IDENTIFICATION AND CONTROL OF NONLINEAR DYNAMICAL SYSTEMS USING NEURAL NETWORKS

by
Mehmet Önder Efe
B.S. in Electronics and Telecommunications Eng., İ.T.Ü., 1993

Submitted to Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Systems and Control Engineering

Boğaziçi University
1996

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor Prof. Okyay Kaynak for his endless and motivating support in all phases of my M.Sc. course and this thesis study. He enabled us to use nice facilities of B.U. Mechatronics laboratory.

I am grateful to Assoc. Prof. M. Kemal Cılız and Assoc. Prof. H. Levent Akın for their valuable suggestions and friendly attitudes.

Thanks to Ind. Eng. M. Özgür Efe for his unbelievable help in all phases of this thesis. He motivated me to work hard and provided the substantial technical equipment.

Thanks to Elec. Eng. Ender Abadoğu and Elec. Eng. Mete Erkal for their continuous discussions with me.

I am grateful to my family who illuminated my way with their love and wisdom.

ABSTRACT

Artificial neural networks opened a new horizon in many research areas. This understanding, also, brought a new way of thinking into the concept of control. The ever-increasing technological demands steered the control engineers to design more sophisticated controllers. In this respect, artificial neural networks were proposed as a new approach because of their massively parallel data processing properties, adaptiveness and powerful mapping capabilities. Especially the learning property of these networks made them extremely attractive.

There are various methods that are used for the training of artificial neural networks. Two of them are included in this study. These are, namely, the backpropagation method and the Levenberg-Marquardt optimization technique. The learning time for the former is excessively long especially around the minima since it uses the first order derivatives of the performance function, while in the latter, the learning time is very short because of the extra information coming form the second derivatives of the performance function. The computational complexity and the hardware requirements are large for the latter.

The identification of nonlinear dynamical systems is a substantial part of the controller training therefore it is included in this work and discussed in the simulation results. The main idea that lie under the procedure is obtaining a regular and mathematically tractable model of the system which is of interest.

Based on these two learning methods and concept of system identification, various control strategies are discussed in this work. Design methodology for error backpropagation, inverse control, self-tuning control, model reference adaptive control, self-learning control and dynamical neural unit based control are explained and numerous simulation results are discussed.

ÖZET

Yapay sinir ağları birçok araştırma alanlarında yeni bir ufuk açtı. Bu anlayış, kontrol kavramına da yeni bir düşünüş tarzı getirdi. Artan teknolojik talepler kontrol mühendislerini daha yetenekli denetleyiciler tasarlamaya yöneltti. Bu noktada yeni bir yaklaşım olarak paralel veri işleyebilme özellikleri, uyarlanabilirlik ve güçlü dönüştürme yetenekleri nedeniyle yapay sinir ağları önerildi. Özellike öğrenebilme özelliği bu ağları daha çekici kıldı.

Yapay sinir ağlarının eğitiminde birçok yöntem kullanılmaktadır. Bu çalışmada iki yöntem ele alınmıştır: hata geriye yayma yöntemi ve Levenberg-Marquardt optimizasyon tekniği. Hata geriye yayma yönteminde performans fonksiyonunun birinci türevi kullanıldığından öğrenme zamanı özellikle minimum etrafında çok uzundur, buna karşılık Levenberg-Marquardt yönteminde öğrenme zamanı daha kısadır ve bu, ikinci türevlerden gelen ek bilginin de kullanılmasına atfedilir. Levenberg-Marquardt yönteminde işlem karmaşıklığı ve donanım ihtiyaçları daha fazladır.

Doğrusal olmayan dinamik sistemlerin tanımlanması denetleyici eğitiminde önemli bir bölümdür. Bu yüzden tanıma, bu çalışmanın kapsamına alınmış ve simulasyon sonuçları ile birlikte tartışılmıştır. Prosedürün altında yatan ana düşünce ilgilenilen sisteme ilişkin düzenli ve matematiksel olarak kolay işlenebilir bir modelin elde edilmesidir.

Bu iki eğitim yöntemine ve tanıma kavramına dayanarak çesitli kontrol stratejileri bu çalışma içerisinde tartışılmıştır. Hata geriye yayma ile kontrol, sistem tersini elde etme ile kontrol, kendini ayarlayan kontrol, model referanslı uyarlamalı kontrol, kendi kendine öğrenme ile kontrol ve dinamik sinir ağı modeli ile kontrol yaklaşımları ile denetleyici tasarımı metodolojisi birçok simulasyon sonuçları ile ele alınmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

Page

Page

LIST OF TABLES

LIST OF SYMBOLS

E^k	Performance function defined on kth layer
J	Jacobian matrix for Levenberg-Marquardt algorithm
n_k	The number of neurons in the kth layer
o_j^k	Output of jth neuron in the kth layer
r(t)	Reference signal
u(t)	Control signal
S_j^k	Linear summation output of the jth neuron in the kth layer
t_j	jth entry of the target output vector
w_{ij}^k	Synaptic weight between ith neuron in the kth layer and jth neuron in the (k+1)th layer
α	Momentum coefficient for backpropagation algorithm
γ , β	Learning rate adaptation parameters for backpropagation algorithm
$\Phi(w)$	Approximate error component for Levenberg-Marquardt algorithm
η	Learning rate for backpropagation algorithm
Ψ	Nonlinear activation function
∇	Gradient operator

1. INTRODUCTION

During the last three decades, artificial neural networks have gained a lot importance because of their extensive capabilities in many application domains. Therefore the concept of neurocomputing became an interdisciplinary research area.

The model of a neuron was formed by imitating the biological one. A neuron model was thought of having some inputs from other neurons, and an activation or thresholding operation and an output to other neurons. Actually, this model is compatible with the biological neurons in the central nervous system. Highly complex interconnected structure of these neurons makes the analysis difficult and also makes the brain capable of relating all kinds of data with each other. This is called intelligence. Hopfield [1], a theoretical physicist, says that;

"Simple elements often display complicated behavior when they come in large group" [1].

Examining the nature of learning is reduced to its components and examining their elementary functions. Because the autonomous behavior of the brain gains learning property from billions of interconnected adaptive neurons. The dynamics of biological learning process was also developed for the artificial neural networks. In our study, we elaborated two learning methods, namely, backpropagation method and Levenberg-Marquardt optimization technique. These methods are used in both the identification and the control of nonlinear dynamical systems.

The concept of intelligent control is a primary interest of systems and control area. Technological demands, today, necessitate control of highly complex systems. As the complexity of the system to be controlled increases the need for more sophisticated controllers arises. The lack of precise knowledge about the process, significant parameter changes over time and issues of saturation and long delays make the design of a conventional controller extremely difficult. There are various methods for the control of linear systems but if the system of interest has a nonlinear dynamics, design of a controller requires a highly involved analysis. Using artificial neural networks is a new approach which facilitates the controller design. In some control strategies, the system to be controlled is firstly identified by neural networks then a controller training takes place. The autonomous or self governing properties of the neural networks, enable the controller to compensate the system failures under significant uncertainties without external intervention. Gaining this autonomy requires both algorithmic and numeric methods.

The main problem in designing a controller is that the target outputs of the plant may be known but not the control signals that produce them [2]. Different control strategies such as self-tuning adaptive control, model reference adaptive control, self learning control and control via inversion of the plant and many other methods provide valuable information that constructs the control signal by employing various techniques introduced to the area systems and control.

Our first nonlinear control strategy is the error backpropagation technique. By propagating the output error back through the neural identification model, the error in neural controller output is obtained. This error is then backpropogated through the controller with adjusting the weights and biases. In fact, the neural controller is expected to realize the inverse dynamics of the plant. Provided that the dynamics of the plant is invertible, method works well. Psaltis, Sideris and Yamamura [3] discuss the error backpropagation technique with general and specialized learning architectures.

Inversion of the plant equations can be carried out by the use of state transition knowledge. In this approach, controller is expected to produce a control such that, in response to this control, plant performs the desired state transition. This method is applicable as long as the states of the process are observable and the dynamics of the plant is invertible.

Self-tuning adaptive control is another nonlinear neural control technique considered in this study. The method works on-line and its applicability is limited to single-input-single-output (SISO) and feedback linearizable systems. Based on the past control inputs and the outputs of the plant, controller evaluates a control input such that, plant output follows the desired trajectory. Chen [4] proposed this scheme and he gives some simulation results in his work.

Model reference adaptive control (MRAC) is also considered in this thesis. In MRAC technique, the parameters of the controller are adjusted so that the plant output follows the output of a stable reference model. We applied the method to various types of plant models. The method is explained by Narendra and Parthasarathy [5] in detail. They discuss the concept with example simulation results. In our study, we also duplicated some of their work. Especially for the bioreactor control problem, we constructed a reference model and carried out a detailed analysis parallel to the methodology discussed in [5].

Next, we considered self-learning control scheme. In this scheme, in finite number of steps, the system output is brought to its desired value. No external command signal is used, the parameter adjustment is carried out by the use of error backpropagation technique. Nguyen and Widrow [6] applied the method to a truck backer upper plant. They claim that;

> "Without the learning process, however, substantial amounts of human effort and design time would have been required to devise the controller" [6].

Finally, dynamical neural units were taken into consideration. The approach adopts a new neuron model including a second order pulse transfer function in the synaptic part and a nonlinear activation function with variable slope in the somatic part. Adaptation is carried out on the feedforward, feedback and somatic gains of the neuron model. The neuron model is mathematically tractable for the stability analysis. Gupta and Rao [7] tested the success of this approach for different types of plants. Model uncertainties, noisy observations, and varying input signals were simulated in [7], and the performance of the controller is evaluated.

This study includes a comparison for these control strategies from several performance measures such as tracking performance, applicability to different nonlinear plant models, robustness under perturbations, mathematical tractability for stability analysis, noise reduction and capability of fault tolerance.

2. ARTIFICIAL NEURAL NETWORKS

2.1 Neuron Model

In the development of the artificial neuron models, the main inspiration was the biological neurons which process and communicate information. From an information processing point of view, the biological neuron possesses four main parts that are illustrated in Fig. 2.1.

1. The synapse is the junction point of an axon with a dendrite. Synapses provide memory to the past accumulated experience, that is, knowledge is stored in the synaptic strengths which are called weights.

2. The dendrites receive the information from other neurons. In other words, they carry the information obtained at the neuron side of a synaptic connection to the axon.

3. The soma combines the information come via dendrites. It functions as an evaluation unit, because it collects thousands of information coming from other neurons, and depending upon the current state, it generates a response that is to be sent to other neurons.

4. The axon transmits the response of the neuron to other neurons.

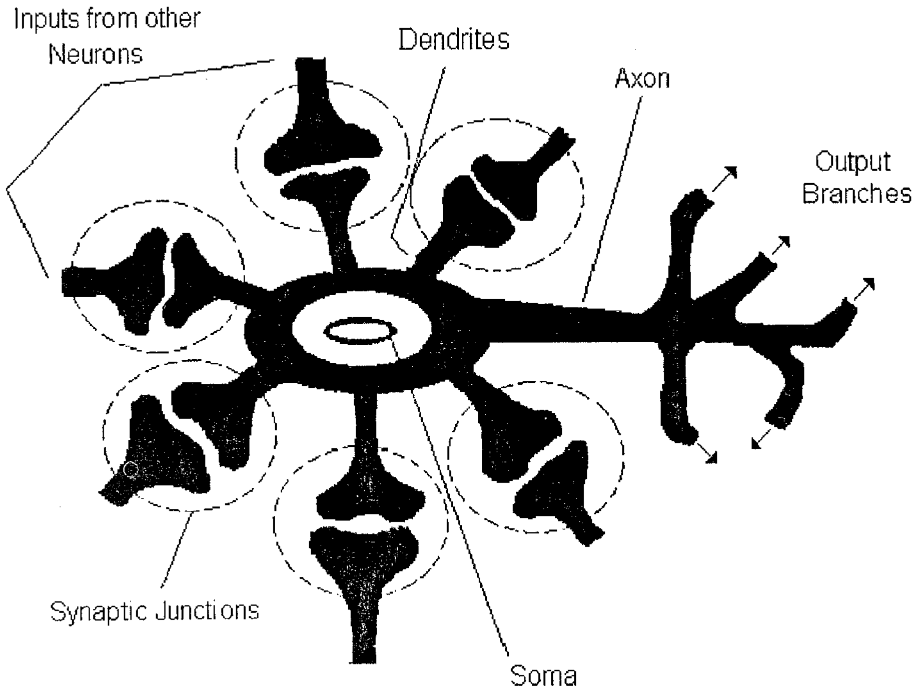


Figure 2.1 A schematic view of the biological neuron body

From a systems theoretic point of view, a neuron can be considered as a multi-input-single-output system which is illustrated in Fig. 2.2.

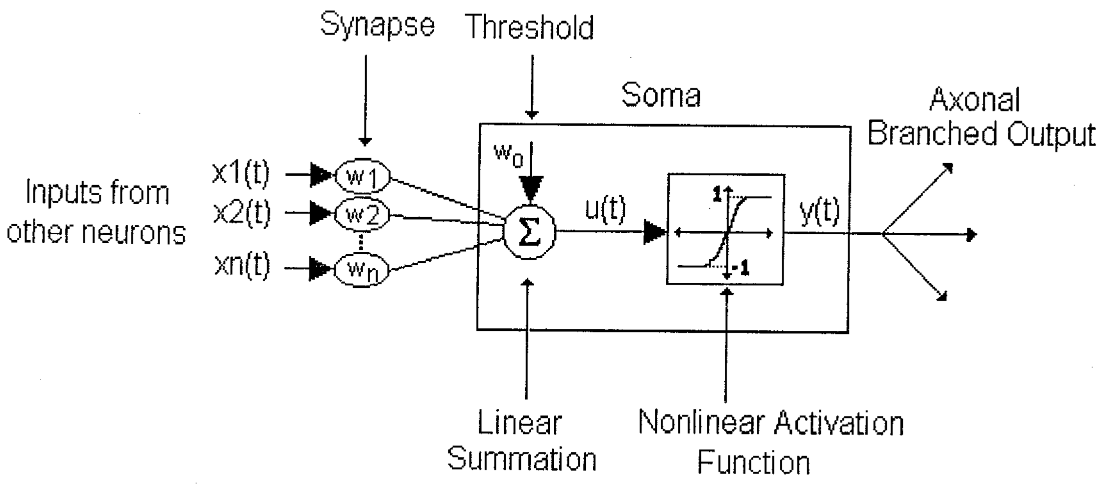


Figure 2.2 Artificial neuron model

As can be seen from the Fig. 2.2, the neuron model performs a weighted sum of n neuronal inputs then adds a bias value to the summation. The resulting value is passed through a nonlinear activation function. The neuron output is then branched for other neurons. The mathematical explanation of a single neuron is as follows;

$$u(t) = w_1x_1(t) + w_2x_2(t) + \dots + w_nx_n(t) + w_0 \tag{2.1}$$

$$y(t) = \Psi\left(\sum_{i=1}^{n} w_ix_i(t) + w_0\right) \tag{2.2}$$

The nonlinear activation function cited in Equation (2.2) is a sigmoidal or hard-limiting function which is used in the vast literature. In Equations (2.3), (2.4) and (2.5), different forms of commonly used nonlinear activation functions are given and in Fig. 2.3, characteristics of these functions are illustrated.

$$\Psi(x) = \frac{1-\exp(-\lambda x)}{1+\exp(-\lambda x)} \tag{2.3}$$

$$\Psi(x) = \frac{1}{1+\exp(-\lambda x)} \tag{2.4}$$

$$\Psi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.5}$$

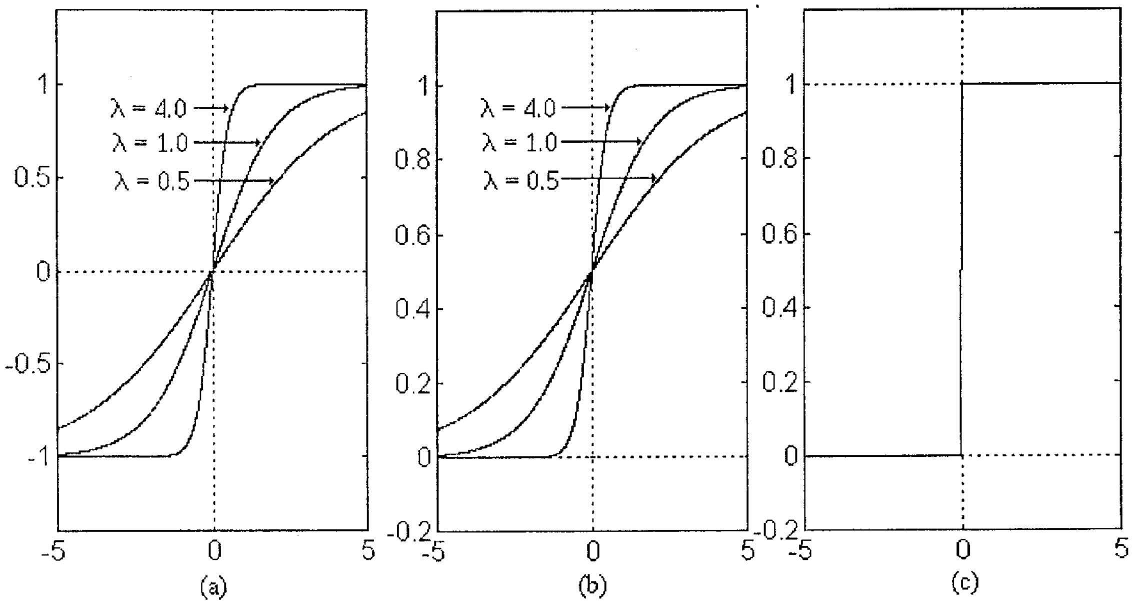


Figure 2.3 Behavior of nonlinear activation function (a) Bipolar, sigmoidal nonlinearity, (b) Unipolar, sigmoidal nonlinearity, (c) Unipolar hard-limiting nonlinearity

The choice of nonlinear activation function is important in examining the behavior of the neuron. The main idea that lies under the choice of these kinds of squashing functions is that at the center of the nonlinearity the derivative is maximum, on the other extreme, if the input to the nonlinearity is large positive or large negative, the derivative tends to zero. This means that, in order to produce a meaningful output the input must be around the center of the nonlinearity, extreme cases result in saturation and this enables the neuron to handle large input values. Thus, the output of the neuron is limited to [-1,1] or [0,1]. In this study, we used the nonlinearity defined in Equation (2.3).

