Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Màster

Màster Universitari en Enginyeria Informàtica i la seva gestió

Modelling non-linear dynamical systems by orthogonal forward regression

Alumne
Albert Velasco Droguet

Professor Ponent
Xavier Vilasís Cardona

ACTA DE L'EXAMEN DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Albert Velasco Droguet

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

Modelling non-linear dynamical systems by orthogonal forward regression

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL

VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Modeling non-linear dynamical systems by Orthogonal Forward Regression

Albert Velasco Droguet

Enginyeria i Arquitectura La Salle - Universitat Ramon Llull
Quatre Camins 2, 08022 Barcelona
`tl14103@salle.URL.edu`

Abstract

In this work we are going to model Non-Linear Dynamical Systems by applying Orthogonal Forward Regression algorithm.

We will study Non-Linear dynamical systems, because are systems with lots of applications such as weather prediction, synchronization, numerical simulations etc. These systems are characterized to be able to produce chaotic behavior.

The aim of this work is to develop a program able to model chaotic time-series. To do it, we will use Chua's Circuit as one of the most known systems able to produce chaos.

Summary

The aim of this work is to create models of Non-Linear Dynamical Systems such as Lorenz Attractor, Chua's Circuit etc.

We are interested in that kind of systems because they have multiple applications in several fields such as in the study of the atmosphere, control, communications etc. These systems despite it seems that present a random and nondeterministic behavior (known as chaos) they are completely deterministic systems. This behavior is due to the main property of these systems, that consist of sensitive dependence on the initial conditions. This property means that a small difference in the initial conditions of the system may affect widely the output of the system.

The aim of modeling this systems, consist of minimizing the amount of data to be treat, but without losing quality. Once a good modeling of the system is built, it's easier to analyze the functions that compound the model. One of the applications that we are going to study, consists of modeling an ECG applying OFR (Orthogonal Forward Regression). That application is able to detect ventricular beats with a great rate of success.

A software tool will be developed, with the aim of testing all the concepts studied in this report, such as Orthogonal Forward Regression, Non-Linear Dynamical Systems, Embedding etc. To test the modeling, we will use Chua's Circuit as a well-known model able to produce chaos. To model Chua's circuit, a dataset of Radial Basis Functions has been created, and embedding has been applied to reconstruct the attractor. To optimize the model that we are going to obtain, several optimization techniques are going to be applied such as Least Squares with a temporal and vectorial approaches, Gradient Descent Method, Levenberg-Marquartd etc.

Finally, the obtained results will be shown and discussed, and we will present some further lines of work such as testing the software with other chaotic systems, try other optimizations methods such as Newton's etc.

Contents

List of Figures

List of Tables

Chapter 1

Goals and motivation of the thesis

This chapter provides a full description of the different goals of our project, such as studying Non-Linear Dynamical Systems, Orthogonal Forward Regression, Embedding etc. With the final goal of implementing a software able to model Non-Linear Dynamical Systems applying Orthogonal Forward Regression.

1.1 Goals of the thesis

The aim of this project, consist of studying all the concepts related with Statistical Learning techniques, Non-Linear Dynamical Systems, and modeling. The final goal of the project, consist of applying all these concepts into a software that should be able to model that kind of systems.

In the following lines we are going to detail each one of the previously commented goals of our project:

- **Statistical Learning Theory:** One of the goals of our thesis is to describe what is Statistical Learning Theory. This project is based on Statistical Learning, and despite we just will apply some of the several techniques that compound it, we need to describe in detail, all the fields this project is related witg.

  In this chapter we are going to describe what is it classification, and why is so important nowadays. Some techniques related with it, also will be studied, like Support Vector Machines, Neural Networks, RBF Networks etc.

- **Non-Linear Dynamical Systems:** The main goal of our thesis is to study Non-Linear Dynamical Systems. These systems are complex, and have been widely studied in the past, but still nowadays are relevant, with several applications in various fields, such as mathematics, synchronization phenomena, weather prediction etc.

  We will study Non-Linear Dynamical Systems from the basis, to introduce finally, two of the most known systems able to produce, what we know as chaos, that are Chua's Circuit and Lorenz Oscillator.

- **Embedding** Embedding also will be used to analyze the chaotic data. Applying embedding, we will be able to reconstruct the phase space of our function, and thanks to it we will be able to model our chaotic data.

- **Orthogonal Forward Regression:** Orthogonal Forward Regression, is one of the techniques that able us to do Regression of a certain function. OFR is a well-known technique, that let us as obtain a good regression with a small algorithmic and computational effort.

  OFR uses a library of functions to do the regression. For that purpose we will create a library of RBF's (Radial Basis Functions) because are a set of functions easy to generate (few computational effort to build), and highly adaptable to our goal function.

- **Implementation issues:** The final goal of our thesis will be to implement a software tool able to model Non-Linear Dynamical Systems, by applying the techniques previously cited, such as Orthogonal Forward Regression, Embedding etc. To do it, we will do all the tests with the most known system able to produce Chaos, the Chua's Circuit.

  Once we will obtain the model of our goal function, and depending on the degree of error, we will apply some optimization methods, to improve the model obtained, trying to reduce the error below the 10%.

1.2 Motivation

The motivation before starting our project is quite clear, analyzing in detail the above goals. Non-Linear Dynamical Systems are highly important for their multiple applications in our world such as weather prediction, communications etc. But, the output of these systems is complex, and there's the necessity of modeling this systems, to analyze in detail their behavior.

The aim of this work is to develop a software, able to model these systems, with the minimum error possible and the minimum computational effort.

Chapter 2

Introduction to Statistical Learning

This chapter provides a brief introduction to Statistical Learning. We are going to describe what is Statistical Learning, which are the techniques that form it, and which are the applications of that algorithms.

2.1 Statistical Learning Theory

The aim of statistical learning theory is to provide a framework for studying the problem of inference. But, what is inference? Inference is the process of giving a conclusion by applying logical or statistical to certain observations or hypotheses. In other words, inference imply to gain knowledge, make predictions, construct models, etc from a set of data.

The theory of inference should be able to give a formal definition of words such as learning, generalization, overfitting etc, with the aim of improving the practical aspects such as designing better learning algorithms.

As proposed by Bousquet et al. 2004 the steps that form any process of inductive inference are:

1. Observation of a certain phenomenon
2. Construction of a mode for that phenomenon
3. Make predictions using that model

2.2 Classification

Classification is a supervised machine learning technique, consisting of identifying the class to which a new instance belongs to.

To do this, we need to construct previously a model, by analyzing a training set with labeled elements.
In other words, for classifying elements, first we need to construct the pattern (a model) that will let our classifier to classify properly.

In the following figure, we can observe the classification flow of any kind of classifier:

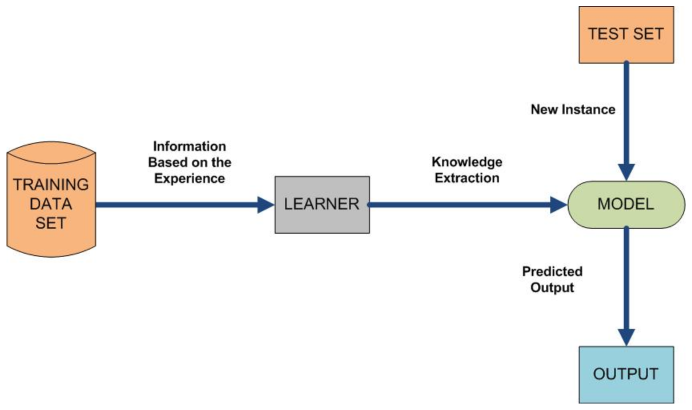

Figure 2.1: *Classification scheme*

The model that we need to construct can be implemented in different ways such as:

- Rules
- Mathematical Formulae
- Decision Trees

But, how we can classify for example different kind of dogs? To classify any object, we need to identify which are their characteristics.
Once the characteristics will be well-defined, we just need to define a good training set.
The quality of our model and therefore the performance of our classification will depend basically on the quality of that training set.

As defined in Mitchell 1997, there are different types of Classification Models, the most important ones are:

- **Interval based Classifier:** The classification is based on a quadratic separation of the model.
- **Instance based Classifier:** The classification is based on instances (multi-linear classifier). KNN (K-Nearest Neighbors) algorithm is based on an Instance based Classifier.
- **Linear Classifier:** The classification is based on a linear separation of the model. The following algorithms are based on a linear classifier:
  - Perceptron
  - Naive Bayes
  - SVM (Support Vector Machines)
  - etc

2.3 Function Approximation

A function approximation problem consists of finding another function (or combinations of functions) that matches a target function (the function that we want to approximate).

We should distinguish two different types of function approximation problems, for well-known functions and for unknown functions.

In the first case (well-known functions), we know the target function and we have to look for the functions from a library (polynomials, rationals) that best fits our target function. That functions have a bunch of desirable characteristics such as inexpensive calculation, continuity etc.

To sum up, the aim of this kind of problems consists of finding a function that approximates our target function but, with a set of characteristics that are better for our type of problem.

In the second case (unknown functions), we don't have an explicit function, just a set of points. To look for a function that best fits this points we can apply different techniques such as regression, interpolation, extrapolation etc.

Regression, is a function approximation method that aims to look for the best approximation but avoiding the noise. This method does an average of the noise, and look for the function that best fit the function with the averaged noise.

In the following Image we can see an example of regression:

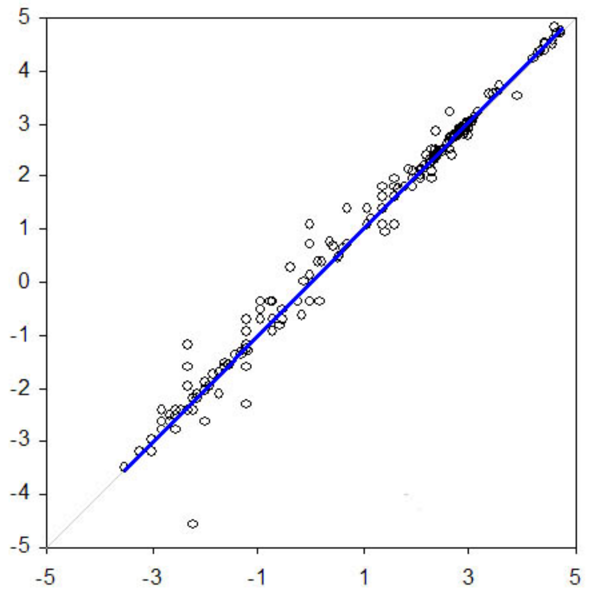

Figure 2.2: *Example of Regression*

We understand as Interpolation, to a specific case of curve fitting, in which the function must fit exactly the data points of our specific problem. In that case, we create new points, between the previous well-known points.

In the following Image we can see an example of interpolation:

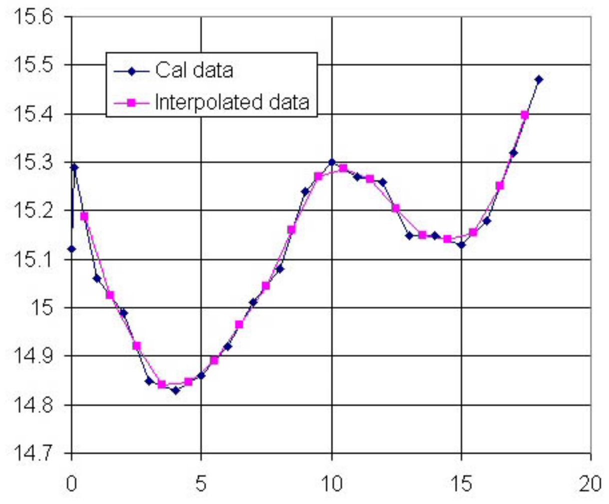

Figure 2.3: *Example of Interpolation*

Extrapolation, is a method quite similar to Interpolation, but with the difference that the new points that are created are outside the range of known points. In other words, is the process of creating new data points extending the known one's.

In the following Image we can see an example of extrapolation, the red line is the line that interpolates and extrapolates the dataset:

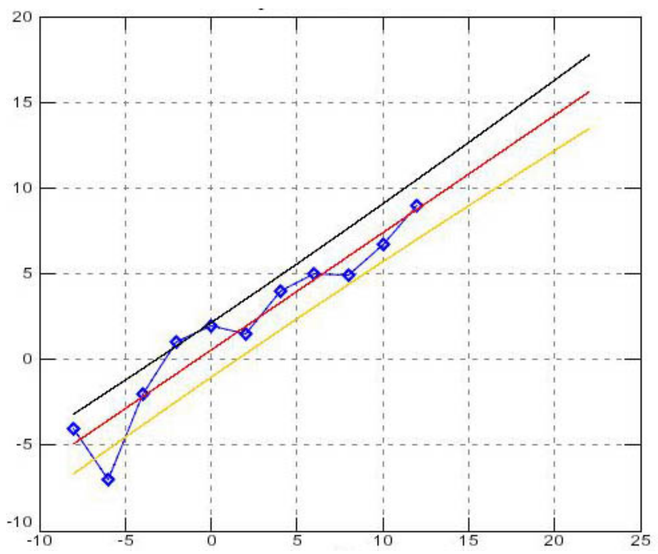

Figure 2.4: *Example of Extrapolation*

2.4 Neural Networks

Neural Networks (NN) also known as Artificial Neural Networks (ANN), are mathematical models inspired by the structure of biological Neural Networks (brain neurons).

Why NN are inspired with our brain? We know that our brain is much more powerful than a computer in certain tasks such as image processing. The aim of NN then is to simulate how our brain processes an image. But, how we can simulate our brain with a computer? We can simulate our brain by defining a set of Artificial Neurons, interconnected between each other that transmit information between them.

As stated by Widrow and Lehr 1990 ANN's are adaptive systems, their structure will evolve based on the information flow through all the network, that can be transmitted from outside the network (external information) or between neurons (internal information).

The Perceptron is the simplest type of an ANN. It's represented by the following image (borrowed from: http://www.clagir.com/computronica/perceptron/):

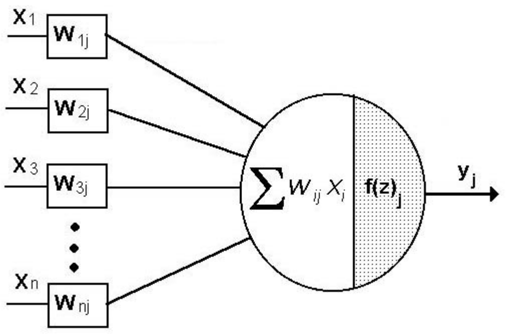

Figure 2.5: *Representation of a perceptron*

Mathematically can be represented by the following expression:

$$y_j = \sum_{i=1}^{n} w_{ij} \cdot x_i \cdot f(z)_j. \tag{2.1}$$

Where:

- $x_1, x_2, ..., x_n$ are the inputs.
- $w_1, w_2, ..., w_n$ are the weights.
- f(z) is the activation function.

The ADALINE (Adaptive Linear Element) can be considered the 2nd generation of ANN's. ADALINE is similar to the perceptron, but presents an important difference. In the learning phase of the ADALINE, the weights are adjusted according to the weighted sum of the inputs, while in the perceptron the weights are tuned using the output of the activation function.

In the following image (borrowed from Orriols-Puig 2009) we can see the scheme of ADALINE:

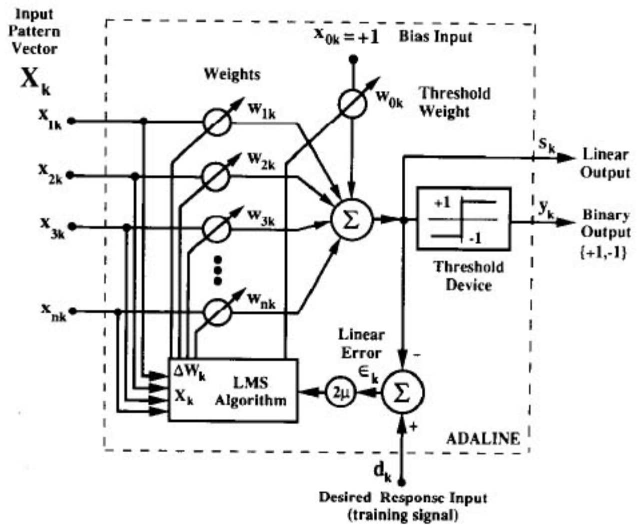

Figure 2.6: *Model of ADALINE*

As we can see in the image above, the weights are adjusted using the Least-Mean Square algorithm. We can describe this process mathematically using the next formula:

$$w = w + \eta(do_a o) \cdot x. \tag{2.2}$$

where:

- w is the weight.
- do is the desired output.
- ao is the actual output.
- η is the learning rate

ADALINE is a good linear classifier, but has problems with nonlinear problems. These problems can be solved by using polynomial discriminant functions or by using other methods such the one's that we are going to detail in the following lines.

A 3rd generation of ANN's, appeared with the aim of solving the problems that ADALINE had to tackle with nonseparable data. MADALINE (Multiple ADALINE's) is a two layer NN, based on multiple ADALINEs connected in parallel as input layer. Its output layer is formed by a single ADALINE.

Until that moment we've seen different kinds of ANNs, able to classify different kinds of data, but, all these ANNs present the problem of getting the proper weights to solve our problems.

There are several algorithms to calculate the weights in a precise way:

- Backpropagation
- α-LMS
- α-Perceptron
- May Algorithm
- Perceptron rule

In this work, we are going to focus just in the backpropagation algorithm because is considered the most relevant one.

Backpropagation is an algorithm able to find suitable weights in a multiple layer network by adjusting the error, propagating it backwards.

Backpropagation algorithm can be seen as a 4-step algorithm.
The Following images has been borrowed from:
http://home.agh.edu.pl/ vlsi/AI/backp_t_en/backprop.html

1. Insert data into the network (through the inputs $x_1, x_2, ...x_k$). That data will go through all the network until reaching the output Y.

