Master's thesis

IDENTIFICATION OF NONLINEAR SYSTEMS BASED ON EXTREME LEARNING MACHINE AND MULTILAYER NEURAL NETWORKS

Emerson Grzeidak

Brasília
2016, May

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

Master's thesis

IDENTIFICATION OF NONLINEAR SYSTEMS BASED ON EXTREME LEARNING MACHINE AND MULTILAYER NEURAL NETWORKS

Emerson Grzeidak

Report submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Master in Mechatronic Systems

Examination board

Prof. José Alfredo Ruiz Vargas, ENE/UnB _____
Advisor

Prof. Carlos Humberto Llanos Quintero, ENM/UnB _____
Chair member

Prof. Bismark Claure Torrico, DEE/UFC _____
Chair member

FICHA CATALOGRÁFICA

GRZEIDAK, EMERSON

Identification of Nonlinear Systems based on Extreme Learning Machine and Multilayer Neural Networks

[Distrito Federal] 2016.

x, 152p, 210 x 297 mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2016).

Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Mecânica

1. Identificação Online	2. Redes Neurais
3. Métodos de Lyapunov	4. Aprendizado Extremo
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

GRZEIDAK, E. (2016). *Identification of Nonlinear Systems based on Extreme Learning Machine and Multilayer Neural Networks*, Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-101/2016, Departamento de Engenharia Mecânica, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 152p.

CESSÃO DE DIREITOS

AUTOR: Emerson Grzeidak.

TÍTULO: *Identification of Nonlinear Systems based on Extreme Learning Machine and Multilayer Neural Networks*.

GRAU: Mestre ANO: 2016

_____

Emerson Grzeidak
Departamento de Eng. Mecânica (ENM) – FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF – Brasil.

Dedication

To Thaís Cristina Cohen Grzeidak.

Emerson Grzeidak

"*I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the seashore, diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, while the great ocean of truth lay all undiscovered before me.*"

Isaac Newton

Acknowledgements

I am deeply grateful to my supervisor Prof. José Alfredo Ruiz Vargas for his friendly advice, constructive criticism and invaluable help all throughout the project. His devotion and enthusiasm to the study of control systems ignited my interest in the Master's thesis research topics.

I am also indebted to my master advisory committee members and college representatives for their careful evaluation of my Master's thesis and providing valuable corrections and insightful comments. I would like to express my gratitude to the University of Brasília (UnB) and the Department of Mechanical Engineering (ENM) for providing such a great learning and friendly atmosphere.

Finally, I would also like to thank my mother for her unwavering support and my father for instilling me curiosity and passion for life. I must acknowledge my best friend and love Thaís Cristina Cohen Grzeidak for all the support, encouragement and love she has given me. My days are complete with you.

Emerson Grzeidak

ABSTRACT

The present research work considers the identification problem of nonlinear systems with uncertain structure and in the presence of bounded disturbances. Given the uncertain structure of the system, the state estimation is based on single-hidden layer neural networks and then, to ensure the convergence of the state estimation residual errors to zero, the learning laws are designed using the Lyapunov stability theory and already available results in adaptive control theory. First, an identification scheme via extreme learning machine neural network is developed. The proposed model ensures the convergence of the state estimation residual errors to zero and boundedness of all associated approximation errors, even in the presence of approximation error and disturbances. Lyapunov-like analysis using Barbalat's Lemma and a dynamic single-hidden layer neural network (SHLNN) model with hidden nodes randomly generated to establish the aforementioned properties are employed. Hence, faster convergence and better computational efficiency than conventional SHLNNs is assured. Furthermore, with a few modifications regarding the selection of activation function and the regressor vector's structure, the proposed algorithm can be applied to any linearly parameterized neural network model.

Next, as an extension of the proposed methodology, a nonlinearly parameterized single-hidden layer neural network model (SHLNN) is studied. The hidden and output weights are simultaneously adjusted by robust adaptive laws that are designed via Lyapunov stability theory. The second scheme also ensures the convergence of the state estimation residual errors to zero and boundedness of all associated approximation errors, even in the presence of approximation error and disturbances. Additionally, as in the first scheme, it is not necessary any previous knowledge about the ideal weights, approximation error and disturbances. Extensive simulations to validate the theoretical results and show the effectiveness of the two proposed methods are also provided.

IDENTIFICAÇÃO DE SISTEMAS NÃO LINEARES BASEADO EM APRENDIZADO EXTREMO E REDES NEURAIS MULTICAMADAS

RESUMO ESTENDIDO

O presente trabalho considera o problema de identificação de sistemas não-lineares com estrutura incerta na presença de distúrbios limitados.Dado a estrutura incerta do sistema, a estimação dos estados é baseada em redes neurais com uma camada escondida e então, para assegurar a convergência dos erros residuais de estimação dos estados para zero, as leis de aprendizagem são projetadas usando a teoria de estabilidade de Lyapunov e resultados já disponíveis na teoria de controle adaptativo. Primeiramente, um esquema de identificação usando aprendizagem extrema é apresentado. O modelo proposto assegura a convergência dos erros residuais de estimação dos estados para zero e a limitação de todos os demais erros e distúrbios. Usando o lema de Barbalat e uma análise tipo Lyapunov, é empregado um modelo de rede neural dinâmica com uma camada escondida (SHLNN) gerada aleatoriamente para assegurar as propriedades supramencionadas. Dessa maneira, assegura-se uma convergência mais rápida e melhor eficiência computacional do que os modelos SHLNN convencionais. Além disso, com algumas modificações que envolvem a seleção da função ativação e a estrutura do vetor regressor, o algoritmo proposto pode ser aplicado para qualquer rede neural parametrizável linearmente.

Em seguida, como uma extensão da metodologia proposta, um modelo de rede neural com uma camada escondida e parametrizável não-linearmente (SHLNN) é estudado. Os pesos da camada escondida e de saída são ajustados simultaneamente por leis adaptativas robustas obtidas através da teoria de estabilidade de Lyapunov. O segundo esquema também assegura a convergência dos erros residuais de estimação dos estados para zero e a limitação de todos os demais erros de aproximação associados, mesmo na presença de erros de aproximação e distúrbios. Adicionalmente, como no primeiro esquema, não é necessário conhecimento prévio sobre os pesos ideais, erros de aproximação ou distúrbios. Simulações extensivas para a validação dos resultados teóricos e demonstração dos métodos propostos são fornecidos.

A dissertação está organizada da seguinte forma. Capítulo 1 apresenta a introdução e motivação da pesquisa proposta e preliminares matemáticas necessárias para a compreensão da análise de estabilidade de Lyapunov. O capítulo 2 fornece uma breve descrição do desenvolvimento histórico da modelagem e identificação de sistemas assim como é apresentado uma revisão do estado da arte dos métodos de identificação baseado em redes neurais com uma camada escondida. A fundamentação teórica das redes neurais, suas propriedades, diferentes topologias e algoritmos de aprendizado capítulo são descritas no 3 assim como a notação que será utilizada nos capítulos seguintes.

No capítulo 4, usando aprendizado extremo, propõe-se um novo esquema de identificação neural adaptativo online para uma classe de sistemas não lineares na presença de dinâmica desconhecida e distúrbios limitados. É de salientar que, além da hipótese de limitação, nenhum conhecimento prévio sobre a dinâmica do erro de aproximação, pesos ideais ou perturbações externas é necessário. Aprendizado extremo é uma classe de redes neurais com uma camada escondida onde os pesos da

camada escondida são gerados de forma aleatória de acordo com qualquer distribuição de probabilidade contínua, e adicionalmente nesta dissertação os pesos da camada de saída são atualizados de acordo com uma lei adaptativa estável derivada da análise de Lyapunov. A análise baseada na teoria de estabilidade de Lyapunov prova que o algoritmo de aprendizado adaptativo converge assintoticamente na estimação de sistemas não lineares. A metodologia proposta combina eficiência computacional em termos de velocidade de convergência do algoritmo de aprendizado extremo com a estabilidade do sistema sob distúrbios garantida pela análise de Lyapunov.

O resultado das simulações para um sistema caótico unificado e um sistema hipercaótico financeiro demonstram a eficácia e desempenho da abordagem proposta na presença de distúrbios. Adicionalmente, para mostrar a eficiência do algoritmo de aprendizado proposto para sistemas de várias dimensões, uma simulação para um sistema hipercaótico complexo é demonstrada sem comprometer a velocidade e a qualidade da convergência. Finalmente, uma comparação do algoritmo proposto com [1] é exibida para mostrar as vantagens e peculiaridades do método proposto na presença de distúrbios. Os erros de estimação dos estados mostram melhor convergência na presença de perturbações externas e evita a deriva dos parâmetros, assim como a norma dos pesos mostra valores quase constantes.

Posteriormente, no capítulo 5, os resultados obtidos no capítulo anterior são estendidos para redes neurais com uma camada escondida. O esquema é baseado na topologia de uma rede neural com uma camada escondida para a parametrização das não linearidades desconhecidas, onde a camada escondida e de saída são ajustadas simultaneamente por leis adaptativas projetadas com base na teoria de estabilidade de Lyapunov. Condições necessárias são estabelecidas para assegurar a convergência dos erros residuais de estimação dos estados para zero e todos os erros associados são limitados, mesmo na presença de erros de aproximação e distúrbios desconhecidos limitados.

O resultado das simulações para o sistema caótico unificado mostram a eficácia e o desempenho da abordagem proposta na presença de distúrbios. Adicionalmente, para mostrar a aplicabilidade do algoritmo de aprendizado proposto para sistemas com várias dimensões, uma simulação com um sistema hipercaótico complexo é exibida. Finalmente, uma comparação do algoritmo proposto com [2] é realizada para mostrar as vantagens e peculiaridades do método proposto na presença de distúrbios. Capítulo 6 resume as contribuições teóricas da pesquisa bem como os resultados obtidos. Sugestões para pesquisa futura também são discutidas. Os apêndices contém a implementação via software dos identificadores neurais propostos nesta dissertação.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$V(x,t), \bar{V}$	: Lyapunov function candidate
W	: output layer weight matrix for the SHLNN
V	: hidden layer weight matrix for the SHLNN V
V_R	: hidden layer weight matrix with random values for the ELM V
W^*	: matrix of optimal weights for W
V^*	: matrix of optimal weights for V
$\hat{W}$	: estimation of ideal weight matrix W^*
$\hat{V}$	: estimation of ideal weight matrix V^*
x	: n-dimensional state vector
$\hat{x}$	: estimation of the n-dimensional state vector
$\tilde{x}$	: estimation error of the n-dimensional state vector
u	: m-dimensional admissible input vector
z	: regressor vector, where $z = [x_1, \ldots, x_n, u_1, \ldots, u_m]$
$\sigma(.)$	: activation function

Acronyms

NN	: Neural Network
MLP	: Multilayer Perceptron
SLFN	: Unified Single-hidden Layer Feedforward Neural network
SHLNN	: Single-Hidden Layer Neural Network
RBF	: Radial Basis Function
WNN	: Wavelet Neural Network
FBF	: Fuzzy Basis Network
HONN	: Higher-Order Neural Network
RHONN	: Recurrent Higher-Order Neural Network
ELM	: Extreme Learning Machine
SOM	: Self-Organizing Maps
TD	: Temporal Difference
LMS	: Least Mean Square
BP	: Backpropagation
MSE	: Mean Squared Error

Chapter 1

Introduction

1.1 Motivation of the Thesis

Nonlinearity is a widespread phenomenon in nature. From fields such as chaos theory, thermodynamics, fluid mechanics, space engineering, ecology, photonics and robotics, phenomenons driven by nonlinear equations are the rule rather than the exception. However, in many industrial and engineering applications that exhibit nonlinear behavior, conventional linear models based on approximate linearization for system identification and control has been used. Furthermore, the use of linear models can result in a poor control performance and impose considerable restrictions for many nonlinear plants. Also, the presence of nonlinearities in control systems may difficult the design stages and in many practical situations be infeasible to obtain an accurate mathematical model due to lack of knowledge of some parameters or even the structure of the system. Thus, the modelling of nonlinear dynamical systems received considerable attention in the recent years, as it is an important step toward controller design of nonlinear systems in many situations. Research over past decades has produced several nonlinear control strategies based on mathematical foundations and there is an increasing demand for developing more effective nonlinear system identification methods. As a consequence, the research area of nonlinear system identification is intrinsically diversified and highly active [6].

From [7], system identification can be defined as the process of obtaining mathematical models of systems using input-output behavior. Thus, the subject of system identification is concerned with techniques and methods for studying a process or system through observed data, mainly for developing a suitable mathematical description of that system. Additionally making it of paramount importance in prediction, control, monitoring, design and innovation of systems and process. Two contrasting approaches are generally followed for model development: a theoretical approach that is based on methods derived from calculus, and an experimental approach that is based on analysis of experimental observations or measured data.

For the theoretical approach, in most cases, simplifying assumptions regarding the system are usually necessary to make the mathematical treatment feasible. By applying mathematical methods from calculus, a set of partial or ordinary equations is obtained to describe the system.

Thus leading to a theoretical model with a certain structure and defined parameters. However, in many cases, the model may become too complex and not trivial, needing to be further simplified in order to be relevant for subsequent applications. Especially nowadays, where high computational power and complex simulation programs make the inclusion of as many physical descriptions as possible for the models an attractive idea. Nonetheless, such practice may hinder the relevant physical effects and observations, turning both the understanding and work with such models tiresome and non intuitive [8].

In the experimental approach, the mathematical model is obtained from measurements. Here, based on some a priori assumptions, the input as well as the output data are submitted to a chosen identification method in order to find a mathematical model that describes the relation between them. Thus, the choice of employing one or both approaches depends mainly on the purpose of the derived model. Although theoretical analysis may deliver more information about the system once internal behavior is known and mathematical description is feasible, the experimental approach has attracted increasing interest over the past decades from the scientific community. The main reason is that such analysis permits the development of mathematical models by measurement of the input-output behavior of systems of arbitrary complex composition. Therefore, identified models can be obtained in shorter time with less effort, which is sufficient for many areas of application.

Several approaches to identify nonlinear systems have been proposed, such as swarm intelligence, genetic algorithms and neural networks [6, 9]. Particularly successful have been neural networks, since universal approximation properties make them specially attractive and promising for applications to modelling and control of nonlinear systems. Also, in parallel, there remains a number of unsolved problems in nonlinear system control. For instance, the design and implementation of adaptive control schemes for nonlinear systems is remarkably difficult. In most cases the designed adaptive control methods largely rely on some a priori information on the nonlinear structure of the plant to be controlled. Thus, neural networks may contribute in the development of adaptive control for unknown nonlinear systems. If the dynamics between the input and the output of an unknown nonlinear system is modelled by a proper chosen neural network, the model obtained can be used to design a controller through conventional nonlinear control techniques in the literature. Furthermore, the whole approach of the training and construction of the controller can be performed online. The neural network model is updated by measured plant input and output data and then the controller parameters are directly adapted using the updated model. This approach is highly attractive for industry and engineering applications [10, 11, 12].

1.2 Thesis Statement

The objective of this Master's thesis is to develop two adaptive neural identification schemes for dynamical nonlinear systems. In these two schemes, the single-hidden layer feedforward neural network topology is used as the function approximator to estimate the unknown nonlinear systems. The first one, differently from the existing methods, a recently proposed neural algorithm referred to as Extreme Learning Machine (ELM) [13] is employed with modifications. Additionally to what is

already established in the literature, a stable online learning algorithm based on Lyapunov stability theory is developed to guarantee the convergence stability and approximation error boundedness of the ELM algorithm. The hidden-layer matrix is settled down in a random form and remains fixed and its online approximation capability in the presence of disturbances is enhanced by a robustifying term. The proposed neural network ensures that all associated errors are bounded and the convergence of the state estimation residual errors to zero is assured, in contrast to [14, 15, 16, 17, 1, 18]. Furthermore, with a few modifications regarding the selection of activation function and the regressor vector's structure, the achieved results can be applied to any linearly parameterized neural network model.

However, linearly parameterized models are known to suffer from the "curse of dimensionality" which may degrade their generalization performance. Also, the first scheme may present slow convergence if proper initial values for the hidden layer are not selected. One way to alleviate such limitations is to simultaneously adjust the hidden and output layers. Although it demands greater computational effort, this approach also allows for a faster adaptation of the identifier in the presence of disturbances that may appear, for example, as a consequence of faults. Considering the aforementioned problems, the approach employed in the first scheme is extended to a single-hidden layer feedforward neural network (SHLNN), where the results in [19] are extended in order to identify dynamical systems based on SHLNNs. All conditions are established to ensure the convergence of the residual state error to zero and all associated errors are bounded, even in the presence of approximation error and internal or external perturbations. Also, the dependence between the residual state error and some independent design parameters is straightforward. Consequently, the residual state error can be arbitrarily and easily reduced. Furthermore, it is not necessary any previous knowledge about the ideal weight, approximation error and disturbances, in contrast to [20, 21]. In addition, the designed methodology is structurally simple, since it does not use a dynamic feedback gain or bounding function employed in [20]. To provide stability, the weight adaptation laws are chosen based on Lyapunov theory. Simulation experiments are performed to illustrate the effectiveness of the proposed method.

1.3 Thesis Overview

The Master's thesis is organized as follows. Following this introductory chapter 1, historical developments of system modelling as well as a literature review on identification methods using neural networks are presented in Chapter 2. Technical background about the artificial neural networks is provided in Chapter 3 .

In Chapter 4, a neural network using extreme learning machine for identification of nonlinear systems is developed based on Lyapunov theory. Examples to illustrate the effectivess of the proposed method are presented. In chapter 5, the results of the previous chapter are extended for a single-hidden layer feedforward neural network. Simulations results are also provided.

Chapter 6 summarizes the research results and future research directions are discussed. The Appendix provide the software implementation of the theoretical contributions.

Chapter 2

Historical Developments and Literature Review

2.1 Historical Developments of System Identification

Modern system identification had its beginnings in the eighteen and nineteenth century breakthroughs of mathematics and probability theory. Among milestones such as Bayesian theory and Fourier transforms, it is often mentioned that the Least Squares Method and its concepts from Gauss [22] had a the major impact on data-based modeling and parameter estimation. Gauss's contribution of the Least Square method was derived from his approach to describe planetary orbits from astronomical data instead of using pure physical laws such as the classical Kleper's laws of motion. This gave impulse to developments largely inclined towards a statistical theory of parameter estimation and modeling of stochastic processes. Thus, much of the pioneering work on identification was developed by the econometrics, statistics, and time-series communities [23, 24, 8].

The formalization of theory and methods of identification as known today was developed mostly through a range of contributions from engineers and statisticians. However, up until the late 1950s, much of control design relied on traditional techniques such as Nyquist, Bode, and Nichols charts or on step response analyses. The scope of these techniques were limited to control design for single-input, single-output (SISO) systems. The necessity of model-based control-design techniques for more complex systems motivated the scientific and engineering community to expand the approach of modern control design beyond the realm of applications for which reasonably accurate low-dimensional dynamical models could already be obtained from the aforementioned approaches. Hence, data-based methods for developing dynamical models for diverse applications such as process control, environmental systems, biological and biomedical systems, and transportation systems [24] has been proposed. Despite several theoretical results on system identification having already been established in the statistics and econometrics literature, the year of 1965 can be pointed as the landmark for identification theory in the control community due to the publication of the pioneering papers [25, 26], which are treated as the foundational works for two streams of identification methods [8].

The Åström-Bohlin paper [26] presented the maximum likelihood framework that has been developed by the time-series community for solving the parameter estimation methods for autoregressive-moving average with exogenous terms (ARMAX) models [27, 28]. These models, later gave rise to the immensely successful prediction-error identification framework and was then extended to the general family of Box-Jenkins models [29]. On the other hand, the work of Ho and Kalman [25] provided a solution to the determination of state-space representations from impulse response coefficients. Subsequently, two significant works by the authors in [30, 31] laid the foundations for what is known as subspace state-space identification.

Following these researches, in the mid-1970s, with the introduction of prediction-error identification methods due to [32, 33, 34, 35, 36], the predominant view experienced a shift in the problem formulation, where the restrictive search for true model structure moved towards an ample and practical search for the best approximate models. Thus, description and explanation of model errors became the primary point of research. Justifying the control engineering approach, where the focus is on the model, rather than the parameters, which is viewed as just a vehicle for describing the model.

This position of prediction-error methods in the field of control was solidified by the authors in [37, 38, 39] where it is shown that by interpreting how the influence of experimental conditions, model structure and design choices translate on the identification model it was possible to tune the design variables in order to accomplish the objective for which the model is being identified. This approach led to a new perspective in which identification became viewed as a design problem. Moreover, this perspective clearly separates the engineering approach to system identification from the statistical and time-series approach. The latter view is that the model must clarify the data as well as possible.

The observation that the quality of a model can be altered by the selection of specific design variables in order to achieve and justify the model's goals introduced a new approach in the 1990s. The main application of this new shift is identification for the objective of model-based control design. Due to the fact that identification for control grasps many concepts of identification and control theory, research areas such as closed-loop identification, data-based robust control analysis and design, uncertainty estimation, experiment design and frequency-domain identification has flourished and developed greatly since 1990 [8, 24].

Due to the fact that in diverse fields of application, obtaining physical laws that describe the structure of the nonlinear system was time consuming and sometimes impractical, nonlinear system identification gained impulse. Since it reduces to estimating unknown parameters in the model on the basis of input-output measured signals [6]. Therefore, special interest has been focused on identification of nonlinear systems with unknown structure by introducing broader classes of nonlinear black-box models such as fuzzy, neural networks, wavelets and radial basis functions. Black-box models aim to model structures that have not been derived from physics laws and whose parameters therefore have a priori no physical significance. Fig. 2.1 shows a brief account of white to black box models (see [3]). Additionally, identification of nonlinear models is probably the most active area in System Identification today [6].

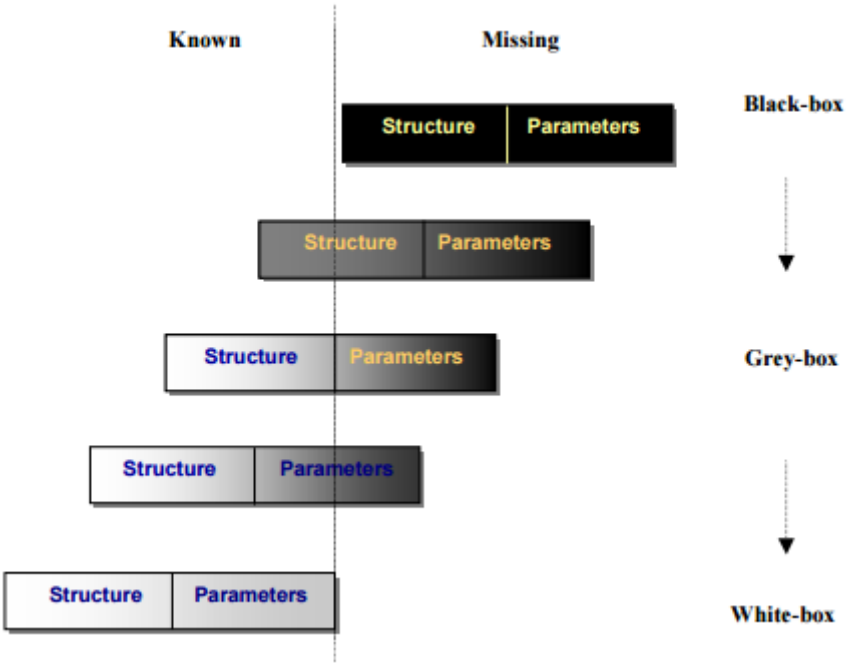


Figure 2.1: Model Categories Based on Prior Information [3]

The approximation capabilities of general continuous functions by neural networks has been extensively applied to system identification and control. Such approximation models are particularly useful in the black-box identification of nonlinear systems where nonexistent or very little a priori knowledge is available. For instance, neural networks have been employed for modeling and approximating of general nonlinear systems based on radial basis networks [40, 41], fuzzy sets and rules [42], neural-fuzzy networks [43] and wavelet neural networks [44, 45, 46].

2.2 State of the Art Review of Identification based on Single-Hidden Layer Neural Networks

It is well known that the mathematical characterization is, often, a prerequisite to observer and controller design. However, in some circumstances, the characterization of the dominant dynamics can be a difficult or even impossible task. In this scenario, the use of online approximators as, for instance, neural networks (NNs) is a possible alternative to parametrization. Since neural networks have good approximation capabilities and inherent adaptivity features, they provide a powerful tool for identification of systems with unknown nonlinearities [47, 48]. Basically, the unknown nonlinearities in the system are replaced by NN models, which have a known structure but unknown weights. In the case of supervised learning, the unknown weights are estimated by using an error signal between the outputs of the actual system and the neural identification model.

The application of neural network architectures to nonlinear system identification has been researched by several authors in discrete time [49, 50, 51, 52, 53, 54] and in continuous time[55, 56, 57]. A significant part of the research in discrete time systems are established by first replacing the

unknown plant in the difference equation by static neural networks and then obtaining update laws based on optimisation techniques (mostly, gradient descent methods) for a cost function (typically quadratic), which has led to the proposal of various backpropagation-like algorithms [58, 59, 60] that performed well in many applications. Nevertheless, the lack of rigorous proof for stability of the overall identification scheme and convergence of the output error remains a problem.

To improve the aforementioned limitations of backpropagation based algorithms, alternative approaches such as Lyapunov stability theory and adaptive control [61, 62] have been applied [55, 56, 57, 63, 64, 65], where the stability of the overall identification scheme is taken into account, which is an important issue. Even when the system is bounded-input bounded-output (BIBO) stable there is no a priori guarantee that the estimated state or the adjustable parameters of the identification model will remain bounded. The overall stability depends not only on the particular chosen identification model and architecture, but also on the parameter adjustment rules that are used. Therefore, under certain sufficient conditions, Lyapunov's theory can guarantee the convergence of the algorithm.

Neural identification models commonly employed are the linearly and nonlinearly parameterized, which can be by nature static or dynamic. Their weights are often adjusted using gradient-based schemes, as the backpropagation algorithm, or their robust modifications [2, 19, 20, 21, 10, 66, 67, 68, 69, 70]. The most widely-used robust modifications in neuro-identification are the σ, switching-σ, ε_1, parameter projection, and dead zone [1-10], which avoid the parameter drift.

Recently, identification schemes have been proposed using a single hidden-layer feedforward network (SHLNN) architecture with weights adjusted by a neural algorithm referred to as extreme learning machine (ELM) [71, 14, 15, 16, 17, 1, 18]. Different from gradient-based and backpropagation methods, the parameters of the hidden nodes need not be adjusted during training. All the hidden node parameters are randomly generated according to any given probability distribution, thus remaining fixed during training. Based on this, a SHLNN may be considered as a linearly parameterized neural network model, giving better computational efficiency in terms of learning speed and generalization performance, easing the "curse of dimensionality" [13, 72, 73, 74]. However, there are drawbacks for the ELM algorithm. Random choosing of input weights and biases may lead to a hidden layer output matrix that is not full column rank. This can make the least square method for obtaining the output weights (linking the hidden layer to the output layer) unsolvable [13, 72, 73, 74]. Further, the ELM and its variants lack the stability analysis and conditions to ensure the asymptotical convergence of the state error to zero. In this context, deriving an ELM-based identification scheme with adaptive output weights is highly desired.

Several approaches have been proposed to address this issue [14, 15, 17, 1, 18]. For instance, in [14] a surface vehicle scheme is identified online by a SHLNN approximator with random hidden nodes and adaptive output weights which are determined by the ELM and Lyapunov synthesis. However, the adaptive law only assures the boundedness of the residual state estimation errors to an arbitrary neighbourhood of zero. In [1], an online system identification algorithm based on the ELM approach for nonlinear systems has been developed using a Lyapunov approach, the adaptive law does not include a robustifying term, which may induce parameter drift and the residual state

estimation error may not converge in the presence of disturbances. The authors address the previous issues and extends the results for the discrete case in [18]. In [15, 17] a sliding controller is incorporated into the ELM based controller activated to work for offsetting the modeling errors brought by the SHLNN and system disturbances. The learning law based on sliding control is discontinuous, which may not be built in practice. In order to be performed, the sliding controller would need to pass through a method of smoothing, compromising the asymptotical convergence of the state error to zero. Therefore, the proposed scheme only ensures the convergence of the residual state estimation errors to an arbitrary neighbourhood of zero.

Despite the remarkable properties of the extreme learning machines, linearly parameterized neural networks typically suffers the so-called "curse of dimensionality", where as the input dimension of the system increases the number of nodes demanded to approximate nonlinear mappings increases exponentially. Thus, the computational demands, both in memory and computational time, can be significantly high for multiple-input multiple-output systems. Also, nonlinearly parameterized neural networks provide greater approximation power than linearly parameterized models. For instance, the authors in [75, 5] shows that for certain classes of functions, single-hidden layer neural network models with a sigmoid activation function can achieve a given approximation accuracy with a number of nodes that is linearly dependent on the dimension of the input vector. Thus, the aforementioned properties make SHLNNs well worth for investigating its application for system identification of nonlinear systems.

For instance, in [67], the neuro-identification of a general class of uncertain continuous-time dynamical systems was proposed, and a σ-modification adaptive law for the weights of recurrent high-order neural networks (RHONNs) was chosen to ensure that the state error converges to the neighborhood of zero. More recently, in [20, 21, 66], neuro identification schemes for open loop systems were proposed. In [20, 21] was established the conditions to ensure the asymptotical convergence of the residual state error to zero, even in the presence of approximation error and bounded internal or external perturbations. The convergence of the state error to zero in both works ([20, 21]) was based, among other, on the previous knowledge of bounds for the approximation error and perturbations, which are usually unknown in practice. In [66], an identification scheme based on a dynamical neural model with scaling and a robust weight adaptive law was proposed. The main peculiarity of [66] is that the residual state error is directly related to two design matrices, which allow the residual state error to be arbitrarily and easily reduced.

Despite the remarkable theoretical contribution in these works ([67, 20, 21, 66]), they are all based on linearly parameterized neural networks and consequently, in general, suffer from “the curse of dimensionality”. That is, these models have a poor capability of interpolation and require a large number of basic functions to deal with multi-dimensional inputs. This drawback can be alleviated by using identification models based on SLHNNs. See, for instance, [19, 2, 68, 70, 69]. In these works, the presence of the two weight matrices to be estimated, approximation errors, and perturbations, however, make the problem challenging.

For example, in [19], an online approximator of multi-input multiple output static functions based on SHLNNs is proposed. In [2], a robust scheme based on SHLNNs to identify nonlinear

systems was proposed. The weight adaptation laws were based on modified backpropagation algorithms. By using the Lyapunov' direct method, it was shown that all errors are uniformly bounded and the residual state error converges to a ball whose radius can be reduced by setting some design parameters in adequate values. Nevertheless, the design parameters related with the performance are dependent and, therefore, arbitrary small residual state error could not be achieved. This drawback is also observed in [68]. Another disadvantage of [2] is that, due to static approximations assumed in the definition of the adaptive laws, the identification process may not converge in the presence of high frequency perturbations. In [70, 69], the discrete case is considered and the stability properties of the approximation errors are presented.

In this Master's thesis, a recently proposed neural algorithm referred to as Extreme Learning Machine (ELM) [13] is proposed with modifications. Additionally to what is already established in the literature, a stable online learning algorithm based on Lyapunov stability theory is developed to guarantee the convergence stability and approximation error boundedness of the ELM algorithm. The hidden-layer matrix is settled down in a random form and remains fixed and its online approximation capability in the presence of disturbances is enhanced by a robustifying term. The proposed neural network ensures that all associated errors are bounded and the convergence of the state estimation residual errors to zero is assured, in contrast to [14, 15, 16, 17, 1, 18]. Furthermore, with a few modifications regarding the selection of activation function and the regressor vector's structure, the achieved results can be applied to any linearly parameterized neural network model. To the best of the author's knowledge, the proposed ELM modification is the first in the literature to ensure the convergence of the state estimation residual errors to zero in the presence of limited disturbances.

Moreover, the approach employed in the first scheme is extended to a single-hidden layer feedforward neural network (SHLNN), where the results in [19] are extended in order to identify dynamical systems based on SHLNNs. The hidden and output weights are simultaneously adjusted by robust adaptive laws that are designed via Lyapunov stability theory. All conditions are established to ensure the convergence of the residual state error to zero and all associated errors are bounded, even in the presence of approximation error and internal or external perturbations. Also, the dependence between the residual state error and some independent design parameters is straightforward. Consequently, the residual state error can be arbitrarily and easily reduced. Furthermore, it is not necessary any previous knowledge about the ideal weight, approximation error and disturbances, in contrast to [20, 21]. In addition, the designed methodology is structurally simple, since it does not use a dynamic feedback gain or bounding function employed in [20]. To provide stability, the weight adaptation laws are chosen based on Lyapunov theory. Extensive simulation results are performed to illustrate the effectiveness of the proposed methods.

2.3 Mathematical Preliminaries

This section provides some fundamental mathematical concepts that are necessary for the remaining chapters.

2.3.1 Function Norms

Definition 1. *Let $f(t) : \Re_+ \to \Re$ be a continuous function or piecewise continuous function. The p-norm of f is defined by*

$$\begin{aligned} \|f\|_p &= \left(\int_0^\infty |f(t)|^p dt \right)^{1/p}, \ for\ p \in [1, \infty) \\ \|f\|_\infty &= \sup_{t \in [0,\infty)} |f(t)|, \ for\ p = \infty \end{aligned} \tag{2.1}$$

Thus, by denoting $p = 1, 2, \infty$, the corresponding normed spaces are called L_1, L_2, L_∞, respectively. Furthermore, from [10], let $f(t)$ be a function on $[0, \infty)$ of the signal spaces, they are defined as

$$\begin{aligned} L_1 &\triangleq \left\{ f : \Re_+ \to \Re \middle| \|f\|_1 = \int_0^\infty |f| dt < \infty, \ convolution\ kernel \right\} \\ L_2 &\triangleq \left\{ f : \Re_+ \to \Re \middle| \|f\|_2 = \int_0^\infty |f|^2 dt < \infty, \ finite\ energy \right\} \\ L_\infty &\triangleq \left\{ f : \Re_+ \to \Re \middle| \|f\|_\infty = \sup_{t \in [0,\infty)} |f(t)| < \infty, \ bounded\ signal \right\} \end{aligned} \tag{2.2}$$

From the signal perspective, the 1-norm, $\|x\|_1$, of the signal $x(t)$ can be viewed as the integral of its absolute value, the square $\|x\|_2^2$ of the 2-norm is often called the energy of the signal $x(t)$, and the ∞-norm is its absolute maximum peak value or amplitude.

2.3.2 Lyapunov Stability Theorem

The following definitions and theorem were extracted from [10, 62].

Definition 2. *A continuous function $\alpha(r) : \Re \to \Re$ belongs to class $\mathcal{K}$ if*

- $\alpha(0) = 0$;
- $\alpha(r) \to \infty$ *as* $r \to \infty$;
- $\alpha(0) > 0\ \forall r > 0$; *and*
- $\alpha(r)$ *is nondecreasing, i.e.,* $\alpha(r_1) \geq \alpha(r_2)$. $\forall r_1 > r_2$.

Definition 3. *A continuous function $V(x, t) : \Re^n \times \Re_+ \to \Re$ is*

- *locally positive definite if there exists a class $\mathcal{K}$ function $\alpha(.)$ such that $V(x, t) \geq \alpha(\|x\|)$ for all $t \geq 0$ and in the neighbourhood $\mathcal{N}$ of the origin $\Re^n$;*
- *positive definite if $\mathcal{N} = \Re^n$;*
- *(locally) negative definite if $-V$ is (locally) positive definite; and*

- *(locally) decrescent if there exists a class $\mathcal{K}$ function $\beta(.)$ such that $V(x,t) \leq \beta(\|x\|)$ for $t \geq 0$ and in (the neighbourhood $\mathcal{N}$ of the origin) $\Re^n$.*

Definition 4. *Given a continuously differential function $V : \Re^n \times \Re_+ \to \Re$, together with a system of differential equations*

$$\dot{x} = f(x,t) \tag{2.3}$$

the derivative of V along the system if defined as

$$\dot{V} = \frac{dV(x,t)}{dt} = \frac{\partial V(x,t)}{\partial t} + \left[\frac{\partial V(x,t)}{\partial x}\right]^T f(t,x) \tag{2.4}$$

Theorem 2.3.1. ***(Lyapunov Theorem)**. Given the nonlinear dynamic system*

$$\dot{x} = f(x,t),\ x(0) = x_0 \tag{2.5}$$

with an equilibrium point at the origin, and let $\mathcal{N}$ be a neighbourhood of the origin, i.e. $\mathcal{N} = \{x : \|x\| \leq \epsilon,\ with\ \epsilon > 0\}$, then, the origin 0 is

- *stable in the sense of Lyapunov if for $x \in \mathcal{N}$, there exists a scalar function $V(x,t)$ such that $V(x,t) > 0$ and $\dot{V}(x,t) \leq 0$;*
- *uniformly stable if for $x \in \mathcal{N}$, there exists a scalar function $V(x,t)$ such that $V(x,t) > 0$ and decrescent and $\dot{V}(x,t) \leq 0$;*
- *asymptotically stable if for $x \in \mathcal{N}$, there exists a scalar function $V(x,t)$ such that $V(x,t) > 0$ and $\dot{V}(x,t) < 0$;*
- *globally asymptotically stable if for $x \in \Re^n$ (i.e. $\mathcal{N} = \Re^n$), there exists a scalar function $V(x,t)$ such that $V(x,t) > 0$ and $\dot{V}(x,t) < 0$;*
- *uniformly asymptotically stable if for $x \in \Re^n$ (i.e. $\mathcal{N} = \Re^n$), there exists a scalar function $V(x,t)$ such that $V(x,t) > 0$ and decrescent and $\dot{V}(x,t) < 0$;*
- *globally, uniformly, asymptotically stable if for $\mathcal{N} = \Re^n$, there exists a scalar function $V(x,t)$ such that $V(x,t) > 0$ and decrescent and is radially unbounded (i.e., $V(x,t) \to \infty$ uniformly in time as $\|x\| \to \infty$) and $\dot{V}(x,t) < 0$*
- *exponentially stable if there exist positive constants α, β, γ such that, $\forall x \in \mathcal{N}$, $\alpha\|x\|^2 \leq V(x,t) \leq \beta\|x\|^2$ and $\dot{V}(x,t) \leq -\gamma\|x\|^2$; and*
- *globally exponentially stable if there exist positive constants α, β, γ such that, $\forall x \in \Re^n$, $\alpha\|x\|^2 \leq V(x,t) \leq \beta\|x\|^2$ and $\dot{V}(x,t) \leq -\gamma\|x\|^2$.*

The function $V(x,t)$ showed in Theorem 2.3.1 is usually called a *Lyapunov function*. The theorem outlines sufficient conditions for the origin to be stable. However, no conclusion on the stability and instability can be defined if a specific choice of Lyapunov candidate does not meet the conditions on $\dot{V}(x,t)$.

A Lyapunov function is not unique, in other words, there may exist multiple Lyapunov functions for the same system. Nonetheless, for a given system, particular choices of Lyapunov functions may return better results than others. For controller design, different choices of Lyapunov functions may yield different forms of controller with different performances.

2.3.3 Boundedness and Ultimate Boundedness

For uncertain systems it can be impossible to determine the equilibrium points, which may limit the applications of the previous definitions. In this case, an useful concept for the stability analysis is the definition of boundedness and ultimate boundedness

Definition 5. *The solutions of $\dot{x} = f(x,t)$ where $f : (0,\infty) \times D \to \Re^n$ is piecewise continuous in t and locally Lipschitz in x on $(0,\infty) \times D$, and $D \in \Re^n$ is a domain that contains the origin are*

- ***uniformly bounded*** *if there exist a positive constant c, independent of $t_0 \geq 0$, and for every $\alpha \in (0,c)$, there is a $\beta = \beta(\alpha) > 0$, independent of t_0, such that*

$$\|x(t_0)\| \leq \alpha \Rightarrow \|x(t)\| \leq \beta, \ \forall t \geq t_0 \tag{2.6}$$

- ***uniformly ultimately bounded*** *if there exist positive constants b and c, independent of $t_0 \geq 0$, and for every $\alpha \in (0,c)$, there is $T = T(\alpha,b) > 0$, independent of t_0, such that*

$$\|x(t_0)\| \leq \alpha \Rightarrow \|x(t)\| \leq b, \ \forall t \geq t_0 + T \tag{2.7}$$

2.3.4 Barbalat's Lemma and Lyapunov-Like Lemma

Generally, asymptotic stability analysis for non-autonomous systems are more complex that for autonomous systems, once that is more difficult to choose Lyapunov candidates with negative definite derivative. The Barbalat's lemma [10, 62] offers a useful set of results that may help in solutions evolving asymptotic stability.

Lemma 2.3.2. *Let $f(t)$ be a differentiable function, if $lim_{t\to\infty} f(t) = k < \infty$ and $\dot{f}(t)$ is uniformly continuous, then*

$$lim_{t\to\infty} \dot{f}(t) = 0 \tag{2.8}$$

Corollary 2.3.3. *If $f(t)$ is uniformly continuous [1], such that*

$$lim_{t\to\infty} \int_0^t f(\tau) d\tau \tag{2.9}$$

exists and is finite, then $f(t) \to 0$ as $t \to \infty$

Corollary 2.3.4. *If $f(t)$, $\dot{f}(t) \in L_\infty$, and $f(t) \in L_p$, for some $p \in [1,\infty)$, then $f(t) \to 0$ as $t \to \infty$.*

[1] A function $f : A \to R$ is uniformly continuous on A if for every $\epsilon > 0$ there exists a $\delta > 0$ such that $|x - y| < \delta$ implies $|f(x) - f(y)| < \epsilon$.

Corollary 2.3.5. *For the differentiable function $f(t)$, if $lim_{t\to\infty} f(t) = k < \infty$ and $\ddot{f}(t)$ exists, then $\dot{f}(t) \to 0$ as $t \to \infty$.*

Barbalat's lemma is merely a mathematical result regarding the asymptotic properties of functions and their derivatives. By properly applying the Barbalat's lemma to the analysis of dynamic systems, particularly non-autonomous systems, the following Lyapunov-like lemma can be obtained.

Lemma 2.3.6. ***("Lyapunov-Like Lemma")*** *If a scalar function $V(x,t)$ satisfies the following conditions*

- *$V(x,t)$ is lower bounded*
- *$\dot{V}(x,t)$ is negative semi-definite*
- *$\dot{V}(x,t)$ is uniformly continuous in time then $\dot{V}(x,t) \to 0$ as $t \to \infty$*

Where as V approaches a finite limiting value V_∞, such that $V_\infty \leq V(x(0),0)$, which does not require uniform continuity.

Chapter 3

Technical Background

3.1 Motivation

In this chapter, technical background about neural networks, their properties and the notation that will be used throughout this Master's thesis will be introduced. Furthermore, a brief description for the most used neural network topologies and the basic types of learning will be given. Our aim is to provide a basic framework to understand the different architectures and strategies that are used for neural based identification. Keeping that goal in mind, we start with a mathematical description for the most basic component of a neural network, the neuron.

3.2 Artificial Neural Networks

3.2.1 Model of a Neuron and General Form of Neural Networks

A neuron is the fundamental information-processing unit for the operation of a neural network [4]. The individual processing unit receives input from other sources or output signals of other units and produces an output. Fig. 3.1 presents the block diagram of a neuron scheme. Basically, there are three components:

- A set of synapses, or connecting links, with each element being characterized by its own weight or strength. The input signal x_m is multiplied by the weight w_{km} between the sending unit m and receiving unit k.
- An adder for summing the inputs signal components, multiplied by the respective synapses weight. The operations described here constitute a linear combiner.
- An activation function where the sum of the weighted inputs is passed through. It transforms the adder output into the output of the neuron by limiting its amplitude. The activation function is also referred in the literature as squashing function, in that it squashes (limits) the permissible amplitude range of the output signal to some finite value.

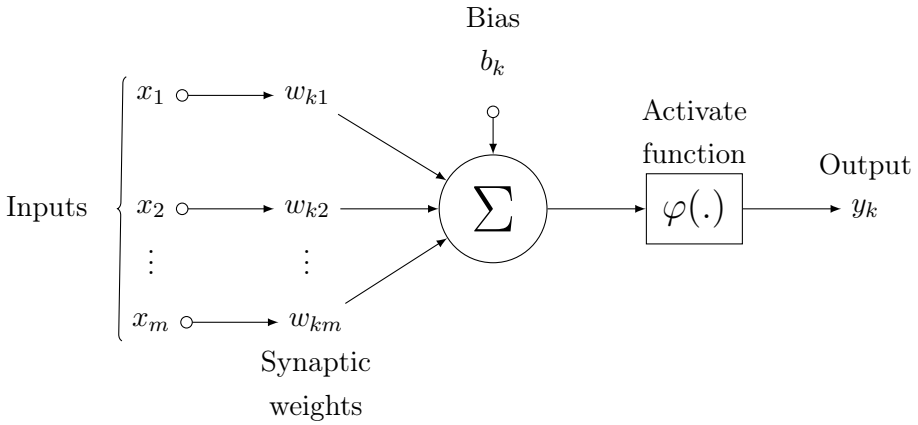


Figure 3.1: Nonlinear model of a neuron [4]

The neuron scheme presented in Fig. 3.1 also includes an externally applied bias or threshold, denoted by b_k. The bias b_k increases or lowers the net input of the activation function, depending on whether it is positive or negative, respectively.

The neuron can be mathematically described by the following pair of equations

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{3.1}$$

$$y_k = \varphi(u_k + b_k) \tag{3.2}$$

where $x_1, x_2, ..., x_m$ are the input signals and m the number of inputs; $w_1, w_2, ..., w_m$ are the respective synaptic weights of the neuron; u_k is the linear combiner output due to the input signals; $\varphi(.)$ denotes the nonlinear activation function; and y_k is the output signal of the neuron. The use of external bias or threshold b_k has the effect of applying an affine transformation to the output of the linear combiner. Equivalently, (3.2) can have the index rewritten to include the external parameter b_k as follows

$$v_k = u_k + b_k \tag{3.3}$$

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{3.4}$$

$$y_k = \varphi(v_k) \tag{3.5}$$

where

$$w_{k0} = b_k \tag{3.6}$$

The mathematical representation of a neuron described in (3.3)-(3.5) forms the basis for designing a large family of neural networks. Essentially, neural networks are parametric models and can

be described as a linear combination of basis functions. Thus, the neural network can be generally denoted by

$$f(u;w) = \sum_{k=1}^{m} w_k \varphi_k(u) \tag{3.7}$$

where w is the parameter vector containing the weights w_k and the set of parameters that define the basis function $\varphi_k(u)$, m is the number of basis functions used in the overall mapping of the network. For each parameter vector $w \in \mathcal{P}$, the network mapping $f \in \mathcal{F}_w$ where $\mathcal{P}$ is the parameter set and $\mathcal{F}_w$ the set of functions which can be represented by the chosen neural network [12, 39].

In general, a neural network is characterized by the following three major components [4, 75]:

- An activation function $\varphi_{\cdot}$ which describes the nonlinear mapping between the input and output of a neuron. The performance of a neural network to a given application depends on the proper choice of the activation function.
- The network architecture that specifies what variables are involved in the model and their topological relationships. Thus the neural network structure is determined based on deciding the number of neurons in each layer and how these neurons are linked to each other by weights. The choices made in this step will determine the complexity of the implementation, the type and level of performance that can be achieved.
- The learning algorithm to train the network which describes how the neural network's weights should change with time or adapt based on the data and control performance.

3.2.2 Universal Approximation of Artificial Neural Networks

Considering that most nonlinear processes show complex behavior, the class of models generally applied are not capable of describing the process exactly. The bias error, also called approximation error, can be defined as the error between the process and model purely as a result of the structural inflexibility of the model. Since a nonlinear process can not be normally modeled without an approximation error, it can only be approximated by some universal approximator such as polynomial, fuzzy system or neural networks. However, by raising model complexity (degree of the polynomial, number of rules or number of neurons) it can be expected that the approximation error reduces to zero. If this property is achieved by an approximator for all smooth processes, then it is called a universal approximator.

The first research attempts to show the approximation properties of multilayer perceptrons were introduced by [76, 77, 78], the authors argued that Kolmogorov's theorem on the representation of functions provided the theoretical support for neural networks as models for the representation of arbitrary continuous functions. After, [79] proved that a single hidden-layer multilayer perceptron with cosine-sigmoidal function behaves like a special case of a Fourier network whose output is analog to a Fourier series approximation for a given function. However, rigorous and mathematically concise proofs for the universal approximation capability of single-hidden layer feedforward

neural networks were given, independently, by [80, 81, 82]. These papers demonstrated that these networks can approximate not only an unknown function, but also approximate its derivative. Furthermore, [83] showed that networks using sigmoid type functions can also approximate piecewise differentiable functions.