As can be seen easily, the nonlinear activation function will form a hypersurface. The properties of this hypersurface are influenced depending on the type of nonlinear activation function. For the hard-limiting activation function, the hypersurface will exhibit abrupt transitions while for the sigmoidal activation functions it exhibits smooth transitions. Therefore, the former is very sensitive to the small changes around the origin.

This neuron model (which is also called perceptron in the literature) can be used for the basic classification problems. For instance, a single perceptron is able to realize logical OR and logical AND functions but it is unable to solve logical XOR problem. Once the synaptic weights are set to fixed values, in response to any input pattern, the neuron yields a single output value representing the measure of similarity. Nevertheless, it is not so powerful from the point of multidimensional view. In our study, we used multilayer feedforward neural network structures which is illustrated in Fig. 2.4. For convention, we define the terms frequently used in this work.

1. Layer : Neurons are connected to each other with weights in a certain pattern so that, in a layer no neurons have an operational priority. Input vector dimension is fixed for all neurons in the same layer, also, the input vector is applied to all of them simultaneously. Generally, in a layer, all neurons possess the same type of nonlinear activation function. The output vector dimension of a layer is equal to the number of neurons in that layer.

2. Hidden Layer : Hidden layers have the same properties defined above but their input and output vectors are not directly accessible. They take inputs form outputs of preceding layer and their outputs are the inputs of the next layer.

3. Input Layer : Input layer gets the input vector from real world and transmits it to the first hidden layer of the neural network. Sometimes input layer is called fan-in layer.

4. Output Layer : In response to a specific input vector, the neural network produces an output which is carried to real world by means of output layer.

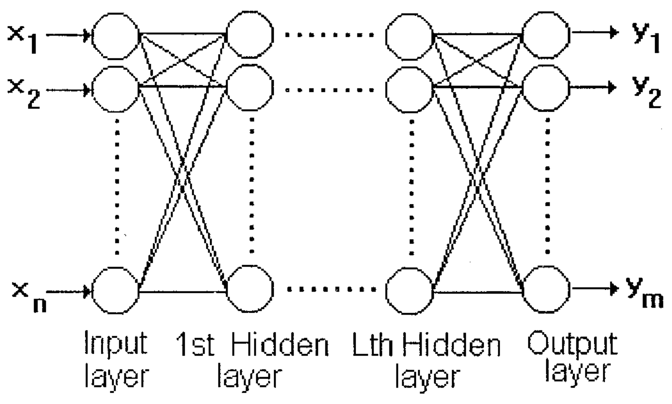


Figure 2.4 Multilayer feedforward neural network

By utilizing a proper method, the weights of the neural network can be adjusted such that the network performs a mapping in a desired sense. This type of learning implies a kind of supervision which provides training patterns obtained from the process inputs and outputs and which generates the necessary information for the weight updating. There are various types of learning methods, and two of them will be used in our study.

The functional capabilities of the multilayer perceptrons (MLP) can be viewed from three different perspectives. Firstly, they have the ability of implementing Boolean logic functions, secondly they can partition the pattern space for classification problems, lastly they can perform highly nonlinear functional approximation problems.

2.2 Approximate Realization via Neural Networks

Parallel, layered structure of the artificial neural networks are responsible for solving formidable problems in the fields such as system identification, control, learning, adaptation and pattern classification. In this section, we give the mathematical preliminary for the mapping capabilities of artificial neural networks. We studied the problem of approximate realization from the point of input-output mappings. In conjunction with these, we shall consider the representation of continuous mappings by means of neural networks whose nonlinear activation functions (squashing or logistic functions) in the hidden layers are sigmoidal.

Let $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ be a vector in the n-dimensional Euclidean space $\mathbf{R}^n$. Then the norm of $\mathbf{x}$ is defined by;

$$\| x \| = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{2.6}$$

Theorem 1

Let $\Psi(x)$ be a nonconstant, bounded, monotone increasing continuous function, and, let K be a compact subset (bounded and closed subset) of $\mathbf{R}^n$ and let $f(x_1, x_2, \dots, x_n)$ be a real valued continuous function on K. Then, for any arbitrary $\varepsilon>0$, there exists an integer N and real constants c_i, θ_i, w_{ij} ($i = 1, 2, \dots, N$ and $j = 1, 2, \dots, n$) such that

$$\hat{f}(x_1,x_2,\dots,x_n) = \sum_{i=1}^{N} c_i \Psi\left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i \right) \tag{2.7}$$

satisfies Equation (2.8).

$$\max_{x \in K} \left\| f(x_1,x_2,\dots,x_n) - \hat{f}(x_1,x_2,\dots,x_n) \right\| < \varepsilon \tag{2.8}$$

In other words, for an arbitrary $\varepsilon>0$, there exists a three-layer network whose output functions for the hidden layer are $\Psi(x)$, whose output functions for input and output layers are linear and which realizes $f(x_1, x_2, \dots, x_n)$ approximately over the range of interest such that Equation (2.8) is satisfied [8]. The proof of this theorem is given in [8].

Theorem 2

Let $\Psi(x)$ be a nonconstant, bounded, monotone increasing continuous function, and, let K be a compact subset (bounded and closed subset) of $\mathbf{R}^n$ and fix an integer $k \geq 3$, then any continuous mapping $f : K \rightarrow R^m$ defined by $x = (x_1, x_2, \dots , x_n) \rightarrow (f_1(x), f_2(x), \dots, f_m(x))$ can be approximated in the sense of uniform topology on K by input-output mappings of k-layer (k-2 hidden layers) networks whose output functions for hidden layers are $\Psi(x)$, and whose output functions for input and output layers are linear [8]. The proof of this theorem is given in [8].

Remark 1

Any mapping is approximately realized by a three layer (one hidden layer) network. However, it should be theoretically noted that $k > 3$-layer networks can realize a given mapping with less costs (number of neurons and connections) than three-layer networks within error ε [8].

Remark 2

The main idea that theorems 1 and 2 imply is that any mapping can be realized perfectly by the use of three-layer neural networks. Hornik [9] points out that this sort of a result has a little practical usefulness, despite its great theoretical utility. Because, the approach necessitates a continuum of hidden neurons. Consequently, using a finite number of hidden neurons puts a limit to the approximation accuracy.

2.3 Lipschitz Condition

Consider a dynamical system which is given in the state space model as described by Equation (2.9) in which **F** is the nonlinearity of the equation set. More explicitly, **F** is a vector of functions that govern the behavior of the dynamical system. The argument of the nonlinearity **F** is an N-by-1 vector, $\mathbf{x} = (x_1 , x_2 , \dots , x_n)^T$, which is called state vector.

$$\frac{d}{dt} x(t) = F[x(t)] \tag{2.9}$$

F(x) is defined to be continuous in all of its arguments. In order to have an unique solution for Equation (2.9) Lipschitz condition must be satisfied. Let the norm of Euclidean length be denoted by the operator || . ||, and let x_1 and x_2 are different vectors from state space, then there exists a constant K such that

$$\| F(x_1) - F(x_2) \| \leq K \| x_1 - x_2 \| \tag{2.10}$$

Provided that **F(x)** satisfies Equation (2.10) then it is said to be Lipschitz, and, all of its partial derivatives are finite everywhere. Also, existence and uniqueness of the solution of state-space equation are ensured. In the neural control theory this condition plays an important role because it gives a valuable knowledge about the stability around the neighborhood of the equilibrium state under some perturbations.

2.4 Controllability and Observability

Control of nonlinear dynamical systems is generally a difficult problem even in the cases that we have a priori knowledge about the nonlinearity or the order of the system. When neural networks are used for the identification and control of these systems, certain assumptions have to be made. Without loss of generality, the governing equations of the system can be given as follows;

$$\begin{aligned} x(k+1) &= f[x(k), u(k)] \\ y(k) &= h[x(k)] \end{aligned} \tag{2.11}$$

where $x \in R^n$, $u \in R^r$ and $y \in R^m$ denote the state, input and output respectively.

Controllability

The main aim in the control action is finding of an input sequence so that the response of the system to this input sequence is the desired behavior. Desired behavior may be one of the following;

1. System is stabilized around an equilibrium point.
2. System tracks a desired trajectory.

Conceptually, controllability states that for some sequence of inputs which is of finite length, the state vector of the system can be moved from its initial position to a target position in $\mathbf{R}^n$. Generally, explaining the controllability of a nonlinear system for the entire state space requires highly involved analysis. Therefore this property is usually restricted to a local region around the point which is of interest. Based on this, Narendra [10] points out that;

"A system is said to be locally controllable around a specific state vector, if for every neighborhood $\aleph$ of this specific state, there is some neighborhood $\wp$ of this state such that for any two states x_1 and x_2 from $\wp$, there exists an input sequence of finite length that will transfer the system from x_1 to x_2 without leaving $\aleph$." [10]

Observability

A dynamical system is said to be observable if an input sequence which is of length l is suffice to determine the time evolution of the state variables uniquely. This requirement is difficult to satisfy even when the system is linear, consequently this global observability definition, in some sense, needs to be relaxed for nonlinear systems such that if an input sequence of length l, where l may be greater than the order of the system, is sufficient to determine the time evolution of the states uniquely, then the system is said to be observable.

Observability is substantial for neural network based controller design because it implies the construction of the state by the use of input and output pairs. What makes it so important, in the sense of neurocontrol, is that this construction is made by the neural network. For any $\varepsilon>0$,

$$\| \vec{x}(k) - NN_f[\vec{y}(k), \vec{u}(k)] \| < \varepsilon \tag{2.12}$$

where the dynamical system is described by Equation (2.11). It should be noted that if the system is Lipschitz, its neural representation is obtained easily by the use of powerful learning algorithms. In the next chapter, we shall elaborate two learning methods which is widely used in the literature.

3. LEARNING ALGORITHMS

As mentioned in the previous chapter, neural networks are able to perform complex nonlinear mappings under certain conditions. Neural networks gain this property through learning. Learning is the key step in the neural network based implementations. The parameters of the neural network are adjusted such that a previously defined performance function is minimized. Here, parameters are the synaptic weights and biases of all individual neurons. Conceptually, neural network learning can be divided into two main categories [5];

1. Supervised learning
2. Unsupervised learning

In supervised learning, desired response to a specific input vector is available. As a consequence of this, the algorithm tries to relate all presented input vectors to their desired output vectors. The main problem in this approach is the choice of the training data set. Because, the representative adequacy of the neural network is proportional to how well the training data set represents the behavior of the actual process.

In unsupervised learning, there is no external supervision providing the desired input-output pairs to the neural network. In this approach, the neural network discovers the patterns, associations and regularity in the data. The network response means that how similar the input pattern is to average patterns seen in the past.

In this work, we elaborated two learning methods that are commonly used in the neural network applications.

3.1 Backpropagation Learning Algorithm

The backpropagation algorithm is the most popular training method which is widely used in the neural network applications [5]. The method is applicable to the multilayer perceptron and it minimizes a performance or cost function defined on the actual and desired outputs of the network. The choice of this kind of a performance function stems from the fact that, first, it should represent each training pair's contribution to the total output error, second, it must be differentiable. In updating of each individual weight, the gradient information obtained from the differentiation of the cost function is used. As a matter of fact, we are looking for least mean squares. This can be attained by moving the weight vector in a direction that the performance function decreases. It is obvious that this direction is the negative gradient direction of the performance function.

$$E = \frac{1}{2} \sum_{j=1}^{n_{k+1}} (t_j - o_j^{k+1})^2 \qquad (3.1)$$

Assuming that the objective is to minimize this kind of a performance function, the weight updating rule is given by Equation (3.2)

$$w(t+1) = w(t) - \eta \nabla E \qquad (3.2)$$

As can be seen from Equation (3.2), the weight updating is performed by evaluating the gradient of the performance function with respect to each individual weight in the network. By convention, superscripts denote layer order, w_{ij}^k denotes the weight between ith neuron of kth layer and jth neuron of (k+1)th layer, o_j^k denotes the output of the jth neuron in the kth layer and S_j^k denotes the linear summation output of the jth neuron in the kth layer.

Assume that (k+1)th layer is the output layer, the derivative of the performance function with respect to a weight belonging to outmost weight matrix can be evaluated easily.

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial o_j^{k+1}} \frac{\partial o_j^{k+1}}{\partial w_{ij}^k} \qquad (3.3)$$

Since Equation (3.4) holds for overall network, the chain rule can be applied once more.

$$o_j^{k+1} = \Psi(S_j^{k+1}) \qquad (3.4)$$

The first term of Equation (3.5) is evaluated from Equation (3.1), the second term in Equation (3.5) is differentiation of the nonlinear activation function with respect to its argument, and the last term is differentiation of a linear summation with respect to a certain weight.

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial o_j^{k+1}} \frac{\partial o_j^{k+1}}{\partial S_j^{k+1}} \frac{\partial S_j^{k+1}}{\partial w_{ij}^k} \qquad (3.5)$$

$$\frac{\partial E}{\partial o_j^{k+1}} = -(t_j - o_j^{k+1}) \qquad (3.6)$$

$$\frac{\partial o_j^{k+1}}{\partial S_j^{k+1}} = \frac{d\,\Psi(S_j^{k+1})}{d\,S_j^{k+1}} = \Psi'(S_j^{k+1}) \qquad (3.7)$$

Equation (3.8) represents the combination of Equations (3.6) and (3.7).

$$\frac{\partial E}{\partial S_j^{k+1}} = -(t_j - o_j^{k+1})\Psi'(S_j^{k+1}) \qquad (3.8)$$

As mentioned earlier, the function of linear summation is given by Equation (3.9), consequently, the derivative with respect to a certain weight will result in its multiplier which is a neuron output comes from the kth layer. This is given in Equation (3.10);

$$S_j^{k+1} = \sum_{i=1}^{n_k} w_{ij}^k\, o_i^k \qquad (3.9)$$

$$\frac{\partial S_j^{k+1}}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_{i=1}^{n_k} w_{ij}^k\, o_i^k\right) = o_i^k \qquad (3.10)$$

Thus, the change in weight vector is stated as follows;

$$\Delta w_{ij}^k = -\eta \nabla E \qquad (3.11)$$

For a weight value in the outmost weight matrix, the update rule is given by Equation (3.12)

$$\Delta w_{ij}^k = \eta\,(t_j - o_j^{k+1})\,\Psi'(S_j^{k+1})\,o_i^k \qquad (3.12)$$

It must be designated that the derivation which has been carried out so far is only for the output layer. What made the derivation of Equation (3.12) easy is the availableness of the target or desired outputs. Therefore, evaluation of the error is easy. For the hidden neurons desired outputs are not available, so the error measure must be transformed into a tractable form for these neurons.

