Sahand University of Technology

Electrical Engineering Department

A Thesis Submitted in Partial Fulfillment of the Requirement for the Degree of Bachelor of Science in Biomedical Engineering

Title

Adaptive Control Using NARMA-L2 Model for Nonlinear Systems

By

Jamal Saeedi

Supervisor

Dr. Mahdi Zeinali

July 2007

To my father Saeed, my mother Shahrbanoo

and my sisters Zari and Roza

Abstract

This thesis considers the problem of using approximate methods for realizing the neural controllers for nonlinear multivariable systems. The nonlinear autoregressive-moving average (NARMA) model is an exact representation of the input–output behavior of finite-dimensional nonlinear discrete time dynamical systems in a neighborhood of the equilibrium state. However, it is not convenient for purposes of adaptive control using neural networks due to its nonlinear dependence on the control input. Hence, quite often, approximate methods are used for realizing the neural controllers to overcome computational complexity. In this thesis, we introduce two classes of models which are approximations to the NARMA model, and which are linear in the control input, namely NARMA-L1 and NARMA-L2. The latter fact substantially simplifies both the theoretical analysis as well as the practical implementation of the controller. Extensive simulation studies have shown that the neural controllers designed using the proposed approximate models perform very well, and in many cases even better than an approximate controller designed using the exact NARMA model. In view of their mathematical tractability as well as their success in simulation studies, a case is made in this paper that such approximate input–output models warrant a detailed study in their own right. The MATLAB simulation of the project is also provided from the following link: [**code**].

Key Words: Neural Networks, Identification, Adaptive Control, Input-output Models, Error Back propagation.

Table of Contents

Chapter 1. Introduction

This Chapter describes definitions, neural networks, adaptive control of non-linear dynamic systems, and the thesis' organization.

Models of neural networks have been studied for many years in hopes of creating a similar function to the human brain in topics such as speech recognition, identification, control, and so on [1]. These models consist of a large number of nonlinear computational members that work parallel to each other and are connected to each other through adaptive weights.

In fact, the following four characteristics of neural networks make them the best candidate for identifying and controlling the nonlinear dynamic systems [2-4]:

1. Parallel processing with a large amount of information: We know that the control systems have a large amount of information through the sensors at any moment. Therefore, to carry out an accurate control, the need for rapid processing of this information is vital; neural networks are capable of doing so.

2. Neural network has the capability of generalizing nonlinear functions to any desired degree of accuracy.

3. Learning Ability: As we know, there are uncertainties in the control of a series of systems and include a series of un-known parameters. Parameters must be identified to solve this problem, and neural networks with high learning ability can do this job.

4. High reliability: We know that in a neural network a complex problem is divided into small parts, and each nerve takes up one part of the problem, so the control system is not affected by the failure of one or more nerve cells. It is very useful property for a control system.

Given the mentioned advantages, neural networks would be the best candidate for identifying the nonlinear dynamic systems. The remarkable learning capability of neural networks is leading to their application in identification and adaptive control of dynamical systems. A neural network is basically composed of many neurons and interconnections with a particular architecture. Neural networks with relatively complex architectures tend to be more powerful in learning functional mapping but are more difficult to train [5-8].

The problem of controlling a plant can be conveniently divided into the regulation and tracking problems. In the former, the main objective is to stabilize the plant around a fixed operating point. In the later, the aim is to make the output of the plant follow a specified signal asymptotically. While our ultimate goal is to determine the control input, u, based only on output measurement for both regulation and tracking. We will confine our attention in this thesis to the problem of tracking when the multivariable system is unknown and only input and output values are available. One standard model that is used to represent general discrete-time nonlinear systems is the nonlinear autoregressive-moving average (NARMA) model. In [1], it is shown that NARMA-L1 and NARMA-L2 models were introduced as approximations of the NARMA model for the representation of SISO nonlinear dynamical systems. It was found that the availability of a NARMA model for a (single input-single output) SISO nonlinear plant does not automatically imply a method of determining the control input to track a desired output. If a neural network is used as a controller, the parameters of the latter have to be adjusted to achieve on-line control. This involves dynamic gradient methods that are computationally intensive. In contrast to that, since the control input u(k) occurs linearly in the NARMA-L1 and NARMA-L2 models, it can be computed directly from the identification model, which use static gradients. Even though the NARMA model results in better identification of the unknown plant,

the NARMA-L1 and NARMA-L2 models may actually result in better control. Specifically, in this thesis the identification and control of unknown non-linear dynamic systems using NARMA-L2 model is investigated.

The thesis is organized as follows: In Chapter 2, the neural networks is stated in detail. Chapter 3 gives the description of adaptive control using neural network. In Chapter 4, simulation results are presented. Finally, conclusion is given in Chapter 5.

Chapter 2. Neural Networks

Mathematical systems theory has made major advances in the past four decades and has evolved into a scientific discipline, which cuts across boundaries, extending from design and development on the one hand to mathematics on the other. The best developed part of the theory concerns linear systems and important concepts as well as major theoretical results have been introduced in such areas as stability theory, optimal control, multivariable theory and adaptive control. The powerful techniques developed in the area of adaptive control complement current computing technology and have enormous potential in the world of applications, where systems have to be controlled in the presence of uncertainty. Although adaptive systems are by their very nature nonlinear, most of the theories of such systems are deeply rooted in linear systems theory [9].

In recent years, the multilayer neural network and the recurrent network have emerged as important components, which have proved to be extremely successful in pattern recognition and optimization problems. From the system-theoretic point of view, these networks can be considered as components, which can be effectively used in complex nonlinear systems.

Two classes of networks which have received considerable attention in the area of artificial neural networks are (1) multilayer neural networks, and (2) recurrent networks. A typical multilayer neural network with an input layer, an output layer and two hidden layers is shown in Fig. 2.1. For convenience, this can be denoted in

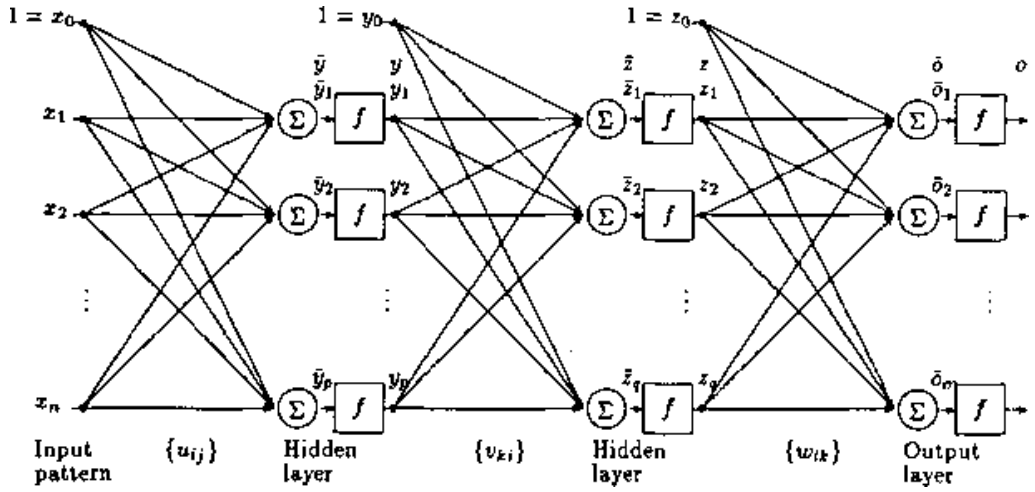


Figure 2.1. A three layer neural network.

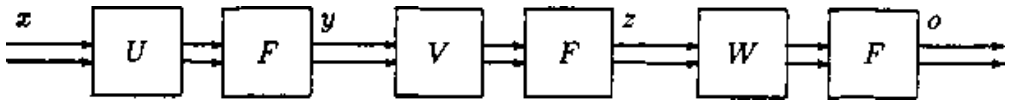


Figure 2.2. Block diagram of a three layer neural network.

block diagram form as shown in Fig 2.2 with three weight matrices U, V and W. The multilayer network represents a nonlinear map $\hat{f}$ where $\hat{f}(x) = F\left[WF\left[VF[Ux]\right]\right]$ and the elements of U, V and W are adjustable weights. Such networks have been used successfully in pattern recognition, where the weights are adjusted to minimize a suitable error function.

In contrast to the above, the recurrent network, based on the work of Hopfield [10], has been used as a content-addressable memory and in optimization problems. One version of the Hopfield network is shown in Fig. 2.3 and consists of a single layer neural network in the forward path connected to a delay in the feedback path. The choice of the weights determines the equilibrium states of the dynamical systems and hence the specific equilibrium to which the state trajectory converges depends upon the initial conditions. This fact has been used for content-addressable memories as well as optimization problems.

From a system-theoretic point of view, the multilayer network represents merely a versatile nonlinear map. The recurrent network on the other hand, in most cases,

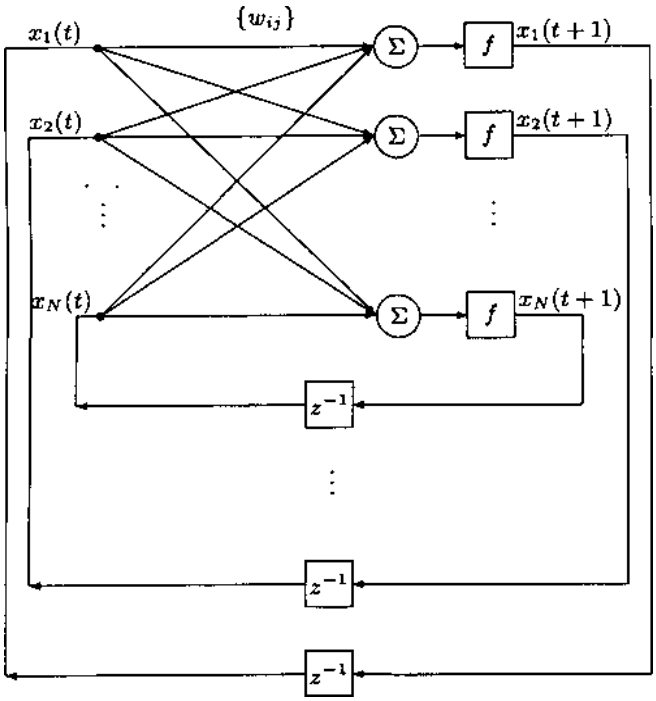


Figure 2.3. The Hopfield network.

represents a dynamical system with no input. In system analysis, both types of operators play an important role. Hence, it is desirable to study the properties of feedback systems, which contain both types of networks as components. In [7] the back-propagation method used for adjusting the weights in a multilayer neural network was also suggested for adjusting the weights in a recurrent network to increase the number of stored states.

2.1 Multilayer perceptron (MLP)

The multilayer perceptron neural network is built up of simple components. We will begin with a single-input neuron, which we will then extend to multiple inputs. We will next stack these neurons together to produce layers. Finally, we will cascade the layers together to form the network.

A single-input neuron is shown in Fig 2.4. The scalar input p is multiplied by the scalar weight w to form wp; one of the terms that is sent to the summer. The other input, 1, is multiplied by a bias b and then passed to the summer. The summer

output n; often referred to as the net input, goes into a transfer function f ; which produces the scalar neuron output a. The neuron output is calculated as:

$$a = f(wp + b) \tag{2.1}$$

Note that w and b are both adjustable scalar parameters of the neuron. Typically, the transfer function is chosen by the designer, and then the parameters w and b are adjusted by some learning rule so that the neuron input/output relationship meets some specific goal.

The transfer function in Fig. 2.4 (a) may be a linear or a nonlinear function of n: One of the most commonly used functions is the log-sigmoid transfer function, which is shown in Fig. 2.4 (b).

This transfer function takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0–1, according to the expression

$$a = \frac{1}{1+e^{-n}} \tag{2.2}$$

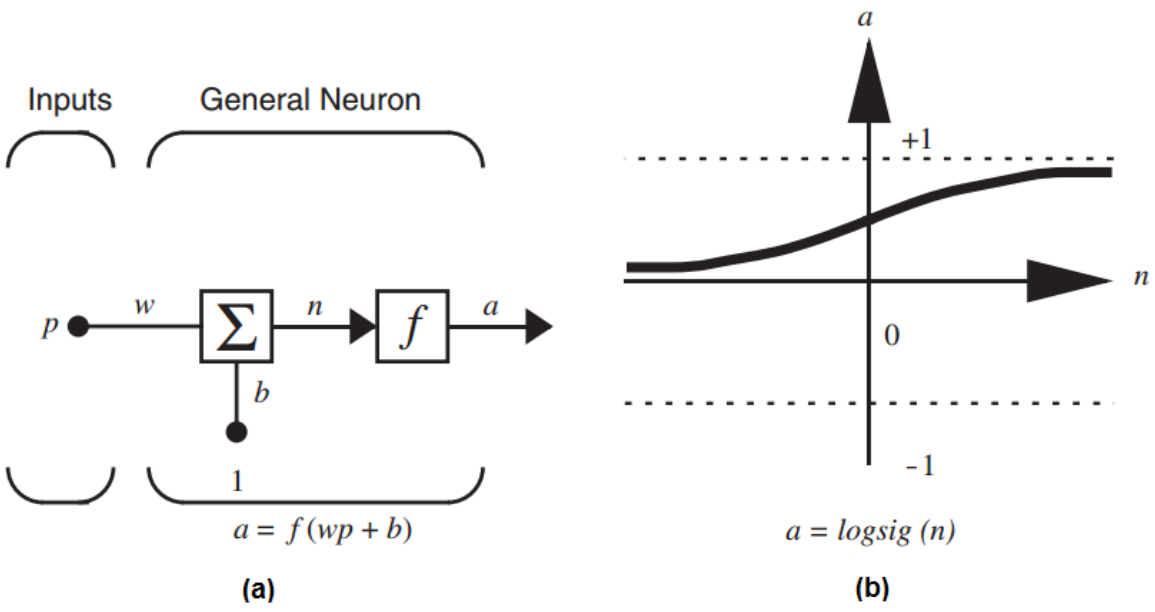


Fig. 2.4. (a) Single-input neuron, and (b) Log-Sigmoid Transfer Function.

The log-sigmoid transfer function is commonly used in multilayer networks that are trained using the back-propagation algorithm, in part because this function is differentiable.

Typically, a neuron has more than one input. A neuron with R inputs is shown in Fig. 2.5 (a). The individual inputs $p_1, p_2, \ldots, p_R$ are each weighted by corresponding elements $w_{1,1}, w_{1,2}, \ldots, w_{1,R}$ of the weight matrix **W**.

The neuron has a bias b; which is summed with the weighted inputs to form the net input n:

$$n = w_{1,1}p_1 + w_{1,2}p_2 \ldots + w_{1,R}p_R \tag{2.3}$$

This expression can be written in matrix form

$$n = \mathbf{Wp} + b \tag{2.4}$$

where the matrix **W** for the single neuron case has only one row.

Now the neuron output can be written as

$$a = f(\mathbf{Wp} + b) \tag{2.5}$$

Fig. 2.5 (b) represents the neuron in matrix form.

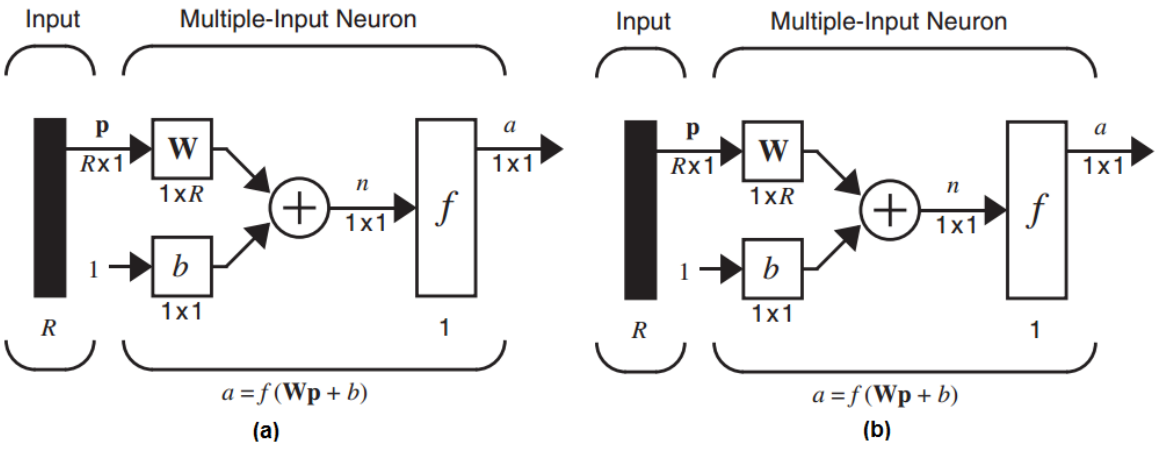


Figure 2.5. (a) Multiple-input neuron, (b) Neuron with R inputs, matrix notation.

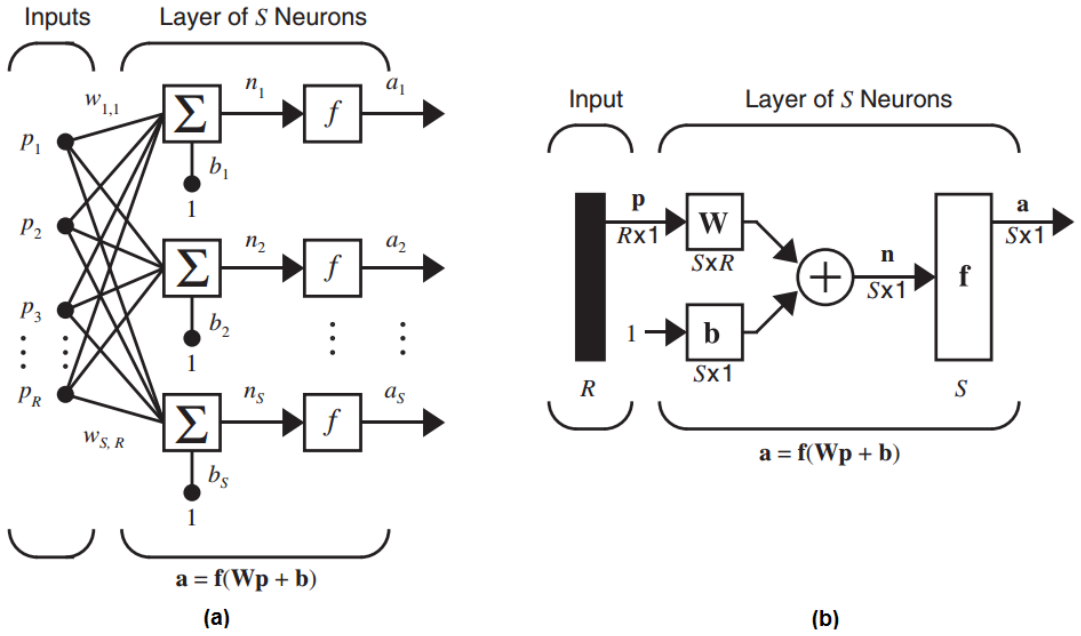


Figure 2.6. (a) Layer of S neurons, and (b) matrix notation.