The previous researches focused on the approximation properties of neural networks for sigmoidal neural networks. However, using a theorem proposed by [84], the authors in [85] extended the results in [80] to any continuous function $f \in \Re^n$ and proved that signed integer weights and thresholds are sufficient to guarantee a proper approximation. The universality of single-hidden layer networks with neurons having non-sigmoid activation functions was formally proven by [86]. Additionally, [87] showed that a sufficient condition for universal approximation can be obtained by using continuous, bounded, and nonconstant activation functions. Finally, [88, 89]) have developed these results by determining that a neural network with locally bounded piecewise continuous activation function for hidden neurons is a universal approximator if and only if the function is not a polynomial. The theorem in mathematical terms:

Theorem 3.2.1. *Let $\varphi(.)$ be a nonconstant, bounded and monotone increasing continuous function. Let $S \subseteq \Re^m$ and S is compact. The space of continuous functions on S is denoted by C(S). Then, given any function $f \in C(S)$, and any $\varepsilon > 0$, there exists an integer $n \in N$ and real constants $a_{i,j}$, b_i, $w_i \in \Re$, where $i \in 1 \ldots n$, $j \in 1 \ldots m$ such that we may define*

$$f_{nn}(x) = \sum_{i=1}^{n} w_i \varphi(\sum_{j=1}^{m} a_{ij} x_j + b_i) \tag{3.8}$$

as an approximation of the function $f(.)$ that is

$$\|f(x) - f_{nn}(x)\| < \varepsilon \tag{3.9}$$

The universal approximation capability is an important property since it justifies the application of the neural networks to any function approximation problem. The theorem is an *existence theorem* in the sense that it provides the mathematical justification for the approximation of an arbitrary continuous function. However, the proof is not constructive due to the fact that no method is provided for finding the ideal weights, optimum learning time and no information about how many hidden neurons would be required to achieve a given accuracy.

3.2.3 Capabilities and Limitations of Neural Networks

The following features of artificial neural networks make them specially attractive and promising for a wide range of applications for modelling and control of nonlinear systems [10]

- Neural networks with one or more hidden layers has universal approximation abilities, i.e., can approximate any continuous nonlinear function arbitrarily well over a compact set, provided sufficient hidden neurons are available.

- The network has a highly parallel structure and computation speed and consists of many simple elements, which is attractive from the viewpoint of feasibility for hardware implementation. Furthermore, the connected structure of numerous neurons exhibit fault tolerance in the sense that a failure in some units may not significantly affect the general performance of the network. This property is know in the literature as "graceful degradation" [90].
- Online learning and adaptation of neural networks are possible due to their generalization abilities with respect to fresh and unknown data.
- Neural networks eliminate the need to develop an explicit model of a process that may be hard to identify, making them practical and efficient "black box" models to implicitly detect complex nonlinear relationships between dependent and independent variables.

At the same time, it also has the following limitations:

- Depending on the chosen learning technique, neural networks may require long training time and present slow learning speed.
- It is not trivial to extract ideal training samples for a given learning algorithm, which may also result in local minima problem.
- It is not easy to optimize the network structure in the sense that a network with insufficient number of neurons may not converge accordingly, but also an oversized network will result in poor generalization performance and be prone to overfitting. Without a priori knowledge of the problem, the topology must be determined on a trial and error basis.
- The "black box" nature does not provide physical meaning or explanation. Since a trained neural network extracts knowledge from a training sample and creates its own internal representation, it is difficult to delineate an intuitive interpretation about input-output behavior of the system.
- It is theoretically difficult to solve the convergence problem completely and assure a proper learning for the neural network algorithm.

3.2.4 Linearly and Nonlinearly Parametrized Approach

Based on the location of the adjustable parameters, the neural networks can be classified into linearly and nonlinearly parametrized approximators. From an analytical viewpoint it is convenient to provide a common framework for the study of the various topologies that belong to each class. It should be noted that neural networks are never truly linearly parametrized. The class of basis function neural networks, such as radial, fuzzy or wavelets basis function networks, only turn into linearly parametrized approximators when a technique has been used to fix or to select the basis functions. For these classes of neural networks, learning can be performed in two steps. In the first step, the hidden layer which is a set of basis functions is determined. Then, the second step of learning becomes a linear learning problem. The linearly parametrized approach for neural

network learning refers to this two-step process. The main advantage of this approach is that it avoids complex optimization techniques [75, 10].

When the hidden layer of a neural network performs a fixed nonlinear transformation with no adjustable parameters, i.e., the input space is mapped into a new space and then the outputs are combined linearly in the output layer. The neural network, in this case, belongs to a class of linearly parametrized approximators. $\mathcal{F}$ is of the form

$$\mathcal{F}(W, z) = W\varphi(z) \tag{3.10}$$

where $\varphi(.)$ is a nonlinear activation function, W and z are the weight and the input vector, respectively. Approximators whose structure is such that the parameters appear in a nonlinear fashion are referred to as nonlinearly parametrized approximators. Thus $\mathcal{F}$ is of the form

$$\mathcal{F}(W, z) = W\varphi(Vz) \tag{3.11}$$

where the hidden layer weight V has a nonlinear behavior given the nonlinear nature of the activation function. In the context of approximation theory, linearly parameterized approximation corresponds to the special case of the nonlinearly parameterized methodology.

Although the linearly parametrized approach may simplify the stability analysis, the nonlinearly parametrized approach has better representation power and significantly smaller approximation errors. Furthermore, it also alleviates the "curse of dimensionality" common in linearly parameterized approximators [75].

3.3 Neural Network Structures

Topology of a Neural Network (also called architecture or structure) refers to the way the neurons are interconnected. Since proper design requires selection of a family of function approximators, specification of the structure of the neural network and choosing the proper parameter estimation or learning laws, the choice of how the neurons are structured plays an important factor in network functioning and learning behavior.

Artificial Neural Networks can be classified in two major groups: feedforward (or static or non-recurrent) networks, and feedback (or dynamical or recurrent) networks. In the former, the information flows only in one direction, i.e., the output relies only on the actual values of the input and there is no cycle or loops in the network. The latter, contrary to static networks, are models where the data flow is bi-directional, having at least one feedback loop.

Although recurrent networks offer great computational advantages for model and storage of temporal information, they are better used in tasks where associative memory is needed, such as time series or sequential tasks. The static networks are ideally suitable for functional mapping problems, where the analysis of how the input variables affect the output behavior is desired. Among the commonly used static neural network structures for system identification are multilayer

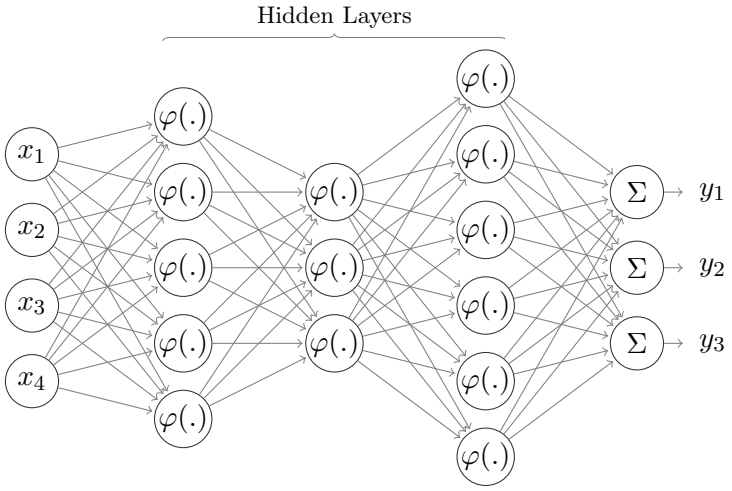


Figure 3.2: Multilayer Perceptron

perceptron, radial basis function, wavelet and fuzzy networks. In practice, Multilayer Feedforward Networks are the most widely studied and used neural network model, being the model of choice in this Master's thesis. However, with a few modifications, the proposed methodology can also be applied to other topologies.

3.3.1 Multilayer Feedforward Neural Network

The first researches in neural networks focused on simple neural networks with one layer of linear or nonlinear output units [91, 92]. However, the scientific community soon realized that the complexity of real applications could hardly be matched by these simple network architectures [93]. To approach these issues, the introduction of multilayer neural networks provided a new impulse to neural network research [94]. A multilayer neural network may contain one or more layers of hidden units between the input and output layers. The input layer performs as an input data holder that assigns the inputs to the first hidden layer. The output from the first layer nodes then becomes inputs to the second layer, and so on. The last layer acts as the network output layer. As an example, figure 3.2 shows a multilayer feedforward neural network with three hidden layers. The hidden units are connected to the input units through the synaptic weights v_{jk}, which form the matrix $\mathbf{V}$, and to the output units through the synaptic weights w_{ij}, which form the matrix $\mathbf{W}$. Thus, it can be expressed in mathematical form as

$$f_i = \sum_{j=1}^{l} [w_{ij} \varphi(\sum_{k=1}^{m} v_{jk} \bar{x}_k + \theta_{v_j})] + \theta_{w_i} \tag{3.12}$$

where $\bar{x}_k = [x, u]^T \in \Re^m$ is the augmented input vector for the neural network and $x \in \Re^n$, $u \in \Re^p$ are respectively the states for estimation and control input vectors, l is the number of hidden neurons, v_{jk} are the input to hidden layer weights, w_{ij} are the hidden to output layer weights, θ_{v_j} and θ_{w_i} are the threshold offsets. Table 3.1 provide some of the common activation functions $\varphi(.)$ used in the literature.

Activation Functions	
Name	**Formula**
Linear	αx
Arc-tangent	$\frac{2}{\pi} arctan \frac{\pi x}{2}$
Squash	$\frac{x}{1+\|x\|}$
Sigmoid	$\frac{1}{1+e^{-\gamma x}},\ \gamma > 0$
Hyperbolic Tangent	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Table 3.1: Common Activation Functions for MLP Networks

For the sake of simplicity of representation, the single-hidden layer network in (3.12) can be rewritten in a matrix format as below [10]. Define

$$\mathcal{F} = \Big[f_1,\ f_2,\ \ldots,\ f_n\Big]^T \in \Re^n \tag{3.13}$$

$$V = \Big[v_1,\ v_2,\ \ldots,\ v_l\Big]^T \in \Re^{(m+1)\times l} \tag{3.14}$$

$$z = \Big[\bar{x}_k, 1\Big]^T \in \Re^{m+1} \tag{3.15}$$

with $v_i = [v_{i1}, v_{i2}, \ldots, v_{i(n+1)}]$, $i = 1, 2, \ldots, l$. The term z_{m+1} in input vector z allows one to include the threshold vector $[\theta_{v1}, \theta_{v2}, \ldots, \theta_{vl}]$ as the last column of V, so that V contains both the weights and thresholds of the first-to-second layer connections. Hence, (3.12) can be expressed

$$\begin{aligned} \mathcal{F} &= \mathbf{W}\varphi(\mathbf{V}z) \\ \varphi(\mathbf{V}z) &= [\varphi(v_1^T z),\ \ldots,\ \varphi(v_l^T z),\ 1] \\ \mathbf{W} &= [w_1,\ w_2,\ \ldots,\ w_{l+1}] \in \Re^{l+1} \end{aligned} \tag{3.16}$$

where $w_i = [w_{i1}, w_{i2}, \ldots, w_{in}]$, $i = [1, 2, \ldots, l+1]$ and the last element in $\varphi(\mathbf{V}z)$ incorporates the threshold θ_w as w_{l+1} of weight $\mathbf{W}$. The convenience of the matrix format can be verified by using the notation from [50], where (3.16) can be extended for N layers $L_1, L_2, \ldots, L_n$ with L_1 being the input layer, L_n the output layer. Define $\varphi_i(.)$ the layer's activation function following each of the weight matrices, $\mathbf{W}_i$ the matrix containing the weights that connect the layers $L_i \to L_{i+1}$, where $i = [1, 2, \ldots, n-1]$. Thus the input-output mapping of the N-layer neural network can be represented by

$$\mathcal{F} = \varphi_n(\mathbf{W}_{n-1}\varphi_{n-1}(\ldots \mathbf{W}_2\varphi_2(\mathbf{W}_1 z))) \tag{3.17}$$

An MLP realizes an overall input-output mapping: $\mathcal{F} : \Re^n \to \Re$. The multilayer perceptron is an universal approximator, and theoretical works [82, 80, 81] have shown that multilayer perceptrons with one hidden layer are sufficient to approximate any continuous function, provided

that there are enough hidden nodes. In nonlinear system identification problems, the number of the output nodes is equal to the number of the system outputs, and the number of the network input is equal to the dimension of system inputs. From (3.17) network input-output mapping $\mathcal{F} : \Re^n \rightarrow \Re$ is clearly highly nonlinear in $\mathbf{W}_i$, and training an multilayer perceptron is analogue to estimating this parameter vector. After the learning process is complete, the network mapping can be assigned as a model of the system for control and other engineering applications.

Although modern research has revealed considerable drawbacks of the multilayer perceptron with respect to many applications, it is still the most widely known and used neural network architecture, sometimes even being used as a synonym for NN. One of the advantages is that the architecture permits possible extensions in the use of more than one hidden layer. Multiple hidden layers make the network much more powerful and complex, recently giving rise to the deep learning approach [95, 96, 97]. Additionally, accuracy is usually very high, given the nonlinearly parametrized structure, it often results in better approximation performance than the equivalent linearly parametrized network approaches. Also, due to the optimization of the hidden layer weights the multilayer perceptron can be extremely powerful and usually requires fewer neurons and parameters than other model topologies to perform a comparable approximation accuracy. Further, this property can be characterized as a high information compression capability.

3.3.2 High Order Neural Network

Since artificial neural networks for nonlinear system identification is that they categorize as "black-box" models, one of the drawbacks is that information about the system's structure and parameters may not be intuitive for interpretation. Higher Order Neural Networks (HONNs) help alleviate this problem by including nodes at the input layer that contribute the network with a better understanding of the input behavior and their dynamics. Essentially, the inputs are transformed via higher-order functions such as squares, cubes, or sines in which the net input to a neuron node is a weighted sum of products of its inputs. Although, this approach has been shown to accelerate training in some applications, in practice, only second order networks are used. The major drawback of HONNs is that the required number of weights to accommodate all the high-order correlations increases exponentially with the dimensionality of the input vector [67, 98].

The authors in [67, 99, 100] showed that HONNs have great computational, storage and learning capabilities. The order or structure of a HONN can be adapted according to the order or structure of a given problem, this advantage provide a more specialized and efficient approach in solving these problems. Additionally, a priori knowledge about the system can also be incorporated in a HONN when it is available. Also, results in [101] demonstrated HONNs to be at least as powerful as any other feedforward neural topology of same order. Another research [102], by using a piecewise linear HONN with the structure consisting of two layers of modifiable weights, has shown HONN to be seven times faster than standard feedforward neural networks when simulating the XOR problem. Another comparison with FNNs is that [103] points out HONNs to significantly decrease the needed training time for a given task. The authors in [67] studied the approximation and learning properties of a class of recurrent HONNs and successfully applied these topologies to the

identification of dynamical systems. The structure of HONNs can be expressed as

$$\dot{x}_i = -a_i x_i(t) + b_i \sum_{k=1}^{L} w_{ik} z(I_k) \tag{3.18}$$

$$z(I_k) = \prod_{j \in I_k} d_j^{m_j(k)}(x, u) \tag{3.19}$$

$$d = \begin{bmatrix} \mathcal{S}(x_1) \\ \mathcal{S}(x_2) \\ \vdots \\ \mathcal{S}(x_n) \\ \mathcal{S}(u_1) \\ \mathcal{S}(u_2) \\ \vdots \\ \mathcal{S}(u_n) \end{bmatrix} \tag{3.20}$$

where $u \in \Re^m$, $x \in \Re^n$, $I_1, I_2, \ldots, I_L$ is a collection of L nonordered subsets of $1, 2, \ldots, L$, $L \geq m + n$ is the number of high order interactions, $m_j(k)$ are nonnegative integers, a_i and b_i are positive constants and are fixed during training, $w_i k$ are the adjustable net weights and $m_j(k)$ are nonnegative integers. The dynamic behavior of the overall network is described by expressing (3.18) in vector notation as

$$\dot{x} = Ax + BW^T z \tag{3.21}$$

where $x = [x_1 \ldots x_n]^T$, $W = [w_i \ldots w_n] \in \Re^{L \times n}$, $A := diag(-a_1, -a_2, \ldots, -a_n)$ is an $n \times n$ diagonal matrix, also a stability matrix and $B := diag(-b_1, -b_2, \ldots, -b_n)$ is an $L \times L$ diagonal matrix.

It is proved in the literature [67, 94] that HONNs satisfy the conditions of the Stone-Weierstrass Theorem, guaranteeing to approximate any continuous function over a compact set. According to [98], high-order neural networks of any order can be trained by applying any of the learning algorithms proposed for first-order neural networks.

3.3.3 Radial Basis Function Neural Networks

Radial Basis Function (RBF) neural networks are three-layer models with an input layer, a hidden layer and output layer where a linear combination of the hidden inputs neurons occurs. Each of the hidden nodes performs a nonlinear transformation of the input, by means of Radial Basis Functions. For the RBF network, the basis function at the i-th hidden neuron is given by

$$f_i(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{c}_i\|) \tag{3.22}$$

where $\|.\|$ denotes a norm that is typically assumed to be Euclidean, $\mathbf{x} \in \Re^n$ the input vector, $\mathbf{c}_i$ with $i = 1, \ldots, n$ is the unit centre in the input space and $\phi(.)$ a nonlinear activation function. The basis functions are radially symmetric with the centre on $\mathbf{c}_i$ in the input space, hence they are named radial basis functions. The overall input-output response of an n-input m-output RBF network is a mapping $f_{RBF} : \Re^n \rightarrow \Re^m$. Mathematically,

$$f_{RBF_i} = \sum_{j=1}^{n_H} \theta_{j,i}\phi_j = \sum_{j=1}^{n_H} \theta_{j,i}\phi(\|\mathbf{x} - \mathbf{c}_i\|; \sigma_j), \qquad 1 \leq i \leq n_O \tag{3.23}$$

where $\theta_{j,i}$ are the weights of the linear combiners, σ_j are some positive scalars called the widths, $\mathbf{c}_i$ are the RBF centers and n_H is the number of hidden nodes. Also, (3.23) can be rewritten in vector form as

$$f_{RBF}(x) = \mathbf{W}\boldsymbol{\phi} \tag{3.24}$$

where

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \quad and \quad \boldsymbol{\phi} = \begin{bmatrix} \phi(\|\mathbf{x} - \mathbf{c}_1\|) \\ \phi(\|\mathbf{x} - \mathbf{c}_2\|) \\ \vdots \\ \phi(\|\mathbf{x} - \mathbf{c}_n\|) \end{bmatrix} \tag{3.25}$$

The RBF network is known to be a general function approximator, and theoretical research have concluded that the choice of $\phi(.)$ is not essential for network approximation capabilities [104, 105]. Nonetheless, deciding the nonlinearity of the hidden nodes according to the application can often improve performance. Examples of common activation functions for local and global RBFs are shown in table 3.2, where $r = \|\mathbf{x} - \mathbf{c}_i\|$, σ is a real number usually called receptive width or the width of the locally-tuned function which describes the declivity of the hyperbolic cone used in the radial function.

Figure 3.3 shows the architecture of a RBF network. The topology of the RBF network is similar to that of the single-hidden layer perceptron, and the difference lies in the description of the hidden node. Hence, its input to the hidden layer connection transforms the input into a distance from a point in the input space, unlike in the multilayer perceptron (MLP), where it is transformed into a distance from a hyperplane in the input space. Then, while the hidden neurons of a MLP with sigmoidal activation covers wide regions in the input space, the hidden neurons of a RBF covers small specific regions. Thus, multilayer networks are more efficient as function approximators. Now, for tasks such as pattern classification, RBF networks tends to outperform the MLP [106].

Activation Functions	
Name	**Formula**
Gaussian	$exp\left(\frac{r^2}{\sigma^2}\right)$
Inverse Multiquadric	$\left(r^2+\sigma^2\right)^{-\frac{1}{2}}$
Linear	r
Cubic	r^3
Shifted Logarithms	$ln\left(r^2+\sigma^2\right)$
Thin Plate Splines	$r^2 ln(r)$
Pseudo Potential Functions	$(1-exp\left(-\frac{r^2}{\sigma^2}\right) ln(r))$

Table 3.2: Common Activation Functions for RBF Networks

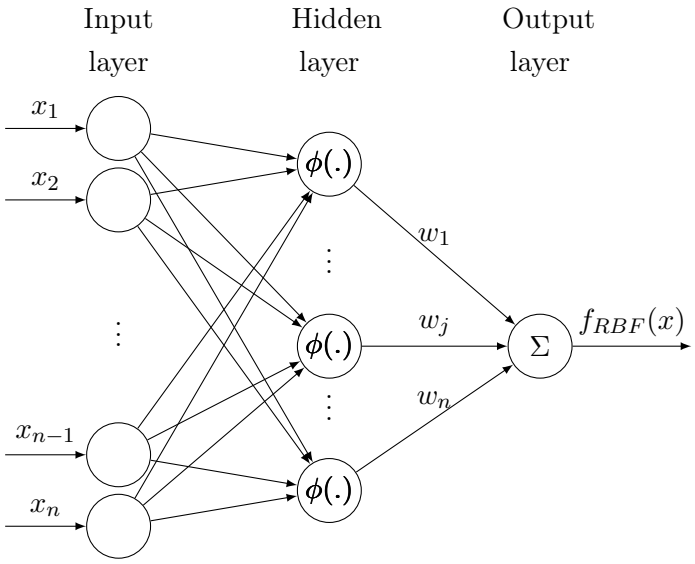


Figure 3.3: Radial Basis Function Neural Network

In general, the RBF network has a nonlinear-in-the-parameters structure. However, learning can be implemented in two steps. First, a learning mechanism is performed to select a suitable set of RBF centers and widths. This effectively determines the hidden layer of the RBF network. After this step, learning the remaining output-layer weights becomes a linear problem. In this aspect, the RBF network becomes a linear-in-the-parameters structure. This is true only after the hidden layer has been fixed separately before the second step. Because of this property, learning procedures for the RBF network can be straightforward and reliable [5].

3.3.4 Fuzzy Neural Networks

Fuzzy neural networks have their origin from fuzzy sets and fuzzy inference systems, which were initially introduced by [107] as a extension of Boolean logic. A basic configuration of a n-input m-output fuzzy system is shown in 3.4. A fuzzy system consists of four basic components: a fuzzifier, a fuzzy rule set, a fuzzy inference engine, and a defuzzifier. The fuzzifier deals with a mapping from the input space to the fuzzy sets defined in the input space. The fuzzy rule base consists of a set of n_F linguistic rules in the forms of IF-THEN values. The fuzzy inference engine is a decision-making logic that uses the fuzzy rules provided by the fuzzy rule base to determine a mapping from the fuzzy sets in the input space to the fuzzy sets in the output space. The efficiency of a fuzzy inference engine heavily relies on the knowledge base of the problem considered. The defuzzifier provides a mapping from the fuzzy sets in $\Re^m$ to the crisp outputs $\mathbf{y} \in \Re^{\mathrm{n}}$. A class of fuzzy systems typically used in practice is constructed based on product inference, singleton fuzzification and centroid or weighted average defuzzification [5]. Such fuzzy systems can be represented as series expansions of fuzzy basis functions known as fuzzy basis function (FBF) networks or models [108, 43].

A ni-input m-output fuzzy system can be mathematically expressed as m single-output fuzzy subsystems [5]. The rule base of the ith fuzzy subsystem consists of n_F rules defined as follows:

$$RB_j^i: \text{ IF } x_1 \textit{ is } A_{1,j} \text{ AND ... AND } x_{nI} \textit{ is } A_{nI,j}, \textit{ THEN } y_i \text{ is } B_j^i \tag{3.26}$$

where $1 \leq j \leq n_F$ for $1 \leq l \leq n$ are the inputs to the fuzzy system; $y_i$is the ith output of the fuzzy system, $1 \leq i \leq m$; and $A_{i,j}$ and B_j^i are the fuzzy sets described by fuzzy membership functions $\mu_{A_{l,j}}(x_l)$ and $\mu_{B_j^i}(y_i)$, respectively. Under the assumptions of singleton fuzzifier, product inference, and centroid defuzzifier, the input-output mapping of such a fuzzy system, $f_{FBF} : \Re^n \rightarrow \Re^m$can be demonstrated to have the following form [108, 43, 109]

$$f_{FBF_i}(\mathbf{x}) = \frac{\sum_{j=1}^{n_F} \bar{y}_j^i (\prod_{l=1}^{n} \mu_{A_{l,j}}(x_l))}{\sum_{j=1}^{n_F} (\prod_{l=1}^{n} \mu_{A_{l,j}}(x_l))}, \qquad 1 \leq i \leq m \tag{3.27}$$

where $\bar{y}_j^i$ is the point at which $\mu_{B_j^i}(y_i)$ achieves its maximum value. Define

$$\phi_i(\mathbf{x}) = \frac{\prod_{l=1}^{n} \mu_{A_{l,j}}(x_l)}{\sum_{j=1}^{n_F} (\prod_{l=1}^{n} \mu_{A_{l,j}}(x_l))}, \qquad 1 \leq i \leq n_F \tag{3.28}$$

which are referred to as fuzzy basis functions (FBFs). Then the fuzzy system (3.27) is equal to an FBF expansion,

$$f_{FBF_i}(\mathbf{x}) = \sum_{j=1}^{n_F} \phi_j(x) \theta_{j,i}, \qquad 1 \leq i \leq m \tag{3.29}$$

where $\theta_{j,i}$ are coefficients of parameters of the FBF model.

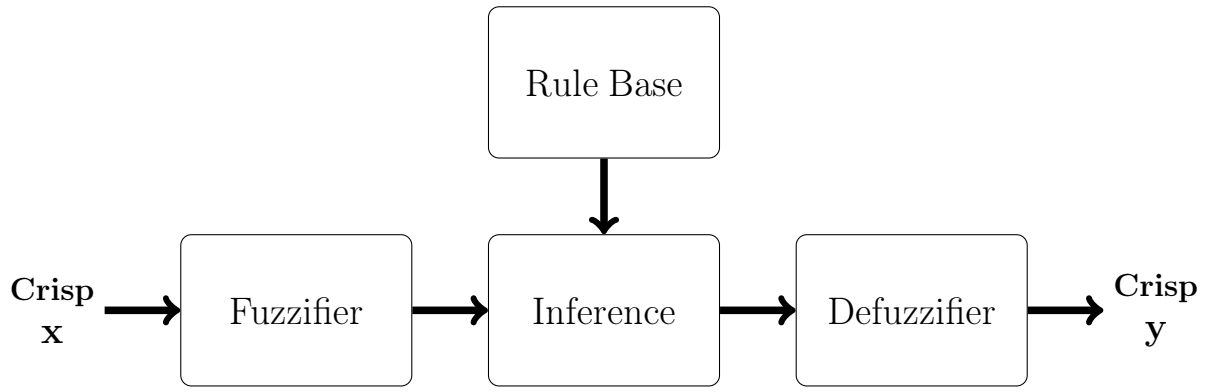


Figure 3.4: Fuzzy System Architecture, adapted from [5]

Fuzzy systems are universal approximators, and theoretical studies have proved that the FBF network (3.29) can approximate any continuous functions to within any degree of accuracy, provided that a sufficient number of fuzzy rules are used [108, 43, 109]. Although the FBF network is derived within the framework of fuzzy logic, it obviously has many similarities to neural networks such as the RBF network. In fact, under some conditions, singleton neuro-fuzzy systems are equivalent to normalized RBF networks [110, 111]. An advantage of the FBF network is that linguistic information from human knowledge in the form of the fuzzy IF-THEN rules can be integrated into the model, allowing improved understanding of the process and exploitation of prior knowledge [112].

Generally fuzzy systems are employed in engineering applications where the number of inputs and outputs is small. This is because the number of rules or FBFs grows exponentially as the number of inputs and outputs increases. This curse of "rule explosion" limits further uses of fuzzy systems to complex large systems. Furthermore, several restrictions and constraints must be assumed on the fuzzy in order to keep it interpretable. Several applications of fuzzy systems in areas such as financial prediction, nonlinear systems identification and control has been proposed [113, 114, 115]. Notice that the FBF network (3.29) has a linear-in-the-parameters structure once the rules have been specified.

3.3.5 Wavelet Neural Networks

Wavelet neural networks (WNNs) were introduced in the 1990s [44, 45], based on wavelet transform theory initiated by the works in [116]. Wavelet transform theory was developed to analyse signals with varied frequency resolutions such as a unifying idea of looking at non stationary signals at various time locations. For reviews and tutorials on wavelets see, for example [117, 118]. There are some significant differences between wavelet series expansions and classical Fourier series, which are

- Wavelets are local in both the frequency domain (via dilations) and in the time domain (via translations). On the other hand, Fourier basis functions are localised only in the frequency domain but not in the time domain. Small frequency changes in the Fourier transform will

cause changes everywhere in the time domain.

- Many classes of functions can be described in a more compact way by wavelets than by the Fourier series. Also, the wavelet basis functions are more effective than classical Fourier basis ones in achieving a comparable function approximation.

The above advantages give wavelets strong compression abilities via dilation and translation properties, helping alleviate the local minima problem of the classical sigmoidal neural networks [44]. The wavelets refer to a family of functions generated from one single function $\varphi(.)$ by the operation of dilation and translation. In the continuous case it takes the following form

$$\psi_i(x) =\mid a_i^{-1/2} \mid \varphi(\frac{x-b_i}{a_i}) \tag{3.30}$$

where $x = [x_1, \ldots, x_n] \in \Re^n$ is the input vector and the parameters $a_i, b_i \in \Re$ and $i \in \mathcal{Z}$ are named the scale and translation parameters, respectively. The parameter a_i is a scaling or dilation factor and b_i a translation factor of the original function $\varphi(.)$.

In the standard form of a wavelet neural network, output is given by

$$f(x) = \sum_{i=1}^{m} w_i \psi_i(x) = \sum_{i=1}^{m} w_i \mid a_i \mid^{-1/2} \varphi(\frac{x-b_i}{a_i}) \tag{3.31}$$

where m is the number of hidden neurons, ψ_i is the wavelet activation function of i-th unit of the hidden layer and w_i is the weight connecting the i-th unit of the hidden layer to the output-layer unit. An example of the WNN architecture is shown in 3.5.It should be noted that for the n-dimensional input space, the multivariate wavelet-basis function can be computed by the tensor product of n single wavelet-basis functions given by

$$\varphi(x) = \prod_{j=1}^{n} \varphi(x_j) \tag{3.32}$$

Wavelet neural networks have the universal approximation property [119, 44]. Some examples of wavelet families used as activation function are shown in table 3.3, where $r = x^T x$ and $k = dim(x)$, whereas there are many other types of wavelet functions which depend on the specific application. It should be noted in (3.31) that the wavelet neural networks have a linearly parametrized structure. However, if the translation and dilation parameters a_i and b_i are handled as adjustable parameters the referred network becomes nonlinearly parametrized and can be trained like any multilayer perceptron neural network [120].

Wavelet theory and networks have been widely employed in applications in diverse fields, such as system identification [46, 121] geophysics [122], signal denoising [123], electric load forecasting [124], time-series prediction [125] and control [126]. However, according to [120], wavelet networks suffer from "curse of dimensionality". To train a wavelet neural network, the gradients with respect to all the unknown parameters must be expressed explicitly. Thus, the calculation of gradients may

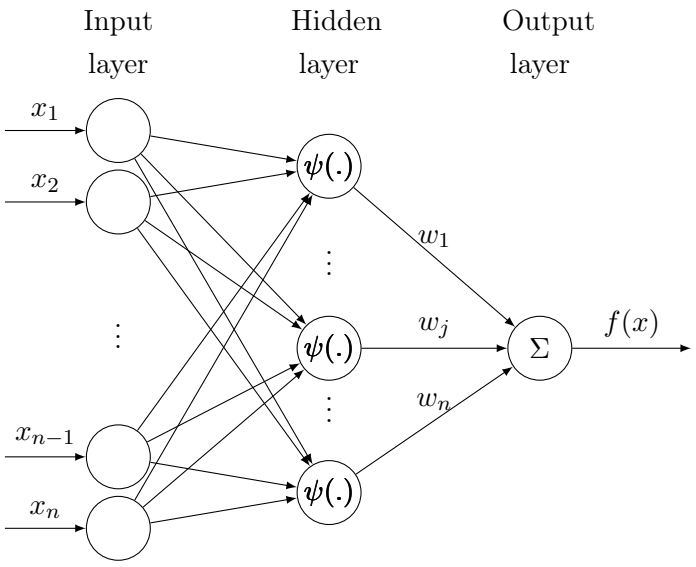


Figure 3.5: Wavelet Neural Network

Activation Functions	
Name	**Formula**
Gaussian derivative	$(k-r)e^{-r/2}$
Mexican Hat	$\frac{2}{\sqrt{3}\pi^{1/4}}(k-r^2)e^{-r^2/2}$
Morlet Wavelet	$cos(5r)e^{-r^2/2}$
Haar wavelet	$\begin{cases} 1\ if\ 0 \leq x < 1/2 \\ -1\ if\ 1/2 \leq x < 1 \\ 0\ otherwise \end{cases}$

Table 3.3: Common Activation Functions for Wavelet Networks

be heavy and difficult in high-dimensional models as the number of candidate wavelet terms often increases with the model order. Also, another problem is how to determine the initial number of wavelets associated with the network. These major drawbacks often limit the application of WNNs to low dimensions for dynamical identification problems.

3.4 Categories of Learning Algorithms

Neural networks are trained by two main types of learning algorithms: supervised and unsupervised learning algorithms. In addition, there is a third type, reinforcement learning, which can be regarded as a special form of supervised learning [4]. They can be classified as shown in Fig. 3.6:

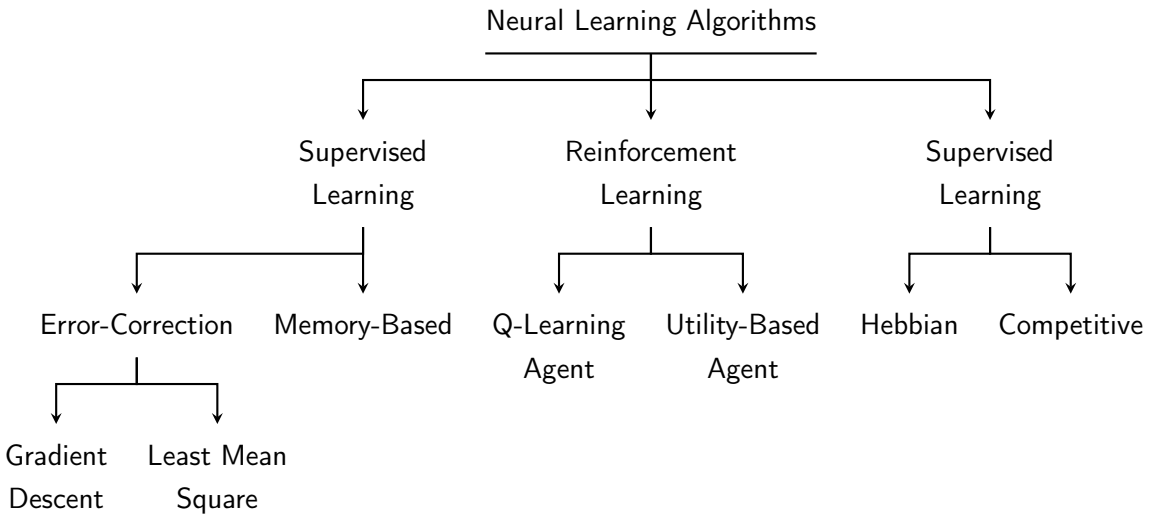


Figure 3.6: Learning Rules of Artificial Neural Networks

3.4.1 Supervised Learning

Supervised learning algorithms adjust the network parameters or weights by a direct comparison between the network's current output and the desired output. Thus, supervised learning is characterized as a closed-loop feedback system, wherein the error is the feedback signal. Also, it requires a teacher or supervisor to provide the desired or target output signals. The error measure, which shows the difference between the network response and the desired response from the training samples, is used to guide the learning process. For this reason, supervised learning is classified as error-based learning. A commonly used error measure is the mean squared error (MSE) defined as

$$E = \frac{1}{N} \sum_{p=1}^{N} \|y_p - \hat{y}_p\|^2 \tag{3.33}$$

where N is the number of pattern pairs in the training sample set, y_p is the output of the pth pattern pair, $\hat{y}_p$ is the network output corresponding to the pattern pair p and the norm distance is typically euclidean. The learning process is terminated when E is sufficiently small or a chosen error criterion is met.

Several methods are proposed in order to decrease E toward zero, the most commonly applied are derived from gradient-descent procedure. The gradient-descent method converges to a local minimum in a neighborhood of the initial solution of network parameters. The Least Mean Square (LMS) and backpropagation (BP) algorithms are two popular examples of gradient-descent based algorithms. Other examples are second-order methods, which are based on the computation of the Hessian matrix. Although the backpropagation learning algorithm is one of the most used learning approaches in feedforward neural networks, it faces several issues [13]:

- When the learning rate is too small, the convergence of the learning algorithm is very slow.

However, if it is too large, the algorithm becomes unstable and diverges.

- Another challenge is the presence of local minima in the error surface that affects the performance of the backpropagation algorithm [4]. This may result in an undesirable performance if the learning algorithm is trapped at a local minima located far from a global minima.
- Neural network are prone to overtraining and obtain worse generalization performance when using backpropagation methods. Thus, validation and suitable stopping criteria are necessary in the cost function minimization procedures.
- Gradient-based learning may require long training times. For example, the mean square error may remain high in a number of interactions and suddenly reach small values. Thus, without prior experience, it may be difficult to estimate the training time given a particular task.

The extreme learning machine (ELM) has been proposed in order to alleviate the aforementioned drawbacks for single-hidden layer neural networks. The advantages of this learning algorithm is being easy to implement, tends to reach the smallest training error, obtains the smallest norm of weights and the reasonable generalization performance and also runs extremely fast [13, 127, 128, 73].

3.4.2 Unsupervised Learning

Unsupervised learning refers to algorithms that seeks to learn structure in the absence of either an identified output or feedback. They do not require the desired outputs to be known. During training, only input patterns are presented to the neural network which automatically adapts the weights to its connections to cluster the input patterns intro groups with similar features. Thus, unsupervised learning is purely based on the correlations among the input data, and is used to find the significant patterns or features in the input data without the help of a teacher. Making it particularly suitable for biological learning in that it does not rely on a teacher and it uses intuitive primitives like neural competition and cooperation. For this reason, unsupervised learning is classified as output-based learning.

A criterion is needed to terminate the learning process. Without a stopping criterion, a learning process continues even when a pattern, which does not belong to the training patterns set, is presented to the network. The network is adapted according to a constantly changing environment. Examples of unsupervised learning algorithms include Hebbian learning, competitive learning, Adaptive Resonance Theory and the self-organizing maps (SOM). In general, unsupervised learning is slow to settle under stable conditions [4].

3.4.3 Reinforcement Learning

Reinforcement learning describes a class of computational algorithms that specifies how an artificial agent can learn to select actions in order to maximize the received reward over time. Instead of using a teacher to give target outputs, reinforcement learning is a learning procedure

that rewards the neural network for its good output result and punishes it for the bad output result [4].

As mentioned before, reinforcement learning is a special case of supervised learning, where the exact desired output is unknown. Unlike in supervised learning problems, in reinforcement-learning problems, there are no labeled examples of correct and incorrect behavior. However, unlike unsupervised learning problems, a reward signal can be perceived. Since the teacher supplies only feedback about success or failure of an answer. In some applications, it may be more plausible than supervised learning since a fully specified correct answer might not always be available to the learner or even the teacher. It is based only on the information as to whether or not the actual output is close to the estimate. Furthermore, explicit computation of derivatives is not required. However, this results in a slower learning process. In control applications, if the controller of a system still works accurately after an input, the output is labeled good; otherwise, it is judged bad. The evaluation of the binary output, also called external reinforcement, is used as the error signal. Examples of reinforcement learning algorithms are Temporal Difference (TD) learning, Monte-Carlo methods, $Q-learning$, genetic and evolutionary algorithms [129].

3.4.4 Offline and Online Identification

The previous learning methods can be applied in a online or offline fashion for system identification. For the offline case, the measured data is first stored and then the model parameters are estimated. On the other hand, the online identification is performed parallel to the experiment and the algorithm estimates the parameters of the model when new data is available during the operation of the model. It should be noted that for nonlinear systems, the system operation can vary with time or the real system input space may be different from the one which was used for offline identification. In order to achieve good identification results, both the structure and the weights of the neural network model may need to be modified in response to changes in the plant characteristics. Thus, the main advantage of online identification is that parameter values using online estimation can vary with time, but parameters estimated using offline methods do not. Hence, adaptive control is an example of such an application where it is useful to identify the model online, simultaneously with the acquisition of measurements [130].

Chapter 4

Online Neuro-Identification of Nonlinear Systems using Extreme Learning Machine

4.1 Motivation and Difference Between Neural Networks and Extreme Learning Machines

In the last decade, the extreme learning machine (ELM) approach has been proposed for training unified single hidden layer feedforward neural networks (SLFNs). In ELM, the hidden nodes are randomly initiated and remain fixed during the learning process without iteratively tuning. Moreover, the hidden nodes in ELM are not even necessary to be identical to a neuron. The only free parameters which need to be tuned are the weights between the hidden layer and the output layer. Hence, ELM is classified as a linearly parameterized model which reduces to solving a linear system. Compared to conventional feedforward neural network learning techniques, ELM is considerably efficient and turns to reach a global optimum. Theoretical studies have proved that despite the presence of randomly generated hidden nodes, ELM preserves the universal approximation capability of SLFNs [13, 71, 127].

ELM models share the same architecture of a traditional feedforward neural network, however there are some subtle differences that make the ELM models remarkably attractive compared to ANNs. In a conventional ANN model, both the input and the output layers parameters are specified in the training process. Thus, a nonlinear technique is typically required, which may have limitations such as slow iterative training and local minima trapping. ELM alleviates these limitations by eliminating the need for tuning the hidden layer parameters from the training process. As the input layer parameters are randomly generated according to a probability distribution, the training involves determining the output layer weights only. Since training involves a linear least squares problem, the achieved solution by the ELM approach is extremely fast and is a global optimum [13, 71, 127].

In spite of the aforementioned advantages, as the number of hidden neurons increase, an over-parametrized ELM suffers from ill conditioning problem when learning laws based on recursive least squares are performed [131]. This drawback may lead to an unbounded growth of the model parameters and unbounded model predictions, specially when the system is under external disturbances or unknown dynamics. For any general case, the convergence of ELM based identifier is never guaranteed to be stable. Hence it is of extreme importance to guarantee the convergence stability and approximation error boundedness for control related applications. To tackle this issue, stable online learning algorithms based on Lyapunov stability theory are developed in this chapter. The Lyapunov approach provides a stability guarantee and performs well with no undesirable parameter growth.

This chapter is organized as follows. Sections 4.2-4.3 provides a mathematical description of the ELM along with the problem formulation. In section 4.4, the identification model and the state estimate error equation is introduced followed by the stable adaptive laws for adjusting the output weights together with a Lyapunov analysis in section 4.5. Section 4.6 shows several simulation results. Finally, section 4.7 presents the summary from this chapter.

4.2 Description of Extreme Learning Machine

The output of a unified single hidden-layer feedforward neural network (SLFN) with $\tilde{N}$ hidden nodes can be represented by

$$f_{\tilde{N}}(x) = \sum_{i=1}^{\tilde{N}} \beta_i \sigma(x; w_i, \theta_i) \tag{4.1}$$

where $w_i \in \Re^n$ is the weight vector connecting the input layer to the ith hidden node, β_i is the output weight connecting the ith hidden node to the output node, θ_i is the bias of the ith hidden node with respect to the input $x \in \Re^n$, $\sigma(x; w_i, \theta_i)$ is the output of the ith hidden node with respect to the input x and $\sigma(.)$ is the activation function of ELM. Commonly used activation functions $\sigma(.)$ include sigmoid and hyperbolic tangent functions.

The standard SLFN with $\tilde{N}$ hidden nodes with activation function $\sigma(.)$ can approximate N arbitrary distinct samples $(x_k, y_k) \in \Re^n \times \Re^m$ with zero error, it then implies that there exist β_i, w_i and θ_i such that

$$\sum \beta_i \sigma(x_k; w_i, \theta_i) = y_k, \;\; k = 1, \ldots, N \tag{4.2}$$

The above equation can be written compactly as

$$\mathbf{H}\beta = \mathbf{Y} \tag{4.3}$$

where

$$\mathbf{H}(w_1,\ldots,w_{\tilde{N}},\theta_1,\ldots,\theta_{\tilde{N}},x_1,\ldots,x_N)=\begin{bmatrix}\sigma(x_1,w_1,\theta_1) & \cdots & \sigma(x_1,w_{\tilde{N}},\theta_{\tilde{N}})\\ \vdots & \ddots & \vdots\\ \sigma(x_N,w_1,\theta_1) & \cdots & \sigma(x_N,w_{\tilde{N}},\theta_{\tilde{N}})\end{bmatrix}_{N\times\tilde{N}} \tag{4.4}$$

$$\beta=\begin{bmatrix}\beta_1^T\\ \vdots\\ \beta_{\tilde{N}}^T\end{bmatrix}_{\tilde{N}\times m} \quad and \quad \mathbf{Y}=\begin{bmatrix}y_1^T\\ \vdots\\ y_N^T\end{bmatrix}_{N\times m} \tag{4.5}$$

$\mathbf{H}$ is called the hidden layer output matrix of the neural network [13, 72, 73, 74], the ith column of $\mathbf{H}$ is the ith hidden node's output vector with respect to inputs $x_1,\ldots,x_N$ and the kth row of $\mathbf{H}$ is the output vector of the hidden layer with respect to input x_k.

If the activation function $\sigma(.)$ is infinitely differentiable we can prove that the required number of hidden nodes is $\tilde{N}\leq N$. Thus, the adaptive neural identification scheme will be designed with the help of the following lemma.

Lemma 4.2.1. *(See [13]) Given a standard SLFN with $\tilde{N}$ hidden nodes and activation function $\sigma:\Re\to\Re$ which is infinitely differentiable in any interval, there exists $\tilde{N}\leq N$ such that for N arbitrary distinct samples (x_i,y_i), where $x_i\in\Re^n$ and $y_i\in\Re^m$, for any w_i and θ_i randomly chosen from any intervals of $\Re^n$ and $\Re$, respectively, according to any continuous probability distribution, then with probability one, $\|\boldsymbol{H}_{N\times\tilde{N}}\beta_{\tilde{N}\times m}-\boldsymbol{Y}_{N\times m}\|<\varepsilon$, where $\varepsilon>0$ is a small positive value.*

As discussed in [13, 72, 73, 74], the parameters of hidden neurons does not need to be tuned and can be randomly generated permanently according to any continuous probability distribution and if the activation function is infinitely differentiable. Hence (4.3) becomes a linear system and the output weights $\hat{\beta}$ are estimated as

$$\hat{\beta}=\mathbf{H}^{\dagger}\mathbf{Y} \tag{4.6}$$

where $\mathbf{H}^{\dagger}$ denotes the Moore-Penrose generalized inverse of H [13]. Since the hidden-to-output weights are determined by the Moore-Penrose generalized inverse, they are actually the least square solution for (4.3). Hence the solution obtained is extremely fast and is a global optimum for the chosen $\tilde{N}$, w_i and θ_i. This learning algorithm is known as Extreme Learning Machine (ELM) in [13, 72, 73, 74].

From [10, 81, 82], SLFNs can also be expressed mathematically in matrix format as

$$f_{\tilde{N}}(W,V,x,u)=W\sigma(Vz) \tag{4.7}$$

where $V\in\Re^{\tilde{N}\times n_1}$, n_1 the number of neurons from the input layer and $\tilde{N}$ is the number of neurons in the hidden layer, $W\in\Re^{n\times\tilde{N}}$, $\sigma\in\Re^{\tilde{N}}$, $z=[x_1,\ldots,x_n,u_1,\ldots,u_m]\in\Re^{n_1\times 1}$ and $x\in X$ is the n-dimensional state vector, $u\in U$ is a m-dimensional admissible input vector and $\sigma(.)$ is the activation function.