Let's define (k+1)th layer as a hidden layer. The gradient of the performance function by employing the chain rule is stated in Equation (3.13). Evaluation of the second term in Equation (3.13) is given by Equation (3.10).

$$\nabla E^{k+1} = \frac{\partial E^{k+1}}{\partial w_{ij}^{k}} = \frac{\partial E^{k+1}}{\partial S_j^{k+1}} \frac{\partial S_j^{k+1}}{\partial w_{ij}^{k}} \qquad (3.13)$$

We know from the previous derivation, the first term can be expanded once more. The evaluation of the second term in Equation (3.14) is given by Equation (3.7)

$$\frac{\partial E^{k+1}}{\partial S_j^{k+1}} = \frac{\partial E^{k+1}}{\partial o_j^{k+1}} \frac{\partial o_j^{k+1}}{\partial S_j^{k+1}} \qquad (3.14)$$

Since the hidden neuron output is connected to all neurons in the output layer, obviously, the change in its output will affect the network outputs. In conjunction with this, the value of the performance function will be affected from this change. Therefore, the first term in Equation (3.14) will be a summation of derivatives which must be evaluated from (k+2)th layer back through the (k+1)th layer by utilizing an additional chain rule. Equation (3.15) represents this case.

$$\frac{\partial E^{k+1}}{\partial o_j^{k+1}} = \sum_{h=1}^{n_{k+2}} \frac{\partial E^{k+2}}{\partial S_h^{k+2}} \frac{\partial S_h^{k+2}}{\partial o_j^{k+1}} \qquad (3.15)$$

If the linear summation is written in open form, differentiation ends up with Equation (3.17)

$$\frac{\partial E^{k+1}}{\partial o_j^{k+1}} = \sum_{h=1}^{n_{k+2}} \left[\frac{\partial E^{k+2}}{\partial S_h^{k+2}} \frac{\partial}{\partial o_j^{k+1}} \left(\sum_{j=1}^{n_{k+1}} w_{jh}^{k+1} o_j^{k+1} \right) \right] \qquad (3.16)$$

$$\frac{\partial E^{k+1}}{\partial o_j^{k+1}} = \sum_{h=1}^{n_{k+2}} \frac{\partial E^{k+2}}{\partial S_h^{k+2}} w_{jh}^{k+1} \qquad (3.17)$$

If we define delta terms as stated in Equation (3.18), Equation (3.15) is rewritten as Equation (3.19)

$$\delta_j^{k+1} = -\frac{\partial E^{k+1}}{\partial S_j^{k+1}} \qquad (3.18)$$

$$\frac{\partial E^{k+1}}{\partial o_j^{k+1}} = -\sum_{h=1}^{n_{k+2}} \delta_h^{k+2} w_{jh}^{k+1} \tag{3.19}$$

If the results are combined together, the derivative of the performance function is obtained as follows;

$$\frac{\partial E^{k+1}}{\partial w_{ij}^{k}} = \frac{\partial E^{k+1}}{\partial S_j^{k+1}} \frac{\partial S_j^{k+1}}{\partial w_{ij}^{k}} = \left(-\sum_{h=1}^{n_{k+2}} \delta_h^{k+2} w_{jh}^{k+1}\right) \Psi'(S_j^{k+1})\, o_i^k \tag{3.20}$$

The weight change for the hidden layer neurons is given by Equation (3.21)

$$\Delta w_{ij}^{k} = -\eta \nabla E^{k+1} \tag{3.21}$$

If the gradient information given in Equation (3.20) is substituted into Equation (3.21), the general weight update rule for the overall network can be given by Equation (3.23).

$$\Delta w_{ij}^{k} = -\eta\,(-\delta_j^{k+1})\, o_i^k \tag{3.22}$$

$$\Delta w_{ij}^{k} = \eta\, \delta_j^{k+1}\, o_i^k \tag{3.23}$$

The delta values for the output layer is evaluated by using Equation (3.24).

$$\delta_j^{k+1} = (t_j - o_j^{k+1})\, \Psi'(S_j^{k+1}) \tag{3.24}$$

On the other hand, the delta values for the hidden layer neurons are evaluated using Equation (3.25).

$$\delta_j^{k+1} = \left(\sum_{h=1}^{n_{k+2}} \delta_h^{k+2} w_{jh}^{k+1}\right) \Psi'(S_j^{k+1}) \tag{3.25}$$

In the derivation of the algorithm, we used a scaling factor η, which is generally called the learning rate or convergence rate. This variable determines the step size and usually chosen between zero and one.

The name of the algorithm comes from the operation itself, because the error measure is evaluated at the output neurons and then propagated back through the network. The algorithm works iteratively, and at each iteration, it tries to improve the performance of the network. Algorithmically, the method can be summarized as follows;

1. Set all the network parameters (weights and biases) to small random values.
2. Set epoch counter to 1.
3. Set epoch error to 0.
4. Set sample error to 0.
5. Present a pattern.
6. Evaluate the outputs and the delta values attached to output neurons.
7. Evaluate the sample error, epoch error = epoch error + sample error
8. Backpropagate the delta values.
9. Update weights and biases.
10. If all patterns are presented then continue
    else go to step 4.
11. If the convergence criterion is satisfied then continue
    else increase epoch counter by one, go to step 3
12. Test for completion.

In the implementation level, convergence criterion may be one of the followings:

1. The epoch error, which is the cumulative error measure over one pass of entire training set, decreases under a previously defined error level.
2. The iterative procedure terminates until a previously defined number of epochs reached.

This study adopts the first type of convergence criterion.

3.2 Effect of Momentum Term

In the previous section, we derived the weight updating equations of the backpropagation algorithm. In some cases where the cost function exhibits narrow and steepest valleys that have small floor slope, backpropagation method shows an oscillatory behavior. Since it utilizes only the first order derivatives of the performance function, it has a tendency to follow a zigzag shaped trajectory on the hypersurface which is defined by the cost function.

The momentum term is the most basic method to improve the shape of the trajectory. The weight update rule which is given in Equation (3.23) is modified by adding a fraction of previous parameter change vector.

$$\Delta w_{ij}^{k}(t) = \eta \delta_j^{k+1} o_i^k + \alpha \Delta w_{ij}^{k}(t-1) \tag{3.26}$$

Basically, this term damps the instantaneous fluctuations. Long term component becomes dominant during the time evolution of the algorithm. Moreover, in many cases, momentum term prevents getting stuck to a local minimum.

3.3 Effect of Learning Rate Adaptation

The most important drawback of the backpropagation learning is its slow convergence property. Since it employs the first derivatives, the path to be followed on the cost function is determined by the negative gradient direction in the weight space. Additionally, the backpropagation error surface is generally flat along a weight direction. Therefore the instantaneous gradient information will be small in magnitude. Consequently, to reach the desired performance, the algorithm will need to carry out many iterations. If the weight vector is near to the global minimum, the weight change vector will have small values in magnitude, therefore, as the time evolution progresses speed will decrease logarithmically.

The most useful method to handle these difficulties is adapting the learning rate by comparing the instantaneous epoch error with previous epoch error, and giving a change to the learning rate. This is given by Equation (3.27).

$$\Delta \eta = \begin{cases} +\gamma & \Delta E < 0 \\ -\beta\eta & \Delta E > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.27}$$

This last equation simply states that, if a decrease is occurred in the cost function then the magnitudes in weight change vector decrease, in order to speed up the convergence increase learning rate by adding a constant γ. If the cost function is increased through several weight changes in the past iterations this means that the learning rate is so large that the trajectory tends to overshoot the minima. In conjunction with this, more precise steps are needed to be taken and learning rate should be decreased by $(1-\beta)$.

3.4 Levenberg-Marquardt Learning Algorithm

Levenberg-Marquardt method is an approximation to Newton's method [11]. Newton's original approach assumes that a function E(**w**) is minimized if the successive changes defined by Equation (3.28) are given to the parameter vector **w**.

$$\Delta w = -(\nabla^2 E(w))^{-1} \nabla E(w) \tag{3.28}$$

We assume the performance function is defined as follows;

$$E(w) = \frac{1}{2} \sum_{q=1}^{Q} (t_q - o_q)^T (t_q - o_q) \tag{3.29}$$

In Equation (3.29) q indexes the training pairs, and Q is the number of training pairs, t denotes the target output and o denotes the actual output of the network. If Equation (3.29) is rewritten with respect to the output error which depends on the weight vector, we obtain Equation (3.30).

$$E(w) = \frac{1}{2} \sum_{q=1}^{Q} e_q^T(w) e_q(w) \tag{3.30}$$

The objective is to minimize each individual multiplication in Equation (3.30). If Taylor series expansion is applied to $e_q(w)$ around w_0;

$$e_q(w) \approx \hat{e}_q(w) = e_q(w_0) + J^T(w - w_0) \tag{3.31}$$

In Equation (3.31) J is the Jacobian matrix and evaluated at w_0. The entries of this matrix represent the derivative of the error evaluated at the ith output with respect to the jth parameter of the parameter vector. And, this statement obviously implies that the number of rows of the Jacobian matrix is equal to the multiplication of the number of network outputs and the number of the training pairs. Similarly, the parameter vector will have an entry for all weights and biases of the network.

$$J_{ij} = \frac{\partial e_{q_i}(w)}{\partial w_j} \tag{3.32}$$

We define an approximate error component as described in Equation (3.33). Differentiating this equation with respect to parameter vector **w** and equating zero gives the usual "normal equation" of the linear least squares problem.

$$\phi(w) = \hat{e}_q^T(w)\,\hat{e}_q(w) \tag{3.33}$$

$$J^T J(w - w_0) + J^T e_q(w) = 0 \tag{3.34}$$

From Equation (3.34), the change in parameter vector turns out to be Equation (3.35).

$$\Delta w = -(J^T J)^{-1} J^T e_q(w) \tag{3.35}$$

The terms appear in Equation (3.35) are explained as follows;

$$\nabla E(w) = J^T e_q(w) \tag{3.36}$$

$$\nabla^2 E(w) = J^T J \tag{3.37}$$

Equation (3.36) represents the first derivative of the performance function, Equation (3.37) represents the second derivative of the performance function and is called Hessian matrix. Levenberg and Marquardt modified Equation (3.35) by adding an extra term to the Hessian matrix. The resulting parameter update rule is then introduced to be;

$$\Delta w = -(\, J^T J + \mu\, I\,)^{-1} J^T e_q(w) \tag{3.38}$$

In fact, the reason for this modification is that this additional term compensates the approximation errors. Actually, the method seems to minimize Φ but this may not always imply that E(w) is minimized. Therefore, a scaling is introduced to Hessian matrix evaluation part of the method. If a step reduces E(w) then μ is decreased, otherwise μ is increased by some factor greater than one. Note that, if μ is large, the method becomes steepest descent because the modification dominates the term $J^T J$ term and only the first order derivative information remains, on the other extreme, if μ is too small then the method becomes pure Gauss-Newton method. Therefore the method is considered as a trust region modification to Gauss-Newton [11]. The key step in the algorithm is the evaluation of the Jacobian matrix. Below, we explain the evaluation of this matrix for the multilayer perceptron case.

1. The number of rows of the Jacobian is equal to QxN where Q denotes the number of training pairs and N denotes the number of network outputs. Therefore, the first N rows correspond to the evaluated derivatives for the first training pattern. For each training pattern such a submatrix is formed.

2 The number of columns of the Jacobian is equal to number of parameters of the network. These are the weights and biases.

3. For the entries corresponding to a weight w_{ht}^{k}, Equation (3.39) defines the evaluation of that entry. The entries that correspond to kth layer's hth neuron bias are evaluated by using Equation (3.40).

$$J_{ij} = \delta_h^k o_t^{k-1} \tag{3.39}$$

$$J_{ij} = \delta_h^k \tag{3.40}$$

4. Equation (3.39) and Equation (3.40) imply that the classical error backpropagation is utilized. Since each output's derivatives appear in different rows of the Jacobian, in backpropagation of the error vector, only the delta value that is of interest is backpropagated, others are temporarily set to zero and backpropagated next.

We considered this method as the second method for neural network based controller training. Below, the application to MLP is explained procedurally.

1. Set all weights and biases to small random values.
2. Set the μ and β. (In our tests we chosen $\mu = 0.01$ $\beta = 10$)
3. Present an input pattern to the network.
4. Evaluate delta values and compute N rows of the Jacobian
5. If all inputs are presented then continue
   else go to step 3.
6. Solve the Equation (3.38) obtain the changes in the parameter vector
7. Pass an epoch by using new parameter vector
8. If new sum squared error decreased then reduce μ by β, continue
   else cancel the last update and increase μ by β go back to step 6.
9. If the algorithm converged then stop
   else go back to step 3.

3.5 Results and Comparison

We tested both of the algorithms for many types of input-output relations. In order to make an assessment on the performance of these algorithms, we included the XOR problem and a function approximation problem.

The simulation results presented in this thesis were carried out by the use of a 486DX4-100 computer and all of the source codes were developed in Turbo C 3.1 environment.

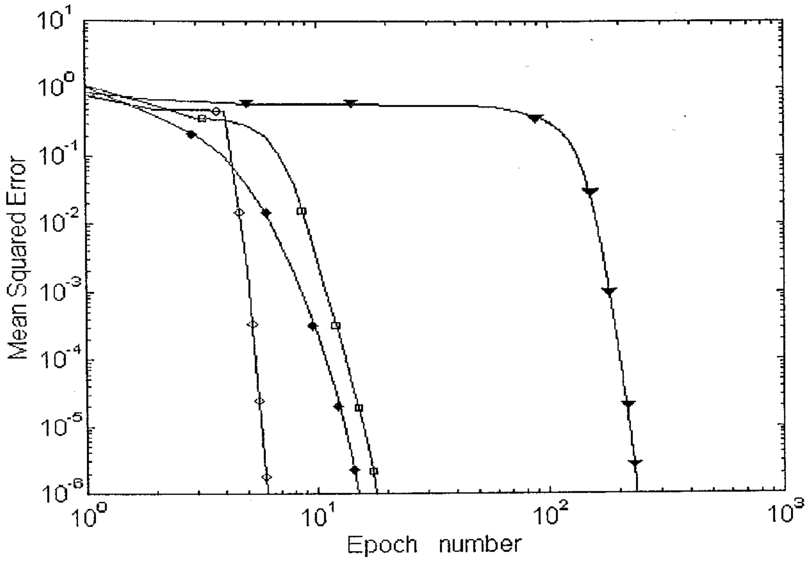
In Figures 3.1 and 3.2, mean squared error trends are given for backpropagation, backpropagation with momentum, backpropagation with momentum and adaptive learning rate and for the Levenberg-Marquardt methods.

In backpropagation simulations, for the XOR problem 2-4-1 network with linear output neuron was used with the parameters $\eta = 0.1$, momentum = 0.7, $\beta = 0.1$ and $\gamma = 0.001$; for the function approximation problem 1-4-1 network with linear output neuron was used with the same parameters. In the Levenberg-Marquardt training simulations, same network structures were used with the parameters $\beta = 10$ and $\mu = 0.01$.

Table 3.1 describes the input-output relation of logical XOR function.

Table 3.1 XOR Function

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

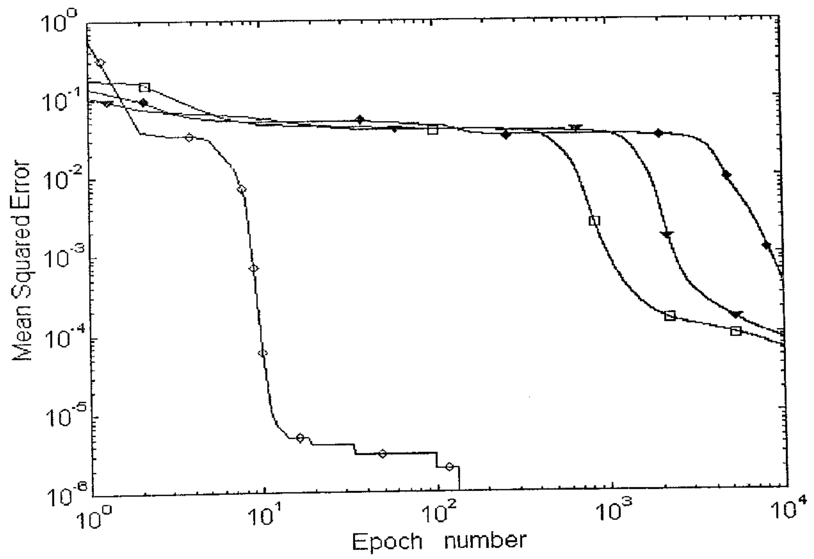


◇ Levenberg - Marquardt Method
▼ Backpropagation Method
□ Backpropagation with Momentum
◆ Backpropagation with Momentum and Adaptive Learning Rate

Figure 3.1 Mean squared error graph for XOR problem

The second test was approximation of a function described by Eqn. (3.41), and the MSE graph is illustrated in Fig. 3.2. In this problem 20 patterns were used as the training set.

$$f(x) = \frac{1+\sin(2\pi x)}{2} \qquad [0 \le x \le 1] \tag{3.41}$$



◇ Levenberg - Marquardt Method
▼ Backpropagation Method
□ Backpropagation with Momentum
◆ Backpropagation with Momentum and Adaptive Learning Rate

Figure 3.2 Mean squared error graph for function approximation problem

As can be seen from the error trends of the cited problems, momentum term considerably improves the performance for the backpropagation algorithm. In the XOR problem, adaptive learning rate sped up the convergence while in the function approximation it resulted in some fluctuations in the level of MSE. Nevertheless, the varieties introduced to the classical backpropagation algorithm did not make it as fast as the Levenberg-Marquardt method.

This obvious result stems from the fact that, the Levenberg-Marquardt numerical optimization technique utilizes the second order partial derivatives of the cost function whereas the classical backpropagation or simple gradient descent rule employs only the first order partial derivatives. More explicitly, in the latter, the trajectory to be followed on the cost surface simply states that moving along the negative gradient direction with respect to the parameter vector. Therefore, the successive parameter updates may follow a zigzag shaped or spiral shaped trajectory and this slows down the convergence. In contrast to this, Levenberg-Marquardt method finds the best parameter change vector that does not infringe the previously taken steps. Because second derivatives provide extra information about the cost surface and this results in the fast convergence property.

Another comparison metric is the computational complexity and hardware requirements of the above mentioned methods. From the derivation of these algorithms, one may easily see that the memory requirement of the Levenberg-Marquardt method is larger than the backpropagation method. In the former the dimension of the problem exponentially increases as the number of parameters (weights and biases) increases. This stems from the fact that the number of columns of the Jacobian is equal to the number of parameters. Therefore the complexity of Levenberg-Marquardt method can be expressed as $O(n^2)$. Consequently, when the matrix inversion is needed the number of computations becomes proportional to the dimensions of the matrix. Similarly, this requires more memory area to store the Jacobian and the matrix which is to be inverted. In the backpropagation method, memory requirement is relatively low with respect to the Levenberg-Marquardt method. The necessary memory area increases linearly as the number of parameters of the network increases. The complexity of backpropagation training method is expressed as $O(n)$.