Commonly one neuron, even with many inputs, is not sufficient. We might need 5 or 10, operating in parallel, in what is called a layer. A single-layer network of S neurons is shown in Fig. 2.6 (a). Note that each of the R inputs is connected to each of the neurons and that the weight matrix now has S rows. The layer includes the weight matrix W; the summers, the bias vector b; the transfer function boxes and the output vector a: Some authors refer to the inputs as another layer, but we will not do that here. It is common for the number of inputs to a layer to be different from the number of neurons (i.e. R = S). The S-neuron, R-input, one-layer network also can be drawn in matrix notation, as shown in Fig. 2. 6 (b).

Now consider a network with several layers. Each layer has its own weight matrix **W**; its own bias vector **b**; a net input vector **n** and an output vector **a**: We need to introduce some additional notation to distinguish between these layers. We will use superscripts to identify the layers. Thus, the weight matrix for the first layer is written as $\mathbf{W^1}$; and the weight matrix for the second layer is written as $\mathbf{W^2}$: This notation is used in the three-layer network shown in Fig. 2.7. As shown, there are R inputs, S^1 neurons in the first layer, S^2 neurons in the second layer, etc. As noted, different layers can have different numbers of neurons. The outputs of layers

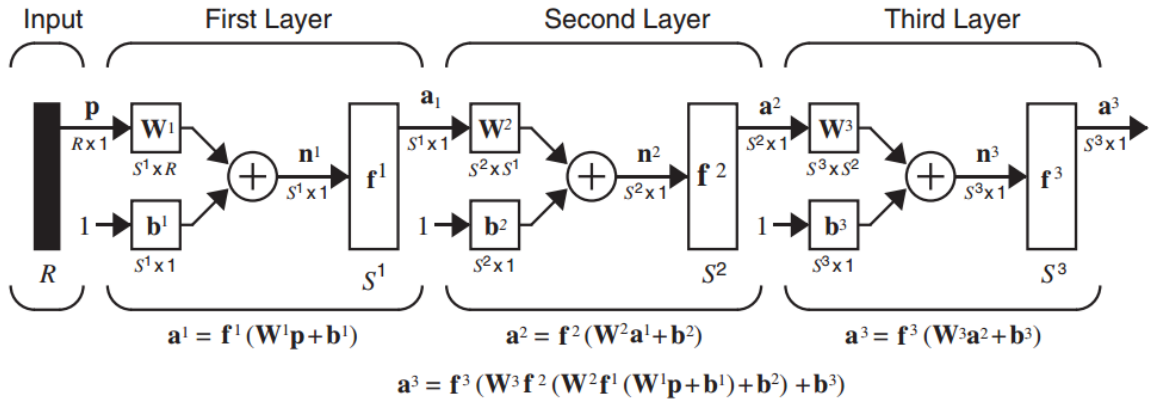


Figure 2.7. Three-layer network.

one and two are the inputs for layers two and three. Thus, layer 2 can be viewed as a one-layer network with $R = S^1$ inputs, $S = S^2$ neurons, and an $S^2 \times S^1$ weight matrix $\mathbf{W^2}$: The input to layer 2 is $\mathbf{a^1}$; and the output is $\mathbf{a^2}$: A layer whose output is the network output is called an output layer. The other layers are called hidden layers. The network shown in Fig. 2.7 has an output layer (layer 3) and two hidden layers (layers 1 and 2).

2.2 Error back propagation (EBP) training

Now that we know multilayer networks are universal approximators, the next step is to determine a procedure for selecting the network parameters (weights and biases) that will best approximate a given function. The procedure for selecting the parameters for a given problem is called training the network. In this Chapter, we will outline a training procedure called back-propagation [11-12], which is based on gradient descent. More efficient algorithms than gradient descent are often used in neural network training [13].

As we discussed earlier, for multilayer networks the output of one layer becomes the input to the following layer (see Fig. 2.7). The equations that describe this operation are:

$$a^{m+1} = f^{m+1}(\mathbf{W^{m+1}a^m} + \mathbf{b}^{m+1}) \; for \; m = 0,1,\dots,M-1 \qquad (2.6)$$

where $\boldsymbol{M}$ is the number of layers in the network. The neurons in the first layer receive external inputs:

$$a^0 = \mathbf{p} \tag{2.7}$$

which provides the starting point for Equation (2.6). The outputs of the neurons in the last layer are considered the network outputs:

$$\mathbf{a} = \mathbf{a^M} \tag{2.8}$$

The back-propagation algorithm for multilayer networks is a gradient descent optimization procedure in which we minimize a mean square error performance index. The algorithm is provided with a set of examples of proper network behavior:

$$\{\mathbf{p_1}, \mathbf{t_1}\}, \{\mathbf{p_2}, \mathbf{t_2}\}, \ldots, \{\mathbf{p_Q}, \mathbf{t_Q}\} \tag{2.9}$$

where p_q is an input to the network and t_q is the corresponding target output. As each input is applied to the network, the network output is compared to the target. The algorithm should adjust the network parameters in order to minimize the sum-squared error:

$$\boldsymbol{F(x)} = \sum_{\boldsymbol{q=1}}^{\boldsymbol{Q}} \boldsymbol{e_q^2} = \sum_{\boldsymbol{q=1}}^{\boldsymbol{Q}} \boldsymbol{(t_q - a_q)^2} \tag{2.10}$$

where **x** is a vector containing all network weights and biases. If the network has multiple outputs this generalizes to

$$\boldsymbol{F(x)} = \sum_{\boldsymbol{q=1}}^{\boldsymbol{Q}} \boldsymbol{e_q^T e_q} = \sum_{\boldsymbol{q=1}}^{\boldsymbol{Q}} \mathbf{(t_q - a_q)^T (t_q - a_q)} \tag{2.11}$$

Using a stochastic approximation, we will replace the sum-squared error by the error on the latest target:

$$\boldsymbol{\widehat{F}(x)} = \mathbf{(t(k) - a(k))^T (t(k) - a(k))} = \boldsymbol{e^T(k) e(k)} \tag{2.12}$$

where the expectation of the squared error has been replaced by the squared error at iteration k. The steepest descent algorithm for the approximate mean square error is:

$$\boldsymbol{w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha \frac{\partial \widehat{F}}{\partial w_{i,j}^{m}}} \tag{2.13}$$

$$\boldsymbol{b_{i}^{m}(k+1) = b_{i}^{m}(k) - \alpha \frac{\partial \widehat{F}}{\partial b_{i}^{m}}} \tag{2.14}$$

where a is the learning rate.

For a single-layer linear network, these partial derivatives in Equations (2.13) and (2.14) are conveniently computed, since the error can be written as an explicit linear function of the network weights. For the multilayer network, the error is not an explicit function of the weights in the hidden layers; therefore, these derivatives are not computed so easily. Because the error is an indirect function of the weights in the hidden layers, we will use the chain rule of calculus to calculate the derivatives in Equations (2.13) and (2.14):

$$\boldsymbol{\frac{\partial \widehat{F}}{\partial w_{i,j}^{m}} = \frac{\partial \widehat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial w_{i,j}^{m}}} \tag{2.15}$$

$$\boldsymbol{\frac{\partial \widehat{F}}{\partial b_{i}^{m}} = \frac{\partial \widehat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial b_{i}^{m}}} \tag{2.16}$$

The second term in each of these equations can be easily computed, since the net input to layer m is an explicit function of the weights and bias in that layer: The second term in each of these equations can be easily computed, since the net input to layer m is an explicit function of the weights and bias in that layer:

$$\boldsymbol{n_{i}^{m} = \sum_{j=1}^{s^{m-1}} w_{i,j}^{m} a_{j}^{m-1} + b_{i}^{m}} \tag{2.17}$$

Therefore,

$$\boldsymbol{\frac{\partial n_{i}^{m}}{\partial w_{i,j}^{m}} = a_{j}^{m-1}, \frac{\partial n_{i}^{m}}{\partial b_{i}^{m}} = 1} \tag{2.18}$$

If we now define:

$$\boldsymbol{s_{i}^{m} = \frac{\partial \widehat{F}}{\partial n_{i}^{m}}} \tag{2.19}$$

(the sensitivity of $\hat{F}$ to changes in the ith element of the net input at layer m), then Equations (2.15) and (2.16) can be simplified to:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^{m}} = s_i^m a_j^{m-1} \tag{2.20}$$

We can now express the approximate steepest descent algorithm as:

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha s_i^m a_j^{m-1} \tag{2.21}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha s_i^m \tag{2.22}$$

In matrix form, this becomes:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T \tag{2.23}$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m \tag{2.24}$$

where the individual elements of $\mathbf{s}^m$ are given by Equation (2.19).

In some ways it is unfortunate that the algorithm we usually refer to as back-propagation, given by Equations (2.23) and (2.24), is in fact simply a steepest descent algorithm. There are many other optimization algorithms that can use the back-propagation procedure, in which derivatives are processed from the last layer of the network to the first. For example, conjugate gradient and quasi-Newton algorithms [14-16] are generally more efficient than steepest descent algorithms, and yet they can use the same back-propagation procedure to compute the necessary derivatives. The Levenberg–Marquardt algorithm is very efficient for training small to medium-size networks [17].

2.3 Nonlinear systems identification using neural networks

(Models for the Identification of Nonlinear Dynamical Systems)

In the adaptive control of unknown and non-linear systems, the system must firstly be identified, that is, considering a model for the system and estimate its parameters in which it relates the system's input and output with minimum error. With regard to the capabilities of neural networks in general approximation, our goal here is to use them in identifying nonlinear systems.

In this sub-section, four models, which were introduced in [4] for the representation of a single-input single output (SISO) nonlinear plant, are presented. These models were chosen both for their generality as well as for analytical tractability. The models are motivated by corresponding models, which have been used in the adaptive systems literature for the identification of linear systems and can be considered as their generalizations to nonlinear systems. Since back-propagation is the principal method that we shall use for the adjustment of parameters of the identification model, the parameterization of the plant (and hence the model) is such as to make the application of the procedure relatively straightforward.

The models of the four classes of plants introduced here can be described by the following nonlinear difference equations:

$$MODEL\ I: y_p(k+1) = \sum_{i=0}^{n-1} a_i y_p(k-i) + g[u(k), u(k-1), \dots, u(k-m+1)] \quad (2.25)$$

$$MODEL\ II: y_p(k+1) = f\left[y_p(k), y_p(k-1), \dots, y_p(k-n+1)\right] + \sum_{i=0}^{m-1} \beta_i u(k-i) \quad (2.26)$$

$$MODEL\ III: y_p(k+1) = f\left[y_p(k), y_p(k-1), \dots, y_p(k-n+1)\right] + g[u(k), u(k-1), \dots, u(k-m+1)] \quad (2.27)$$

$$MODEL\ IV: y_p(k+1) = f\left[y_p(k), y_p(k-1), \dots, y_p(k-n+1), u(k), u(k-1), \dots, u(k-m+1)\right] \quad (2.28)$$

where $[u(k), x(k)]$represents the input-output pair of the SISO plant at time k. The functions f and g are assumed to be differentiable functions of their arguments. It is evident that Model IV subsumes Models I-III. However, Model IV is analytically

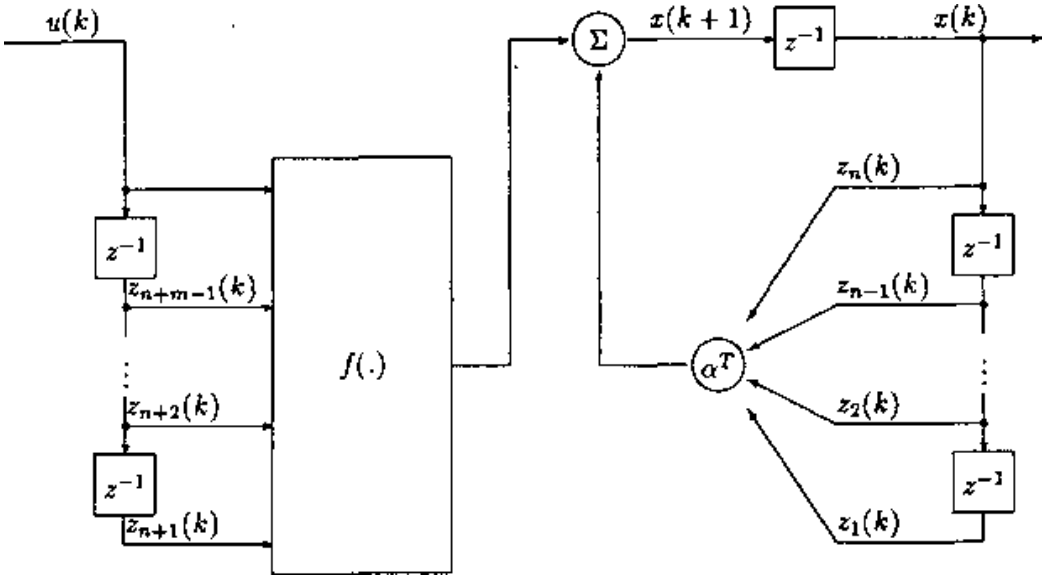


Figure 2.8. Model I: Structure of the plant.

the least tractable and hence for practical applications, some of the other models might prove more attractive. In this Chapter, each of these models is briefly described.

Model I: The output of the unknown nonlinear plant in this case is assumed to depend linearly on its past values and nonlinearly on the past values of the input. The latter is realized as shown in Fig. 2.8 and consists of tapped delay lines at the input and the feedback path.

Model II: This model is realized as shown in Fig. 2.9. In this case, the output depends linearly on the input $u(k)$ and its past values but nonlinearly on its own past values. The advantage of this model is that it lends itself readily to control in practical situation.

Model III: The unknown plant in this case is described by a nonlinear difference equation of the form:

$$x(k+1) = f[x(k), x(k-1), \dots, x(k-n+1)] + g[u(k), u(k-1), \dots, u(k-m+1)] \qquad (2.29)$$

and hence depends nonlinearly on both its past values as well as those of the input. However, the effects of the input and output values are additive as shown in equation (2.29). The representation of equation (2.29) is shown in Fig. 2.10.

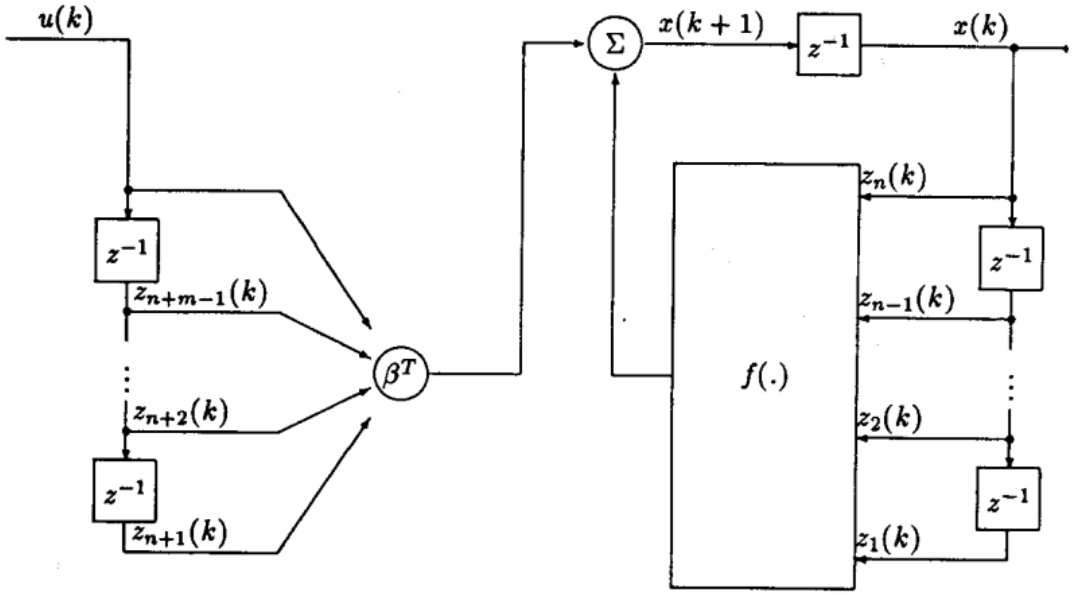


Figure 2.9. Model II: Structure of the plant.

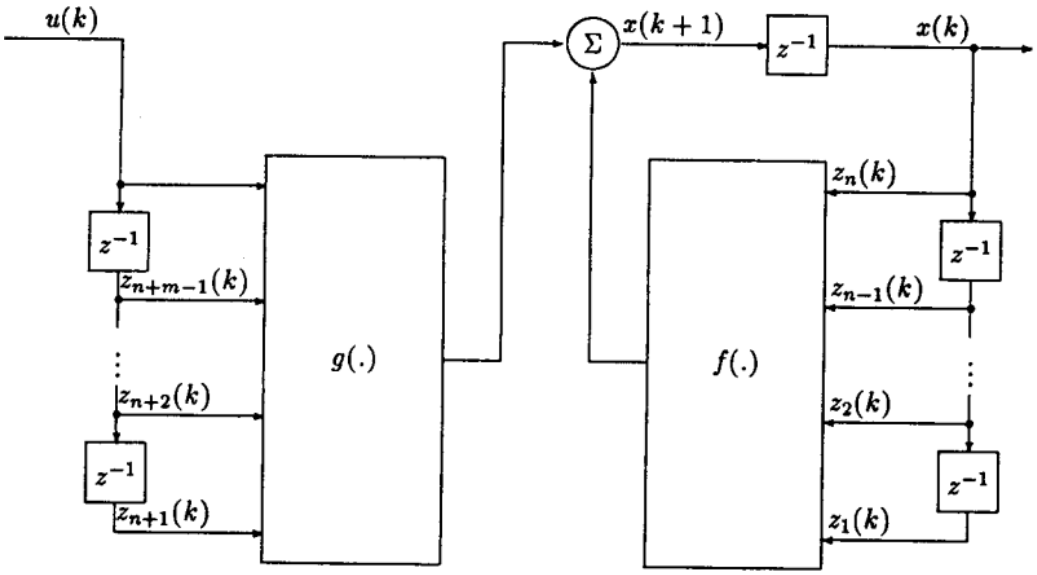


Figure 2.10. Model III: Structure of the plant.

Model IV: As mentioned earlier, this is the most general of all models introduced here and subsumes the earlier models. The output at any instant in this case is a nonlinear function of the past values of both the input and the output. Once again, the representation of the model using tapped delay lines is shown in Fig. 2.11.