Remark 1. The mathematical description for unified SLFN present in (4.1) is quite common in ELM literature. However, for ease of use in stability analysis and without loss of generality, the SLFN matrix representation (4.7) will be adopted for the remaining of this chapter.

4.3 Problem Formulation

Consider the following nonlinear differential equation

$$\dot{x} = F(x, u, v, t), \quad x(0) = x_0 \tag{4.8}$$

where $x \in X$ is the n-dimensional state vector, $u \in U$ is a m-dimensional admissible input vector, $v \in V \subset \Re^q$ is a vector of time varying uncertain variables and $F : X \times U \times V \times [0, \infty) \mapsto \Re^n$ is a continuous map. In order to have a well-posed problem, we assume that X, U, V are compact sets and F is locally Lipschitzian with respect to x in $X \times U \times V \times [0, \infty)$, such that (4.8) has a unique solution.

We assume that the following can be established:

Assumption 1. *On a region* $X \times U \times V \times [0, \infty)$

$$\|d(x, u, v, t)\| \leq d_0 \tag{4.9}$$

where

$$d(x, u, v, t) = F(x, u, v, t) - f(x, u) \tag{4.10}$$

f is an unknown map, d are internal or external disturbances, and $\bar{d}_0$, such that $\bar{d}_0 > d_0 \geq 0$, is a known constant. Note that (4.9) is verified when x and u evolve on compact sets and the temporal disturbances are bounded.

Hence, except for the Assumption 1, we say that $F(x, u, v, t)$ is an unknown map and our aim is to design an online SLFN identifier with random hidden nodes and adaptive output weights which are determined by the ELM and Lyapunov synthesis for (4.8) to ensure the state error convergence, which will be accomplished despite the presence of approximation error and disturbances.

4.4 Identification Model and State Estimate Error Equation

We start by presenting the identification model and the definition of the relevant errors associated with the problem.

Now, by adding Ax to and subtracting from (4.8), where $A \in \Re^{n \times n}$ is an arbitrary Hurwitz matrix, then the system becomes

$$\dot{x} = Ax + g(x,u) + d(x,u,v,t) \tag{4.11}$$

where $g(x,u) = f(x,u) - Ax$ describes the system nonlinearity.

From Lemma 1 and by using SLFNs, the nonlinear mapping $g(x,u)$ can be replaced by $W^*\sigma(V_R z)$ plus an approximation error term $\varepsilon(x,u)$. More exactly, (4.11) becomes

$$\dot{x} = Ax + BW^*\sigma(V_R z) + B\varepsilon(x,u) + d(x,u,v,t) \tag{4.12}$$

where $B \in \Re^{n\times n}$ is a scaling matrix, $V_R \in \Re^{\tilde{N}\times n_1}$ is a randomly generated matrix according to any given continuous probability distribution and $W^* \in \Re^{n\times\tilde{N}}$ is the "optimal" or ideal matrix which can be defined as

$$W^* := \arg\min_{(W)} \left\{ \sup_{z\in\Omega_z} |W\sigma(V_R z) - g(z)| \right\} \tag{4.13}$$

where Ω_z is a predefined compact set for z, which is defined as

$$\Omega_z = \{z \in \Re^{n_1} : \|x\| \le M_z\} \tag{4.14}$$

where M_z is a positive constant specified by the designer. The approximation, reconstruction, or modeling error $\varepsilon(x,u)$ in (4.12) is a quantity that arises due to the incapacity of SLFN to match the unknown map $g(x,u)$. In general, W^* is unknown and needs to be estimated in function approximation. Let $\hat{W}$ be the estimate of W^*, respectively, and the weight estimation error be $\tilde{W} = \hat{W} - W^*$. From Lemma 1, the following can be established

Assumption 2. *On a compact set Ω_z, the ideal neural network weight and the NN approximation error are bounded by*

$$\|W^*\|_F \le w_m, \ \|\varepsilon(x,u)\| \le \varepsilon_0 \tag{4.15}$$

where w_m and ε_0 are known positive constants.

Remark 2. Assumption 1 is usual in identification or robust control literature. Assumption 2 is quite natural since g is continuous and their arguments evolve on compact sets.

Remark 3. It should be noted that W^* was defined as being the value of $\hat{W}$ that minimizes the L_∞-norm difference between $g(x,u)$ and $\hat{W}\sigma(V_R z)$. The scaling matrix B from (4.12) is introduced to manipulate the magnitude of uncertainties and hence the magnitude of the approximation error. This procedure improves the performance of the identification process.

Remark 4. It is noteworthy that in the original ELM algorithm the output weight W is adjusted based on the least-square error method. However in the proposed identification scheme, the adaptive law for updating the output weight $\hat{W}$ is derived using Lyapunov methods in order

to guarantee the stability of the entire identification process. Nonetheless, as in the original ELM algorithm, the hidden node parameters are also randomly determined.

Remark 5. Notice that the proposed neuro-identification scheme is a black-box methodology, hence the external disturbances and approximation error are related. Based on the system input and state measurements, the uncertain system (including the disturbances) is parameterized by a neural network model plus an approximation error term. However, the aim for presenting the uncertain system in the form (4.12), where the disturbance d is explicitly considered, is also to highlight that the proposed scheme is in addition valid in the presence of unexpected bounded changes in the system dynamics that can emerge, for instance, due to environment change, aging of equipment or faults.

The structure (4.12) suggests an identification model of the form

$$\dot{\hat{x}} = A\hat{x} + B\hat{W}\sigma(V_R z) - l \tag{4.16}$$

where $\hat{x}$ is the estimated state, $B \in \Re^{n\times n}$ is a positive diagonal matrix introduced to adjust the magnitude of uncertainties and hence the magnitude of the approximation error, and l is a vector function to be defined afterwards. It will be demonstrated that the identification model (4.16) used in conjunction with a convenient adjustment law for $\hat{W}$, to be proposed in the next section, ensures the convergence of the state error to a neighborhood of the origin, even in the presence of the approximation error and disturbances, whose radius depends on the design parameters.

By defining the state estimation error as $\tilde{x} := \hat{x} - x$, from (4.12) and (4.16), we obtain the state estimation error equation

$$\dot{\tilde{x}} := A\tilde{x} + B\tilde{W}\sigma(V_R z) + \Lambda - l \tag{4.17}$$

where $\tilde{W} = \hat{W} - W^*$ is the estimated parameter error for the output weight and $\Lambda = -B\varepsilon(x,u) - d(x,u,v,t)$ is a bounded residual term.

Remark 6. The sections 4.2-4.3 are motivated by [20]. However, in contrast to [20], it is considered in the model for identification a feedback function l with a time variant gain to ensure the convergence of the state errors to zero.

4.5 Adaptive Laws and Stability Analysis

We now state and prove the main theorem of this chapter.

Theorem 4.5.1. *Consider the class of general nonlinear systems described by (4.8) which satisfies Assumptions 1-2, the identification model (4.16) with*

$$l = \frac{\gamma_0 \tilde{x}}{\lambda_{min}(K)[\|\tilde{x}\| + \gamma_1 exp(-\gamma_2 t)]} \tag{4.18}$$

Let the weight adaptation law be given by

$$\dot{\hat{W}} = -\gamma_w \left[\|\tilde{x}\|(\hat{W} - W_0) + BK\tilde{x}\sigma(V_R z) \right] \tag{4.19}$$

$\gamma_w > 0$, W_0 is a constant matrix, V_R are the randomly generated hidden layer weights and K is a matrix such that

$$K = P + P^T \tag{4.20}$$

where P is a positive definite matrix. Then, the signal errors $\tilde{x}$ and $\tilde{W}$ are bounded. In addition, if $\gamma_0 > \alpha_0$ where α_0 is a positive constant, then $lim_{t\to\infty}\tilde{x}(t) = 0$.

Proof. Consider the Lyapunov function candidate

$$\bar{V} = \tilde{x}^T P\tilde{x} + \frac{\|\tilde{W}\|_F^2}{2\gamma_w} \tag{4.21}$$

By evaluating the time derivative of (4.21) along the trajectories of (4.17) and (4.19), we obtain

$$\begin{aligned} \dot{\bar{V}} &= \dot{\tilde{x}}^T P\tilde{x} + \tilde{x}^T P\dot{\tilde{x}} + \frac{tr\left(\tilde{W}^T\dot{\tilde{W}}\right)}{\gamma_w} \\ &= \left(\tilde{x}^T A^T + \hat{\sigma}^T\tilde{W}^T B^T - \Lambda^T\right) P\tilde{x} - \tilde{x}^T Kl \\ &\quad + \tilde{x}^T P\left(A\tilde{x} + B\tilde{W}\hat{\sigma} - \Lambda\right) + \frac{tr\left(\tilde{W}^T\dot{\tilde{W}}\right)}{\gamma_w} \end{aligned} \tag{4.22}$$

where $\dot{\tilde{W}} = \dot{\hat{W}}$, since $\dot{W}^* = 0$.

By using (4.20) and the Lyapunov equation $A^T P + PA = -Q$, the Lyapunov derivative can be written as

$$\begin{aligned} \dot{\bar{V}} =& -\tilde{x}^T Q\tilde{x} + \tilde{x}^T KB\tilde{W}\hat{\sigma} - \tilde{x}^T K\Lambda - \tilde{x}^T Kl \\ & - tr\left\{\tilde{W}^T\left[\|\tilde{x}\|\left(\hat{W} - W_0\right) + BK\tilde{x}\hat{\sigma}^T\right]\right\} \end{aligned} \tag{4.23}$$

Furthermore, by using the following representations

$$tr\left\{\tilde{W}^T BK\tilde{x}\hat{\sigma}^T\right\} = \tilde{x}^T KB\tilde{W}\hat{\sigma} \tag{4.24}$$

and further rearranging terms, (4.23) implies

$$\dot{\bar{V}} = -\tilde{x}^T Q\tilde{x} - \tilde{x}^T K\Lambda - \tilde{x}^T Kl - tr\left\{\tilde{W}^T\left[\|\tilde{x}\|\left(\hat{W} - W_0\right)\right]\right\} \tag{4.25}$$

By considering the facts

$$
\begin{aligned}
&2tr\left[\tilde{W}^T\left(\hat{W}-W_0\right)\right] = \|\tilde{W}\|_F^2 + \left\|\left(\hat{W}-W_0\right)\right\|_F^2 - \|(W^*-W_0)\|_F^2 \\
&\|\Lambda(t)\| \leq \Lambda_0
\end{aligned}
\tag{4.26}
$$

where $\Lambda_0 > 0$, (4.26) results

$$
\begin{aligned}
\dot{\tilde{V}} \leq & -\lambda_{min}(Q)\|\tilde{x}\|^2 - 0.5\left\|\tilde{W}\right\|_F^2\|\tilde{x}\| \\
& - \frac{\left[(\gamma_0-\alpha_0)\|\tilde{x}\| - \gamma_1\alpha_0 exp(-\gamma_2 t)\right]}{\|\tilde{x}\| + \gamma_1 exp(-\gamma_2 t)}\|\tilde{x}\|
\end{aligned}
\tag{4.27}
$$

where $\lambda_{min}(Q)$ denotes the minimum eigenvalue of Q and $\alpha_0 = \Lambda_0\|K\|_F + 0.5\|W^* - W_0\|_F^2$.

At first, note that (4.27) implies

$$
\dot{\tilde{V}} \leq -\lambda_{min}(Q)\|\tilde{x}\|^2 - 0.5\left\|\tilde{W}\right\|_F^2\|\tilde{x}\| + \alpha_0\|\tilde{x}\| \tag{4.28}
$$

or, by completing the square,

$$
\dot{\tilde{V}} = -\lambda_{min}(Q)\left(\|\tilde{x}\| - \frac{\alpha_0}{2\lambda_{min}(Q)}\right)^2 - 0.5\left\|\tilde{W}\right\|_F^2\|\tilde{x}\| + \frac{\alpha_0^2}{4\lambda_{min}(Q)} \tag{4.29}
$$

Hence, $\dot{\tilde{V}} < 0$ outside the compact set $\Omega_0 = \left\{\left(\tilde{x},\tilde{W}\right) \mid \|\tilde{x}\| \leq \alpha_{\tilde{x}} \text{ or } \left\|\tilde{W}\right\|_F \leq \alpha_{\tilde{W}}\right\}$ where $\alpha_{\tilde{x}} = \alpha_0/\lambda_{min}(Q)$ and $\alpha_{\tilde{W}} = (\alpha_0/2\lambda_{min}(Q))^{1/2}$. Thus, since $\alpha_{\tilde{x}}$ and $\alpha_{\tilde{W}}$ are positive constants, by employing usual Lyapunov arguments [62], we concluded that all error signals are uniformly bounded.

Define now

$$
\Omega = \left\{\left(\tilde{x},\tilde{W}\right) \mid \|\tilde{x}(t)\| \leq \eta exp(-\gamma_2 t), \eta = \gamma_1\alpha_0/(\gamma_0-\alpha_0)\right\} \tag{4.30}
$$

Note that the numerator in the bracket of (4.27) is greater than zero for $\|\tilde{x}\| > \eta exp(-\gamma_2 t)$ (or $\tilde{x} \in \Omega^c$), hence

$$
\dot{\tilde{V}} \leq -\lambda_{min}(Q)\|\tilde{x}\|^2 \tag{4.31}
$$

Further, since $\bar{V}$ is bounded from below and non-increasing with time, we have

$$
lim_{t\to\infty}\int_0^t \|\tilde{x}(\tau)\|^2 d\tau \leq \frac{\bar{V}(0)-\bar{V}_\infty}{\lambda_{min}(Q)} < \infty \tag{4.32}
$$

where $lim_{t\to\infty}\bar{V}(t) = \bar{V}_\infty < \infty$. Notice that, based on (4.17), with the bounds on $\tilde{x}$, $\tilde{W}$ and l, $\dot{\tilde{x}}$ is also bounded. Thus, $\dot{\tilde{V}}$ is uniformly continuous. Hence, by applying the Barbalat's Lemma [62], we conclude that $lim_{t\to\infty}\tilde{x}(t) = 0$ for all $\tilde{x} \in \Omega^c$.

Once the synchronization error $\tilde{x}(t)$ has entered Ω, it will remain in Ω forever, due to (4.30) and (4.31). Consequently, we conclude that $lim_{t\to\infty}\tilde{x}(t) = 0$ holds in the large, i.e., whatever the initial value of $\left(\tilde{x}(t), \tilde{W}(t)\right)$ (inside or outside Ω). □

Remark 7. It should be noted that resizing unknown nonlinearities has a positive impact on the identification performance. The matrix B is introduced to attenuate the effect of uncertainties, as can be seen in (4.29).

4.6 Simulation

This section presents four examples to validate the theoretical results and to show effectiveness in the presence of disturbances. The examples were chosen based on their bounded and complex nature, with promising applications in engineering. In all simulations, Solver ode23 (Bogacki-Shampine) of Matlab/Simulink®, with a relative tolerance of 1e-6 was used to obtain the numerical solutions. Also, the time in which the exponential function in (4.18) is turned off was set to $t_s = 3s$ for all cases. First, the identification of a Chen system under disturbances using the proposed methodology is achieved, after it is considered a hyperchaotic finance system submitted to the presence of disturbances. An example of complex hyperchaotic system with 7 states is identified to show the feasibility of the ELM model for higher dimensional systems. Finally, a proposed algorithm in the literature [1] is used here for comparison.

4.6.1 Chen System

Consider the unified chaotic system [132], which is described by

$$\begin{aligned}
\dot{x} &= (25\alpha + 10)(y - x) + d_x \\
\dot{y} &= (28 - 35\alpha)x - xz + (29\alpha - 1)y + d_y \\
\dot{z} &= xy - \left(\frac{8+\alpha}{3}\right) z + d_z
\end{aligned} \tag{4.33}$$

where x, y and z are state variables and is always chaotic in the whole interval $\alpha \in [0, 1]$ and d_x, d_y and d_z are unknown disturbances. It should be also noted that system (4.33) becomes the Lorenz system for $\alpha = 0$, Chen system for $\alpha = 1$. In particular, system (4.33) bridges the gap between the Lorenz system and the Chen system. In the following simulation, we consider the Chen system.

To identify the chaotic system (4.33), the proposed identification model (4.16) and the adaptive law (4.19) were implemented. It should be noted that V_R is randomly assigned and remains fixed through the simulation. The design parameters were chosen as $\gamma_w = 0.001$, $\gamma_0 = 0.1$, $\gamma_1 = 1$, $\gamma_2 = 0.00001$, the sigmoid function is $\sigma(.) = 100/1 + exp - 1(.)$, the design matrices are

$$A = \begin{bmatrix} -10.5 & 0 & 0 \\ 0 & -9 & 0 \\ 0 & 0 & -9.5 \end{bmatrix}, B = 10^2 \times \begin{bmatrix} 4.1 & 0 & 0 \\ 0 & 3.9 & 0 \\ 0 & 0 & 4 \end{bmatrix} \tag{4.34}$$

with $W_0 = 0$ and $P = 0.5 \times I_{3\times 3}$, where I is the identity matrix. The chosen inputs are

$$z = \begin{bmatrix} x & y & z & x^2 & y^2 & z^2 & 1 \end{bmatrix} \tag{4.35}$$

The chosen initial conditions for the system are $x(0) = -1.5$, $y(0) = -2$, $z(0) = -5$, $\hat{x}(0) = 5$, $\hat{y}(0) = 5$, $\hat{z}(0) = 5$ and $\hat{W} = 0$. To check the robustness of the proposed method, it is considered the presence of the following time-dependent disturbance

$$d(x, u, v, t) = \begin{bmatrix} 3sin(7t)\|(x, y, z)\| \\ 10cos(9t)x \\ cos(20t) + 10exp(-t) \end{bmatrix} \tag{4.36}$$

for the system (4.33). We consider the emergence, at $t = 5s$, of disturbances of the form (4.36).

The performances in the estimation of state variable are shown in Fig. 4.1-4.3. It can be seen that the simulations confirm the theoretical results, that is, the algorithm is stable and the residual state error is small. The Frobenius norms associated to the estimated weight matrix W is shown in Fig. 4.4. After a transient phase, due to the large initial uncertainty, this norm seems to converge, indicating that most of the state estimation error has been removed. It should be noted that the transient can be shaped according to the user's desired convergence.

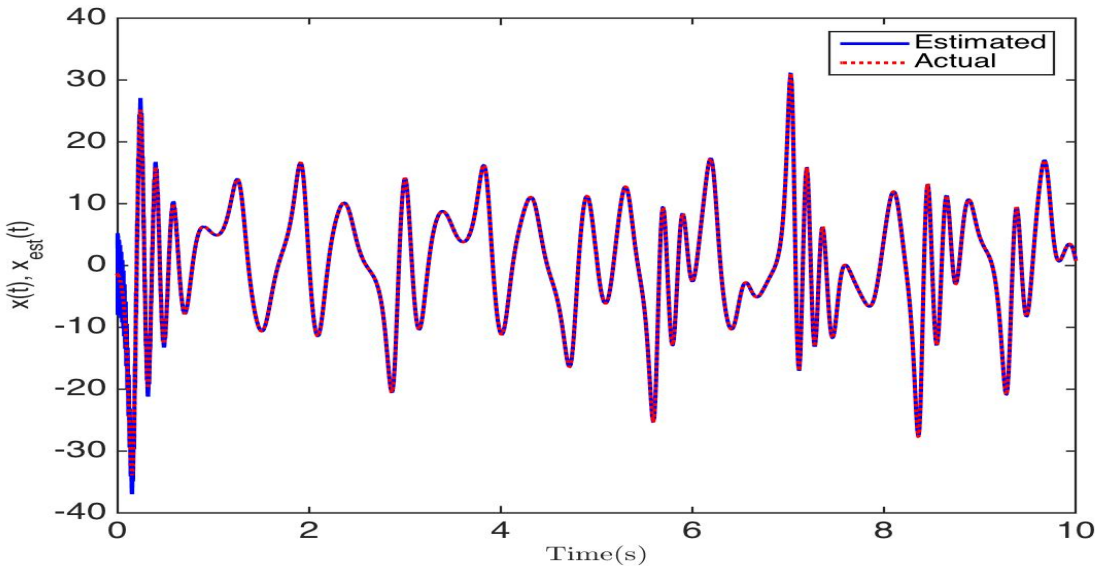


Figure 4.1: Performance in the estimation of x

In the simulations of the proposed algorithm, the design matrices A, B, P and K were initially chosen as identity matrices. In the sequence, these values were adjusted, by a trial and error procedure, to fulfill requirements of performance (such as small residual state error and transient). As a remark, the design parameters can be appropriately chosen so as to suit the requirements on overshoot and settling time of the parameter estimation. It was selected appropriate gains for

the design parameters to force a fast tracking; however, the transient can be adjusted arbitrarily depending on the project requirements. This gives additional flexibility and control on the proposed model's performance.

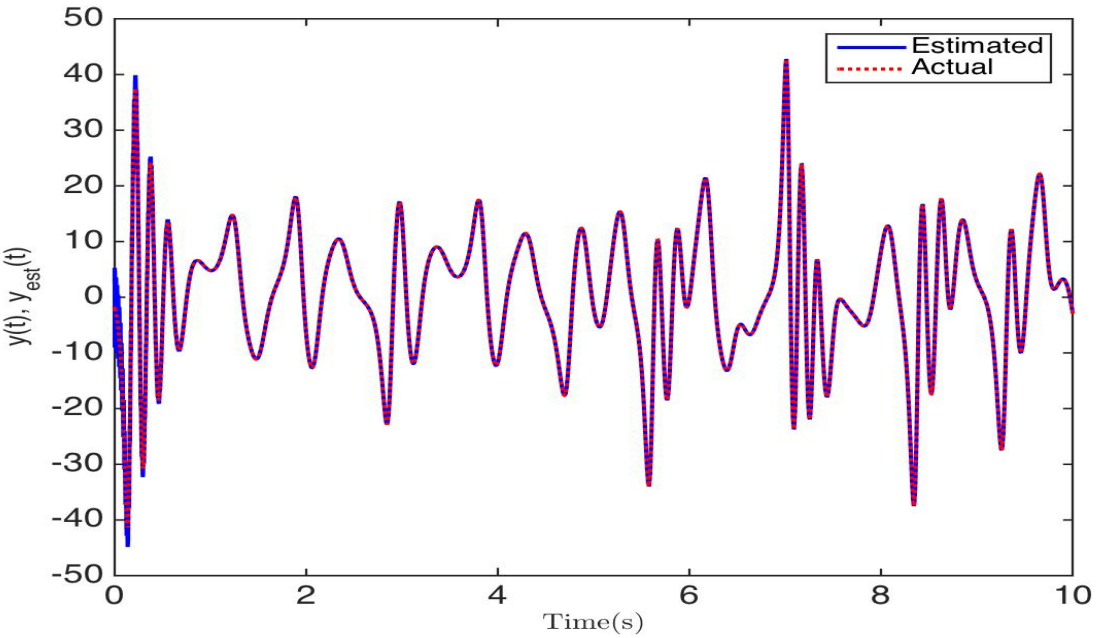


Figure 4.2: Performance in the estimation of y

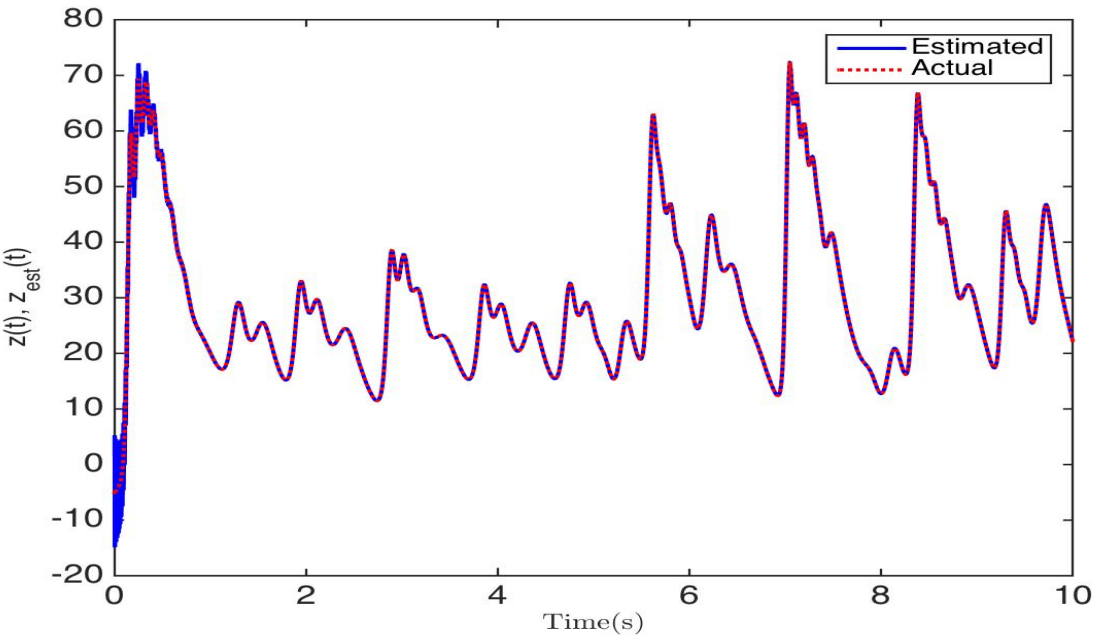


Figure 4.3: Performance in the estimation of z

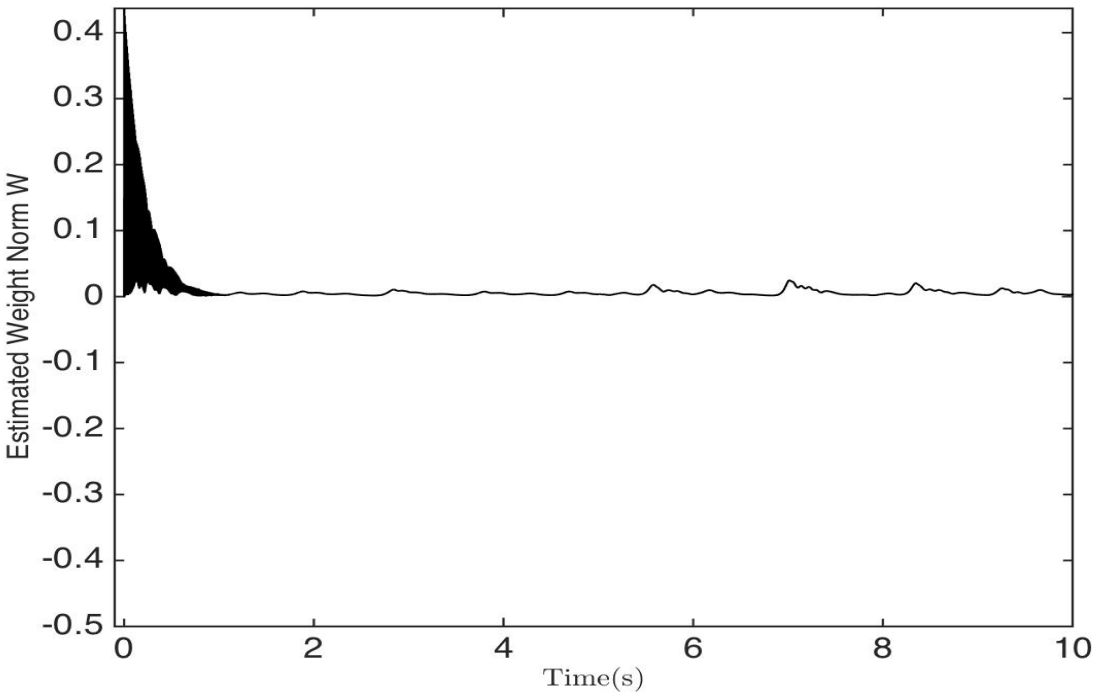


Figure 4.4: Frobenius norm of the estimated weight matrix W

From Fig. 4.4, it can be concluded that the identification scheme is robust in the presence of disturbances and unknown dynamics, handling abrupt perturbations with practically no degradation of performance.

4.6.2 Hyperchaotic Finance System

Consider a hyperchaotic finance system described by [133]

$$\begin{aligned} \dot{x} &= z + (y - a)x + u + d_x \\ \dot{y} &= 1 - by - x^2 + d_y \\ \dot{z} &= -x - cz + d_z \\ \dot{u} &= -dxy - ku - d_u \end{aligned} \tag{4.37}$$

where a, b, c, d and k are constant parameters and d_x, d_y, d_z and d_u are unknown disturbances. It was considered that $a = 0.9$, $b = 0.2$, $c = 1.2$, $d = 0.2$ and $k = 0.17$. Notice that system (4.37) satisfies assumption 1, since the state variables evolve on compact sets.

To identify the uncertain system (4.37), the proposed identification model (4.16) and the adaptive law (4.19) were implemented. The design parameters were chosen as $\gamma_w = 0.01$, $\gamma_0 = 0.1$, $\gamma_1 = 1$, $\gamma_2 = 0.00001$, the sigmoid function is $\sigma(.) = 150/1 + exp - 1(.)$, the matrices parameters are

$$A = \begin{bmatrix} -29 & 0 & 0 & 0 \\ 0 & -30.5 & 0 & 0 \\ 0 & 0 & -30 & 0 \\ 0 & 0 & 0 & -28 \end{bmatrix}, B = 10^2 \times \begin{bmatrix} 1.1 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 \\ 0 & 0 & 1.2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.38}$$

with $W_0 = 0$ and $P = 0.05 \times I_{4\times4}$, where I is the identity matrix. The chosen inputs are

$$z = \begin{bmatrix} x & y & z & x^2 & y^2 & z^2 & 1 \end{bmatrix} \tag{4.39}$$

The chosen initial conditions for the system are $x(0) = 1$, $y(0) = 2.5$, $z(0) = 0.5$, $u(0) = 0.5$, $\hat{x}(0) = -2$, $\hat{y}(0) = -2$, $\hat{z}(0) = -2$, $\hat{u}(0) = -2$ and $\hat{W} = 0$. To check the robustness of the proposed method, it is considered the presence of the following time-dependent disturbance

$$d(x, u, v, t) = \begin{bmatrix} 8sin(7t)u \\ 10cos(9t)x^2 \\ (cos(20t) + 5exp(-t))y \\ 2sin(20t) + 3cos(15t) \end{bmatrix} \tag{4.40}$$

We consider the emergence, at $t = 5s$, of disturbances of the form (4.40).

The performances in the estimation of state variable are shown in Fig. 4.5-4.8. It can be seen that the simulations confirm the theoretical results, that is, the algorithm is stable and the residual state error is small. The Frobenius norms associated to the estimated weight matrix W is shown in Fig. 4.9. One can find that the system outputs achieved by the proposed neural controller still asymptotically track the desired trajectories well.

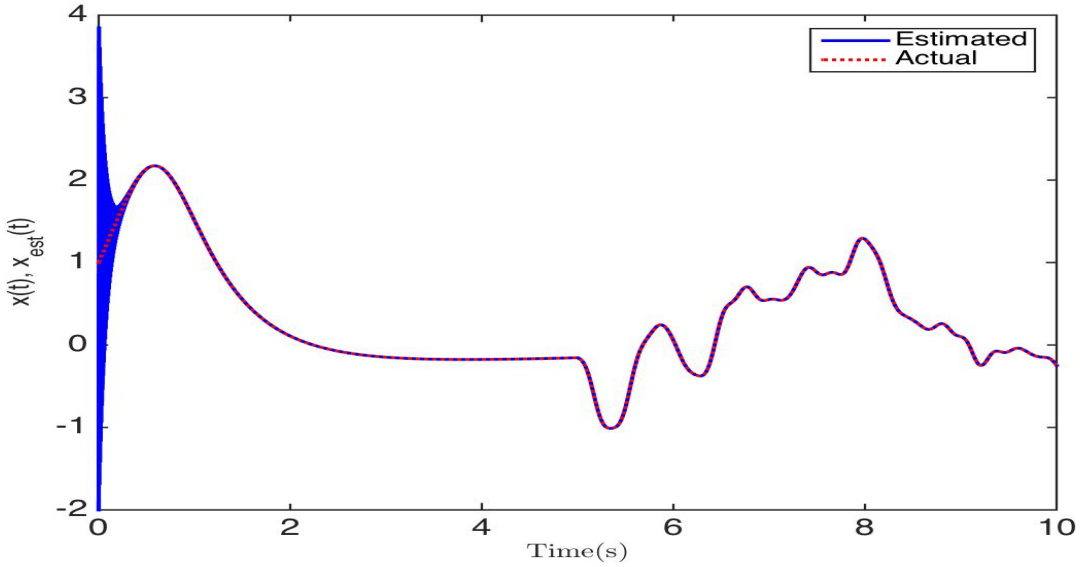


Figure 4.5: Performance in the estimation of x

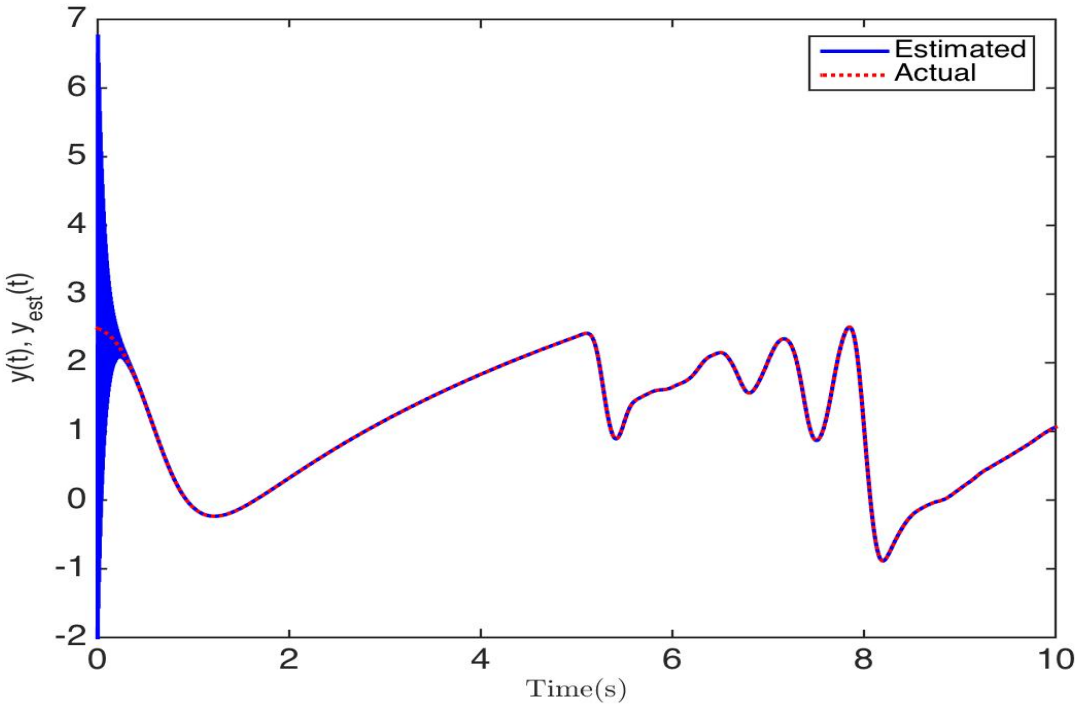


Figure 4.6: Performance in the estimation of y

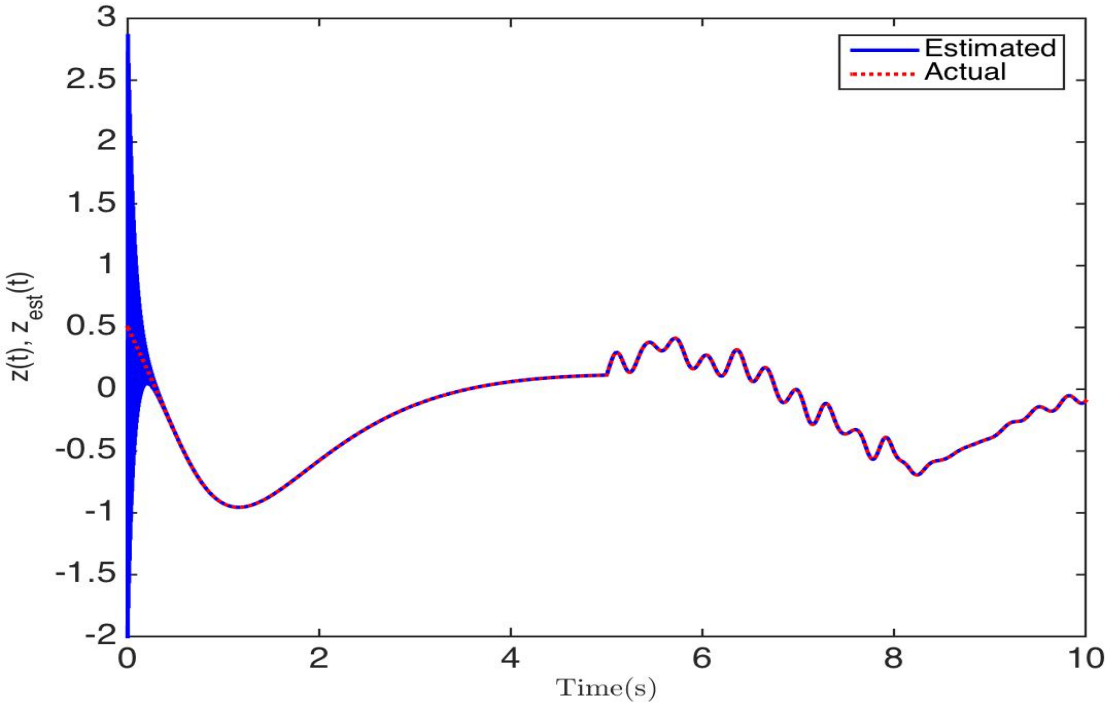


Figure 4.7: Performance in the estimation of z

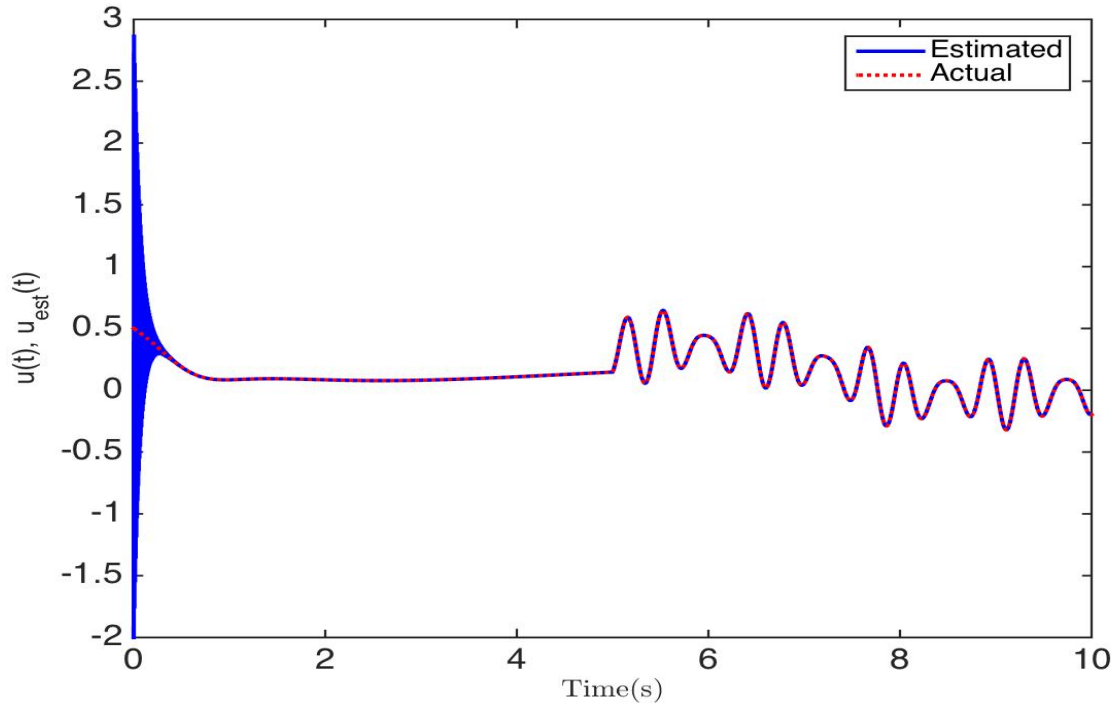


Figure 4.8: Performance in the estimation of u

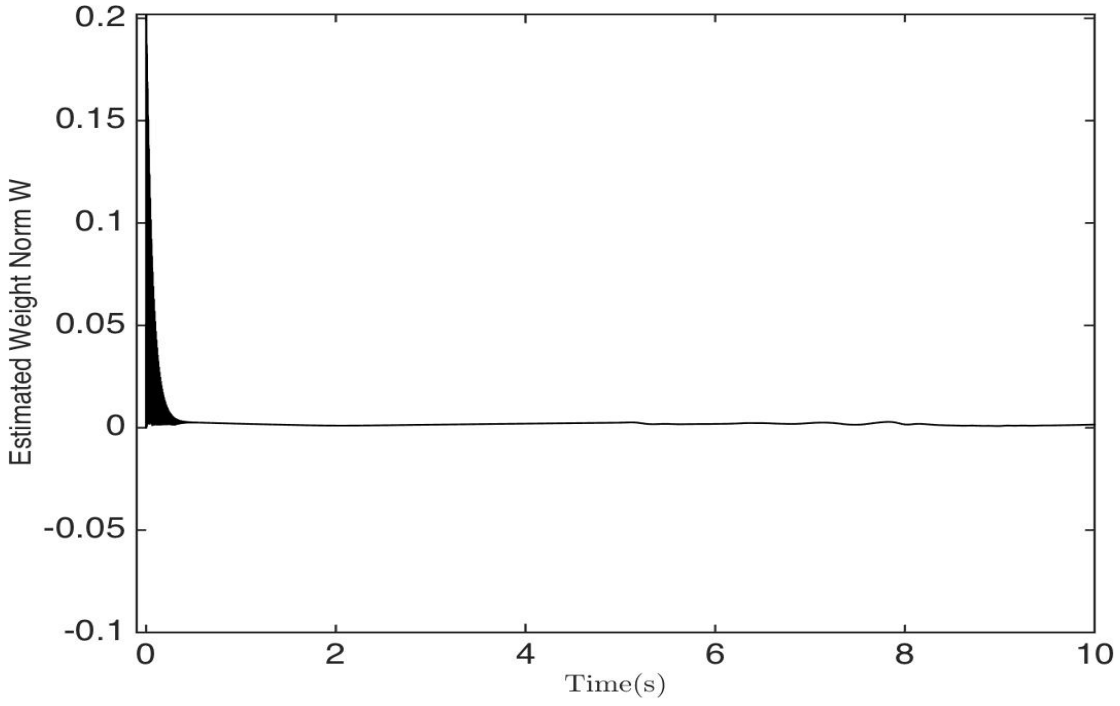


Figure 4.9: Frobenius norm of the estimated weight matrix W

4.6.3 Hyperchaotic System

Consider a hyperchaotic system described by [134]

$$\begin{aligned}
\dot{x}_1 &= \alpha(x_3 - x_1) + d_{x1} \\
\dot{x}_2 &= \alpha(x_4 - x_2) + d_{x2} \\
\dot{x}_3 &= \gamma x_1 - x_1 x_5 - x_3 + x_6 + d_{x3} \\
\dot{x}_4 &= \gamma x_2 - x_2 x_5 - x_4 + x_7 + d_{x4} \\
\dot{x}_5 &= x_1 x_3 + x_2 x_4 - \beta x_5 + d_{x5} \\
\dot{x}_6 &= k_1 x_1 + k_2 x_3 + d_{x6} \\
\dot{x}_7 &= k_1 x_2 + k_2 x_4 + d_{x7}
\end{aligned} \tag{4.41}$$

where α, β, γ, k_1 and k_2 are constant parameters and d_{x1}, d_{x2}, d_{x3}, d_{x4}, d_{x5}, d_{x6} and d_{x7} are unknown disturbances. It was considered that $\alpha = 14$, $\beta = 3$, $\gamma = 50$, $k_1 = -5$ and $k_2 = -4$. Notice that system (4.41) satisfies assumption 1, since the state variables evolve on compact sets.

To identify the uncertain system (4.41), the proposed identification model (4.16) and the adaptive law (4.19) were implemented. The design parameters were chosen as $\gamma_w = 0.1$, $\gamma_0 = 0.1$, $\gamma_1 = 1$, $\gamma_2 = 0.00001$, the sigmoid function is $\sigma(.) = 150/1 + exp - 0.1(.)$, the matrices parameters are

$$A = -16 \times \begin{bmatrix} 1.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, B = 50 \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.8 \end{bmatrix} \tag{4.42}$$

with $W_0 = 0$ and $P = 0.05 \times I_{7\times 7}$, where I is the identity matrix. The chosen inputs are

$$z = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & 1 \end{bmatrix} \tag{4.43}$$

The chosen initial conditions for the system are $x_1(0) = 0$, $x_2(0) = 1$, $x_3(0) = 2$, $x_4(0) = 3$, $x_5(0) = 4$, $x_6(0) = 5$, $x_7(0) = 6$, $\hat{x}_1(0) = -20$, $\hat{x}_2(0) = -30$, $\hat{x}_3(0) = -40$, $\hat{x}_4(0) = 20$, $\hat{x}_5(0) = 40$, $\hat{x}_6(0) = 50$, $\hat{x}_7(0) = 40$ and $\hat{W} = 0$. To check the robustness of the proposed method, it is considered the presence of the following time-dependent disturbance

$$d(x,u,v,t) = 2\|x\| \begin{bmatrix} sin(t)u \\ 1.2sin(2t) \\ cos(4t) \\ 1.2sin(t) \\ 1.1sin(2t) \\ 0.5sin(4t) \\ exp(-0.5t) \end{bmatrix} \tag{4.44}$$

where $x = [x_1\ x_3\ x_3\ x_4\ x_5\ x_6\ x_7]^T$. We consider the emergence, at $t = 5s$, of disturbances of the form (4.44).

The performances in the estimation of state variable are shown in Fig. 4.10-4.16. It can be seen that the simulations confirm the theoretical results, that is, the algorithm is stable and the residual state error is small. The Frobenius norms associated to the estimated weight matrix W is shown in Fig. 4.17. One can find that the system outputs achieved by the proposed neural controller still asymptotically track the desired trajectories well.