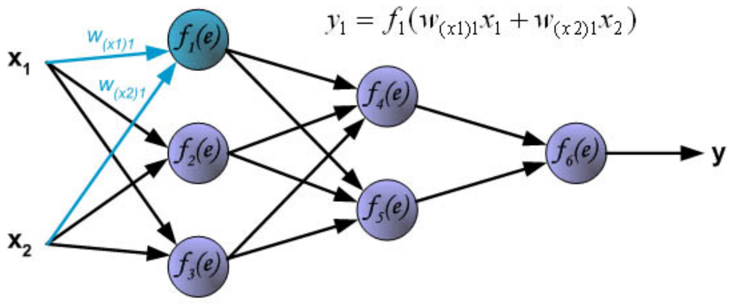

Figure 2.7: *Forward propagation of the data through all the network*

2. Computation of the square error (error between the expected output, and the actual output).

$$\epsilon_k = \sqrt{\sum_{i=1}^{n}(d_{ik} - y_{ik})^2} \tag{2.3}$$

3. Propagation of the error to the previous layer (backpropagation)

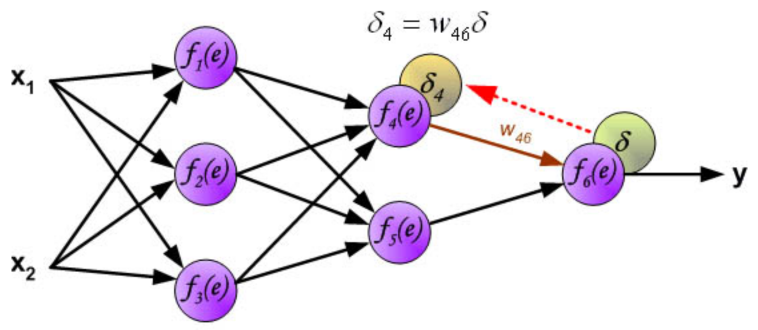

Figure 2.8: *Backpropagation of the error*

4. When the error has been completely propagated through all the network, the weights of the input need to be recalculated. After recalculating the weights we jump to step 1.

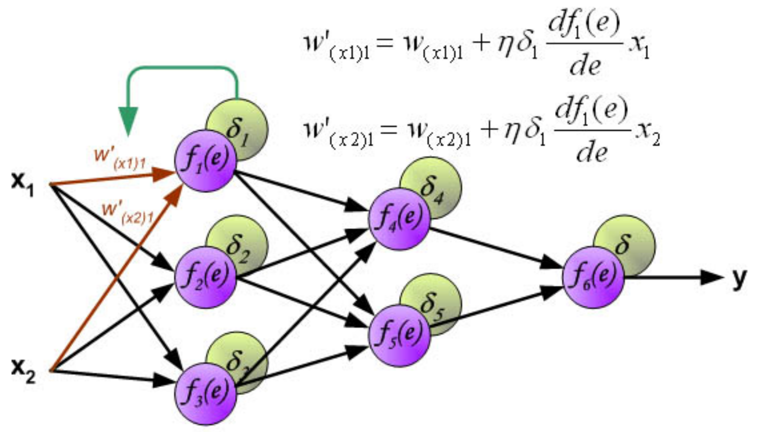

Figure 2.9: *Recalculation of the weights*

The algorithm ends when the square error is smaller than a certain threshold.

2.5 Radial Basis Function Network

Radial Basis Function (RBF) Network emerged as a variant of ANN's. As mentioned in Bors 2001, RBF Networks has excellent approximation capabilities, being able to model complex mappings, which traditional ANNs can only model by using multiple intermediary layers. RBF networks have been successfully applied to several applications such as interpolation, chaotic time-series modeling (the objective of our work), speech recognition etc.

In a RBF Network, RBF's are embedded in a two layer NN, where each hidden layer implements a radial activated function.

Typically RBF Networks has 3 layers:

1. **Input Layer:** The input of any RBF Network contains Nonlinear Data.
2. **Hidden Layer:** Contains a RBF activation function.
3. **Output Layer:** The output of any RBF Network contains Linear Data.

In the following image we can see the scheme of a RBF Network:

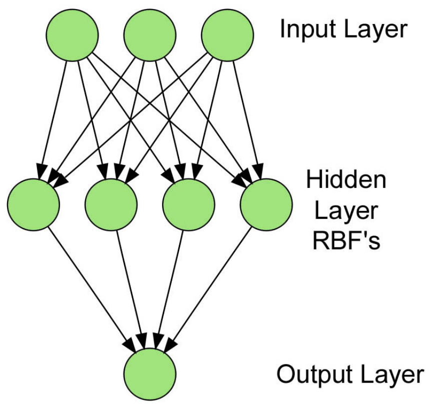

Figure 2.10: *Radial Basis Function Network*

Mathematically a RBF Network can be represented by the following expression:

$$\varphi(x) = \sum_{i=1}^{n} w_i \cdot RBF(x, c_i) = \sum_{i=1}^{n} w_i \cdot \rho(\|x - c_i\|). \tag{2.4}$$

where:

- N: Number of Neurons in the hidden layer
- c_i: Center for neuron i
- x: Input data
- $\rho(\|x - c_i\|)$: RBF defined by the following formula:

$$\rho(\|x - c_i\|) = \exp[-\beta \|x - c_i\|^2]. \tag{2.5}$$

Up to now, we defined what is it a generic RBF network, but we need to explain how we can adapt the different elements of a RBF network to solve a certain kind of problem. The way the network is used for a certain kind of problem, for example for modeling, is different when use it for approximating time-series. For that reason, we have to consider the following points:

1. **Specification of the hidden layer activation function:** Several activation functions have been tested. Each application use a certain activation function, by example, the following activations functions have been largely tested in the next applications:

- **Time-series Modeling:** In that application, on of the most used functions is thin-plate spline (Chen et al. 1991), that it's defined by the following expression:

$$\phi(v) = v^2 \log(v). \tag{2.6}$$

- **Pattern classification:** In that field, gaussian functions are the most common ones. The specific gaussian function used in pattern classification is expressed by the following expression:

$$\phi_j(X) = \exp[-(X - \mu_J)^T \sum_{j=1}^{L} (X - \mu_j)]. \tag{2.7}$$

  where:

  - L: is the number of hidden neurons.
  - X: is the input feature vector
  - μ_j and $\sum_{j=1}^{L}$: are the mean and the covariance matrix of the Gaussian function j.

- **Other fields:** In that fields, a mixture of gaussian functions have been largely used.

2. **Definition of a training algorithm:** The most used training algorithms are supervised. So, we provide a training set to adjust the parameters of our network.

   The following 3 parameters can be optimized:

   (a) **The centers of the RBF's (c_i)**
   (b) **The output weights (w_i)**
   (c) **The amplitude of the RBF's (σ)**

   To look for the appropriate parameters that minimize the error, we apply the following cost function:

$$\min \sum_{i=1}^{Q} (Y_k(X_i) - F_k(X_i))^T - (Y_k(X_i) - F_k(X_i)). \tag{2.8}$$

   where:

   - Q: Total number of vector in the training set.
   - $Y_k(X_i)$: RBF output vector.
   - X_i: Data Sample.
   - $F_k(X_i)$: Output vector associated with the data Sample X_i.

   There are several training algorithms applied to RBF networks.

   The first approach consisted of assigning a certain basis function to a specific data set, but this solution was expensive in terms of computational time, memory and number of parameters.

The second approach consisted of algorithms that look for the exactly fit of the training data, this algorithms didn't generalize correctly.

Other approaches consisted of choosing randomly the hidden layer weights, and calculate the output weights solving a system of equations. That approach, was also computationally expensive due to the required matrix inversion.

Nowadays, there are several works which applied different training algorithms witch successful results, such as Interpolation, OLS using Gram-Schmidt, Backpropagation etc.

2.6 Support Vector Machines

As defined by Cortes and Vapnik 1995 Support Vector Machines (SVMs) can be considered as a smart type of perceptron, being able to classify linearly separable and nonlinearly separable data with a significance less effort than with perceptron or NNs.

SVM is considered a group of supervised learning methods able to solve problems of classification and regression. We considered SVM as a group of methods because since the creation of SVMs, several improvements appeared such as Kernel Tricks, Gaussian processes etc.

The aim of SVMs is to classify data, but how they classify the data? A SVM construct 1 or more hyperplanes in a high dimensional space, with the objective of separating it, maximizing the margin between the elements of the different classes.

Formalizing the SVMs, given a dataset D, expressed by the following formula:

$$D = \{(x_i, c_i)/x_i \varepsilon \mathbb{R}, c_i \varepsilon (-1, 1)\}. \tag{2.9}$$

where:

- D is the Dataset
- x_i is a vector
- c_i is the class to which the vector i pertain

Given a Hyperplane expressed by:

$$w \cdot x - b = 0. \tag{2.10}$$

Where, w is a normal vector (perpendicular to the Hyperplane).

Supposing a linearly separable dataset, we can formulate the following expression, that all the training of the class $y_i = -1$ must satisfy:

$$w \cdot x_i - b \leq -1. \tag{2.11}$$

And the class $y_i = 1$:

$$w \cdot x_i - b \geq +1. \tag{2.12}$$

We can combine both expressions, resulting:

$$y_i(x_i \cdot w + b) - 1 \geq 0 \, \forall i. \tag{2.13}$$

We can express all these formula's graphically, as in the following image, borrowed from: http://en.wikipedia.org/wiki/Support_vector_machine

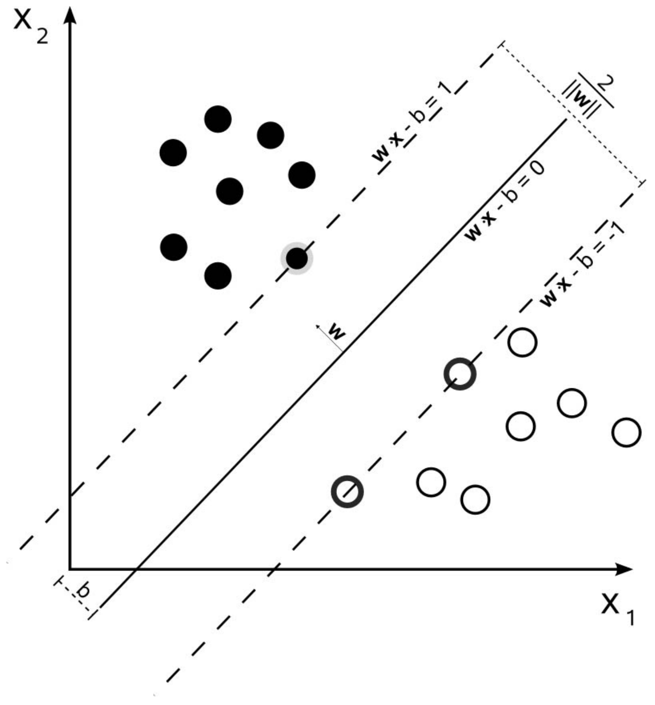

Figure 2.11: *Support Vectors*

As we can see in the image above.
The equations of the hyperplanes (the support vectors) that let separate our data are:

Hyperplane 1:

$$w \cdot x + b = 1. \tag{2.14}$$

Hyperplane 2:

$$xi \cdot w + b = -1. \tag{2.15}$$

The margin between both Hyperplanes is described by the following expression:

$$\frac{2}{\|w\|}. \tag{2.16}$$

To maximize the margin between the hyperplanes we need to find the hyperplanes that minimize $\|w^2\|$.

But, taking in consideration that any point fall inside that margin. To prevent this we can use the previous expression 2.13.

This an optimization problem that can be solved by applying quadratic programming optimizers.

2.6.1 Primal form optimization

We can apply for example, Lagrange Formulation, that is a well-known method that will let us obtain unambiguous and consistent equations.

Applying this optimization method to the previously stated problem, we obtain the following expression:

$$LP \equiv \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{k} \alpha_i. \tag{2.17}$$

where α_i is the Lagrangian multiplier

To look for the optimal w, we need to minimize LP.

2.6.2 Dual form optimization

Transforming the problem to a dual form, we can see that the problem becomes easier to solve maximizing the following expression with regard to α_i:

$$Ldual \equiv \sum_{i} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot x_i \cdot x_j. \tag{2.18}$$

We need to maximize the above expression considering the following constraints:

- $\sum_i \alpha_i \cdot y_i = 0$
- $\alpha_i \geq 0$

2.6.3 Non-Separable datasets

Up to now, we supposed that our datasets are separable with a pair of hyperplanes. But, we can find datasets that we can't separate. In that cases, we need to define another procedure that will let us deal with that cases because the procedure that defined previously can't be applied in that cases.

The idea consists of relaxing the constraints permitting some errors on the classification process. That errors will be considered as noise, but it's a noise that we can't avoid.

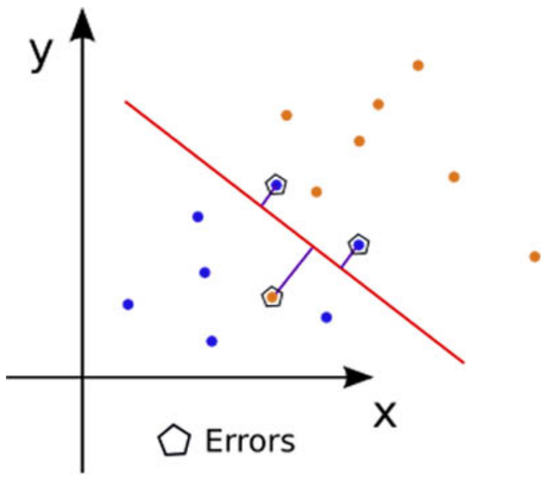

Figure 2.12: *Dataset with Non Linearly Separable Data*

In the image above (borrowed from:
http://www.math.cornell.edu/ numb3rs/kostyuk/num219.htm), we can observe that are some dots that can't be classified properly because are between the dots of the other class, so we can't separate them with a line.

Expressing formally the previously idea, we need to re-formulate the previous expressions 2.11 2.12:

Class 1:

$$w \cdot x_i - b \leq -1 - \xi_i. \tag{2.19}$$

Class -1:

$$w \cdot x_i - b \geq +1 - \xi_i. \tag{2.20}$$

where: ξ_i is the error and $\xi_i \geq 0 \, \forall i$

After changing the above expressions, also the Lagrangian expression change by:

$$LP \equiv \frac{\|w\|^2}{2} + C(\sum i\xi_i)^k. \tag{2.21}$$

Note that we need to maximize w, but minimize the error. C is a constant chosen by the user.

In the dual form, we keep the previous expression 2.18, but the constraints change:
$0 \leq \alpha_i \leq C$.

2.6.4 Non-Linear datasets

SVMs also can deal with Non-Linear datasets, but not like we already explained.

To do it, we apply what is called the kernel trick, that consists of mapping the data in a higher-dimensional space.

But, why mapping data in a higher-dimensional space? We can classify data that is not linear mapping it in a higher-dimensional. This is explained by Mercer theorem that says that: "any symmetric, positive-definite kernel function K(x, y) can be expressed as a dot product in a high-dimensional space".

We can define a Kernel function as the dot product of 2 functions, the formal definition of it is:

$$K(x, y) = \phi(x) \cdot \phi(y). \tag{2.22}$$

By applying kernel functions, the maximization by Lagrangian formulation still can be applied, by modifying the above expression 2.17:

$$LP \equiv \sum_{i=1}^{Ns} \alpha_i y_i \cdot x_i \cdot K(s_i, x) + b. \tag{2.23}$$

where K(si,x) should be substituted by an adequate kernel function to solve our specific problem.

Chapter 3

Orthogonal Forward Regression

This chapter provides a brief introduction to OFR, starting from the basics of the method and ending describing real applications of it. As we are going to see, this method is widely use nowadays and has several advantages in some applications, compared with other methods such as, Neural Networks and Support Vector Machines.

3.1 What is Orthogonal Forward Regression?

Orthogonal forward regression is a method commonly used in different kind of applications such as compression, pattern recognition, feature extraction etc. As was suggested in Dubois et al. 2007 we can describe this method as a 3-step algorithm.

1. Generation of a library of functions, we are going to call it D. This library will be of finite size, and will be constructed using Radial Basis Functions, wavelets etc.
2. Selection of M functions chosen from the library for modeling the goal function by an orthogonalization method applying Gram-Schmidt.
3. Optimization of the selected functions to fit the best our goal function

We are going to describe in detail these 3 steps:

1. **Generation of the library:** In this step we are going to create a library of functions that must fit the best our goal function. There are different families of functions for constructing a library of functions, the most common are:
   (a) Radial Basis Functions
   (b) Wavelets
   (c) Neural Networks
   (d) Polynomials

   As we are going to see further in this project, it is important to create a library with a small number of functions to optimize the computational time.