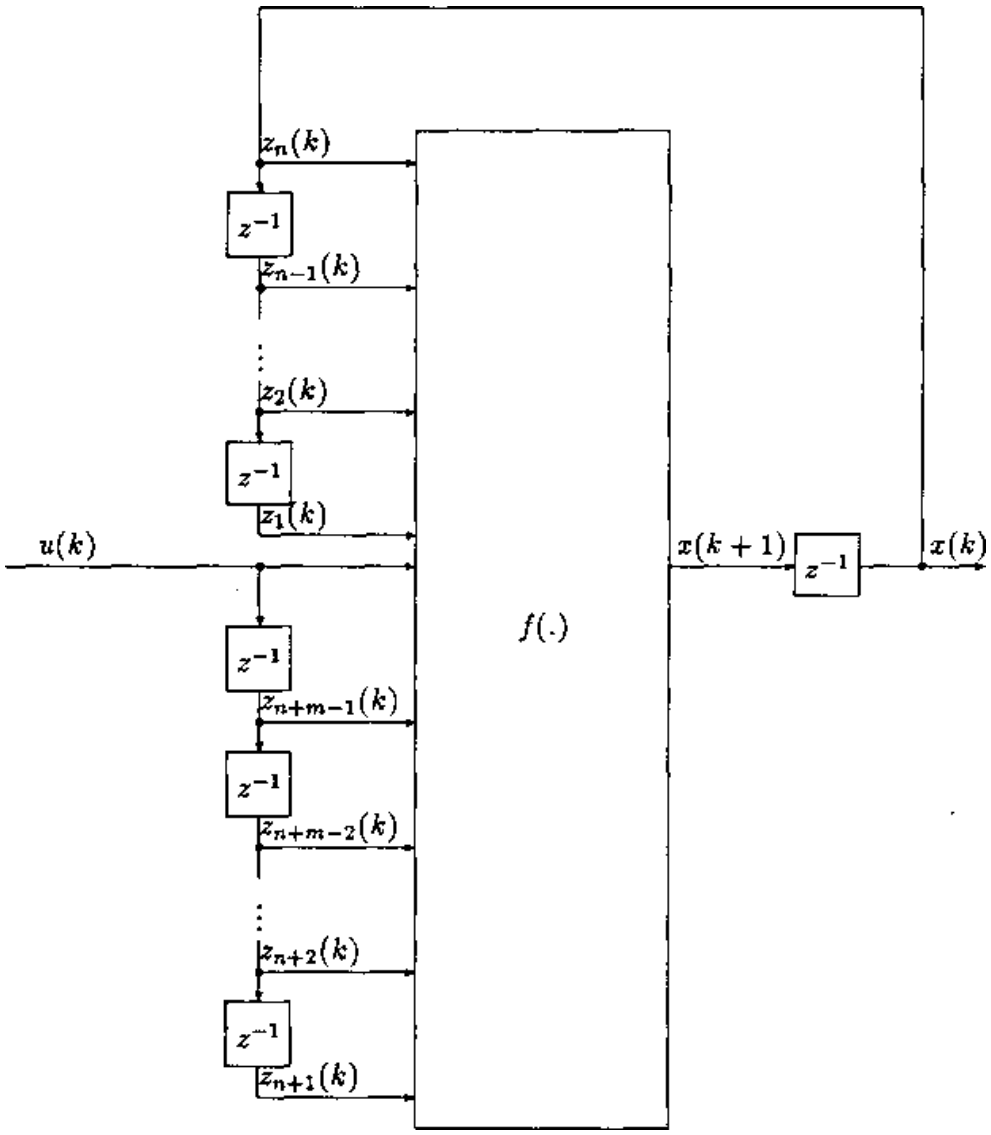


Figure 2.11. Model IV: Structure of the plant.

The identification model of the plant is composed of neural networks and tapped delay lines. In each case, the neural network is assumed to contain sufficient number of layers, and nodes in each layer, so as to be able to match exactly the input-output characteristics of the corresponding nonlinear mapping in the given plant. Prom the point of view of mathematical analysis, this implies that the nonlinear functions in the difference equations describing the plant can be replaced by neural networks with fixed but unknown weight matrices W_i^*. Hence, a theoretical solution to the adaptive identification problem is assumed to exist from the outset.

To identify the plant, an identification model is chosen based on prior information concerning the class to which it belongs. For example, assuming that the plant has a structure described by Model III, the model is chosen to have the form shown in Fig. 2.12. The aim then is to determine the weights of the two neural networks N_1 and N_2 so that the mapping N_1 is equal to $g[.]$ and the mapping N_2 is equal to $f[.]$. If $x(k+1)$ and $\hat{x}(k+1)$ are respectively the outputs at stage $k+1$ of the plant and the identification model, the error $e(k+1) = \hat{x}(k+1) - x(k+1)$ is used to update the weights of N_1 and N_2; static or dynamic back-propagation can be used, depending on the structure of the identifier used.

a. Parallel Model: In this case, the structure of the identifier is identical to that of the plant with f and g replaced by N_2 and N_1 respectively. This is shown in Fig. 2.12. Since N_2 is in a dynamic feedback loop, the parameters of N_1 and N_2 have to be adjusted using dynamic back-propagation.

b. Series-Parallel Model: In this case $x(k+1)$ rather than $\hat{x}(k+1)$ is used to generate the output of the model. This implies that the model is described by the equation:

$$\hat{x}(k+1) = N_2[x(k), x(k-1), \dots, x(k-n+1)] + N_1[u(k), u(k-1), \dots, u(k-m+1)] \quad (2.30)$$

Since the model does not include a feedback loop containing a nonlinear element, static back-propagation rather than dynamic back-propagation of the error can be used to adjust the weights of the neural network

The two methods outlined above have been discussed extensively in the context of the identification of linear time-invariant systems with unknown parameters [9]. While the series- parallel method has been shown to be globally stable, similar results are not available for the parallel model. To avoid many of the analytical difficulties encountered, as well as to assure stability and simplify the identification procedure, the series-parallel model was used in [4]. Extensive computer simulations have revealed that a large class of nonlinear plants can be identified using the above procedure. However, theoretical studies concerning stability and

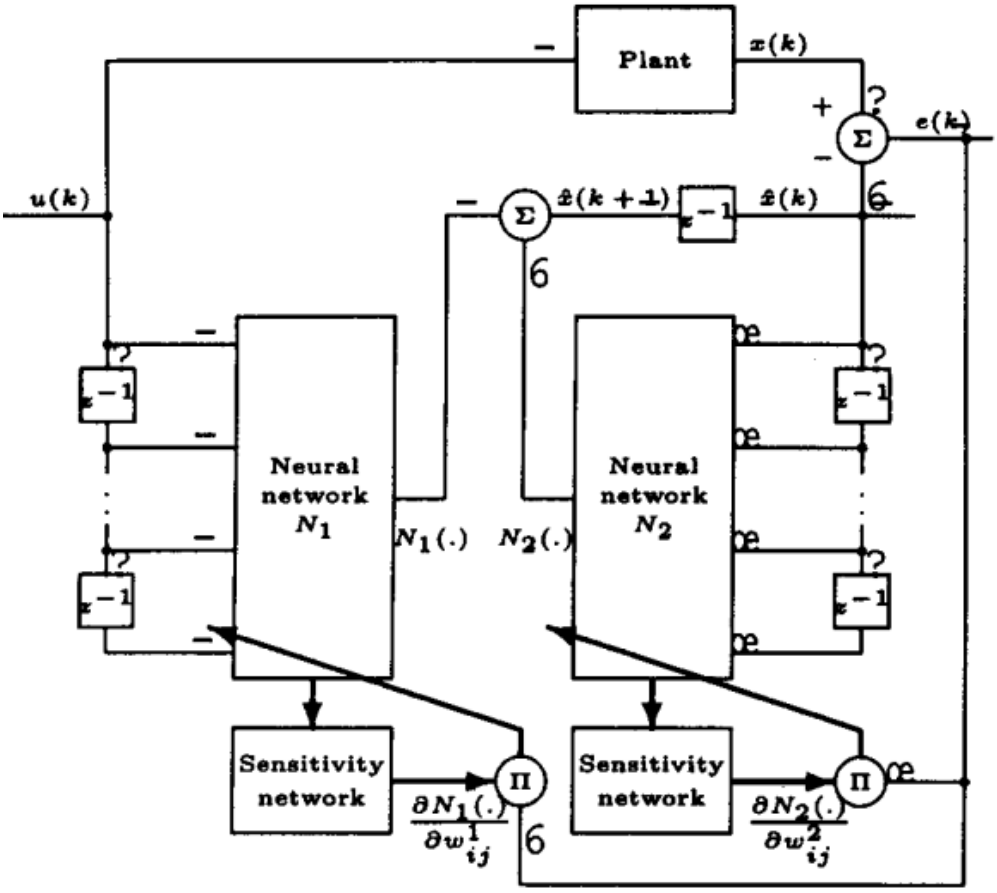


Figure 2.12. Model III: Structure of identification model.

convergence are still in the initial stages and numerous questions have yet to be answered.

Chapter 3. Adaptive Control Using Neural Networks

Adaptive control is the control method used by a controller, which must adapt to a controlled system with parameters, which vary, or are initially uncertain. For example, as an aircraft flies, its mass will slowly decrease because of fuel consumption; a control law is needed that adapts itself to such changing conditions. Adaptive control is different from robust control in that it does not need a priori information about the bounds on these uncertain or time-varying parameters; robust control guarantees that if the changes are within given bounds the control law need not be changed, while adaptive control is concerned with control law changing itself. Block diagram of adaptive control is shown in Fig. 3.1.

The foundation of adaptive control is parameter estimation, which is branch of system identification. Common methods of estimation include recursive least squares and gradient descent. Both of these methods provide update laws which are used to modify estimates in real time (i.e., as the system operates). Lyapunov stability is used to derive these update laws and show convergence criterion (typically persistent excitation, relaxation to this condition are studied in Concurrent Learning adaptive control). Projection (mathematics) and normalization are commonly used to improve the robustness of estimation algorithms.

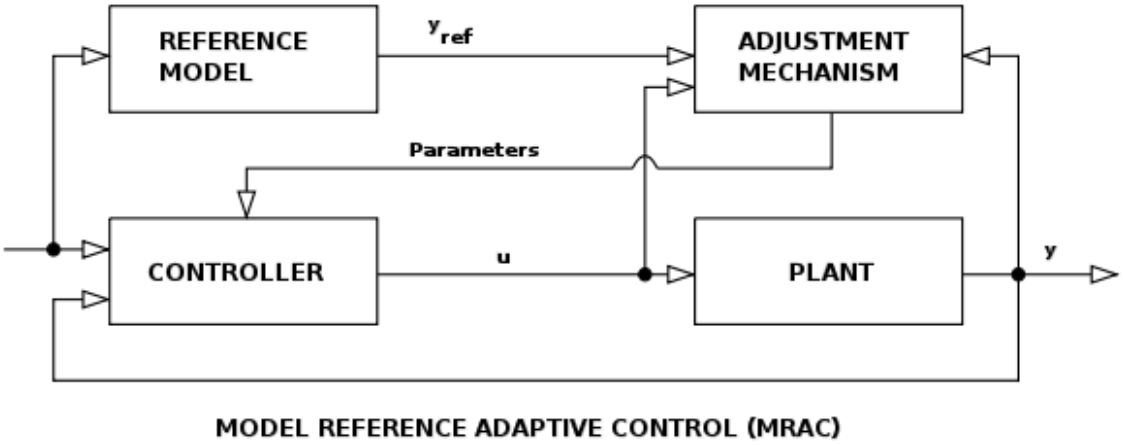


Figure 3.1. Block diagram of adaptive control.

The first notable and widespread use of ‘adaptive control’ was in the aerospace industry during the 1950s in an attempt to further the design of autopilots [18]. After the successful implementation of jet engines into aircraft, flight envelopes increased by large amounts and resulted in a wide range of operating conditions for a single aircraft. Flight envelopes grew even more with developing interest in hypersonic vehicles from the community. The existing autopilots at the time left much to be desired in the performance across the flight envelope, and engineers began experimenting with methods that would eventually lead to Model Reference Adaptive Control (MRAC). One of the earliest MRAC designs, developed by Whitaker [19-20], was used for flight control. During this time however, the notion of stability in the feedback loop and in adaptation was not well understood or as mature as today. Parks was one of the first to implement Lyapunov based adaptation into MRAC [21]. An immature theory coupled with bad and/or incomplete hardware configurations led to significant doubts and concerns in the adaptive control community, especially after the crash of the X-15. This caused a major, albeit necessary, detour from the problem of adaptation to focus on stability.

The idea of Neural Networks as a mathematical logic system was developed during the 1940s by McCulloch and Pitts [22]. The first presentation of a learning rule for synaptic modification came from Hebb in 1949 [23]. While many papers and books were published on subjects related to neural networks over the next two

decades, perhaps the most important accomplishment was the introduction of the Perceptron and its convergence theorem by Rosenblatt in 1958 [24]. Widrow and Hoff then proposed the trainable Multi-Layered Perceptron in 1962 using the Least Mean Square Algorithm [25], but Minsky and Papert then showed the fundamental limitations of single Perceptron, and also proposed the 'credit assignment problem' for Multi-Layer Perceptron structures [26]. After a period of diminished funding and interest, these problems were finally solved in the early 1980s. Shortly after this, Hopfiel [27] showed that information could be stored in these networks which led to a revival in the field. He was also able to prove stability, but convergence only to a local minimum not necessarily to the expected/desired minimum. This period also saw the re-introduction of the back-propagation algorithm [28], which has become extremely relevant to neural networks in control. Radial Basis Functions (RBFs) were created in the late 80s by Broomhead and Lowe [29] and were shortly followed by Support Vector Machines (SVMs) in the early 90s [30].

Parametric adaptive control is the problem of controlling the output of a system with a known structure but unknown parameters. To make the problem analytically tractable, in adaptive systems theory the plant to be controlled is assumed to be linear time-invariant with unknown parameters. These parameters can be considered as the elements of a vector p. If p is known, the parameter vector θ of a controller can be chosen as θ^* so that the plant together with the fixed controller behaves like a reference model described by a linear difference (or differential) equation with constant coefficients. If p is unknown, the vector $\theta(t)$ has to be adjusted on-line using all the available information concerning the system.

Two distinct approaches to the adaptive control of an unknown plant axe (i) direct control and (ii) indirect control. In direct control, the parameters of the controller axe directly adjusted to reduce some norm of the output error. In indirect control, the parameters of the plant are estimated as $p(t)$ at any time instant and the parameter vector $\theta(t)$ of the controller is chosen assuming that $\hat{p}(t)$ represents the true value of the plant parameter vector. Even when the plant is assumed to be

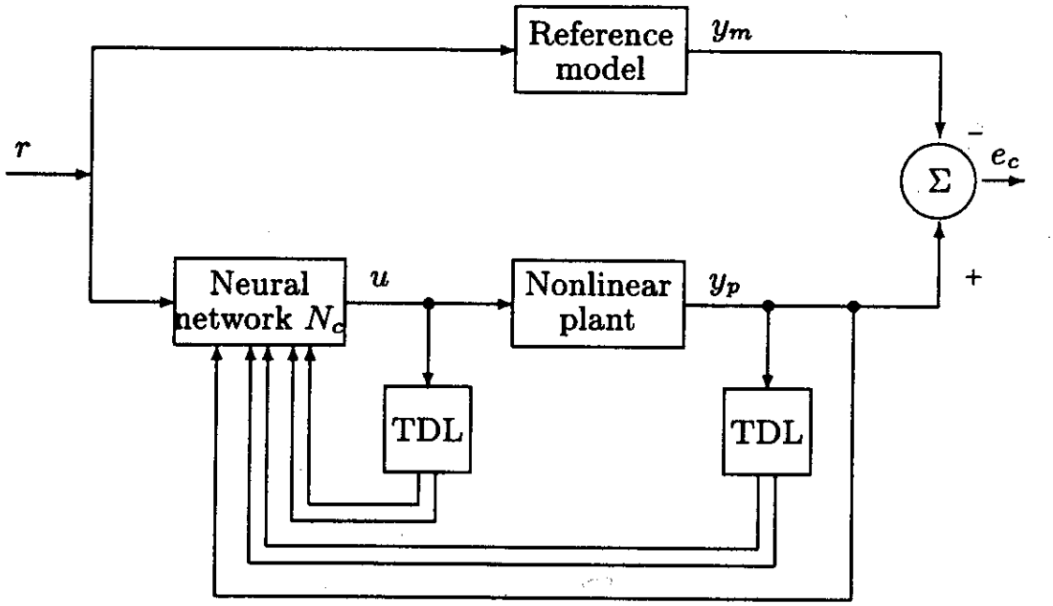


Figure 3.2. Direct adaptive control using neural networks.

linear and time-invariant, both direct and indirect adaptive control result in nonlinear systems. When the plant is nonlinear and dynamic (i.e. the present value of its output depends upon the past values of the input and the output respectively), a neural network can be used as a controller as shown in Fig. 3.2. This corresponds to direct control.

3.1 Direct adaptive control

In conventional direct adaptive control theory, methods for adjusting the parameters of a controller based on the measured output error rely on concepts such as positive realness and/or passivity. By making suitable assumptions concerning the plant and the reference model, it is shown that the direction in which a parameter is to be adjusted can be obtained by correlating two signals that can be measured. Using either Liapunov theory or hyper-stability theory it is shown that the adjustments of all the controller parameters based on such adaptive laws result in the stability of the overall system.

At present, methods for directly adjusting the parameters of the controller (the neural network N_c in Fig. 3.2) in a stable fashion based on the output error axe not available. This is due to the nonlinear nature of both the plant and the controller. Even back-propagation cannot be used directly, since the plant is unknown and hence cannot be used to generate the desired partial derivatives. Hence, until direct control methods are developed, adaptive control of nonlinear dynamical systems has to be carried out using indirect control methods.

3.2 Indirect adaptive control

As mentioned earlier, when indirect control is used to control a nonlinear system, the plant is parameterized using one of the models described in the previous Chapter and the parameters of the model are updated using the identification error. The controller parameters in turn are adjusted by back-propagating the error (between the identified model and the reference model outputs) through the identified model. A block diagram of such an adaptive system is shown in Fig. 3.3.

Both identification and control can be carried out at every instant or after processing the data over finite intervals. When external disturbances and/or noise are not present in the system, it is reasonable to adjust the control and identification parameters synchronously. However, when sensor noise or external disturbances are present, identification is carried out at every instant while control parameter updating is carried out over a slower time scale, to assure robustness (i.e., control parameters are adjusted with a lower frequency than identification parameters).

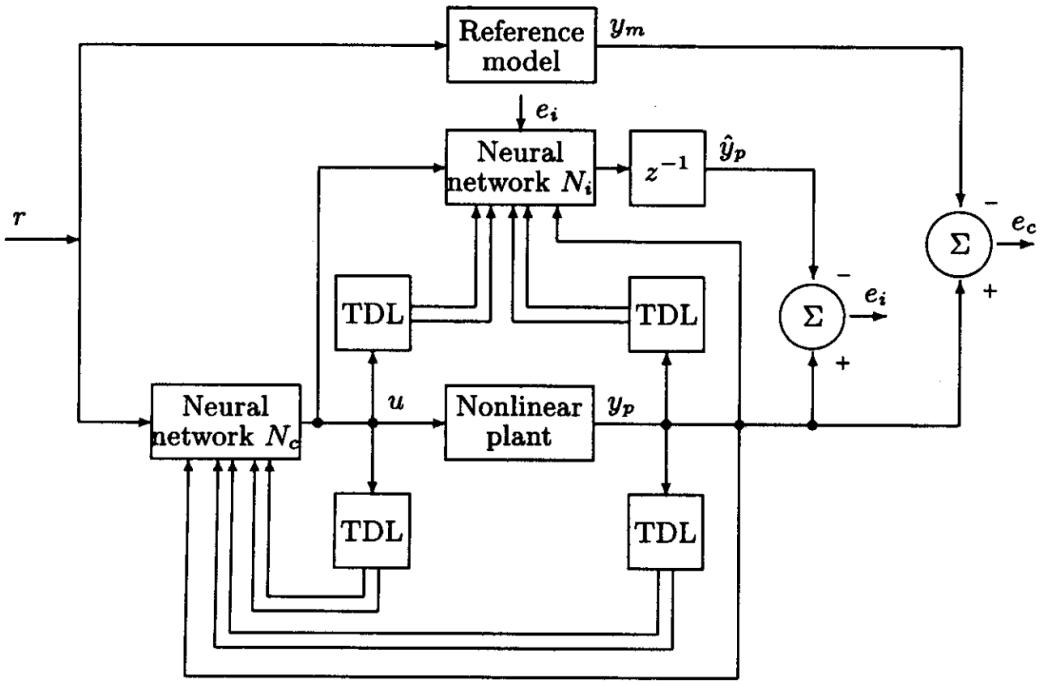


Figure 3.3. Indirect adaptive control using neural networks.