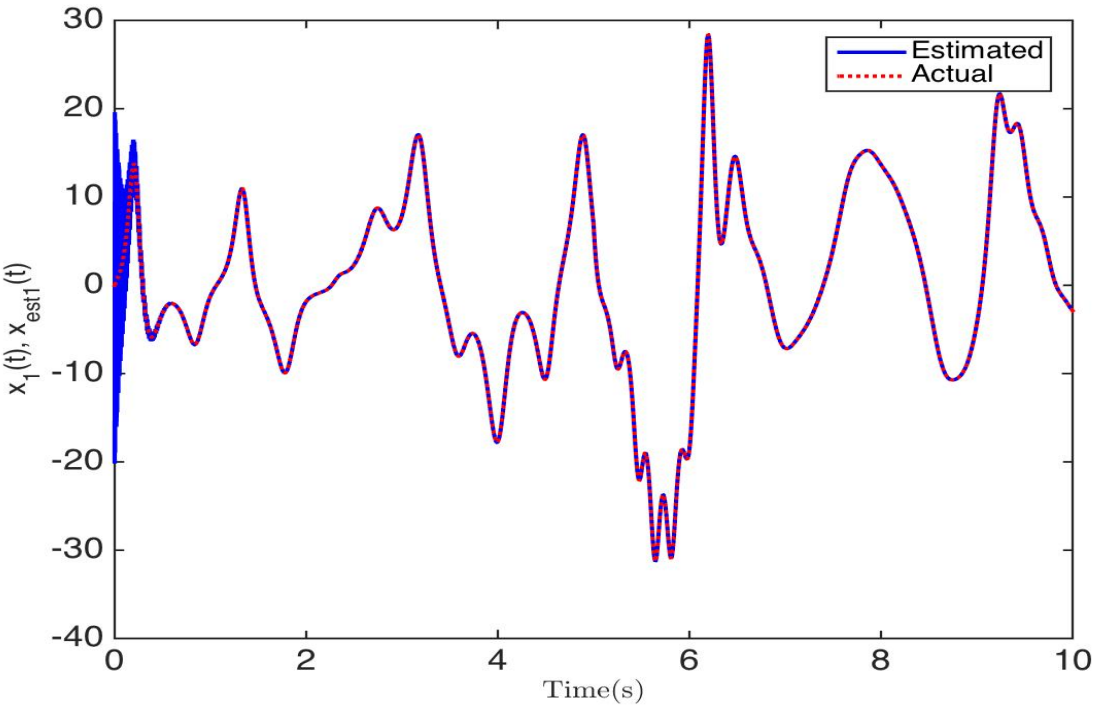


Figure 4.10: Performance in the estimation of x_1

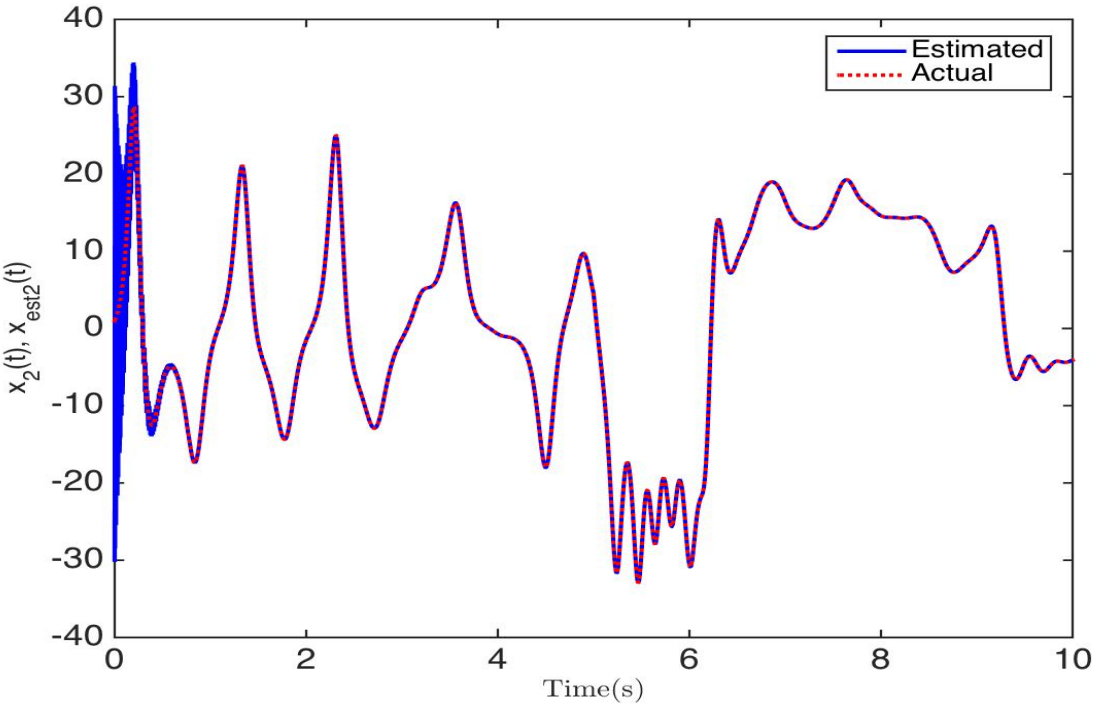


Figure 4.11: Performance in the estimation of x_2

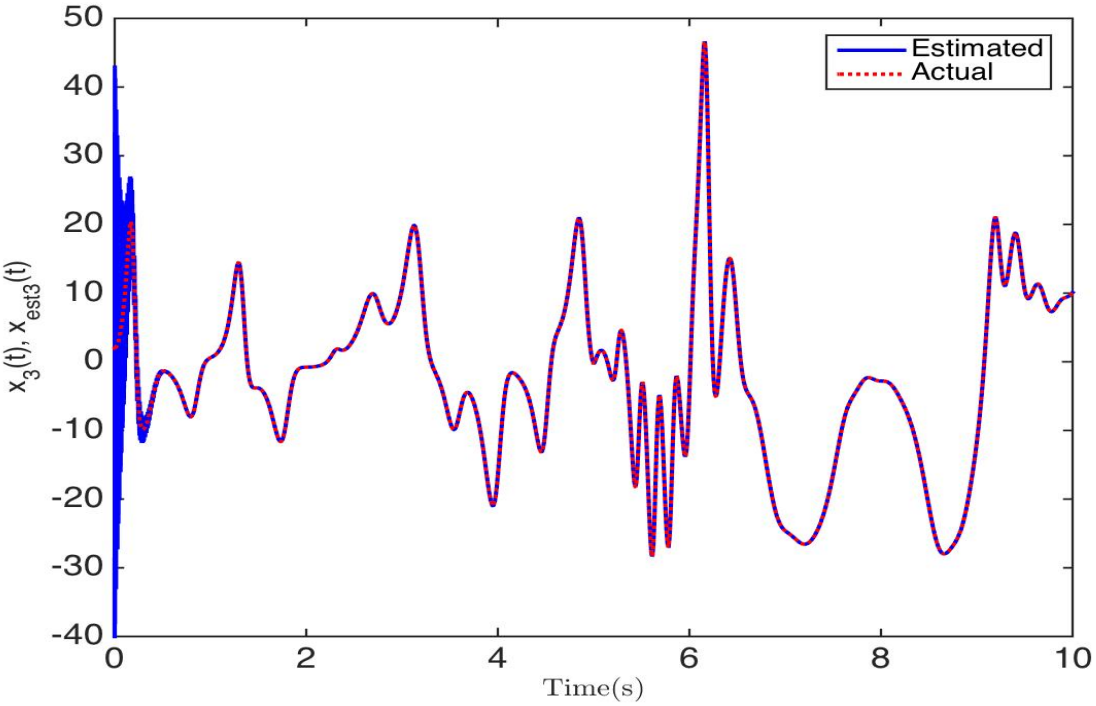


Figure 4.12: Performance in the estimation of x_3

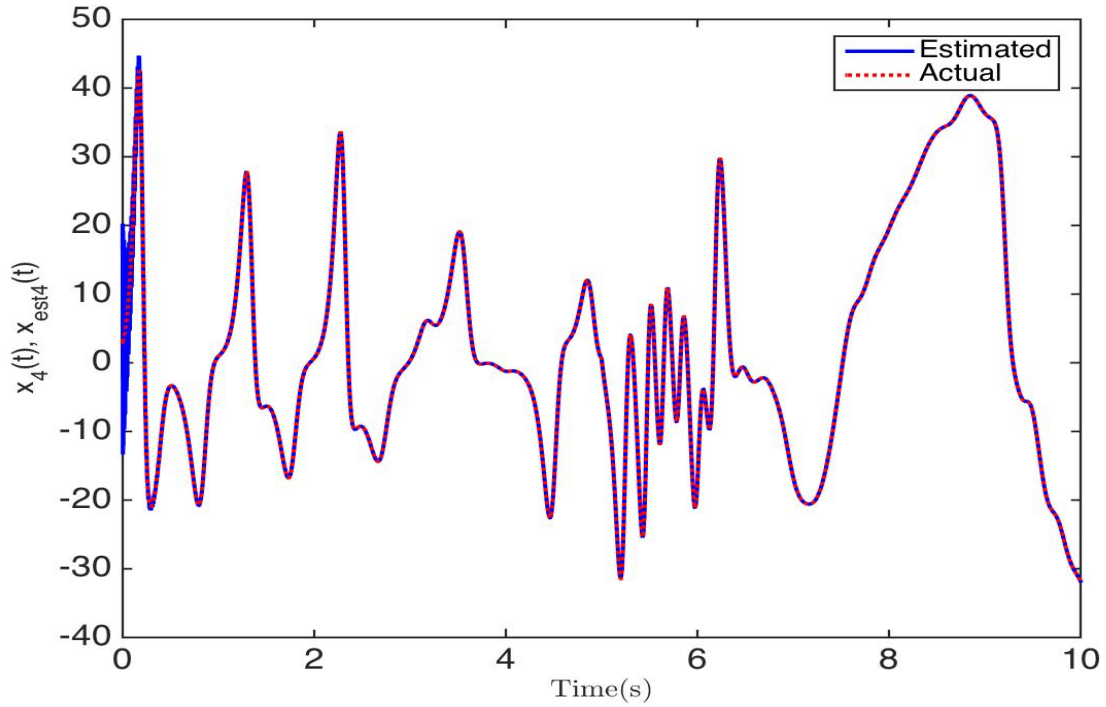


Figure 4.13: Performance in the estimation of x_4

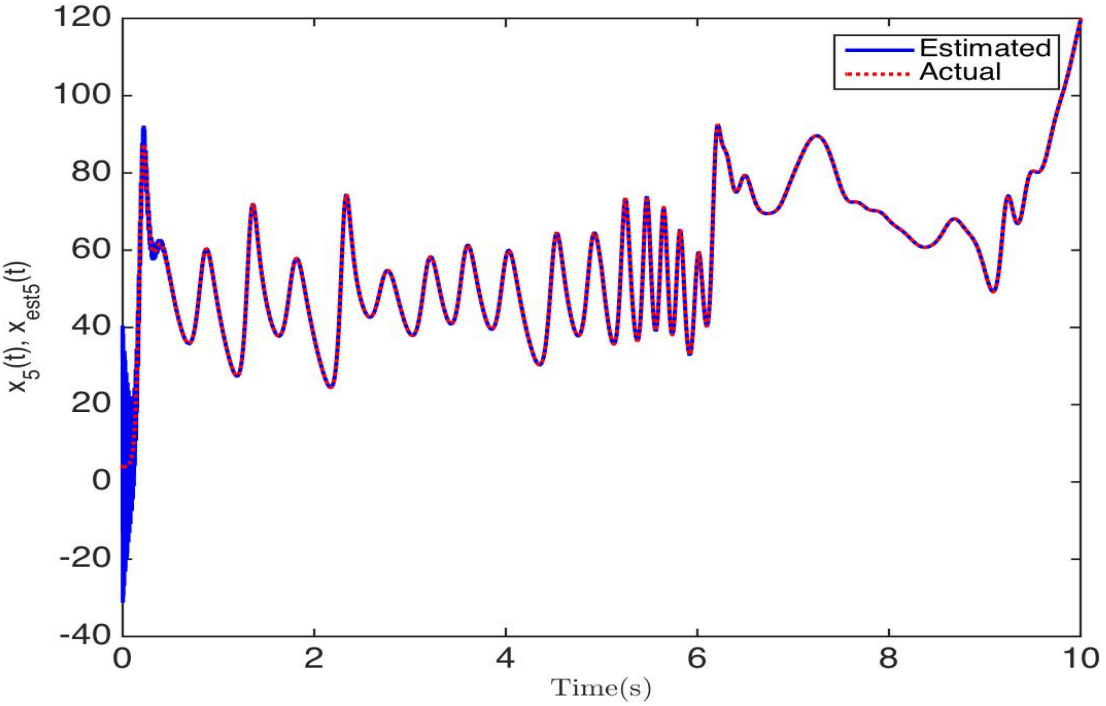


Figure 4.14: Performance in the estimation of x_5

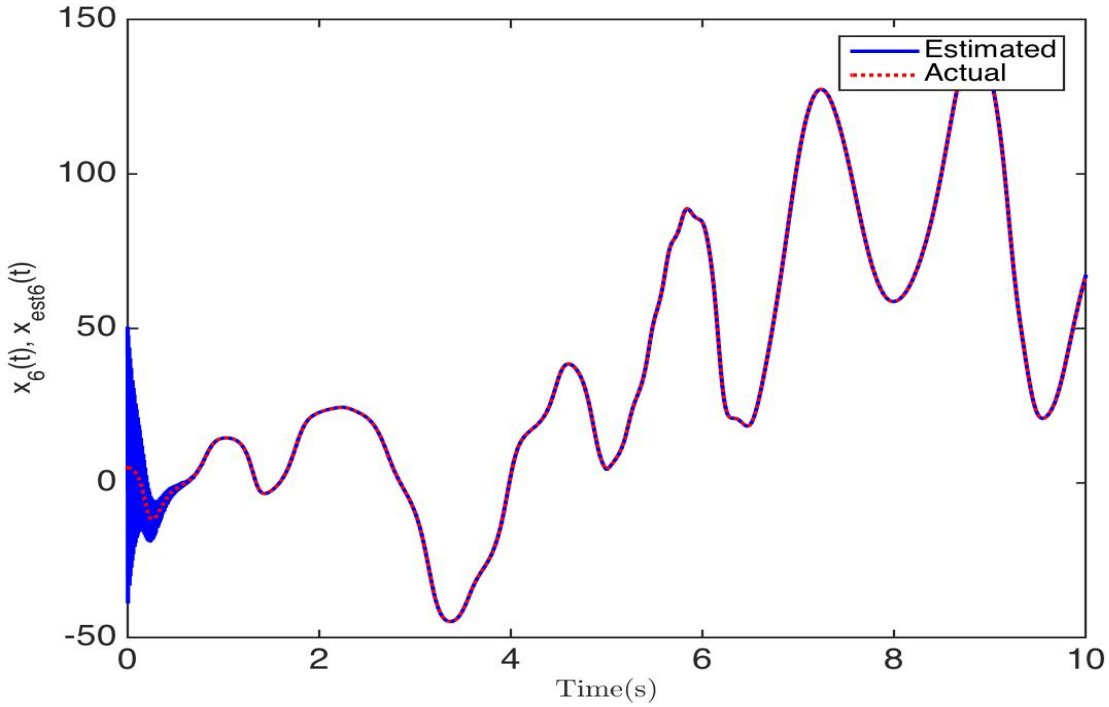


Figure 4.15: Performance in the estimation of x_6

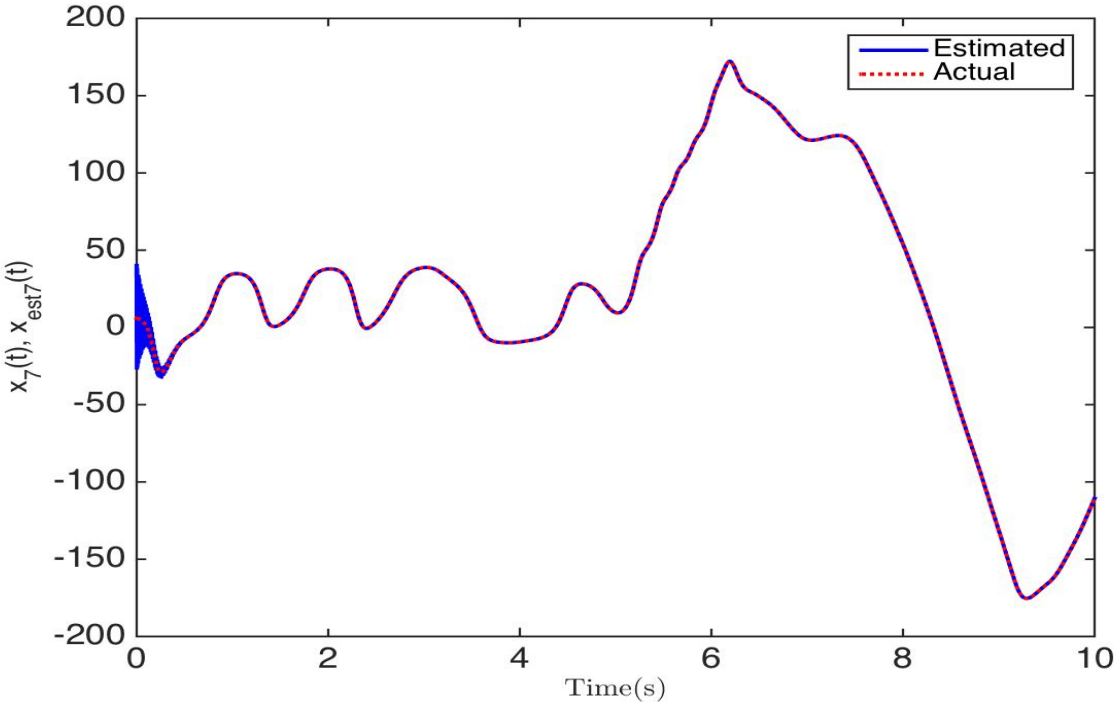


Figure 4.16: Performance in the estimation of x_7

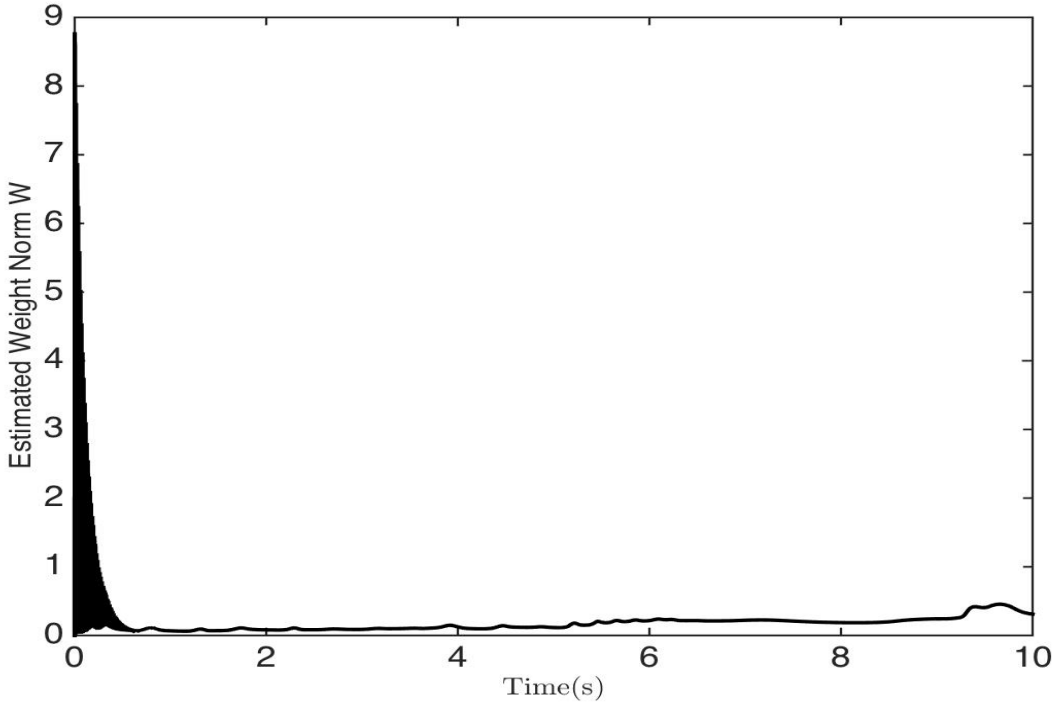


Figure 4.17: Frobenius norm of the estimated weight matrix W

4.6.4 Comparison with Ref. [1]

To illustrate the advantages of the proposed methodology, the identification model introduced in [1] is used here for comparison. Consider the Lyapunov based ELM algorithm proposed in [1] described as

$$\begin{aligned} \dot{\hat{z}} &= A\hat{z}(t) + \hat{W}^T\phi \\ \dot{\hat{W}} &= -\phi e^T \end{aligned} \tag{4.45}$$

where the state vector $z \in \Re^{n\times 1}$, ϕ is the activation function for the regressor vector $V_R z$ and V_R the hidden layer weight matrix with random values. The Lorenz system is considered, where $\alpha = 0$ in (4.33). The sigmoid function is $\sigma(.) = 150/1 + exp - 0.001(.)$. The design matrix A is chosen as [1], where

$$A = \begin{bmatrix} -60 & 0 & 0 \\ 0 & -60 & 0 \\ 0 & 0 & -120 \end{bmatrix} \tag{4.46}$$

The chosen initial conditions for the system are $x(0) = 1$, $y(0) = 2.5$, $z(0) = 0.5$, $\hat{x}(0) = 5$, $\hat{y}(0) = 5$, $\hat{z}(0) = 5$ and $\hat{W} = 0$. It was stated in [1] that were chosen 12 inputs, as it was not specified, they are chosen here as

$$z = [x \quad y \quad z \quad x^2 \quad y^2 \quad z^2 \quad xy \quad xz \quad yz \quad xyz \quad x^2y \quad x^2z] \tag{4.47}$$

The initial conditions and design parameters of the proposed algorithm of this chapter for this simulation are the same as in 4.6.1. To check the robustness of the proposed method, it is considered the presence of the following time-dependent disturbance for the comparison

$$d(x,u,v,t) = \|w\| \begin{bmatrix} 3sin(7t)u \\ 10cos(9t) \\ (cos(20t) + 10exp(-t))y \end{bmatrix} \tag{4.48}$$

where $w = [x \ y \ z]^T$. We consider the emergence, at $t = 5s$, of disturbances of the form (4.40).

Figs. 4.18-4.20 show the state error norms comparisons for each state variable. It should be pointed out that the adjustment of the design parameters in [1] was not trivial given the parameter drift, mainly due to the lack of tuning parameters in the learning law $\hat{W}$ for compensating this drawback. Furthermore, it seems that the absence of a robustifying term for the adaptive parameter law in [1] has a negative impact on the performance when the system is under external disturbance, as can be seen in Figs. 4.18-4.22

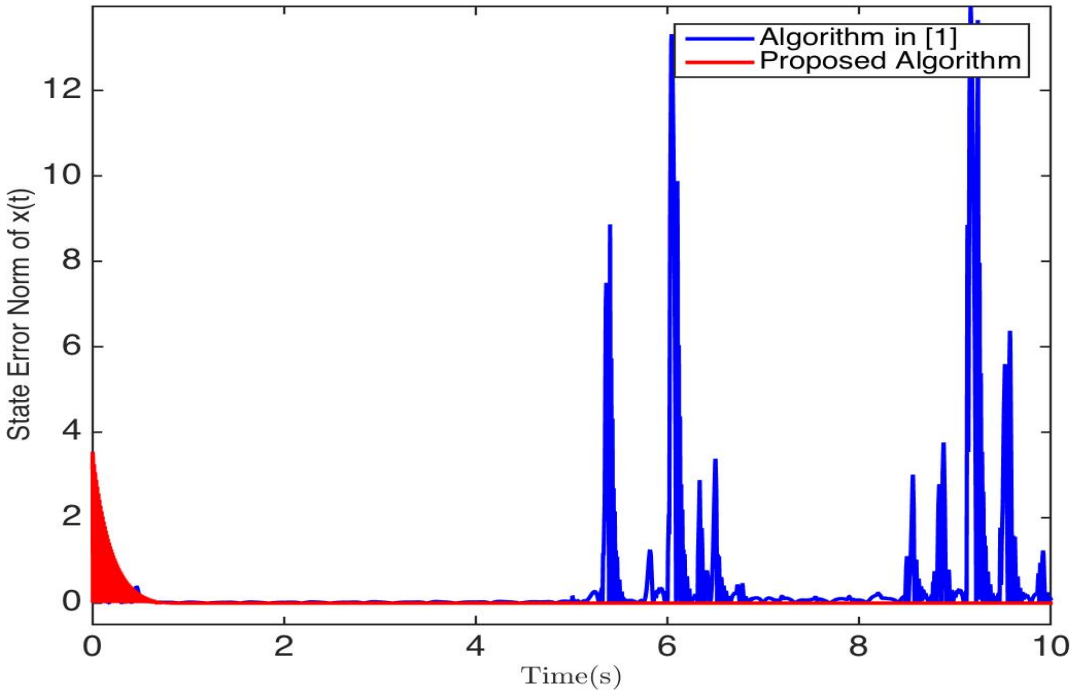


Figure 4.18: Performance comparison in the estimation of x

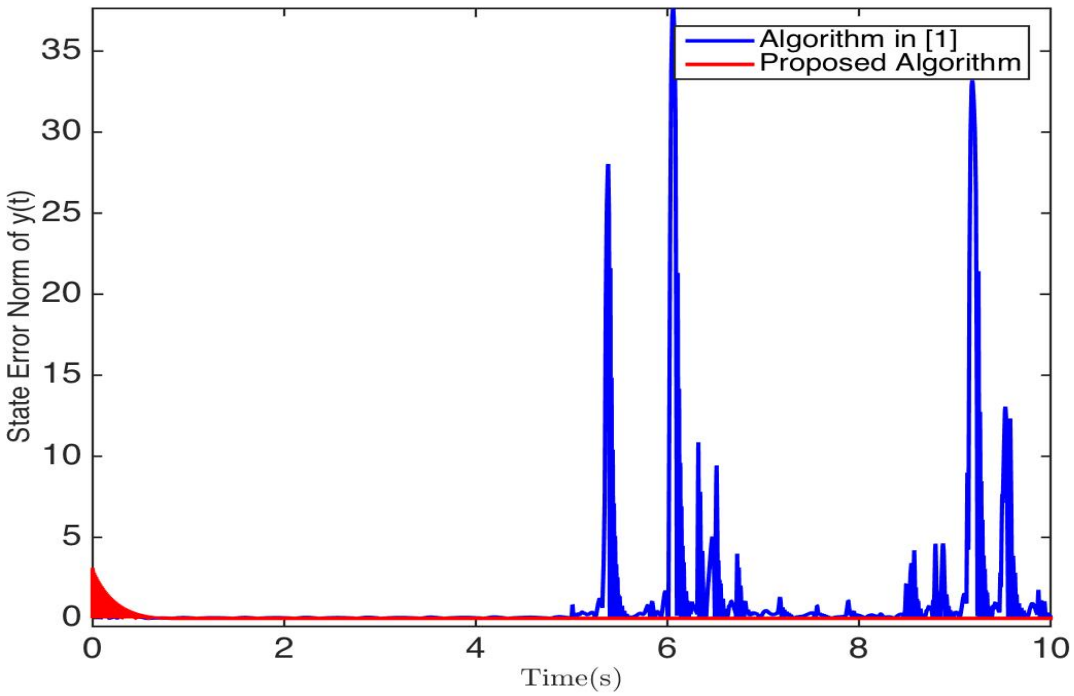


Figure 4.19: Performance comparison in the estimation of y

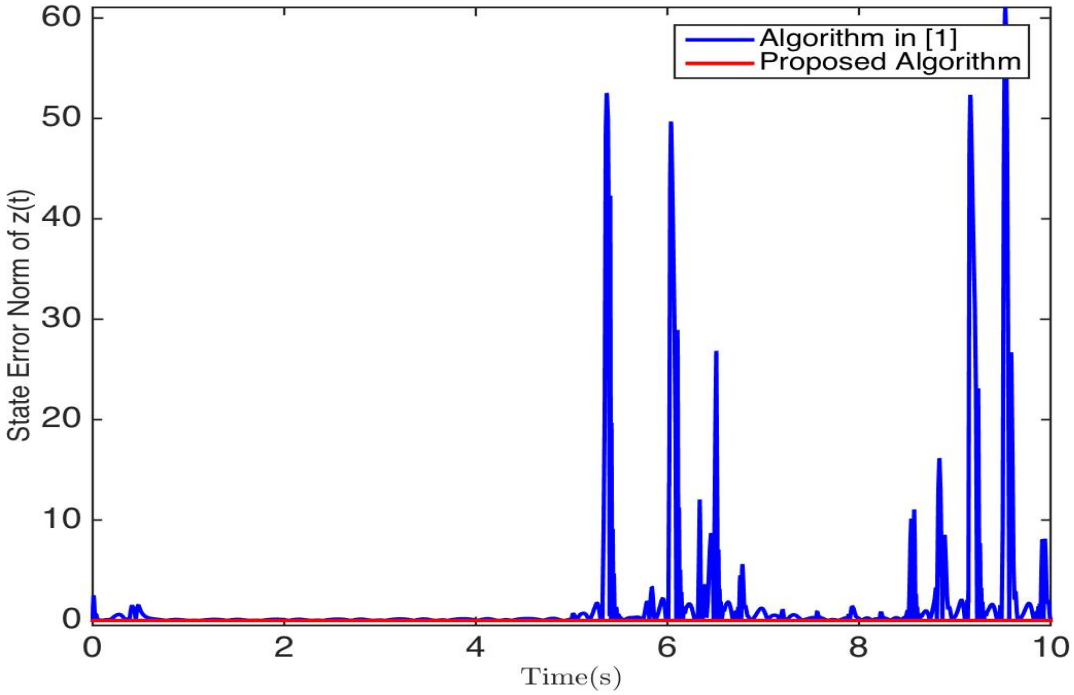


Figure 4.20: Performance comparison in the estimation of z

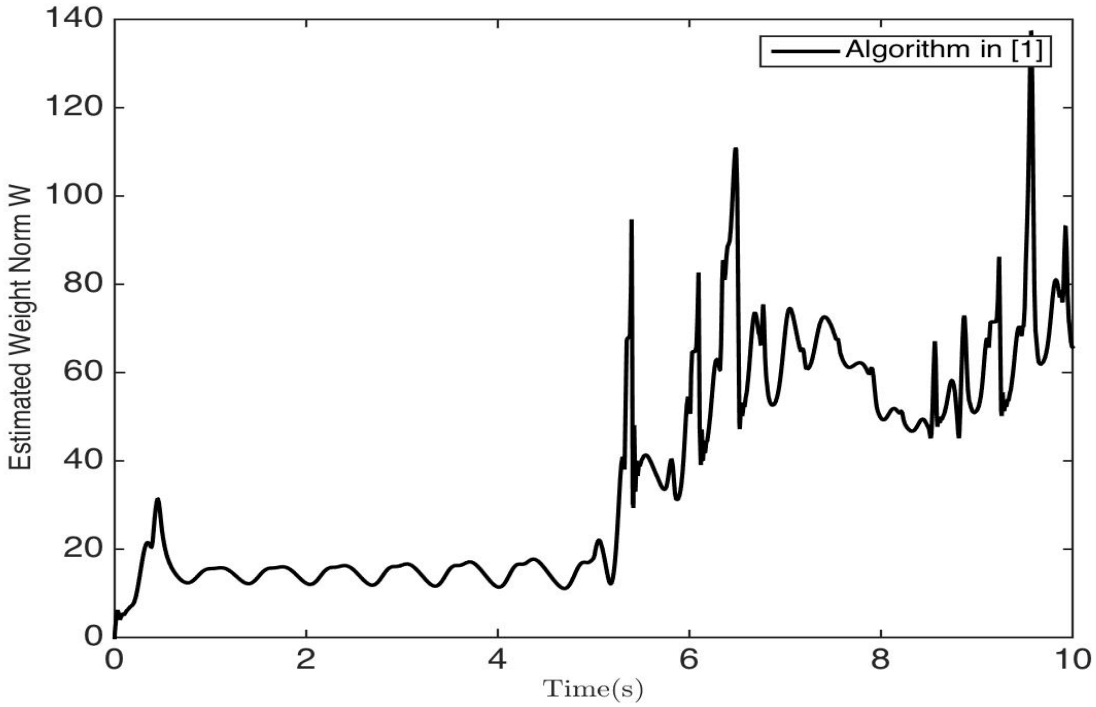


Figure 4.21: Frobenius norm of the estimated weight matrix W

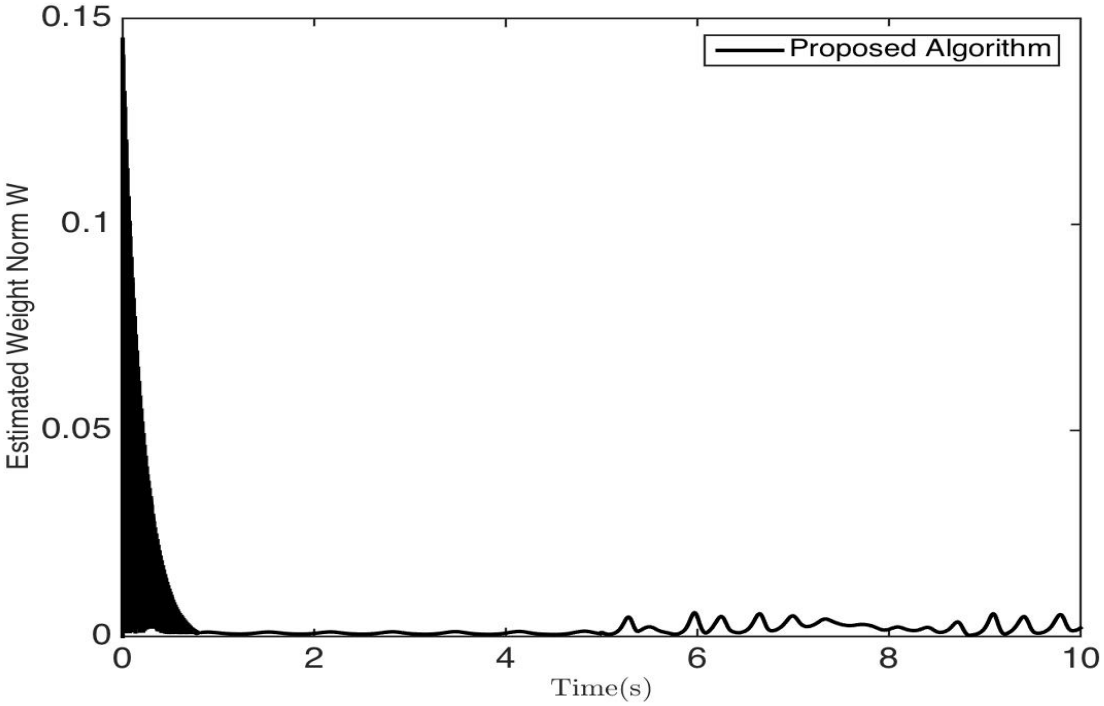


Figure 4.22: Frobenius norm of the estimated weight matrix W

4.7 Summary

In this chapter, by using neural networks and extreme learning technique, an online adaptive neuro-identification scheme in the presence of unknown dynamics and external disturbances is proposed. The hidden node parameters of the SLFN are randomly generated using the extreme learning machine (ELM) algorithm, and the output weights of the SLFN are updated based on the stable adaptive laws derived from Lyapunov analysis. The proposed adaptive learning law ensures the convergence of the residual state estimation errors to zero. Furthermore, it is not necessary any previous knowledge about the ideal weight, approximation error and disturbances. The proposed methodology also combines computational efficiency in terms of learning speed from the ELM technique with the stability of the system under disturbances guaranteed by the Lyapunov analysis.

The first two simulation results demonstrated the effectiveness and performance of the proposed approach in the presence of disturbances. Additionally, to show the efficiency of the proposed learning algorithm for high-dimensional dynamical systems, a simulation for a complex hyperhaotic system is demonstrated without compromising the learning speed and generalization ability, easing the "curse of dimensionality" usual in linearly parameterized neural networks. Finally, a comparison of the proposed algorithm with that in [1] was performed to show the advantages and peculiarities of the proposed method under disturbances. The results obtained and presented in this chapter have been reported in [135].

Chapter 5

Identification of Unknown Nonlinear Systems based on Multilayer Neural Networks

5.1 Motivation

Feedforward multilayer neural networks are of particular interest in this Master's thesis, since it has been shown that with enough hidden units and under certain conditions they are capable of approximating any nonlinear mapping [80, 81, 86]. Moreover, linearly parametrized neural networks typically suffers the so-called "curse of dimensionality", where as the input dimension of the system increases the number of nodes demanded to approximate nonlinear mappings increases exponentially. Thus, the computational demands, both in memory and computational time, can be significantly high for multiple-input multiple-output systems.

Also, nonlinearly parametrized neural networks provide greater approximation power than linearly parametrized models. For instance, the author in [136, pp. 11] shows that for certain classes of functions, single-hidden layer neural network models with a sigmoid activation function can achieve a given approximation accuracy with a number of nodes that is linearly dependent on the dimension of the input vector. Thus, the above properties make SHLNNs well worth for investigating its application for system identification of nonlinear systems.

This chapter is organized as follows. Sections 5.2-5.3 give a brief review of the SHLNNs along with the problem formulation. In section 5.4, the design procedure of the adaptive neural identifier and the state estimate error equation is introduced followed by the stable adaptive laws for adjusting the output weights based on a Lyapunov synthesis approach in section 5.5. Section 5.6 shows the simulation results. Finally, section 5.8 presents the summary from this chapter.

5.2 Single Hidden Layer Neural Networks

A class of multilayer NNs used here can be expressed mathematically in matrix form as

$$g_{nn}(W, V, x, u) = W\sigma(Vz) \tag{5.1}$$

where $V \in \Re^{n_2 \times (n_1+1)}$, n_1 is the number of neurons from the input layer, n_2 is the number of neurons in the hidden layer, $W \in \Re^{n \times n_2}$, $\sigma \in \Re^{n_2}$ is a basis function vector and $z = [x_1, \ldots, x_n, u_1, \ldots, u_m, 1] \in \Re^{n_1+1}$, where $n_1 = n + m$. In standard SHLNNs each entry of $\sigma(.)$ is a linear combination of either an external input or the state passed through a scalar activation function $s(.)$. Commonly used $s(.)$ are the sigmoid and hyperbolic tangent function [10].

Universal approximation results in [10, 67] indicate that:

Property 1. *Let $s(.)$ be a nonconstant, bounded and monotone increasing continuous function. Let Ω_z be a compact subset of $\Re^n$ and $g(z)$ be a real valued continuous function on Ω_z. Then for any arbitrary $\mu > 0$, there exists an integer n_2 and ideal matrices W^* and V^* such that*

$$\max_{z \in \Omega_z} |g(z) - g_{nn}(W^*, V^*, z)| < \mu \tag{5.2}$$

Based on Property 1, we have

$$g(z) = W^*\sigma(V^*z) + \varepsilon(z) \tag{5.3}$$

with $\varepsilon(z)$ satisfying $max_{z \in \Omega_z} |\varepsilon(z)| < \mu$, $\forall z \in \Omega_z$.

5.3 Problem Formulation

Consider the following nonlinear differential equation

$$\dot{x} = F(x, u, v, t), \quad x(0) = x_0 \tag{5.4}$$

where $x \in X$ is the n-dimensional state vector, $u \in U$ is a m-dimensional admissible input vector, $v \in V \subset \Re^q$ is a vector of time varying uncertain variables and $F : X \times U \times V \times [0, \infty) \mapsto \Re^n$ is a continuous map. In order to have a well-posed problem, we assume that X, U, V are compact sets and F is locally Lipschitzian with respect to x in $X \times U \times V \times [0, \infty)$, such that (5.4) has a unique solution.

We assume that the following can be established:

Assumption 1. *On a region $X \times U \times V \times [0, \infty)$*

$$\|d(x,u,v,t)\| \leq d_0 \tag{5.5}$$

where

$$d(x,u,v,t) = F(x,u,v,t) - f(x,u) \tag{5.6}$$

f is an unknown map, d are internal or external disturbances, and $\bar{d}_0$, such that $\bar{d}_0 > d_0 \geq 0$, is a known constant. Note that (5.5) is verified when x and u evolve on compact sets and the temporal disturbances are bounded.

Hence, except for the Assumption 1, we say that $F(x,u,v,t)$ is an unknown map and our aim is to design a NNs-based identifier for (5.4) to ensure the state error convergence, which will be accomplished despite the presence of approximation error and disturbances.

5.4 Identification Model and State Estimate Error Equation

We start by presenting the identification model and the definition of the relevant errors associated to the problem.

By adding and subtracting Ax, where $A \in \Re^{n\times n}$ is an Hurwitz matrix, (5.4) can be rewritten as

$$\dot{x} = Ax + g(x,u) + d(x,u,v,t) \tag{5.7}$$

where $g(x,u) = f(x,u) - Ax$.

By using SHLNNs, the nonlinear mapping $g(z)$ can be replaced by $W^*\sigma(V^*z)$ plus an approximation error term $\varepsilon(x,u)$. More exactly, (5.4) becomes

$$\dot{x} = Ax + BW^*\sigma(V^*z) + B\varepsilon(x,u) + d(x,u,v,t) \tag{5.8}$$

where $B \in \Re^{n\times n}$ is a scaling matrix, $W^* \in \Re^{n\times n_2}$ and $v^* \in \Re^{n_2\times(n_1+1)}$ are the "optimal" or ideal matrices, which can be defined as

$$(W^*, V^*) := \arg\min_{(W,V)} \left\{ \sup_{z\in\Omega_z} |W\sigma(Vz) - g(z)| \right\} \tag{5.9}$$

Let $\hat{W}$ and $\hat{V}$ be the estimates of W^* and V^*, respectively, and the weight estimation errors be $\tilde{W} = \hat{W} - W^*$ and $\tilde{V} = \hat{V} - V^*$. From (5.3), the following can be established

Assumption 2. *On a compact set Ω_z, the ideal neural network weight and the NN approximation error are bounded by*

$$\|W^*\|_F \leq w_m, \|V^*\|_F \leq v_m, \ \|\varepsilon(x,u)\| \leq \varepsilon_0 \tag{5.10}$$

where w_m, v_m and ε_0 are known positive constants.

Remark 1. Assumption 1 is usual in identification. Assumption 2 is quite natural since g is continuous and their arguments evolve on compact sets.

Remark 2. It should be noted that W^* and V^* were defined as being the values of $\hat{W}$ and $\hat{V}$ that minimize the L_∞-norm difference between $g(x,u)$ and $\hat{W}\sigma(\hat{V}z)$. The scaling matrix B from (5.8) is introduced to manipulate the magnitude of uncertainties and hence the magnitude of the approximation error. This procedure improves the performance of the identification process.

Remark 3. As multilayer neural networks are classified as nonlinear in the parameters approximators, since the hidden layer weight V appears in a nonlinear fashion, it is far more complex to derive the learning algorithms in comparison with the linear in the parameters models. When applying function approximators such as $\hat{W}\sigma\left(\hat{V}z\right)$ for solving the identification problem, it is desired to have a linearly parameterized form in terms of $\tilde{W}$ and $\tilde{V}$. One approach [10] is by applying the Taylor series expansion of $\sigma(V^*z)$ about $\hat{V}z$, where we have

$$\sigma(V^*z) = \sigma(\hat{V}z) - \hat{\sigma}'\tilde{V}z + \Theta \tag{5.11}$$

where $\hat{\sigma}' = \frac{\partial\sigma(V^*z)}{\partial V^*z}\Big|_{\hat{V}z}$ and Θ represents the high order terms in the Taylor expansion.

The structure (5.8) suggests an identification model of the form

$$\dot{\hat{x}} = A\hat{x} + B\hat{W}\sigma(\hat{V}z) - l_0\tilde{x} - l \tag{5.12}$$

where $l_0 > 0$, l is a vector function to be defined afterwards, $\hat{x}$ is the estimated state and $\tilde{x} := \hat{x} - x$ is the state estimation error. It will be demonstrated that the identification model (5.12) used in conjunction with convenient adjustment laws for $\hat{W}$ and $\hat{V}$ to be proposed in the next section, ensures the convergence of the state error to a neighborhood of the origin, even in the presence of the approximation error and disturbances, whose radius depends on some design parameters.

From (5.8) and (5.12), we obtain the state estimation error equation

$$\dot{\tilde{x}} = A\tilde{x} + B\hat{W}\sigma(\hat{V}z) - BW^*\sigma(V^*z) - B\varepsilon(x,u) - d(x,u,v,t) - l_0\tilde{x} - l \tag{5.13}$$

From (4.11), we can rewrite the state estimation error equation as

$$\dot{\tilde{x}} = A\tilde{x} + B\tilde{W}\sigma(\hat{V}z) + B\hat{W}\hat{\sigma}'\tilde{V}z - B\tilde{W}\hat{\sigma}'(\hat{V}z)\tilde{V}z - \Lambda - l_0\tilde{x} - l \tag{5.14}$$

where $\Lambda = -BW^*\Theta - B\varepsilon(x,u) - d(x,u,v,t)$ is a bounded residual term.

5.5 Adaptive Laws and Stability Analysis

We now state and prove the main theorem of this chapter.

Theorem 5.5.1. *Consider the class of general nonlinear systems described by (5.4), which satisfies Assumptions 1-2, the identification model (5.12) with*

$$l = \frac{\gamma_0 \tilde{x}}{\lambda_{min}(K)[\|\tilde{x}\| + \gamma_1 exp(-\gamma_2 t)]} \tag{5.15}$$

Let the weights adaptation laws be given by

$$\dot{\hat{W}} = -\gamma_W \left[2\alpha_W \|\tilde{x}\| \left(\hat{W} - W_0 \right) + BK\tilde{x}\hat{\sigma}^T - BK\tilde{x} \left(\hat{\sigma}' \hat{V} z \right)^T \right] \tag{5.16}$$

$$\dot{\hat{V}} = -\gamma_V \left[2\alpha_V \|\tilde{x}\| \left(\hat{V} - V_0 \right) + \hat{\sigma}' \hat{W}^T BK\tilde{x} z^T \right] \tag{5.17}$$

where $\gamma_0 \geq 0$, $\gamma_1 > 0$, $\gamma_2 > 0$, $\gamma_W > 0$, $\gamma_V > 0$, $\alpha_W > 0$, $\alpha_V > 0$, W_0 and V_0 are constant matrices, K is a matrix such that

$$K = P + P^T \tag{5.18}$$

and P is a positive definite matrix. Then, if $\gamma_0 = 0$, the estimation errors $\tilde{W}$ and $\tilde{V}$ are bounded, and $\tilde{x}$ is uniformly ultimately bounded with ultimate bound $\alpha_{\tilde{x}}$. If $\gamma_0 > \alpha_3$, where $\alpha_3 > 0$, the state error converges to zero, i.e., $lim_{t\to\infty}\tilde{x}(t) = 0$.

Proof. Consider the Lyapunov function candidate

$$\bar{V} = \tilde{x}^T P \tilde{x} + \frac{\left\| \tilde{W} \right\|_F^2}{2\gamma_W} + \frac{\left\| \tilde{V} \right\|_F^2}{2\gamma_V} \tag{5.19}$$

By evaluating the time derivative of (5.19) along the trajectories of (5.14), (5.16) and (5.17), we obtain

$$\begin{aligned} \dot{\bar{V}} = & -\tilde{x}^T Q \tilde{x} + \tilde{x}^T KB\tilde{W}\hat{\sigma} - \tilde{x}^T KB\tilde{W}\hat{\sigma}'\tilde{V}z + \tilde{x}^T KB\hat{W}\hat{\sigma}'\tilde{V}z - \tilde{x}^T K\Lambda \\ & - tr\left\{ \tilde{W}^T \left[2\alpha_W \|\tilde{x}\| \left(\hat{W} - W_0 \right) + BK\tilde{x}\hat{\sigma}^T - BK\tilde{x} \left(\hat{\sigma}'\hat{V}z \right)^T \right] \right\} \\ & - tr\left\{ \tilde{V}^T \left[2\alpha_V \|\tilde{x}\| \left(\hat{V} - V_0 \right) + \hat{\sigma}'\hat{W}^T BK\tilde{x}z^T \right] \right\} - l_0\tilde{x}^2 - \tilde{x}^T Kl \end{aligned} \tag{5.20}$$

where $A^T P + PA = -Q$, Q is a Hurwitz matrix, $\dot{\tilde{W}} = \dot{\hat{W}}$ and $\dot{\tilde{V}} = \dot{\hat{V}}$, since $\dot{W}^* = 0$ and $\dot{V}^* = 0$.

Furthermore, by using the following representations

$$
\begin{aligned}
&tr\left\{\tilde{W}^T BK\tilde{x}\hat{\sigma}^T\right\} = \tilde{x}^T KB\tilde{W}\hat{\sigma} \\
&tr\left\{\tilde{W}^T BK\tilde{x}\left(\hat{\sigma}'\hat{V}z\right)^T\right\} = \tilde{x}^T KB\tilde{W}\hat{\sigma}'\hat{V}z \\
&tr\left\{\tilde{V}\hat{\sigma}'\hat{W}^T BK\tilde{x}z^T\right\} = \tilde{x}^T KB\hat{W}\hat{\sigma}'\tilde{V}z
\end{aligned}
\tag{5.21}
$$

and further rearranging terms, (5.20) results

$$
\begin{aligned}
\dot{\tilde{V}} = &- \tilde{x}^T Q\tilde{x} - \tilde{x}^T K\Lambda - tr\left\{\tilde{W}^T\left[2\alpha_W \left\|\tilde{x}\right\|\left(\hat{W} - W_0\right)\right]\right\} \\
&- tr\left\{\tilde{V}^T\left[2\alpha_V \left\|\tilde{x}\right\|\left(\hat{V} - V_0\right)\right]\right\} - l_0\tilde{x}^2 - \tilde{x}^T Kl
\end{aligned}
\tag{5.22}
$$

By considering the facts

$$
\begin{aligned}
&-\tilde{V} + \hat{V} = V^*, \\
&2tr\left[\tilde{W}^T\left(\hat{W} - W_0\right)\right] = \left\|\tilde{W}\right\|_F^2 + \left\|\left(\hat{W} - W_0\right)\right\|_F^2 - \left\|(W^* - W_0)\right\|_F^2 \\
&2tr\left[\tilde{V}^T\left(\hat{V} - V_0\right)\right] = \left\|\tilde{V}\right\|_F^2 + \left\|\left(\hat{V} - V_0\right)\right\|_F^2 - \left\|(V^* - V_0)\right\|_F^2 \\
&\left\|\hat{\sigma}'(t)\right\| \leq \sigma_{0d},\ \left\|\Lambda(t)\right\| \leq \Lambda_0,\ \left\|z(t)\right\| \leq z_0,\ \forall t \geq 0
\end{aligned}
\tag{5.23}
$$

where σ_{0d}, Λ_0 and z_0 are positive constants, and, in the sequence, by completing the square, (5.22) implies

$$
\dot{\tilde{V}} = - \left\|\tilde{x}\right\|\left[l_0 \left\|\tilde{x}\right\| + \alpha_W\left(\left\|\tilde{W}\right\|_F - \frac{\alpha_1}{2\alpha_W}\right)^2 + \alpha_W\left\|\tilde{V}\right\|_F^2 - \alpha_0\right]
\tag{5.24}
$$

where $\alpha_0 = \alpha_1^2/4\alpha_w + \alpha_2 + \alpha_w \left\|W^* - W_0\right\|_F^2 + \alpha_W \left\|V^* - V_0\right\|_F^2$, $\alpha_1 = z_0\alpha_{0d}\left\|KB\right\|_F \left\|W^*\right\|_F$ and $\alpha_2 = \Lambda_0 \left\|K\right\|_F$.