2. **Selection of M functions from the library:**

- **Iteration 1:**

  (a) Selection of the most relevant function in D (library), function that has the smallest angle in respect of f.

$$cos^2(f, g_{\gamma 1}) = max\ \cos^2(f, g_{\gamma 1}) \tag{3.1}$$

$$cos^2(f, g_\gamma) = [\frac{\langle f, g_\gamma \rangle}{\|f\|\ \|g_\gamma\|}]^2 \tag{3.2}$$

  (b) Orthogonalization of f

$$r = f - \frac{\langle f, u1 \rangle}{\langle u1, u1 \rangle} \cdot u1 \tag{3.3}$$

  (c) Orthogonalization of D

$$D = g_\gamma = g_\gamma - \langle g_\gamma, u1 \rangle \cdot u1 g_\gamma \epsilon D \tag{3.4}$$

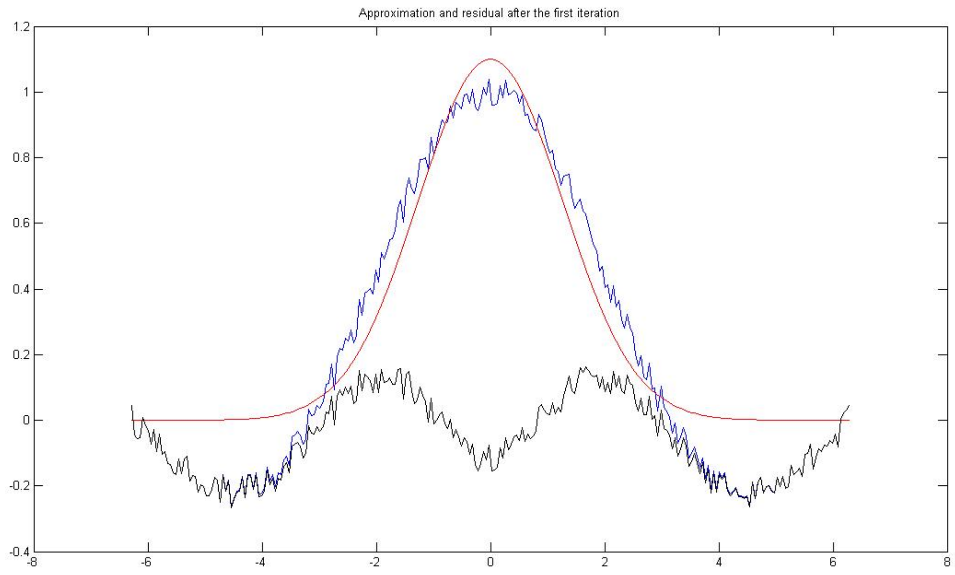

Figure 3.1: *First Approximation of the function* $\frac{sin(x)}{x}$

- **Iteration n:**

  (a) Selection of the most relevant function in D (library), function that has the smallest angle in respect of f.

$$\cos^2(r_n, g_{\gamma_n}) = max\ \cos^2(r_n, g_{\gamma_n}). \tag{3.5}$$

(b) Orthogonalisation of Rn

$$r_{n+1} = r_n - \frac{\langle r_n, u_n \rangle}{\langle u_n, u_n \rangle} \cdot u_n. \tag{3.6}$$

(c) Orthogonalization of Dn

$$^{n+1}D = \left\{ ^{n+1}g_\gamma = g_\gamma - \sum_{i=1}^{n} \langle g_{\gamma i}, u_i \rangle \cdot u_i g_\gamma \epsilon D \right\}. \tag{3.7}$$

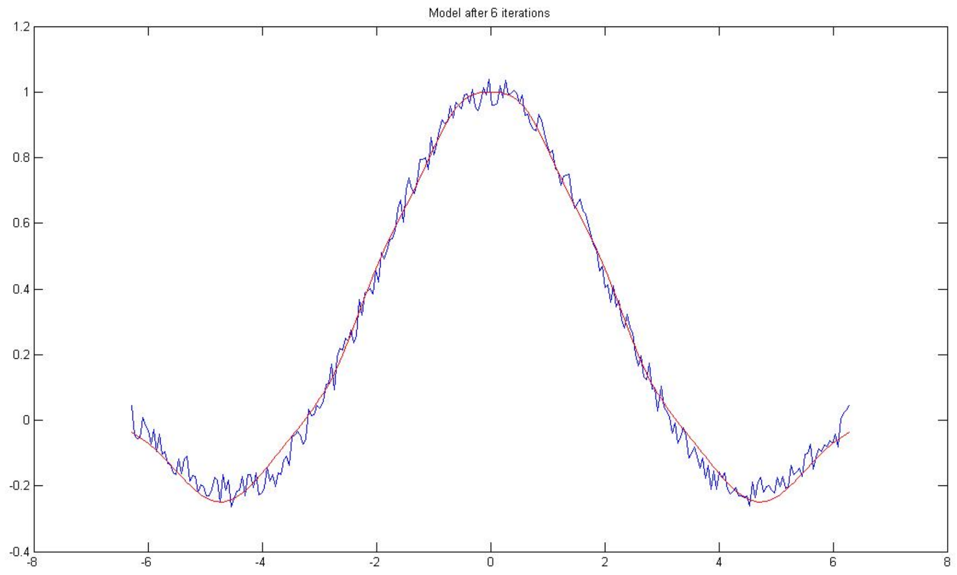

Figure 3.2: *Sixth Approximation of the function* $\frac{sin(x)}{x}$

After n iterations, where n can be any positive number we will obtain the model of the function. In every new iteration we will obtain a new function. Depending on the number of iterations and the functions of our library we will obtain a better model or worst model.

3. **Optimization:** There are different methods to optimize the functions to fit the best our goal function after the selection process. The most common are:

- Least Squares Method
- Simple Gradient Method
- Levenberg-Marquardt Method
- Newton Method
- Quasi-Newton Method

3.2 Applications

As we said the Orthogonal Forward Regression has lots of applications in different kind of fields. The most known applications are:

- **Regression of ECG recordings:** ECG stands for Electrocardiogram that is the graphical representation of the electric activity of the heart. Analyzing an ECG, an expert can detect different kind of problems, being able to save the life of people that suffers from certain kind of disease.

  The drawback of this process is that we need an expert (doctor) to analyze each one of the ECGs of their patients. The described process is slow because depends completely in the doctor.

  In that work Dubois et al. 2007 OFR was applied to an ECG to speed up this process, based on creating a general model of an ECG. This model will be composed by a single model of each one of the waves (P, Q, R, S, T) that form an ECG.
  Once a complete model of the ECG is constructed, is easy for the specialists to detect diseases by pattern recognition methods.

  As we saw previously, the first step of the Orthogonal Forward Regression consists of creating a library of functions.
  In this application, a set of Gaussian Mesa Functions will be created because by their properties can fit precisely all the ECG waves.

  Once the library is constructed, the aim of the regression process consists of obtaining all the waves in a precise way because the shape and the position of each wave will be useful to determine if a person suffers or not a certain disease.
  In the following image(borrowed from Dubois et al. 2007) we can observe how the ECG was decomposed in the waves P,Q,R,S,T:

  Once this process is done, we obtained a set of waves that need to be labeled, the process of labeling these waves is not related with OFR, for that purpose a Neural Network Classifier will be used.
  Labeling the different waves that compose an ECG, requires start labeling the R-wave to classify it as normal or ventricular beat, once this process finishes the rest of waves need to be also classified.
  The results of this study, were really satisfactory being able to detect ventricular beats with a ratio of 95% of success.

- **Modeling Etna volcanomagnetic dynamics:** Etna is a volcano situated in Italy and nowadays still active. The behavior of the volcano despite seems unpredictable, follows some patterns that are objective of the scientists to discover.
  OFR was applied to the data obtained in the stations that surround the Volcano with the aim of discover some behavior of it.
  The idea of this work Jankowski et al. 2010, consists of finding a mapping rule, between, the past values of the observed process, and its prediction.
  For that reason, the algorithm has 2 phases, in the first phase, the model state inputs

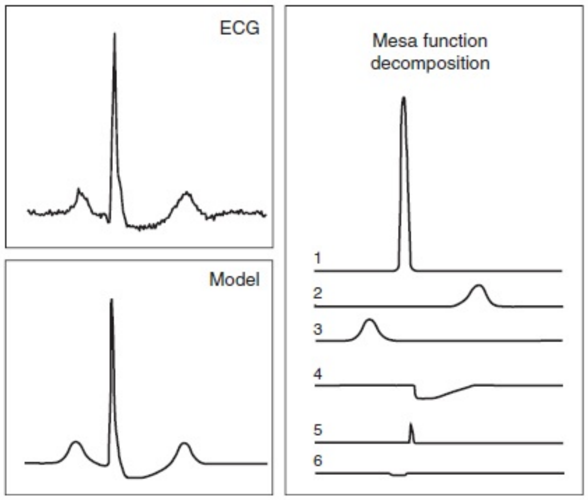

Figure 3.3: *Normal heartbeat broken up into Gaussian Mesa Functions*

are delayed measured outputs. In the second phase, the measured outputs are replaced by the estimated output values of the predictor, before the new learning phase takes place.
In the following image (borrowed from: Jankowski et al. 2010) we can see how the volcano data has been modeled:

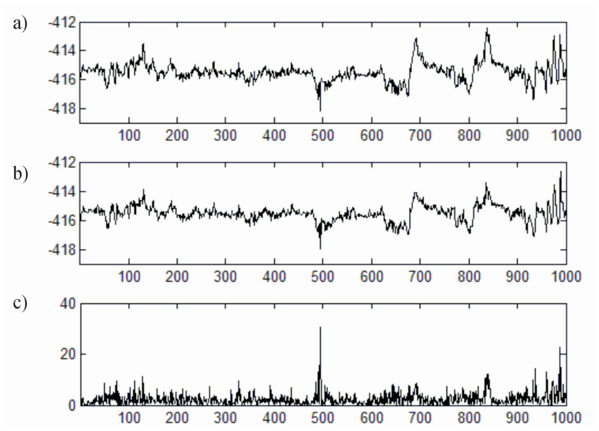

Figure 3.4: *Data from Etna control station, Model of the data and Percentage of Error*

Compared with the results obtained with other techniques applied to the same data, such as recurrent least-squares support vector machine (RLS-SVM) in this work Jankowski et al. 2009, the obtained model using OFR is better in all the senses.

Related with the quality of the results obtained, the OFR models are accurate enough to explain the chaotic mechanism of the observed processes, and to distinguish various modes of behavior.
And, in the performance sense, OFR requires 10 times less regressors at higher accuracy.

Comparing the Root Mean Squared Error (RMSE) of both methods, the RLS-SVM error is 0,79 and OFR error is less than the half with an error of 0,32.

Chapter 4

Non-linear dynamical systems

This chapter provides a brief introduction to Dynamical Systems, In this chapter we are going to describe what is a non-linear dynamical system, which is the relationship between that systems and chaos

4.1 What is a dynamical system?

A dynamical system is a formalization for any rule which describes the time dependence of a point in its space. At any time, a dynamical system has a state, which can be represented by a point in an appropriate space.
The evolution rule of this kind of systems, is a fixed rule, that let us know which will be the future states, so that rule is clearly a deterministic rule.
We find different types of dynamical systems, but in this work we are going to focus on non-linear dynamical systems.

4.2 Non-Linear dynamical systems and chaos

Nonlinear dynamical systems can exhibit a completely unpredictable behavior, which might seem to be random, despite are completely deterministic systems. This behavior has been called chaos.
These systems, form part of what we know as chaos theory, which studies the behavior of dynamical systems, that are very sensitive to initial conditions. Those kind of dynamical systems, are characterized by small differences in initial conditions affect widely the output of the system.
It's important to point out that, not all dynamical systems are considered as chaotic systems. A dynamical system, to be considered as chaotic system, need to satisfy the next properties:

1. **Must be sensitive to initial conditions:** A little change in the initial conditions is able to change completely the system. This property is known as Butterfly Effect, and we are going to study it in detail later in this chapter.

2. **Must be topologically mixing:** This property means that system will evolve over time, and any region of the set will probably overlap any other region.

3. **Its periodic orbits (attractors) must be dense:** This property means that every point is approached closely and arbitrarily by periodic orbits.

Despite not only the chaotic systems has attractors (it's common in all the dynamical systems) it's a very useful topic to understand the chaotic systems. We can describe an attractor as a set towards a dynamical system evolve over time.

We can give also a mathematical definition of it:

Being f(t,·) a function which specifies the dynamics of the system. And being x a point in the phase space, the state of the system a at certain time is f(0,x)=x. And for a positive value of a t, f(t,x) is the evolution of the state defined above after t units of time.

So, if the above system defined is a free particle in one dimension, therefore the phase space of it is the plane R^2.

We can define the evolution of this system by the next formula:

$$f(t,(x,v)) = (x + tv, v). \tag{4.1}$$

As we can see in the formula above, we define the system with 3 parameters:

- **t**: Time measure unit
- **x**: Position of the particle
- **v**: Speed of the particle

As we can see in the formula above, the system will evolve following a simple rule, the future position of the particle is the actual position plus the product of the amount of time spend it and the speed of the particle.

4.2.1 Types of Attractors

Despite attractors can be of any shape, we can distinguish different types.
We can distinguish between 2 well-known types of attractors (Fixed point and Limit Cycle). There are many other types of attractors that should be also classified in a certain category, but, and due to such a big number of different kind of attractors, all them are going to be classified in a third group known as Strange Attractors.

In the following lines we are going to define in detail the characteristics of each one of these groups:

1. **Fixed point:** Point of the model that does not change. It's contradictory that such a dynamical systems has fixed points due to the evolution of a dynamical system, but in every attractor can be one of this points. This point will be considered as the final state of the model.
   It's difficult to identify that kind of attractors in physical space, even though are well-defined in the phase space as the final state of it.

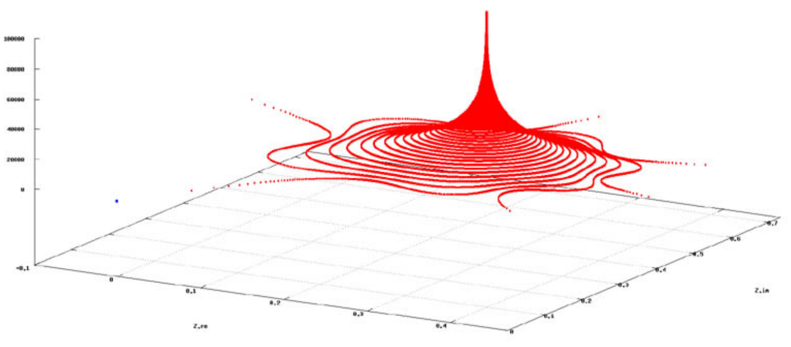

Figure 4.1: *Example of Fixed Point*

As we can see in the image above (borrowed from:
http://en.wikipedia.org/wiki/Attractor), the attractor presented has a fixed point as a final state (in the top part of the attractor).

2. **Limit cycle:** Periodic orbit of the system that is isolated. Real world examples can be a pendulum clock, or an ECG of a person that is sleeping (where we suppose there's no change in his/her beat rate). In the following image we present an example of limit cycle attractor (borrowed from: http://en.wikipedia.org/wiki/Limit-cycle)

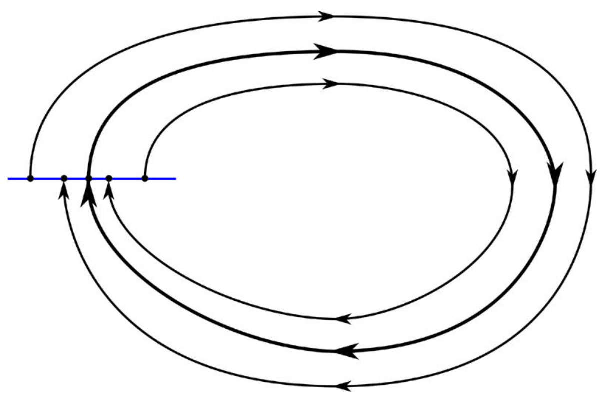

Figure 4.2: *Example of Limit Cycle*

We can differentiate a sub-type of attractor in that type of attractor.
That sub-type is called Limit tori, and we consider it as a sub-type of Limit Cycle and not as a Strange Attractor.We consider this because, is a Limit Cycle with more than one frequency in the periodic trajectory.
If two or more of these frequencies form an irrational fraction, the trajectory won't be close, therefore the limit cycle becomes a limit torus.
In the following image we present an example of limit tori attractor (borrowed from: http://www.math.harvard.edu/archive/21a_spring_06/exhibits/torus/index.html)

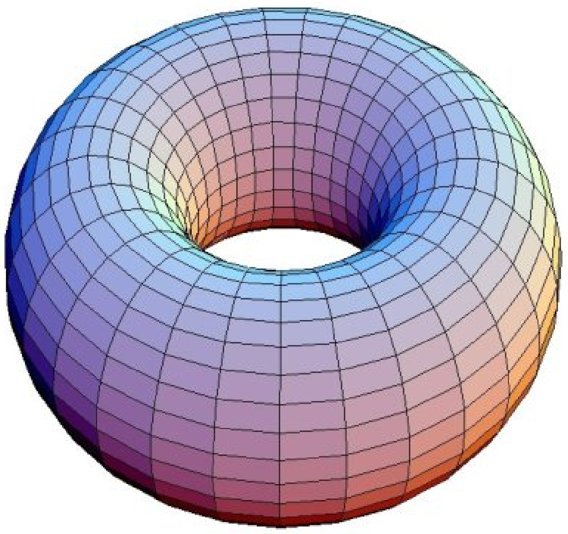

Figure 4.3: *Example of Limit Tori*

3. **Strange Attractors:** Attractors that can't be classified in any of two previous classifications will be classified as Strange Attractors.Strange attractors can be defined also, as that kind of attractors that has non-integer dimension.

   An example of well-known strange attractor is Lorenz Attractor, that we will study in detail later in this chapter.

   In the following image we present an example of strange attractor (borrowed from: http://math-art.net/2007/12/02/lorenz-attractor-a-3d-render/)

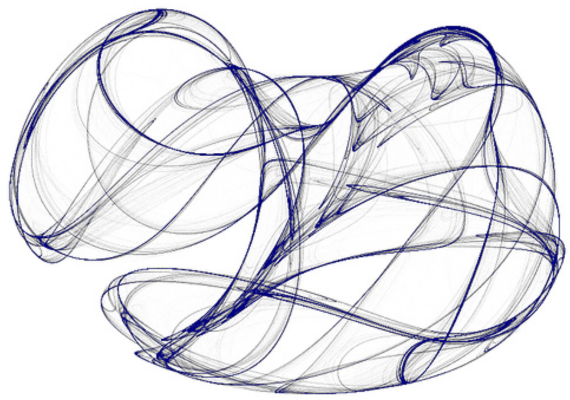

Figure 4.4: *Example of Strange Attractor*

4.2.2 Butterfly Effect

The butterfly effect is a well-known metaphor nowadays used in several fields, such as films, literature, television etc. In some of these fields, the Effect Butterfly is related with time travel. In popular culture this term is related with the ability of traveling to the past and change something of it. It is said, that if we could change a small aspect of the past, this change would affect widely the future.

In this chapter we are not going to focus on science-fiction, we will study the Effect Butterfly as a tool to understand what is the Chaos Theory.

The Effect Butterfly is also known as "system with sensitive dependence on initial conditions", and this is what characterize Chaos theory that, by small differences in the initial conditions of the system can produce big variations in the output of it.

This is what Lorenz discover in his experiments related with weather pattern. Lorenz measure lots of data related with the weather, and introduce it in an algorithm, with twelve recursive equations.

Lorenz realized that, his algorithm suffer big changes in the output when by mistake used a decimal with 3 instead of 6 significant digits.

Lorenz realized that small changes don't affect the system in the first recursive call, but and due to the recursive nature small changes in data become big changes at the n-call of the algorithm.

Lorenz can be considered the first scientific that discovered Chaos
Finally, to be able to explain in an easy way what he discovered, he decided to create the next metaphor.