3.3 NARMA model and its approximations

The controller structure in general is dependent on the system's identification model, so the system's identification model can be considered so that a simple structure controller can be obtained.

As shown in Chapter 2 the NARMA model is an exact representation of the nonlinear plant in a neighborhood of the equilibrium state. For reasons given toward the end of Chapter 2, the model is not convenient for the computation of a control input to the plant to track a desired reference signal. In view of this, we propose two approximations to the NARMA model called the NARMA-L1 and the NARMA-L2 models. The main feature of these models is that the control input at time (the instant of interest in the control problem) occurs linearly in the equation relating inputs and outputs. This, in turn, permits easy algebraic computation of the control inputs without requiring a separate controller neural network. The fact that the use of neural networks is restricted to the identification model implies that only

static gradient methods need to be used. The equations for the two proposed approximate models are given below.

NARAM model:

$$y(k+d) = \bar{F}[y(k), y(k-1), \dots, y(k-n+1), u(k), u(k-1), \dots, u(k-n+1)] \quad (3.1)$$

Now, with the Taylor series expansion of $\bar{F}$ around the vector $(y(k), \dots, y(k-n+1), u(k) = 0, \dots, u(k-n+1) = 0)$ and $u(k)$, and then with the removal of high order sentences of the Taylor series expansion, the following models can be approximated, respectively:

NARAM_L1 model:

$$y(k+d) = f_0[y(k), y(k-1), \dots, y(k-n+1)] + \sum_{i=0}^{n-1} g_i[y(k), y(k-1), \dots, y(k-n+1)]\, u(k-i) \quad (3.2)$$

NARAM_L2 model:

$$y(k+d) = \overline{f_0}[y(k), y(k-1), \dots, y(k-n+1)] + \overline{g_0}[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-n+1)]u(k) \quad (3.3)$$

It is seen that f_0 and g_0 in the equation describing NARMA-L1 are only functions of the past values of the outputs, and $u(k-1)$ ... $u(k-n+1)$as well as $u(k)$ occur linearly on the right-hand side (RHS) of (3.2). In contrast to this, NARMA-L2 model is described by only two terms in the RHS of (3.3) where both $\overline{f_0}$ and $\overline{g_0}$ are functions of $y(k), y(k-1), \dots, y(k-n+1)$ and $u(k), u(k-1), \dots, u(k-n+1)$.

In the rest of this Chapter as well as in the following Chapter we justify the use of NARAM_L2 model in identification and control problems.

3.4 Identification using NARAM-L2 model

As it can be seen for the equations (3.2) and (3.3), if the NARMA-L1 model is used to identify a system, $n + 1$ neural networks are required for approximation of the following functions: $f_0, g_0, \dots, g_{n-1}$. However, to identify the system using the approximate model NARMA-L2, two neural networks is only needed for the approximation of the following functions: $\overline{f_0}, \overline{g_0}$. Therefore, from the practical point of view, the NARMA-L2 model is easier to apply than the NARMA-L1 model. Fig. 3.4 is shown block diagram of NARMA-L2 model in which $y^*(k)$ is an approximate of system's output and $e(k)$ is used to train the neural networks.

3.5 Adaptive control using NARAM-L2 model

Here, the objective is controlling an unknown nonlinear system which is based on its input and output data, so that the system follows desired signal $yr(k)$. The

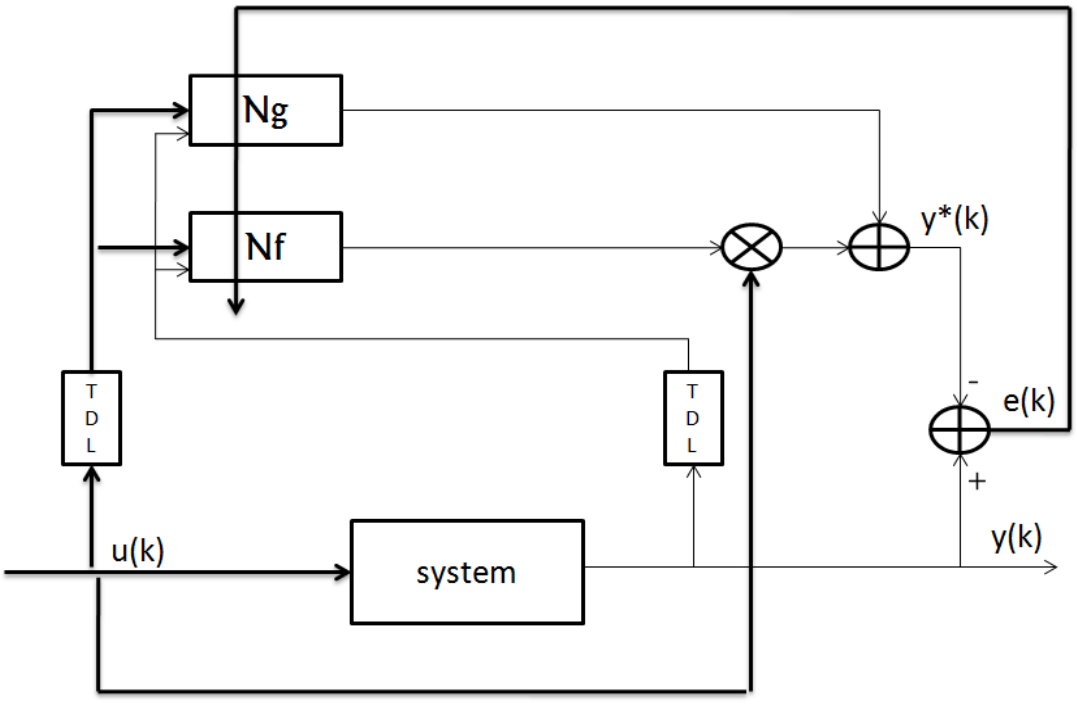


Figure 3.4. Block diagram of NARMA-L2 model.

control that we are using in this system is adaptive control indirectly, which is discussed before in this Chapter.

Now, given the goal of matching $y(k)$ with $yr(k)$, if we substitute the $y(k)$ with $yr(k)$ in the above approximation equations, and solve the resulting equation in terms of $u(k)$, the actual output of the real system can be matched to its optimal value, so the control objective is obtained:

$$y(k+d) = \overline{f_0}[y(k), y(k-1), \dots, y(k-n+1)] + \overline{g_0}[y(k), y(k-1), \dots, y(k-n+1), u(k-1), \dots, u(k-n+1)]u(k) \quad (3.4)$$

if

$$y(k+d) = yr(k+d) \quad (3.5)$$

then

$$u(k) = \frac{[yr(k+d) - \overline{f_0}(y(k),\dots,y(k-n+1),u(k-1),\dots,u(k-n+1))]}{\overline{g_0}(y(k),\dots,y(k-n+1),u(k-1),\dots,u(k-n+1))} \quad (3.6)$$

As you can see from Fig. 3.5 and 3.6, the controller is classic and only performs simple algebra actions based on the neural network signals and does not require a separate neural network for control action.

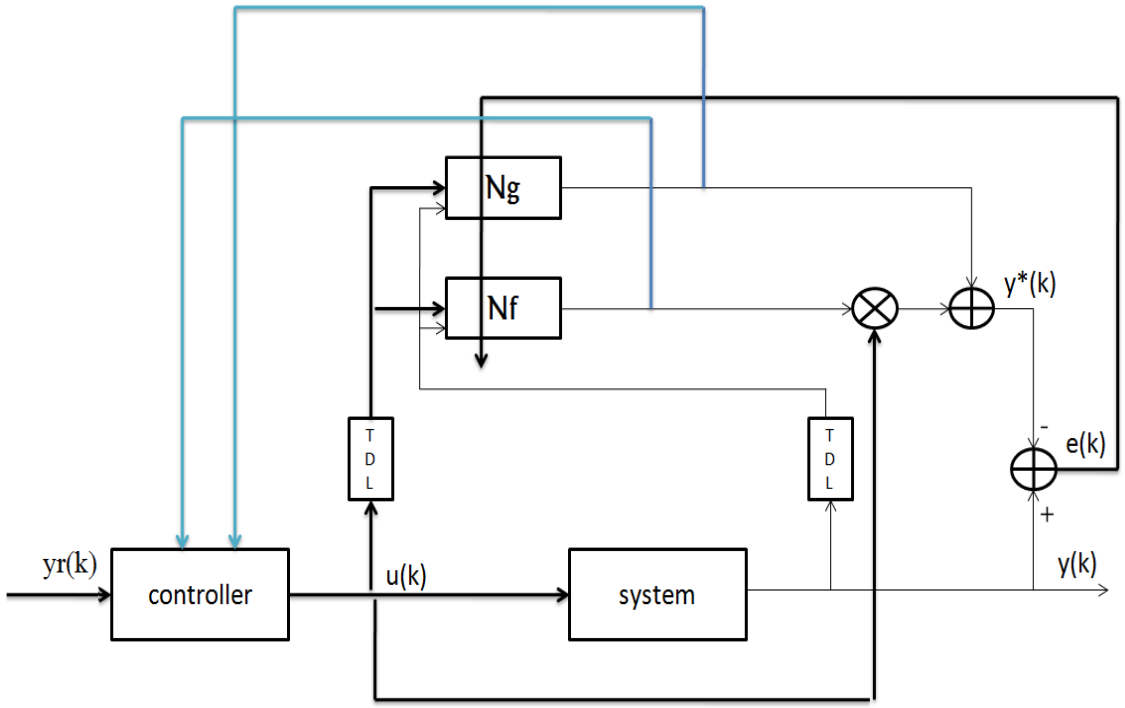


Figure 3.5. Block diagram of adaptive control system for NARMA-L2 model.

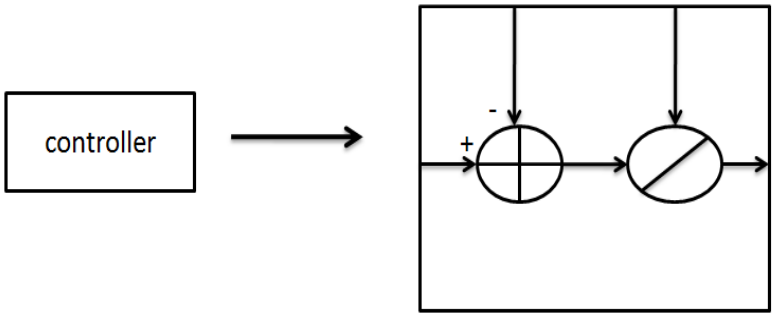


Figure 3.6. Algebraic controller.

Chapter 4. Simulation Results

In this Chapter, algorithm design for identification and adaptive control based on NARMA-L2 model will be described. Matlab simulation along with different examples of nonlinear systems are also provided.

4.1 Algorithm design for identification

The following steps are considered for identification process:

1. Choose the initial conditions for $w^{i,j}$ and b^i, the number of input and output sampling, number of network neurons, activation functions, network learning rates, the number of epochs or cycles.
2. For each input, the output of the network is obtained by the number of cycles.
3. In each cycle, the identification error $(y - y^*)$ is obtained, which is equal to the output of the system minus the output of the network).
4. Setting network parameters with respect to the identification error.

The neural network used for identification purpose is shown in Fig. 4.1. This neural network uses the $tanh$ activation function for the first and second layers and the linear activation function for the third layer.

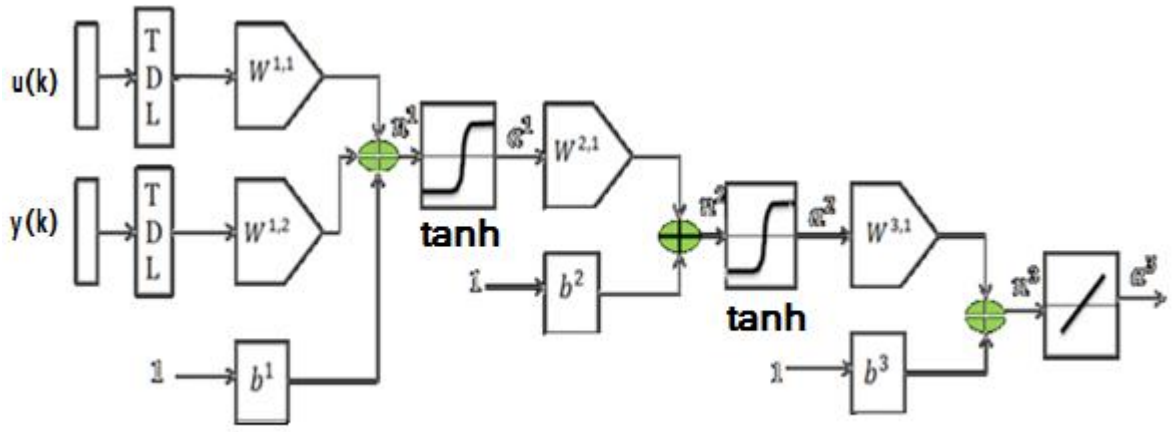


Figure 4.1. Neural network with the structure of $\frac{3}{2}$, 20,10,1

4.1.1 MATLAB simulation

In this sub-section, MATLAB simulation for the identification process is described. The number of neurons in the first layer of the neural network, the second layer, the number of sampling from the system input and the number of sampling from the system output are assigned as follows:

```
S1=20;%number of neuron1%
S2=10;%number of neuron2%
dy=2;%number of delay plant output%
du=3;%number of delay plant input%
```

The initial conditions for weights and bias are assigned as follows:

```
a=.0003;
w11=zeros(s1,du);
w12=zeros(s1,dy);
b1=a*rand(s1,1);
w13=zeros(s2,s1);
b2=a*rand(s2,1);
wend1=zeros(1,s2);
```

```
bend1=(0);
w21=zeros(s1,du);
w22=zeros(s1,dy);
b3=a*rand(s1,1);
w23=zeros(s2,s1);
b2=a*rand(s2,1);
wend2=zeros(1,s2);
bend2=(1);
```

The number of epochs or cycles as well as the learning rate are set as follows:

```
Epoch = 50;
alfa = .1; %learning rate%
```

The main part of the program is as follows. The data is given on-line to the network, u is the system input and y output system.

```
l = length (u);
for k=1:l
```

For each data, the network trains with the number of epochs.

```
for i=1:epoch
```

Sampling is done as follows:

```
for j=1:du
    utdl(j,1)=u(k-j+1);
end
for j=1:dy
    ytdl(j,1)=y(k-j+1);
end
```

ytdl, utdl are selected as the network input. The first layer of the neural network is formed as follows:

```
n1=(w11*utdl)+(w12*ytdl)+b1;
a1=tanh(n1);
n3=(w21*utdl)+(w22*ytdl)+b3;
a3=tanh(n3);
```

and the second and third layers are formed as follows:

```
n2=(w13*a1)+b2;
a2=tanh(n2);
n2=(w23*a3)+b2;
a2=tanh(n2);
aend1=(wend1*a2)+bend1;
aend2=(wend2*a2)+bend2;
```

The network's output that has to be similar to the actual system's output, is obtained here:

```
y❊(k)=aend1+(aend2*u(k));
```

At this step, the identification error is obtained.

```
e(k)=(y(k)-y❊(k));
```

With respect to the identification error of the neural network, the sensitivities are calculated based on the error back-propagation method:

```
delta13=-2*e(k);
delta12=(1-(a2.^2)).*wend1'*delta13;%s2*1
```

```
delta11=(1-(a1.^2)).*(w13'*delta12);%s1*1
delta23=-2*e(k);
delta22=(1-(a2.^2)).*wend2'*delta23;%s2*1
delta21=(1-(a3.^2)).*(w23'*delta22);%s1*1
```

Depending on the sensitivities' values, as well as the learning rate's value, the weights and bias will changed in each epoch as follows:

```
wend1=wend1-alfa*delta13*a2';
bend1=bend1-alfa*delta13;
w13=w13-alfa*delta12*a1';
b2=b2-alfa*delta12;
w11=w11-alfa*delta11*utdl';
w12=w12-alfa*delta11*ytdl';
b1=b1-alfa*delta11;
wend2=wend2-alfa*delta23*a2';
bend2=bend2-alfa*delta23;
w23=w23-alfa*delta22*a3';
b2=b2-alfa*delta22;
w21=w21-alfa*delta21*utdl';
w22=w22-alfa*delta21*ytdl';
b3=b3-alfa*delta21;
   end
end
```

4.1.2 Identification examples

Two examples have been considered here to show the identification process using NARMA-L2 model.

Example 1: Identification of a first-degree plant that is characterized by the following equation:

$$y(k+1) = sin[y(k)] + u(k) * (5 + cos[(y(k) * u(k)]) \qquad (4.1)$$

The experimental results are obtained for different inputs, different training rates and different epochs. Fig 4.2 shows the first input using the following formula:

$$u = .5 * \left(sin\left(\frac{2\pi k}{50}\right) + sin\left(\frac{2\pi k}{100}\right)\right) \qquad (4.2)$$

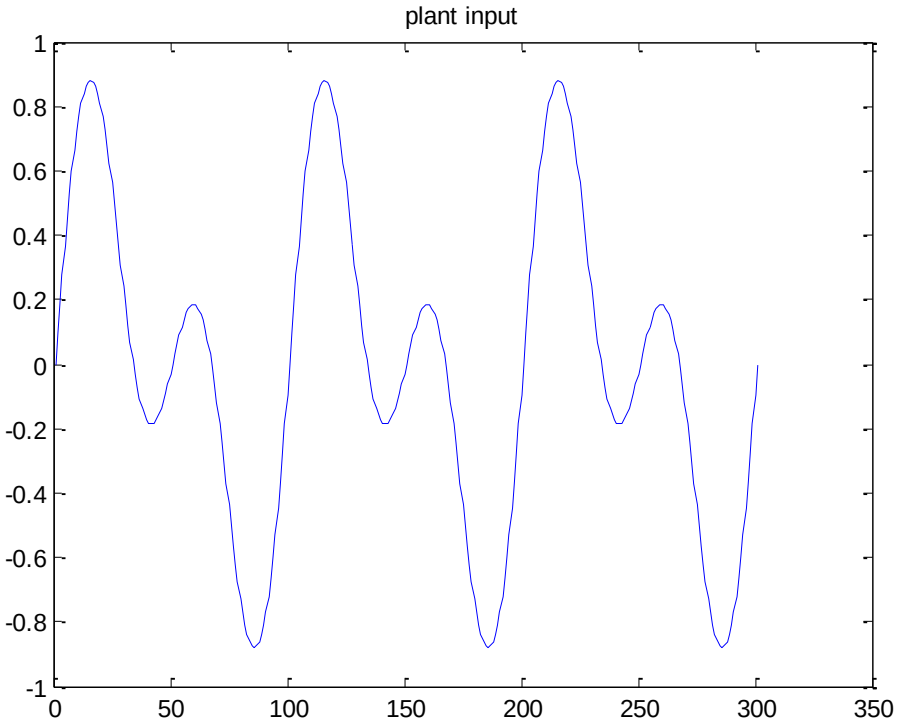


Fig. 4.2. Plant input using equation (4.2) for first example.