Hence, with $\dot{\tilde{V}} < 0$ outside the compact set $\Omega = \left\{\left(\tilde{x}, \tilde{W}, \tilde{V}\right) \mid \left\|\tilde{x}\right\| \leq \alpha_{\tilde{x}}\ or\ \left\|\tilde{W}_F\right\| \leq \alpha_{\tilde{W}}\ \text{or}\ \left\|\tilde{V}\right\|_F \leq \alpha_{\tilde{V}}\right\}$ where $\alpha_{\tilde{x}} = \alpha_0/l_0$, $\alpha_{\tilde{W}} = (\alpha_0/l_0)^{1/2} + \alpha_1/2\alpha_W$ and $\alpha_{\tilde{V}} = (\alpha_0/\alpha_V)^{1/2}$. Thus, since $\alpha_{\tilde{x}}$, $\alpha_{\tilde{W}}$ and $\alpha_{\tilde{V}}$ are positive constants, by employing usual Lyapunov arguments [11], we concluded that all error signals are uniformly bounded. In addition, since l_0, $\alpha_{\tilde{W}}$ and $\alpha_{\tilde{V}}$ can be arbitrarily selected, $\tilde{x}(t)$ is uniformly ultimately bounded with ultimate bound $\alpha_{\tilde{x}}$.

In case that $\gamma_0 > \alpha_3$, (5.22) implies

$$
\dot{\tilde{V}} \leq - \left[\lambda_{min}(Q) + l_0\right]\left\|\tilde{x}\right\|^2 - \frac{\alpha_3\gamma_1\left[\left\|\tilde{x}\right\| - \eta exp(\gamma_2 t)\right]}{\eta\tilde{x} + \gamma_1 exp(\gamma_2 t)}
\tag{5.25}
$$

where $\alpha_3 = \alpha_1 + \alpha_2 + \alpha_W \left\|W^* - W_0\right\|_F^2 + \alpha_V \left\|V^* - V_0\right\|_F^2$ and $\eta = \gamma_1\alpha_3/(\gamma_0 - \alpha_3)$.

Define now

$$
\Omega = \left\{\left(\tilde{x}, \tilde{W}\right) \mid \left\|\tilde{x}(t)\right\| \leq \eta exp(-\gamma_2 t)\right\}
\tag{5.26}
$$

Note that the numerator in the bracket of (5.25) is greater than zero for $\|\tilde{x}\| > exp(-\gamma_2 t)$ (or $\tilde{x} \in \Omega^c$), hence

$$\dot{\bar{V}} \leq -\left[\lambda_{min}(Q) + l_0\right] \|\tilde{x}\|^2 \tag{5.27}$$

Further, since $\bar{V}$ is bounded from below and non increasing with time, we have

$$lim_{t\to\infty} \int_0^t \|\tilde{x}(\tau)\|^2 \, d\tau \leq \frac{\bar{V}(0) - \bar{V}_\infty}{\lambda_{min}(Q) + l_0} < \infty \tag{5.28}$$

where $lim_{t\to\infty}\bar{V}(t) = \bar{V}_\infty < \infty$. Notice that, based on (5.14), with the bounds on $\tilde{x}$, $\tilde{W}$, $\tilde{V}$, and l, $\dot{\tilde{x}}$ is also bounded. Thus, $\dot{\bar{V}}$ is uniformly continuous. Hence, by applying the Barbalat's Lemma [62], we conclude that $lim_{t\to\infty}\tilde{x}(t) = 0$ for all $\tilde{x} \in \Omega^c$.

Once the synchronization error $\tilde{x}(t)$ has entered Ω, it will remain in Ω forever, due to (5.26). Consequently, we conclude that $lim_{t\to\infty}\tilde{x}(t) = 0$ holds in the large, i.e., whatever the initial value of $\left(\tilde{x}(t), \tilde{W}(t), \tilde{V}(t)\right)$ (inside or outside Ω). □

Corollary 5.5.2. Consider the class of general nonlinear systems described by (5.4), with satisfies Assumptions 1-2, the identification model (4.12), (5.15-5.17) with $\gamma_2 = 0$. Then, the state error $\tilde{x}(t)$ converges to the residual set

$$\Xi = \left\{\tilde{x} \;\middle|\; \|\tilde{x}(t)\| \leq \gamma_1 \alpha_1\right\} \tag{5.29}$$

where $\alpha_t = exp(-\gamma_2 t_s)$ and t_s is the time in which the exponential function in (5.15) is turned off.

Remark 4. Corollary 5.5.2 establish an interesting peculiarity of the proposed method. The exponential function used in the identification model can be turned off when the residual state error has entered to any desired neighborhood of the origin. It is important to overcome numerical errors that can appear when the exponential function on the right-hand side of (5.15) has practically decayed to zero.

Remark 5. It should be noted that the disturbances in an infinite horizon are related to the residual state error. However, this residual error can be controlled, for instance, by the design parameter l_0 and matrix B. For further details, see [66].

5.6 Simulation

This section presents four examples to validate the theoretical results. In the following simulations, it should be noted that the identification models does not depend explicitly on perturbations. Moreover, in all simulations, Solver ode23 (Bogacki-Shampine) of Matlab/Simulink®, with a relative tolerance of 1e-6 was used to obtain the numerical solutions. Also, the time in which the

exponential function in (5.15) is turned off was set to $t_s = 3s$ for all cases. First, the identification of a Chen system using the proposed methodology is achieved. Then, the same system and identification model are submitted to the presence of disturbances. To check the performance for higher dimensional systems, a complex hyperchaotic system with 7 states is identified. Finally, a proposed algorithm in the literature [2] is used here for comparison.

5.6.1 Chen System with proposed algorithm

Consider the unified chaotic system [132], which is described by

$$\begin{aligned} \dot{x} &= (25\alpha + 10)(y - x) \\ \dot{y} &= (28 - 35\alpha)x - xz + (29\alpha - 1)y \\ \dot{z} &= xy - \left(\frac{8+\alpha}{3}\right) z \end{aligned} \tag{5.30}$$

where x, y and z are state variables and is always chaotic in the whole interval $\alpha \in [0, 1]$. It should be also noted that system (5.30) becomes the Lorenz system for $\alpha = 0$, Chen system for $\alpha = 1$. In particular, system (5.30) bridges the gap between the Lorenz system and the Chen system. In the following simulation, we consider the Chen system.

To identify the chaotic system (5.30), the proposed identification model (5.12) and the adaptive laws (5.14) and (5.16) were implemented. The design parameters were chosen as $\alpha_W = 1$, $\alpha_V = 1$ $\gamma_W = 0.02$, $\gamma_V = 0.001$, $\alpha_W = 0.5$, $\alpha_V = 0.5$, $l_0 = 10$, the sigmoid function is $\sigma(.) = 100/1 + exp - 1(.)$, the design matrices are

$$A = \begin{bmatrix} -7.8 & 0 & 0 \\ 0 & -7.8 & 0 \\ 0 & 0 & -7.8 \end{bmatrix}, B = \begin{bmatrix} 121 & 0 & 0 \\ 0 & 127.6 & 0 \\ 0 & 0 & 143 \end{bmatrix} \tag{5.31}$$

with $W_0 = 0$, $V_0 = 0$ and $P = 0.05 \times I_{3\times 3}$, where I is the identity matrix. The chosen inputs are

$$z = \begin{bmatrix} x & y & z & x^2 & y^2 & z^2 & 1 \end{bmatrix} \tag{5.32}$$

The chosen initial conditions for the system and the identification model are $x(0) = 2$, $y(0) = 1$, $z(0) = 2$, $\hat{x}(0) = 5$, $\hat{y}(0) = 5$, $\hat{z}(0) = 5$, $\hat{W}(0) = 0$ and $\hat{V}(0) = 0$. The performances in the estimation of state variables are shown in Fig. 5.1-5.3. It can be seen that the simulations confirm the theoretical results, that is, the algorithm is stable and the residual state error is small. The Frobenius norms associated to the estimated weight matrices W and V are shown in Fig. 5.4 and 5.5, respectively. After a transient phase, due to the large initial uncertainty, these norms seem to converge, indicating that most of the state estimation error has been removed. It should be noted that the transient can be shaped according to the user's desired convergence.

In the simulations of the proposed algorithm, the design matrices A, B, P and K were initially chosen as identity matrices. In the sequence, these values were adjusted, by a trial and error

procedure to fulfill requirements of performance.

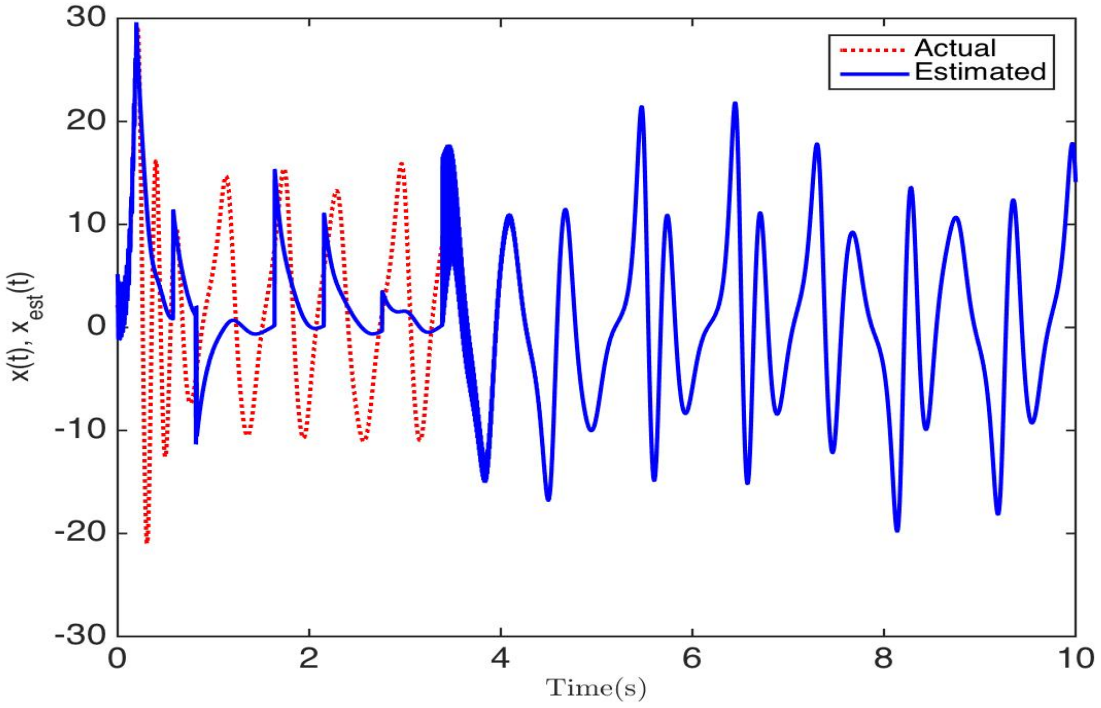


Figure 5.1: Performance in the estimation of x

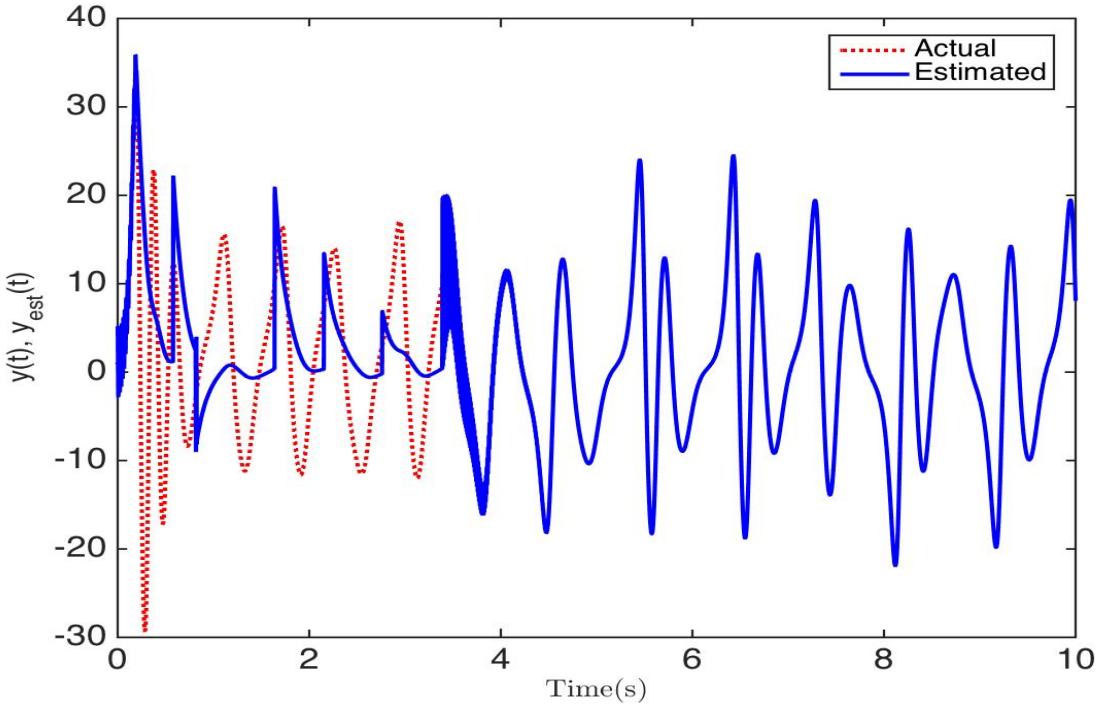


Figure 5.2: Performance in the estimation of y

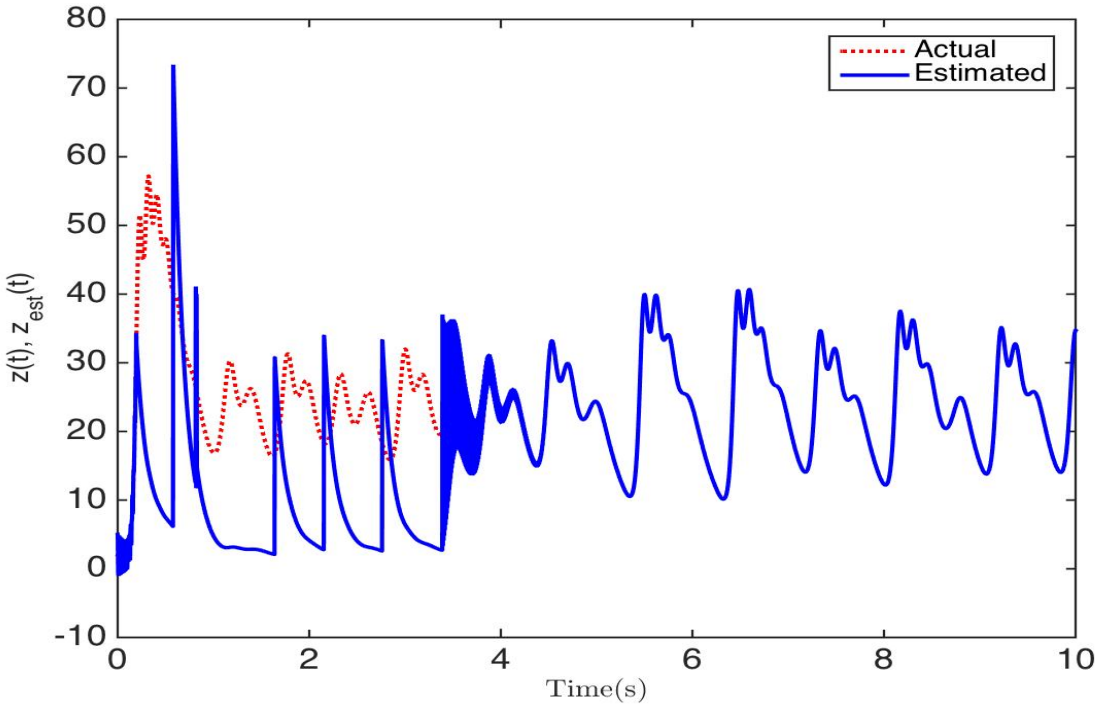


Figure 5.3: Performance in the estimation of z

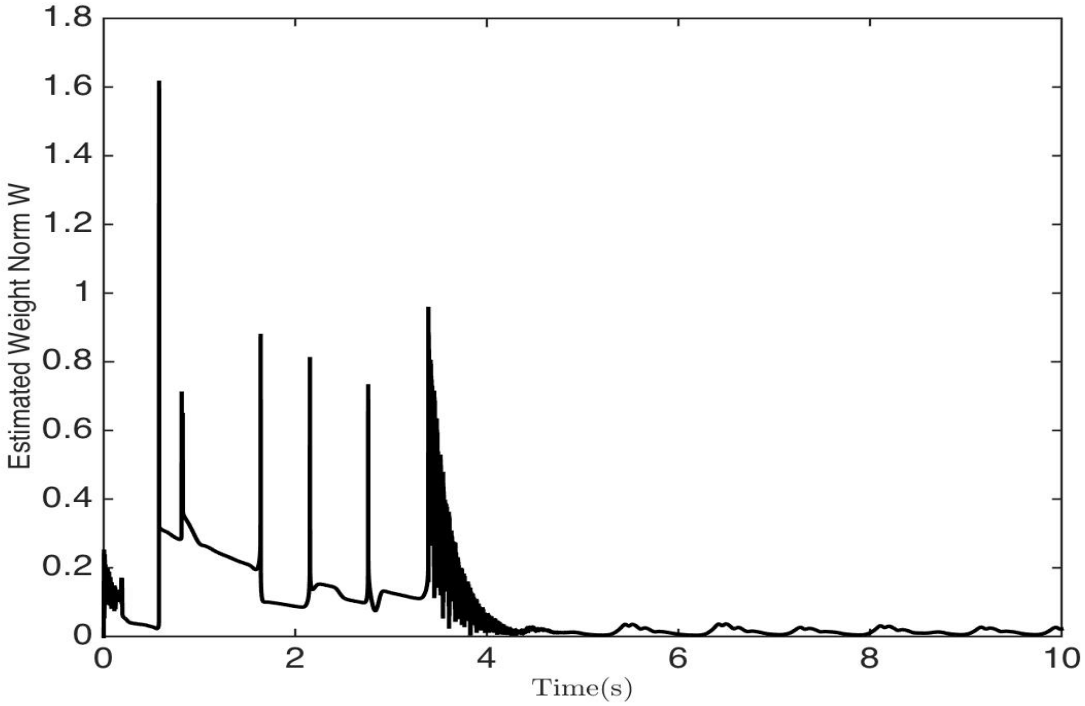


Figure 5.4: Frobenius norm of the estimated weight matrix W

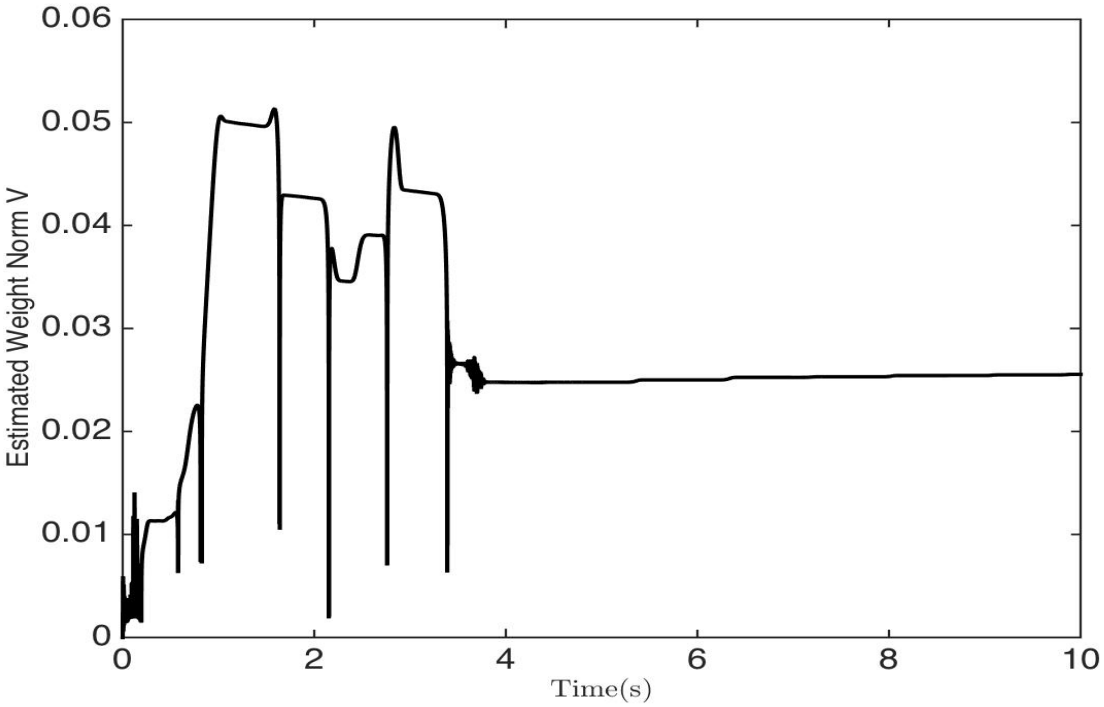


Figure 5.5: Frobenius norm of the estimated weight matrix V

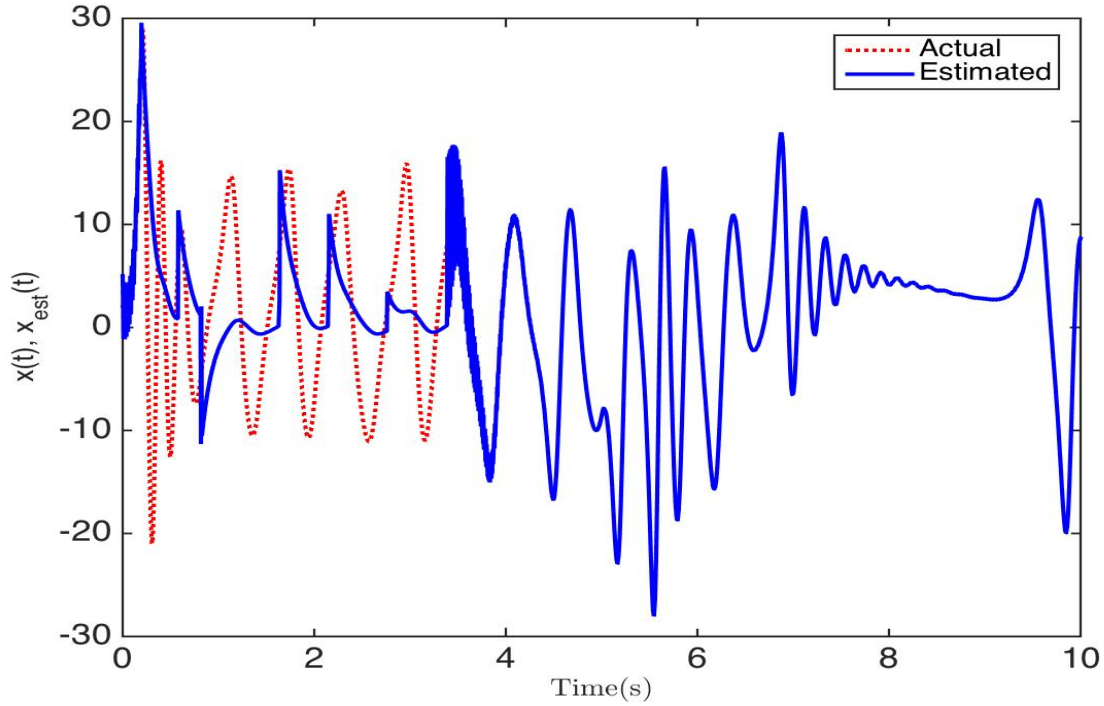


Figure 5.6: Performance in the estimation of x

To check the robustness of the proposed method, it is now considered an additional case with

the presence of the following time-dependent disturbance:

$$d(x,u,v,t) = 3sin(t)\,\|\bar{x}\| + 50sin(200t) + 10cos(400t) \times \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \tag{5.33}$$

where $\bar{x} = [x\ y\ z]^T$. Note that the disturbance appears only to states y and z in (5.30). Keeping all design parameters and initial conditions as before, we consider the emergence, at $t = 5s$, of disturbances of the form 5.33. The obtained results are shown in Fig. 5.6-5.8. From Fig. 5.9 and 5.10, it can be concluded that the identification scheme is robust in the presence of perturbation without, practically, any degradation of performance.

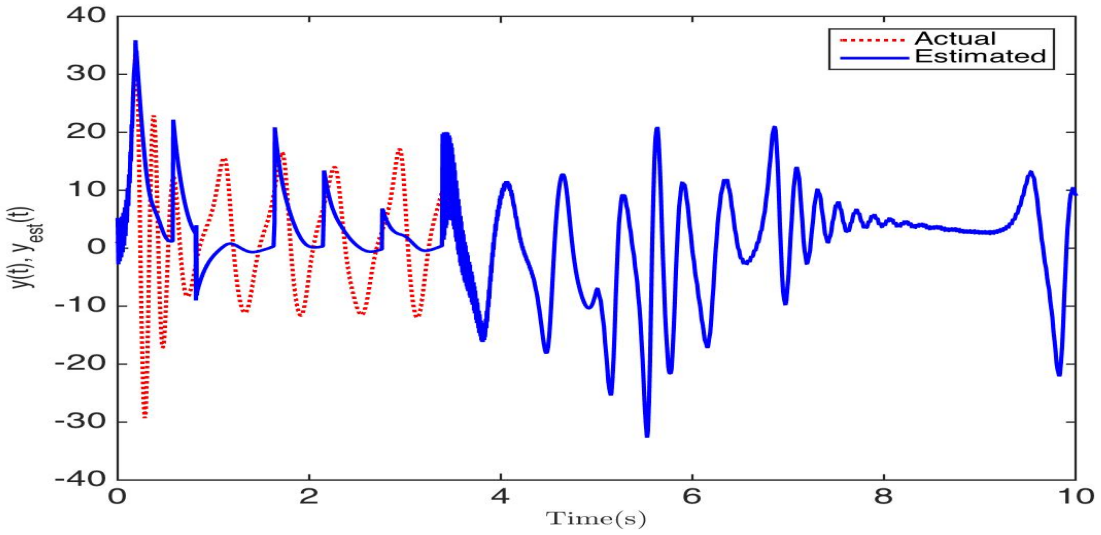


Figure 5.7: Performance in the estimation of y

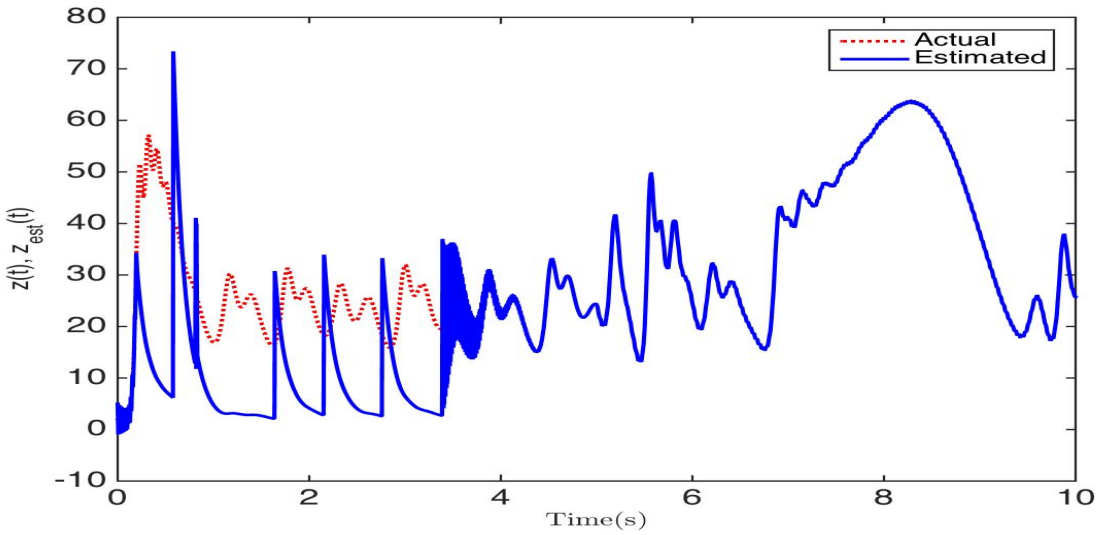


Figure 5.8: Performance in the estimation of z

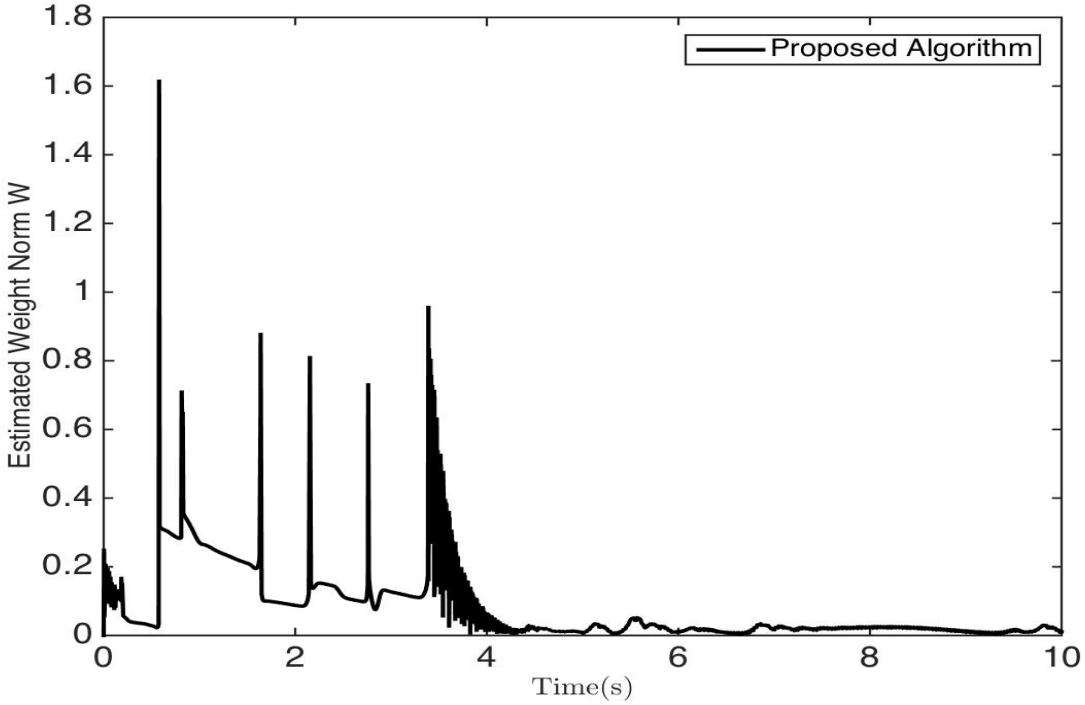


Figure 5.9: Frobenius norm of the estimated weight matrix W

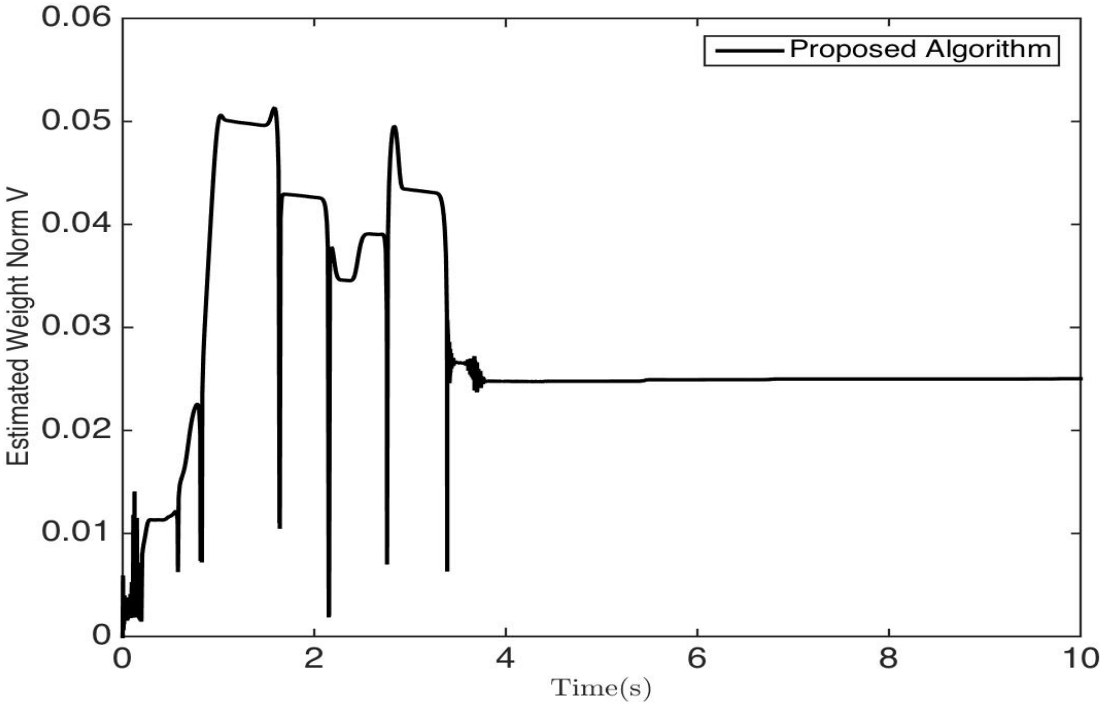


Figure 5.10: Frobenius norm of the estimated weight matrix V

5.6.2 Hyperchaotic System

Consider a hyperchaotic system described by [134]

$$\begin{aligned}
\dot{x}_1 &= \alpha(x_3 - x_1) + d_{x1} \\
\dot{x}_2 &= \alpha(x_4 - x_2) + d_{x2} \\
\dot{x}_3 &= \gamma x_1 - x_1 x_5 - x_3 + x_6 + d_{x3} \\
\dot{x}_4 &= \gamma x_2 - x_2 x_5 - x_4 + x_7 + d_{x4} \\
\dot{x}_5 &= x_1 x_3 + x_2 x_4 - \beta x_5 + d_{x5} \\
\dot{x}_6 &= k_1 x_1 + k_2 x_3 + d_{x6} \\
\dot{x}_7 &= k_1 x_2 + k_2 x_4 + d_{x7}
\end{aligned} \tag{5.34}$$

where α, β, γ, k_1 and k_2 are constant parameters and d_{x1}, d_{x2}, d_{x3}, d_{x4}, d_x5, d_{x6} and d_{x7} are unknown disturbances. It was considered that $\alpha = 14$, $\beta = 3$, $\gamma = 50$, $k_1 = -5$ and $k_2 = -4$. Notice that system (5.34) satisfies assumption 1, since the state variables evolve on compact sets.

To identify the uncertain system (5.34), the proposed identification model (5.12) and the adaptive laws (5.14) and (5.16) were implemented. The design parameters were chosen as $\gamma_V = 1$, $\gamma_w = 0.01$, $\alpha_W = 1$, $\alpha_V = 1$, $l_0 = 10$, the sigmoid function is $\sigma(.) = 150/1 + exp - 1(.)$, the matrices parameters are

$$A = -20 \times I_{7\times7},\ B = 100 \times I_{7\times7},\ P = 0.05 \times I_{7\times7} \tag{5.35}$$

where I is the identity matrix, with $W_0 = 0$, $V_0 = 0$. The chosen inputs are

$$z = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & 1 \end{bmatrix}^T \tag{5.36}$$

The chosen initial conditions for the system are $x_1(0) = 0$, $x_2(0) = 1$, $x_3(0) = 2$, $x_4(0) = 3$, $x_5(0) = 4$, $x_6(0) = 5$, $x_7(0) = 6$, $\hat{x}_1(0) = -50$, $\hat{x}_2(0) = -40$, $\hat{x}_3(0) = -30$, $\hat{x}_4(0) = 20$, $\hat{x}_5(0) = 40$, $\hat{x}_6(0) = 50$, $\hat{x}_7(0) = 400$, $\hat{W}(0) = 0$ and $\hat{V}(0) = 0$. To check the robustness of the proposed method, it is considered the presence of the following time-dependent disturbance

$$d(x,u,v,t) = 1.8\|x\| \begin{bmatrix} sin(t)u \\ 1.2sin(2t) \\ cos(4t) \\ 1.2sin(t) \\ 1.1sin(2t) \\ 0.5sin(4t) \\ exp(-0.5t) \end{bmatrix} \tag{5.37}$$

where $x = [x_1\ x_3\ x_3\ x_4\ x_5\ x_6\ x_7]^T$. We consider the emergence, at $t = 5s$, of disturbances of the form (5.37).

The performances in the estimation of state variable are shown in Fig. 5.11-5.17. It can be seen that the simulations confirm the theoretical results, that is, the algorithm is stable and the residual state error is small. The Frobenius norms associated to the estimated weight matrices W and V are shown in Fig. 5.18-5.19 respectively. One can find that the system outputs achieved by the proposed neural controller still asymptotically track the desired trajectories well. Further, in comparison with the same simulation using the ELM model in chapter 4 shows the better convergence property of the SHLNNs.