The metaphor, apply chaos theory in the earth. Lorenz said that a simple flap of the wings of a butterfly (by example in Poland) is able to create a tornado far away from it (by example in United States).

That metaphor that sounds difficult to believe it was well justified by Lorenz. The butterfly flap generated small air currents surrounding it, but those air currents at the same time affect larger air currents, which affected still larger air currents and so on.
After all this process, Lorenz said that wind patterns of the earth could change completely being able to generate a tornado.

This metaphor, that is almost impossible to test, can be easily tested in a Chaotic system where a slightly change in the conditions of it can change completely its behavior.

4.2.3 Lyapunov Exponent

Lyapunov exponent of a dynamical system is a formula that, characterizes the rate of separation of infinitesimally close trajectories.

Looking for a simpler definition of Lyapunov exponent we can say that is a formula that able us to classify different types of orbits, by measuring how near are 2 different orbits.

Lyapunov says that, two trajectories in phase space (space where all possible states of a certain system are represented) diverge following the next formula:

$$|\delta Z(t)| \approx e^{t\lambda} |\delta Z_0| \tag{4.2}$$

where: δZ_0 is the initial condition and λ the Lyapunov Exponent.

The formula above, is the general formula of the Lyapunov Exponent, but that formula is not useful when looking for the Maximal Lyapunov Exponent (MLE).
It's interesting to look for the MLE, because is a number that will give us information about, future behavior of the dynamical system, being able to predict which will the behavior of the system in the future.

So we need to look for the ∞ limit of the formula above, to find the MLE:

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln \frac{|\delta Z(t)|}{|\delta Z_0|} \tag{4.3}$$

Once we obtained the formula to look for the MLE, we need can classify the different types of orbits by its λ:

- $\lambda < 0$: Orbits with a negative λ are attracted to a stable fixed point (as seen previously) or attracted to a periodic orbit.
  That kind of systems are characterized to be asymptotical stable, this means that the more negative the exponent, greater stability present. For that reason, orbits with $\lambda = -\infty$ are known as super-stable.

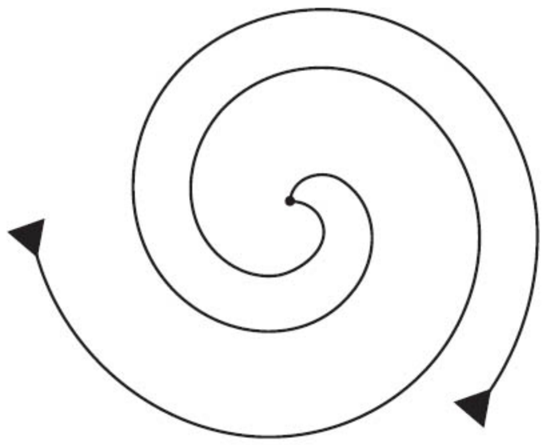

Figure 4.5: *Example of Orbit with a negative lambda (attracting fixed point)*

- $\lambda = 0$: Orbits with a zero λ are considered to present a steady state. Two orbits in this situation would create concentric circles, maintaining a constant separation between them.

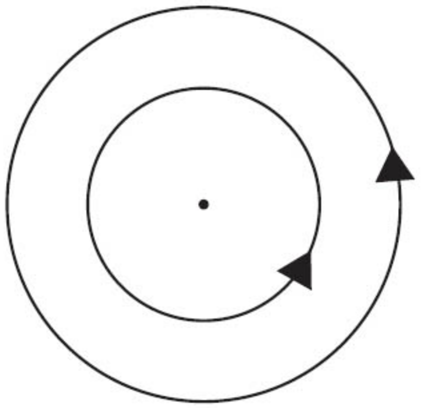

Figure 4.6: *Example of an Orbit with a zero lambda (neutral fixed point)*

- $\lambda > 0$: Orbits with a λ greater than 0 are known as chaotic orbits. That kind of orbits are characterized by small differences in the input parameters may have big differences in the output.

4.3 Lorenz Attractor

Lorenz Attractor is a fractal structure that corresponds to the behavior of Lorenz Oscillator, that can be modeled by a set of equations.

As we said previously, Lorenz discover by chance what is chaos while developing a model of earth atmosphere, when he realize that small differences in the starting point of their model create big differences in the output of it.

The Lorenz model has several applications in climate prediction. The model was designed to study the earth atmosphere and is considered and statement for the scientific community that planetary atmospheres may behave in a quasi-periodic regime, that is completely deterministic.

From the mathematical point of view Lorenz Oscillator is classified as a non-linear 3-Dimensional dynamical system, completely deterministic.

The equations that model the Lorenz oscillator are:

$$\begin{aligned} \tfrac{dx}{dt} &= \sigma(y-x) \\ \tfrac{dy}{dt} &= x(\rho-z) \\ \tfrac{dz}{dt} &= xy-\beta z \end{aligned} \tag{4.4}$$

where:

- σ: Is known as Prandtl Number, number that measures the relationship between momentum diffusivity and thermal diffusivity of some natural elements.
- β: Is known as Rayleigh Number, number used to measure the convection in fluid mechanics.

The following image (borrowed from: http://www.math.dartmouth.edu/archive/m53f07/public_html/) has been generated using the following values:

- $\sigma = 10$
- $\beta = 8/3$
- $\rho = 28$
- Iterations = 100000

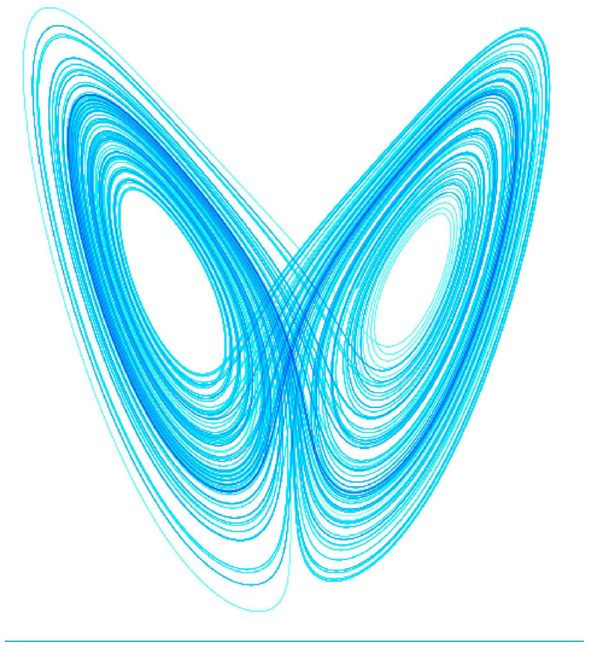

Figure 4.7: *Well-Known Butterfly Attractor generated using Lorenz Model*

The attractor shown in the image above, is really well-known in all the community, because chaotic behavior and looks like a butterfly. This attractor is useful to explain what is it the effect butterfly in chaos theory (already explained in this chapter).

The following image has been generated using the following values:

- $\sigma = 10$
- $\beta = 8/3$
- $\rho = 90$
- Iterations = 100000

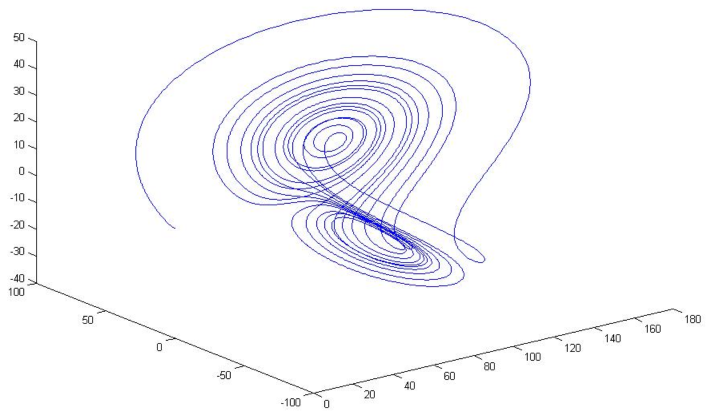

Figure 4.8: *Attractor generated using Lorenz Model*

4.4 Embedding

4.4.1 What is it Embedding?

Embedding surged as the necessity to analyze chaotic data, because other techniques such as Fourier can't be applied in chaotic signals.

To model a dynamical system, we need to study its phase space. The problem of chaotic systems is that its phase space is often unknown. For that reason, we need to reconstruct its phase space. To reconstruct it, we need to generate some signals as a function of time.

To reconstruct the phase space is important to take in consideration the following points:

- We need to separate the real signal from noise.
- We need to determine which is the appropriate space to analyze the signal.

The process of reconstruction of the phase space is composed 2 by steps that we are going to study in detail:

1. **Choose a time delay** This consists of generating a new set of coordinates by creating what is called 'time-lagged' coordinates.

   This theory was first announced by Taken, that create a theorem (Taken's Theorem) that provide the conditions under which an attractor can be reconstructed, from the time-delayed observations of a generic function.

   In the following formula, we present an easy way to create time-lagged coordinates:

$$dataset = [Y(t), Y(t-\tau), Y(t-2\tau), ...]. \tag{4.5}$$

In the formula above we created a new set of coordinates, specifically a dataset with 3 time-lagged signals with certain τ (time delay variable).

Despite choosing an arbitrary τ is enough to reconstruct any chaotic signal, it's important to choose an appropriate τ to maximize the efficiency of our embedding process. There are several ways to determine which is the best τ for a certain signal, but the most common ones are using autocorrelation or mutual information indexes.

2. **Choose an embedding dimension** To realize a good embedding it's also important to choose the appropriate dimension. It's said that as long as we choose a high d, the Embedding process will be successful, because with a higher d we have more possibilities to unfold completely the dynamics of the system.

   To choose the appropriate dimension, it's not as easy as to choose an arbitrary high d. We have to take into consideration the next aspects:

   - Large d, imply an increase in the computational cost.
   - If our signal present noise, the higher the dimension, the higher will be that noise.

   For the reasons exposed, it's important to apply some procedures that let us calculate which will be the best dimension, taking into consideration what we explained previously.

   To solve this problem, we can apply the approach of Kennel and Abarbanel that is known as 'False Nearest Neighbors'. This approach consist of, determining if the nearest neighbors in one dimension are also nearest neighbors in the next dimension (upper dimension). If this condition is true, this means that we arrive to the optimum dimension (completely unfolded dynamics). If not, we have to increase the dimension and keep checking this condition.

   To check that distance and Euclidean distance function can be used.

In the following lines we are going to present an example, that shows the effect of embedding applied to a signal generated by Lorenz Oscillator.

As we already know, Lorenz Oscillator has 3 output functions: x, y and z. In this example we are going to use just x, to show the effect of embedding with 1 function.

Once we run Lorenz Oscillator, if we plot the time series of x coordinate:
In the image above we observe X(t) in function of t, we don't observe any attractor because we're plotting the function in 1 Dimension.
In the next images, we want to present the effect of embedding, applying different values of τ:

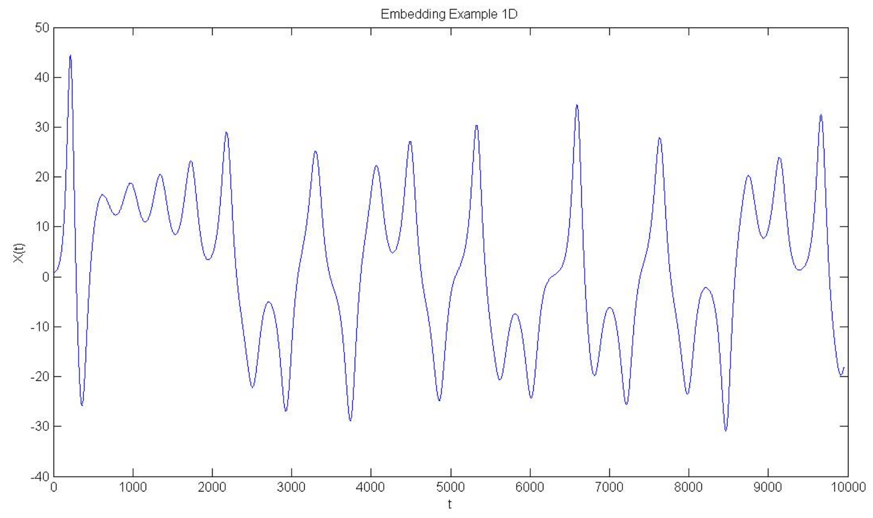

Figure 4.9: *Lorenz Attractor Plot with no Embedding*

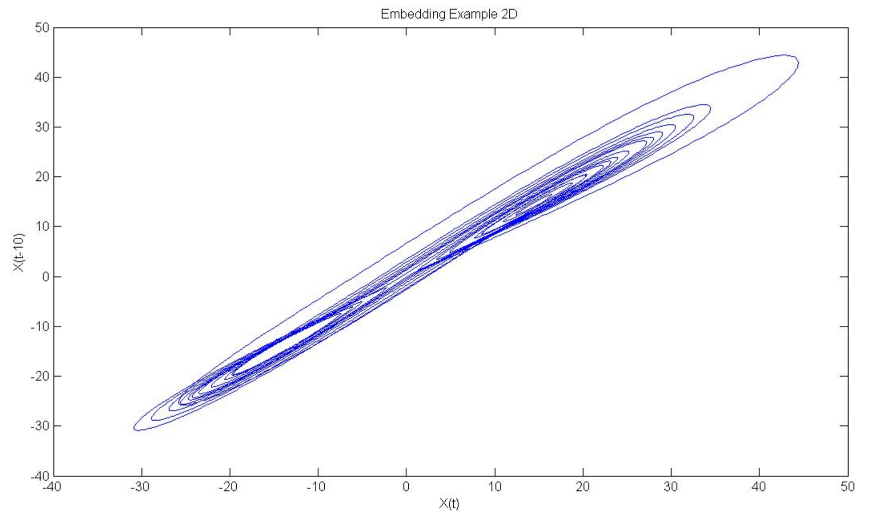

Figure 4.10: *Lorenz Attractor Plot with 2D Embedding and* $\tau = 10$

As we can see in the above images, we reconstructed the Lorenz Attractor by applying embedding to it. As we can see, we tried two different τ, and we observe that, the small the τ the thinner the attractor appears.

In the image above we can observe the reconstruction of the attractor in 3D. As we can observe, we decided to use $\tau = 25$ because with it, the attractor appears graphically better than with other options, e.g. $\tau = 10$.

Finally we can compare the reconstructed attractor, with the attractor generated with x, y and z coordinates.

Comparing both images we can observe that we applied successfully Embedding to Lorenz Oscillator.

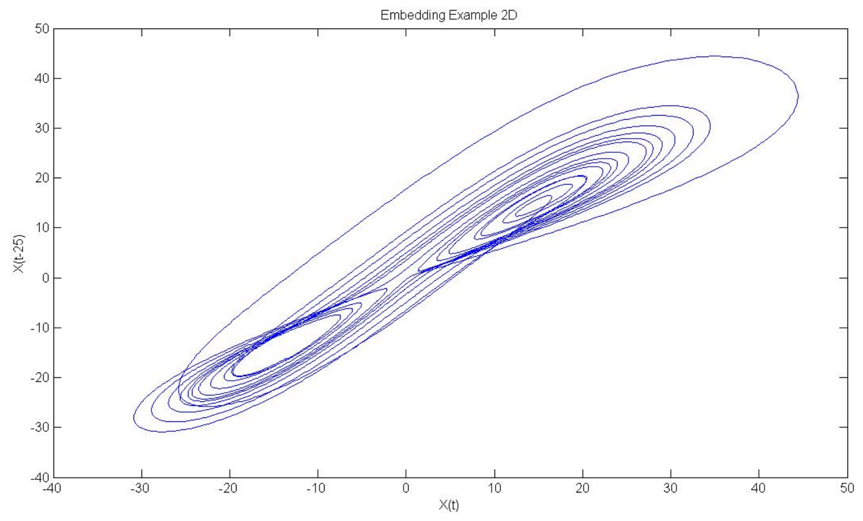

Figure 4.11: *Lorenz Attractor Plot with 2D Embedding and* $\tau = 25$

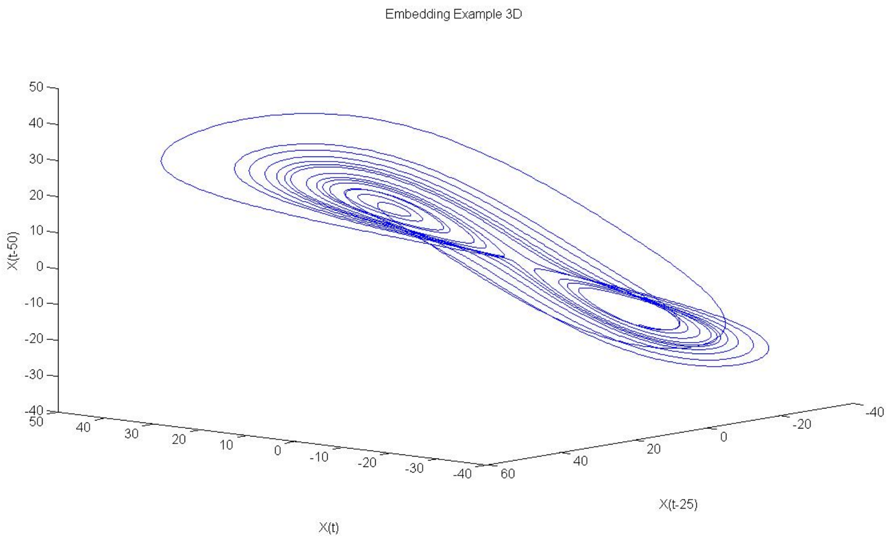

Figure 4.12: *Lorenz Attractor Plot with 3D Embedding*

4.4.2 Applications

There are several applications of Embedding, in the following lines we are going to describe shortly some of the most important ones:

- **ECG arrhythmia classification using simple reconstructed phase space:** In this work, Embedding was used to reconstruct the phase space of an ECG. The phase space then, is used to classify the ECG by checking how is the attractor distributed in that phase space.
- **Phase space reconstruction in the restricted three-body problem:** In this

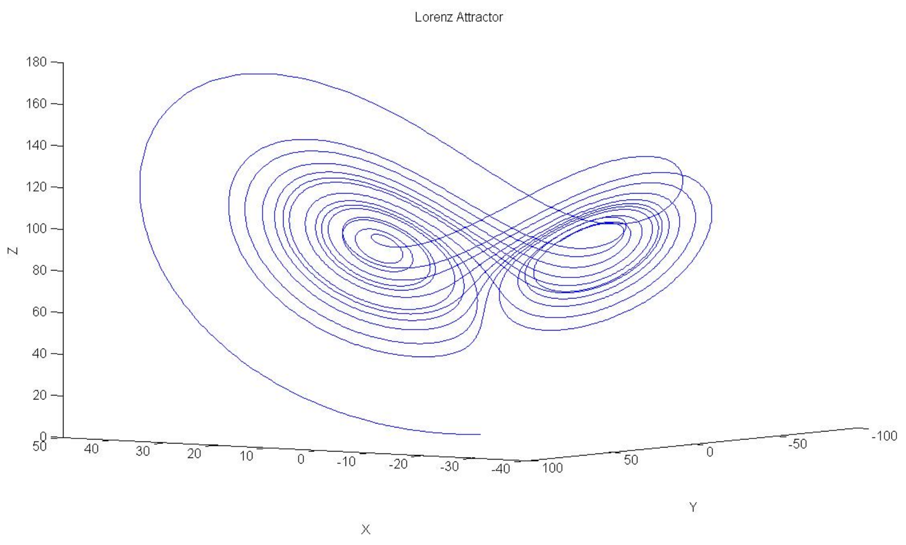

Figure 4.13: *Lorenz Attractor Plot with x, y and z coordinates*

work, Embedding was used to reconstruct the phase space of the three-body problem, to analyze the dynamics of it (to distinguish the different types of motion of this kind of system).