As a result, the Levenberg-Marquardt method performs a precise approximation but it necessitates more memory and more computations, while the backpropagation method necessitates less memory but it takes too long time to reach the same MSE level when it is compared with the Levenberg-Marquardt method.

4. SYSTEM IDENTIFICATION USING NEURAL NETWORKS

Control of systems requires representative knowledge about the system to be controlled. In many applications, model uncertainty arises as a problem that control engineers frequently encounter. Therefore, the representative adequacy of the model should have as many properties as the actual plant has. What make the modeling so difficult are the nonlinearity, time delay, saturation and time varying parameters of the actual plant. Therefore, including these kinds of challenging difficulties in a mathematically tractable model is a formidable problem.

Artificial neural networks opened a new horizon in identification and control of highly nonlinear and complex structured systems. These networks are implemented using massive connections among the neurons with variable strengths. Moreover, their parallel, distributed and fault tolerant processing properties make them powerful tools for both identification and control of nonlinear dynamical systems. Especially learning capabilities of these networks enable them to process the information adaptively.

This chapter presents a brief mathematical background for neural network based identification of systems. Then the simulation results are given.

4.1 Concept of Identification

The system that is to be identified can be represented by a transformation operator T_p, which maps the compact subset $U \in \mathbf{R}^n$ to $Y \in \mathbf{R}^m$. The purpose is to find a class T_i such that T_p is represented by T_i adequately well. The operator T_p is defined by specific input-output pairs that are obtained form the inputs and the outputs of the system to be identified. The objective is expressed as follows;

$$\left\| T_i(u) - T_p(u) \right\| \leq \varepsilon \quad , u \in U \tag{4.1}$$

for some desired $\varepsilon>0$. $T_i(u)$ denotes the identification model output. As can be seen easily, the approach requires the input-state-output representation of the system. Generally, a continuous time dynamical system can be given by;

$$\begin{aligned} \frac{d\,x(t)}{d\,t} &= \dot{x}(t) = \Phi[x(t), u(t)] \quad t \in R^+ \\ y(t) &= \Psi[x(t)] \end{aligned} \tag{4.2}$$

where, $x(t) = [x_1(t)\ x_2(t)\ ...\ x_n(t)]^T$, $u(t) = [u_1(t)\ u_2(t)\ ...\ u_p(t)]^T$, $y(t) = [y_1(t)\ y_2(t)\ ...\ y_m(t)]^T$ denoting state, input and output vectors respectively. In discrete domain, Equation (4.2) becomes;

$$\begin{aligned} x(k+1) &= \Phi[x(k), u(k)] \\ y(k) &= \Psi[x(k)] \end{aligned} \tag{4.3}$$

Even in the cases where Φ and Ψ are not known, neural networks can construct an approximate model which when the same input vector is applied to both the actual plant and the identification model, the difference between the outputs remains within a predefined error level. In Fig. 4.1, system identification structure is illustrated.

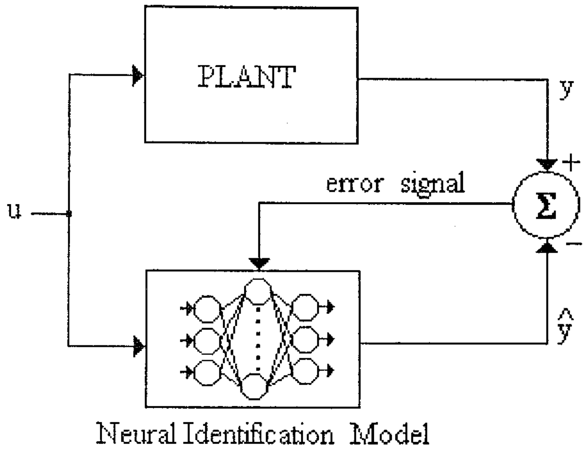


Figure 4.1 System Identification Structure

The emphasis on the neural network based identification is determination of an adaptive algorithm that minimizes the difference between the actual plant and the identification model outputs by using a set of training pairs which represent the approximate behavior of the actual plant.

We simulated many identification problems. In order to convey a concrete idea, we give some of the simulation outputs in the section 4.3.

There are some widely used identification models which are mentioned in [5].

Model 1:

$$y_p(k+1) = \sum_{i=0}^{n-1} \alpha_i \, y_p(k-i) + g[u(k), u(k-1), \ldots, u(k-m+1)] \tag{4.4}$$

Model 2:

$$y_p(k+1) = f[y_p(k), y_p(k-1), \ldots, y_p(k-n+1)] + \sum_{i=0}^{n-1} \beta_i \, u(k-i) \tag{4.5}$$

Model 3:

$$y_p(k+1) = f[y_p(k), y_p(k-1), \ldots, y_p(k-n+1)] + g[u(k), u(k-1), \ldots, u(k-m+1)] \tag{4.6}$$

Model 4:

$$y_p(k+1) = f[y_p(k), y_p(k-1), \ldots, y_p(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)] \tag{4.7}$$

In all of these models, the plant under consideration is a SISO plant whose response to permissible inputs is assumed to be bounded. In the first model, the output of the plant is linearly dependent on its past n values. Since the model chosen for the plant has to be stable, this directly implies that the characteristic equation has the roots that lie within the unit circle. In the second model, the output is linearly dependent on past input values. The third and the fourth models can represent highly nonlinear dependencies to past outputs and past inputs. In [5], the fourth model is said to be analytically least tractable. The functions f(.) and g(.) in these models are assumed to be differentiable with respect to their arguments.

4.2 Training and Overtraining in Identification

The function of neural networks in identification is representation of a process adequately. In this respect, two questions arise: how well a process can be represented by neural networks and what are the conditions for accurate representation. If we were able to train the network by utilizing infinite number of training samples that are obtained from a time-invariant process, and if we were able to wait for zero epoch error, one would easily say that the resulting network has the same dynamics as the actual process possesses. But fulfillment of these two conditions is practically impossible. Therefore, we choose a set of training pairs and we terminate the training process at an admissible level of epoch error.

Two important cases must be taken into consideration. Firstly, if the training set includes all possible input vectors, just as in the case of XOR problem, it is desired to terminate the training with zero epoch error. Obviously, this conclusion is valid for discrete or Boolean functions' realization. On the other hand, the second case concerns the continuous function approximation. Since the function is continuous, we may have limited number of training pairs and the neural network interpolates a surface passing through these training data points. If the identification procedure terminates with zero epoch error, it is guaranteed that all of the training data points are on the interpolated surface.

The main emphasis on the neuronal morphology is performing a generalization. If an input vector is similar to another vector which is in the training data set, the response to this vector should be near to the response to the vector in the training data set. More explicitly, similar inputs should cause similar outputs. This is the main idea that lie behind the concept of generalization.

At the beginning of this discussion we mentioned about an admissible level of epoch error. If the network is trying to identify a process, the network should be tested with another set of data obtained from the same process. The elements of this new set do not intersect the elements of the training data set. This obviously prevents the memorization of the training data set. If the performance of the network is compared by using these two sets, after a while, the network begins to memorize training data set and its performance decreases for the test set. This instant is the proper instant for the termination for the identification procedure.

Another method for preventing the memorization problem is regularly changing the training data set. After a predefined number of epochs passed, training data set is regenerated. In this case, the period for refreshing the training data set is chosen experimentally.

In our simulations, especially for the *bioreactor benchmark problem* which is explained in detail in Appendix A, we used the last technique. The identification results are given in the next section. In the bioreactor identification problem we regenerated 670 training pairs for every four epochs, and we observed a highly precise approximation.

Briefly, limited number of exemplar patterns generally can not represent the dynamics of a continuous process perfectly. Therefore, this may lead on the network to memorize only a part of the actual dynamics. This can be prevented by the use of a test set and stopping the learning by considering the performance to this test set or by regularly changing the training data set.

4.3 Identification Results and Performance Assessment

Plant Type : Model 1

Governing equation : $y(k+1) = 0.3\,y(k) + 0.6\,y(k-1) + \tanh(u(k))$ (4.8)

Input signal : $u(k) = \sin\left(\dfrac{2\pi k}{160}\right)$ (4.9)

Network Structure : 3-6-1, with linear output neurons

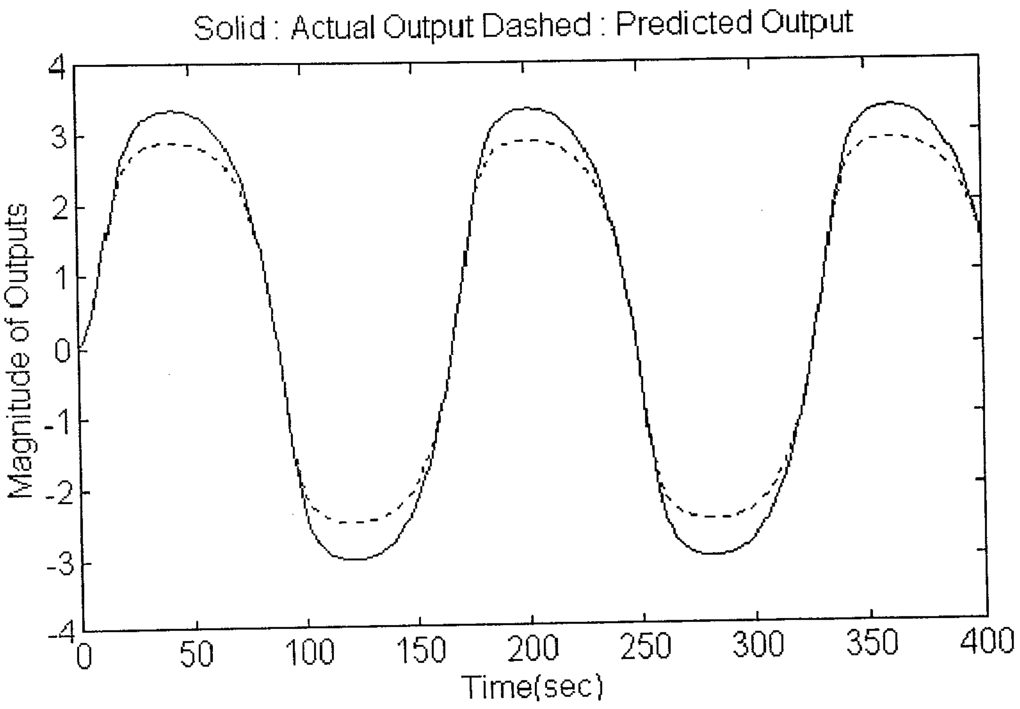


Figure 4.2(a) Actual and predicted outputs for the plant defined by Equation (4.8)

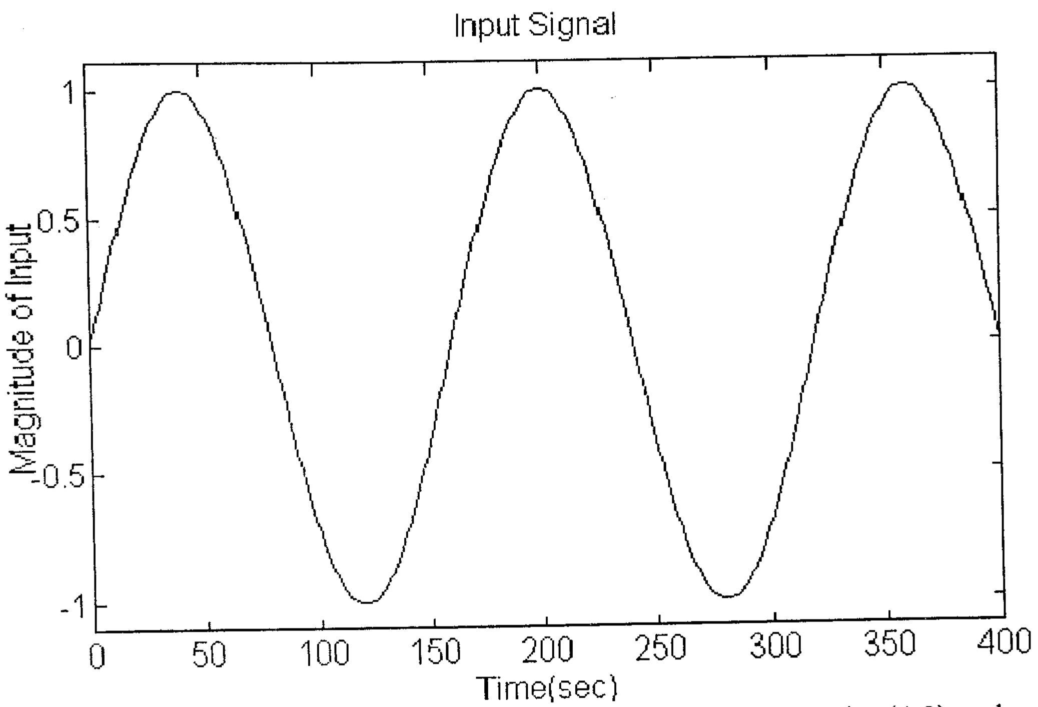


Figure 4.2(b) Applied input signal to the plant defined by Equation(4.8) and identification model

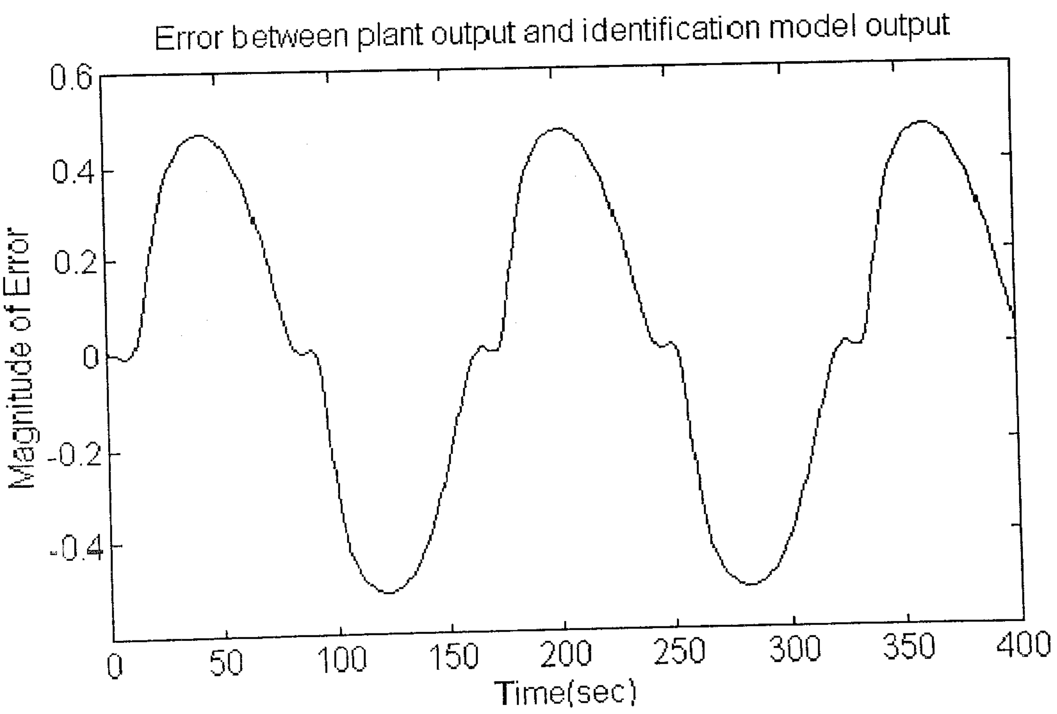


Figure 4.2(c) Error between actual and predicted outputs for the plant defined by Equation (4.8)

Plant Type : Model 2

Governing equation :
$$y(k+1) = \frac{\tanh(0.3\,y(k) + 0.6\,y(k-1) - 0.1\,y(k-2))}{1 + y^2(k)} + 0.3\,u(k) - 0.6\,u(k-1) + 0.1\,u(k-2) \qquad (4.10)$$

Input signal :
$$u(k) = \frac{1}{2}\sin\left(\frac{2\pi k}{150}\right)\sin\left(\frac{2\pi k}{40}\right) \qquad (4.11)$$