The output of system based on equation (4.1) and (4.2) is shown in Fig. 4.3.

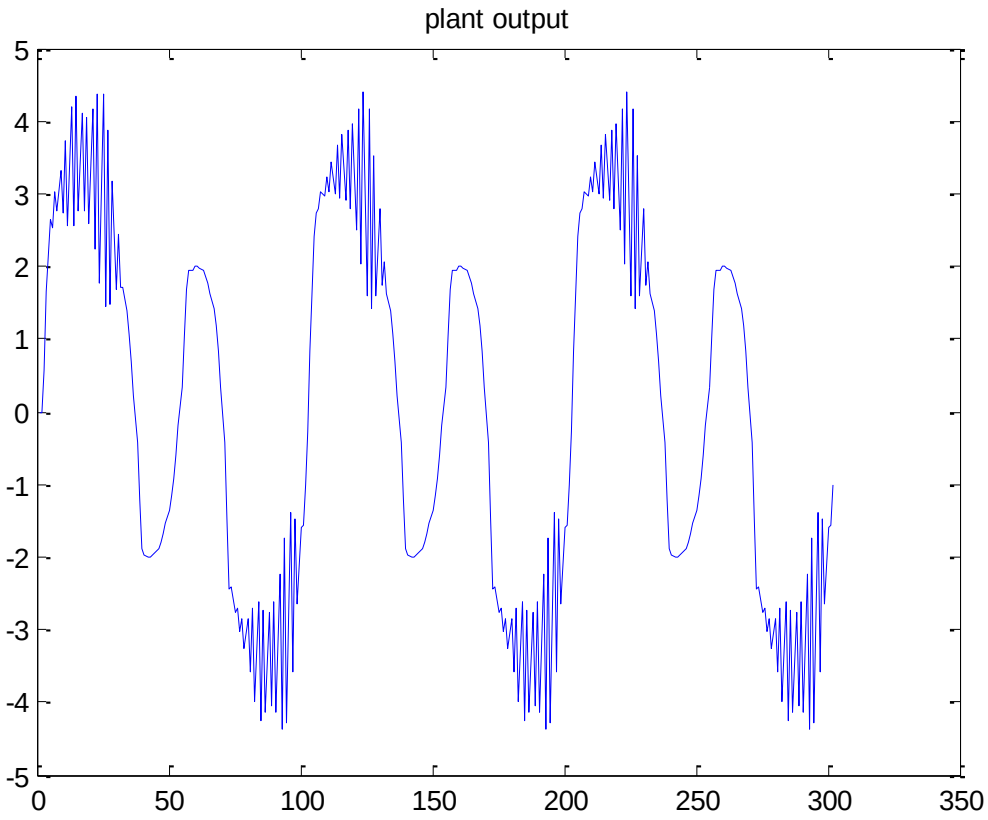


Fig. 4.3. Plant output for first example.

The identification results for different epochs and learning rates along with identification errors are shown in Figs. 4 and 5. As you can see from the results, the network will be unstable using high learning rates.

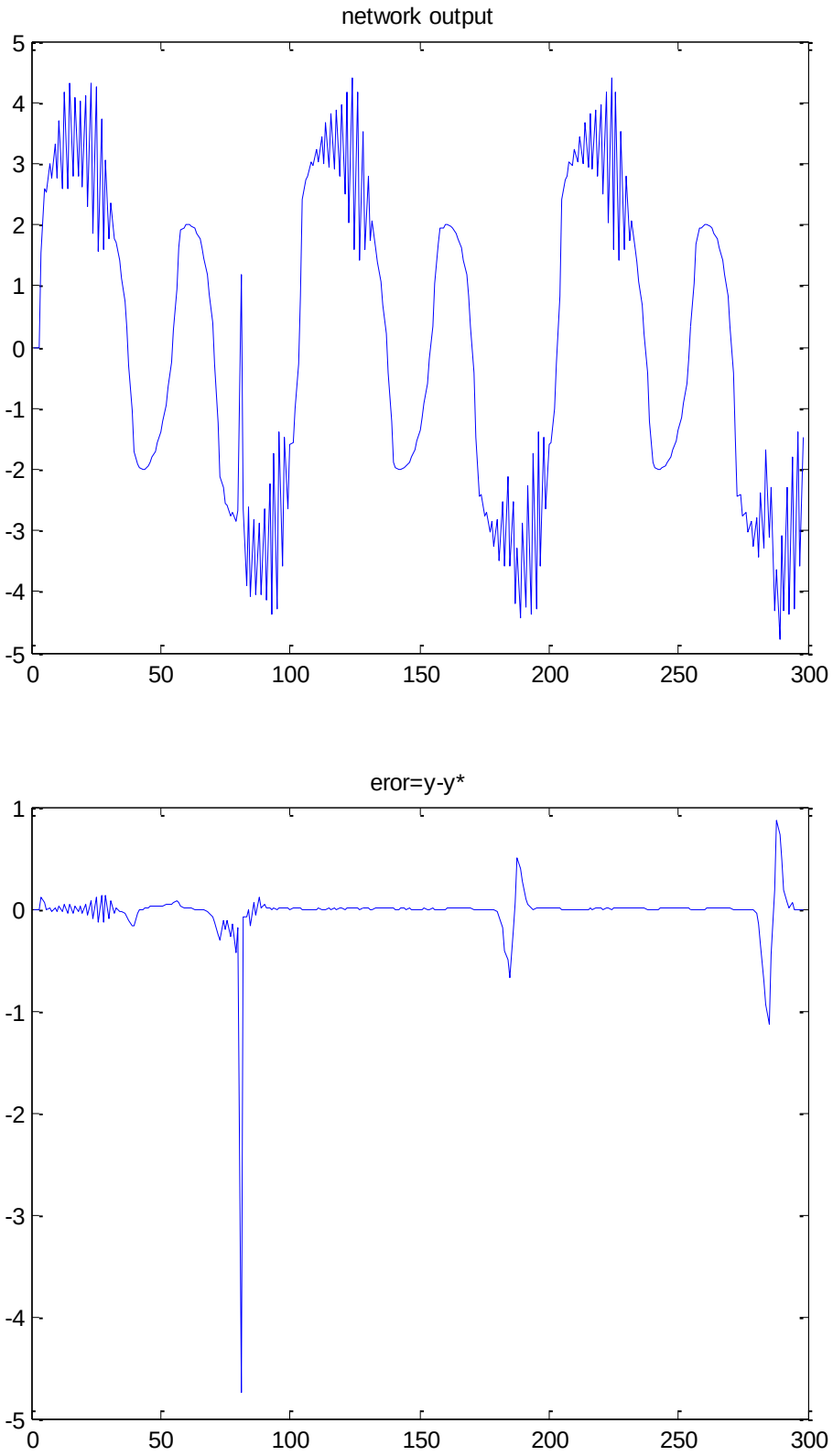


Fig. 4.4. Top: network output and bottom: identification error, for 100 epochs and learning rate equal to 0.01.

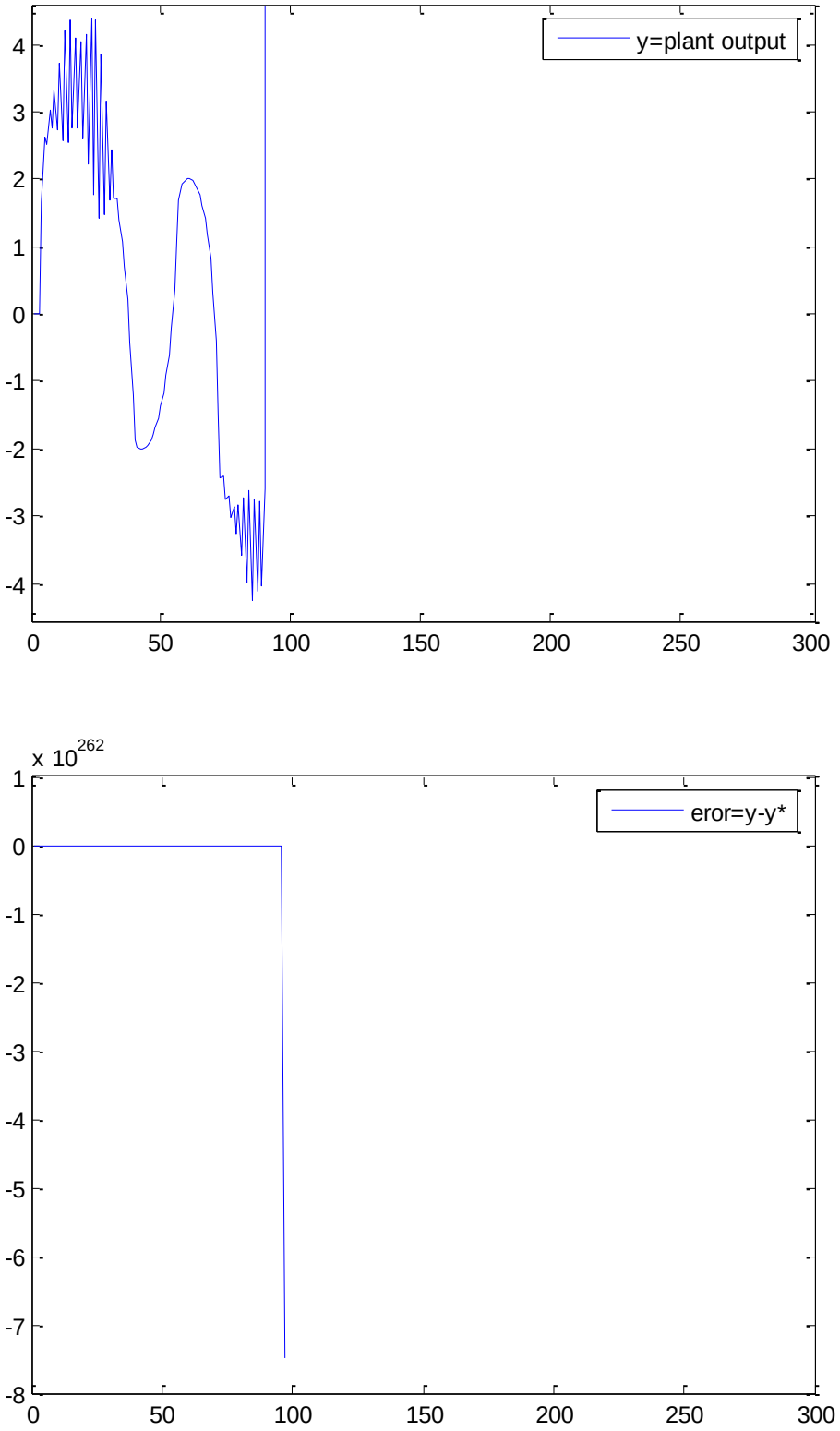


Fig. 4.5. Top: network output and bottom: identification error, for 100 epochs and learning rate equal to 0.02.

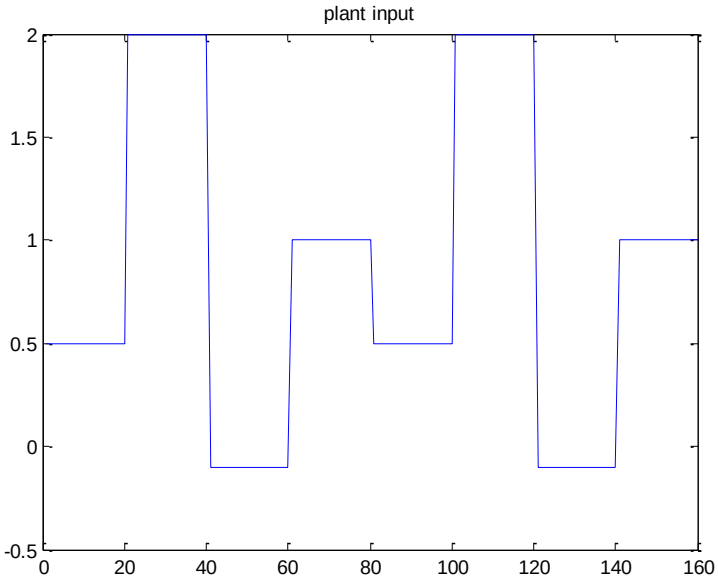


Fig. 4.6. Top: Step function as second input.

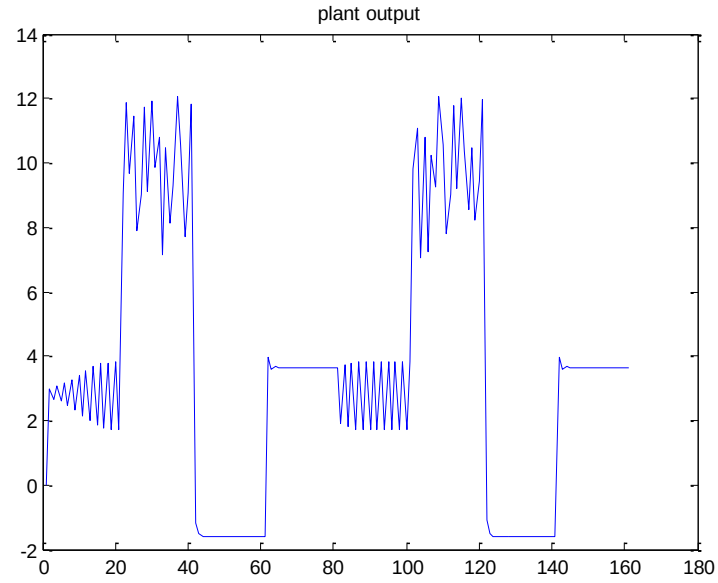


Fig. 4.7. Plant output using step function.

Fig 4.6 shows the second input for first example using step function. The output of system based on step function as input and equation (4.2) is shown in Fig. 4.7. In addition, the identification results along with identification errors are shown in Fig. 8. As it can be seen, the identification operation is carried out with high precision.

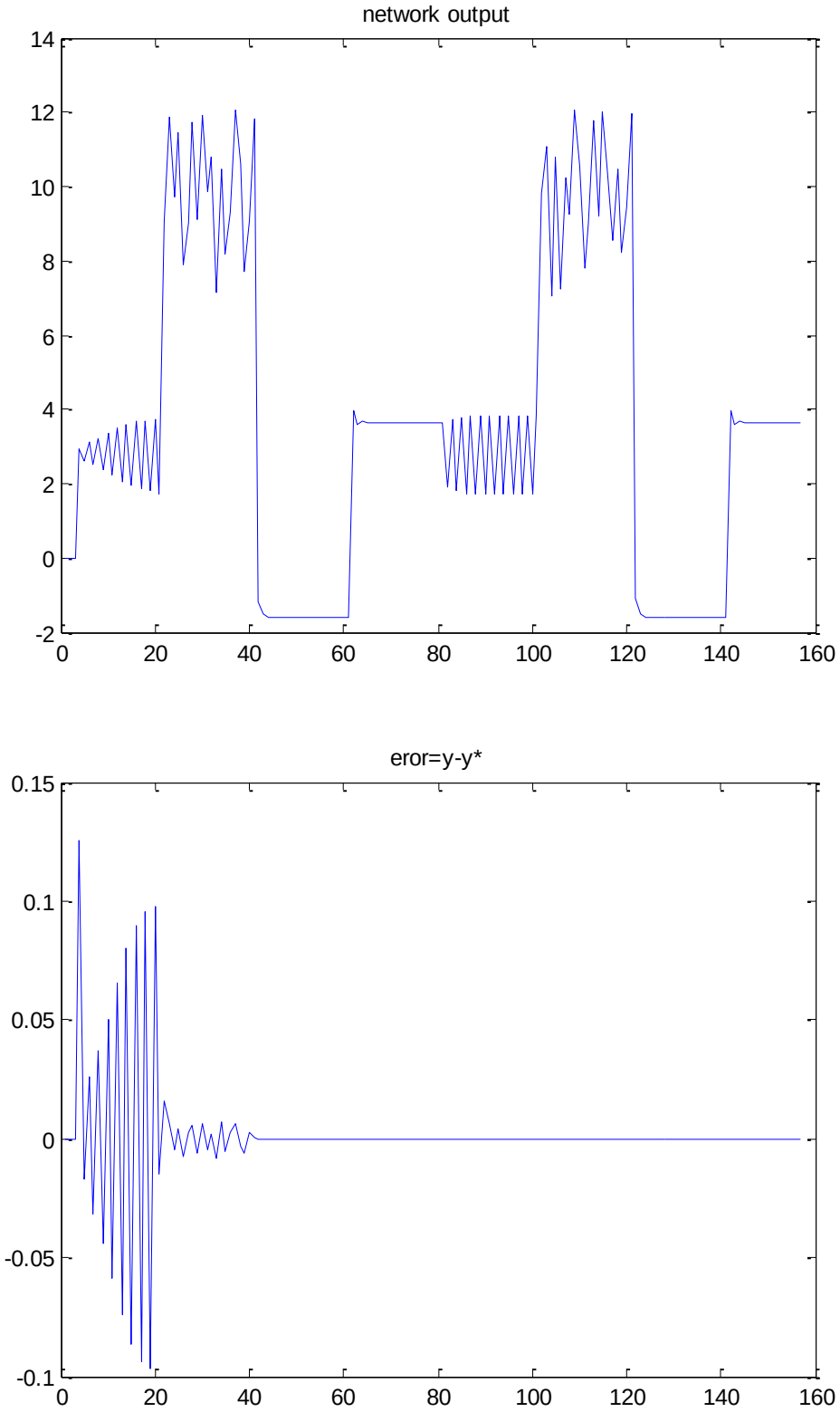


Fig. 4.8. Top: network output and bottom: identification error, for 100 epochs and learning rate equal to 0.001.

Example 2- Identification of a second-order plant that is characterized by the following equation:

$$x_1(k+1) = .1 * x_1(k) + 2\frac{u(k) + x_2(k)}{1 + (u(k) + x_2(k))^2} \quad (4.3)$$

$$x_2(k+1) = .1 * x_2(k) + u(k)\left(2 + \frac{u^2(k)}{1 + x_1{}^2(k) + x_2{}^2(k)}\right) \quad (4.4)$$

$$y(k) = x_1(k) + x_2(k) \quad (4.5)$$

The output of system based on equation (4.2) and (4.5) is shown in Fig. 4.9. The identification results for different epochs and learning rates along with identification errors are shown in Figs. 10 and 11. As you can see from the results, for higher learning rates and similar epoch, the network at a higher rate performs better identification.