- **Determination of unstable periodic orbits and symbolic dynamics**
- **Approximation of attractor dimensions**
- **Noise Reduction applying time-series prediction**

4.5 Practical Learning Models

In the literature we realize that there are several learning models mentioned. In this section, we are going to describe the most common ones:

- **AR** Autoregressive, model that represents time series generated by passing a white noise to a recursive filter.

  The output of that kind of filters in the t moment is the weighted sum of previous values of the filter.

  Mathematically, this model can be represented by the following expression:

$$Y_t = \sum_{i=1}^{m} a_i \cdot Y_{t-i} + \varepsilon_t. \tag{4.6}$$

where:

a_1, a_2, a_3 are the coefficients of the recursive filter.
m is the order of the model.
ε_t uncorrelated error.

- **MA** Moving Average, model that represents time series by passing a white noise (stationary random process with zero autocorrelation) to a non-recursive filter.

  Mathematically, this model can be represented by the following expression:

  $$Y_t = \sum_{i=1}^{n} b_i \cdot X_{t-i} + \varepsilon_t. \tag{4.7}$$

  where:

  b_1, b_2, b_3 are the coefficients of the non-recursive filter.
  n is the order of the model.
  x_t elements of the white noise
  ε_t uncorrelated error.

- **ARMA** Autoregressive Moving-Average, model composed by two parts, an autoregressive (AR) part and a moving average (MA) part. The model is usually referred as ARMA(m,n), where m and n are two integers that refer to the orders of the autoregressive and the moving average part respectively.

  Mathematically, this model can be represented by the following expression, that as we can see is the composition of the two models that we have seen previously:

  $$Y_t = \sum_{i=1}^{m} a_i \cdot Y_{t-i} + \varepsilon_t + \sum_{i=1}^{n} b_i \cdot X_{t-i} + \varepsilon_t. \tag{4.8}$$

  where:
  a_1, a_2, a_3 are the coefficients of the recursive filter.
  b_1, b_2, b_3 are the coefficients of the non-recursive filter.
  m is the order of the AR model.
  n is the order of the MA model.
  x_t elements of the white noise
  ε_t uncorrelated error.

- **NARMAX** Non-linear Auto-Regressive Moving Average with exogenous inputs, nonlinear autoregressive with a Moving Average part and Exogenous Inputs.

  Mathematically, this model can be represented by the following expression:

  $$Y_t = F(y_{t-1}, ..., y_{t-n_y}, u_{t-1}, ..., u_{t-d-n_u+1}, e_t, ..., e_{k-n_e}). \tag{4.9}$$

where:

F is a nonlinear function such as Polynomial, Neural Network, Wavelet Network etc.
y_t output signal.
u_t input signal.
e_t noise, uncertainty etc.
d delay measured in sampling intervals.
n_y maximum index of the vector y.
n_u maximum index of the vector u.
n_e maximum index of the vector e.

- **NARX** Non-linear autoregressive exogenous, nonlinear autoregressive model which has exogenous inputs. That model relates the present value of the time series to:
  - Past values of the same series
  - Present and past values of the exogenous series
  - Residual term

  Mathematically, this model can be represented by the following expression:

$$Y_t = F(y_{t-1}, y_{t-2}, ..., u_{t-1}, u_{t-2}, ...) + \varepsilon_t. \tag{4.10}$$

where:

F is a nonlinear function such as Polynomial, Neural Network, Wavelet Network etc.
y_t variable of interest.
u_t variable associated with y_t.
ε_t uncorrelated error.

Chapter 5

Chua's Circuit

This chapter provides a brief introduction to Chua's Circuit. In this chapter we are going to describe what is Chua's circuit, why is this circuit so important, and why we decided to focus our study in Chua's Circuit.

5.1 What is Chua's Circuit?

Chua's circuit is a famous electronic circuit created by Leon O. Chua, which with few simple electronic elements is able to create chaotic behavior.
Chua's circuit is so important because, was the first, and simplest circuit able to create what we know as chaos. After the creation of the circuit, several researchers studied in detail such in Matsumoto 1984 or Matsumoto et al. 1985.
Chua defined, that any circuit constructed with the aim of producing chaos must contain:

1. **One or more nonlinear elements**
2. **One or more locally active resistors**
3. **Three or more energy-storage elements**

Despite, there are different versions of Chua's Circuit, the most common contain:

1. **Nonlinear elements**: 2 Resistors and 2 Diodes.
2. **Active resistors**: 1 Resistor.
3. **Energy-storage elements**: 2 Capacitors and 1 Inductor

Until that moment, we referred Chua's Circuit as a circuit, but it also can be described in a mathematical way.
From now on, in this project, we will make reference to Chua's Circuit as a system of 3 differential equations in the variables x, y and z. These variables, give the voltage and intensity of the different elements of the circuit.

$$\begin{array}{l} \frac{dx}{dt} = \alpha[y - x - f(x)] \\ \frac{dy}{dt} = x - y + z \\ \frac{dz}{dt} = -\beta y \end{array} \tag{5.1}$$

Derived from the formula we can describe in detail the next elements:

- α **and** β: Constant real numbers.
- **f(x)**: Scalar function that depends on x.

The function f(x) describes the response of the nonlinear resistor; the graphical representation of the circuit response, depends on the particular configuration of its components.

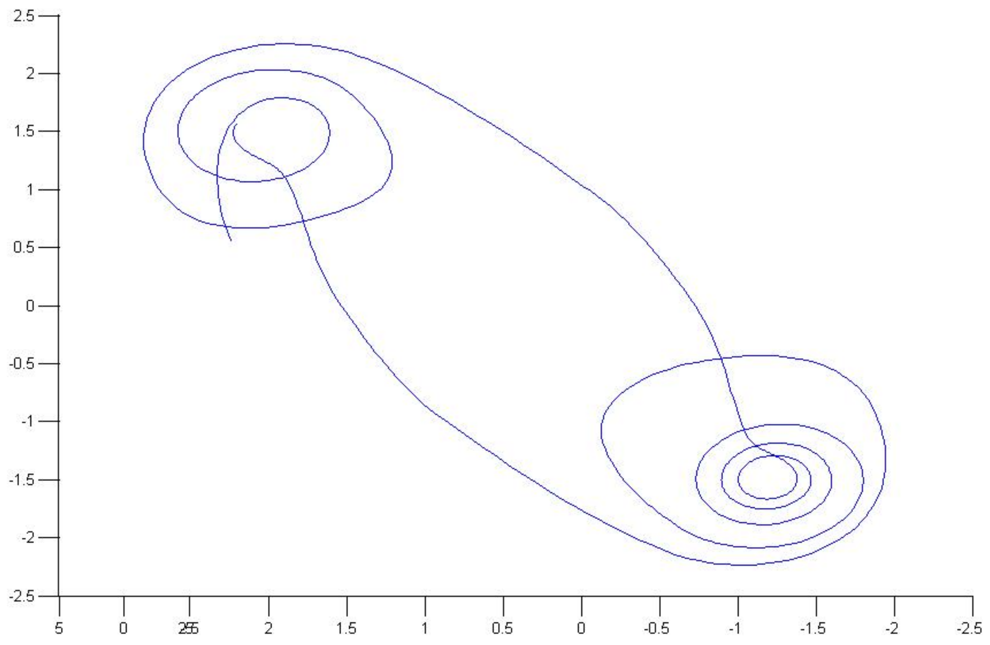

Figure 5.1: *Chua's Circuit representation with 11000 Iterations*

5.2 Generalizations of the circuit

There are several generalizations of the circuit, the most relevant ones are:

- **Modification of the nonlinear function to produce homoclinic orbits:**

  Substituting the non-linear function of the Chua Equations by the following formula:

$$f(x) = \frac{1}{16}x^3 - \frac{1}{6}x. \tag{5.2}$$

  and setting $\alpha = 10.91$ and $\beta = 14$

  Chua's circuit will be able to obtain a pair of homoclinic orbits

- **Modification of the third equation of Chua to produce a big number of topologically distinct chaotic attractors:**

  The modification of the third equation of Chua by the following Equation:

  $$z = -\beta y - \gamma z. \tag{5.3}$$

  In the equation above we observed the parameter γ that let us to produce a big amount of different attractors, Bilotta had reported more than a thousand distinct attractor with different geometrical structures (Bilotta et al, 2007).

- **Modification of the nonlinear function to produce multispiral attractors:**

  Modifying the non-linear function of the circuit, we can obtain multiple attractors

  $$f(x) = m_1 \cdot z + \frac{1}{2} \cdot (m_0 - m_1) \cdot [|x+1| - |x-1|]. \tag{5.4}$$

  Applying this modification we can obtain Chua's Circuit with up to 21 spirals.

  In the following image (borrowed from: http://membres.multimania.fr/maaziz/sp_gallery_chua.html) we present a 4-spiral strange attractor generated with Chua's Circuit:

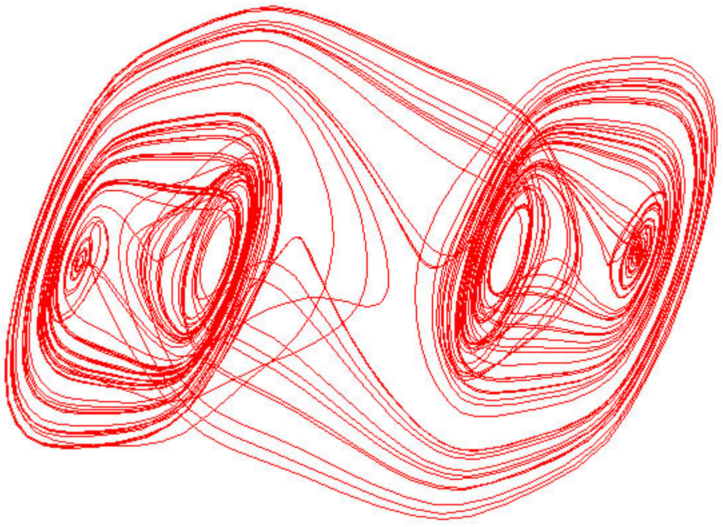

Figure 5.2: *4-spiral strange attractor by the modified Chua's circuit*

5.3 Applications

Chua's Circuit has several applications. Has been used as a physical source of chaotic signals in several experiments related with synchronization, such communications systems. It also has been used in numerical evolutions and in the evolution of natural languages.

Chua's Circuit have been used to generate 2D and 3D waves, for multiple applications such as image processing, neural networks, dynamical associative memories, complexity etc.

In the following lines we are going to describe briefly some of the applications mentioned above:

- **Trajectory Recognition:** Traditionally the process of trajectory recognition of the hands was based on the analysis frame by frame of the objects. This technique is highly-sensitive to the gestures and depending on that gestures the results are not reliable.

  In that work Altman 1993, Chua's Circuit has been used to recognize gestures by other techniques. To do this, the Circuit has been adapted and the piece-wise linear function has been changed by a cubic non-linearity function.

  To accomplish this task, a mapping of the input onto a dynamical system is required. Once this process is done, the trajectory recognition process is based on viewing each class of trajectories as a motion.

  Finally the motion trajectories from hand gestures will drive our dynamical system onto an attracting surface.

- **Synchronization:** Synchronization of chaos is an important technique for engineering, in applications such as secure communications. The aim of this work Zhong et al. 1998 and many other consist of coupling two chaotic systems in a way that their common signals are identical.

  The process of synchronizing a set of two identical Chua's circuits can be achieved in a mutual and an unidirectional coupling manner.

  The aim of this work is to check the robustness of the mentioned synchronization by modifying the system parameters. Additionally a set of critical regions have been defined to classify the chaos in the context of synchronization.

- **Control of Chaos:** In this work Wang and Tanaka 1996, a framework for the stability and design of nonlinear fuzzy control systems has been developed.

  Chua's Circuit has been used as an application to test the performance of the framework.

  The process of creating the fuzzy controller is based on the following steps:

  1. **Creation of the model:** In this type of fuzzy model created, local dynamics in different space regions are represented by linear models.
  2. **Design of the Control:** The control designed, for each local liner model will create a linear feedback control.
  3. **Application to Chua's Circuit:** Finally the developed framework has been tested and semi-global stability of Chua's Circuit has been achieved.

Chapter 6

Software Requirements Specification

In this chapter we are going to specify the requirements of the software that we are going to implement. This a study needed to realize before the implementation of any software to detect all the requirements of an application, by diagraming and describing the expected behavior of the software.

6.1 Introduction

6.1.1 Purpose

This Software Requirements Specification provides a complete description of all the functions and specifications of the project: "Modeling non-linear dynamical systems by Orthogonal Forward Regression".
The expected audience of this document are the two universities related with the project, "Universitat Ramon Llull La Salle" and "Politechnika Warszawska".
As a reference to realize this study we used Iee 1998.

6.1.2 Scope

The scope of this project is limited to the departments of the faculties or companies that research in statistical learning and specifically in non-linear dynamical systems.

6.1.3 Glossary

Term	**Definition**
OFR	Orthogonal Forward Regression
RBF	Radial Basis Function
LS	Least Squares
MSE	Mean Square Error
IDE	Integrated Development Environment
SRS	Software Requirements Specification

Table 6.1: *Glossary of terms*

6.1.4 Document Overview

This document is organized as follows, the first provides a full description of the project. In the second, each of the elements that compound the application are detailed for the software developer assistance.

6.2 Overall Description

6.2.1 System environment

The project described in this SRS, is a new product, doesn't form part of any bigger system neither related with any other system. For that reason we considered, that it isn't needed to create a package diagram of it.

6.2.2 Product Features

The system that we are defining will be able to:

1. Create chaos.
2. Model chaos.
3. Optimization of the results.

The inputs and outputs of our system will be the following:

- **Program Inputs:** The observed one-dimensional sampled signal of selected observable, of a given dynamical system as function of time.
- **Program Outputs:** Approximation of the trajectory of a given dynamical system calculated by OFR method

Description of the results:

- **Model Specification:** Table of selected orthogonal components of approximation (RBF functions described by their weights, position in 3-dimensional space, width parameter etc).

- **Evaluation of the results:** Root mean square error of approximation, visualization of the studied observable, 3-dimensional trajectory.

6.2.3 User classes and characteristics

Our system doesn't have different users neither different classes. Just one kind of user will run the application.

6.2.4 Operating Environment

The environment in which the software will run is the following:

- **Operative System:** Windows 7
- **IDE:** Matlab R2009b
- **Hardware**: Dell XPS 13

6.2.5 Design and Implementation Constraints

We should consider that this software is running in a powerful laptop, but must run without problems in any other computer with Matlab installed, for that reason we are going to present some hardware constraints:

- **Memory:** The software should not demand more than 1GB.
- **Time:** The software should not spend more than 1 minute for returning the results.

6.2.6 User Documentation

A user manual will be delivered together with the software.

6.2.7 Assumptions and Dependencies

We assume that the computers where the software that we present should run should have installed Matlab.

6.3 System Features

1. Create chaos dynamics by using Chua's Circuit Equations.
2. Model chaos by applying OFR.
3. Optimize the results by applying different Optimization Methods.

6.3.1 Create chaos

Description

We are going to create behavior by using the Chua's Circuit equations. Once, we run the algorithm, an Embedding process will be carried on.

Response Sequences

The response is a dataset with chaotic data.

6.3.2 Model chaos

Description

To model the chaotic behavior OFR will be used. OFR is an algorithm able to model any kind of function, if a suitable library of functions (to do the regression process) is created. In our case a library of RBF's functions will be given.

Response Sequences

The response is a model that is close to our goal function.

6.3.3 Optimization of the results

Description

To optimize the results obtained after the OFR algorithm we are going to implement different optimization methods such as Least Squares Method, Simple Gradient Method etc.

Response Sequences

The response, is a function closer to the goal function.

6.4 External Interface Requirements

None.