Network Structure : 6-20-10-1, with linear output neuron

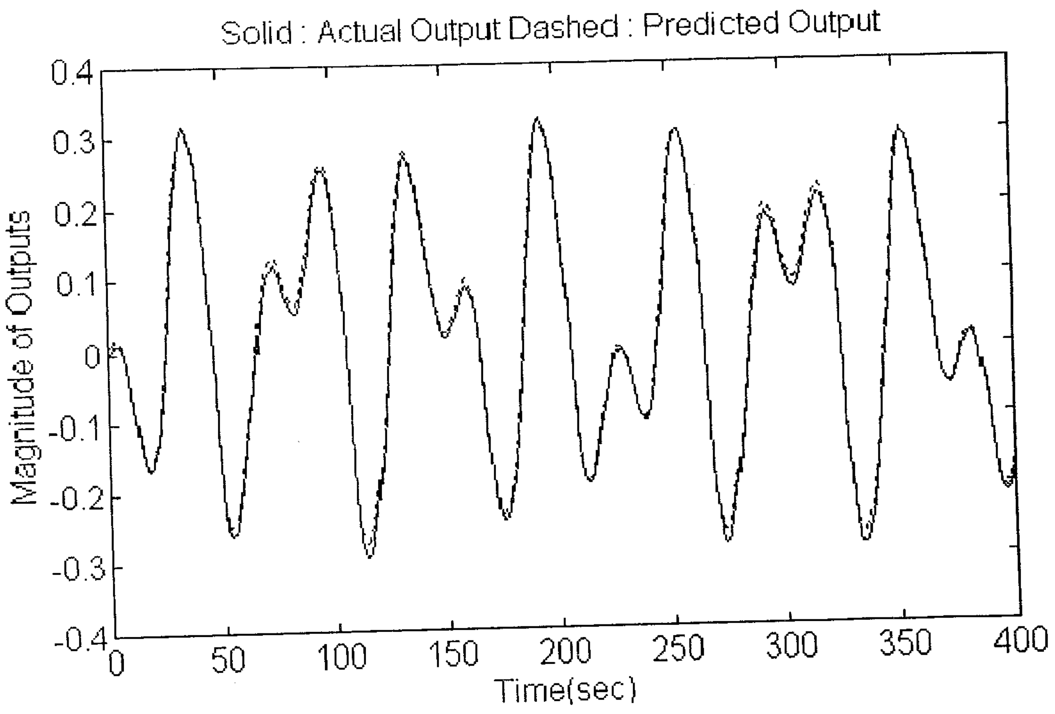


Figure 4.3(a) Actual and predicted outputs for the plant defined by Equation (4.10)

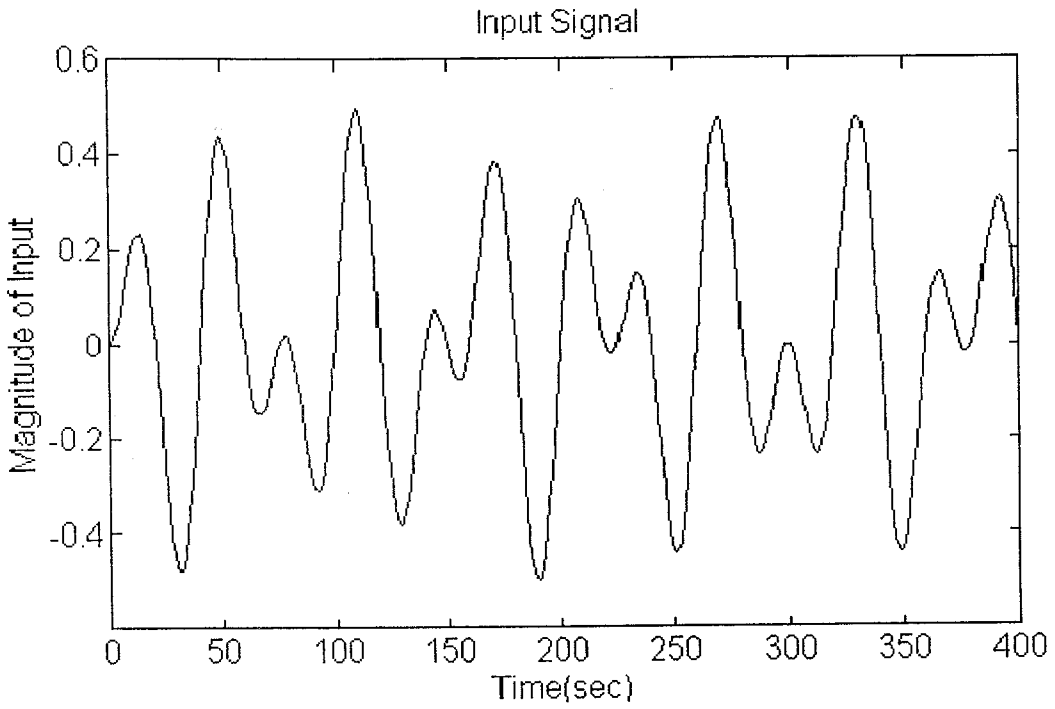


Figure 4.3(b) Applied input signal to the plant defined by Equation (4.10) and identification model

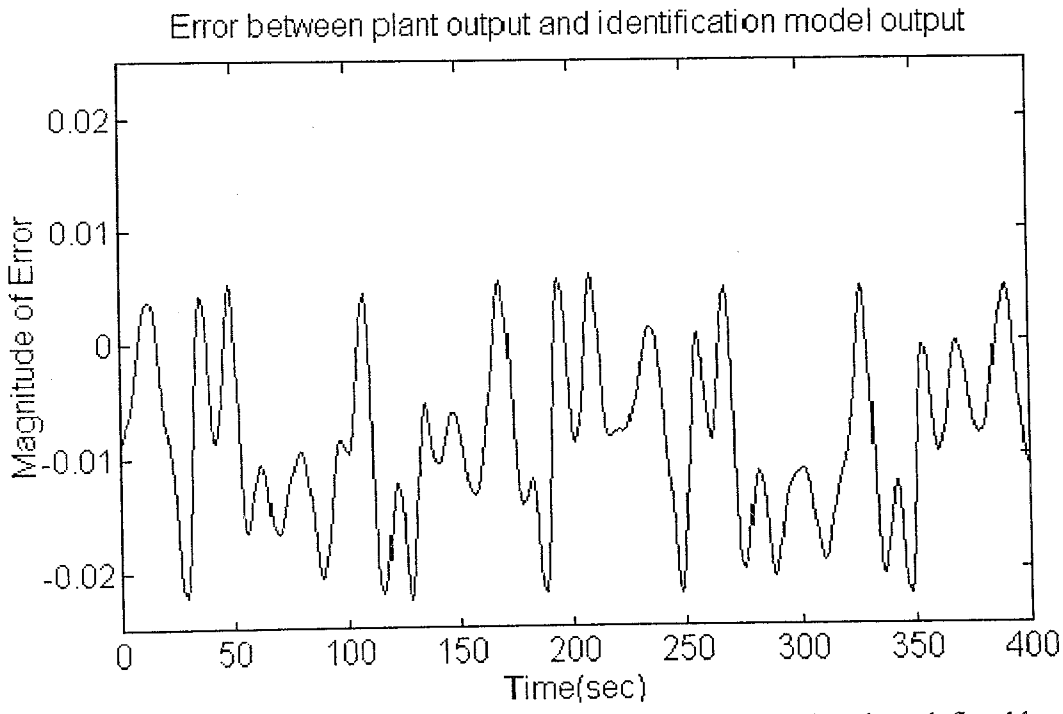


Figure 4.3(c) Error between actual and predicted outputs for the plant defined by Equation (4.10)

Plant Type : Model 3

Governing equation : $$y(k+1) = \frac{y(k)}{1 + y^2(k)} + u^3(k) \tag{4.12}$$

Input signal : $$h(k) = \begin{cases} 0.01k & 0 \le k \le 50 \\ -0.01k + 0.5 & 50 \le k \le 100 \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

$$u(k) = \sum_{i=0}^{\infty} h(k-100i) \tag{4.14}$$

Network Structure : 2-6-1, with linear output neuron

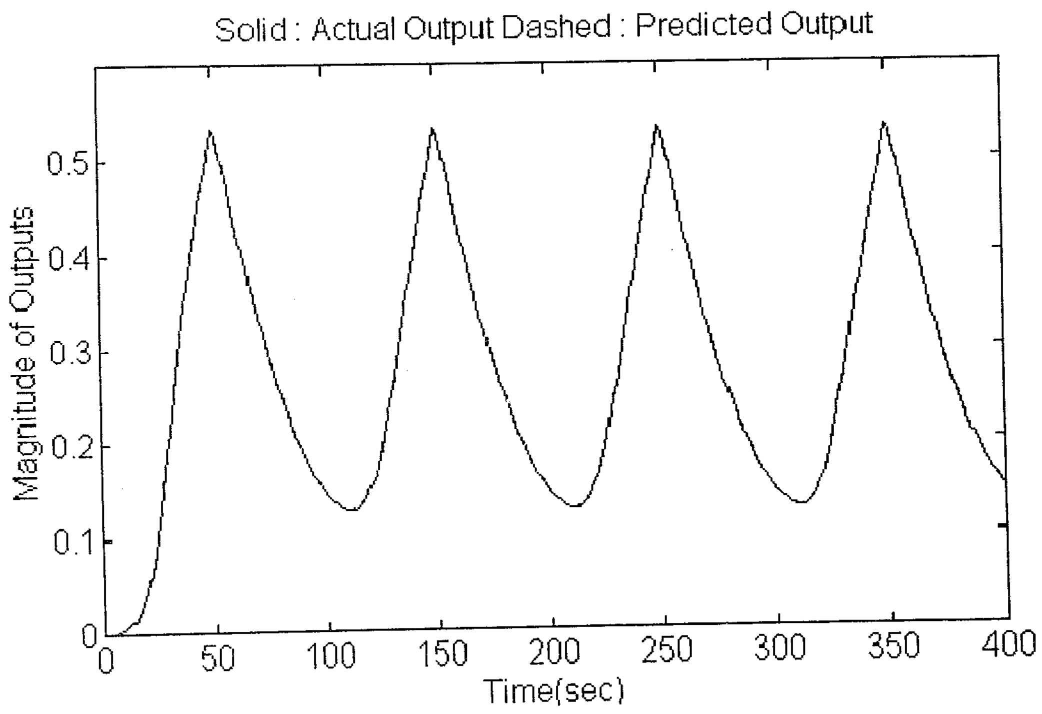


Figure 4.4(a) Actual and predicted outputs for the plant defined by Equation (4.12)

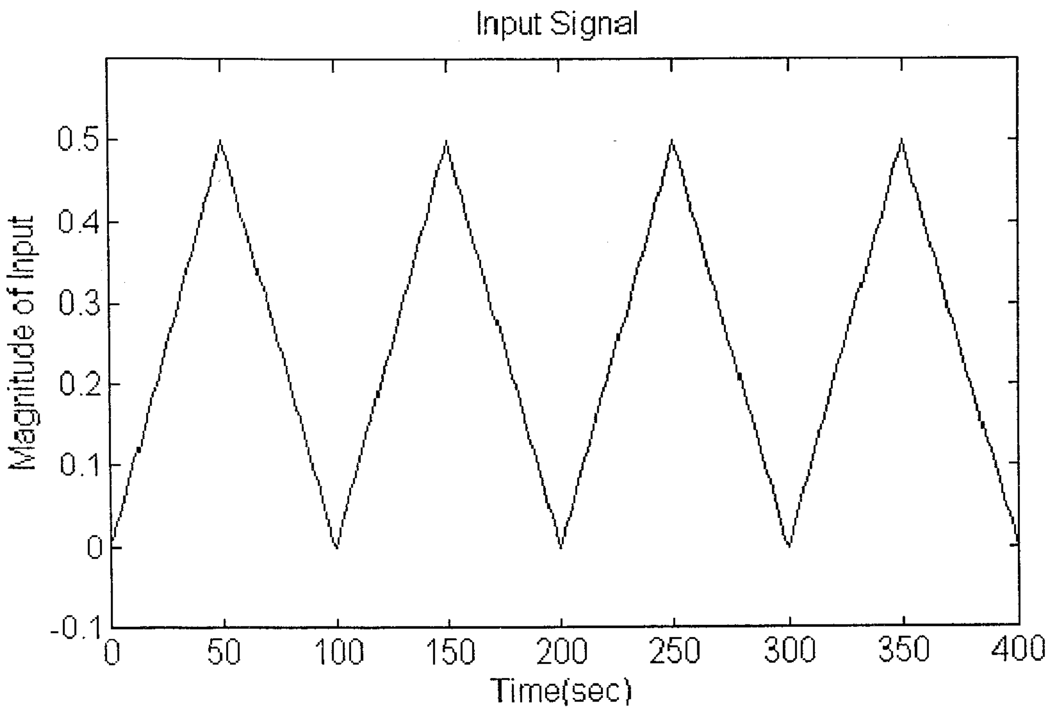


Figure 4.4(b) Applied input signal to the plant defined by Equation (4.12) and identification model

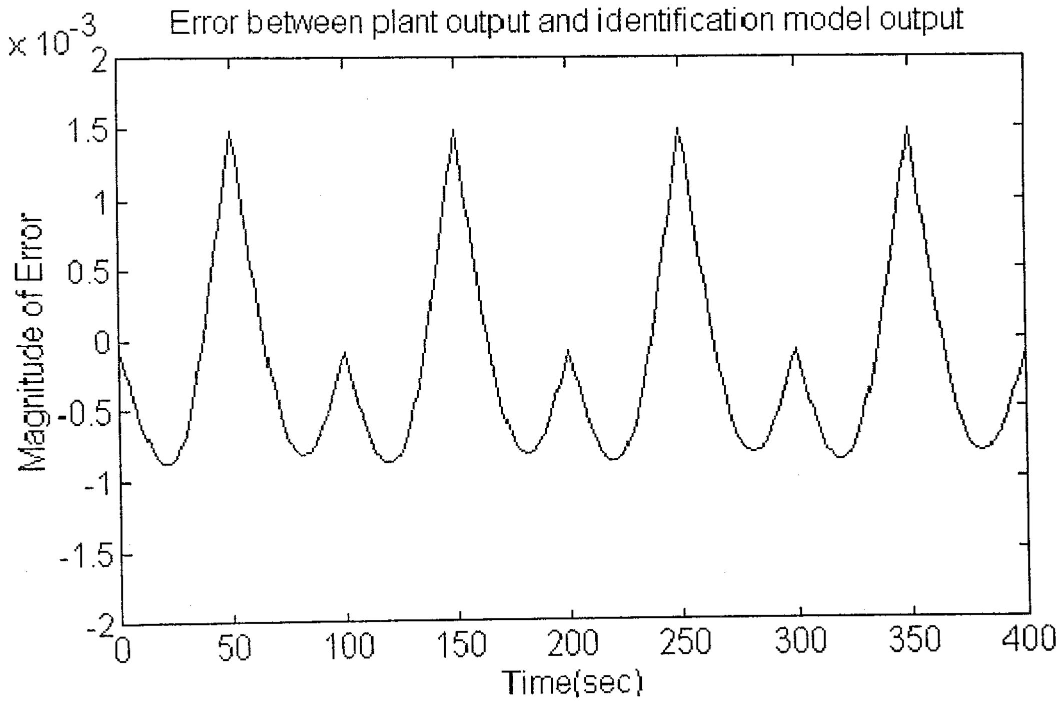


Figure 4.4(c) Error between actual and predicted outputs for the plant defined by Equation (4.12)

PlantType : Model 4

Governing equation : $y(k+1) = f(-y(k) - 0.6\,y(k-1) + 0.3\,y(k-2) + u(k) + u(k-1))$ (4.15)

$$f(x) = \frac{\tanh(x)}{1+x^2} \tag{4.16}$$

Input signal : $u(k) = \frac{1}{2}\sin\left(\frac{2\pi k}{150}\right)\sin\left(\frac{2\pi k}{40}\right)$ (4.17)

Network Structure : 5-20-10-1, with linear output neuron

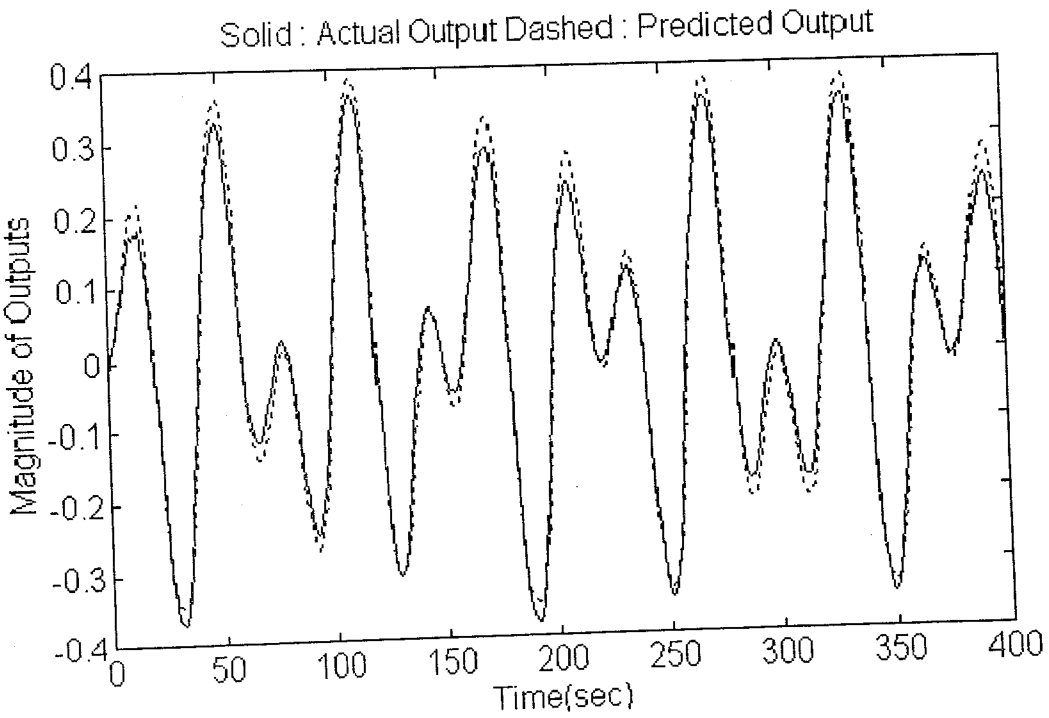


Figure 4.5(a) Actual and predicted outputs for the plant defined by Equation (4.15)