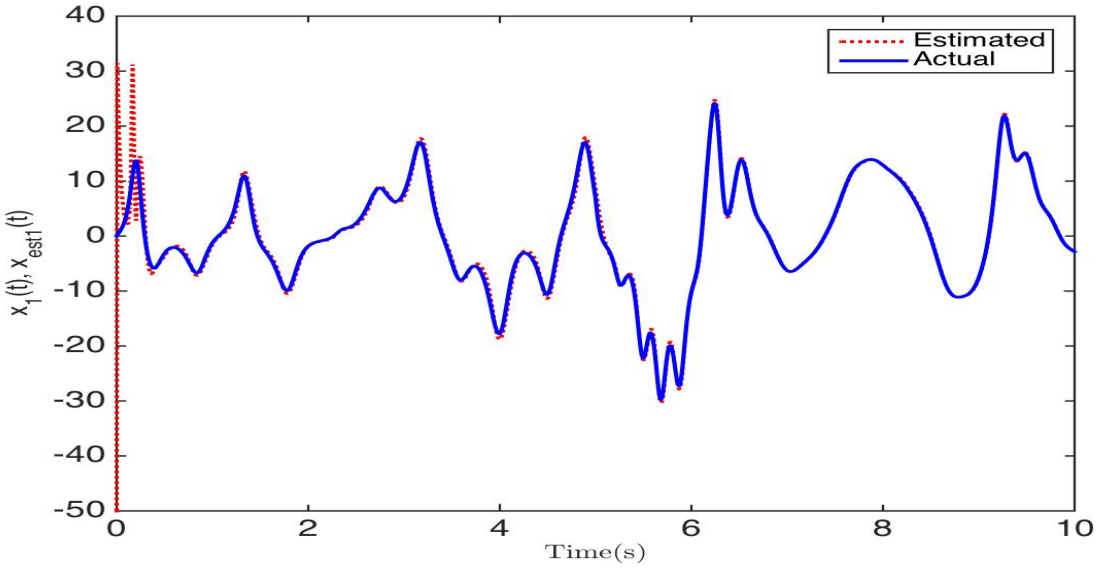


Figure 5.11: Performance in the estimation of x_1

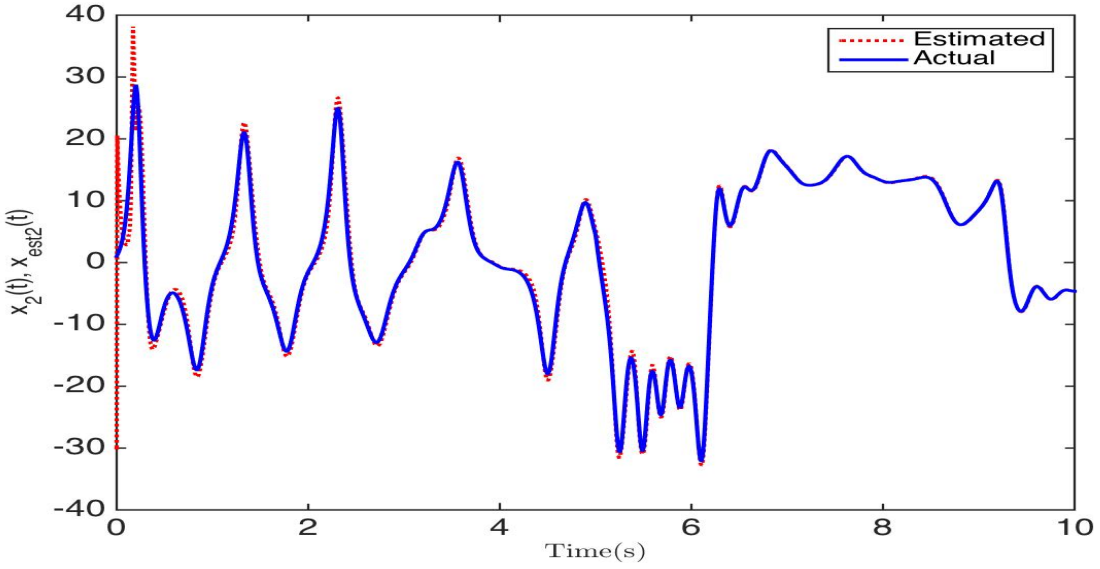


Figure 5.12: Performance in the estimation of x_2

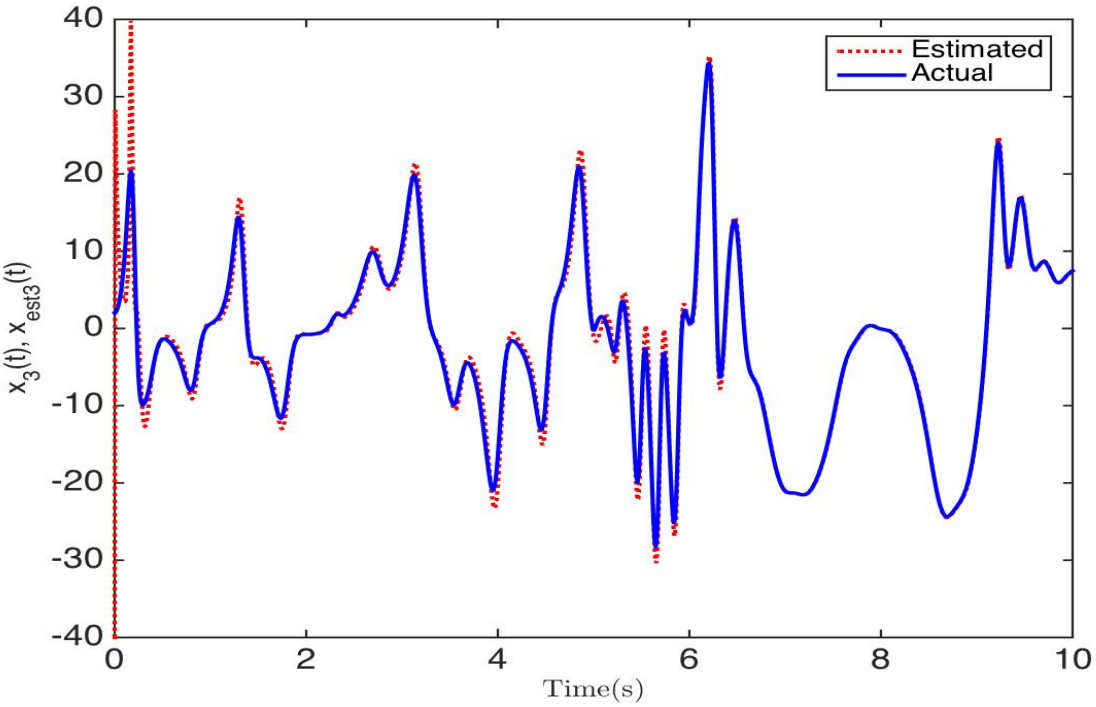


Figure 5.13: Performance in the estimation of x_3

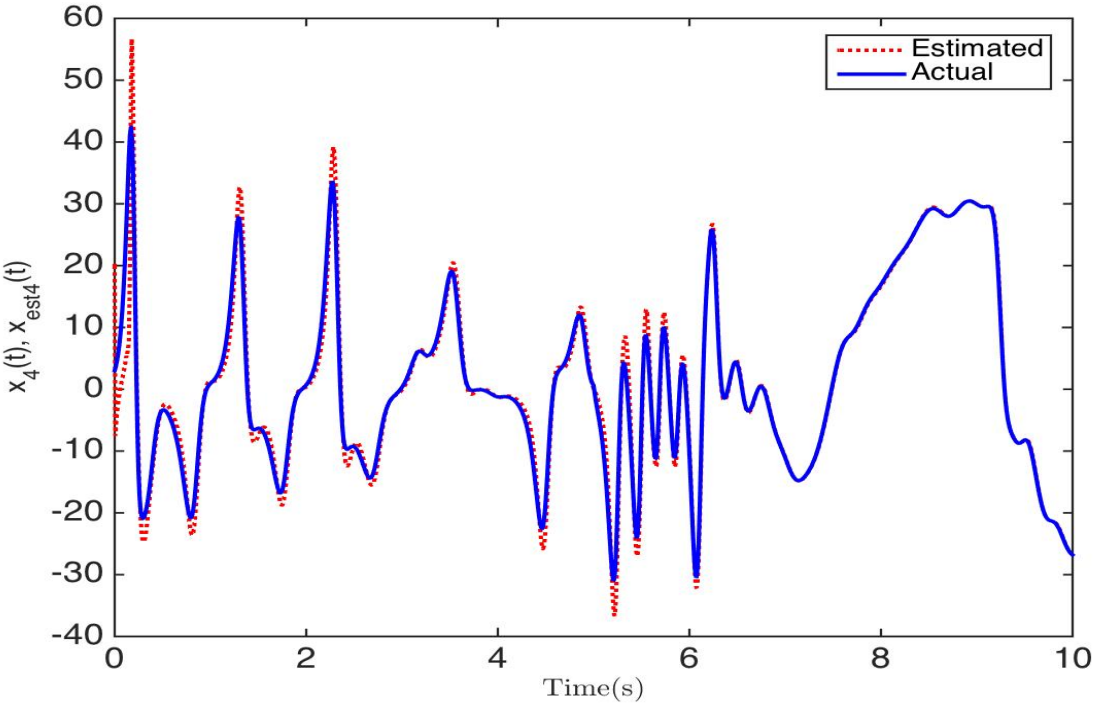


Figure 5.14: Performance in the estimation of x_4

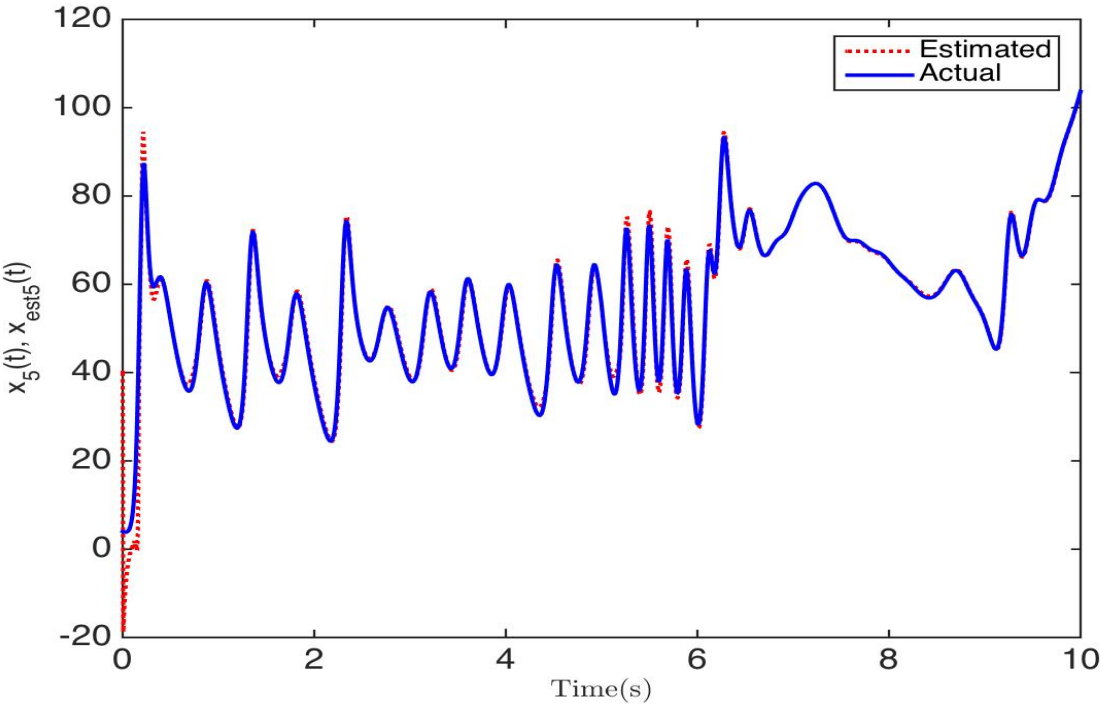


Figure 5.15: Performance in the estimation of x_5

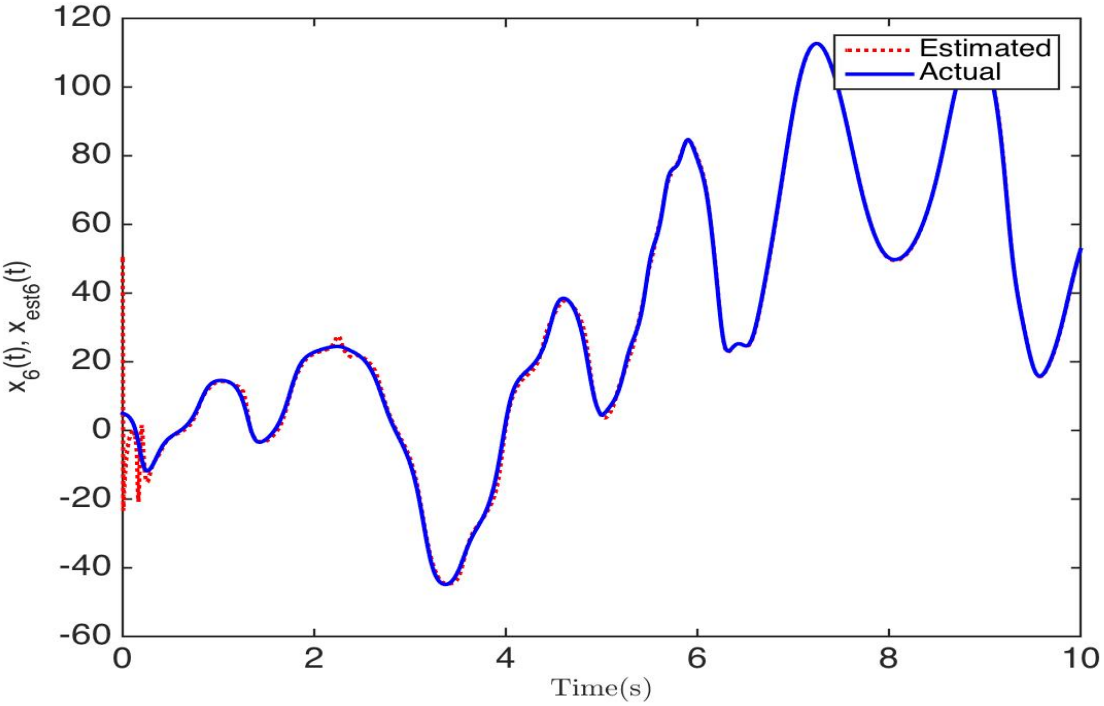


Figure 5.16: Performance in the estimation of x_6

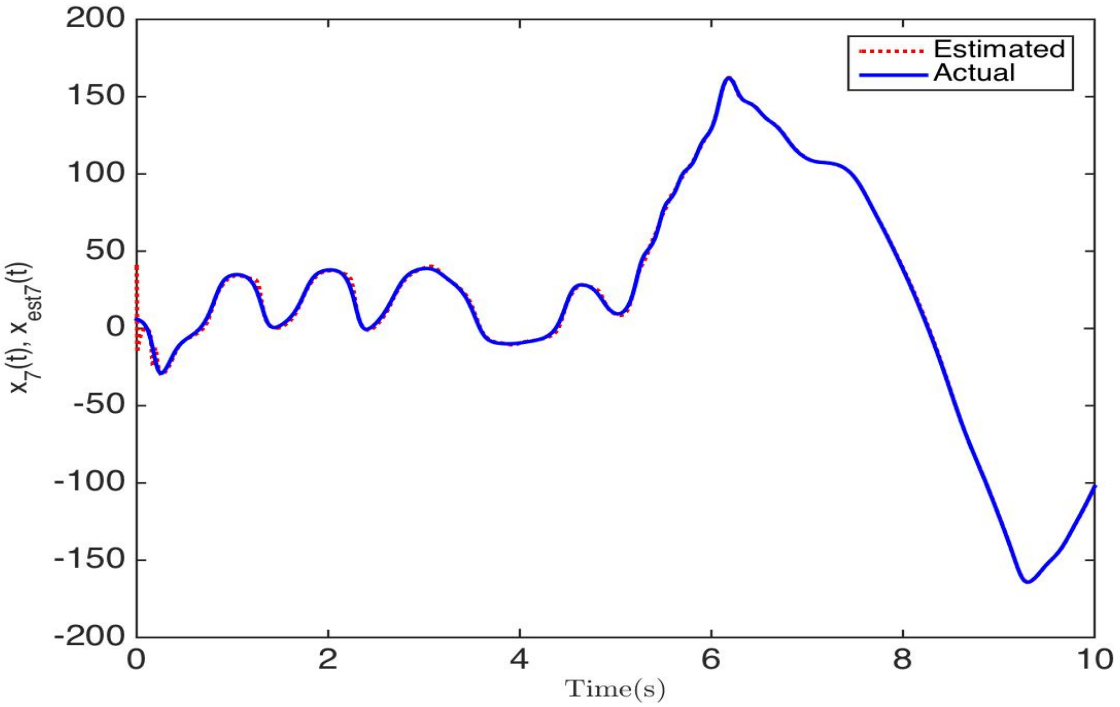


Figure 5.17: Performance in the estimation of x_7

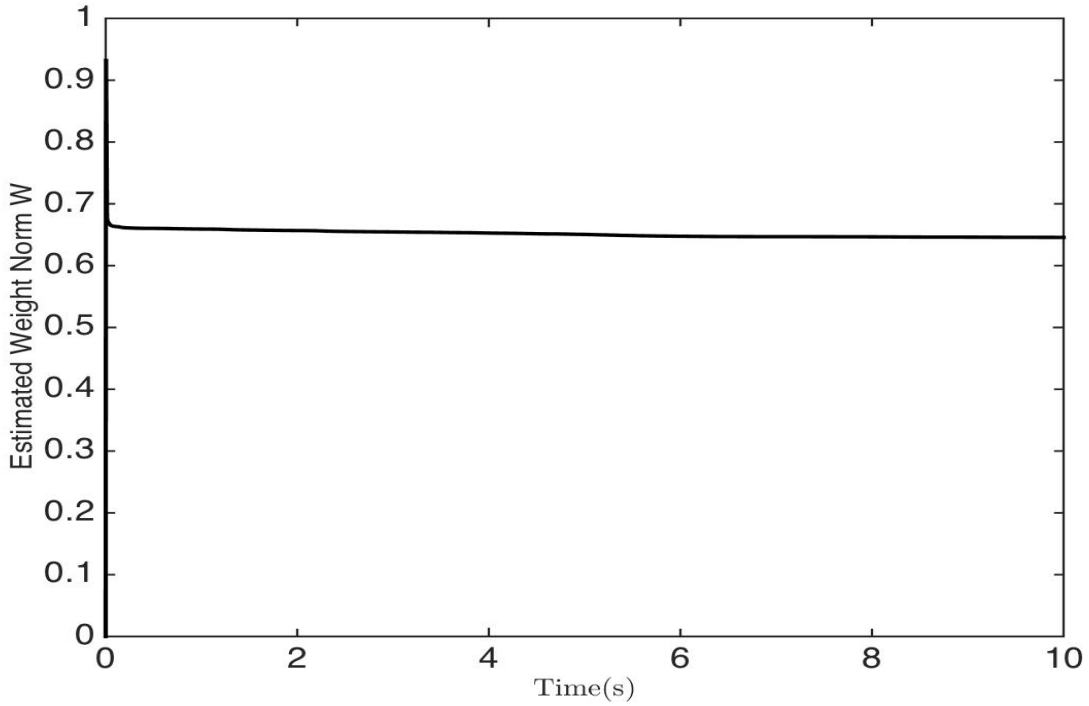


Figure 5.18: Frobenius norm of the estimated weight matrix W

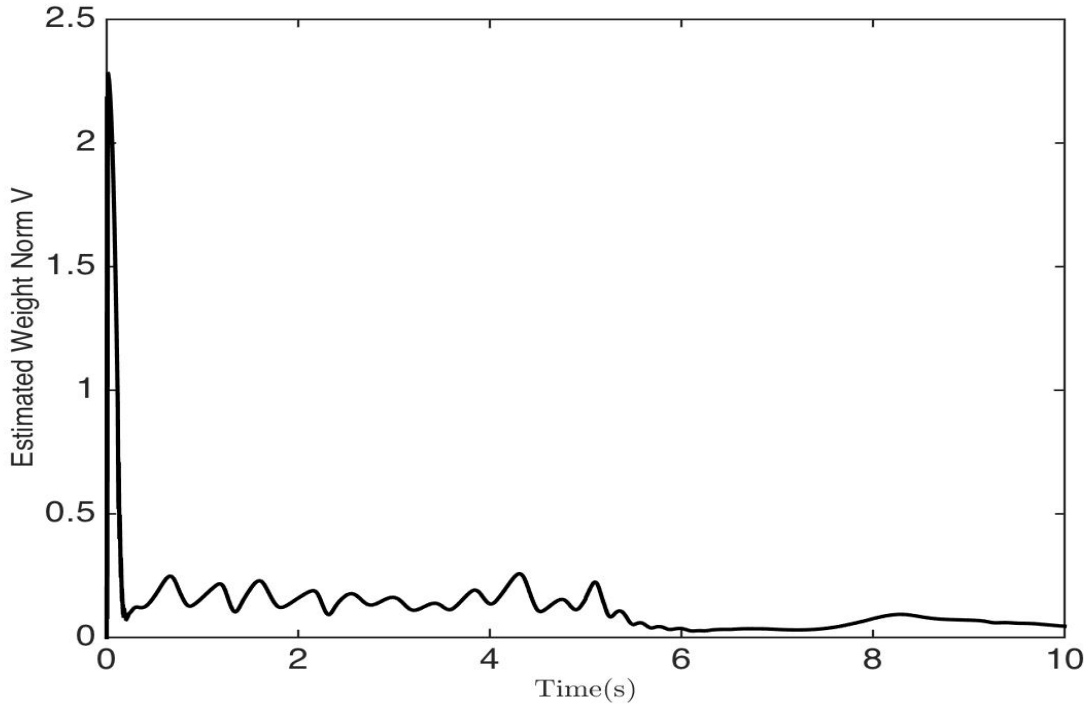


Figure 5.19: Frobenius norm of the estimated weight matrix V

5.6.3 Comparison with Ref. [2]

To illustrate the advantages of the proposed methodology, the identification model introduced in [2] is used here for comparison. Consider the online identification multilayer neural network algorithm proposed in [2] described as

$$\begin{aligned}
\dot{\hat{x}} &= A\hat{x} + \hat{W}\sigma\left(\hat{V}\hat{\tilde{x}}\right) \\
\dot{\hat{W}} &= -\eta_1\left(\tilde{x}^T A^{-1}\right)^T\left(\sigma\left(\hat{V}\hat{\tilde{x}}\right)\right)^T - \rho_1\|\tilde{x}\|\hat{W} \\
\dot{\hat{V}} &= -\eta_2\left(\tilde{x}^T A^{-1}\hat{W}\left(I - \bar{\Lambda}\left(\hat{V}\hat{\tilde{x}}\right)^T\hat{\tilde{x}}^T - \rho_2\|\tilde{x}\|\hat{V}\right)\right)
\end{aligned} \tag{5.38}$$

where $\bar{\Lambda}\left(\hat{V}\hat{\tilde{x}}\right) = diag\left\{\sigma_i^2\left(\hat{V}\hat{\tilde{x}}\right)\right\}$. The Chen system (5.30) is considered. The design parameters are chosen as $\eta_1 = 25$ and $\eta_2 = 0.4$, $\rho_1 = 0.00012$, $\rho_2 = 0.00012$, the sigmoid function is $\sigma(.) = 500/1 + exp - 0.5(.)$. The design matrix A is chosen as

$$A = \begin{bmatrix} -0.0078 & 0 & 0 \\ 0 & -0.0078 & 0 \\ 0 & 0 & -0.0078 \end{bmatrix} \tag{5.39}$$

Other design parameters and initial conditions were chosen as in section 5.6.1.

We consider the same disturbance from the previous example in (5.33). Figs. 5.20-5.22 show the

state error norms comparisons for each state variable. It should be pointed out that the adjustment of the design parameters in [2] was not trivial, mainly due to the mutual dependence between the design matrices P e Q. Furthermore, it seems that the static approximation hypothesis assumed in [2] has a negative impact on the performance, as can be seen in Figs. 5.20-5.25

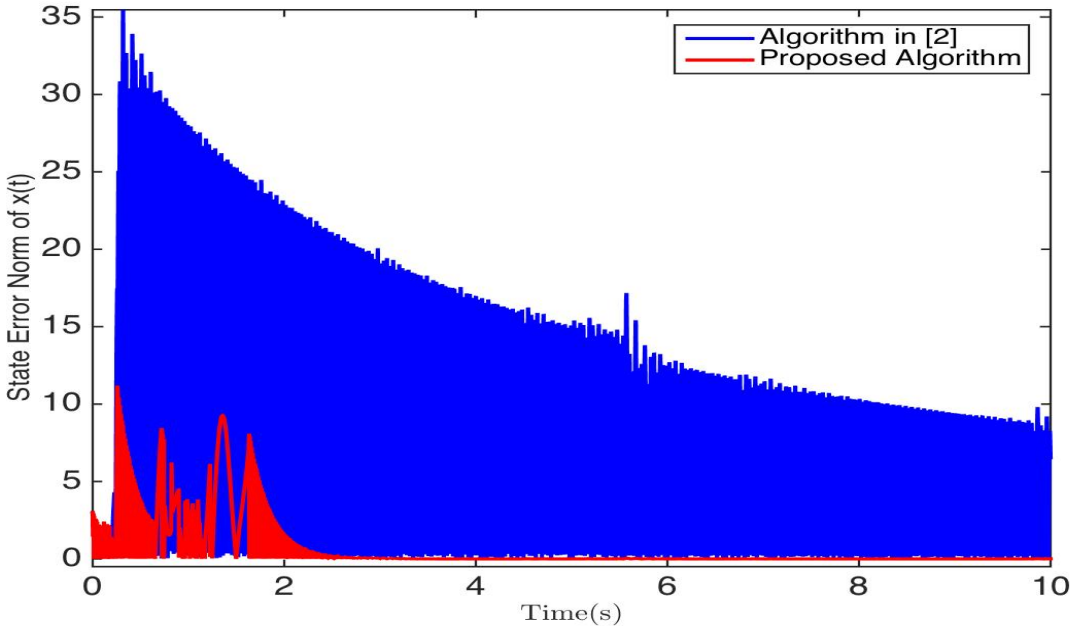


Figure 5.20: Performance comparison in the estimation of x

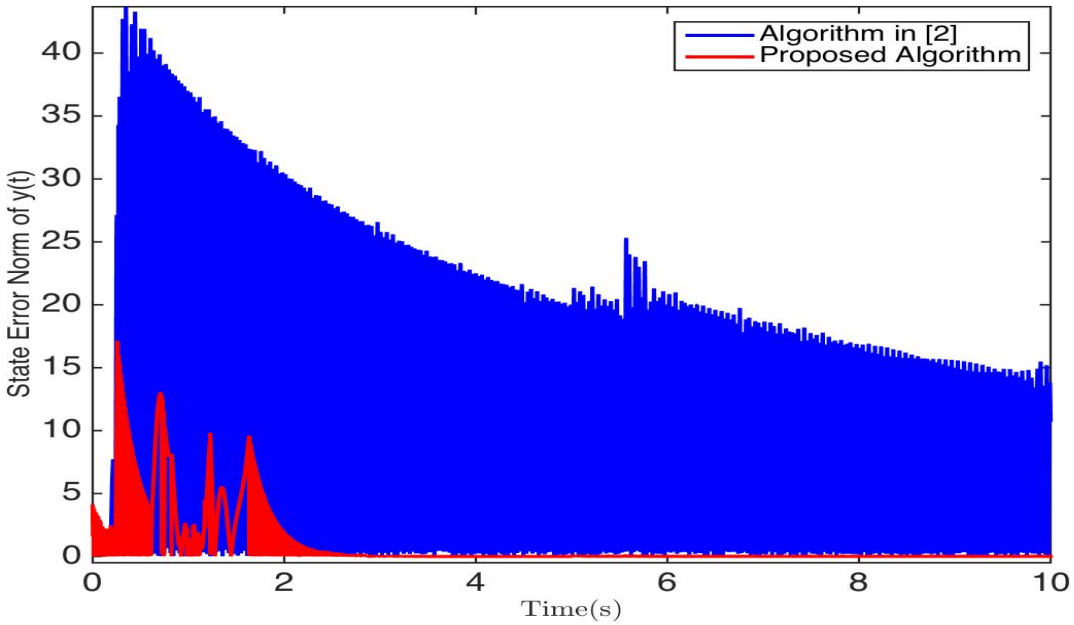


Figure 5.21: Performance comparison in the estimation of y

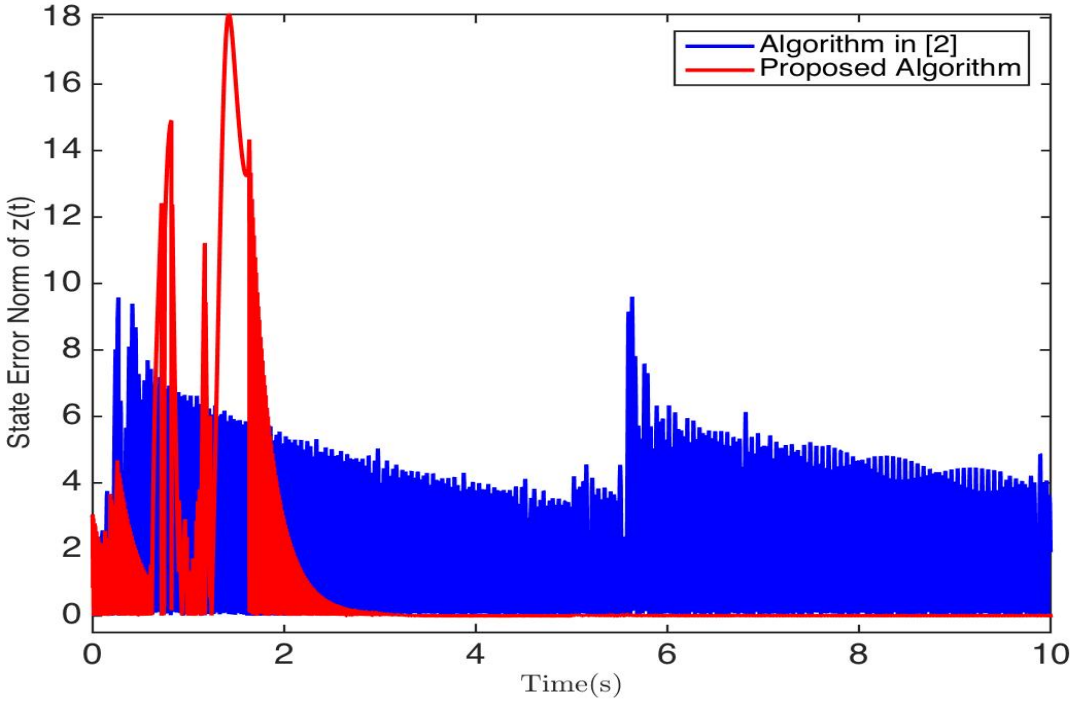


Figure 5.22: Performance comparison in the estimation of z

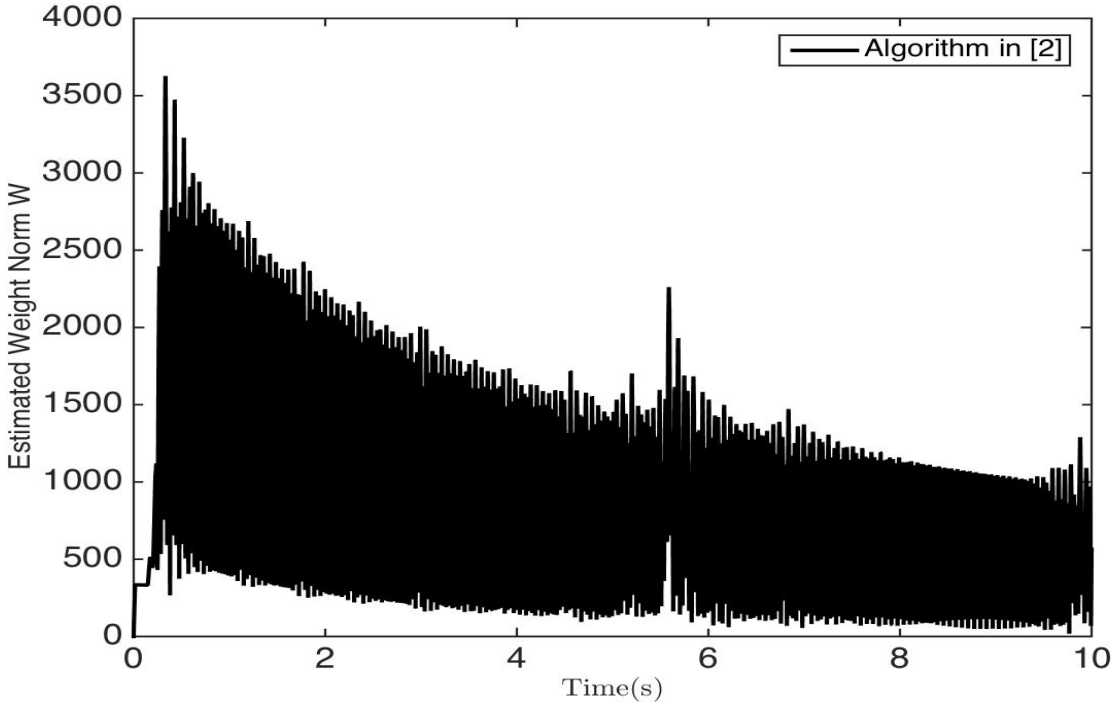


Figure 5.23: Frobenius norm of the estimated weight matrix W

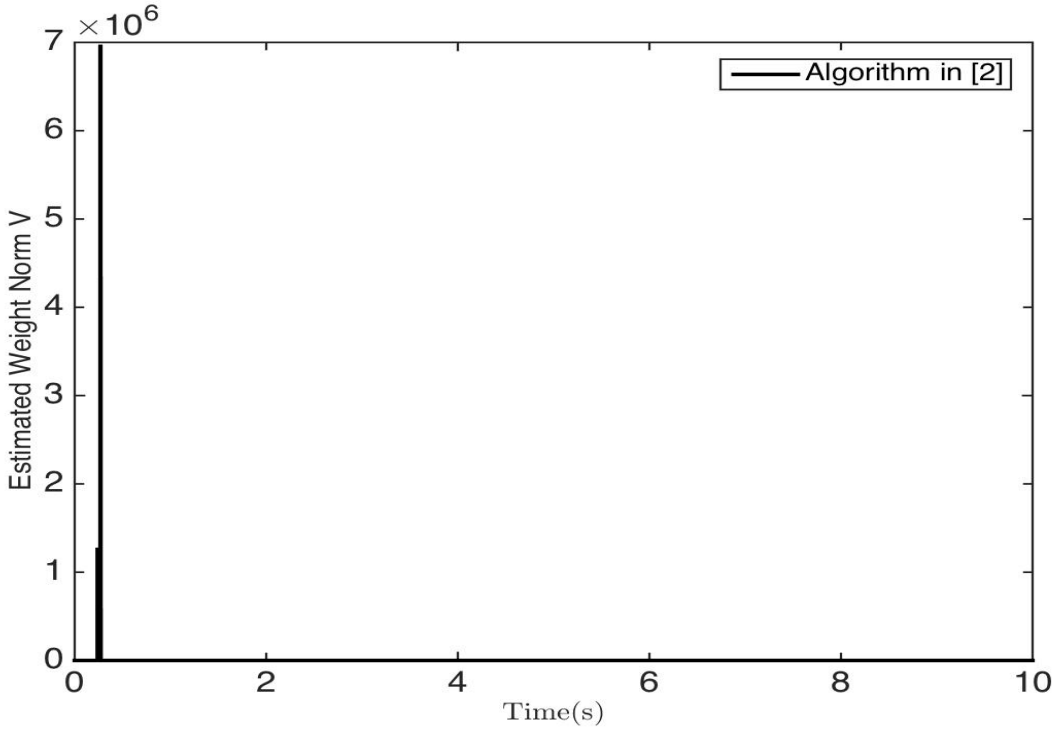


Figure 5.24: Frobenius norm of the estimated weight matrix V

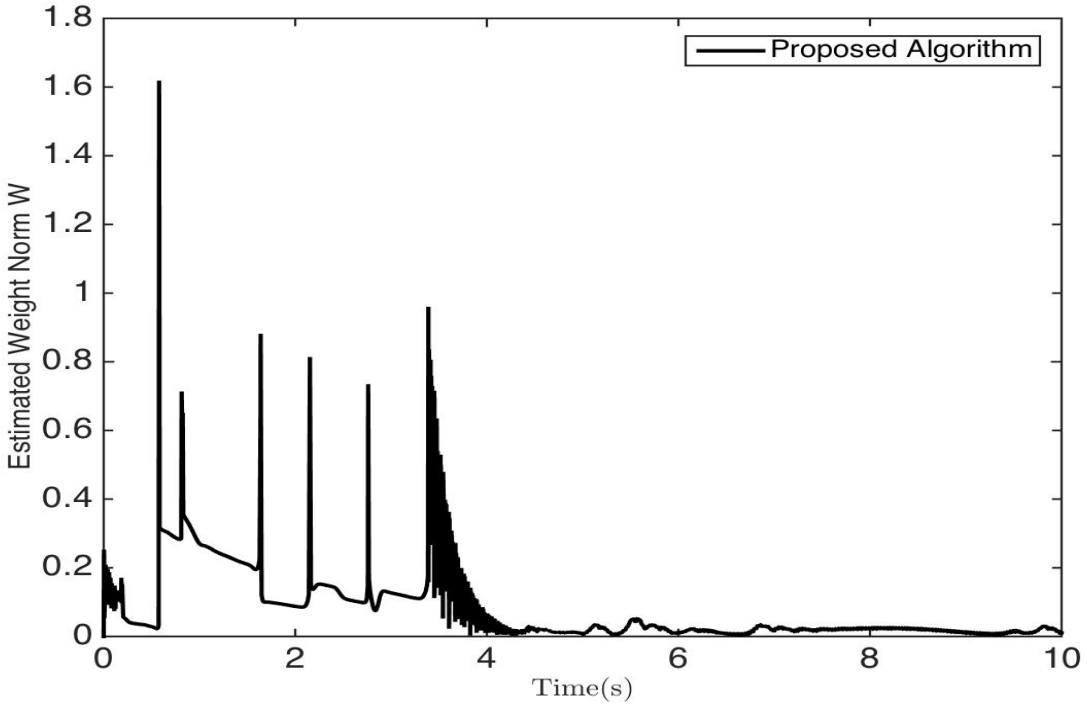


Figure 5.25: Frobenius norm of the estimated weight matrix W

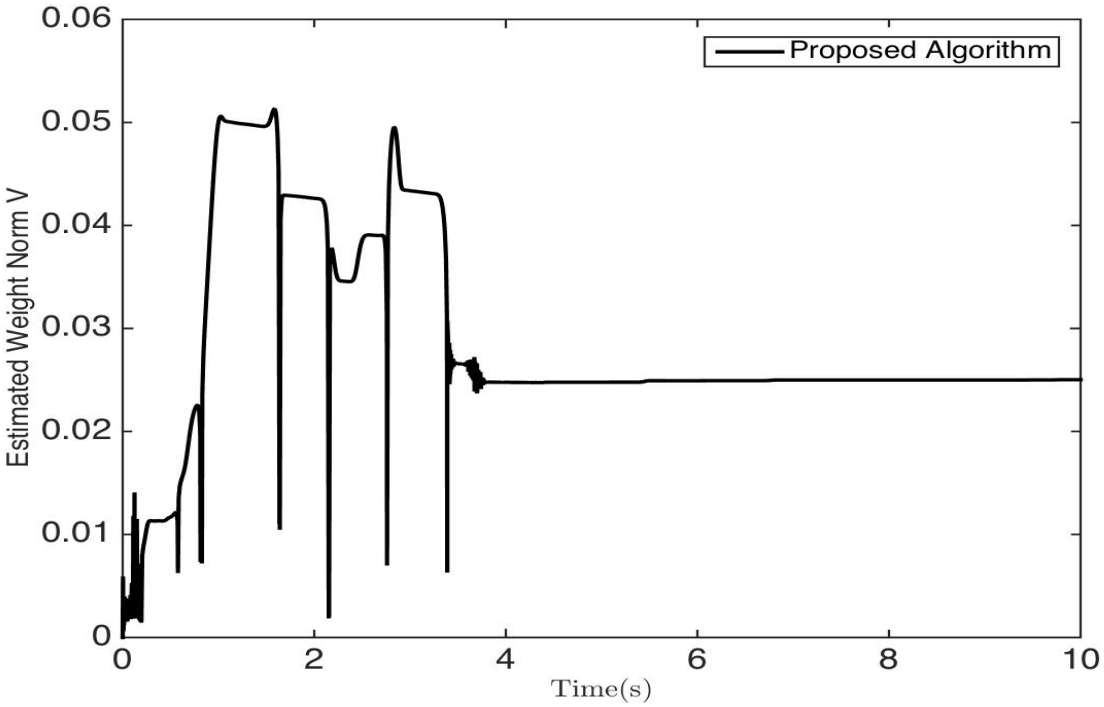


Figure 5.26: Frobenius norm of the estimated weight matrix V

5.7 Discussions

As stated in previous sections, for all the simulations of the proposed algorithms, the design matrices A, B, P and K were initially chosen as identity matrices. In the sequence, these values were adjusted, by a trial and error procedure to fulfill requirements of performance. After a transient phase, due to the large initial uncertainty, these norms seem to converge, indicating that most of the state estimation error has been removed. It should be noted that the transient can be shaped according to the user's desired convergence as well as the learning speed can be adjusted given the project requirements.

The transients of the simulations with the algorithm proposed in Chapter 4 showed higher frequency behavior than the algorithm proposed in this chapter. Although they can be modeled to show a slower frequency, it is generally easier and more intuitive to shape the transient with SHLNN scheme. However, the trade-off involved is that the former methodology is faster than the latter given that only needs tuning for the output weights once the hidden layer is randomly generated and remains fixed during the simulation. Additionally, despite the heavier computational cost, the SHLNN model required less inputs nodes than the algorithm proposed in Chapter 4, alleviating the so called "curse of dimensionality" for linearly parametrized models.

5.8 Summary

In this chapter, a novel identification scheme for the approximation of nonlinear dynamical systems is proposed. The scheme is based on a single-hidden layer neural network architecture to parametrize the unknown nonlinearities, whose hidden and output weights are simultaneously adjusted by adaptive laws designed using Lyapunov theory. All conditions are established to ensure the convergence of the residual state error to zero and all associated errors are bounded, even in the presence of approximation error and internal or external perturbations. Also, the dependence between the residual state error and some independent design parameters is straightforward. Consequently, the residual state error can be arbitrarily and easily reduced. Furthermore, it is not necessary any previous knowledge about the ideal weight, approximation error and disturbances.

Simulation results were performed to show the effectiveness and performance of the proposed approach in the presence of perturbations. Also, to show the applicability of the proposed learning algorithm for high-dimensional dynamical systems, a simulation using SHLNNs for a complex hyperhaotic system is demonstrated. Finally, a comparison of the proposed algorithm with that in [2] was performed to show the advantages and peculiarities of the proposed method under disturbances. The results obtained and presented in this chapter have been reported in [137].

Chapter 6

Conclusions

This Master's thesis develops two novel online learning algorithms for neuro-based identification of a general class of nonlinear dynamical systems. The neuro identifiers were designed based on Lyapunov theory to ensure state error convergence to zero, despite the presence of approximation error and disturbances. This approach demonstrated its capabilities in terms of having a short development time and high accuracy, solving complex problems, and possessing the beneficial design properties of stability.

Chapter 2 provides a description of the historical developments of system identification along with a state of the art review of identification based on single-hidden layer neural networks. The literature review shows the relevancy and need for neural learning algorithms in control applications where attributes such as convergence and stability can be verified analytically. Despite successful use of neural networks for a wide range of applications, they are prone to local optimal problem and have slow convergence rates which limits itself to relatively simpler problems. In chapter 3, technical background about neural networks, their properties and the notation that was used throughout this Master's thesis is introduced. Furthermore, a brief description is given for the most used neural network topologies and the basic types of learning.

Several approaches has been recently proposed in the literature to tackle the aforementioned challenges. One has been extreme learning machine models which are linearly parametrized models efficient in terms of computation as well as optimality. The ELM and its variants lacks the stability analysis and conditions to ensure error boundedness or asymptotical convergence of the state error to zero. In this context, deriving an ELM-based identification scheme with adaptive output weights is highly desired. Recent researches based on Lyapunov theory have been proposed to address these issues [1, 15] by providing error boundedness, however they lack the assurance of asymptotical convergence of the residual state error to zero and boundedness of all associated approximation errors in the presence of approximation error and disturbances.

Aiming to address the above drawbacks. In chapter 4, by using an extreme learning machine, an online adaptive neuro-identification scheme for a general class of nonlinear systems in the presence of unknown dynamics and external disturbances is proposed. It is noteworthy that, besides the bounds assumption, no previous knowledge about the dynamics of the approximation error, ideal

weights or external perturbations is needed. The ELM is a class of SHLNNs where the hidden layer weights are randomly generated according to any continuous probability distribution, and here the output weights of the ELM are updated based on the stable adaptive laws derived from Lyapunov analysis. Analysis based on Lyapunov theory shows that the adaptive learning algorithm converges asymptotically for estimation of nonlinear systems with no prior knowledge about their system dynamics and under perturbations. The proposed methodology combines computational efficiency in terms of learning speed from the ELM technique with the stability of the system under disturbances guaranteed by the Lyapunov analysis.

Simulation results for a unified chaotic system and a hyperchaotic finance system demonstrated the effectiveness and performance of the proposed approach in the presence of disturbances. Additionally, to show the efficiency of the proposed learning algorithm for high-dimensional dynamical systems, a simulation for a complex hyperhaotic system is demonstrated without compromising the learning speed and convergence property. Finally, a comparison of the proposed algorithm with that in [1] was performed to show the advantages and peculiarities of the proposed method under disturbances. The state estimation error shows better convergence under external disturbances and avoids the parameter drift, as the weight norms show nearly constant values.

Further, as a more general case of neural identification, multilayer neural networks present better generalization performance for higher-dimensional problems, alleviating the "curse of dimensionality" common on linearly parametrized problems. Nevertheless, a considerable share of proposed learning algorithms for SHLNNs are modifications of gradient descent type algorithms, lacking stability analysis and robust terms to ensure convergence under external disturbances and may suffer from slow convergence.

Considering the previous limitations, in chapter 5, a novel identification scheme for the approximation of nonlinear dynamical systems is proposed. The scheme is based on a single-hidden layer neural network architecture to parametrize the unknown nonlinearities, whose hidden and output weights are simultaneously adjusted by adaptive laws designed with Lyapunov theory. All conditions are established to ensure the convergence of the residual state error to zero and all associated errors are bounded, even in the presence of approximation error and internal or external perturbations. Also, the dependence between the residual state error and some independent design parameters is straightforward. Consequently, the residual state error can be arbitrarily and easily reduced. Furthermore, it is not necessary any previous knowledge about the ideal weight, approximation error and disturbances.

Simulation results were performed to show the effectiveness and performance of the proposed approach in the presence of perturbations. Also, to show the applicability of the proposed learning algorithm for high-dimensional dynamical systems, a simulation for a complex hyperhaotic system is demonstrated. Finally, a comparison of the proposed algorithm with that in [2] was performed to show the advantages and peculiarities of the proposed method under disturbances. The state estimation error shows faster convergence near the origin, as the weight norms show nearly constant values, however the main difficulty was heavier computational cost. There are several other opportunities to extend the present work in the future as below:

- The universal approximation capability is an important property since it justifies the application of the neural networks to any function approximation problem. However, the theorem is not constructive due to the fact that no method is provided for finding the ideal weights, optimum learning time and no information about the number and characteristics of the hidden neurons that would be required to achieve a given accuracy. Thus, tuning the design parameters of a neural network to a given target performance can be a rather tiresome task and may lead to sub-optimal solution, specially in identification of nonlinear systems. For future work, applications in evolutionary computation may help alleviate this issue. Since they belong to a family of trial and error problem solvers and are mostly applied to black box applications, they can be a valuable tool for global dynamic optimization search in identification of complex nonlinear systems.

- Another issue for identification of nonlinear systems is the need of all inputs for measuring. Nevertheless, this is not often possible in practical situations, since physical sensors typically have shortcomings that can degrade a control system for a number of reasons. To cite a few, sensors may be expensive and substantially raise the cost of a control system, they can induce significant errors such as limited responsiveness and stochastic noise, also some signals are impractical to measure given the environment. To mitigate such limitations, it could be an interesting research direction for the future to investigate the potential of the proposed learning algorithms for observer models. Since observers are considered a general case of system identification, they provide better and cheaper implementation in engineering applications.

- The simulations included in this Master's thesis are classified as open-loop identification. However, there are plants which may contain an integrator or are unstable in open-loop operation. The integrator is an inherently unstable device which tends to make the system less stable as, for instance, its response to a step input, a bounded signal, is a ramp, a unbounded signal. In many engineering applications, the performance of the closed-loop system can be improved by using a controller based on the identified model from the closed-loop data. Thus, further researches could focus on identification for robust control where a plant model is identified in closed-loop operation.

References

[1] JANAKIRAMAN, V.; ASSANIS, D. Lyapunov method based online identification of nonlinear systems using extreme learning machines. *Computing Research Repository (CoRR)*, abs/1211.1441, p. 1–8, 2012.

[2] ABDOLLAHI, F.; TALEBI, H.; PATEL, R. Stable identification of nonlinear systems using neural networks: Theory and experiments. *IEEE Transactions on Mechatronics*, v. 11, n. 4, p. 488–495, 2006.

[3] ABLAMEYKO, S. et al. *Neural Networks for Instrumentation, Measurement and Related Industrial Applications*. [S.l.]: IOS Press, 2003.

[4] HAYKIN, S. *Neural networks and learning machines*. [S.l.]: Pearson Prentice Hall, 2008.

[5] LEONDES, C. *Control and Dynamic Systems*. [S.l.]: Academic Press, 1997.

[6] LJUNG, L. Perspectives on system identification. *Annual Reviews in Control*, v. 34, n. 4, p. 1–12, 2010.

[7] ZADEH, L. From circuit theory to system theory. *IRE Proceedings*, v. 50, n. 5, p. 856–865, 1962.

[8] TANGIRALA, A. *Principles of System Identification: Theory and Practice*. [S.l.]: CRC Press, 2015.

[9] FU, L.; LI, P. The research survey of system identification method. In: *5th International Conference on Intelligent Human-Machine Systems and Cybernetics*. [S.l.]: IEEE, 2013. p. 397 – 401.

[10] GE, S. et al. *Stable Adaptive Neural Network Control*. [S.l.]: Springer, 2002.

[11] IOANNOU, P.; SUN, J. *Robust adaptive control*. [S.l.]: Dover Publications, 2012.

[12] LIU, G. *Nonlinear Identification and Control: A Neural Network Approach*. [S.l.]: Springer, 2001.

[13] HUANG, G.; ZHU, Q.-Y.; SIEW, C.-K. Extreme learning machine: Theory and applications. *Neurocomputing*, v. 70, p. 489–501, 2006.

[14] SUN, J. et al. Extreme learning control of surface vehicles with unknown dynamics and disturbances. *Neurocomputing*, v. 167, n. 1, p. 535–542, 2015.

[15] RONG, H.; ZHAO, G. Direct adaptive neural control of nonlinear systems with extreme learning machine. *Neural Comput. and Applic.*, v. 22, n. 3, p. 577–586, 2013.

[16] TANG, Y.; LI, Z.; GUAN, X. Identification of nonlinear system using extreme learning machine based hammerstein model. *Commun. Nonlinear Sci. Numer. Simulat.*, v. 19, n. 9, p. 3171–3182, 2014.

[17] RONG, H. et al. Adaptive neural control for a class of mimo nonlinear systems with extreme learning machine. *Neurocomputing*, v. 149, p. 405–414, 2015.

[18] JANAKIRAMAN, V.; NGUYEN, X.; ASSANIS, D. Stochastic gradient based extreme learning machines for online learning of advanced combustion engines. *Neurocomputing*, v. 177, p. 304–316, 2016.

[19] BAZAEI, A.; MOALLEM, M. Online neural identification of multi-input multi-output systems. *IET Control Theory Applications*, v. 1, n. 1, p. 44–50, 2007.

[20] VARGAS, J.; HEMERLY, E. Neural adaptive observer with asymptotic convergence in the presence of time-varying parameters and disturbances. *(in Portuguese) Sba Controle Autom.*, v. 19, p. 18–29, 2008.

[21] VARGAS, J.; HEMERLY, E.; VILLARREAL, E. Stability analysis of a neuro-identification scheme with asymptotic convergence. *International Journal of Artificial Intelligence and Applications*, v. 3, n. 4, p. 35–50, 2012.

[22] GAUSS, C. *Theoria Motus Corporum Coelestium*. [S.l.]: Dover Publications, 1809.

[23] DEISTLER, M. System identification and time series analysis. In: *Proc. Stochastic Theory and Control*. [S.l.]: Festschrift for Tyrone Duncan, 2002. p. 97–108.

[24] GEVERS, M. A personal view of the development of system identification. *IEEE Control Systems Magazine*, v. 26, n. 6, p. 93–105, 2006.

[25] HO, B.; KALMAN, R. Effective contruction of linear state-variable models from input-output functions. *Regelungstechnik*, v. 12, p. 545–548, 1965.

[26] ASTROM, K.; BOHLIN, T. Numerical identification of linear dynamic systems from normal operating records. In: *Proc. IFAC Symp. Self-Adaptive Systems*. [S.l.]: Teddington, U.K., 1992. p. 96–111.

[27] KOOPMANS, T.; RUBIN, H.; LEIPNIK, R. Measuring the equation systems of dynamic economics. *Cowles Comission Monograph*, v. 10, 1950.

[28] HANNAN, E. *Time Series Analysis*. [S.l.]: Methuen, 1960.

[29] BOX, G.; REINSEL, G. J. aand G. *Time Series Analysis: Forecasting and Control*. [S.l.]: John Wiley and Sons, 2008.

[30] KUNG, S. A new identification method and model reduction algorithm via singular value decomposition. In: *12th Asilomar Conf. on Circuits, Systems and Computers*. [S.l.]: Asilomar, CA, 1978. p. 705–714.

[31] AKAIKE, H. Stochastic theory of minimal realization. *IEEE Transaction on Automation and Control*, v. 19, n. 6, p. 667–674, 1974.

[32] LJUNG, L. On consistency and identifiability. *Math. Program Study*, v. 5, p. 169–190, 1976.

[33] ANDERSON, B.; MOORE, J.; HAWKES, R. Model approximation via prediction error identification. *Automatica*, v. 14, n. 6, p. 615–622, 1978.

[34] LJUNG, L.; CAINES, P. Asymptotic normality of prediction error estimators for approximative system models. *Stochastics*, v. 3, p. 29–46, 1979.

[35] LJUNG, L. Convergence analysis of parametric identification methods. *IEEE Transactions on Automatic Control*, v. 23, n. 5, p. 770–783, 1978.

[36] LJUNG, L. Asymptotic variance expressions for identified black-box transfer function estimation. *IEEE Transaction on Automation and Control*, v. 30, n. 9, p. 834–844, 1985.

[37] WAHLBERG, B.; LJUNG, L. Design variables for bias estimation in transfer function estimation. *IEEE Transaction on Automation and Control*, v. 31, n. 2, p. 133–144, 1986.

[38] GEVERS, M.; LJUNG, L. Optimal experiment designs with respect to the intended model application. *Automatica*, v. 22, n. 5, p. 543–554, 1986.

[39] LJUNG, L. *System Identification - Theory for the User*. [S.l.]: Prentice-Hall, 1999.

[40] POGGIO, T.; GIROSI, F. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, v. 247, n. 4945, p. 978–982, 1990.

[41] POGGIO, T.; GIROSI, F. Networks for approximation and learning. *Proceedings of the IEEE*, v. 78, n. 9, p. 1481–1497, 1990.

[42] ZADEH, L. Fuzzy logic, neural networks, and soft computing. *Communication ACM*, v. 37, n. 3, p. 77–86, 1994.

[43] WANG, L. *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*. [S.l.]: Pearson Prentice Hall, 1994.

[44] ZHANG, Q.; BENVENISTE, A. Wavelet networks. *IEEE Transactions on Neural Networks*, v. 3, n. 6, p. 889–898, 1992.

[45] LIU, G.; BILLINGS, S.; KADIRKAMANATHAN, V. Nonlinear system identification using wavelet networks. *Proceedings of the UKACC International Conference on Control*, p. 1248–1253, 1998.

[46] LIU, G.; BILLINGS, S.; KADIRKAMANATHAN, V. Identification of nonlinear dynamical systems using wavelet networks. *International Journal of Systems Science*, v. 31, p. 1531–1541, 2000.

[47] ANTSAKLIS, P. Special issue on neural networks in control systems. *IEEE Control Systems Magazine*, v. 10, p. 3–5, 1990.

[48] MILLER, T.; SUTTON, R.; WERBOS, P. *Neural Networks for Control*. [S.l.]: MIT Press, 1990.

[49] CHEN, S.; BILLINGS, S.; GRANT, P. Nonlinear system identification using neural networks. *International Journal of Control*, v. 51, n. 6, p. 1191–1214, 1990.

[50] NARENDRA, K.; PARTHASARATHY, K. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, v. 1, n. 1, p. 4–27, 1990.

[51] BILLINGS, S.; CHEN, S. Neural networks and system identification. *Neural Networks for Systems and Control*, v. 7, p. 181–205, 1992.

[52] QIN, S.; SU, H.; MCAVOY, T. Comparison of four net learning methods for dynamic system identification. *IEEE Transactions on Neural Networks*, v. 3, n. 1, p. 122–130, 1992.

[53] WILLIS, M. et al. Artificial neural networks in process estimation and control. *Automatica*, v. 28, n. 6, p. 1181–1187, 1992.

[54] KUSCHEWSKI, J.; HUI, S.; ZAK, S. Application of feedforward neural networks to dynamical system identification and control. *IEEE Transactions on Control Systems Technology*, v. 1, n. 1, p. 37–49, 1993.

[55] POLYCARPOU, M.; IOANNOU, P. *Identification and control of nonlinear systems using neural network models: design and stability analysis*. [S.l.], 1991. Tecnical Report 91-09-01.

[56] SANNER, R.; SLOTINE, J. Gaussian networks for direct adaptive control. *IEEE Transactions on Neural Networks*, v. 3, n. 6, p. 837–863, 1992.

[57] SADEGH, N. A perceptron network for functional identification and control of nonlinear systems. *IEEE Transactions on Neural Networks*, v. 4, n. 6, p. 982–988, 1993.

[58] WERBOS, R. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, v. 78, n. 10, p. 1550–1560, 1990.

[59] WILLIAMS, R.; ZIPSER, D. A learning algorithm for continuous running fully recurrent neural networks. *Neural Computation*, v. 1, n. 2, p. 270–280, 1989.

[60] NARENDRA, K.; PARTHASARATHY, K. Gradient methods for the optimization of dynamical systems containing neural networks. *IEEE transactions on Neural Networks*, v. 2, n. 2, p. 252–262, 1991.

[61] NARENDRA, K.; PARTHASARATHY, K. *Stable Adaptive Systems*. [S.l.]: Prentice-Hall, 1989.