6.5 Apendix

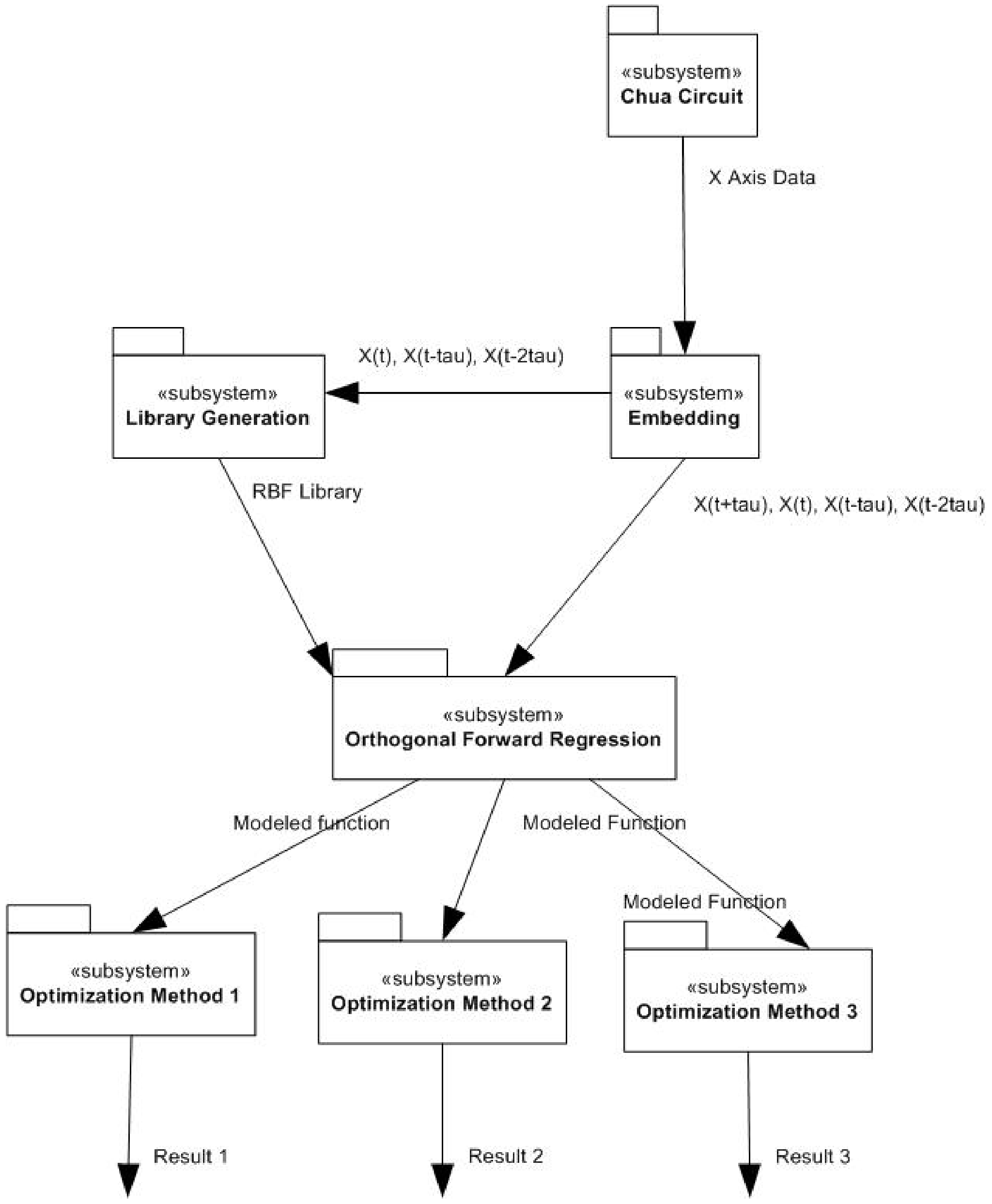

Figure 6.1: *Module Diagram of our Application*

Chapter 7

Modeling Chua Attractor by ORF

In this chapter we are going to specify all the details related with the implementation of our program. From the creation of the dataset, to the explanation of the different optimization methods that will let us getting a better result.

7.1 General scheme of our Software

Up to now, we analyze the theory of OFR, Embedding, Non-Linear Dynamical Systems etc. In this section we are going to describe how these elements work in practice.

In the following lines we are going to describe all the process that follows our software from the very beginning to the last step, in which the results are obtained:

1. **Creation of the Dataset:** This part of the project consists of Generating the Dataset that we are going to use in the rest of the application. We divide this part in the following 2 sub-parts:

   (a) **Chua's Circuit:** The first part of our project consist of generate Chaos.
   To do it we used Chua's Circuit as a well-known Non-Linear Dynamical System able to produce chaos. To produce that chaos we created an algorithm that uses Chua Equations 5.1.
   In the following lines we can observe the algorithm that we created for that aim:

```
function [x,y,z]= chua(x,y,z,alfa,beta,ab,ba,dt)

    x1=x;
    y1=y;
    z1=z;
    z2=z1-beta*y1*dt;
    y2=y1+(x1-y1+z1)*dt;

    if (x1<-1.0)
        x2=x1+(alfa*(y1-ba*(x1)-ba+ab))*dt;
    else
        if (x1>1.0)
            x2=x1+(alfa*(y1-ba*(x1)-ab+ba))*dt;
        else
```

```
                x2=x1+(alfa*(y1-ab*(x1)))*dt;
            end
        end

        x=x2;
        y=y2;
        z=z2;
    end
```

To analyze the algorithm above, we are going to study its inputs and outputs:

- **Inputs:** The inputs of the algorithm are:
  - **x, y and z:** Data vectors, contain all the data already generated by the algorithm. Will have the same length than the number of times we iterate this algorithm.
  - **alfa, beta:** Constants of the Chua Equations. The values that we are going to use for that constants are: 9.2 and 14.5 respectively.
  - **ab, ba, dt:** Values of the different resistors and diodes, that we need to simulate the behavior of Chua's Circuit.
- **Outputs:** The only outputs of the algorithm are the above commented data vectors x, y and z. This are the vectors that we will use to generate our dataset.

(b) **Embedding:** Applying Embedding to our problems enables to obtain sparse approximation of the dynamical rule: the unknown mapping of a set of past values of the observed process into its future value.
To do it The D-dimensional nonlinear dynamical system is observed by only one observable. The sampled observable x(t) is used to perform the reconstruction of the trajectory in D-dimensional space by using the delayed coordinates.
The reconstruction is calculated by the embedding theorem.
The model of the considered autonomous dynamical system is:

$$x(t+\tau) = F[x(t), x(t-\tau), ..., x(t-(D-1)\tau)] \tag{7.1}$$

Finally we are going to define the following summary table, which defines how our system is going to embedded and which will be our measured and model outputs.

	Measured Inputs			**Measured Output**	**Modeled Output**
t	**x(t)**	**x(t-τ)**	**x(t-2τ)**	**x(t+τ)**	$\hat{x}$**(t+τ)**

Table 7.1: *Definition of the inputs and outputs of our system after embedding*

As we can see in the table above, our goal function is $x(t+\tau)$, and the model function that we are going to create is $\hat{x}(t+\tau)$, to do it, we will use OFR algorithm. x(t), $x(t-\tau)$ and $x(t-2\tau)$ after the embedding process will be considered as equivalent to X, Y and Z axis.

2. **OFR** OFR (Orthogonal Forward Regression) is the algorithm that we are going to use to find a function as close as possible to our goal function, that as we said previously is $x(t+\tau)$. We are going to divide the explanation of this algorithm in 3 parts:

   - **Generation of the library** OFR algorithm needs a library of functions to do the regression process.
     We decided to built a library of RBF's functions because are a kind of functions that are easily computed.
     The number of functions that we create in that library is an important aspect. A few number of functions may cause bad regression, and big number may cause a high computation cost without gaining regression quality.
     Good results have been obtained setting the generation of functions to scale 3, or what is the same, using 160 functions.

     As we are going to see in detail in the following section 7.2, the process of generating the library is a complex process, to which depends to obtain or not a good model.

   - **Select (and optimize) M functions from the library** As described theoretically in the section 3.1, the process of selecting M functions from the library, with the aim of modeling our goal function, consists of the following steps:
     - **Iteration 1:**
       (a) **Selection of the most relevant function in D:** By using the correlation coefficient 3.1 we select the function from the Library that fit the best the goal function.
       (b) **Tunning of the function:** In the Temporal Approach, once a function is selected, we need to tune it using MSE.
       (c) **Orthogonalization of f:** Once we selected one function from the library (u1) we need to orthogonalize our goal function (f) in relation with that function (u1) to obtain the residual part of f (r) 3.3.
       (d) **Orthogonalization of D:** Once we selected one function from the library (u1), we need to orthogonalize it and store it again to the library of functions 3.4.
     - **Iteration n:**
       (a) **Selection of the most relevant function in D:** By using the correlation coefficient 3.5 we select the function from the Library that fit the best the residual part (r_n) of our goal function.
       (b) **Tuning of the function:** No change in respect of the Iteration 1. We tune the function selected in the above step.
       (c) **Orthogonalization of R_n:** Once we selected one function from the library (u_n) we need to orthogonalize the residual part of our goal function (R_n) in relation with that function (u_n) to obtain the residual part of R_n (R_{n+1}) 3.6.
       (d) **Orthogonalization of D_n:** Once we selected one function from the library, we need to orthogonalize it and store it again to the library of functions 3.7.

3. **Optimization** As we are going to see further in detail in the section 7.3 there are 2 kinds of optimizations methods that can be applied to our aim. Are the following:

    (a) **Temporal Approach:** Temporal approach optimizers, are the one's that are able to optimize functions just one by one. It's the case of the optimization that takes place while the OFR process is running. In other words, the temporal approach optimization, is that kind of optimization, that optimize a function just after being selected in the OFR process.

    (b) **Vectorial Approach:** Vectorial approach optimizers, are the one's that are able to optimize all the functions altogether. Vectorial approach optimization, optimizes all the functions after the OFR process. That approach is the one that follows the Vectorial LS, Simple Gradient Method, Levenberg-Marquardt etc.

7.2 Creating the library of functions

As we saw previously in the OFR chapter, OFR algorithm needs a library of functions to do the regression process. In the following lines we are going to detail which family of functions we decided to use, and how we built them.

After studying the different candidate families of functions to be part of the OFR library, we decided to use RBF functions. We decided to use this family of functions because are easily computed and highly adaptive to our problem.

To create the library of RBF functions, we are going to apply the following formula:

$$RBFi = e^{-}\left(\frac{(xi - c1)^2 + (yi - c2)^2 + (zi - c3)^2}{2\sigma^2}\right). \tag{7.2}$$

To apply that formula we need, the data from the Chua's circuit (x, y and z Vectors), but we also need to know the centers and the σ of our rbf's.

The centers and the sigma are strongly related, and we can describe the next property:

$$2 \cdot \sigma <= center, \tag{7.3}$$

where σ is the amplitude of the half of the function

As we already know, our function is a 3 Dimensional function, and for that reason we need to define the centers for each one of their Dimensions.

To generate these centers we first think in assigning them by hand, but this process becomes too complicated

To generate them in an easy way, we can think in a cube.

The idea consists of using the corners of a cube as centers of our RBF, each one of the dimensions of the Cube, represents a dimension of our function.

As we know, a cube has 8 corners, so in fact we have just 8 different centers, and only 8 RBF's.

If we want to increase the number of RBF's that will form our library, we are going to divide by 2 the cube several times. Doing this, we will obtain new sub-cubes and more edges. From now on, this process will be called increasing of the Scale.

In the next Sections we are going to describe how we can generate the centers in different Scales.

7.2.1 Creating RBF's of Scale 1

If we suppose a cube of Scale 1, we will obtain 8 centers. But which will be the values of these centers?
These values will be calculated using combinatory properties, assuming that we have 3 dimensions and in each dimension we have 2 possible values, the maximal value and the minimal, we have 8 different combinations.
That combinations are:

Center	**Value 1**	**Value 2**	**Value 3**
Center 1	Xmin	Ymin	Zmin
Center 2	Xmin	Ymin	Zmax
Center 3	Xmin	Ymax	Zmin
Center 4	Xmin	Ymax	Zmax
Center 5	Xmax	Ymin	Zmin
Center 6	Xmax	Ymin	Zmax
Center 7	Xmax	Ymax	Zmin
Center 8	Xmax	Ymax	Zmax

Table 7.2: *Values of the centers in Scale 1*

Supposing the next values for the maximal and minimal values of each dimension:

Description	**Value**
Min. value in X Dim.	-1
Max. value in X Dim.	1
Min. value in Y Dim.	-2
Max. value in Y Dim.	2
Min. value in Z Dim.	-3
Max. value in Z Dim.	3

Table 7.3: *Example of values of the centers in Scale 1*

The values that we will obtain for each one of the centers are:

Center	**Value 1**	**Value 2**	**Value 3**
Center 1	-1	-2	-3
Center 2	-1	-2	3
Center 3	-1	2	-3
Center 4	-1	2	3
Center 5	1	-2	-3
Center 6	1	-2	3
Center 7	1	2	-3
Center 8	1	2	3

Table 7.4: *Set of Example centers in Scale 1*

With the table above, we could be able to generate 8 RBF's. But, we still need to define Sigma, that will be defined as: $Sigma = xmax - xmin$.

With all this information, we can describe the RBF 1 as it follows:

$$RBF1 = e^{-}\left(\frac{(xi-(-1))^2+(yi-(-2))^2+(zi-(-3))^2}{2\cdot(-1-(-3))^2}\right). \tag{7.4}$$

7.2.2 Creating RBF's of Scale 2

Due to the fact that, 8 centers let us create just 8 RBF's we need to improve our method to increase the number of centers and with this also the number of RBF's.

The process of increasing the Scale is easy using combinatory and imagining a cube as the next one:

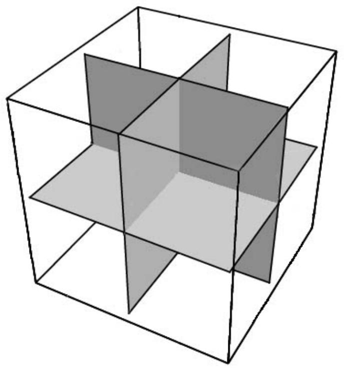

Figure 7.1: *Division of a Cube as in Scale 2*

As we said, to increase the Scale means to divide all the sides of the cube by 2. Dividing every side by 2, what we are doing is creating more edges.

Considering that we have 3 dimensions, and in each side 3 edges, we have 3^3 centers in Scale 2.

As we said, in Scale 2, our cube has 3 edges in any side, but which are the values of these edges? That values are:

1. **Value 1**: Minimal Value of the Dimension.
2. **Value 2**: Medium Value of the Dimension. This value can be calculated by $\frac{maxval-minval}{2}$.
3. **Value 3**: Maximal Value of the Dimension.

Following the process described above, in that Scale we also need to:

1. **Generation of combinations**: In Scale 2, we need to generate, 3^3 set of centers, for doing this, we are going to use combinatory.
2. **Assign values to the combinations**: Once we generated the different combinations, we need to assign the values described above to each one of that combinations.
3. **Generation of the RBF's**: With the data described above, we just need to generate the RBF's using it.

7.2.3 Creating RBF's of any Scale

As we described previously, it's possible to generate RBF's with Scales 1 and 2. The process of generation of RBFs with higher Scales follow the same procedures described above, and for that reason we are not going to describe it in such a deepness.

The values that characterize each one of the Scales between 1 to 4 are:

1. **Scale 1:**
   - **N**umber of Centers: $2^3 = 8\ Centers$
   - **S**igma: $(xmax - xmin)$
2. **Scale 2:**
   - **N**umber of Centers: $3^3 = 127\ Centers$
   - **S**igma: $\frac{(xmax-xmin)}{2}$
3. **Scale 3:**
   - **N**umber of Centers: $5^3 = 125\ Centers$
   - **S**igma: $\frac{(xmax-xmin)}{4}$
4. **Scale 4:**
   - **N**umber of Centers: $9^3 = 729\ Centers$

- **S**igma: $\frac{(xmax-xmin)}{8}$

Until that moment, we described the process of increasing the Scales as just the process of dividing the cube by 2. But what happens for example, with the centers of Scale 1 when we are generating the Centers in Scale 2?
To increase the Scale means to Sum the centers in previous Scales, in that table we summarize the number of centers that are generated in every Scale:

Scale	**Centers**
1	8
2	8 + 27 = 35
3	8 + 27 + 125 = 160
4	8 + 27 + 125 + 729 = 889

Table 7.5: *Number of centers in different Scales*

7.3 Optimization Methods

In this section we are going to describe the different optimization methods used to tune our functions to fit the best our goal functions.

7.3.1 Least Squares Method Temporal Approach

Using this kind of optimization, we look for the optimal weights for every one of our RBF's that form our Model.
The weights are calculated one by one for each new RBFs selected. The weights are not going to be changed once calculated, for that reason we can consider this method as a statical calculation of the weights.

Using this method we are going to apply MSE between the residual part of the Goal Function and the RBFs.

$$error = \sum_{i=1}^{N}[f_i - w \cdot u_i]^2 \tag{7.5}$$

With the formula above we calculate the error between our goal function (f) and the RBF selected from the library, multiplied by certain weight (u and w respectively). The aim of this process consist of identifying which is the weight (w) that minimizes our error.

Once the weight that minimizes our error is identified, our RBF, will be the extracted RBF from the library multiplied by the weight that we found, as the weight that minimizes the error.

7.3.2 Least Squares Method Vectorial Approach

Using this kind of optimization, we look for the optimal weights of every one of our RBF's that form our Model but not as described in the Temporal Approach. In that case all the weights are calculated all in once, when all the RBF's have been already selected.

The Formula used to calculate the weights is:

$$wopt = (X^T \cdot X)^{-1} \cdot (X^T \cdot Y_p) \tag{7.6}$$

where:

- **X**: is a Matrix with all the RBF's selected without being orthogonalized. That matrix will have as many columns as the number of RBFs and the length of each column will be the length of the elements that form each RBF.
- **XT**: is X transposed.
- **Y**: is the goal Function.
- **Wopt**: vector with the Weights of all the RBF's. That vector will have as many elements as the number of RBFs.

Once this process finishes, we obtained the vector of weights, Wopt. To obtain the optimized RBF we need to multiply each RBF by their correspondent Weight.

We are going to describe this process in the next formula:

$$\begin{bmatrix} RBFopt_{1,1} & \cdots & RBFopt_{1,N} \\ RBFopt_{2,1} & \cdots & RBF_{2,N} \\ \vdots & \vdots & \vdots \\ RBF_{N_E L,1} & \cdots & RBF_{N,N} \end{bmatrix} = \begin{bmatrix} wopt_1 \\ \\ wopt_2 \\ \\ \vdots \\ \\ wopt_n \end{bmatrix} \cdot \begin{bmatrix} RBF_{1,1} & \cdots & RBF_{1,N_E L} \\ RBF_{2,1} & \cdots & RBF_{2,N_E L} \\ \vdots & \vdots & \vdots \\ RBF_{N,1} & \cdots & RBF_{N,N_E L} \end{bmatrix} \tag{7.7}$$

In the formula above, we can see that the result of the multiplication of the vector of weights by the matrix of non-optimized RBF's is another matrix with the optimized RBF's.