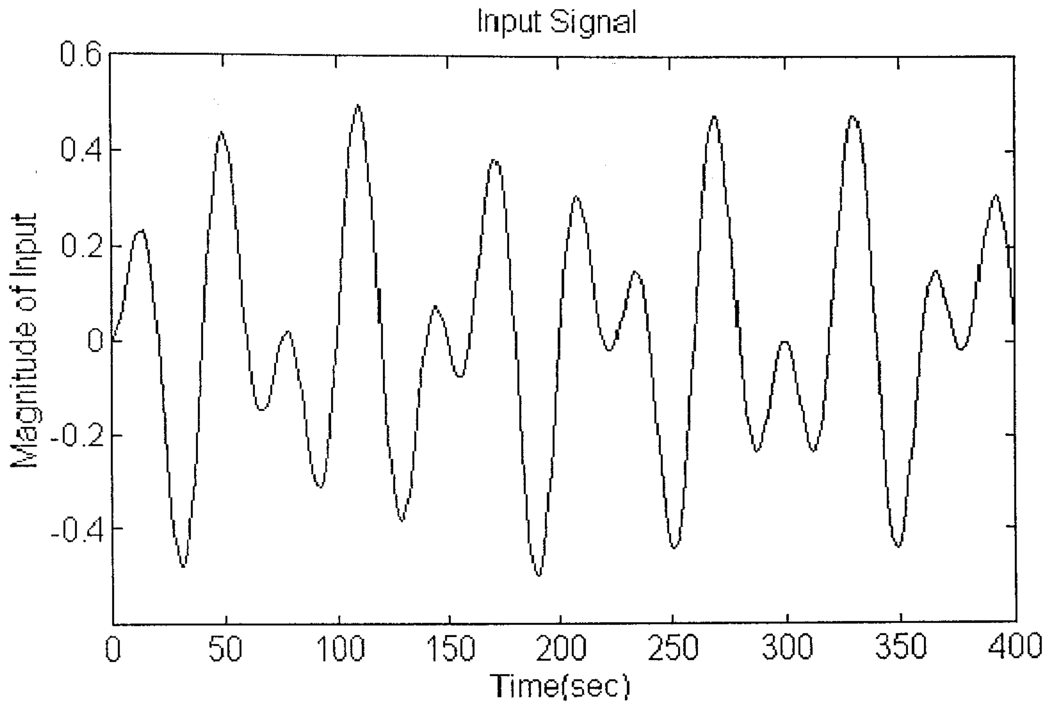


Figure 4.5(b) Applied input signal to the plant defined by Equation (4.15) and identification model

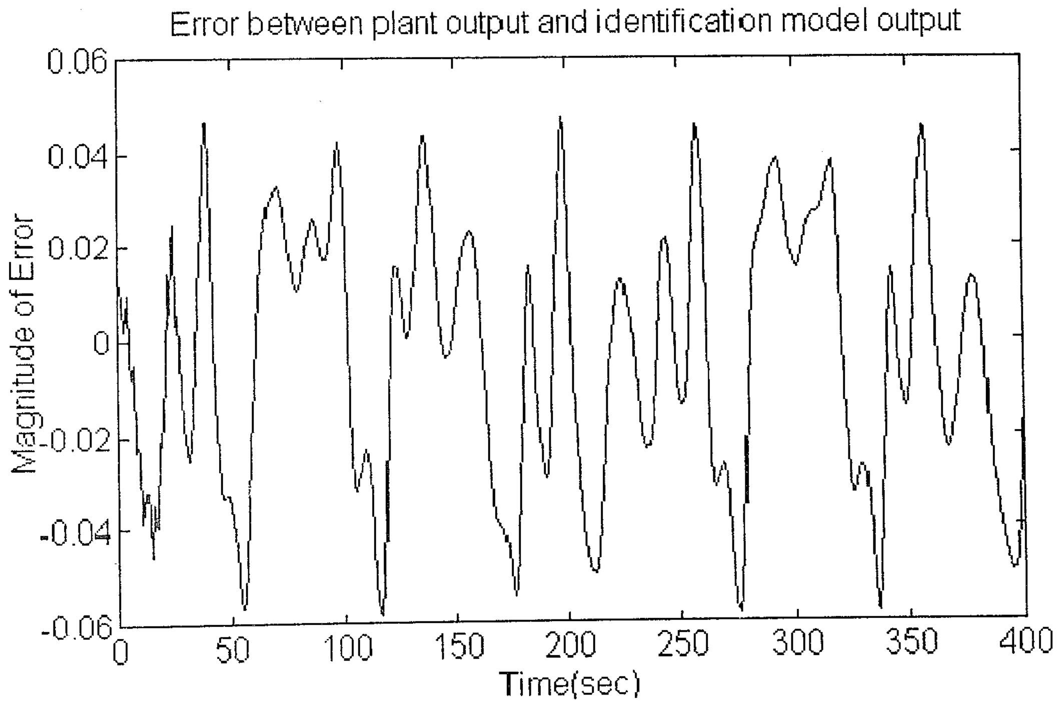


Figure 4.5(c) Error between actual and predicted outputs for the plant defined by Equation (4.15)

Plant Type : Model 4

Governing equation : $y(k+1) = f(-0.2\,y(k) + 1.8\,u(k))$ (4.18)

$$f(x) = \frac{x}{1+x^2} \quad (4.19)$$

Input signal : $u(k) = \frac{1}{2}\sin\left(\frac{2\pi k}{150}\right)\sin\left(\frac{2\pi k}{40}\right)$ (4.20)

Network Structure : 2-4-1, with linear output neuron

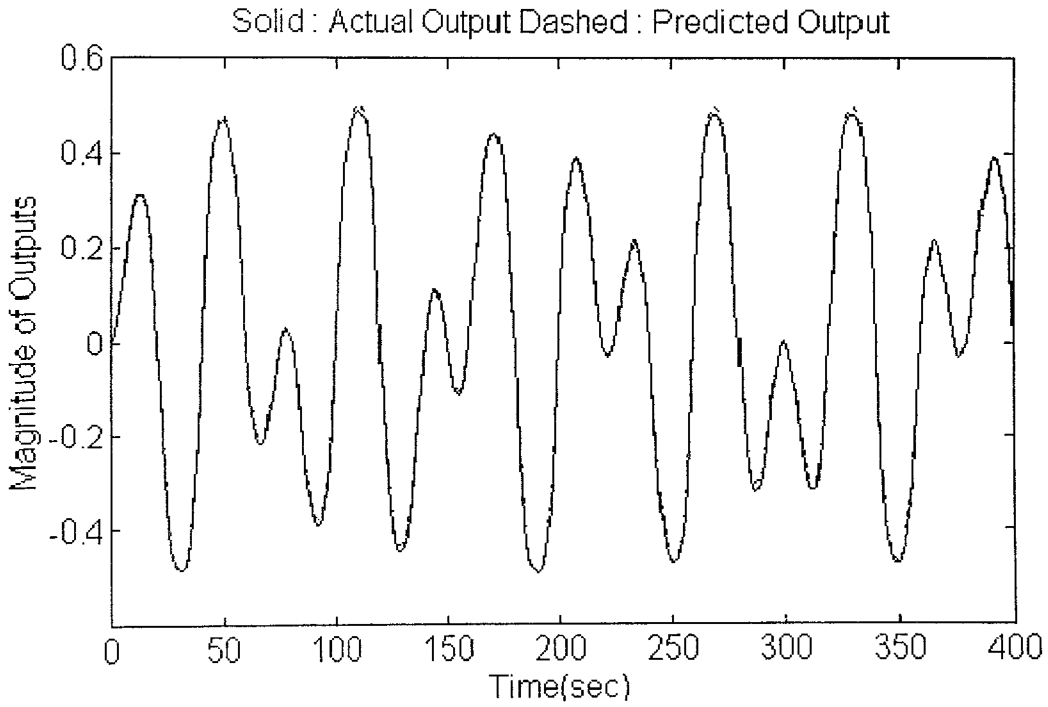


Figure 4.6(a) Actual and predicted outputs for the plant defined by Equation (4.18)

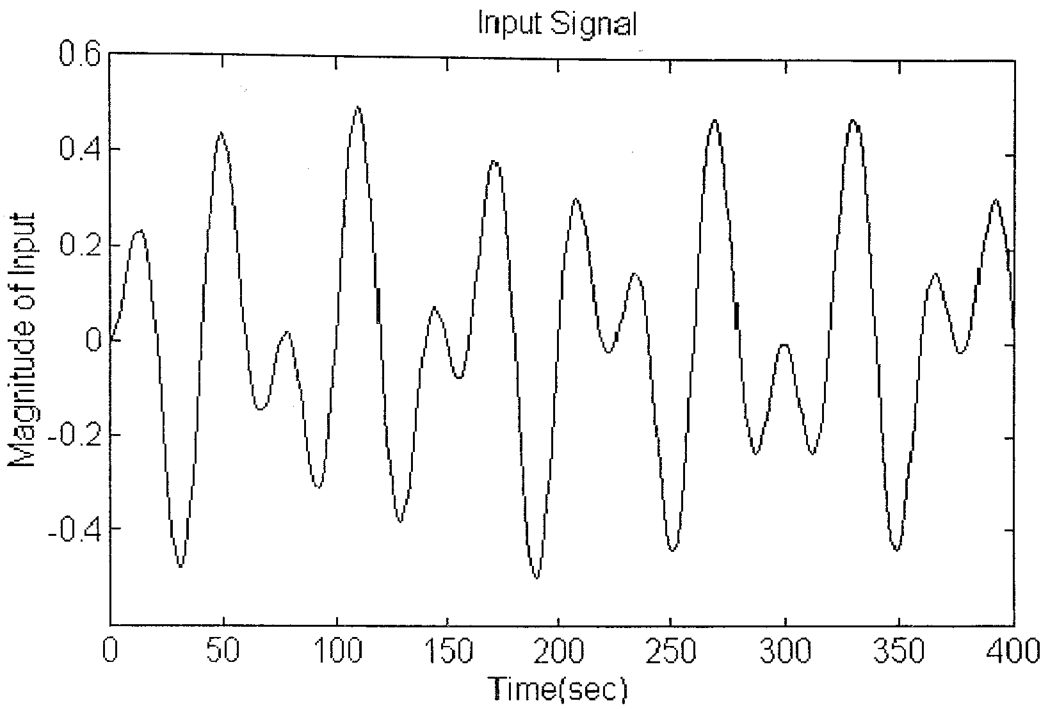


Figure 4.6(b) Applied input signal to the plant defined by Equation (4.18) and identification model

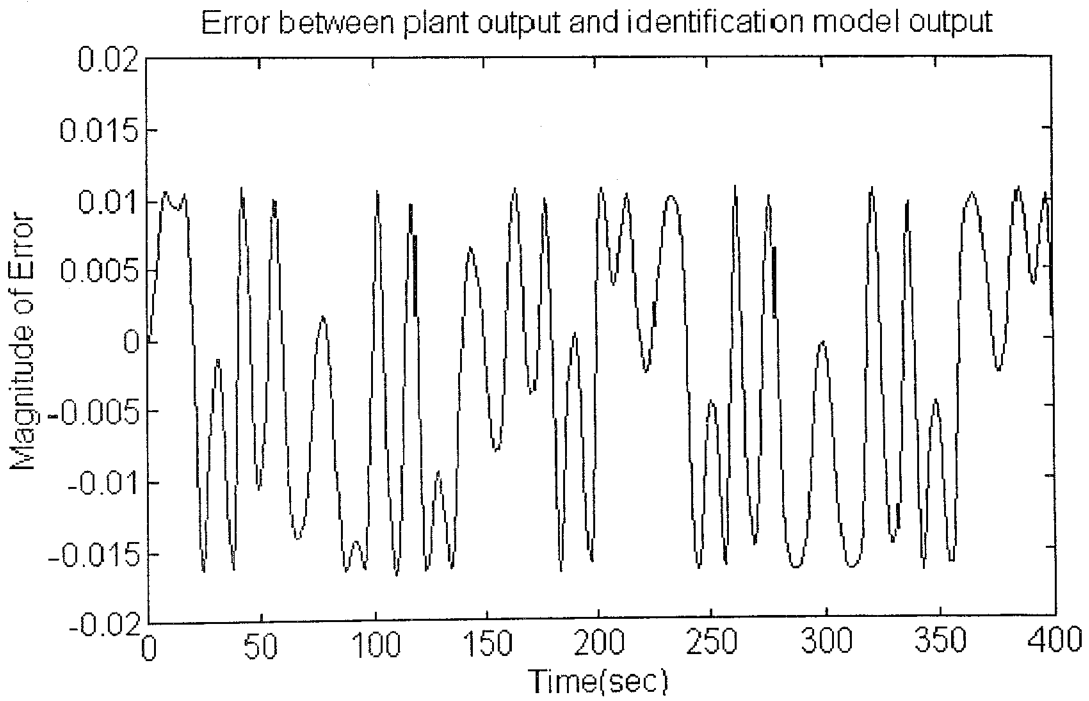


Figure 4.6(c) Error between actual and predicted outputs for the plant defined by Equation (4.18)

Plant Type : Biorector model with coupled nonlinear difference eqns.

$$c_1(k+1) = c_1(k) + \Delta\left(- c_1(k)\, r(k) + c_1(k)\left(1-c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \right) \tag{4.21}$$

$$c_2(k+1) = c_2(k) + \Delta\left(-c_2(k)\, r(k) + c_1(k)\left(1 - c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \frac{1+\beta}{1+\beta - c_2(k)} \right) \tag{4.22}$$

Input signal : Randomly generated between [0,1] for $c_1(k)$ and $c_2(k)$, between [0,2] for r(k).

