to be equal to 5, while the number of Fourier coefficients was chosen to be 25.

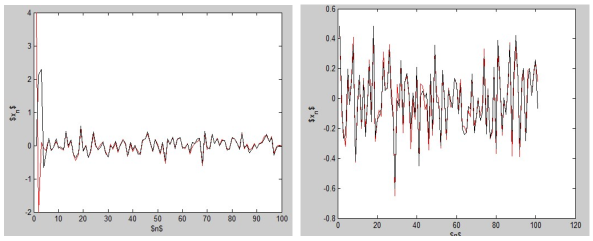

Figure 4.1: Simulation of the RFFKF algorithm, over 500 samples generated by the equation given in Example 4.1. With the red colour, the real values of x are denoted, while the predicted ones, are denoted with the black colour.

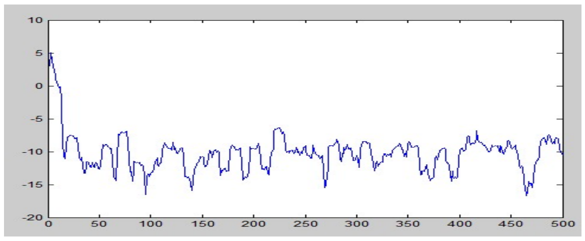

Figure 4.2: Simulation of the RFFKF algorithm, over 500 samples generated by the equation given in Example 4.1. The figure, represents the error value at each time instant in Decibel.

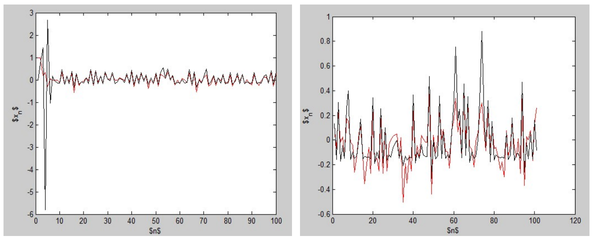

Figure 4.3: Simulation of the RFFKF algorithm, over 500 samples generated by the equation given in Example 4.2. With the red colour, the real values of x are denoted, while the predicted ones, are denoted with the black colour.

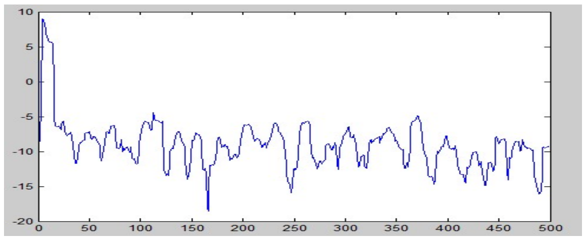

Figure 4.4: Simulation of the RFFKF algorithm, over 500 samples generated by the equation given in Example 4.2. The figure, represents the error value at each time instant in Decibel.

Also, concerning the RFFRLS, the regularization parameter is $\lambda = 0.00001$ while the user defined parameter is $\beta = 0.9995$. Now given the values of y, the algorithm has to predict the values of $\boldsymbol{x}$. In **Figure 4.1** the real values of $\boldsymbol{x}$ are plotted (with red), together with the predicted values provided by the algorithm. Also, in **Figure 4.2**, the error in dB, at each time step is presented.

• **Example 4.2**: In this example, the samples are generated by the following chaotic series model:

$$d_n = u_n + 0.5u_n - 0.2d_{n-1} + 0.35d_{n-1}$$

$$\phi(d_n) = \begin{cases} \frac{d_n}{3(0.1+0.9d_n^2)^{1/2}} & d_n{>}0 \\ \frac{-d_n^2(1-exp(0.7d_n))}{3} & d_n{<}0 \end{cases}$$

$$y_n = \phi(d_n) + \eta_n,$$

where η_n is zero mean i.i.d. Gaussian noise with $\sigma_n = 0.001$, u_n is also zero-mean i.i.d. Gaussian with $\sigma_u^2 = 0.0156$ and $u_n = 0.5u_n + \eta'_n$, where η'_n is also i.i.d Gaussian noise with $\sigma^2 = 0.0156$. The σ in the Gaussian kernel (used in the RFFRLS which provides the known function $h(\cdot)$) was chosen to be equal to 5, while the number of Fourier coefficients was chosen to be 25. Also, concerning the RFFRLS, the regularization parameter is $\lambda = 0.00001$ while the user defined parameter is $\beta = 0.9995$. Now given the values of y, the algorithm has to predict the values of $\boldsymbol{x}$. In **Figure 4.3** the real values of $\boldsymbol{x}$ are plotted (with red), together with the predicted values provided by the algorithm. Also, in **Figure 4.4**, the error in dB, at each time step is presented.