As we are going to describe further in the results section, the above method has a problem when the number of RBF's is higher than 16.
This is due to a rank deficiency problem. This problem happens when in the matrix X appear the same RBF twice. In others words happens when the matrix has not full rank.
To solve this problem, we looked for alternative methods to compute the weights. We found another method using what we know as Moore-Penrose pseudoinversion.

With this method, we can compute X^+ without taking care if X, is or not a full rank matrix.

$$X^+ = (X^T \cdot X)^{-1} \cdot (X^T). \tag{7.8}$$

X^+ is equivalent to Moore-Penrose pseudoinversion.
To compute the weights using this new method, we will apply the following formula:

$$wopt = X^+ \cdot Y_p. \tag{7.9}$$

Applying this formula we extended the number of approximations to more than 16 as we used to do. We can observe the results of the application of this method in the results section.

7.3.3 Simple Gradient Method

This method is also known as steepest descent, and consists of finding the parameters of our function that let us minimize the error, going on the direction of the minus gradient.

The steepest descent method, can be seen as a 3-step iterative method, that steps are:

1. Computation of the Gradient.
2. Construction of the next Point.
3. Repeat step 2 until we arrive to a certain number of iterations or we arrive to a certain degree of minimization.

The computation of the Gradient is an easy but, long task that we are going to describe in detail.

Our aim is to minimize the error between, our goal function and the RBF's that approximate that function. To do that, we have to review the definition of a RBF function:

$$b(x, y, z) = \exp\left(-\frac{(x-c1)^2 + (y-c2)^2 + (z-c3)^2}{2\sigma^2}\right). \tag{7.10}$$

As we can observe, any RBF function has 5 parameters that are the ones that we want to tune, to minimize the error, that parameters are:

$$\theta = [w, \sigma, c1, c2, c3]^T,\ q = 5. \tag{7.11}$$

In the following formula we are going to detail how we can obtain the Mean Square Error between our goal function and a certain RBF:

$$E = \frac{1}{2N}\sum_{i=1}^{N}[f_i - wb(x_i, y_i, z_i)]^2 = \frac{1}{2N}\sum_{i=1}^{N}\left[f_i - w\exp\cdot\left[\frac{(x-c1)^2 + (y-c2)^2 + (z-c3)^2}{2\sigma^2}\right]\right]^2 \tag{7.12}$$

$$\begin{aligned}
\frac{\partial E}{\partial \theta} &= \frac{\partial}{\partial \theta}\left[\frac{1}{2N_{total}} \sum_{i=1}^{N_{total}} [f_i - wb(\sigma, c1, c2, c3, x_i, y_i, z_i)]^2\right] \\
&= \frac{\partial}{\partial \theta}\left[\frac{1}{2N_{total}} \sum_{i=1}^{N_{total}} [f_i - Y_p(\theta, x_i, y_i, z_i)]^2\right] \\
&= -\frac{1}{2N_{total}} \sum_{i=1}^{N_{total}} [f_i - Y_p(\theta, x_i, y_i, z_i)]^2 \frac{\partial Y_p}{\partial \theta} \\
&= -\sum_{i=1}^{N_{total}} e(\theta) \frac{\partial Y_p}{\partial \theta}
\end{aligned}$$

where:

- $\frac{\partial Y_p}{\partial \theta}$: Jacobian Matrix, formed by N columns (number of elements of each RBF), q rows (where q equals 5, the number of parameters of our RBF), i is the actual iteration.

$$\frac{\partial Y_p}{\partial \theta} = \begin{bmatrix} \frac{\partial Y_p}{\partial w} \\ \frac{\partial Y_p}{\partial \sigma} \\ \frac{\partial Y_p}{\partial c1} \\ \frac{\partial Y_p}{\partial c2} \\ \frac{\partial Y_p}{\partial c3} \end{bmatrix} = \begin{bmatrix} -b[x(t), x(t-\tau), x(t-2\tau)] \\ wb[x(t), x(t-\tau), x(t-2\tau)][\frac{(x(t)-c1)^2+(x(t-\tau)-c2)^2+(x(t-2\tau)-c3)^2}{\sigma^3}] \\ -wb[x(t), x(t-\tau), x(t-2\tau)][\frac{(x(t)-c1)}{\sigma^2}] \\ -wb[x(t), x(t-\tau), x(t-2\tau)][\frac{(x(t-\tau)-c2)}{\sigma^2}] \\ -wb[x(t), x(t-\tau), x(t-2\tau)][\frac{(x(t-2\tau)-c3)}{\sigma^2}] \end{bmatrix} \tag{7.13}$$

- **e**: Residual vector, formed by 1 column and N rows (number of elements of each RBF), i is the actual iteration.

$$e = \begin{bmatrix} f_1 - wb[x(t_1), x(t_1-\tau), x(t_1-2\tau)] \\ f_2 - wb[x(t_2), x(t_2-\tau), x(t_2-2\tau)] \\ \vdots \\ f_N - wb[x(t_N), x(t_N-\tau), x(t_N-2\tau)] \end{bmatrix} \tag{7.14}$$

Finally we obtain:

$$\frac{\partial E}{\partial \theta} = -\begin{bmatrix} \frac{\partial Y_p^1}{\partial \theta_1} e^1 + & \cdots & + \frac{\partial Y_p^N}{\partial \theta_1} e^N \\ \vdots & \vdots & \vdots \\ \frac{\partial Y_p^1}{\partial \theta_q} e^1 + & \cdots & + \frac{\partial Y_p^N}{\partial \theta_q} e^N \end{bmatrix} = -Z^T \cdot e = -\begin{bmatrix} \frac{\partial Y_p^1}{\partial \theta_1} & \cdots & \frac{\partial Y_p^N}{\partial \theta_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial Y_p^1}{\partial \theta_q} & \cdots & \frac{\partial Y_p^N}{\partial \theta_q} \end{bmatrix} \begin{bmatrix} f_i - Y_p(\theta, x_1, y_1, z_1) \\ \vdots \\ f_N - Y_p(\theta, x_N, y_N, z_N) \end{bmatrix} \tag{7.15}$$

where Z is a Jacobian matrix: Nxq, N=N_{total}

Until that moment, we defined all the elements needed to construct $\frac{\partial E}{\partial \theta}$, that as we are going to see, is the key element of the Simple Gradient Method:

$$\theta(i+1) = \theta(i) - step \cdot \frac{\partial E}{\partial \theta}. \tag{7.16}$$

In the formula above, in every iteration we are going to obtain new parameters of a RBF. In every new iteration, that parameters should be better tuned.
The variable step, is an important parameter of the method, depending on the value of it, we can obtain a really good or bad optimization. That parameter means which is the jump in the direction of the minus gradient, that we do in every iteration.

In the following lines, we are going to detail how works this method, supposing a step of 0.01:

$$\begin{bmatrix} w(i+1) \\ \\ \sigma(i+1) \\ \\ c1(i+1) \\ \\ c2(i+1) \\ \\ c3(i+1) \end{bmatrix} = \begin{bmatrix} w(i) \\ \\ \sigma(i) \\ \\ c1(i) \\ \\ c2(i) \\ \\ c3(i) \end{bmatrix} - \begin{bmatrix} 0.01 \cdot w(0) \\ \\ 0.01 \cdot \sigma(0) \\ \\ 0.01 \cdot c1(0) \\ \\ 0.01 \cdot c2(0) \\ \\ 0.01 \cdot c3(0) \end{bmatrix} \frac{\partial E}{\partial \theta} = \begin{bmatrix} w(i) \\ \\ \sigma(i) \\ \\ c1(i) \\ \\ c2(i) \\ \\ c3(i) \end{bmatrix} - \begin{bmatrix} 0.01 \cdot w(0) \\ \\ 0.01 \cdot \sigma(0) \\ \\ 0.01 \cdot c1(0) \\ \\ 0.01 \cdot c2(0) \\ \\ 0.01 \cdot c3(0) \end{bmatrix} \begin{bmatrix} \frac{\partial Y_p^1}{\partial \theta_1} e^1 + & \cdots & + \frac{\partial Y_p^N}{\partial \theta_1} e^N \\ \vdots & \vdots & \vdots \\ \frac{\partial Y_p^1}{\partial \theta_q} e^1 + & \cdots & + \frac{\partial Y_p^N}{\partial \theta_q} e^N \end{bmatrix} \tag{7.17}$$

7.3.4 Levenberg Marquardt Method

Algorithm able to provide a numerical solution to the problem of minimizing generally a nonlinear function.

Like Simple Gradient Method, Levenberg-Marquardt is an Iterative method that is able to optimize the arguments of our function, using the Gradient of it, and certain initial conditions.
We can consider this method also as a similar method than Gauss-Newton Algorithm.

Despite being similar algorithms, have relevant differences between them:

- Levenberg-Marquardt is considered more robust than Gauss-Newton, because the first one is able to find a solution even if the start point is far from the optimal arguments.
- When the initial arguments are close to the optimal arguments, and our function is a well-behaved function, Gauss-Newton find a solution quicker than Levenberg-Marquardt.

In the next lines, we present the formula that called iteratively let us optimize the parameters of a non-linear function:

$$\theta(i+1) = \theta(i) + [Z(i)^T \cdot Z(i) + \lambda I_{qxq}]^{-1} \cdot Z(i)^T \cdot e(i). \tag{7.18}$$

where:

λ is: 0.1

Z is the gradient Matrix, presented in the Simple Gradient section

e is the residual vector, presented in the Simple Gradient section

and I is an Identity Matrix of size 5:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.19}$$

θ is the vector of arguments. From this initial arguments depends a good optimization. In many cases, and standard guess will work fine, but in other cases, the algorithm can converge only if the initial arguments are close to the final solution.

Chapter 8

Results and discussion

In this chapter we are going to present and describe the results obtained using the different Optimization Methods. This chapter is strongly related with the Implementation Chapter, to which we will make reference in different occasions. The results will be presented classified by, the different optimizations methods and inside each one of these methods, we will do different tests with different number of iterations of the Chua's Circuit equations.

8.1 Chua's Circuit Test with 11000 Iterations

In this first test we will run Chua's Circuit and the OFR modeling with the following parameters:

- **Number of Iterations:** 11000
- **Tau (τ):** 250
- **Start Iteration:** 600
- **Number of RBFs:** 160, Scale 3

In the following sections, we will present the results obtained applying different optimization methods, to the obtained model of Chua's Circuit, built with the parameters previously defined.

8.1.1 Least Squares Method Temporal Approach

Using this method we obtained the next degrees of error:

Number of RBfs	Error	Error Ratio
4	$4.351012e^{-1}$	29.918675
6	$3.856483e^{-1}$	26.518168
10	$3.387364e^{-1}$	23.292388
16	$3.237243e^{-1}$	22.260113

Table 8.1: *Results obtained applying Temporal LS and 11000 Iterations*

As we can see in the table above, when the number of RBF's is bigger than 10, the obtained Error is almost the same. No gain in increasing the number of RBF's.

In the following table, we are going to present all the parameters of the RBF's that we selected. In that case we used 6 approximations and we set the dataset generation to Scale 3:

Approx.	Number of RBF	Center 1	Center 2	Center 3	Sigma	Weight
1	51	-2.236	0.012	-2.236	1.122	-2.937
2	144	2.253	0.0042	0.012	1.122	3.089
3	45	-2.236	0.004	2.253	1.122	-2.36
4	136	2.253	-2.236	-2.236	1.122	1.973
5	46	-2.236	2.253	-2.236	1.122	-3.006
6	140	2.253	-2.236	2.253	1.122	4.324

Table 8.2: *Selected RBF's with 6 approximations, applying Temporal LS and 11000 Iterations*

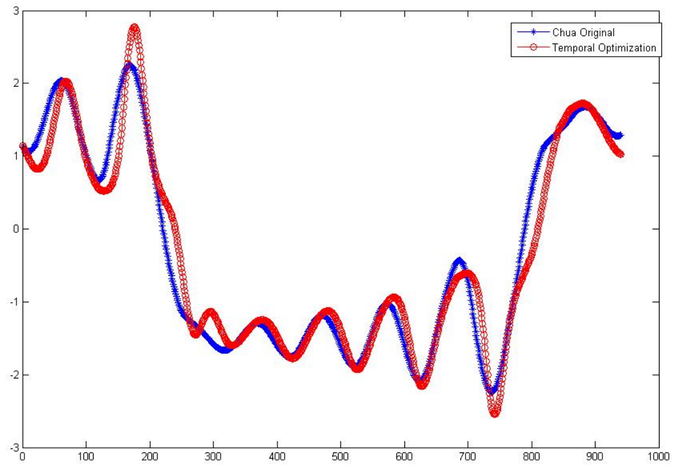

Figure 8.1: *Results of the Temporal Approach in 2D and 10 approximations*

In these images we can observe the graphical results of the model applying Temporal Least Squares Optimization.

The images show the results in 2D 8.1 and 3D 8.2. In blue color it's drawn the original Chua's Circuit output, and in red color appear the model built. As we can observe in the 3D image, appear the attractor reconstructed in red line. As we can see graphically, there's big difference between the 2 attractors (around 22 %).

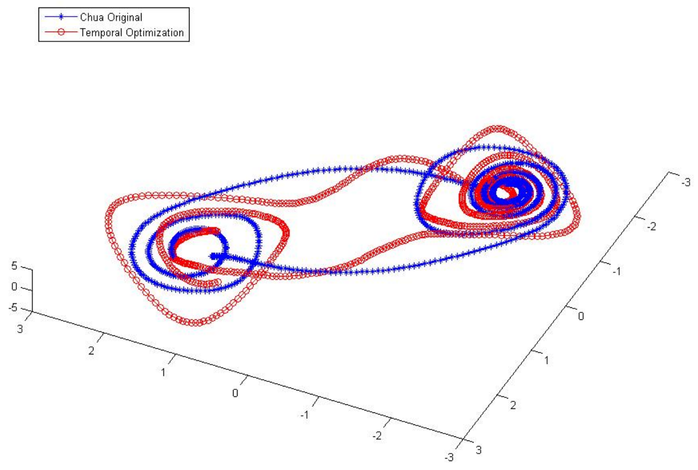

Figure 8.2: *Results of the Temporal Approach in 3D and 10 approximations*

8.1.2 Least Squares Method Vectorial Approach

Using this method we obtained the next degrees of error:

Number of RBfs	Error	Error Ratio
4	$4.207566e^{-1}$	28.932310
6	$3.138084e^{-1}$	21.578275
10	$2.974459e^{-1}$	20.453143
16	$1.891788e^{-1}$	13.008423

Table 8.3: *Results obtained applying Vectorial LS and 11000 Iterations*

As we can see in the table above, and comparing this results with the one's obtained applying Temporal LS optimization 8.1, we observe that this method works much better. With 16 approximations, applying this method we obtained a 13% of error, while applying Temporal LS we obtained 22% of error.

This is due to the way the RBF's are optimized. Applying vectorial LS, the RBF's are tuned all in once, as 1 unique function.
In the other hand, when optimizing the RBF's applying temporal LS, the RBF's are optimized one by one, optimizing each RBF with respect to the residual part of our goal function. Logically, when we sum all the RBF's optimized using Temporal LS, the result we obtained is not as good as applying Vectorial LS. This is due to the fact, that if we tune the functions globally, this tuning will be much more effective than treating the functions as individual one's, because in fact, afterwards we will treat them as a global function.

In the following table, we are going to present all the parameters of the RBF's that we selected. In that case, we used 6 approximations and we set the dataset generation to Scale 3:

Approx.	Number of RBF	Center 1	Center 2	Center 3	Sigma	Weight
1	51	-2.236	0.012	-2.236	1.122	-2.620
2	144	2.253	0.0042	0.012	1.122	1.538
3	45	-2.236	0.004	2.253	1.122	-3.072
4	136	2.253	-2.236	-2.236	1.122	3.674
5	46	-2.236	2.253	-2.236	1.122	-6.35
6	140	2.253	-2.236	2.253	1.122	15.741

Table 8.4: *Selected RBF obtained applying Vectorial LS and 11000 Iterations*

We observed a problem of rank deficiency applying this method with more than 16 approximations.
As we described in the Implementation section, we found another method 7.8 able to extend the number of approximations up to 25. In the following table we are going to present the results derive from it:

Number of RBfs	Error	Error Ratio
17	$1.888236e^{-1}$	12.983999
20	$1.685432e^{-1}$	11.589462
23	$1.413846e^{-1}$	9.721966
25	$1.402513e^{-1}$	9.644037

Table 8.5: *Results obtained applying Vectorial LS using Moore-Penrose pseudoinversion and 11000 Iterations*

As we can observe in the table above, applying Moore-Penrose pseudoinversion, we reduce our Error Ratio significatively, without increasing drastically the computation time.

Comparing this results, with the one's obtained without applying the Moore-Penrose pseudoinversion, we reduce our ratio significatively, from the 13% with 16 approximations to 9% with 25 approximations.