Network Structure : 3 - 25 - 16 - 2, with linear output neuron

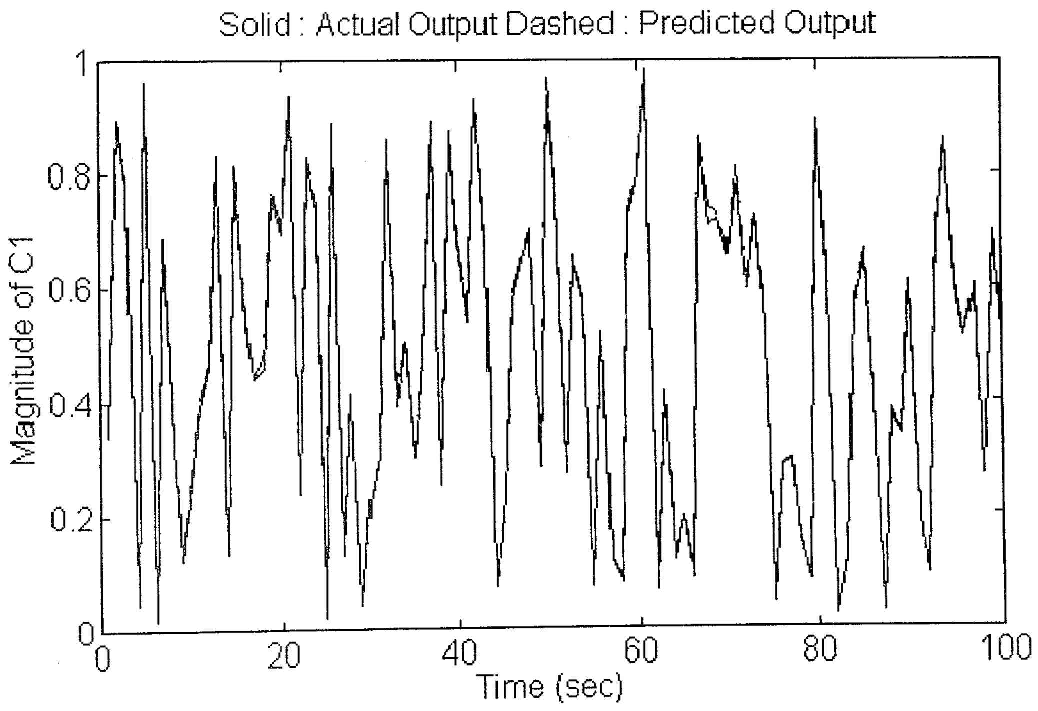


Figure 4.7(a) Actual and predicted c_1 values

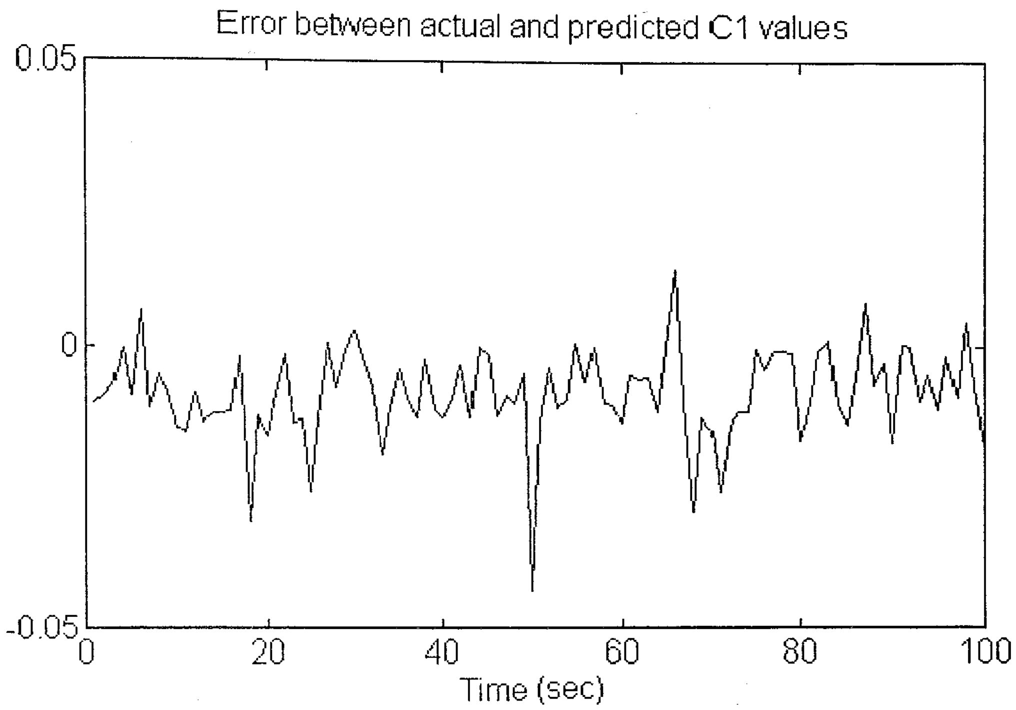


Figure 4.7(b) Error between actual and predicted c_1 values

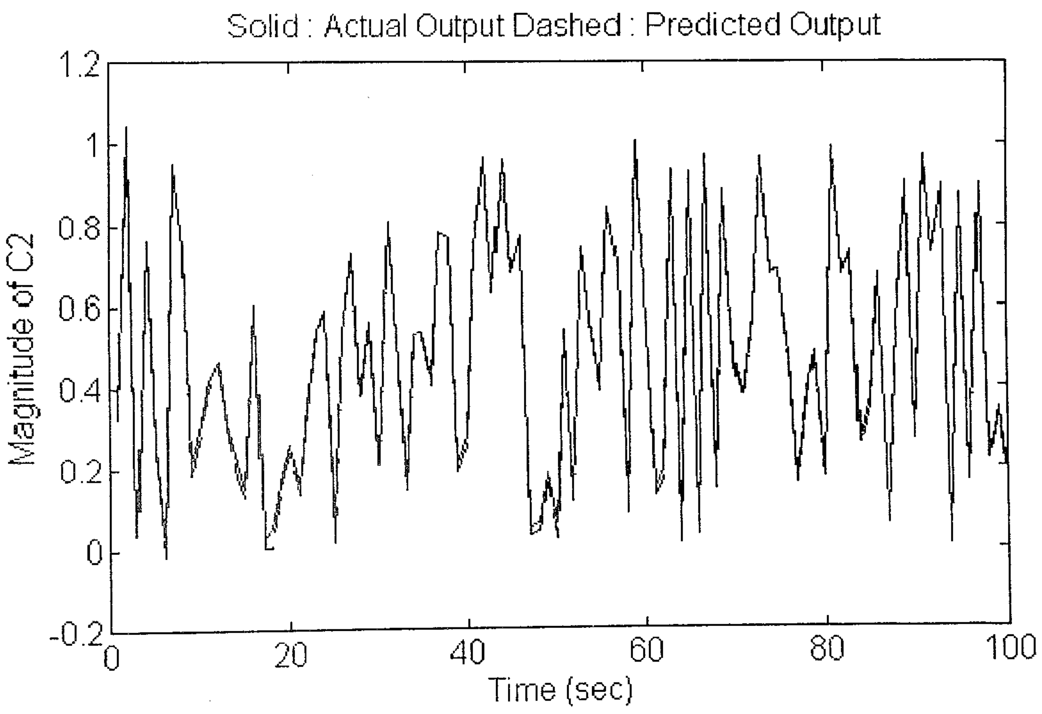


Figure 4.7(c) Actual and predicted c_2 values

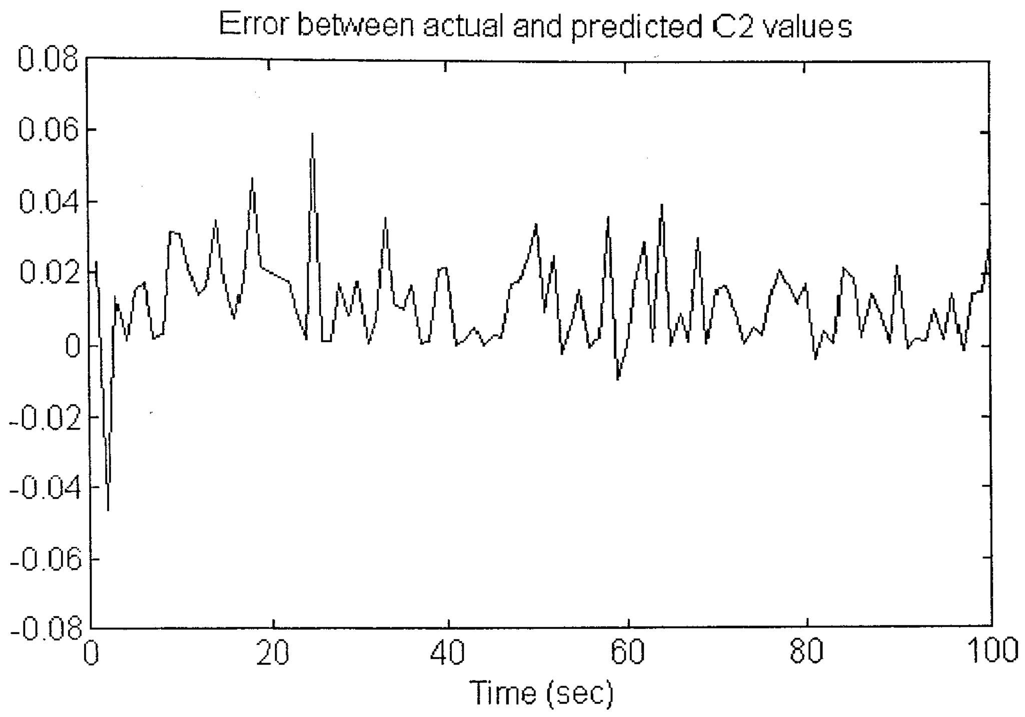


Figure 4.7(d) Error between actual and predicted c_2 values

As can be seen from the simulation results, neural identifiers are capable of representing the dynamical behavior of complex systems. Our simulations showed also that there is a small discrepancy between the outputs of the neural identifiers and the actual plants. We attribute this discrepancy to two important causes;

1. If the training set does not represent the plant dynamics sufficiently well, then the resulting neural identification model will exhibit some deviations from the actual plant output.

2. If the neural identification model is not well trained, then the same deviations will occur at the outputs.

Briefly, provided that these two adverse effects are eliminated, the procedure yields admissible neural models which can be used, in turn, for the control of nonlinear dynamical processes. In the remainder of this thesis, we elaborated different control strategies by the use of neural networks.

5. CONTROL ARCHITECTURES FOR ERROR BACKPROPAGATION

In the previous chapters of this thesis, we mainly dwelt on two widely used learning algorithms and system identification using neural networks. This chapter presents control of nonlinear dynamical systems using neural networks and employs the concepts that have been explained so far and the error backpropagation technique.

As mentioned earlier, the problem in controlling of a plant is finding of an input sequence so that the plant behaves in a desired fashion. Unfortunately, the complexity of the plant makes the issue highly involved. From this point of view, neural networks can be used as controllers because of their powerful mapping capabilities. The main emphasis in this approach is that the mapping generated by the neural network must adequately represent the system's behavior in the range of interest. In this chapter, we considered the neural controller synthesis for various types of plant models. The error backpropagation technique is utilized in training of the neurocontroller.

The second topic in this chapter is performance improvement in output tracking by adding a conventional controller to the control system. The need for adding such a controller arose because of the fact that the model mismatches in the plant identification model results in learning of an inexact inverse dynamics of the plant. Therefore the resulting neurocontroller is not able to control the plant at some operating points and it causes some steady state errors even in the cases that the reference signal is sufficiently smooth.

Lastly, we considered the direct inversion of the plant dynamics where the controller training procedure resembles to an identification procedure.

5.1 Control Strategy

Error backpropagation constitutes the basis of the neural network based controller design. Once the neurocontroller is trained, it simply cancels the effect of the plant under control. This means that, without loss of generality, controller realizes the inverse dynamics of the plant and the overall transfer function of the control system is therefore reduced to the system which is comprised of unity transfer function.

The key step in this approach is how the controller learns the inverse dynamics of the plant. Two main architectures are proposed by Psaltis, Sideris and Yamamura [3]. These architectures illustrated in Fig. 5.1 and Fig. 5.2.

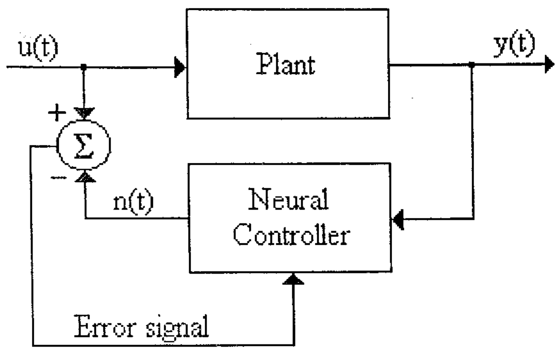


Figure 5.1 Generalized learning architecture

In Fig. 5.1, generalized learning architecture is illustrated. The objective is minimization of the error signal in magnitude. In response to u(t), the plant generates an output y(t) which is then applied to the neural network. Neural network is supposed to relate the output value y(t) with the input u(t). As training progresses, neural network learns the inverse dynamics of the plant and will then be able to control the plant. Psaltis, Sideris and Yamamura [3] say that the neurocontroller could not be trained selectively in the regions of interest. This basically stems from the fact that we do not know which control inputs will cause the desired plant outputs. Therefore, by the use of this architecture, neural controller just gains the general behavior of the inverse dynamics of the plant over a wide range of inputs.

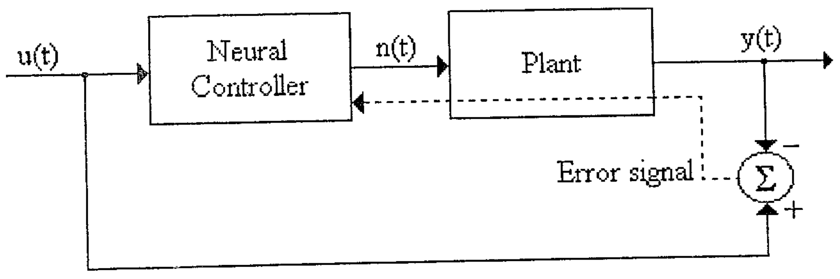


Figure 5.2 Specialized learning architecture

In Fig. 5.2, specialized learning architecture is shown. In this approach, neural controller is trained only in the region of interest and the desired response of the overall system is provided to the controller. Again, as the training progresses, the error signal is minimized in magnitude and this is also achieved through error backpropagation. In [3], it is said that beginning with the generalized learning architecture and continuing with the specialized learning architecture would be a proper training method. Because, general learning provides better initial weights for the specialized learning.

The main difference between these two learning architectures is that the minimization is carried out on different error measures. In the generalized method, the error signal is obtained from the reproduction of the input signal, whereas in the specialized method, error signal is obtained from the difference between desired and actual responses of the plant. Theoretically, both of these methods go to the minima by following different trajectories on the cost surface.

The specialized learning architecture requires the derivatives of the plant. More explicitly, we want to evaluate the effect of a change in each input of the plant on the outputs. Two different methods can be proposed for obtaining the derivatives of the plant. In the first method, derivatives are evaluated by using the iterative values of the inputs and outputs. The second method assumes the plant as an unmodifiable neural network, which is the identification model of the plant, and utilizes the error backpropagation technique. Figure 5.3 illustrates the controller training architecture by the use of error backpropagation technique.

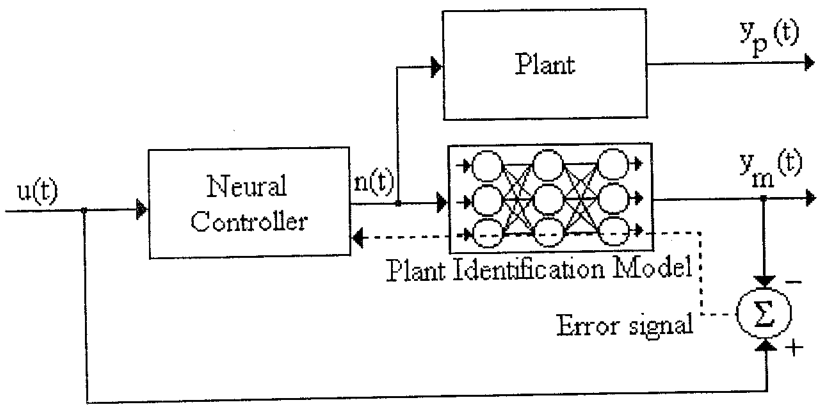


Figure 5.3 Controller training architecture by the use of plant identification model

Once the neural controller is trained, then the identification model is replaced with the real plant. This approach is called off-line training, on the other hand, if the controller is trained in the real operation of the plant, this is called on-line training. This section adopts off-line training method. Besides, the reference model is a system with unity transfer function i.e. the controller is expected to learn the inverse dynamics of the plant over the range of interest.

5.2 Simulation Results

Simulation 1.

Plant : $y(k+1)=\dfrac{u(k)}{1+u^2(k)}$ (5.1)

Identification model	: 1-6-1	MSE : 1e-6
Controller	: 1-10-4-1	MSE : 178e-6

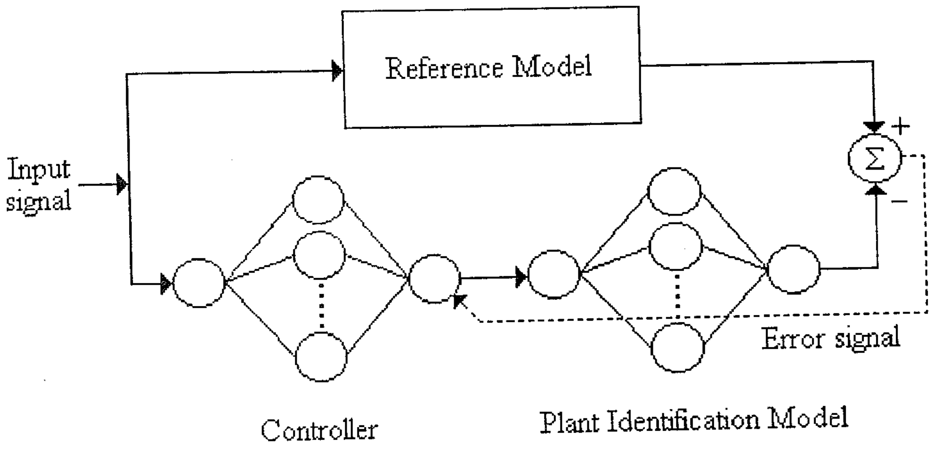


Figure 5.4 Controller training architecture for the first simulation

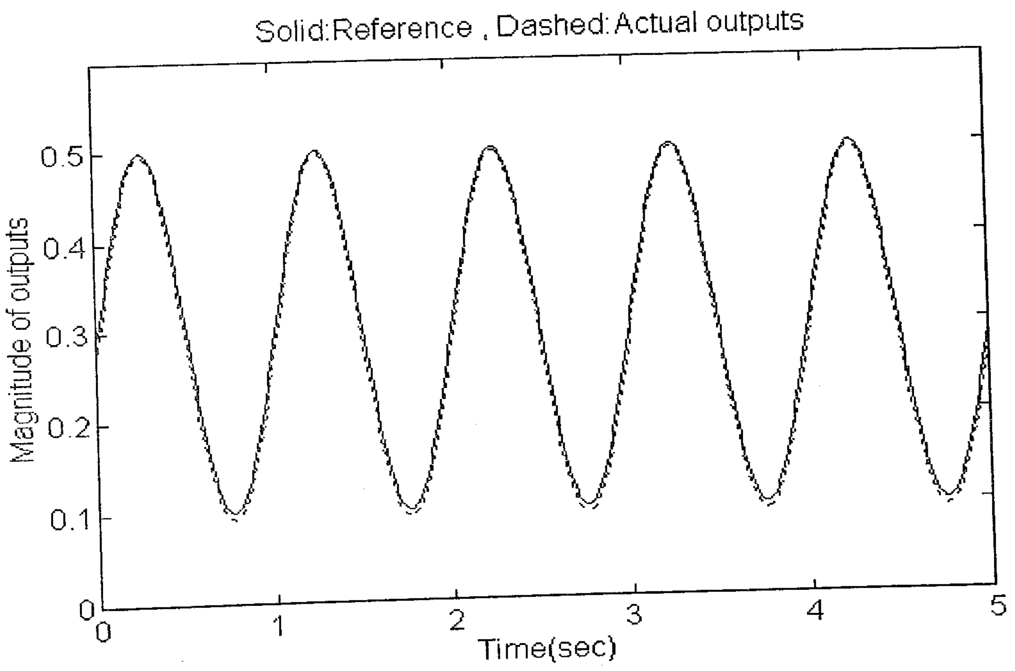


Figure 5.5(a) Reference and actual trajectories for the first plant, reference input is $r(t) = 0.1+0.2*[1+\sin(2\pi t)]$