5. CONCLUSIONS

At this point, a summary of the basic results of this manuscript will be presented. Firstly, some adaptive filtering algorithms were explained, implemented and tested with each other. Concerning the second chapter combined with the third, one can easily observe from the experiments, that the Random Fourier Features technique applied in the case of the KRLS as well as the KLMS algorithm, not only provides better results in terms of the amplitude of the error, but it also gives elegant and much simpler to understand and implement algorithms. The latter was confirmed by various experiments. Also, from the theoretical perspective, it was confirmed that KLMS achieves similar performance compared to the RFFLMS. Furthermore, a simple method for applying the Random Fourier Features technique was provided, and was used later on, in the framework of the Kalman Filter, which comprises a quite difficult problem in the field of Machine Learning. The latter, confirms the ease in the use of this technique, and also shows its potential to be used in a wide variety of problems effectively.

Recall the theory provided in Chapter 1. A problem occurs when this theory is applied in real time problems and is mentioned multiple times in this manuscript. The problem is that the number of coefficients of the solution grows linearly with time, resulting in a unmanageable solution. With the help of the Random Fourier Features technique, we did not only solve this problem, but also provided algorithms with much better performance, as well as easier implementation. As mentioned in the previous paragraph, many experiments were held that confirm this result, in various problems.

However, the Random Fourier Features technique does not fit everywhere and sometimes in order to apply it effectively in some problems, one may need to face many difficulties. For example, in the framework of the Kalman Filters, in the case where both the process and the transition functions are non linear, application of the Random Fourier Features was ending up in some difficult open problems, which were not solved in this manuscript. Also, recall the algorithm provided in section 4.3. This algorithm, uses a Taylor approximation in order to define the process function H. The construction of an algorithm using a purely linear process function and the Random Fourier Features, also faces some huge obstacles.

Nevertheless, it can readily be observed that the Random Fourier Features technique is a powerful tool that can be used in a wide variety of problems, giving efficient algorithms capable of solving effectively many real time, distributive, and other problems.

REFERENCES

[1] : Sergios Theodoridis :Machine Learning-A Bayesian and Optimization Perspective

[2] : Clayton Scott : Kernel Methods and the Representer Theorem

[3] : Stephane Canu : SVM and kernel machines - linear and non-linear classification

[4] : Le Song,Kenji Fukumizu,Arthur Gretton : Kernel Embeddings of Conditional Distributions

[5] : Leila Wehbe : Kernel Properties - Convexity

[6] : Wei Gao, Jie Chen, Cedric Richard, Jianguo Huang : Online Dictionary Learning for Kernel LMS

[7] :Yaakov Engel ,Shie Mannor ,Ron Meir: The Kernel Recursive Least Squares Algorithm

[8] : Koby Crammer,Alex Kulesza,Mark Dredze: NEW $\mathcal{H}^{\infty}$ BOUNDS FOR THE RECURSIVE LEAST SQUARES ALGORITHM EXPLOITING INPUT STRUCTURE

[9] : Ali Rahimi,Ben Recht : Random Features for Large Scale Kernel Machines

[10] : Pantelis Bouboulis,Spyridon Pougkakiotis ,S.Theodoridis : EFFICIENT KLSM AND KRLS ALGORITHMS-A RANDOM FOURIER FEATURE PERSPECTIVE

[11] : Yaakov Engel ,Shie Mannor ,Ron Meir: The Kernel Recursive Least Squares Algorithm

[12] : Koby Crammer,Alex Kulesza,Mark Dredze: NEW $\mathcal{H}^{\infty}$ BOUNDS FOR THE RECURSIVE LEAST SQUARES ALGORITHM EXPLOITING INPUT STRUCTURE

[13] : Pingping Zhu, Badong Chen, and Jose C. Principe: Extended Kalman Filter Using a Kernel Recursive Least Squares Observer

[14] : A. Sayed, Fundamentals Of Adaptive Filtering, New York: Wiley, 2003.