In the following images, we present the results obtained graphically:

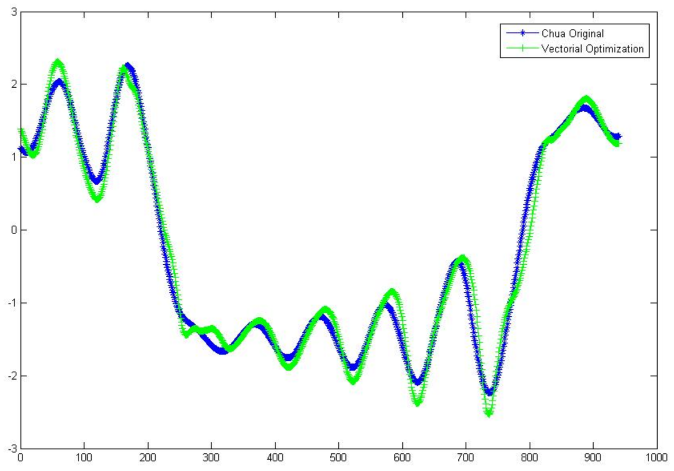

Figure 8.3: *Results of the Vectorial Approach in 2D and 11000 Iterations*

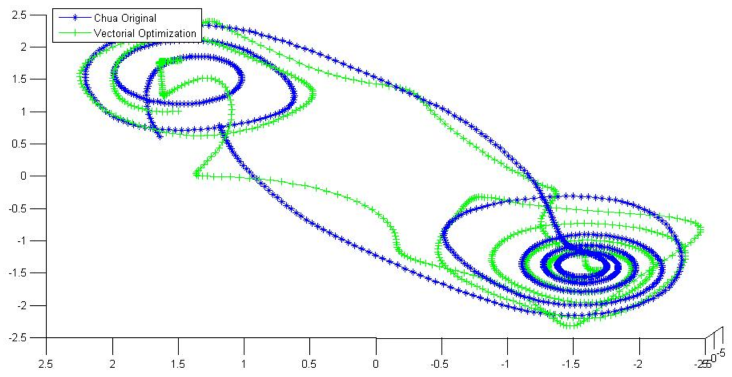

Figure 8.4: *Results of the Vectorial Approach in 3D and 11000 Iterations*

In the images above we can observe the graphical results of the model applying Vectorial Least Squares Optimization.

The images show the results in 2D and 3D with 10 approximations. In blue color the original Chua's Circuit output, and in green color the model built, optimized with the Vectorial Approach.
As we can observe in the 3D image, appear the attractor reconstructed with a green line. There's still an important difference between the 2 attractors, basically in the transition between the attractors.

8.2 Chua's Circuit Test with 5500 Iterations

In that test we will run Chua's Circuit and the OFR modeling with the following parameters:

- **Number of Iterations:** 5500
- **Tau (τ):** 125
- **Start Iteration:** 300
- **Number of RBFs:** 160, Scale 3

In the following sections, we will present the results obtained applying different optimization methods, to the obtained model of Chua's Circuit, built with the parameters previously defined.

8.2.1 Least Squares Method Temporal Approach

Using this method, we obtained the next degrees of error:

Number of RBfs	Error	Error Ratio
4	$2.323771e^{-1}$	15.816313
6	$2.134140e^{-1}$	14.525628
10	$1.891100e^{-1}$	12.871420
16	$1.759247e^{-1}$	11.973985

Table 8.6: *Results obtained applying Temporal LS*

As we can see in the table above, and comparing the results obtained with the previous one's 8.1 obtained with 11000 Iterations, we can observe that the error is much lower in that case. This, is due to the fact that the goal function is simpler in that case, and the models of it are better.

In the following table, we are going to present all the parameters of the RBF's that we selected. In that case we used 6 approximations and we set the dataset generation to Scale 3:

Approx.	Number of RBF	Center 1	Center 2	Center 3	Sigma	Weight
1	157	2.253	2.253	0.126	0.970	3.138
2	51	-1.745	0.380	-1.745	0.999	-3.723
3	55	-1.745	0.380	2.253	0.999	-0.999
4	137	2.253	-1.745	0.126	0.999	2.666
5	60	-1.745	2.253	2.253	0.999	-1.159
6	140	2.253	-1.745	2.253	0.999	1.123

Table 8.7: *Selected RBF's with 4 approximations, applying Temporal LS, and 5500 iterations*

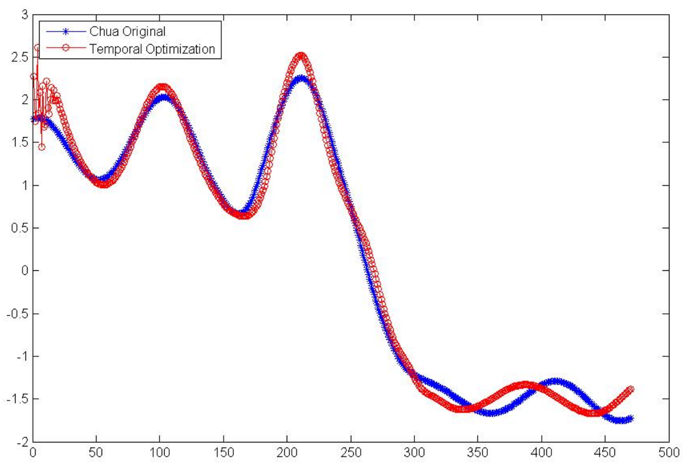

Figure 8.5: *Results of the Temporal Approach in 2D, 5500 Iterations, and 10 approximations*

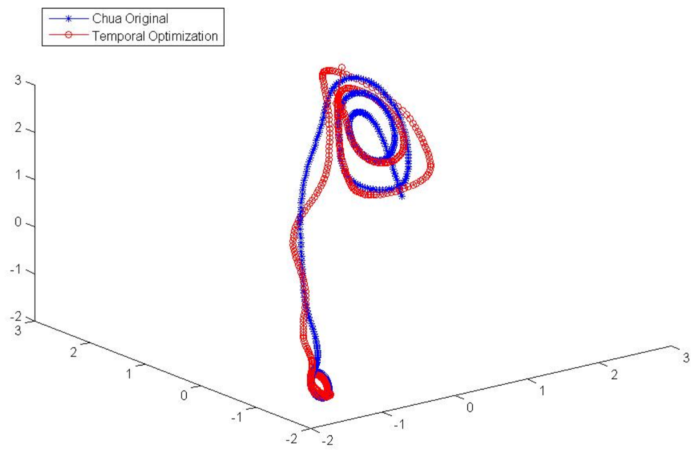

Figure 8.6: *Results of the Temporal Approach in 3D, 5500 Iterations, and 10 approximations*

In the images above, we can observe the graphical results of the model, applying Temporal Least Squares Optimization and 25 approximations.

The original Chua's Circuit output, appear in blue, and in red color appear the model built. In the 3D Image 8.6 we can observe that the model is quite close to the goal function, with just a 12% of error.

8.2.2 Least Squares Method Vectorial Approach

Using this method we obtained the next degrees of error:

Number of RBfs	Error	Error Ratio
4	$1.989610e^{-1}$	13.541910
6	$1.766855e^{-1}$	12.025768
10	$9.368872e^{-2}$	6.376748
16	$7.468140e^{-2}$	5.083050
20	$6.706691e^{-2}$	4.564784
23	$6.523012e^{-2}$	4.439766
25	$4.301469e^{-2}$	2.927715

Table 8.8: *Results obtained applying Vectorial LS*

As we can see in the table above, and comparing these results with the one's obtained applying Vectorial LS optimization with 11000 Iterations 8.5, we observe that this results are better (smaller error), due to the fact, previously explained in 8.6, that the goal function generated with 5500 iterations is simpler. For that reason, the model obtained is better (closer to the goal function).

We also need to note that in that occasion we applied directly Moore-Penrose pseudoinversion 7.8 being able to obtain models with up to 25 approximations.

The results obtained using 25 approximations are really good in that case. We consider this, because we obtained an error around 3%, which is something really positive.

In the following table, we are going to present all the parameters of the RBF's that we selected. In that case, we used 6 approximations and we set the dataset generation to Scale 3:

Approx.	Number of RBF	Center 1	Center 2	Center 3	Sigma	Weight
1	157	2.253	2.253	0.126	0.999	2.571
2	51	-1.745	0.380	-1.745	0.999	-3.576
3	55	-1.745	0.380	2.253	0.999	-0.923
4	137	2.253	-1.745	0.126	0.999	-1.503
5	60	-1.745	2.253	2.253	0.999	-4.588
6	140	2.253	-1.745	2.253	0.999	13.616

Table 8.9: *Selected RBF obtained applying Vectorial LS*

In the following images, we present the results obtained graphically:

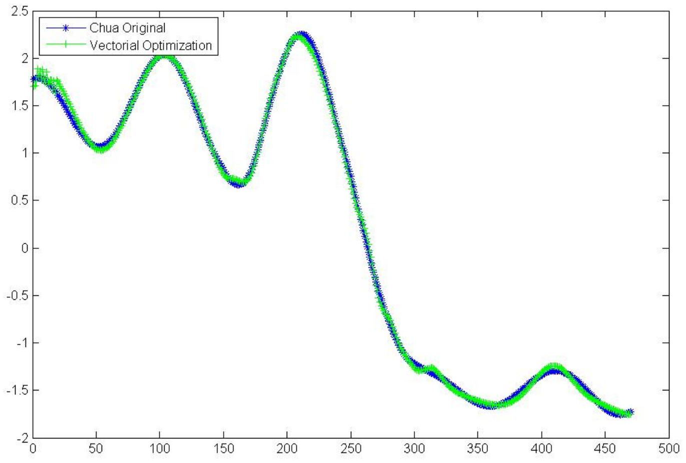

Figure 8.7: *Results of the Vectorial Approach in 2D and 11000 Iterations*

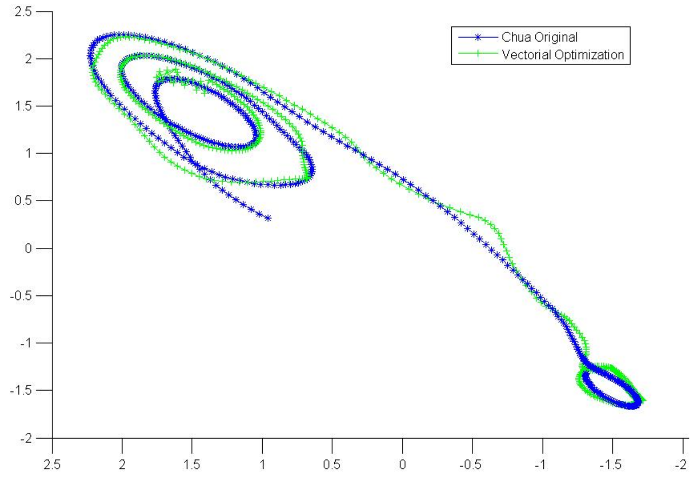

Figure 8.8: *Results of the Vectorial Approach in 3D and 11000 Iterations*

In the images above we can observe the graphical results of the model applying Vectorial Least Squares Optimization.

The images show the results in 2D and 3D with 25 approximations and 5500 Iterations. In blue color the original Chua's Circuit output, and in green color the model built, optimized with the Vectorial Approach.

As we can observe in the 3D image, the results obtained are really good, and the attractor is almost completely well modeled.

Chapter 9

Conclusions and Further Work

In this final chapter we are going to review all the previous chapters starting for the study of Statistical Learning Theory and ending in the chapter of the results. Finally, we are going to comment the results obtained, and to describe the further work related with this project.

9.1 Summary

This thesis started analyzing what is it Statistical Learning, and defining the aim of classification, that as we have seen is considered a Supervised Learning Technique. Classification is considered a Supervised Learning Technique because we know the class to which one object belongs to.
Afterwards we defined what is it Function Approximation, and we studied different techniques such as Regression, Interpolation, Extrapolation etc.
Finally, in this chapter, we studied in detail different well-known classifiers and regressors such as Neural Networks, Radial Basis Function Networks, Support Vector Machines etc.

In the third chapter of our thesis, we studied in detail the algorithm that we used to model our goal function, called Orthogonal Forward Regression. We have seen in detail how the algorithm works, how we can adapt it, and finally how we can implement it easily.
As we have seen, OFR algorithm has been widely use and readapted to fit in many different kinds of applications. We studied two of these applications in detail, the Regression of an ECG, and the model of Etna volcanomagnetic signal.

In the fourth chapter, we focused our attention in defining and studying Non-Linear Dynamical Systems. As we have seen, these systems are complex, but widely used in several applications of several different fields. First of all in this chapter, we defined the conditions that a Non-Linear Dynamical System need to accomplish to be considered as a chaotic system. We also studied in detail the different kinds of attractors, the butterfly effect, the Lyapunov exponent etc. Other important aspects that have been studied in this chapter were Lorenz Attractor, what is it and how we can apply Embedding to our systems and finally we studied the most important Practical Learning Methods such as AR, MA, ARMA, NARX etc.

In the fifth Chapter we defined and studied in detail the Chua's Circuit, that was the system that we chose as a test for our experiments. We chose Chua's Circuit as the reference for our experiments, because it is a well-known system for all the scientific community, and has lots of references and applications that were really useful while trying to understand its complex behavior.

In the sixth chapter, we specified the requirements of our software. Detailing which functionalities needed to have, and how we should implement each part. This chapter was accompanied of different diagrams that helped us to implement the software, and would help anybody else that would like to improve it.

In the seventh chapter, we studied all the details related with the implementation of the software that we created. One of the main aspects of this chapter was related in detailing how we created our dataset, specifying aspects such as which are the functions of our library, how we tuned the functions of our library etc. Finally we focused on detailing how we implemented the different optimizations methods, explaining each one of the formulas and techniques used.

Finally in the eight chapter, we presented and discussed the results obtained running our algorithm several times with different configurations. Our first test consisted on creating a dataset with the first 11000 Iterations of Chua's Circuit. Analyzing the results obtained with the different optimizers, we observed that the results weren't completely good, due to the complexity of the function created.
We run another test with the aim of creating an easier function to model. For that purpose, we created a dataset with the first 5500 Iterations of Chua's Circuit, and we analyze again the results obtained applying the different optimizers, concluding that the results obtained are much better in this case due to that the function is simpler and easier to model.

Finally, once we finished this thesis and analyzed in detail all the results obtained, we consider that the results obtained are really satisfactory, with errors around the 10% with 11000 iterations and the 3% with 5500 iterations. Even though we considered different further works to improve some aspects of our software such as testing other optimization methods, testing other chaotic systems etc.

As a personal opinion, we are really proud of what we got, and how we got it. We can't forget that this project started in Politechnika Warszawska (Poland) with the supervision of Stanislaw Jankowski and ended some months later in Barcelona with the supervision of Xavier Vilasís. We are proud of ourselves because we entered in a project quite complex and without any previous idea of Non-Linear Dynamical Systems neither Orthogonal Forward Regression, and we consider we got some interesting results. Thanks to help of Stanislaw and Xavi I couldn't get it without you.

9.2 Further Work

Several open issues had been identified with the conclusion of this thesis. These issues are:

1. Test several optimization methods.
2. Test various chaotic systems.

3. Generate other functions for the OFR library.

Test several optimization methods. In this thesis, several optimization methods has been tested such as Least Squares (Vectorial and Temporal approaches), Simple Gradient Method etc. One interesting line of research could be, to try other optimizations methods, such as Newton Method, quasi-Newton method etc.
Once applied these other optimizations methods, would be interesting to compare the results obtained with all the optimizations methods, comparing the error, but also the computational cost of each one of this methods. Finally with all these results, we could choose the best optimization method for our problem.

Test various Non-linear Dynamical Systems. In the thesis, two Non-linear Dynamical Systems has been studied such as Chua's Circuit and Lorenz Oscillator. But, the software has been tried only with Chua's Circuit. For that reason, we consider that would be interesting to try the software with another chaotic systems.
This may cause some problems. The software that has been designed, works well with Chua's Circuit, but we don't know exactly how would work with other systems, such as Lorenz Oscillator, and we should consider the following, to use other optimization methods, other functions in the OFR library etc.

Generate other functions for the OFR library. Radial Basis Functions (RBF's) has been used, as the functions that OFR will use to do the regression process of our goal function. RBF's were really useful for the regression of Chua's Circuit, but we should consider trying other functions with different characteristics.
Other families of parameterized functions that we could try are Wavelets, Polynomials, Neural Networks etc. The choice between those families is based on different criteria such as the implementation complexity, domain knowledge etc.

9.3 Cost of the Project

Table 9.1 shows the total cost of the project. It is clearly visible that the two main phases were documentation and design.

Phase	Hours	Percentage
Documentation	400	44.44%
Design	100	11.11%
Implementation	100	11.11%
Writing Report	150	16.67%
Tests	150	16.67%
Total	900	100%

Table 9.1: *Total cost of the project.*

Bibliography

E.J. Altman. Normal form analysis of chua's circuit with applications for trajectory recognition, 1993.

A G Bors. Introduction of the radial basis function (rbf) networks, 2001.

Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to statistical learning theory. In *In , O. Bousquet, U.v. Luxburg, and G. Rsch (Editors*, pages 169–207. Springer, 2004.

S. Chen, C.F.N. Cowan, and P.M. Grant. Orthogonal least square learning algorithm for radial basis function networks, 1991.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995.

Rémi Dubois, Pierre Maison-Blanche, Brigitte Quenet, and Gérard Dreyfus. Automatic ecg wave extraction in long-term recordings using gaussian mesa function models and nonlinear probability estimators. *Comput. Methods Prog. Biomed.*, 88(3):217–233, 2007. ISSN 0169-2607. doi: http://dx.doi.org/10.1016/j.cmpb.2007.09.005.

E. Iee. Ieee recommended practice for software requirements specifications. Technical report, IEE, 1998.

S. Jankowski, G. Currenti, R. Napoli, Z. Szymanski, L. Fortuna, C. Del Negro, and Istituto Nazionale Di Geofisica. Modeling volcanomagnetic dynamics by recurrent least-squares svm, 2009.

S. Jankowski, G. Currenti, R. Napoli, Z. Szymanski, L. Fortuna, C. Del Negro, and R. Dubois. Modeling of volcanomagnetic dynamics by recurrent orthogonal least-squares learning system, 2010.

T. Matsumoto. A chaotic attractor from chua circuit. *IEEE Trans. on Circuits and Systems*, pages 1055–1058, 1984.

T. Matsumoto, L. Chua, and M. Komuro. The double scroll. *IEEE Trans. on Circuits and Systems*, 32:797–818, 1985.

T. Mitchell. *Machine Learning.* Prentice Hall, Pittsburgh, 1997. ISBN 0070428077.

A. Orriols-Puig. Introduction to machine learning, 2009.

H.O. Wang and K. Tanaka. An lmi-based stable fuzzy control of nonlinear systems and its application to control of chaos, 1996.

B. Widrow and M. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415 –1442, sep 1990. ISSN 0018-9219. doi: 10.1109/5.58323.

G.Q. Zhong, K.T. Ko, and K.F. Man. Robustness of synchronization in coupled chua circuits. *IEEE International Symposium on Industrial Electronics*, 2:436–440, 1998.