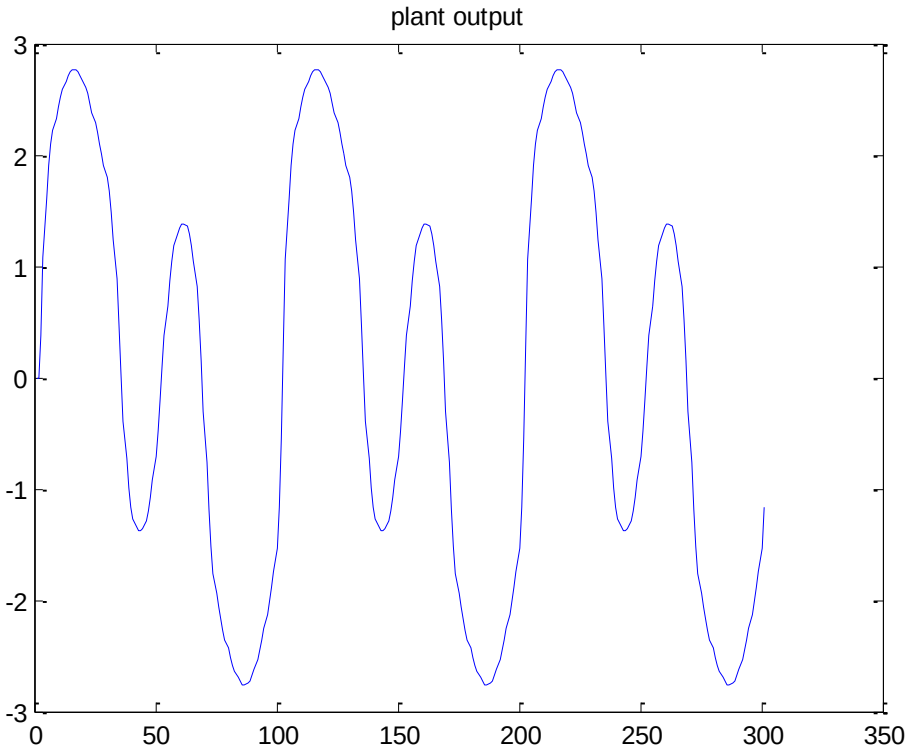


Fig. 4.9. Plant output for second example.

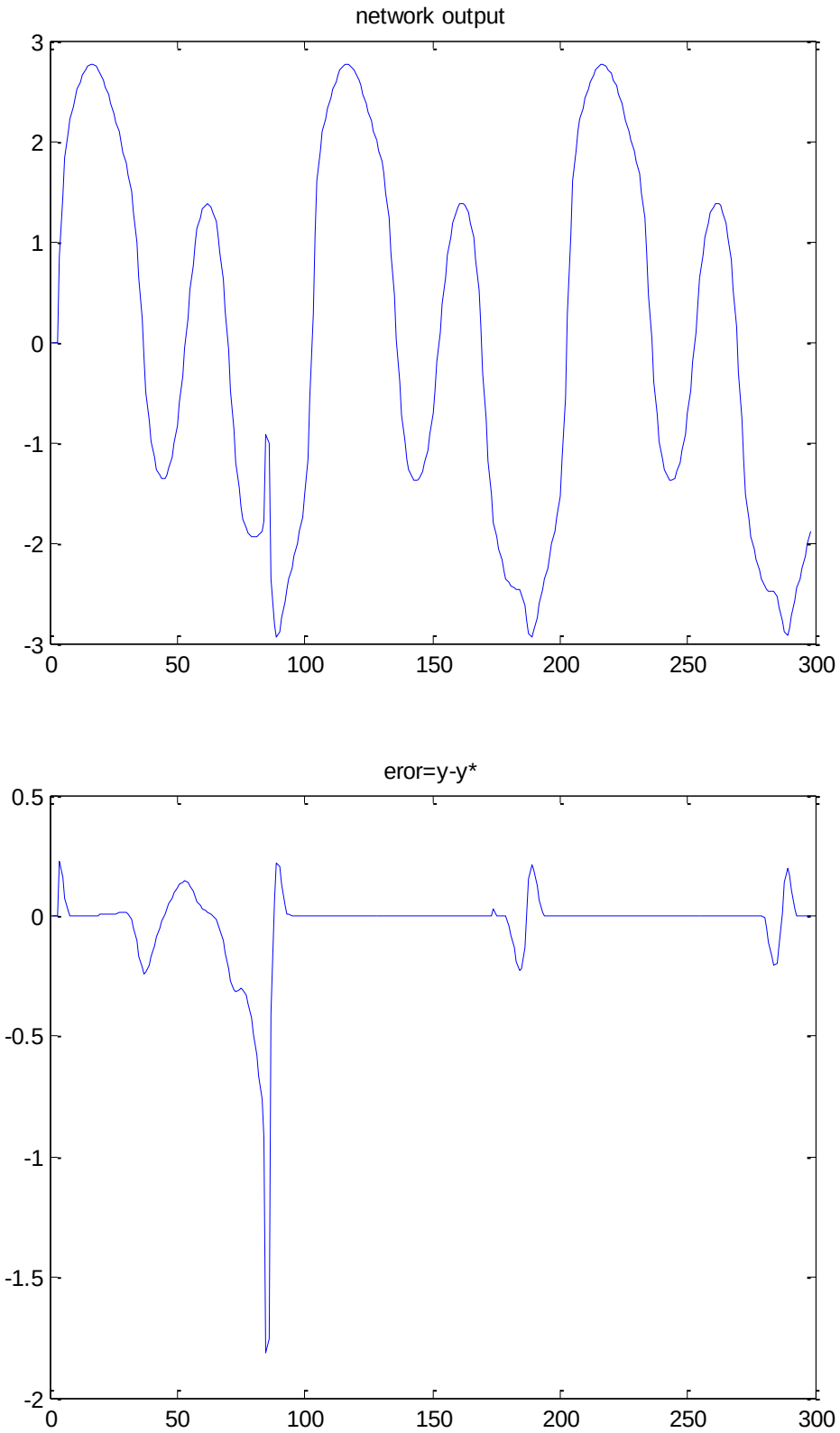


Fig. 4.10. Top: network output and bottom: identification error, for 50 epochs and learning rate equal to 0.01.

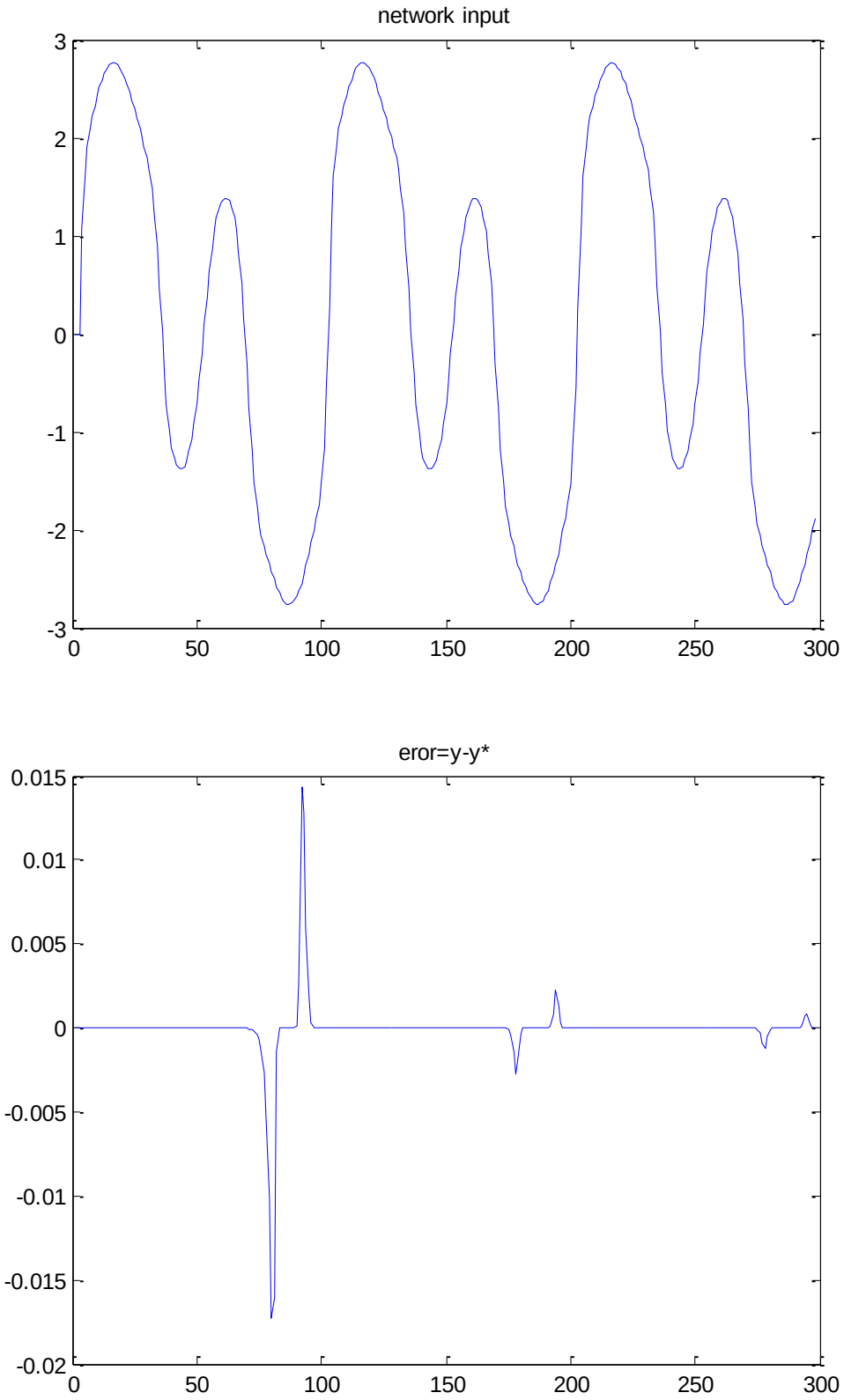


Fig. 4.11. Top: network output and bottom: identification error, for 50 epochs and learning rate equal to 0.1.

The output of system based on step function as input and equation (4.5) is shown in Fig. 4.12. In addition, the identification results along with identification errors are shown in Fig. 4.13. As it can be seen, the identification operation is carried out with high precision.

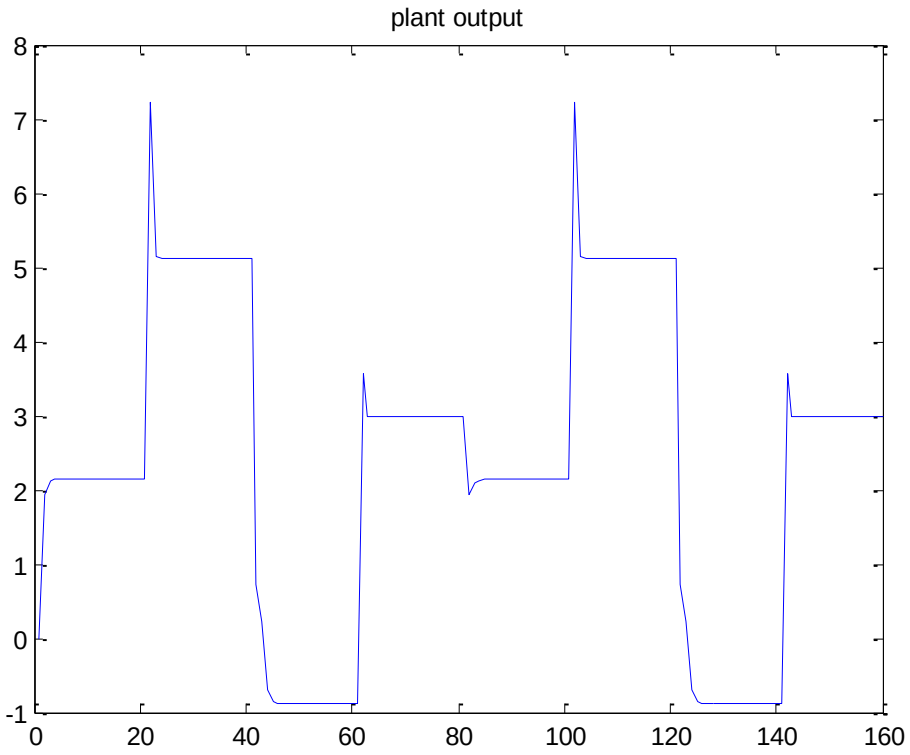


Fig. 4.12. Plant output using step function for second example.

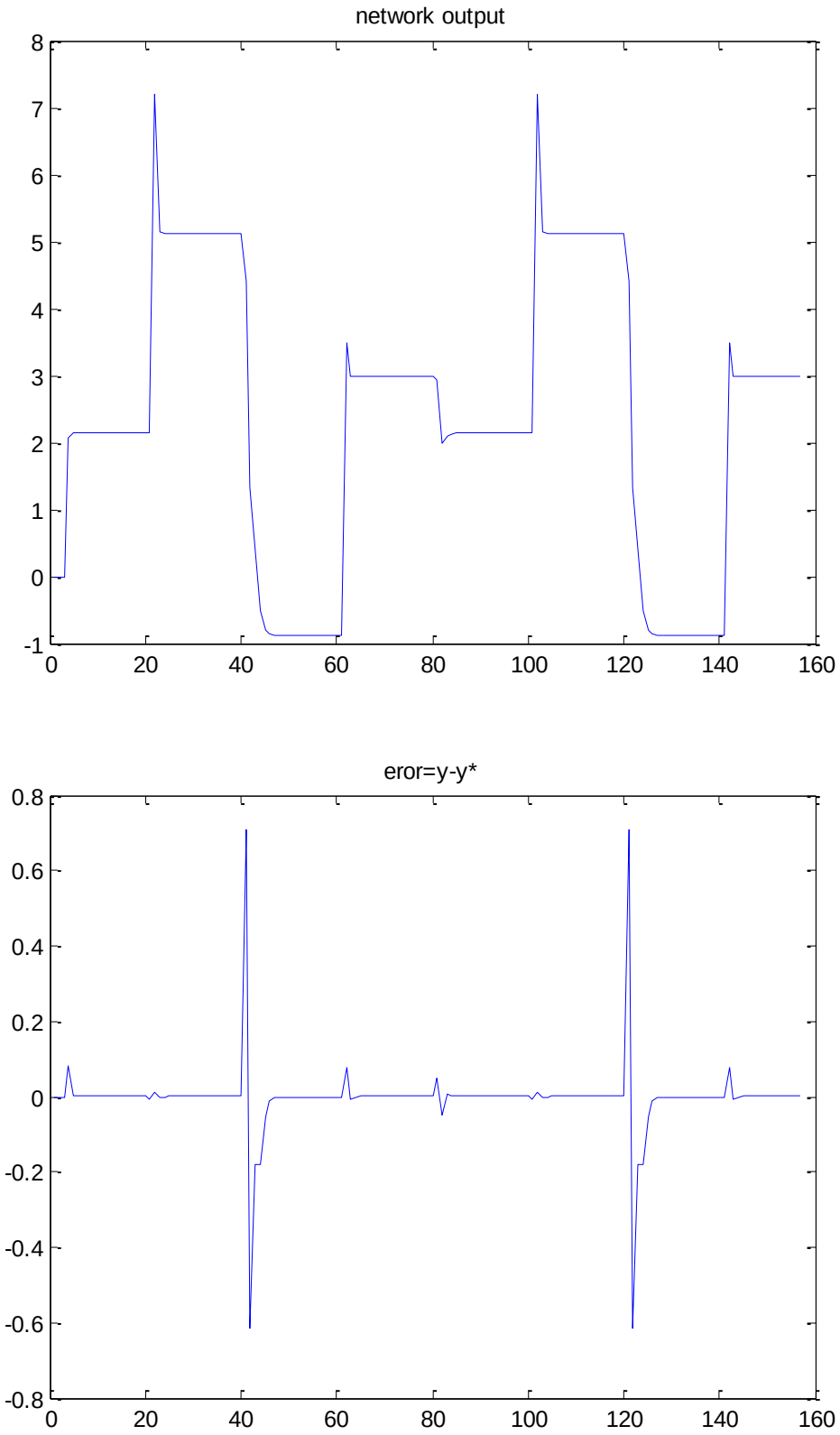


Fig. 4.13. Top: network output and bottom: identification error, for 100 epochs and learning rate equal to 0.01.

4.2 Algorithm design for adaptive control

The following steps are considered for adaptive control process:

1. Choose the initial conditions for $w^{i,j}$ and b^{i}, the number of input and output sampling, number of network neurons, activation functions, network learning rates, the number of epochs or cycles.
2. For each reference input, the system input is obtained by the number of cycles:

$$u(k) = \frac{[yr(k+d) - \bar{f}_0(y(k), \dots, y(k-n+1), u(k-1), \dots, u(k-n+1))]}{\overline{g_0}(y(k), \dots, y(k-n+1), u(k-1), \dots, u(k-n+1))} \quad (4.6)$$

3. In each cycle, the identification error $(y - y^*)$ is obtained, which is equal to the output of the system minus the output of the network).

4. Setting network parameters with respect to the identification error.

4.2.1 MATLAB simulation

In this sub-section, MATLAB simulation for the adaptive control process is described. First, the input of the reference is selected and the initial values of the input, the output of the system, are selected as zero.

```
yr = 2*(sin(2*pi*k/50)+sin(2*pi*k/100));%reference
u = 0.*yr;
y = 0.*yr;
```

The number of neurons in the first layer of the neural network, the second layer, the number of sampling from the system input and the number of sampling from the system output are assigned as follows:

```
S1=20;%number of neuron1%
S2=10;%number of neuron2%
dy=2;%number of delay plant output%
```

```
du=3;%number of delay plant input%
```

The initial conditions for weights and bias are assigned as follows:

```
a=.0003;
w11=zeros(s1,du);
w12=zeros(s1,dy);
b1=a*rand(s1,1);
w13=zeros(s2,s1);
b2=a*rand(s2,1);
wend1=zeros(1,s2);
bend1=(0);
w21=zeros(s1,du);
w22=zeros(s1,dy);
b3=a*rand(s1,1);
w23=zeros(s2,s1);
b2=a*rand(s2,1);
wend2=zeros(1,s2);
bend2=(1);
```

The number of epochs or cycles as well as the learning rate are set as follows:

```
epoch =50;
alfa=.1;%learning rate%
```

The main part of the program is as follows. The data is given on-line to the network, u is the system input and y output system.

```
l = length (u);
for k=1:l
```

For each data, the network trains with the number of epochs.

```
for i=1:epoch
```

Sampling is done as follows:

```
for j=1:du
utdl(j,1)=u(k-j+1);
end
for j=1:dy
ytdl(j,1)=y(k-j+1);
end
```

ytdl, utdl are selected as the network input. The first layer of the neural network is formed as follows:

```
n1=(w11*utdl)+(w12*ytdl)+b1;
a1=tanh(n1);
n3=(w21*utdl)+(w22*ytdl)+b3;
a3=tanh(n3);
```

and the second and third layers are formed as follows:

```
n2=(w13*a1)+b2;
a2=tanh(n2);
n2=(w23*a3)+b2;
a2=tanh(n2);
aend1=(wend1*a2)+bend1;
aend2=(wend2*a2)+bend2;
```

The new system's input is obtained at this stage.

```
u(k) = (((yr(k))-aend1)/aend2);
```

The new system's input is passed through the nonlinear system at this stage to obtain a new output.

```
x1(k+1)=.1*x1(k)+(2*(u(k)+x2(k))/(1+(u(k)+x2(k))^2));
x2(k+1)=.1*x2(k)+u(k)*(2+(u(k)^2/(1+x1(k)^2+x2(k)^2)));
```

```
y(k)=x1(k)+x2(k);
```

The network's output that has to be similar to the actual system's output, is obtained here:

```
y❊(k)=aend1+(aend2*u(k));
```

At this step, the identification error is obtained.

```
e(k)=(y(k)-y❊(k));
```

With respect to the identification error of the neural network, the sensitivities are calculated based on the error back-propagation method:

```
delta13=-2*e(k);
delta12=(1-(a2.^2)).*wend1'*delta13;%s2*1
delta11=(1-(a1.^2)).*(w13'*delta12);%s1*1
delta23=-2*e(k);
delta22=(1-(a2.^2)).*wend2'*delta23;%s2*1
delta21=(1-(a3.^2)).*(w23'*delta22);%s1*1
```

Depending on the sensitivities' values, as well as the learning rate's value, the weights and bias will changed in each epoch as follows:

```
wend1=wend1-alfa*delta13*a2';
bend1=bend1-alfa*delta13;
w13=w13-alfa*delta12*a1';
b2=b2-alfa*delta12;
w11=w11-alfa*delta11*utdl';
w12=w12-alfa*delta11*ytdl';
b1=b1-alfa*delta11;
wend2=wend2-alfa*delta23*a2';
```

```
bend2=bend2-alfa*delta23;
w23=w23-alfa*delta22*a3';
b2=b2-alfa*delta22;
w21=w21-alfa*delta21*utdl';
w22=w22-alfa*delta21*ytdl';
b3=b3-alfa*delta21;
  end
end
```

4.2.2 Adaptive control examples

Two examples have been considered here to show the adaptive control process using NARMA-L2 model.