[62] SLOTINE, J.; LI, W. *Applied Nonlinear Control*. [S.l.]: Prentice-Hall International, 1991.

[63] BEHERA, L.; KUMAR, S.; PATNAIK, A. On adaptive learning rate that guarantees convergence in feedforward networks. *IEEE Transactions on Neural Networks*, v. 17, n. 5, p. 1116–1125, 2006.

[64] MAN, Z. et al. A new adaptive backpropagation algorithm based on lyapunov stability theory for neural networks. *IEEE Transactions on Neural Networks*, v. 17, n. 6, p. 1580–1591, 2006.

[65] YU, X.; EFE, M.; KAYNAK, O. A general backpropagation algorithm for feedforward neural networks learning. *IEEE Transactions on Neural Networks*, v. 13, n. 1, p. 251–254, 2002.

[66] VARGAS, J.; GULARTE, K.; HEMERLY, E. Online neuro-identification of uncertain systems based on scaling and explicit feedback. *Control Automation and Electrical Systems*, v. 24, n. 6, p. 753–763, 2013.

[67] KOSMATOPOULOS, E. et al. High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, v. 6, n. 2, p. 422–431, 1995.

[68] ABDOLLAHI, F.; TALEBI, H.; PATEL, R. A stable neural network-based observer with application to flexible-joint manipulators. *IEEE Transactions on Neural Networks*, v. 17, n. 1, p. 118–129, 2006.

[69] YU, W.; RUBIO, J. Recurrent neural networks traininng with stable bounding ellipsoid algorithm. *IEEE Transactions on Neural Networks*, v. 20, n. 6, p. 983–991, 2009.

[70] RUBIO, J.; ANGELOV, P.; PACHECO, J. Uniformly stable backpropagation algorithm to train a feedforward neural network. *IEEE Transactions on Neural Networks*, v. 22, n. 3, p. 356–366, 2006.

[71] HUANG, G. et al. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man and Cybernetics*, v. 42, n. 2, p. 513–529, 2012.

[72] HUANG, G.; BABRI, H. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transactions on Neural Networks*, v. 17, n. 1, p. 879–892, 1998.

[73] WANG, Y.; CAO, F.; YUAN, Y. A study on effectiveness of extreme learning machine. *Neurocomputing*, v. 74, n. 16, p. 2483–2490, 2011.

[74] DING, S. et al. Extreme learning machine: algorithm, theory and applications. *Artif. Intell. Rev.*, v. 44, n. 1, p. 103–115, 2015.

[75] FARREL, J.; POLYCARPOU, M. *Adaptive Approximation Based Control: Unifying Neural, Fuzzy and Tradicional Adaptive Approximation Approaches*. [S.l.]: Wiley, 2006.

[76] HECHT-NIELSEN, R. Kolmogorov's mapping neural network existence theorem. *First IEEE International Conference on Neural Networks*, v. 3, p. 11–14, 1987.

[77] LIPPMANN, R. An introduction to computing with neural nets. *IEEE ASSP Magazine*, v. 4, n. 2, p. 4–22, 1987.

[78] SPRECHER, D. On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, v. 115, p. 340–355, 1965.

[79] GALLANT, A.; WHITE, H. There exists a neural network that does not make avoidable mistakes. *Neural Networks*, v. 2, p. 657–664, 1989.

[80] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems*, v. 2, n. 4, p. 303–314, 1989.

[81] HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, n. 5, p. 359–366, 1989.

[82] FUNAHASHI, K.-I. On the approximate realization of continuous mappings by neural networks. *IEEE International Conference on Neural Networks*, v. 1, n. 3, p. 183–192, 1989.

[83] HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, v. 3, n. 5, p. 551–560, 1990.

[84] LIGHT, W. Ridge functions, sigmoidal functions and neural networks. In: *In Approximation Theory VII*. [S.l.]: Academic Press, 1992. p. 163–206.

[85] SUN, X.; CHENEY, E. The fundamentals of sets of ridge functions. *Aequationes Math.*, v. 44, p. 226–235, 1992.

[86] STINCHCOMBE, M.; WHITE, H. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In: *International Joint Conference on Neural Networks*. [S.l.: s.n.], 1989. v. 1, p. 613–617.

[87] HORNIK, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, v. 4, n. 2, p. 251–257, 1991.

[88] LESHNO, M. et al. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, v. 6, n. 6, p. 861–867, 1993.

[89] HORNIK, K. Some new results on neural network approximation. *Neural Networks*, v. 6, n. 8, p. 1069–1072, 1993.

[90] KULAKOV, A.; ZWOLINSKI, M.; REEVE, J. Fault tolerance in distributed neural computing. *ArXiv e-prints*, v. 1509.09199, 2015.

[91] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, p. 386–408, 1958.

[92] HEBB, D. *The Organization of Behavior: A Neuropsychological Theory*. [S.l.]: Wiley, 1949.

[93] MINSKY, M.; SELFRIDGE, O. Learning in random nets. In: *Information Theory*. [S.l.]: 4th London Symposium, 1961.

[94] RUMELHART, D.; HINTON, G.; WILLIAMS, R. Learning internal representations by error propagation. *Parallel Distributed Processing*, v. 1, p. 318–362, 1986.

[95] BENGIO, Y. Learning deep architectures for a.i. *Foundations and Trends in Machine Learning*, v. 2, p. 1–127, 2009.

[96] BENGIO, Y.; COURVILLE, A. C.; VINCENT, P. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[97] SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, v. 61, p. 85–117, 2015.

[98] KARAYIANNIS, N.; VENETSANOPOULOS, A. *Artificial Neural Networks: Learning Algorithms, Performance Evaluation and Applications*. [S.l.]: Springer, 1993.

[99] GILES, C.; MAXWELL, T. Learning, invariance, and generalization in higher order neural networks. *Applied Optics*, v. 26, n. 23, p. 4972–4978, 1987.

[100] PARETTO, P.; NIEZ, J. Long term memory storage capacity of multiconnected neural networks. *Biological Cybernetics*, v. 54, p. 53–63, 1986.

[101] REDDING, N.; KOWALCZYK, A.; DOWNS, T. Constructive high-order network algorithm that is polynomial time. *Neural Networks*, v. 6, n. 7, p. 997–1010, 1993.

[102] ESTEVEZ, P.; OKABE, Y. Training the piecewise linear-high order neural network through error back propagation. In: *Proceedings of IEEE International Joint Conference on Neural Networks,*. [S.l.: s.n.], 1991. v. 1, p. 771–716.

[103] SPIRKOVSKA, L.; REID, M. Robust position, scale, and rotation invariant object recognition using higher-order neural networks. *Pattern Recognition*, v. 25, n. 9, p. 975–985, 1992.

[104] PARK, J.; SANDBERG, I. Universal approximation using radial-basis-function networks. *Neural Computation*, v. 3, n. 2, p. 246–257, 1990.

[105] HARTMAN, E.; KEELER, J.; KOWALSKI, J. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, v. 2, n. 2, p. 210–215, 1990.

[106] HUSH, D.; HORNE, B. Progress in supervised neural networks. *IEEE Signal Processing Magazine*, v. 10, n. 1, p. 8–39, 1993.

[107] ZADEH, L. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 3, n. 1, p. 28–44, 1973.

[108] WANG, L. X.; MENDEL, J. M. Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. *IEEE Transactions on Neural Networks*, v. 3, n. 5, p. 807–814, 1992.

[109] KOSKO, B. *Neural Networks and Fuzzy Systems*. [S.l.]: Pearson Prentice Hall, 1992.

[110] JANG, J.-S.; SUN, C.; MIZUTANI, E. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, v. 4, n. 1, p. 156–159, 1993.

[111] KECMAN, V.; PFEIFFER, B.-M. Exploiting the structural equivalence of learning fuzzy systems and radial basis function networks. In: *European Congress on Intelligent Techniques and Soft Computing (EUFIT)*. [S.l.]: Aachen, Germany, 1994. p. 58–66.

[112] NELLES, O. *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. [S.l.]: Springer, 2001.

[113] BARUCH, I. et al. A fuzzy-neural multi-model for nonlinear systems identification and control. *Fuzzy Sets and Systems*, v. 159, n. 20, p. 2650–2667, 2008.

[114] YU, L.; ZHANG, Y. Evolutionary fuzzy neural networks for hybrid financial prediction. *IEEE Transactions on Systems, Man and Cybernetics*, v. 35, n. 2, p. 244–249, 2005.

[115] THEODORIDIS, D.; BOUTALIS, Y.; CHRISTODOULOU, M. Neuro-fuzzy direct adaptive control of unknown nonlinear systems with analysis on the model order problem. *Journal of Zhejiang University*, v. 12, p. 1–16, 2011.

[116] CALDERON, A.; ZYGMUND, A. On existence of certain singular integral. *Acta Mathematics*, v. 88, p. 85–139, 1952.

[117] STRICHARTZ, R. How to make wavelets. *American Mathematical Monthly*, v. 100, n. 6, p. 539–556, 1993.

[118] RIOUL, O.; VETTERLI, M. Wavelets and signal processing. *IEEE Signal Processing Magazine*, v. 8, n. 4, p. 14–38, 1991.

[119] KREINOVICH, V.; SIRISAENGTAKSIN, O.; CABRERA, S. Wavelet neural networks are asymptotically optimal approximators for functions of one variable. In: *IEEE World Congress on Computational Intelligence*. [S.l.: s.n.], 1994. v. 1, p. 299–304 vol.1.

[120] BILLINGS, S.; WEI, H. A new class of wavelet networks for nonlinear system identification. *IEEE Transactions on Neural Networks*, v. 16, n. 4, p. 862–874, 2005.

[121] SJOBERG, J. et al. Nonlinear black-box modelling in system identification: a unified overview. *Automatica*, v. 31, n. 12, p. 1691–1724, 1995.

[122] KUMAR, P.; FOUFOULA-GEORGIOU, E. Multicomponent decomposition of spatial rainfall fields: segregation of large and small scale features using wavelet transforms. *Water Resources Research*, v. 29, n. 8, p. 2515–2532, 1993.

[123] ZHANG, Q. Learning algorithm of wavelet network based on sampling theory. *Neurocomputing*, v. 71, n. 1-3, p. 224–269, 2007.

[124] PANY, P.; GHOSHAL, S. Application of local linear wavelet neural network in short term electric load forecasting. *International Journal of Computer Applications*, v. 51, n. 13, 2012.

[125] CHEN, Y.; YANG, B.; DONG, J. Time-series prediction using a local linear wavelet neural wavelet. *Neurocomputing*, v. 69, n. 4-6, p. 449–465, 2006.

[126] WAI, R.; CHANG, H. Backstepping wavelet neural network control for indirect field-oriented induction motor drive. *IEEE Transactions on Neural Networks*, v. 15, n. 2, p. 367–382, 2004.

[127] HUANG, G.; WANG, D.; LUAN, Y. Extreme learning machine: Theory and applications. *International Journal of Machine Learning and Cybernetics*, v. 2, n. 1-3, p. 107–122, 2011.

[128] SUN, F. et al. *Extreme Learning Machines 2013: Algorithms and Applications*. [S.l.]: Springer, 2014.

[129] KAELBLING, L.; LITTMAN, M.; MOORE, A. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, v. 4, p. 237–285, 1996.

[130] ASTROM, K.; WITTENMARK, P. *Adaptive Control*. [S.l.]: Addison-Wesley, 1995.

[131] ZHAO, G. et al. On improving the conditioning of extreme learning machine: A linear case. *7th International Conference on Information, Communications and Signal Processing*, p. 1–5, 2009.

[132] Lü, J.; CHEN, D.; CELIKOVSKY, S. Bridge the gap between the lorenz system and chen system. *International Journal of Bifurcation and Chaos*, v. 12, n. 12, p. 2917–2926, 2002.

[133] YU, H.; CAI, G.; LI, Y. Dynamic analysis and control of a new hyperchaotic finance system. *Nonlinear Dynamics*, v. 67, n. 3, p. 2171–2182, 2012.

[134] MAHMOUD, E. Dynamics and synchronization of new hyperchaotic complex lorenz system. *Mathematical and Computer Modeling*, v. 55, n. 7-8, p. 1951–1962, 2012.

[135] GRZEIDAK, E.; VARGAS, J.; ALFARO, S. Online neuro-identification of nonlinear systems using extreme learning machine. In: *Accepted in International Joint Conference on Neural Networks IJCNN*. [S.l.]: Vancouver, Canada, 2016.

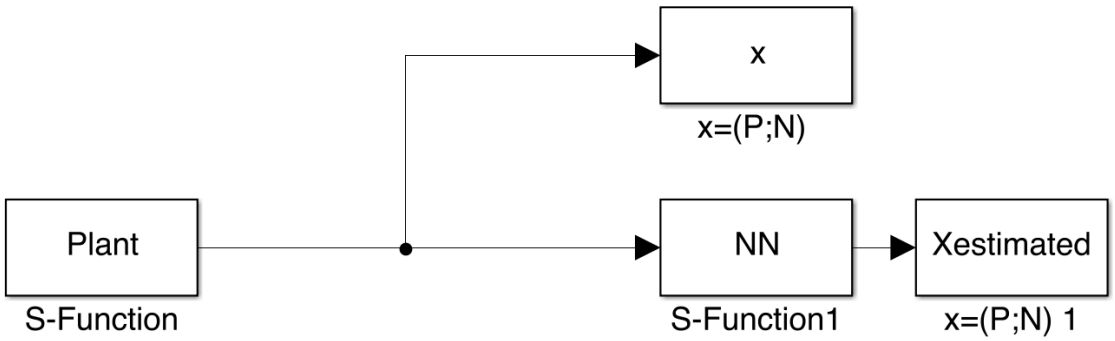
[136] BARRON, A. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, v. 39, n. 3, p. 930–945, 1993.

[137] VARGAS, J.; GRZEIDAK, E.; ALFARO, S. Identification of unknown nonlinear systems based on multilayer neural networks and lyapunov theory. In: *Submitted to IEEE Multi-Conference on Systems and Control*. [S.l.]: Buenos Aires, Argentina, 2016.

Appendix

I. CODES

I.1 Appendix 1 - Simulink plant used for simulations corresponding to Fig. 4.1-4.17 and Fig. 5.1-5.19



I.2 Appendix 2 - Code for plant model corresponding to Fig. 4.1-4.4

```
function [sys,x0,str,ts] = Plant(t,x,u,flag)

%Unified Chaotic System

a=1;  %Constant for the Chen System

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 3;   %Number of Continuous States
    sizes.NumDiscStates  = 0;   %Number of Discret States
    sizes.NumOutputs     = 3;   %Number of Outputs
    sizes.NumInputs      = 0;   %Number of Inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[-1.5 -2 -5];            %Initial Conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
```

```
   % Directives  %
   %%%%%%%%%%%%%%%%
case 1,
   %Unified chaotic system implementation, the chosen constant
   %generates a Chen system.
   sys = [(25*a+10)*(x(2)-x(1));
          (28-35*a)*x(1)-x(1)*x(3)+(29*a-1)*x(2);
          x(1)*x(2)-((a+8)/3)*x(3)]+disturb(x,u,t);
   %%%%%%%%%%
   % output %
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%
   %    End    %
   %%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the system's perturbations
%Arguments: state vector, input vector and time
function disturb = disturb(x,u,t)
if t>=5
    disturb=[3*sin(7*t)*norm(x(1:3));
             10*cos(9*t)*x(1);
             (cos(20*t)+10*exp(-t))];
else
    disturb=0; %Until t=5 secs the disturb is null
end
```

I.3 Appendix 3 - Code for identifier corresponding to Fig. 4.1-4.4

```
%Project description: Online Identification using hidden layer neural networks
%                      with adaptive laws
%Authors: Jose Alfredo Ruiz Vargas and Emerson Grzeidak
%Date: 04/2016          Local: University of Brasilia
function [sys,x0,str,ts] = iden_Caos(t,x,u,flag)

%Diagonal matrix A with negative elements
A = [-10.5 0 0; 0 -9 0; 0 0 -9.5];

%Diagonal matrix A with positive elements
B = 100*[4.1 0 0; 0 3.9 0; 0 0 4];
```

```
%Constant parameters for learning laws
gammaW=0.001;

gamma0 = 0.1;
gamma1 = 1;
gamma2 = 0.00001;

%Adjustment weights to the output layer
W0 = [0 0 0; 0 0 0; 0 0 0];

%Random generated matrix used for this simulation
%The matrix was generated using V0 = randn(3, 7)
%and the kept fixed for the simulation
V0 = [0.8442 0.8963 0.0968 0.3392 0.1062 0.5720 0.8575;
      0.9299 0.5364 0.2585 0.0694 0.8221 0.3607 0.2315;
      0.2341 0.2986 0.3643 0.5960 0.5773 0.7750 0.0110];

%Positive definite matrix P and matrix K
P = 500*[0.001 0 0; 0 0.001 0; 0 0 0.001];
K = P + P';

%Parameters for the sigmoidal function
alpha=100;
beta=1;

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%

case 0,

    sizes = simsizes;
    sizes.NumContStates  = 12;   %Number of continuous states
    sizes.NumDiscStates  = 0;    %Number of discrete states
    sizes.NumOutputs     = 16;   %Number of outputs
    sizes.NumInputs      = 3;    %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(12,1);              %Initial conditions
    x0(1:3)=5;                   %Initial conditions for the estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives  %
   %%%%%%%%%%%%%%%%

case 1,

      %Identification model and learning laws implementation
```

```
      sys = [A*x(1:3) + B*[x(4:6)'; x(7:9)'; x(10:12)']*Sig(x, u, alpha, beta, ...
          V0)-parameter_l(x, u, gamma0, gamma1, gamma2, K, t);
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 1) + w_term1(x, u, B, K, ...
                 1, alpha, beta, V0));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 2) + w_term1(x, u, B, K, ...
                 2, alpha, beta, V0));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 3) + w_term1(x, u, B, K, ...
                 3, alpha, beta, V0))];

   %%%%%%%%%%%%%%%%
   % Outputs      %
   %%%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:12);
              norm(x(4:12));
              norm((u(1)-x(1)));
              norm((u(2)-x(2)));
              norm((u(3)-x(3)))];
case {2,4,9},
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimation errors for the parameter W
%Arguments: state vector, initial weights for W and the desired line

function W_error = W_error(x, W0, line)

%The state vector is in column format, your transposed is needed
%to mount the estimation matrix W
temp = [x(4:6)'; x(7:9)'; x(10:12)'] - W0;

if line == 1
    W_error = temp(1,:)';
end
if line == 2
    W_error = temp(2,:)';
end
if line == 3
    W_error = temp(3,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the vector function l
%Arguments: state vector, system inputs, matrix K, time t and
%control parameters: gamma0, gamma1, gamma2
function parameter_l = parameter_l(x, u, gamma0, gamma1, gamma2, K, t)

denominator=(min(eig(K))*(norm(x_error(x,u))+gamma1*exp(-gamma2*t)));
```

```
parameter_l = - (gamma0*x_error(x,u))/denominator;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return a vector with estimated state's errors
%Arguments: state vector, system inputs

function x_error = x_error(x, u)

X = [x(1); x(2); x(3)];
U = [u(1); u(2); u(3)];

x_error = X - U;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the NN's nonlinear regressor vector
%Arguments: state vector, system inputs and sigmoidal function parameters

function Sig = Sig(x,u, alpha, beta, V0)     %Regressor

%Parameters for the activation function
V0Z = V0*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%A sigmoidal function is used, we pass each of the elements of the vector
Sig=[(z(V0Z(1), alpha, beta));
     (z(V0Z(2), alpha, beta));
     (z(V0Z(3), alpha, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Sigmoidal fucntion that returns the results of the regressor
%Arguments: the product VZ and parameters for the sigmoidal function

function z = z(arg, alpha, beta)  %Sigmoidal activation function

z=alpha/(1+exp(-beta*arg));   %Sigmoidal Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*S(VZ)
%Arguments: state vector, system inputs, matrix B and K,
%desired line of the matrix and parameters of sigmoidal function

function w_term1 = w_term1(x, u, B, K, line, alpha, beta, V0)

%term B*K*Xerror*S(VZ) 3x3
temp = B*K*x_error(x,u)*(Sig(x,u, alpha, beta, V0)');

if line == 1
    w_term1 = temp(1, :)';
end
if line == 2
    w_term1 = temp(2, :)';
end
if line == 3
```

```
    w_term1 = temp(3, :)';
end
```

I.4 Appendix 4 - Code to display the Fig. 4.1-4.4

```
%Displays the actual states and their estimations
fsize=22;
figure;
plot(t,x(:,1),'--',t,Xestimated(:,1),'LineWidth',2); set(0,'DefaultAxesFontSize', ...
    17);
h=legend('Sistema Caotico','NN');
set(h,'FontSize',fsize);
xlabel('Tempo(s)','Fontsize',fsize);
ylabel('x_{1}(t), x_{NN1}(t)','Fontsize',fsize);

figure;
plot(t,x(:,2),'--',t,Xestimated(:,2),'LineWidth',2); set(0,'DefaultAxesFontSize', ...
    17);
h=legend('Sistema Caotico','NN');
set(h,'FontSize',fsize); xlabel('Tempo(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{2}(t), x_{NN2}(t)','Fontsize',fsize);

figure;
plot(t,x(:,3),'--',t,Xestimated(:,3),'LineWidth',2); set(0,'DefaultAxesFontSize', ...
    17);
h=legend('Sistema Caotico','NN');
set(h,'FontSize',fsize);
xlabel('Tempo(s)','Fontsize',fsize);
ylabel('x_{3}(t), x_{NN3}(t)','Fontsize',fsize);

figure;
plot(t,Xestimated(:,13),'LineWidth',1);
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Tempo(s)','Fontsize',fsize);
ylabel('Estimated Weight Norm W','Fontsize',fsize);
```

I.5 Appendix 5 - Code for plant model corresponding to Fig. 4.5-4.9

```
function [sys,x0,str,ts] = Plant(t,x,u,flag)

%Hypercaotic Finance System
```

```
%Constants for the hyperchaotic system
a=0.9;
b=0.2;
c=1.2;
d=0.17;
k=0.17;

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 4;   %Number of continuous states
    sizes.NumDiscStates  = 0;   %Number of discret states
    sizes.NumOutputs     = 4;   %Number of outputs
    sizes.NumInputs      = 0;   %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[1 2.5 0.5 0.5];         %Initial conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives  %
   %%%%%%%%%%%%%%%%
case 1,
   %Unified chaotic system implementation, the chosen constant generates
   %a Chen system.
   sys = [x(3)+(x(2)-a)*x(1)+x(4);
          1-b*x(2)-x(1)*x(1);
          -x(1)-c*x(3);
          -d*x(1)*x(2)-k*x(4)]+disturb(x,u,t);

   %%%%%%%%%%
   % Output %
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%%
   %    End     %
   %%%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the system's perturbations
%Arguments: state vector, input vector and time
```

```
function disturb = disturb(x,u,t)
if t>=5
    disturb=[8*sin(7*t)*x(4);
             10*cos(9*t)*x(1)*x(1);
             x(2)*(cos(20*t)+5*exp(-t));
             2*sin(20*t)+3*cos(15*t)];
else
    disturb=0; %Until t=5 secs the disturb is null
end
```

I.6 Appendix 6 - Code for identifier corresponding to Fig. 4.5-4.9

```
%Project description: Online Identification using hidden layer neural networks
%                      with adaptive laws
%Authors: Jose Alfredo Ruiz Vargas and Emerson Grzeidak
%Date: 04/2015          Local: University of Brasilia
function [sys,x0,str,ts] = iden_Caos(t,x,u,flag)

%Diagonal matrix A with negative elements
A = [-29 0 0 0; 0 -30.5 0 0; 0 0 -30 0; 0 0 0 -28];

%Diagonal matrix A with positive elements
B = 100*[1.1 0 0 0; 0 0.8 0 0; 0 0 1.2 0; 0 0 0 1];

%Constant parameters for the learning laws
gammaW=0.01;

gamma0 = 0.1;
gamma1 = 1;
gamma2 = 0.0001;

%Adjustment weights to the output layer
W0 = [0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0];

%Random generated matrix used for this simulation
%The matrix was generated using V0 = randn(4, 9)
%and the kept fixed for the simulation
V0 = [0.5377 0.3188 0.5784 0.7254 0.1241 0.6715 0.4889 0.2939 0.0689;
      0.8339 0.3077 0.7694 0.0631 0.4897 0.2075 0.0347 0.7873 0.8095;
      0.2588 0.4336 0.3499 0.7147 0.4090 0.7172 0.7269 0.8884 0.9443;
      0.8622 0.3426 0.0349 0.2050 0.4172 0.6302 0.3034 0.1471 0.4384];

%Positive definite matrix P and Matrix K
P = 50*[0.001 0 0 0; 0 0.001 0 0; 0 0 0.001 0; 0 0 0 0.001];
K = P + P';

%Parameters for the sigmoidal function
```

```
alpha=150;
beta=1;

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%

case 0,

    sizes = simsizes;
    sizes.NumContStates  = 20;   %Number of continuous states
    sizes.NumDiscStates  = 0;    %Number of discrete states
    sizes.NumOutputs     = 25;   %Number of outputs
    sizes.NumInputs      = 4;    %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(20,1);              %Initial conditions
    x0(1:4)=[-2 -2 -2 -2];       %Initial conditions for the estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives  %
   %%%%%%%%%%%%%%%%

case 1,

      % Identification model and learning laws implementation
      sys = [A*x(1:4) + B*[x(5:8)'; x(9:12)'; x(13:16)'; x(17:20)']*Sig(x, u, ...
          alpha, beta, V0)-parameter_l(x, u, gamma0, gamma1, gamma2, K, t);
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 1) + w_term1(x, u, B, K, ...
                 1, alpha, beta, V0));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 2) + w_term1(x, u, B, K, ...
                 2, alpha, beta, V0));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 3) + w_term1(x, u, B, K, ...
                 3, alpha, beta, V0));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 4) + w_term1(x, u, B, K, ...
                 4, alpha, beta, V0))];

   %%%%%%%%%%%%%%%%
   % Outputs     %
   %%%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:20);
              norm(x(5:20));
              norm((u(1)-x(1)));
              norm((u(2)-x(2)));
              norm((u(3)-x(3)));
              norm((u(4)-x(4)))];
case {2,4,9},
```

```
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimation errors for the parameter W
%Arguments: state vector, initial weights for W and the desired line

function W_error = W_error(x, W0, line)

%The state vector is in column format, your transposed is needed
%to mount the estimation matrix W
temp = [x(5:8)'; x(9:12)'; x(13:16)'; x(17:20)'] - W0;

if line == 1
    W_error = temp(1,:)';
end
if line == 2
    W_error = temp(2,:)';
end
if line == 3
    W_error = temp(3,:)';
end
if line == 4
    W_error = temp(4,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the vector function l
%Arguments: state vector, system inputs, matrix K, time t and control
%parameters: gamma0, gamma1, gamma2
function parameter_l = parameter_l(x, u, gamma0, gamma1, gamma2, K, t)

denominator = (min(eig(K))*(norm(x_error(x,u))+gamma1*exp(-gamma2*t)));
parameter_l = - (gamma0*x_error(x,u))/denominator;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return a vector with estimated state's errors
%Arguments: state vector, system inputs

function x_error = x_error(x, u)

X = [x(1); x(2); x(3); x(4)];
U = [u(1); u(2); u(3); u(4)];

x_error = X - U;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the NN's nonlinear regressor vector
%Arguments: state vector, system inputs and sigmoidal function parameters
```

```
function Sig = Sig(x,u, alpha, beta, V0)     %Regressor

%Parameters for the activation function
V0Z = V0*[u(1); u(2); u(3); u(4); u(1)^2; u(2)^2; u(3)^2; u(4)^2; 1];

%A sigmoidal function is used, we pass each of the elements of the vector
Sig=[(z(V0Z(1), alpha, beta));
     (z(V0Z(2), alpha, beta));
     (z(V0Z(3), alpha, beta));
     (z(V0Z(4), alpha, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Sigmoidal fucntion that returns the results of the regressor
%Arguments: the product VZ and parameters for the sigmoidal function

function z = z(arg, alpha, beta)  %Sigmoidal activation function

z=alpha/(1+exp(-beta*arg));   %Sigmoidal Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*S(VZ)
%Arguments: state vector, system inputs, matrix B and K,
%desired line of the matrix and parameters of sigmoidal function

function w_term1 = w_term1(x, u, B, K, line, alpha, beta, V0)

%term B*K*Xerror*S(VZ) 3x3
temp = B*K*x_error(x,u)*(Sig(x,u, alpha, beta, V0)');

if line == 1
    w_term1 = temp(1, :)';
end
if line == 2
    w_term1 = temp(2, :)';
end
if line == 3
    w_term1 = temp(3, :)';
end
if line == 4
    w_term1 = temp(4, :)';
end
```

I.7 Appendix 7 - Code to display the Fig. 4.5-4.9

```
%Displays the actual states and their estimations
fsize=22;
figure;
```

```
plot(t,x(:,1), '--',t,Xestimated(:,1),'LineWidth',2);set(0,'DefaultAxesFontSize',17);
h=legend('Sistema Caotico','NN');
set(h,'FontSize',fsize);
xlabel('Tempo(s)','Fontsize',fsize);
ylabel('x_{1}(t), x_{NN1}(t)','Fontsize',fsize);

figure;
plot(t,x(:,2), '--',t,Xestimated(:,2),'LineWidth',2);set(0,'DefaultAxesFontSize',17);
h=legend('Sistema Caotico','NN');
set(h,'FontSize',fsize); xlabel('Tempo(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{2}(t), x_{NN2}(t)','Fontsize',fsize);

figure;
plot(t,x(:,3), '--',t,Xestimated(:,3),'LineWidth',2);set(0,'DefaultAxesFontSize',17);
h=legend('Sistema Caotico','NN');
set(h,'FontSize',fsize);
xlabel('Tempo(s)','Fontsize',fsize);
ylabel('x_{3}(t), x_{NN3}(t)','Fontsize',fsize);

figure;
plot(t,x(:,4),'--',t,Xestimated(:,4),'LineWidth',2);set(0,'DefaultAxesFontSize',17);
h=legend('Sistema Caotico','NN');
set(h,'FontSize',fsize);
xlabel('Tempo(s)','Fontsize',fsize);
ylabel('x_{4}(t), x_{NN4}(t)','Fontsize',fsize);

figure;
plot(t, Xestimated(:,21), 'LineWidth', 1 );
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Tempo(s)','Fontsize',fsize);
ylabel('Estimated Weight Norm W','Fontsize',fsize);
```

I.8 Appendix 8 - Code for plant model corresponding to Fig. 4.10-4.17

```
function [sys,x0,str,ts] = Plant(t,x,u,flag)

%Complex hyperchaotic system

 %Constant parameters for complex chaotic system
alpha=14;
beta=3;
gamma=50;
k1=-5;
k2=-4;
```

```
switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization  %
   %%%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 7;   %Number of constant states
    sizes.NumDiscStates  = 0;   %Number of discret states
    sizes.NumOutputs     = 7;   %Number of outputs
    sizes.NumInputs      = 0;   %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[0 1 2 3 4 5 6];         %Initial conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%%
   % Directives  %
   %%%%%%%%%%%%%%%%%
case 1,
   %Implementation of the Hyperchaotic Complex System
   sys = [alpha*(x(3)-x(1));
          alpha*(x(4)-x(2));
          gamma*x(1)-x(1)*x(5)-x(3)+x(6);
          gamma*x(2)-x(2)*x(5)-x(4)+x(7);
          x(1)*x(3)+x(2)*x(4)-beta*x(5);
          k1*x(1)+k2*x(3);
          k1*x(2)+k2*x(4)]+disturb(x,u,t);
   %%%%%%%%%%
   % Output %
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%
   %    END    %
   %%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the system's perturbations
%Arguments: state vector, input vector and time
function disturb = disturb(x,u,t)
if t>=5
n=sqrt(x(1)^2 + x(2)^2 + x(3)^2+x(4)^2+x(5)^2 + x(6)^2 + x(7)^2);

disturb=2*[n*sin(t);
        n*1.2*sin(2*t);
```

```
        n*cos(4*t);
        n*1.2*sin(t);
        n*1.1*sin(2*t) ;
        n*0.5*sin(4*t);
        exp(-0.5*t)];
else
disturb=[0 ; 0; 0; 0; 0; 0; 0]; %Until t=5 secs the disturb is null
end
```

I.9 Appendix 9 - Code for identifier corresponding to Fig. 4.10-4.17

```
%Project description: Online Identification using hidden layer neural networks
%                     with adaptive laws
%Authors: Jose Alfredo Ruiz Vargas and Emerson Grzeidak
%Date: 04/2015          Local: University of Brasilia
function [sys,x0,str,ts] = iden_Caos(t,x,u,flag)

%Diagonal matrix A with negative elements
A = 16*[-1.1 0 0 0 0 0 0;
     0 -0.9 0 0 0 0 0;
     0 0 -1.2 0 0 0 0;
     0 0 0 -1.3 0 0 0;
     0 0 0 0 -1 0 0;
     0 0 0 0 0 -0.8 0;
     0 0 0 0 0 0 -1];

%Diagonal matrix A with positive elements
B = 50*[1 0 0 0 0 0 0;
         0 0.9 0 0 0 0 0;
         0 0 1.2 0 0 0 0;
         0 0 0 0.9 0 0 0;
         0 0 0 0 1 0 0;
         0 0 0 0 0 0.8 0;
         0 0 0 0 0 0 1];

%Positive parameters for learning laws
gammaW=0.1;
lambdaW=1;

gamma0 = 0;
gamma1 = 1;
gamma2 = 0.00001;

%Adjustment weights to the output layer
W0 = [0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
```

```
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0];

%Random generated matrix used for this simulation
%The matrix was generated using V0 = randn(7, 8)
%and the kept fixed for the simulation
VR = [0.3404 0.8909 0.2543 0.6160 0.2858 0.5308 0.3371 0.2630;
      0.5853 0.9593 0.8143 0.4733 0.7572 0.7792 0.1622 0.6541;
      0.2238 0.5472 0.2435 0.3517 0.7537 0.9340 0.7943 0.6892;
      0.7513 0.1386 0.9293 0.8308 0.3804 0.1299 0.3112 0.7482;
      0.2551 0.1493 0.3500 0.5853 0.5678 0.5688 0.5285 0.4505;
      0.5060 0.2575 0.1966 0.5497 0.0759 0.4694 0.1656 0.0838;
      0.6991 0.8407 0.2511 0.9172 0.0540 0.0119 0.6020 0.2290];

%Positive definite matrix P and matrix K
P = 0.05*[1 0 0 0 0 0 0;
        0 1 0 0 0 0 0;
        0 0 1 0 0 0 0;
        0 0 0 1 0 0 0;
        0 0 0 0 1 0 0;
        0 0 0 0 0 1 0;
        0 0 0 0 0 0 1];
K = P + P';

%Parameters for the sigmoidal function
alpha=150;
beta=0.1;

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%

case 0,

    sizes = simsizes;
    sizes.NumContStates  = 56;   %Number of continuous states
    sizes.NumDiscStates  = 0;    %Number of discrete states
    sizes.NumOutputs     = 8;    %Number of outputs
    sizes.NumInputs      = 7;    %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(56,1);              %Initial conditions
    x0(1:7)=[-20 -30 -40 20 40 50 40];  %Initial conditions for the estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%
   % Directives  %
```

```
   %%%%%%%%%%%%%%%

case 1,

      % Identification model and learning laws implementation
      sys = [A*x(1:7) + B*[x(8:14)'; x(15:21)'; x(22:28)'; x(29:35)'; x(36:42)'; ...
          x(43:49)'; x(50:56)']*Sig(x, u, alpha, beta, VR)-parameter_l(x, u, ...
          gamma0, gamma1, gamma2, K, t);
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 1) + w_term1(x, u, B, K, ...
                 1, alpha, beta, VR));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 2) + w_term1(x, u, B, K, ...
                 2, alpha, beta, VR));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 3) + w_term1(x, u, B, K, ...
                 3, alpha, beta, VR));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 4) + w_term1(x, u, B, K, ...
                 4, alpha, beta, VR));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 5) + w_term1(x, u, B, K, ...
                 5, alpha, beta, VR));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 6) + w_term1(x, u, B, K, ...
                 6, alpha, beta, VR));
             -gammaW*(norm(x_error(x, u))*W_error(x, W0, 7) + w_term1(x, u, B, K, ...
                 7, alpha, beta, VR))];

   %%%%%%%%%%%%%%%
   % Outputs     %
   %%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:7);
              norm([x(8:14)'; x(15:21)'; x(22:28)'; x(29:35)'; x(36:42)'; ...
                  x(43:49)'; x(50:56)'],'fro')];

case {2,4,9},
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimation errors for the parameter W
%Arguments: state vector, initial weights for W and the desired line

function W_error = W_error(x, W0, line)

%The state vector is in column format, your transposed is needed
%to mount the estimation matrix W
temp = [x(8:14)'; x(15:21)'; x(22:28)'; x(29:35)'; x(36:42)'; x(43:49)'; ...
    x(50:56)'] - W0;

if line == 1
    W_error = temp(1,:)';
end
```

```
if line == 2
    W_error = temp(2,:)';
end
if line == 3
    W_error = temp(3,:)';
end
if line == 4
    W_error = temp(4,:)';
end
if line == 5
    W_error = temp(5,:)';
end
if line == 6
    W_error = temp(6,:)';
end
if line == 7
    W_error = temp(7,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the vector function l
%Arguments: state vector, system inputs, matrix K, time t and control
%parameters: gamma0, gamma1, gamma2
function parameter_l = parameter_l(x, u, gamma0, gamma1, gamma2, K, t)

parameter_l = - ...
    (gamma0*x_error(x,u))/(min(eig(K))*(norm(x_error(x,u))+gamma1*exp(-gamma2*t)));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return a vector with estimated state's errors
%Arguments: state vector, system inputs

function x_error = x_error(x, u)

X = [x(1); x(2); x(3); x(4); x(5); x(6); x(7)];
U = [u(1); u(2); u(3); u(4); u(5); u(6); u(7)];

x_error = X - U;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the NN's nonlinear regressor vector
%Arguments: state vector, system inputs and sigmoidal function parameters

function Sig = Sig(x,u, alpha, beta, VR)     %Regressor

%Parameters for the activation function
VZ = VR*[u(1); u(2); u(3); u(4); u(5); u(6); u(7); 1];

%A sigmoidal function is used, we pass each of the elements of the vector
Sig=[(z(VZ(1), alpha, beta));
     (z(VZ(2), alpha, beta));
```

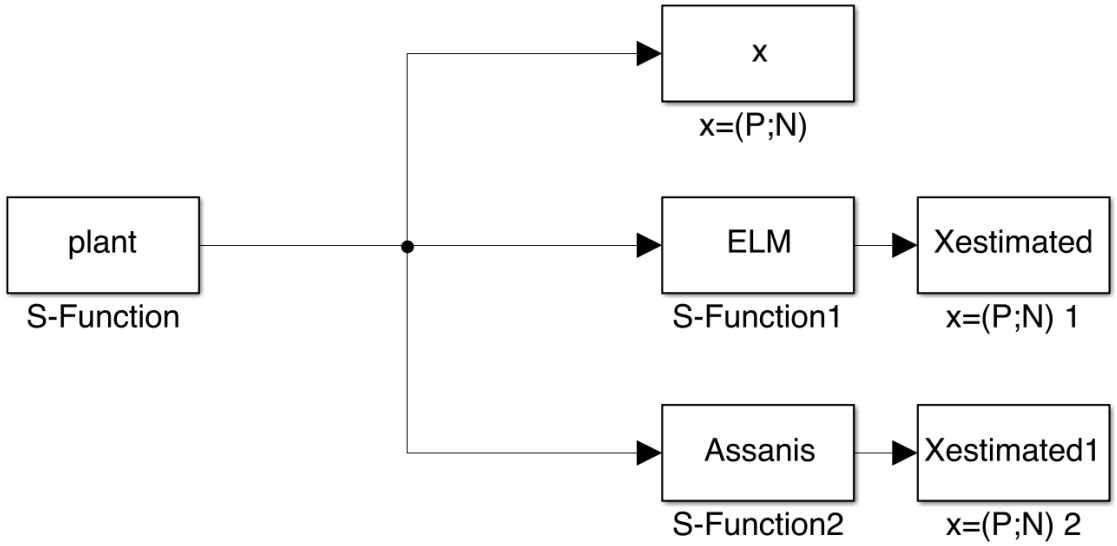
```
     (z(VZ(3), alpha, beta));
     (z(VZ(4), alpha, beta));
     (z(VZ(5), alpha, beta));
     (z(VZ(6), alpha, beta));
     (z(VZ(7), alpha, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Sigmoidal fucntion that returns the results of the regressor
%Arguments: the product VZ and parameters for the sigmoidal function

function z = z(arg, alpha, beta)  %Sigmoidal activation function

z=alpha/(1+exp(-beta*arg));   %Sigmoidal Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*S(VZ)
%Arguments: state vector, system inputs, matrix B and K,
%desired line of the matrix and parameters of sigmoidal function

function w_term1 = w_term1(x, u, B, K, line, alpha, beta, VR)

%term B*K*Xerror*S(VZ) 3x3
temp = B*K*x_error(x,u)*(Sig(x,u, alpha, beta, VR)');

if line == 1
    w_term1 = temp(1, :)';
end
if line == 2
    w_term1 = temp(2, :)';
end
if line == 3
    w_term1 = temp(3, :)';
end
if line == 4
    w_term1 = temp(4, :)';
end
if line == 5
    w_term1 = temp(5, :)';
end
if line == 6
    w_term1 = temp(6, :)';
end
if line == 7
    w_term1 = temp(7, :)';
end
```

I.10 Appendix 10 - Code to display the Fig. 4.10-4.17

```
%Displays the actual states and their estimations
fsize=22;
figure;
plot(t,Xestimated(:,1),'--',t,x(:,1),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{1}(t), x_{NN1}(t)','Fontsize',fsize);

figure;
plot(t,x(:,2),'--',t,Xestimated(:,2),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{2}(t), x_{NN2}(t)','Fontsize',fsize);

figure;
plot(t,x(:,3),'--',t,Xestimated(:,3),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{3}(t), x_{NN3}(t)','Fontsize',fsize);

figure;
plot(t,x(:,4),'--',t,Xestimated(:,4),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{4}(t), x_{NN4}(t)','Fontsize',fsize);

figure;
plot(t,x(:,5),'--',t,Xestimated(:,5),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{5}(t), x_{NN5}(t)','Fontsize',fsize);

figure;
plot(t,x(:,6),'--',t,Xestimated(:,6),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{6}(t), x_{NN6}(t)','Fontsize',fsize);

figure;
```

```
plot(t,x(:,7),'--',t,Xestimated(:,7),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{7}(t), x_{NN7}(t)','Fontsize',fsize);

figure;
plot(t, Xestimated(:,8), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated weight matrix W','Fontsize',fsize);
```

I.11 Appendix 11 - Simulink plant used for simulations corresponding to Fig. 4.18-4.22



I.12 Appendix 12 - Code for plant model corresponding to Fig. 4.18-4.22

```
function [sys,x0,str,ts] = Plant(t,x,u,flag)

%Unified Chaotic System

a=0;  %Constant for Lorenz System

switch flag,
```

```
   %%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 3;   %Number of Constant States
    sizes.NumDiscStates  = 0;   %Number of Discret States
    sizes.NumOutputs     = 3;   %Number of outputs
    sizes.NumInputs      = 0;   %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[1.5 2 5];                 %Initial conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Derivatives %
   %%%%%%%%%%%%%%%%
case 1,
   %implementation of the unified chaotic system, the chosen value
   %for a generates a Lorenz system.
   sys = [(25*a+10)*(x(2)-x(1));
          (28-35*a)*x(1)-x(1)*x(3)+(29*a-1)*x(2);
          x(1)*x(2)-((a+8)/3)*x(3)]+disturb(x,u,t);

   %%%%%%%%%%
   % output %
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%
   %   end     %
   %%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the system's perturbations
%Arguments: state vector, input vector and time
function disturb = disturb(x,u,t)
if t>=5
    n=sqrt(x(1)^2 + x(2)^2 + x(3)^2);
    disturb=n*[3*sin(7*t);
             10*cos(9*t);
             (cos(20*t)+10*exp(-t))];
else
    disturb=0; %Until t=5 secs the disturb is null
end
```

I.13 Appendix 13 - Code for identifier in literature [1] corresponding to Fig. 4.18-4.22

```
%Project description: Lyapunov Method Based Online Identification of
%                     Nonlinear Systems Using Extreme Learning Machine
%Authors: V.M. Janakiraman and D. Assanis
%Date: 2012
function [sys,x0,str,ts] = assanis(t,x,u,flag)

%Diagonal matrix A with negative elements
A = [-60 0 0; 0 -60 0; 0 0 -120];

%V0 = randn(3, 12)
V0 = [0.6948    0.0344    0.7655    0.4898    0.7094    0.6797    0.1190    ...
    0.3404    0.7513    0.6991    0.5472    0.2575;
      0.3171    0.4387    0.7952    0.4456    0.7547    0.6551    0.4984    ...
          0.5853    0.2551    0.8909    0.1386    0.8407;
      0.9502    0.3816    0.1869    0.6463    0.2760    0.1626    0.9597    ...
          0.2238    0.5060    0.9593    0.1493    0.2543];

%Parameters for the sigmoidal function
alfa=150;
beta=0.001;

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%

case 0,

    sizes = simsizes;
    sizes.NumContStates  = 12;   %Number of continuous states
    sizes.NumDiscStates  = 0;    %Number of discrete states
    sizes.NumOutputs     = 16;   %Number of outputs
    sizes.NumInputs      = 3;    %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(12,1);              %Initial conditions
    x0(1:3)=5;
    %Initial conditions for the estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives  %
```

```
   %%%%%%%%%%%%%%%

case 1,

      sys = [A*x(1:3) + [x(4:6) x(7:9) x(10:12)]*Sig(x, u, alfa, beta, V0);
             -W_estimated(x, u, alfa, beta, V0, 1);
             -W_estimated(x, u, alfa, beta, V0, 2);
             -W_estimated(x, u, alfa, beta, V0, 3)];

   %%%%%%%%%%%%%%%
   % Outputs     %
   %%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:12);
              norm(x(4:12));
              norm((u(1)-x(1)));
              norm((u(2)-x(2)));
              norm((u(3)-x(3)))];
case {2,4,9},
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimation errors for the parameter W
%Arguments: state vector, initial weights for W and the desired line

function W_estimated = W_estimated(x, u, alfa, beta, V0, linha)

%The state vector is in column format, your transposed is needed
%to mount the estimation matrix W
temp = Sig(x, u, alfa, beta, V0)*(x_error(x, u));

if linha == 1
    W_estimated = temp(1,:)';
end
if linha == 2
    W_estimated = temp(2,:)';
end
if linha == 3
    W_estimated = temp(3,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return a vector with estimated state's errors
%Arguments: state vector, system inputs

function x_error = x_error(x, u)

X = [x(1); x(2); x(3)];
```

```
U = [u(1); u(2); u(3)];

x_error = (X - U)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the NN's nonlinear regressor vector
%Arguments: state vector, system inputs and sigmoidal function parameters

function Sig = Sig(x,u, alfa, beta, V0)     %Regressor

%Parameters for the activation function
V0Z = V0*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; u(1)*u(2); u(1)*u(3); ...
    u(2)*u(3); u(1)*u(2)*u(3); u(1)^2*u(2); u(1)^2*u(3)];

%A sigmoidal function is used, we pass each of the elements of the vector
Sig=[(z(V0Z(1), alfa, beta));
     (z(V0Z(2), alfa, beta));
     (z(V0Z(3), alfa, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Sigmoidal fucntion that returns the results of the regressor
%Arguments: the product VZ and parameters for the sigmoidal function

function z = z(arg, alfa, beta)  %Sigmoidal activation function

z=alfa/(1+exp(-beta*arg));   %Sigmoidal Function
```

I.14 Appendix 14 - Code for proposed identifier corresponding to Fig. 4.18-4.22

The code is the same as Appendix 3.