Example 1: adaptive control of a first-degree plant that is characterized by the following equation:

$$y(k+1) = sin[y(k)] + u(k) * (5 + cos[y(k) * u(k)]) \qquad (4.7)$$

The experimental results are obtained for different inputs, different training rates, and different epochs are shown in Figs. 15 and 16. Fig 4.14 shows the reference input using the step function:

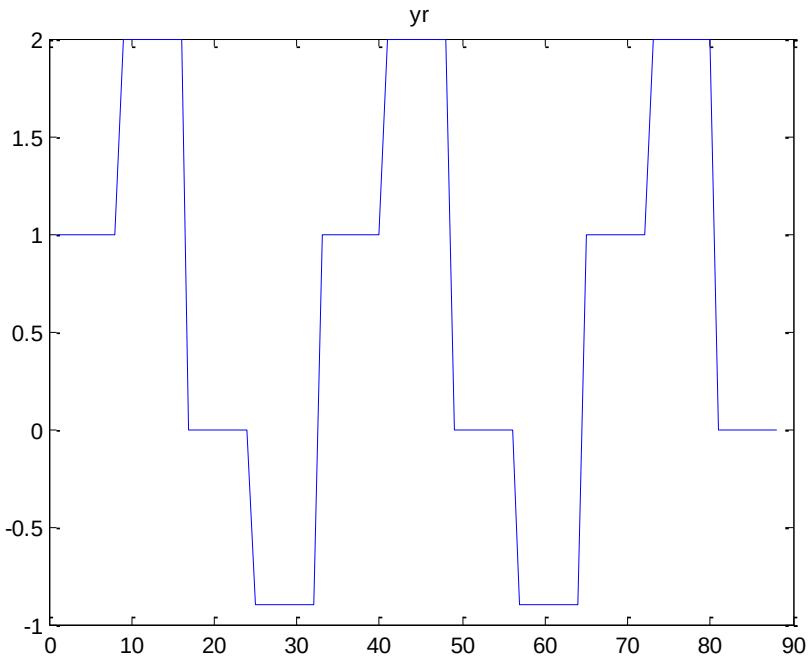
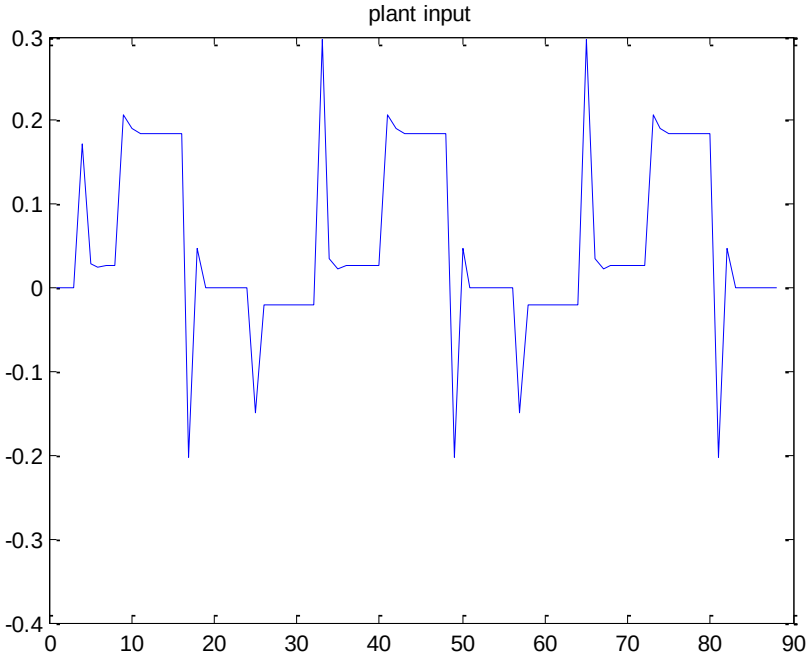


Fig. 4.14. Reference input using step function.



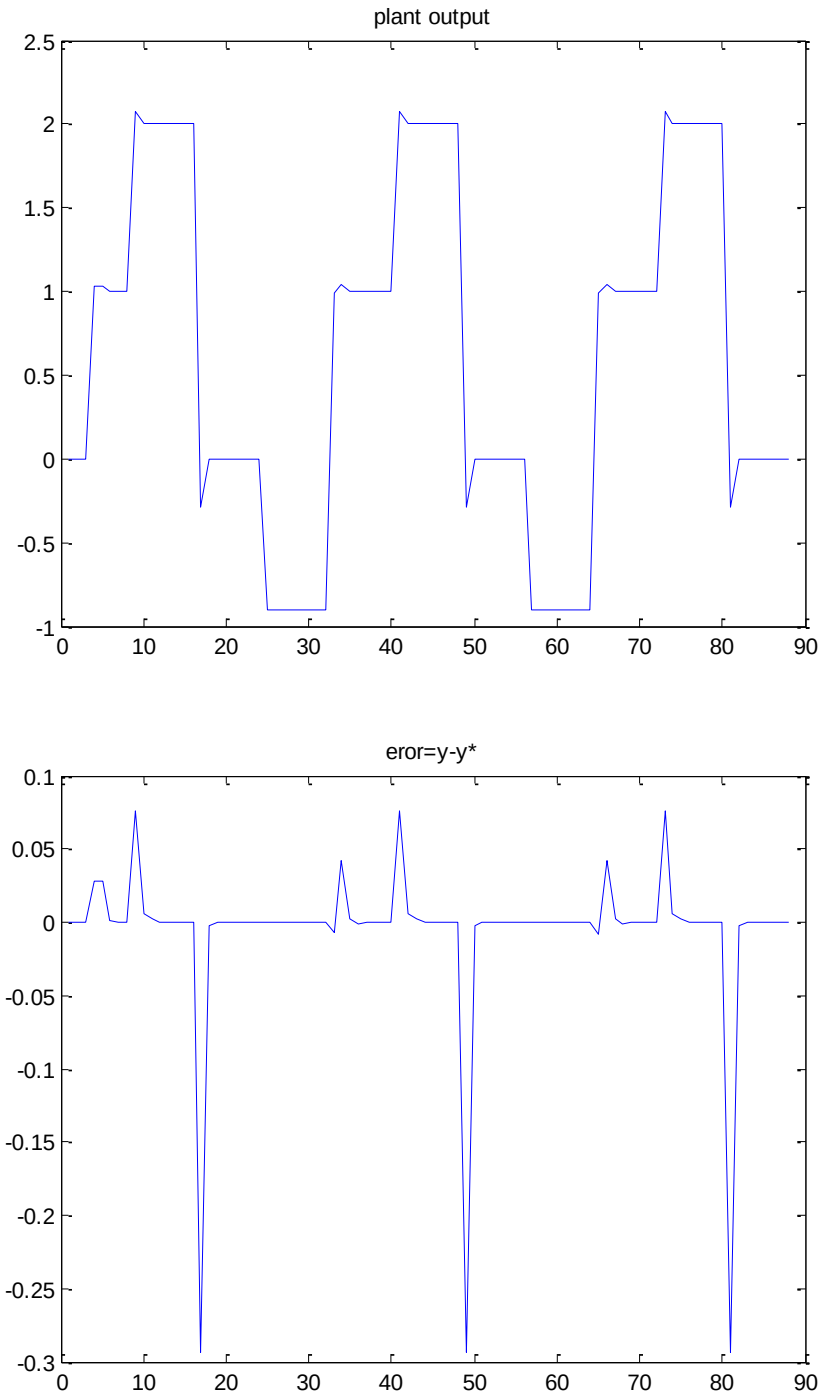
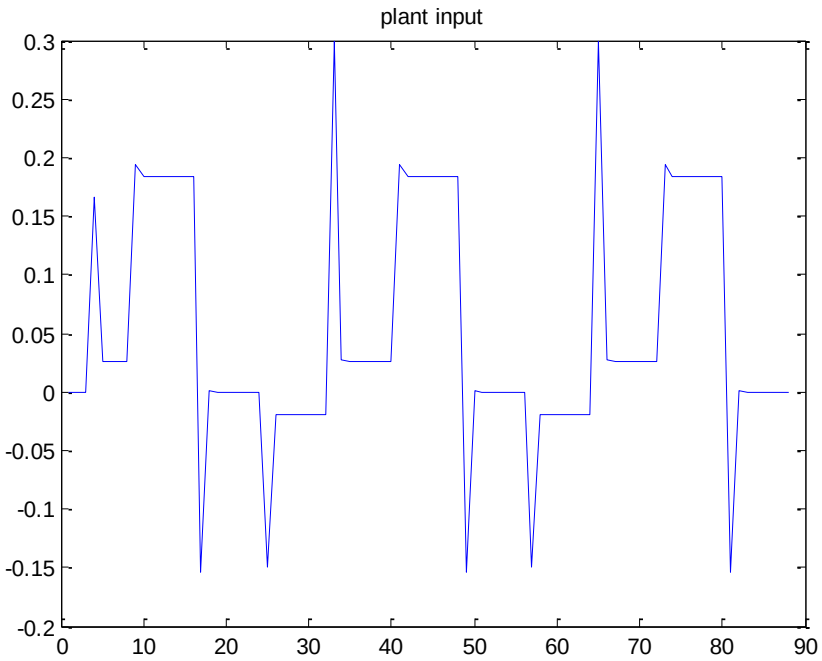
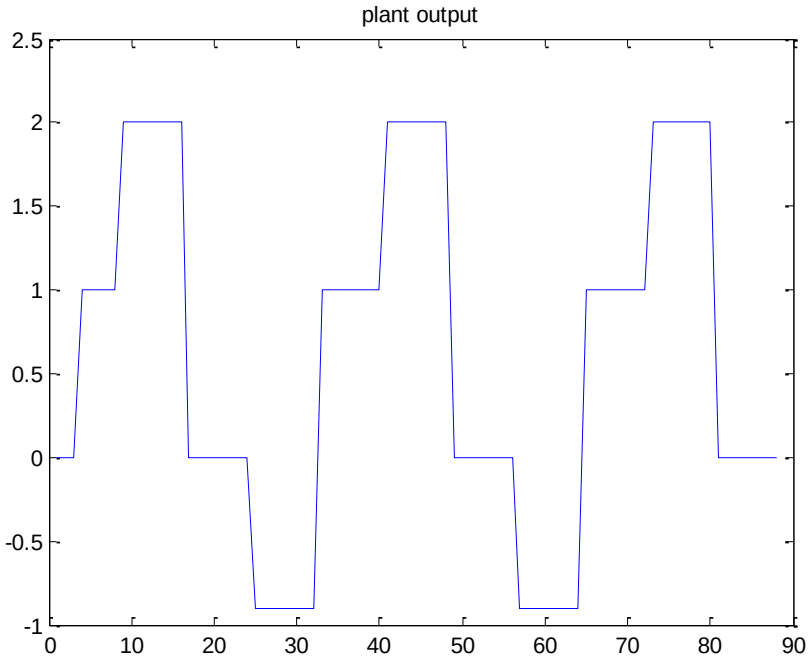


Fig. 4.15. Top: network input, middle: network output and bottom: identification error, for 50 epochs and learning rate equal to 0.01.


plant input
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
10
20
30
40
50
60
70
80
90


plant output
2.5
2
1.5
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
70
80
90

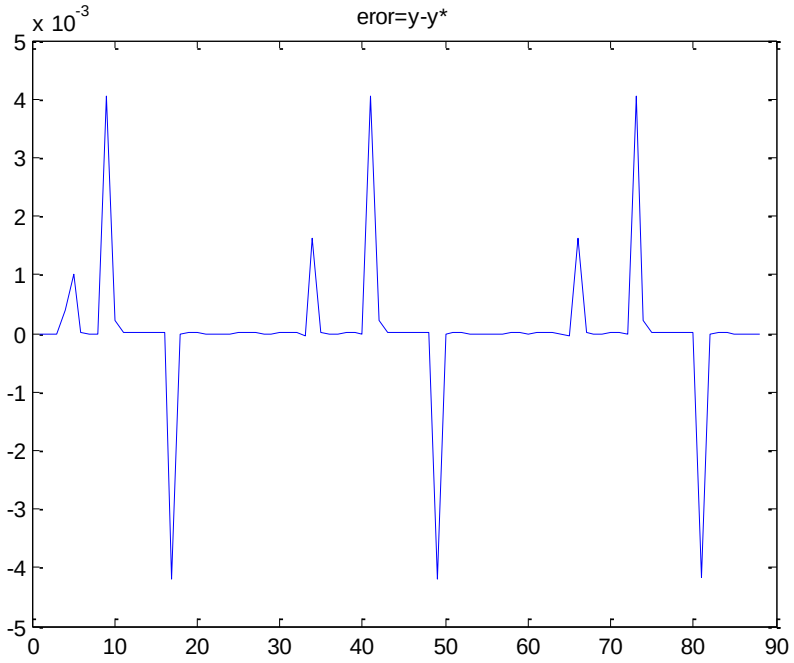


Fig. 4.16. Top: network input, middle: network output and bottom: error signal, for 100 epochs and learning rate equal to 0.01.

Fig 4.17. shows the second reference for first example using sinusoids functions. In addition, the adaptive control results along with system's errors are shown in Fig. 18. As it can be seen, the adaptive control operation is carried out with high precision.

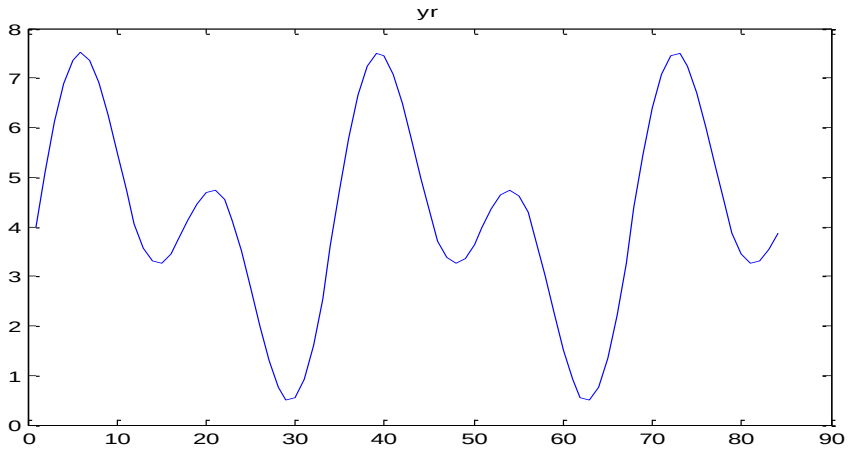
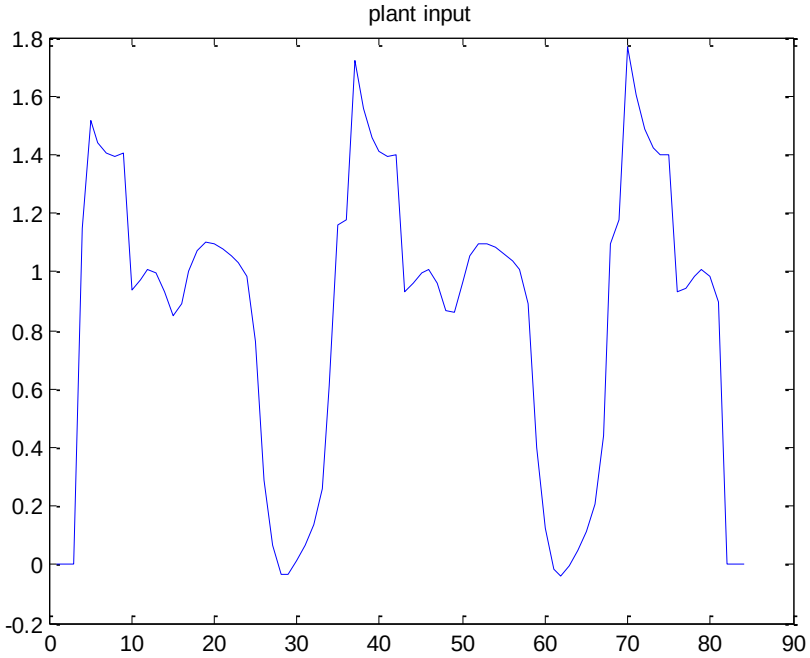
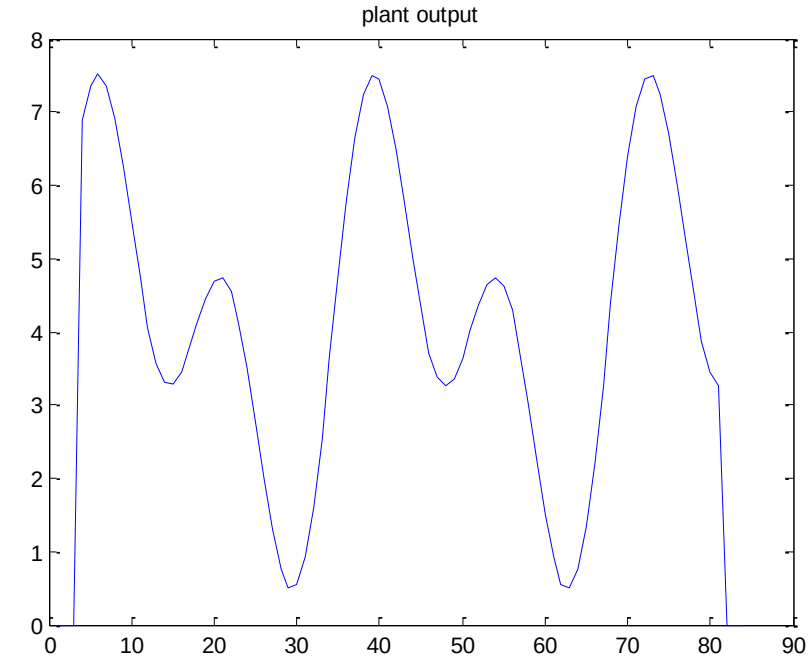


Fig. 4.17. Reference input using sinusoids functions.


plant input
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
10
20
30
40
50
60
70
80
90


plant output
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90

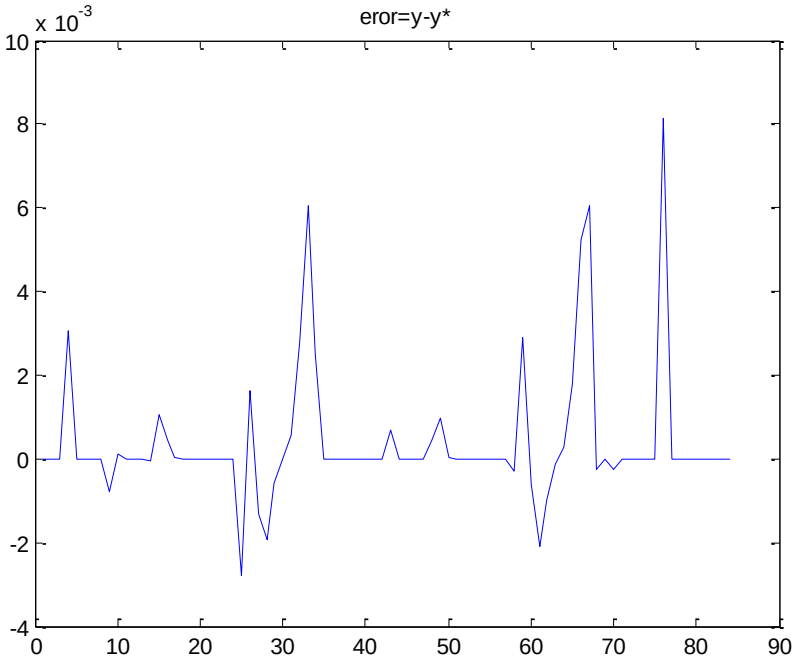


Fig. 4.18. Top: network input, middle: network output and bottom: error signal, for 100 epochs and learning rate equal to 0.01.