I.15 Appendix 15 - Code to display the Fig. 4.18-4.22

```
set(0,'DefaultAxesColorOrder',[0 0 1; 1 0 0; 0 0 0]);

fsize=14;
figure;
plot(t, Xestimated1(:,14),t, Xestimated(:,14), 'LineWidth', 2 );
axis([0 10 -0.5 inf]);
set(0,'DefaultAxesFontSize', 17);
h=legend('Algorithm in [1]','Proposed Algorithm');
set(h,'FontSize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
figure;
plot(t, Xestimated1(:,15),t, Xestimated(:,15), 'LineWidth', 2 );
axis([0 10 -0.5 inf]);
set(0,'DefaultAxesFontSize', 17);
h=legend('Algorithm in [1]','Proposed Algorithm');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('State Error Norm of y(t)','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;
plot(t, Xestimated1(:,16),t, Xestimated(:,16), 'LineWidth', 2 );
axis([0 10 -0.5 inf]);
set(0,'DefaultAxesFontSize', 17);
h=legend('Algorithm in [1]','Proposed Algorithm');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('State Error Norm of z(t)','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%
set(0,'DefaultAxesColorOrder',[0 0 0; 1 0 0; 0 0 0]);

figure;
plot(t, Xestimated1(:,13), 'LineWidth', 2 );
h=legend('Algorithm in [1]')
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated Weight Norm W','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;
plot(t, Xestimated(:,13), 'LineWidth', 2 );
h=legend('Algorithm in [1]')
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated Weight Norm W','Fontsize',fsize);
```

I.16 Appendix 16 - Code for plant model corresponding to Fig. 5.1-5.5

```
function [sys,x0,str,ts] = plant(t,x,u,flag)

%Unified Chaotic System
```

```
a=1;  %Constant for Chen System

switch flag,
   %%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 3;   %Number of Continuous States
    sizes.NumDiscStates  = 0;   %Number of Discret States
    sizes.NumOutputs     = 3;   %Number of Outputs
    sizes.NumInputs      = 0;   %Number of Inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[2 1 2];                 %Initial Conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives %
   %%%%%%%%%%%%%%%%
case 1,
   %Implementation of unified chaotic system, the chosen constant
   %generates a chen system.
   sys(1) = (25*a+10)*(x(2)-x(1));
   sys(2) = (28-35*a)*x(1)-x(1)*x(3)+(29*a-1)*x(2);
   sys(3) = x(1)*x(2)-((a+8)/3)*x(3);
   %%%%%%%%%%
   % Output %
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%
   %    End    %
   %%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
end
```

I.17 Appendix 17 - Code for identifier corresponding to Fig. 5.1-5.5

```
%Project description: Online Identification using hidden layer neural networks
```

```
%                       with adaptive laws
%Authors: Jose Alfredo Ruiz Vargas and Emerson Grzeidak
%Date: 04/2015           Local: University of Brasilia
function [sys,x0,str,ts] = iden_Caos(t,x,u,flag)

%Diagonal matrix A with negative elements
A = [-7.8 0 0; 0 -7.8 0; 0 0 -7.8];

%Diagonal matrix A with positive elements
B = 110*[1.1 0 0; 0 1.16 0; 0 0 1.3];

%Positive parameters for the adaptation laws
gammaW=0.02;
gammaV=0.001;

gamma0 = 0.1;
gamma1 = 1;
gamma2 = 1;

alphaW=0.5;
alphaV=0.5;

l_zero = 1;

%Adjustment weights to the output layer
W0 = [0 0 0; 0 0 0; 0 0 0];

%Adjustment weights to the hidden layer
V0 = [0 0 0 0 0 0 0; 0 0 0 0 0 0 0; 0 0 0 0 0 0 0];

%Positive definite matrix P
P = 50*[0.001 0 0; 0 0.001 0; 0 0 0.001];

%Matrix K
K = P + P';

%Numerator parameter for the sigmoidal function
alpha=85;

%Denominator parameter for the sigmoidal function
beta=1;

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%

case 0,

    sizes = simsizes;
    sizes.NumContStates  = 33;   %Number of continuous states
    sizes.NumDiscStates  = 0;    %Number of discrete states
```

```
    sizes.NumOutputs     = 39;   %Number of outputs
    sizes.NumInputs      = 3;    %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(33,1);              %Initial conditions
    x0(1:3)=5;                   %Initial conditions for the estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives  %
   %%%%%%%%%%%%%%%%

case 1,

      %Identification model and learning laws implementation
      sys = [A*x(1:3) + B*[x(4:6)'; x(7:9)'; x(10:12)']*Sig(x, u, alpha, ...
          beta)-parameter_l(x, u, gamma0, gamma1, gamma2, K, t)-l_zero*x_error(x, u);
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 1) + w_term1(x, ...
                 u, B, K, 1, alpha, beta) - w_term2(x, u, B, K, 1, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 2) + w_term1(x, ...
                 u, B, K, 2, alpha, beta) - w_term2(x, u, B, K, 2, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 3) + w_term1(x, ...
                 u, B, K, 3, alpha, beta) - w_term2(x, u, B, K, 3, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 1) + v_term2(x, ...
                 u, B, K, 1, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 2) + v_term2(x, ...
                 u, B, K, 2, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 3) + v_term2(x, ...
                 u, B, K, 3, alpha, beta))];

   %%%%%%%%%%%%%%%%
   % Outputs     %
   %%%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:33); norm([x(4:6)'; x(7:9)'; x(10:12)'],'fro');
           norm([x(13:19)'; x(20:26)'; x(27:33)'], 'fro');
           norm([x(4:33)], 'fro'); norm((u(1)-x(1)));
           norm((u(2)-x(2))); norm((u(3)-x(3)))];
case {2,4,9},
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimation errors for the W
%Arguments: state vector, initial weights for W and the desired line

function W_error = W_error(x, W0, line)
```

```
%The state vector is in column format, your transposed is needed
%to mount the estimation matrix W
temp = [x(4:6)'; x(7:9)'; x(10:12)'] - W0;

if line == 1
    W_error = temp(1,:)';
end
if line == 2
    W_error = temp(2,:)';
end
if line == 3
    W_error = temp(3,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the vector function l
%Arguments: state vector, system inputs, matrix K, time t and control
%parameters: gamma0, gamma1, gamma2
function parameter_l = parameter_l(x, u, gamma0, gamma1, gamma2, K, t)

if t<=3
    parameter_l = - ...
        (gamma0*x_error(x,u))/(min(eig(K))*(norm(x_error(x,u))+gamma1*exp(-gamma2*t)));
else
    parameter_l = 0.001;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return a vector with estimated state's errors
%Arguments: state vector, system inputs

function x_error = x_error(x, u)

X = [x(1); x(2); x(3)];
U = [u(1); u(2); u(3)];

x_error = X - U;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of matrix of estimated errors for hidden layer  V
%Arguments: state vector, initial weights for V and the desired line

function V_error = V_error(x, V0, line)

%The state vector is in column format, your transposed is needed to mount
%the estimation matrix for the hidden layer weights V
temp = [x(13:19)'; x(20:26)'; x(27:33)']-V0;

if line == 1
    V_error = temp(1,:)';
end
if line == 2
```

```
    V_error = temp(2,:)';
end
if line == 3
    V_error = temp(3,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the NN's nonlinear regressor vector
%Arguments: state vector, system inputs and sigmoidal function parameters

function Sig = Sig(x,u, alpha, beta)     %Regressor

%Parameters for the activation function
VZ = [x(13:19)'; x(20:26)'; x(27:33)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%A sigmoidal function is used, we pass each of the elements of the vector
Sig=[(z(VZ(1), alpha, beta));
     (z(VZ(2), alpha, beta));
     (z(VZ(3), alpha, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Sigmoidal fucntion that returns the results of the regressor
%Arguments: the product VZ and parameters for the sigmoidal function

function z = z(arg, alpha, beta)  %Sigmoidal activation function

z=alpha/(1+exp(-beta*arg));   %Sigmoidal Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the derivative of the NN's nonlinear regressor
%Arguments: state vectors, system inputs and sigmoidal function parameters

function Sigdot = Sigdot(x,u, alpha, beta)     %Regressor

%Parameters for the activation function
VZ = [x(13:19)'; x(20:26)'; x(27:33)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%A sigmoidal function is used, passing each of the elements and creating a
%diagonal matrix
Sigdot=[zdot(VZ(1), alpha, beta) 0 0;
        0 zdot(VZ(2), alpha, beta) 0;
        0 0 zdot(VZ(3), alpha, beta)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Derivative of the sigmoidal function that returns the regressor results
%Arguments: the product VZ, parameters for sigmoidal function

%Derivative of the sigmoidal activation function
function zdot = zdot(arg,alpha,beta)

%derivative of the sigmoidal function
zdot=(alpha*exp(-beta*arg))/(1+exp(-beta*arg)^2);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*S(VZ)
%Arguments: state vector, system inputs, matrix B and K, desired line of the ...
    matrix and parameters of sigmoidal function

function w_term1 = w_term1(x, u, B, K, line, alpha, beta)

%term B*K*Xerror*S(VZ) 3x3
temp = B*K*x_error(x,u)*(Sig(x,u, alpha, beta)');

if line == 1
    w_term1 = temp(1, :)';
end
if line == 2
    w_term1 = temp(2, :)';
end
if line == 3
    w_term1 = temp(3, :)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*(Sdot(VZ)*VZ)'
%Arguments: state vector, system inputs, matrix B and K,
%desired line and parameters of sigmoidal function

function w_term2 = w_term2(x, u, B, K, line, alpha, beta)

%Parameters for the activation function
VZ = [x(13:19)'; x(20:26)'; x(27:33)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%Term B*K*Xerror*(Sdot(VZ)*VZ)'
temp = B*K*x_error(x,u)*((Sigdot(x,u, alpha, beta)*VZ)');

if line == 1
    w_term2 = temp(1, :)';
end
if line == 2
    w_term2 = temp(2, :)';
end
if line == 3
    w_term2 = temp(3, :)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of Sdot(VZ)*W'*B*K*Xerror*Z'
%Arguments: state vector, system inputs, matrix B and K,
%desired line and parameters of sigmoidal function

function v_term2 = v_term2(x, u, B, K, line, alpha, beta)

%The vector of states is in column format, the following matrix is already transposed
```

```
W_transposed = [x(4:6) x(7:9) x(10:12)];

Z = [u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%Term Sdot(VZ)*W'*B*K*Xerror*Z'
temp = Sigdot(x,u, alpha, beta)'*W_transposed*B*K*x_error(x,u)*(Z');

if line == 1
    v_term2 = temp(1, :)';
end
if line == 2
    v_term2 = temp(2, :)';
end
if line == 3
    v_term2 = temp(3, :)';
end
```

I.18 Appendix 18 - Code to display the Fig. 5.1-5.5

```
%Displays the actual states and their estimations
fsize=22;
figure;
plot(t,x(:,1),'--',t,Xestimated(:,1),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{1}(t), x_{NN1}(t)','Fontsize',fsize);

figure;
plot(t,x(:,2),'--',t,Xestimated(:,2),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{2}(t), x_{NN2}(t)','Fontsize',fsize);

figure;
plot(t,x(:,3),'--',t,Xestimated(:,3),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{3}(t), x_{NN3}(t)','Fontsize',fsize);

figure;
plot(t, Xestimated(:,34), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
```

```
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated weight matrix W','Fontsize',fsize);

figure;
plot(t, Xestimated(:,35), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated weight matrix V','Fontsize',fsize);
```

I.19 Appendix 19 - Code for plant model corresponding to Fig. 5.6-5.10

```
function [sys,x0,str,ts] = Plant(t,x,u,flag)

% Unified chaotic system

a=1;  %Constant for chen system

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization  %
   %%%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 3;   %Number of Constant States
    sizes.NumDiscStates  = 0;   %Number of Discret States
    sizes.NumOutputs     = 3;   %Number of Outputs
    sizes.NumInputs      = 0;   %Number of Inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[2 1 2];                 %Initial Conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives %
   %%%%%%%%%%%%%%%%
case 1,
   %Implementation of Unified Chaotic System, the constant value generates
   %a Chen System.
   sys(1) = (25*a+10)*(x(2)-x(1));
   sys(2) = (28-35*a)*x(1)-x(1)*x(3)+(29*a-1)*x(2)+disturb(x,u,t);
   sys(3) = x(1)*x(2)-((a+8)/3)*x(3)+disturb(x,u,t);
   %%%%%%%%%%
   % Outputs %
```

```
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%
   % End       %
   %%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the system's perturbations
%Arguments: state vector, input vector and time
function disturb = disturb(x,u,t)
if t>=5
    disturb=3*sin(t)*(sqrt(x(1)^2 + x(2)^2 + x(3)^2))+50*sin(200*t)+10*cos(400*t);
else
    disturb=0; %Until t=5 secs the disturb is null
end
```

I.20 Appendix 20 - Code for identifier corresponding to Fig. 5.6-5.10

```
%Project description: Online Identification using hidden layer neural
%                     networks with adaptive laws under disturbances
%Authors: Jose Alfredo Ruiz Vargas and Emerson Grzeidak
%Date: 04/2015          Local: University of Brasilia
function [sys,x0,str,ts] = iden_Caos(t,x,u,flag)

%Diagonal matrix A with negative elements
A = [-7.8 0 0; 0 -7.8 0; 0 0 -7.8];

%Diagonal matrix A with positive elements
B = 110*[1.1 0 0; 0 1.16 0; 0 0 1.3];

%Positive constants for fine tunning of the neural network
gammaW=0.02;
gammaV=0.001;

gamma0 = 0.1;
gamma1 = 1;
gamma2 = 1;

alphaW=0.5;
```

```
alphaV=0.5;

l_zero = 1;

%Adjustment weights to the output layer
W0 = [0 0 0; 0 0 0; 0 0 0];

%Adjustment weights to the hidden layer
V0 = [0 0 0 0 0 0 0; 0 0 0 0 0 0 0; 0 0 0 0 0 0 0];

%Positive definite matrix P
P = 50*[0.001 0 0; 0 0.001 0; 0 0 0.001];

%Matrix K
K = P + P';

%Numerator parameter for the sigmoidal function
alpha=85;

%Denominator parameter for the sigmoidal function
beta=1;

switch flag,
   %%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%

case 0,

    sizes = simsizes;
    sizes.NumContStates  = 33;   %Number of continuous states
    sizes.NumDiscStates  = 0;    %Number of discrete states
    sizes.NumOutputs     = 39;   %Number of outputs
    sizes.NumInputs      = 3;    %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(33,1);              %Initial conditions
    x0(1:3)=5;
    %Initial conditions for the estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives  %
   %%%%%%%%%%%%%%%%

case 1,

      %Identification Model and Learning Laws implementation
      sys = [A*x(1:3) + B*[x(4:6)'; x(7:9)'; x(10:12)']*Sig(x, u, alpha, ...
          beta)-parameter_l(x, u, gamma0, gamma1, gamma2, K, t)-l_zero*x_error(x, u);
```

```
            -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 1) + w_term1(x, ...
                u, B, K, 1, alpha, beta) - w_term2(x, u, B, K, 1, alpha, beta));
            -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 2) + w_term1(x, ...
                u, B, K, 2, alpha, beta) - w_term2(x, u, B, K, 2, alpha, beta));
            -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 3) + w_term1(x, ...
                u, B, K, 3, alpha, beta) - w_term2(x, u, B, K, 3, alpha, beta));
            -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 1) + v_term2(x, ...
                u, B, K, 1, alpha, beta));
            -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 2) + v_term2(x, ...
                u, B, K, 2, alpha, beta));
            -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 3) + v_term2(x, ...
                u, B, K, 3, alpha, beta))];

   %%%%%%%%%%%%%%%
   % Outputs     %
   %%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:33); norm([x(4:6)'; x(7:9)'; x(10:12)'],'fro'); ...
           norm([x(13:19)'; x(20:26)'; x(27:33)'], 'fro'); norm([x(4:33)], 'fro'); ...
           norm((u(1)-x(1))); norm((u(2)-x(2))); norm((u(3)-x(3)))];
case {2,4,9},
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimation errors for the parameter W
%Arguments: state vector, initial weights for W and the desired line

function W_error = W_error(x, W0, line)

%The state vector is in column format, your transposed is needed
%to mount the estimation matrix W
temp = [x(4:6)'; x(7:9)'; x(10:12)'] - W0;

if line == 1
    W_error = temp(1,:)';
end
if line == 2
    W_error = temp(2,:)';
end
if line == 3
    W_error = temp(3,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the vector function l
%Arguments: state vector, system inputs, matrix K, time t and control
%parameters: gamma0, gamma1, gamma2
function parameter_l = parameter_l(x, u, gamma0, gamma1, gamma2, K, t)
```

```
if t<=3
    parameter_l = - ...
        (gamma0*x_error(x,u))/(min(eig(K))*(norm(x_error(x,u))+gamma1*exp(-gamma2*t)));
else
    parameter_l = 0.001;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return a vector with estimated state's errors
%Arguments: state vector, system inputs

function x_error = x_error(x, u)

X = [x(1); x(2); x(3)];
U = [u(1); u(2); u(3)];

x_error = X - U;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimated errors for hidden layer ...
    weights V
%Arguments: state vector, initial weights for hidden layer V and the desired line

function V_error = V_error(x, V0, line)

%The state vector is in column format, your transposed is needed to mount
%the estimation matrix for the hidden layer weights V
temp = [x(13:19)'; x(20:26)'; x(27:33)']-V0;

if line == 1
    V_error = temp(1,:)';
end
if line == 2
    V_error = temp(2,:)';
end
if line == 3
    V_error = temp(3,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the NN's nonlinear regressor vector
%Arguments: state vector, system inputs and sigmoidal function parameters

function Sig = Sig(x,u, alpha, beta)     %Regressor

%Parameters for the activation function
VZ = [x(13:19)'; x(20:26)'; x(27:33)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%A sigmoidal function is used, we pass each of the elements of the vector
Sig=[(z(VZ(1), alpha, beta));
     (z(VZ(2), alpha, beta));
```

```
     (z(VZ(3), alpha, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Sigmoidal fucntion that returns the results of the regressor
%Arguments: the product VZ and parameters for the sigmoidal function

function z = z(arg, alpha, beta)  %Sigmoidal activation function

z=alpha/(1+exp(-beta*arg));   %Sigmoidal Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the derivative of the NN's nonlinear regressor
%Arguments: state vectors, system inputs and sigmoidal function parameters

function Sigdot = Sigdot(x,u, alpha, beta)     %Regressor

%Parameters for the activation function
VZ = [x(13:19)'; x(20:26)'; x(27:33)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%A sigmoidal function is used, passing each of the elements and creating a
%diagonal matrix
Sigdot=[zdot(VZ(1), alpha, beta) 0 0;
        0 zdot(VZ(2), alpha, beta) 0;
        0 0 zdot(VZ(3), alpha, beta)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Derivative of the sigmoidal function that returns the regressor results
%Arguments: the product VZ, parameters for sigmoidal function

%Derivative of the sigmoidal activation function
function zdot = zdot(arg,alpha,beta)

%derivative of the sigmoidal function
zdot=(alpha*exp(-beta*arg))/(1+exp(-beta*arg)^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerro*S(VZ)
%Arguments: state vector, system inputs, matrix B and K,
%desired line of the matrix and parameters of sigmoidal function

function w_term1 = w_term1(x, u, B, K, line, alpha, beta)

%term B*K*Xerror*S(VZ) 3x3
temp = B*K*x_error(x,u)*(Sig(x,u, alpha, beta)');

if line == 1
    w_term1 = temp(1, :)';
end
if line == 2
    w_term1 = temp(2, :)';
end
if line == 3
```

```
    w_term1 = temp(3, :)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*(Sdot(VZ)*VZ)'
%Arguments: state vector, system inputs, matrix B and K,
%desired line and parameters of sigmoidal function

function w_term2 = w_term2(x, u, B, K, line, alpha, beta)

%Parameters for the activation function
VZ = [x(13:19)'; x(20:26)'; x(27:33)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%Term B*K*Xerror*(Sdot(VZ)*VZ)'
temp = B*K*x_error(x,u)*((Sigdot(x,u, alpha, beta)*VZ)');

if line == 1
    w_term2 = temp(1, :)';
end
if line == 2
    w_term2 = temp(2, :)';
end
if line == 3
    w_term2 = temp(3, :)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of Sdot(VZ)*W'*B*K*Xerror*Z'
%Arguments: state vector, system inputs, matrix B and K,
%desired line and parameters of sigmoidal function

function v_term2 = v_term2(x, u, B, K, line, alpha, beta)

%The vector of states is in column format, the following matrix is already transposed
W_transposed = [x(4:6) x(7:9) x(10:12)];

Z = [u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; 1];

%Term Sdot(VZ)*W'*B*K*Xerror*Z'
temp = Sigdot(x,u, alpha, beta)'*W_transposed*B*K*x_error(x,u)*(Z');

if line == 1
    v_term2 = temp(1, :)';
end
if line == 2
    v_term2 = temp(2, :)';
end
if line == 3
    v_term2 = temp(3, :)';
end
```

I.21 Appendix 21 - Code to display the Fig. 5.6-5.10

```
%Displays the actual states and their estimations
fsize=22;
figure;
plot(t,x(:,1),'--',t,Xestimated(:,1),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{1}(t), x_{NN1}(t)','Fontsize',fsize);

figure;
plot(t,x(:,2),'--',t,Xestimated(:,2),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{2}(t), x_{NN2}(t)','Fontsize',fsize);

figure;
plot(t,x(:,3),'--',t, Xestimated(:,3),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{3}(t), x_{NN3}(t)','Fontsize',fsize);

figure;
plot(t,Xestimated(:,34),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated weight matrix W','Fontsize',fsize);

figure;
plot(t,Xestimated(:,35),'LineWidth',2);
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated weight matrix V','Fontsize',fsize);
```

I.22 Appendix 22 - Code for plant model corresponding to Fig. 5.11-5.19

```
function [sys,x0,str,ts] = plant(t,x,u,flag)

% Hyperchaotic Complex System

%System's parameters constants
alpha=14;
beta=3;
gamma=50;
k1=-5;
k2=-4;

switch flag,
   %%%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 7;   %Number of Continuous States
    sizes.NumDiscStates  = 0;   %Number of Discret States
    sizes.NumOutputs     = 7;   %Number of Outputs
    sizes.NumInputs      = 0;   %Number of Inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[0 1 2 3 4 5 6];         %Initial Conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives %
   %%%%%%%%%%%%%%%%
case 1,
   %Hyperchaotic Complex System equations
   sys = [alpha*(x(3)-x(1));
          alpha*(x(4)-x(2));
          gamma*x(1)-x(1)*x(5)-x(3)+x(6);
          gamma*x(2)-x(2)*x(5)-x(4)+x(7);
          x(1)*x(3)+x(2)*x(4)-beta*x(5);
          k1*x(1)+k2*x(3);
          k1*x(2)+k2*x(4)]+disturb(x,u,t);
   %%%%%%%%%%
   % Output %
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%%
   %   END      %
   %%%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the system's perturbations
%Arguments: state vector, input vector and time
function disturb = disturb(x,u,t)
if t>=5
n=sqrt(x(1)^2 + x(2)^2 + x(3)^2+x(4)^2+x(5)^2 + x(6)^2 + x(7)^2);

disturb=1.8*[n*sin(t);
        n*1.2*sin(2*t);
        n*cos(4*t);
        n*1.2*sin(t);
        n*1.1*sin(2*t) ;
        n*0.5*sin(4*t);
        exp(-0.5*t)];
else
disturb=[0 ; 0; 0; 0; 0; 0; 0]; %Before t=5s, the disturbance is null
end
```

I.23 Appendix 23 - Code for identifier corresponding to Fig. 5.11-5.19

```
%Project description: Online Identification using hidden layer neural networks
%                      with adaptive laws
%Authors: Jose Alfredo Ruiz Vargas and Emerson Grzeidak
%Date: 04/2016          Local: University of Brasilia
function [sys,x0,str,ts] = iden_Caos(t,x,u,flag)


%Diagonal matrix A with negative elements
A = 10*[-2 0 0 0 0 0 0;
       0 -2 0 0 0 0 0;
       0 0 -2 0 0 0 0;
       0 0 0 -2 0 0 0;
       0 0 0 0 -2 0 0;
       0 0 0 0 0 -2 0;
       0 0 0 0 0 0 -2];

%Diagonal matrix A with positive elements
B = 100*[1 0 0 0 0 0 0;
         0 1 0 0 0 0 0;
         0 0 1 0 0 0 0;
         0 0 0 1 0 0 0;
         0 0 0 0 1 0 0;
         0 0 0 0 0 1 0;
         0 0 0 0 0 0 1];
```

```
% Parameters for the identification model and adaptive laws
gammaW=0.001;
gammaV=1;

gamma0 = 0.1;
gamma1 = 1;
gamma2 = 1;

alphaW=0.5;
alphaV=0.5;

l0 = 10;

%Adjustment weights to the output layer
W0 = [0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0;
      0 0 0 0 0 0 0];

%Adjustment weights to the hidden layer
V0 = [0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0];

%Positive definite matrix P
P = 0.05*[1 0 0 0 0 0 0;
          0 1 0 0 0 0 0;
          0 0 1 0 0 0 0;
          0 0 0 1 0 0 0;
          0 0 0 0 1 0 0;
          0 0 0 0 0 1 0;
          0 0 0 0 0 0 1];

%Matrix K
K = P + P';

%Numerator parameter for the sigmoidal function
alpha=150;
%Denominator parameter for the sigmoidal function
beta=1;

switch flag,
   %%%%%%%%%%%%%%%%%%
```

```
   % Initialization %
   %%%%%%%%%%%%%%%%%%%

case 0,

    sizes = simsizes;
    sizes.NumContStates  = 112;   %Number of continuous states
    sizes.NumDiscStates  = 0;     %Number of discrete states
    sizes.NumOutputs     = 9;     %Number of outputs
    sizes.NumInputs      = 7;     %Number of inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(112,1);              %Initial conditions
    x0(1:7)=[-50 -30 -40 20 40 50 40]; %Initial conditions for the estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Directives  %
   %%%%%%%%%%%%%%%%

case 1,

      %Identification Model and Learning Laws implementation
      sys = [A*x(1:7) + B*[x(8:14)'; x(15:21)'; x(22:28)'; x(29:35)'; x(36:42)'; ...
          x(43:49)'; x(50:56)']*Sig(x, u, alpha, beta)-parameter_l(x, u, gamma0, ...
          gamma1, gamma2, K, t)-l0*x_error(x,u);
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 1) + w_term1(x, ...
                 u, B, K, 1, alpha, beta) - w_term2(x, u, B, K, 1, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 2) + w_term1(x, ...
                 u, B, K, 2, alpha, beta) - w_term2(x, u, B, K, 2, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 3) + w_term1(x, ...
                 u, B, K, 3, alpha, beta) - w_term2(x, u, B, K, 3, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 4) + w_term1(x, ...
                 u, B, K, 4, alpha, beta) - w_term2(x, u, B, K, 4, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 5) + w_term1(x, ...
                 u, B, K, 5, alpha, beta) - w_term2(x, u, B, K, 5, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 6) + w_term1(x, ...
                 u, B, K, 6, alpha, beta) - w_term2(x, u, B, K, 6, alpha, beta));
             -gammaW*(2*alphaW*norm(x_error(x, u))*W_error(x, W0, 7) + w_term1(x, ...
                 u, B, K, 7, alpha, beta) - w_term2(x, u, B, K, 7, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 1) + v_term2(x, ...
                 u, B, K, 1, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 2) + v_term2(x, ...
                 u, B, K, 2, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 3) + v_term2(x, ...
                 u, B, K, 3, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 4) + v_term2(x, ...
                 u, B, K, 4, alpha, beta));
             -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 5) + v_term2(x, ...
                 u, B, K, 5, alpha, beta));
```

```
            -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 6) + v_term2(x, ...
                u, B, K, 6, alpha, beta));
            -gammaV*(2*alphaV*norm(x_error(x, u))*V_error(x, V0, 7) + v_term2(x, ...
                u, B, K, 7, alpha, beta))];

   %%%%%%%%%%%%%%%%
   % Outputs     %
   %%%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:7);
              norm([x(8:14)'; x(15:21)'; x(22:28)'; x(29:35)'; x(36:42)'; ...
                  x(43:49)'; x(50:56)'],'fro');
              norm([x(57:64)'; x(65:72)'; x(73:80)'; x(81:88)'; x(89:96)'; ...
                  x(97:104)'; x(105:112)'],'fro')];
case {2,4,9},
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimation errors for the parameter W
%Arguments: state vector, initial weights for W and the desired line

function W_error = W_error(x, W0, line)

%The state vector is in column format, your transposed is needed
%to mount the estimation matrix W
temp = [x(8:14)'; x(15:21)'; x(22:28)'; x(29:35)'; x(36:42)'; x(43:49)'; ...
    x(50:56)'] - W0;

if line == 1
    W_error = temp(1,:)';
end
if line == 2
    W_error = temp(2,:)';
end
if line == 3
    W_error = temp(3,:)';
end
if line == 4
    W_error = temp(4,:)';
end
if line == 5
    W_error = temp(5,:)';
end
if line == 6
    W_error = temp(6,:)';
end
if line == 7
    W_error = temp(7,:)';
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the vector function l
%Arguments: state vector, system inputs, matrix K, time t and control
%parameters: gamma0, gamma1, gamma2
function parameter_l = parameter_l(x, u, gamma0, gamma1, gamma2, K, t)

if t<=3
    parameter_l = - ...
        (gamma0*x_error(x,u))/(min(eig(K))*(norm(x_error(x,u))+gamma1*exp(-gamma2*t)));
else
    parameter_l = 0.001;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return a vector with estimated state's errors
%Arguments: state vector, system inputs

function x_error = x_error(x, u)

X = [x(1); x(2); x(3); x(4); x(5); x(6); x(7)];
U = [u(1); u(2); u(3); u(4); u(5); u(6); u(7)];

x_error = X - U;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of the matrix of estimated errors
%for hidden layer weights V
%Arguments: state vector, initial weights for hidden layer V and the desired line

function V_error = V_error(x, V0, line)

%The state vector is in column format, your transposed is needed to mount
%the estimation matrix for the hidden layer weights V
temp = [x(57:64)'; x(65:72)'; x(73:80)'; x(81:88)'; x(89:96)'; x(97:104)'; ...
    x(105:112)']-V0;

if line == 1
    V_error = temp(1,:)';
end
if line == 2
    V_error = temp(2,:)';
end
if line == 3
    V_error = temp(3,:)';
end
if line == 4
    V_error = temp(4,:)';
end
if line == 5
    V_error = temp(5,:)';
```

```
end
if line == 6
    V_error = temp(6,:)';
end
if line == 7
    V_error = temp(7,:)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the NN's nonlinear regressor vector
%Arguments: state vector, system inputs and sigmoidal function parameters

function Sig = Sig(x,u, alpha, beta)     %Regressor

%Parameters for the activation function
VZ = [x(57:64)'; x(65:72)'; x(73:80)'; x(81:88)'; x(89:96)'; x(97:104)'; ...
    x(105:112)' ]*[u(1); u(2); u(3); u(4); u(5); u(6); u(7); 1];

%A sigmoidal function is used, we pass each of the elements of the vector
Sig=[(z(VZ(1), alpha, beta));
     (z(VZ(2), alpha, beta));
     (z(VZ(3), alpha, beta));
     (z(VZ(4), alpha, beta));
     (z(VZ(5), alpha, beta));
     (z(VZ(6), alpha, beta));
     (z(VZ(7), alpha, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Sigmoidal fucntion that returns the results of the regressor
%Arguments: the product VZ and parameters for the sigmoidal function

function z = z(arg, alpha, beta)  %Sigmoidal activation function

z=alpha/(1+exp(-beta*arg));   %Sigmoidal Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the derivative of the NN's nonlinear regressor
%Arguments: state vectors, system inputs and sigmoidal function parameters

function Sigdot = Sigdot(x,u, alpha, beta)     %Regressor

%Parameters for the activation function
VZ = [x(57:64)'; x(65:72)'; x(73:80)'; x(81:88)'; x(89:96)'; x(97:104)'; ...
    x(105:112)']*[u(1); u(2); u(3); u(4); u(5); u(6); u(7); 1];

%A sigmoidal function is used, passing each of the elements and creating a
%diagonal matrix
Sigdot=[zdot(VZ(1), alpha, beta) 0 0 0 0 0 0;
        0 zdot(VZ(2), alpha, beta) 0 0 0 0 0;
        0 0 zdot(VZ(3), alpha, beta) 0 0 0 0;
        0 0 0 zdot(VZ(4), alpha, beta) 0 0 0;
        0 0 0 0 zdot(VZ(5), alpha, beta) 0 0;
```

```
        0 0 0 0 0 zdot(VZ(6), alpha, beta) 0;
        0 0 0 0 0 0 zdot(VZ(7), alpha, beta)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Derivative of the sigmoidal function that returns the regressor results
%Arguments: the product VZ, parameters for sigmoidal function

%Derivative of the sigmoidal activation function
function zdot = zdot(arg,alpha,beta)

%derivative of the sigmoidal function
zdot=(alpha*exp(-beta*arg))/(1+exp(-beta*arg)^2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*S(VZ)
%Arguments: state vector, system inputs, matrix B and K,
%desired line of the matrix and parameters of sigmoidal function

function w_term1 = w_term1(x, u, B, K, line, alpha, beta)

%term B*K*Xerror*S(VZ) 3x3
temp = B*K*x_error(x,u)*(Sig(x,u, alpha, beta)');

if line == 1
    w_term1 = temp(1, :)';
end
if line == 2
    w_term1 = temp(2, :)';
end
if line == 3
    w_term1 = temp(3, :)';
end
if line == 4
    w_term1 = temp(4, :)';
end
if line == 5
    w_term1 = temp(5, :)';
end
if line == 6
    w_term1 = temp(6, :)';
end
if line == 7
    w_term1 = temp(7, :)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of B*K*Xerror*(Sdot(VZ)*VZ)'
%Arguments: state vector, system inputs, matrix B and K, %desired line and ...
    parameters of sigmoidal function

function w_term2 = w_term2(x, u, B, K, line, alpha, beta)
```

```
%Parameters for the activation function
VZ = [x(57:64)'; x(65:72)'; x(73:80)'; x(81:88)'; x(89:96)'; x(97:104)'; ...
    x(105:112)' ]*[u(1); u(2); u(3); u(4); u(5); u(6); u(7); 1];

%Term B*K*Xerror*(Sdot(VZ)*VZ)'
temp = B*K*x_error(x,u)*((Sigdot(x,u, alpha, beta)*VZ)');

if line == 1
    w_term2 = temp(1, :)';
end
if line == 2
    w_term2 = temp(2, :)';
end
if line == 3
    w_term2 = temp(3, :)';
end
if line == 4
    w_term2 = temp(4, :)';
end
if line == 5
    w_term2 = temp(5, :)';
end
if line == 6
    w_term2 = temp(6, :)';
end
if line == 7
    w_term2 = temp(7, :)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the line of Sdot(VZ)*W'*B*K*Xerror*Z'
%Arguments: state vector, system inputs, matrix B and K,
%desired line and parameters of sigmoidal function

function v_term2 = v_term2(x, u, B, K, line, alpha, beta)

%The vector of states is in column format, the following matrix is already transposed
W_transposed = [x(8:14) x(15:21) x(22:28) x(29:35) x(36:42) x(43:49) x(50:56)];

Z = [u(1); u(2); u(3); u(4); u(5); u(6); u(7); 1];

%Term Sdot(VZ)*W'*B*K*Xerror*Z'
temp = Sigdot(x,u, alpha, beta)'*W_transposed*B*K*x_error(x,u)*(Z');

if line == 1
    v_term2 = temp(1, :)';
end
if line == 2
    v_term2 = temp(2, :)';
end
if line == 3
    v_term2 = temp(3, :)';
```

```
end
if line == 4
    v_term2 = temp(4, :)';
end
if line == 5
    v_term2 = temp(5, :)';
end
if line == 6
    v_term2 = temp(6, :)';
end
if line == 7
    v_term2 = temp(7, :)';
end
```

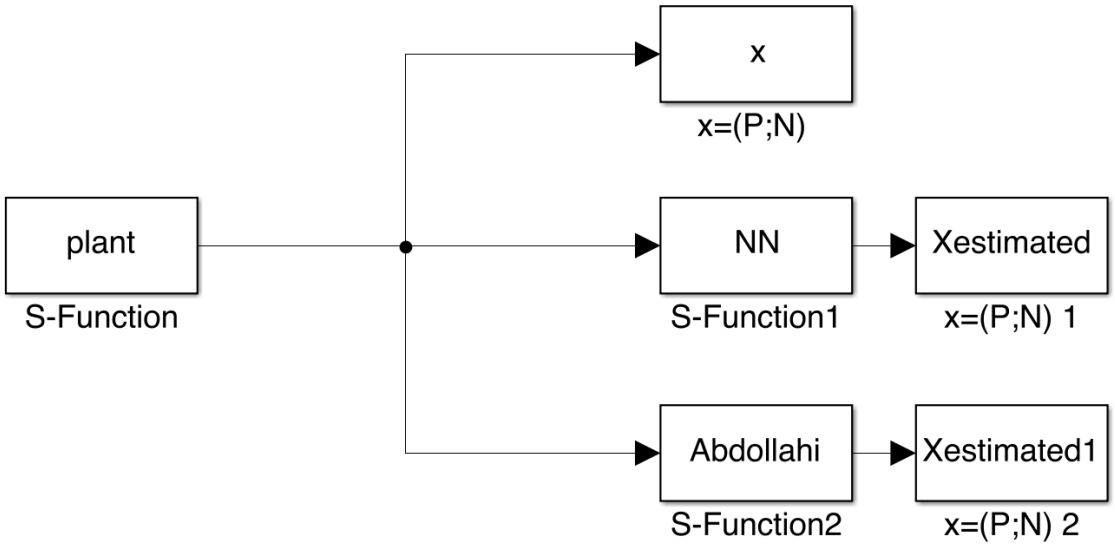
I.24 Appendix 24 - Code to display the Fig. 5.11-5.19

```
%Displays the actual states and their estimations
fsize=22;
figure;
plot(t, x(:,1), '--',t, Xestimated(:,1), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{1}(t), x_{NN1}(t)','Fontsize',fsize);

figure;
plot(t, x(:,2), '--',t, Xestimated(:,2), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{2}(t), x_{NN2}(t)','Fontsize',fsize);

figure;
plot(t, x(:,3), '--',t, Xestimated(:,3), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{3}(t), x_{NN3}(t)','Fontsize',fsize);

figure;
plot(t, x(:,4), '--',t, Xestimated(:,4), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
```

```
ylabel('x_{4}(t), x_{NN4}(t)','Fontsize',fsize);

figure;
plot(t, x(:,5), '--',t, Xestimated(:,5), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{5}(t), x_{NN5}(t)','Fontsize',fsize);

figure;
plot(t, x(:,6), '--',t, Xestimated(:,6), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{6}(t), x_{NN6}(t)','Fontsize',fsize);

figure;
plot(t, x(:,7), '--',t, Xestimated(:,7), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
h=legend('Actual','Estimated');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('x_{7}(t), x_{NN7}(t)','Fontsize',fsize);

figure;
plot(t, Xestimated(:,8), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated weight matrix W','Fontsize',fsize);

figure;
plot(t, Xestimated(:,9), 'LineWidth', 2 );
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated weight matrix V','Fontsize',fsize);
```

I.25 Appendix 25 - Simulink plant used for simulations corresponding to Fig. 5.20-5.26

I.26 Appendix 26 - Code for plant model corresponding to Fig. 5.20-5.26




```
function [sys,x0,str,ts] = plant(t,x,u,flag)

% Unified chaotic system

a=1;  %Constant chosen for Chen System

switch flag,
   %%%%%%%%%%%%%%%%%%
   % Initialization %
   %%%%%%%%%%%%%%%%%%
case 0,
    sizes = simsizes;
    sizes.NumContStates  = 3;   %Number of Continuous States
    sizes.NumDiscStates  = 0;   %Number of Discret States
    sizes.NumOutputs     = 3;   %Number of Outputs
    sizes.NumInputs      = 0;   %Number of Inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=[2 1 2];                 %Initial Conditions
    str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Derivatives %
   %%%%%%%%%%%%%%%%
case 1,
   %Unified chaotic system implementation, the chosen constant
   %generates an Chen System as output.
   sys(1) = (25*a+10)*(x(2)-x(1));
   sys(2) = (28-35*a)*x(1)-x(1)*x(3)+(29*a-1)*x(2)+disturb(x,u,t);
   sys(3) = x(1)*x(2)-((a+8)/3)*x(3)+disturb(x,u,t);
   %%%%%%%%%%
   % Output %
```

```
   %%%%%%%%%%
case 3,
   sys = x;
   %%%%%%%%%%%%%
   %    End    %
   %%%%%%%%%%%%%
case {2,4,9},
    sys = [];

 otherwise
   error(['unhandled flag = ',num2str(flag)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Return the system's perturbations
%Arguments: state vector, input vector and time
function disturb = disturb(x,u,t)
if t>=5
    disturb=3*sin(t)*(sqrt(x(1)^2 + x(2)^2 + x(3)^2))+50*sin(200*t)+10*cos(400*t);
else
    disturb=0; %Before t=5s the disturbance is null
end
```

I.27 Appendix 27 - Code for identifier in literature [2] corresponding to Fig. 5.20-5.26

```
%Project Description: Neural based identification of nonlinear systems with
%                     one hidden layer and online learning laws
%Authors: F. Abdollahi, H. Ali Talebi and R. V. Patel
%Date: 08/2006          Local: IEEE Transactions on Mechatronics
function [sys,x0,str,ts] = Abdollahi(t,x,u,flag)

%Diagonal matrix A with negative elements 3x3
A=0.001*[-7.8 0 0; 0 -7.8 0; 0 0 -7.8];

n1=25;          %positive numbers
n2=0.4;         %positive numbers
rho1=0.02;      %small positive numbers
rho2=0.001;     %small positive numbers

%Parameters for sigmoidal function
alpha=85;
beta=1;

switch flag,
   %%%%%%%%%%%%%%%%%%
   % Initialization %
```

```
   %%%%%%%%%%%%%%%%%%
case 0,

    sizes = simsizes;
    sizes.NumContStates  = 45;   %Number of Continuous States
    sizes.NumDiscStates  = 0;    %Number of Discret States
    sizes.NumOutputs     = 39;   %Number of Outputs
    sizes.NumInputs      = 3;    %Number of Inputs
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0=zeros(45,1);              %Initial Conditions
    x0(1:3)=5;                   %Initial conditions for estimated states
        str=[];
    ts=[0 0];
   %%%%%%%%%%%%%%%%
   % Derivatives %
   %%%%%%%%%%%%%%%%

case 1,

      %Identification Model and Learning Laws implementation
      sys = [A*x(1:3) + [x(4:6)'; x(7:9)'; x(10:12)']*Sig(x, u, alpha, beta);
             -rho1*norm(x_error(x, u))*x(4:6)   + n1*w_term1(x, u, A, 1, alpha, ...
                 beta);
             -rho1*norm(x_error(x, u))*x(7:9)   + n1*w_term1(x, u, A, 2, alpha, ...
                 beta);
             -rho1*norm(x_error(x, u))*x(10:12) + n1*w_term1(x, u, A, 3, alpha, ...
                 beta);
             -rho2*norm(x_error(x, u))*x(13:23) + n2*v_term1(x, u, A, 1, alpha, ...
                 beta);
             -rho2*norm(x_error(x, u))*x(24:34) + n2*v_term1(x, u, A, 2, alpha, ...
                 beta);
             -rho2*norm(x_error(x, u))*x(35:45) + n2*v_term1(x, u, A, 3, alpha, ...
                 beta)];

   %%%%%%%%%%%%%%%%
   %   Outputs    %
   %%%%%%%%%%%%%%%%

  case 3,
       sys = [x(1:33);
              norm([x(4:6)'; x(7:9)'; x(10:12)'],'fro');
              norm([x(13:23)'; x(24:34)'; x(35:45)'], 'fro');
              norm([x(4:45)], 'fro');
              norm((u(1)-x(1)));
              norm((u(2)-x(2)));
              norm((u(3)-x(3)))];
case {2,4,9},
    sys = [];
otherwise
   error(['unhandled flag = ',num2str(flag)]);
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Returns the state estimation error vector
%Arguments: state vector, input vector

function x_error = x_error(x, u)

X = [x(1); x(2); x(3)];
U = [u(1); u(2); u(3)];

x_error = X - U;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Returns nonlinear regressor vector
%Arguments: state vector, input vector and parameters for sigmoidal function

function Sig = Sig(x,u, alpha, beta)     %Regressor

%Parameters for sigmoidal function
VZ = [x(13:23)'; x(24:34)'; x(35:45)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; ...
    u(1)*u(2); u(3)*u(2); u(1)*u(3); u(1)*u(2)*u(3); 1];

%Regressor Vector
Sig=[(z(VZ(1), alpha, beta));
     (z(VZ(2), alpha, beta));
     (z(VZ(3), alpha, beta))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Scalar sigmoidal function
%Arguments: VZ regressor term and parameters for sigmoidal function

function z = z(arg, alpha, beta)  %Activation Function
z = alpha/(1 + exp(-beta*arg)) - 1;   %Sigmoidal Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Returns a vector containing the derivatives of the nonlinear NN regressor
%Arguments: state vector, input vector, matrix B and K, line, parameters for ...
    sigmoidal function

function Sigdot = Sigdot(x,u, alpha, beta)     %Regressor

%Regressor for activation function
VZ = [x(13:23)'; x(24:34)'; x(35:45)']*[u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; ...
    u(1)*u(2); u(3)*u(2); u(1)*u(3); u(1)*u(2)*u(3); 1];

%Approximation using taylor series, this is the derivative of
%the regressor
Sigdot=[(z(VZ(1), alpha, beta))^2 0 0;
        0 (z(VZ(2), alpha, beta))^2 0;
        0 0 (z(VZ(3), alpha, beta))^2];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Returns the line from B*K*Xerro*S(VZ)
%Arguments: state vector, input vector, matrix B and K, line, parameters for ...
    sigmoidal function

function w_term1 = w_term1(x, u, A, line, alpha, beta)

temp = (((x_error(x,u)')*inv(A))')*((Sig(x, u, alpha, beta))');

if line == 1
    w_term1 = temp(1, :)';
end
if line == 2
    w_term1 = temp(2, :)';
end
if line == 3
    w_term1 = temp(3, :)';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Summary: Returns the line for Sdot(VZ)*W'*B*K*Xerro*Z'
%Arguments: state vector, input vector, matrix B and K, line, parameters for ...
    sigmoidal function

function v_term1 = v_term1(x, u, A, line, alpha, beta)

W_est = [x(4:6)'; x(7:9)'; x(10:12)'];

X_est = [u(1); u(2); u(3); u(1)^2; u(2)^2; u(3)^2; u(1)*u(2); u(3)*u(2); ...
    u(1)*u(3); u(1)*u(2)*u(3); 1];

temp = ((x_error(x,u)'*inv(A)*W_est*(eye(3)-Sigdot(x,u, alpha, ...
    beta)))')*sign(X_est');

if line == 1
    v_term1 = temp(1, :)';
end
if line == 2
    v_term1 = temp(2, :)';
end
if line == 3
    v_term1 = temp(3, :)';
end
```

I.28 Appendix 28 - Code for proposed identifier corresponding to Fig. 5.20-5.26

The code is the same as Appendix 17.

I.29 Appendix 29 - Code to display the Fig. 5.20-5.26

```
%Displays the actual states and their estimations
set(0,'DefaultAxesColorOrder',[0 0 1; 1 0 0; 0 0 0]);
fsize=14;

figure;
plot(t, Xestimado(:,37),t, Xestimado1(:,37), 'LineWidth', 2 );
axis([0 10 -0.5 inf]);
set(0,'DefaultAxesFontSize', 17);
h=legend('Proposed Algorithm','Algorithm in [1]');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('State Error Norm of x(t)','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;
plot(t, Xestimado(:,38),t, Xestimado1(:,38), 'LineWidth', 2 );
axis([0 10 -0.5 inf]);
set(0,'DefaultAxesFontSize', 17);
h=legend('Proposed Algorithm','Algorithm in [1]');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('State Error Norm of y(t)','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;
plot(t, Xestimado(:,39),t, Xestimado1(:,39), 'LineWidth', 2 );
axis([0 10 -0.5 inf]);
set(0,'DefaultAxesFontSize', 17);
h=legend('Proposed Algorithm','Algorithm in [1]');
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('State Error Norm of z(t)','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%
set(0,'DefaultAxesColorOrder',[0 0 0; 1 0 0; 0 0 0]);

figure;
plot(t, Xestimado1(:,34), 'LineWidth', 2 );
h=legend('Algorithm in [1]')
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated Weight Norm W','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
figure;
plot(t, Xestimado1(:,35), 'LineWidth', 2 );
h=legend('Algorithm in [1]')
set(0,'DefaultAxesFontSize', 17);
set(h,'FontSize',fsize);
xlabel('Time(s)','interpreter','latex','Fontsize',fsize);
ylabel('Estimated Weight Norm V','Fontsize',fsize);

%%%%%%%%%%%%%%%%%%%%%%%%%
```