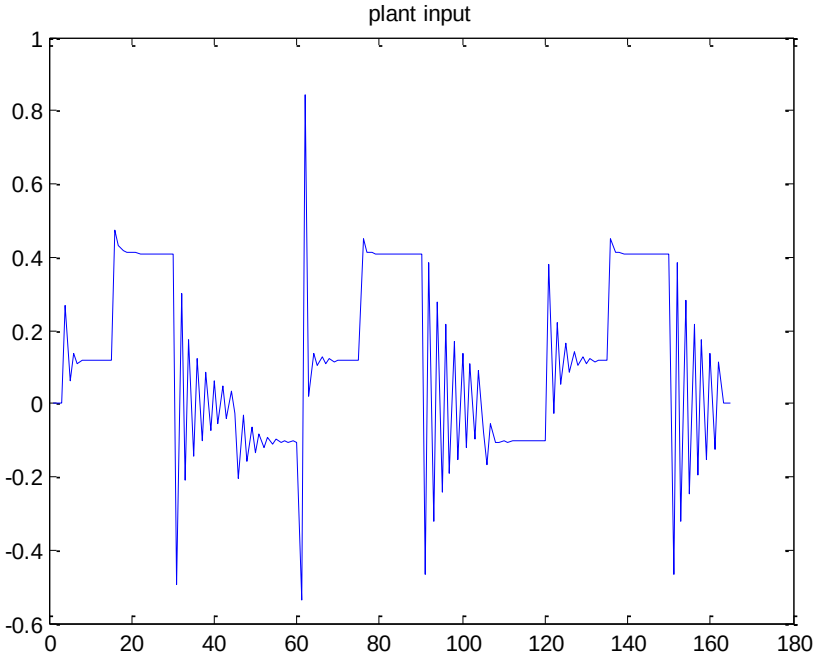
Example 2- Adaptive control of a second-order plant that is characterized by the following equation:

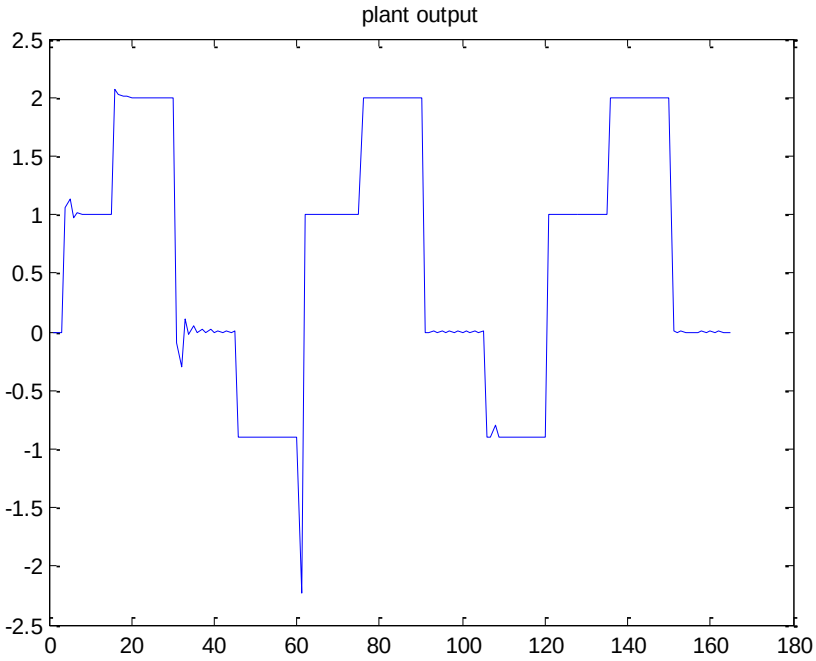
$$x_1(k+1) = .1 * x_1(k) + 2\frac{u(k) + x_2(k)}{1 + (u(k) + x_2(k))^2} \tag{4.8}$$

$$x_2(k+1) = .1 * x_2(k) + u(k)\left(2 + \frac{u^2(k)}{1+{x_1}^2(k)+{x_2}^2(k)}\right) \tag{4.9}$$

$$y(k) = x_1(k) + x_2(k) \tag{4.10}$$

The experimental results are obtained for different inputs, different training rates and different epochs are shown in Figs. 19 and 20. As it can be seen from the results, the system with higher epochs results in better tracking of desired output. and adaptive control of non-linear system.


plant input
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
20
40
60
80
100
120
140
160
180


plant output
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
20
40
60
80
100
120
140
160
180

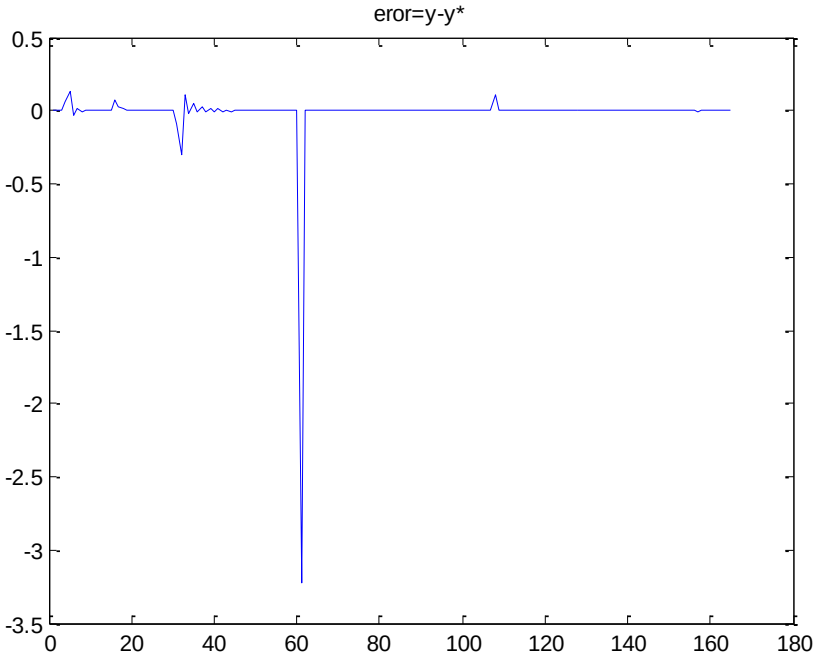
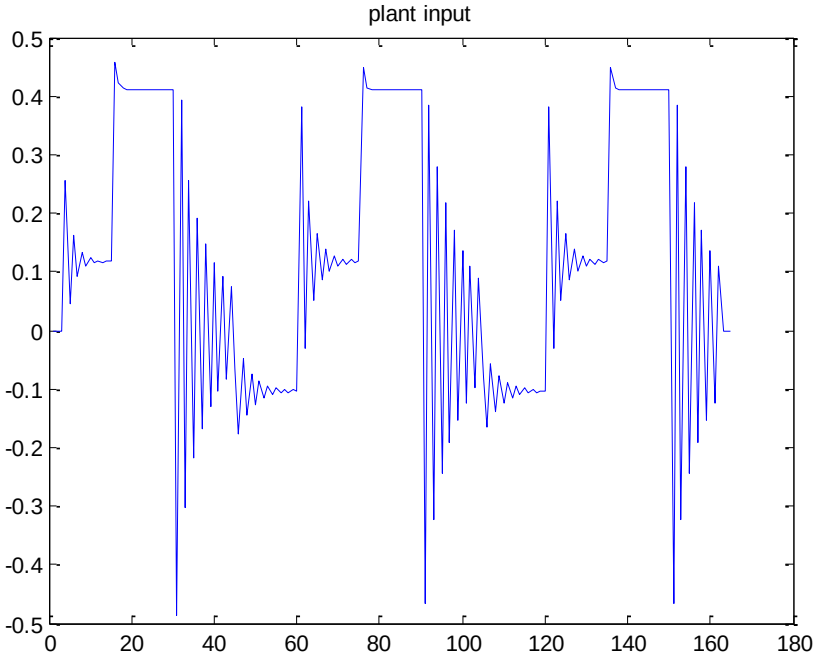


Fig. 4.19. Top: network input, middle: network output and bottom: error signal, for 100 epochs and learning rate equal to 0.005.



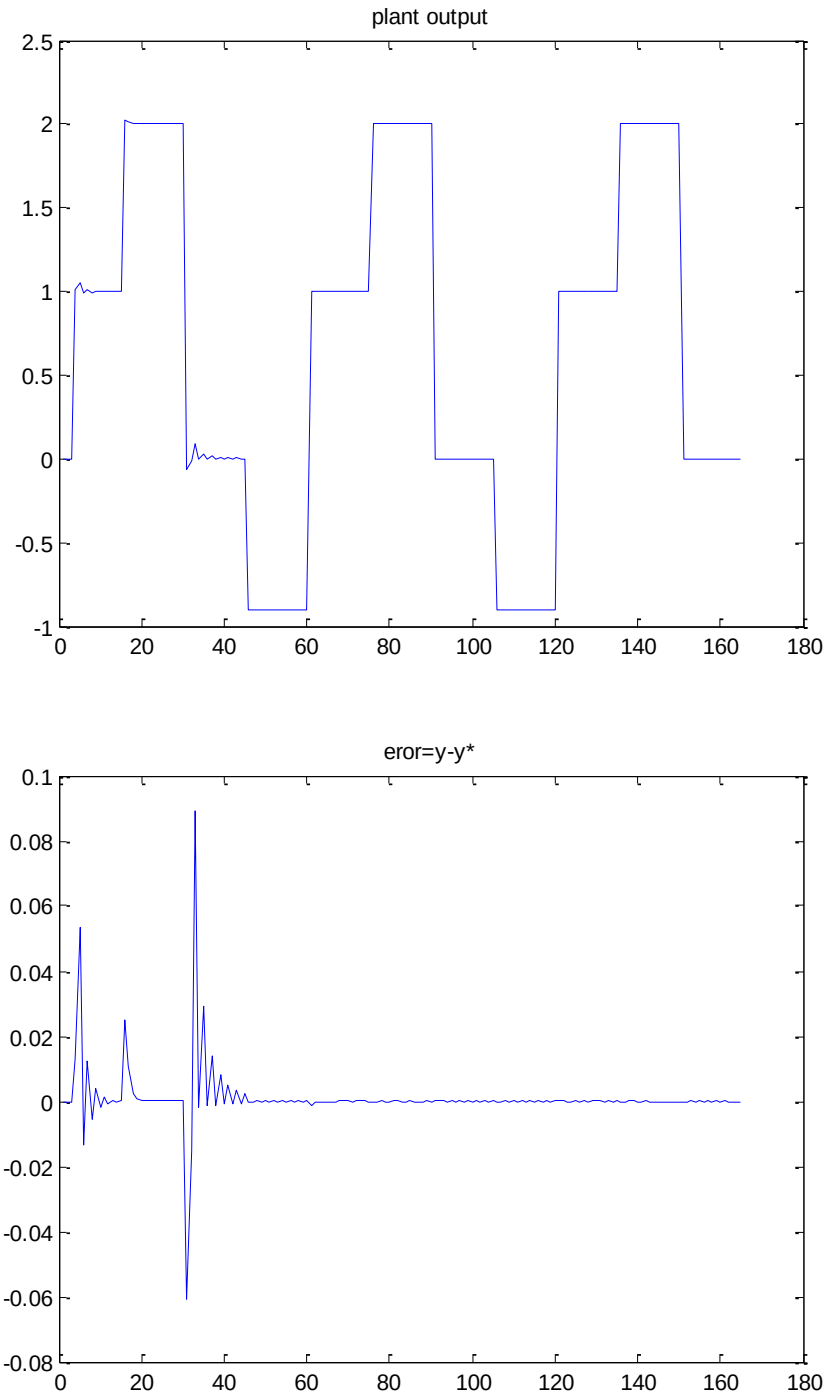


Fig. 4.19. Top: network input, middle: network output and bottom: error signal, for 150 epochs and learning rate equal to 0.005.

Chapter 5. Conclusions

From the above discussion, we conclude that by approximating the exact model of NARMA for each nonlinear system, we can reach to the approximate models of NARMA-L1, NARMA-L2. These approximate models with respect to the actual model is not only highly accurate for identification purpose, but also make the controller a simple classical controller and no longer need a separate neural network for controller with its own problems (weight correction problems). Therefore, the adaptive control structure has been simplified, while the precision is stayed very high and therefore these approximate models are usually used in the adaptive control of nonlinear systems.

Perhaps of greatest significance for the use of neural networks in the control of nonlinear dynamical systems is the fact that the NARMA-L1 and NARMA-L2 models are more tractable analytically than the NARMA model. If the stability, controllability and observability, as well as the zero dynamics of dynamical systems can be studied for the class of systems represented by these approximate models, the results can be extended to NARMA models using robustness arguments. It is believed that this approach may provide a handle for attacking the stable adaptive control problem of nonlinear plants.

References

[1] Narendra, K. S. and Mukhopadhyay S., "Adaptive Control Using Neural Networks and Approximate Models," IEEE Trans. On Neural Networks, vol. 8, pp. 475-485, 1997.

[2] Saerens, M. and Soquet, A., “A neural controller,” in Proc. 1st Int. Conf. Artificial Neural Networks, London, Oct. 1989, pp. 211-215.

[3] Jordan, M. I. and Jacobs, A., “Learning to control an unstable system with forward modeling,” Advances in Neural Inform. Processing Syst., vol. 2, pp. 324-331, 1990.

[4] Narendra, K. S. and Parthasarathy, K., “Identification and control of dynamical systems using neural networks,” IEEE Trans. Neural Networks, vol. 1, pp. 4-27, 1990.

[5] Levin, A. U. and Narendra, K. S., "Control of nonlinear dynamical systems using neural networks: Part II Identification and Part III Control," IEEE Trans. Neural Networks, vol. 4, Mar. 1993, also IEEE Trans. Neural Networks, vol. 7, Jan. 1996, pp. 30-42; also Center Syst. Sci., Yale Univ., New Haven, CT, Tech. Rep. 9116 and 9117.

[6] Cabrera, J. B. D. and Narendra, K. S., "On regulation and tracking in nonlinear discrete-time systems□Part I: State variables accessible." Center Syst. Sci., Yale Univ., New Haven, CT, Tech. Rep.

[7] Narendra, K. S. and Parthasarathy, K., "Gradient methods for the optimization of dynamical systems containing neural networks," IEEE Trans. Neural Networks, vol. 2, pp. 252-262, Mar. 1991.

[8] Noriega, J. R. and Wang, H., "A direct adaptive neural-network control for unknown nonlinear systems and its application," IEEE Trans. Neural Networks, vol. 9, pp. 27-33, Jan. 1998.

[9] Narendra, K. S. and Annaswamy, A. M. (1989). Stable adaptive systems. Englewood Cliffs, NJ: Prentice-Hall.

[10] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences, USA 79:2554-2558.

[11] Werbos PJ. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. Thesis, Harvard University, Cambridge, MA, 1974. Also published as The Roots of Back-propagation. John Wiley & Sons: New York, 1994.

[12] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature 1986; 323:533–536.

[13] Hagan MT, Demuth HB, Beale MH. Neural Network Design. PWS Publishing: Boston, 1996.

[14] Shanno DF. Recent advances in numerical techniques for large-scale optimization. In Neural Networks for Control, Miller, Sutton and Werbos (eds). MIT Press: Cambridge, MA, 1990.

[15] Scales LE. Introduction to Non-Linear Optimization. Springer-Verlag: New York, 1985.

[16] Charalambous C. Conjugate gradient algorithm for efficient training of artificial neural networks. IEEE Proceedings 1992; 139(3):301–310.

[17] Hagan MT, Menhaj M. Training feedforward networks with the Marquardt algorithm. IEEE Transactions on Neural Networks 1994; 5(6):989–993.

[18] Astrom, K.; Wittenmark, B. Adaptive Control; Dover Publications, Inc.: Mineola, NY, USA ,2008.

[19] Whitaker, H.; Yamron, J.; Kezer, A. Design of Model Reference Adaptive Control Systems for Aircraft (Report R-164); MIT Press Instrumentation Laboratory: Cambridge, MA, USA, 1958.

[20] Osburn, P.; Whitaker, H.; Kezer, A. New Developments in the Design of Model Reference Adaptive Control Systems (Paper No. 61-39); Institute of the Aerospace Sciences: Easton, PA, USA, 1961.

[21] Parks, P. Lyapunov Redesign of Model Reference Adaptive Control Systems. IEEE Trans. Autom. Control 1966, 11, 362–367.

[22] McCulloch, W.; Pitts, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. Bull. Math. Biophys. 1943, 5, 115–133.

[23] Hebb, D. The Organization of Behavior; Wiley: Hoboken, NJ, USA, 1949.

[24] Rosenblatt, F. The Perceptron: A Perceiving and Recognizing Automaton (Report: 85-460-1); Cornell Aeronatical Laboratory: Buffalo, NY, USA, 1957.

[25] Widrow, B.; Hoff, M. Adaptive Switching Circuits. IRE WESCON Convention Record, 1960. pp. 96–104. Available online: http://www-isl.stanford.edu/~widrow/papers/c1960adaptive

[26] Minsky, M.; Papert, S. Perceptrons; MIT Press: Cambridge, MA, USA, 1969.Systems 2014, 2 658

[27] Hopfield, J.J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proc. Nat. Acad. Sci. USA 1982, 79, 2554–2558.

[28] Bryson, A.; Denham, W.; Dreyfus, S. Optimal Programming Problems with Inequality Constraints I: Necessary Conditions for Extremal Solutions. AIAA J. 1963, 1, 2544–2550.

[29] Broomhead, D.; Lowe, D. Multivariable Functional Interpolation and Adaptive Networks. Complex Syst. 1988, 2, 321–355.

[30] Cortes, C.; Vapnik, V. Support-Vector Networks. Mach. Learn. 1995, 20, 273–297.