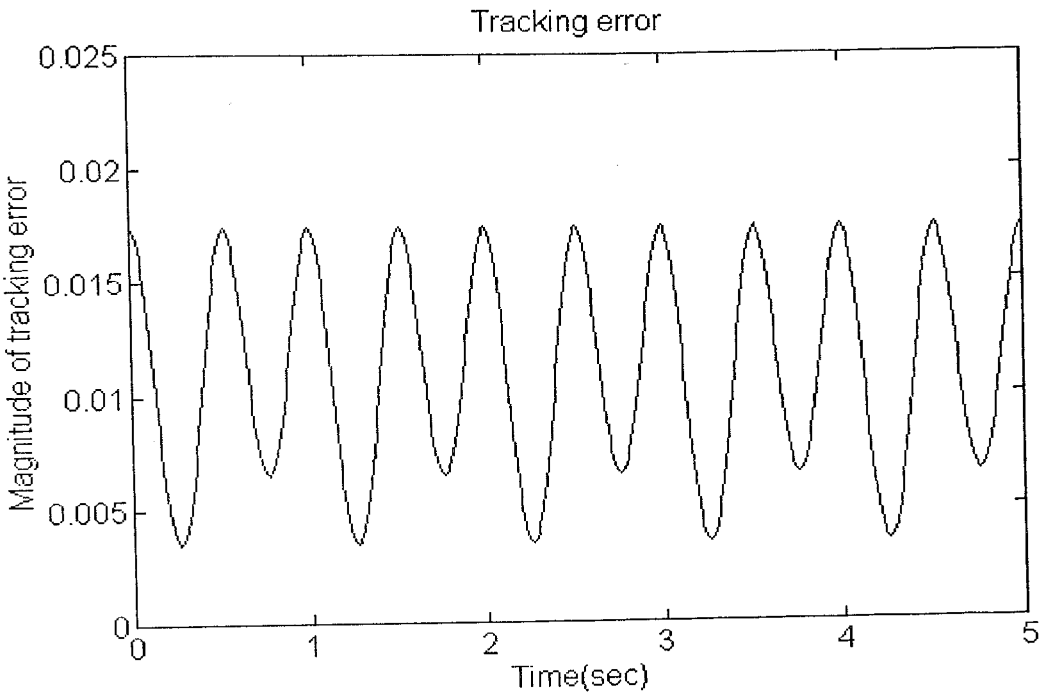


Figure 5.5(b) Tracking error for the first plant, the reference input is
r(t) = 0.1+0.2*(1+sin(2πt))

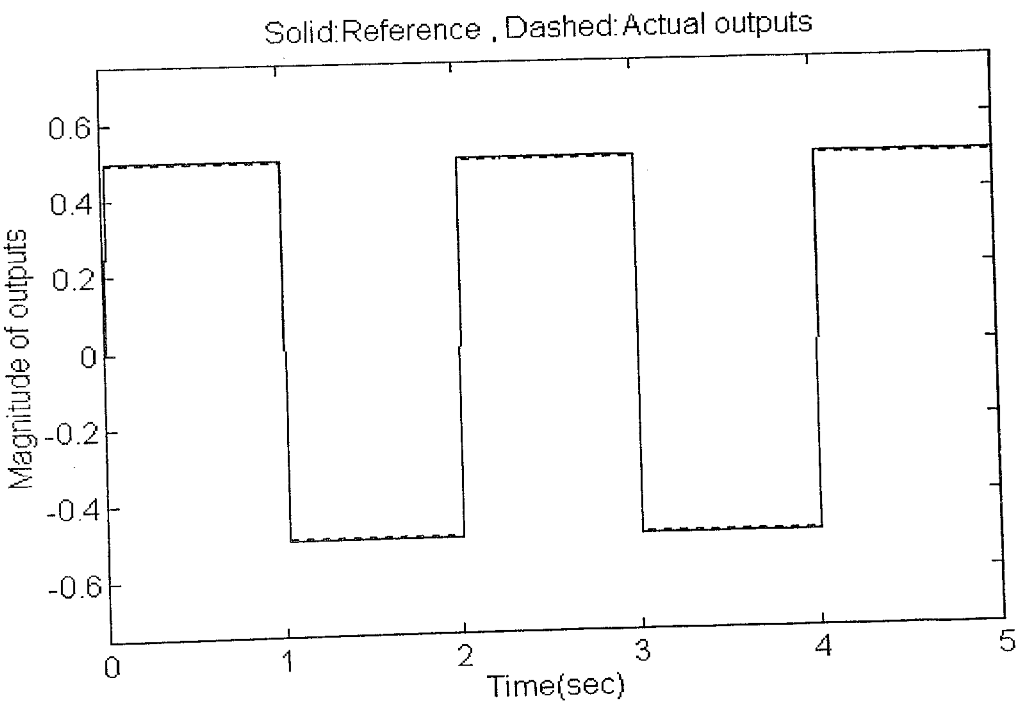


Figure 5.6(a) Reference and actual trajectories for the first plant, reference input is
r(t) = 0.5*sgn[sin(πt)]

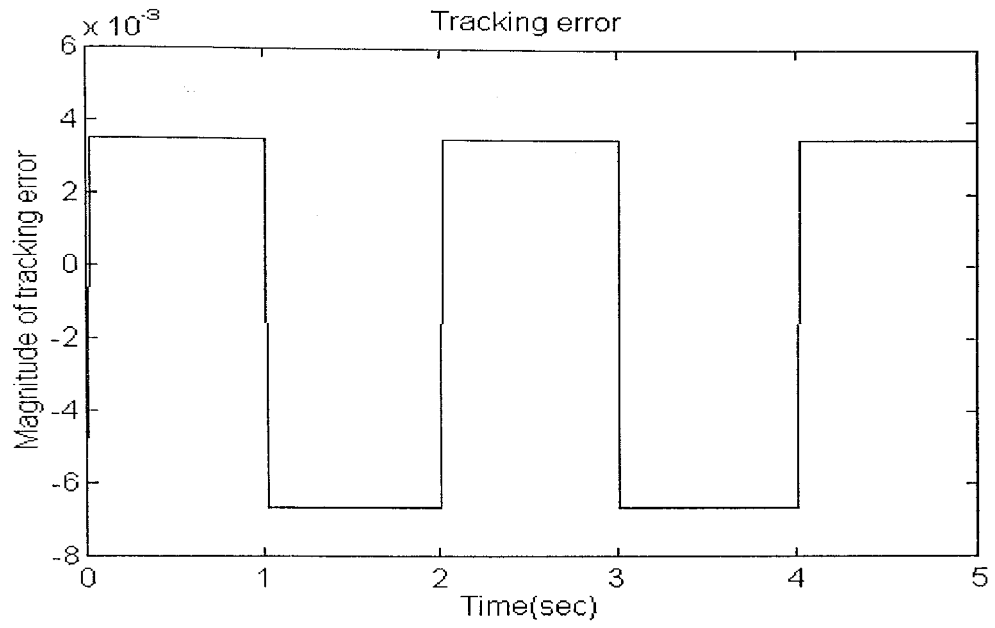


Figure 5.6(b) Tracking error for the first plant, the reference input is $r(t) = 0.5*\text{sgn}[\sin(\pi t)]$

Simulation 2.

Plant : $y(k+1) = \tanh(y(k)) + u(k)$ (5.2)

Identification model	: 2-6-1	MSE : 1e-6
Controller	: 2-10-1	MSE : 843e-6

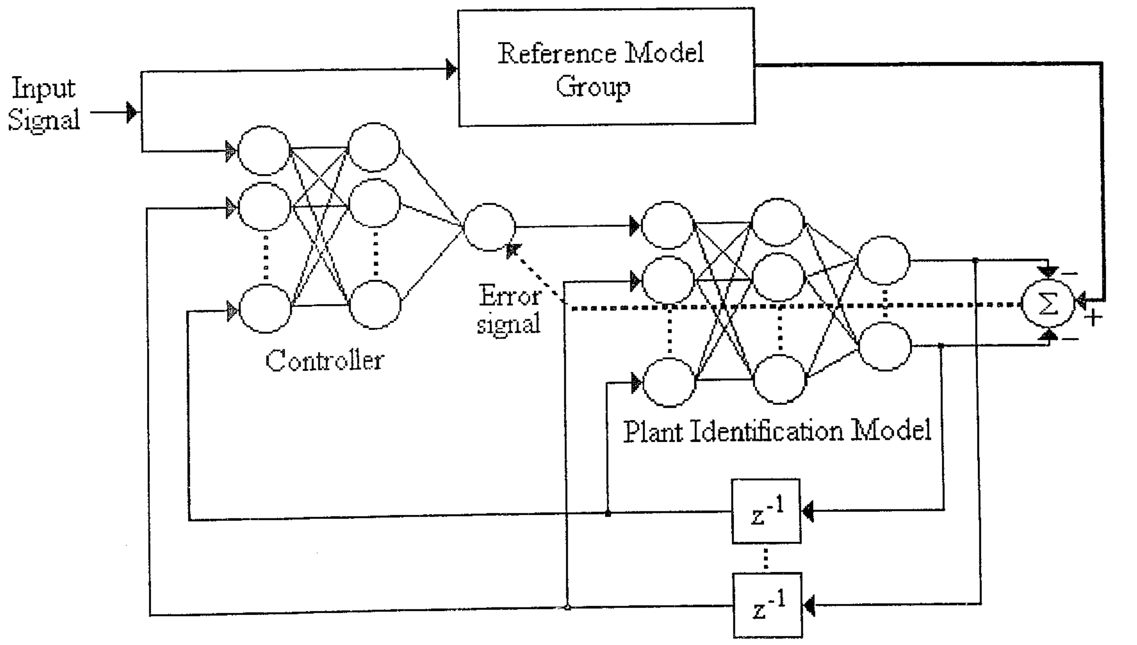


Figure 5.7 Controller training architecture for the second simulation

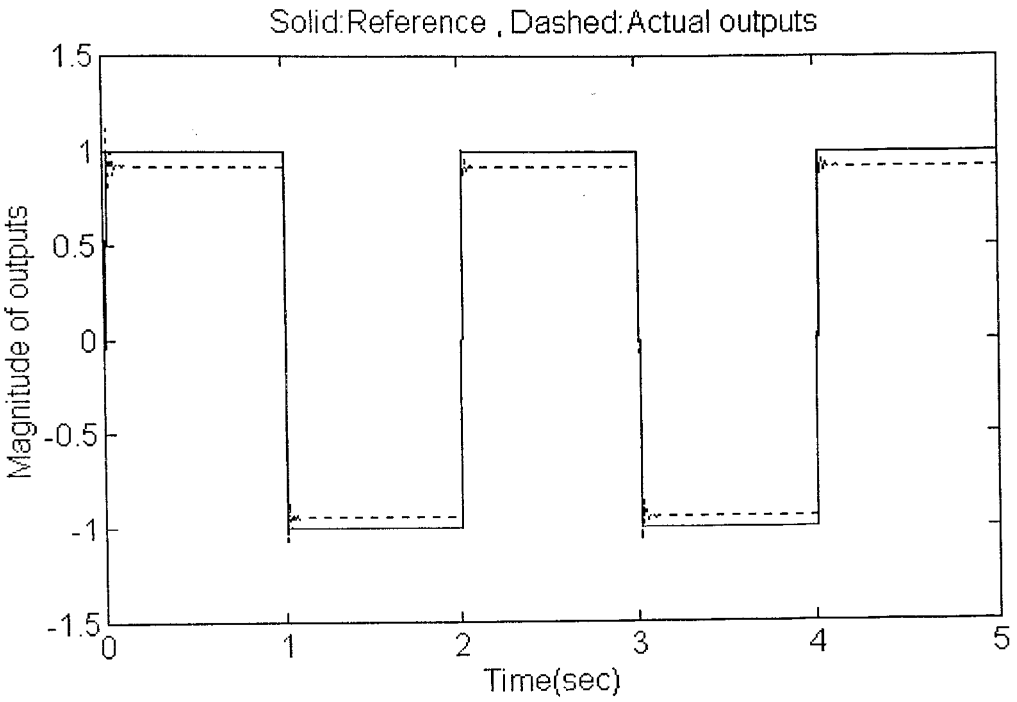


Figure 5.8(a) Reference and actual trajectories for the second plant, reference input is $r(t) = \mathrm{sgn}[\sin(\pi t)]$

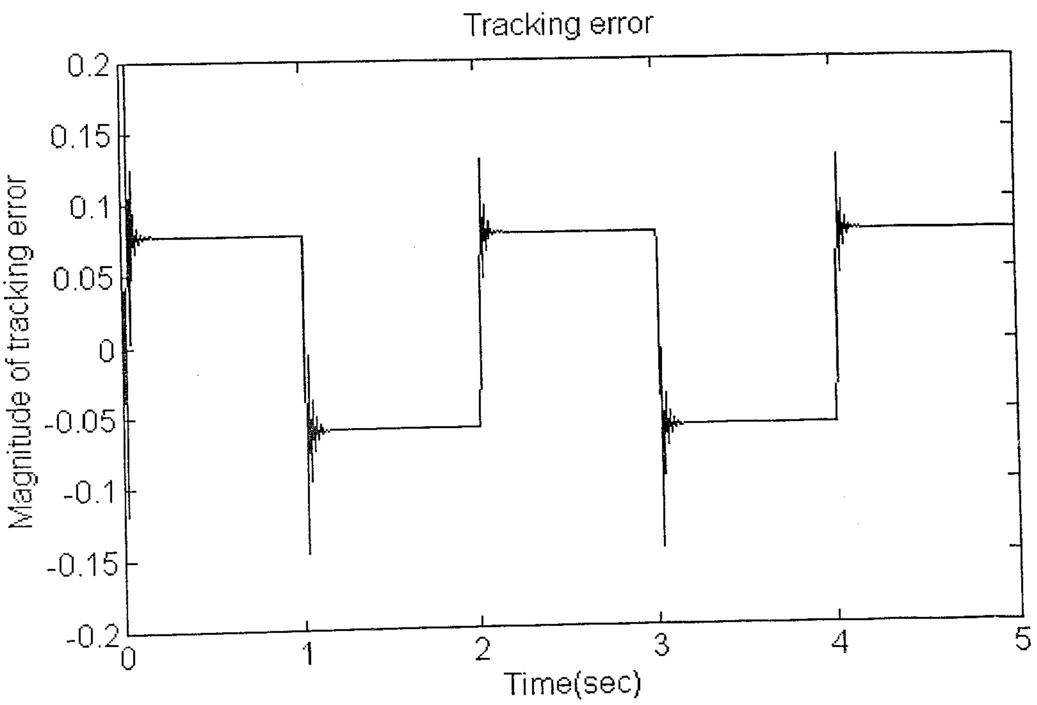


Figure 5.8(b) Tracking error for the second plant, the reference input is $r(t) = \mathrm{sgn}[\sin(\pi t)]$

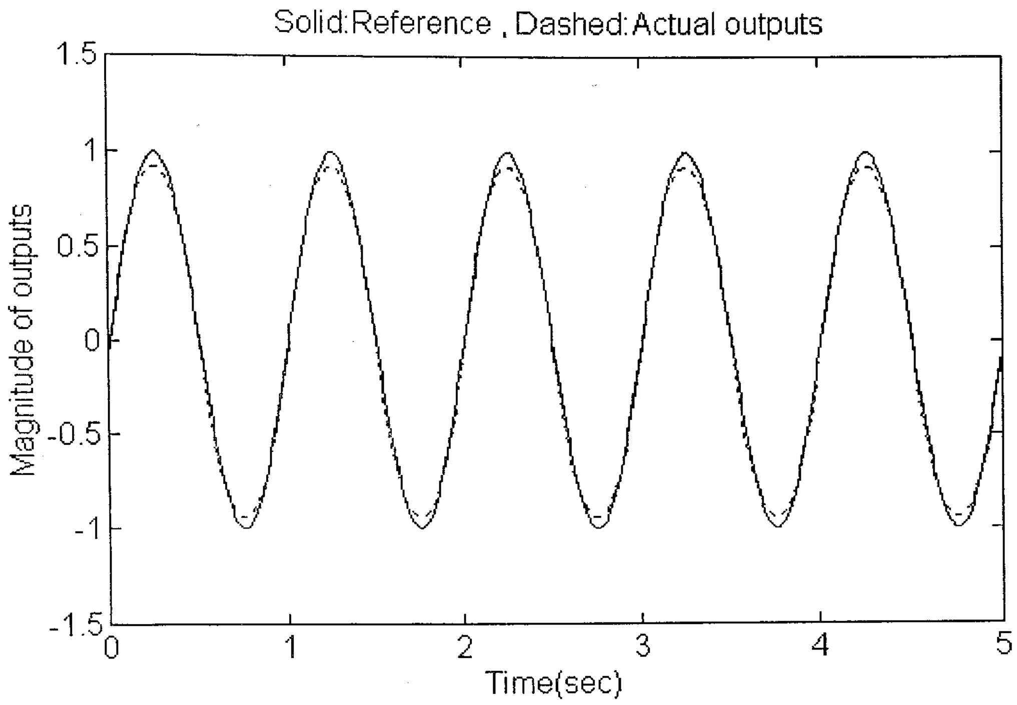


Figure 5.9(a) Reference and actual trajectories for the second plant, reference input is r(t) = sin(2πt)

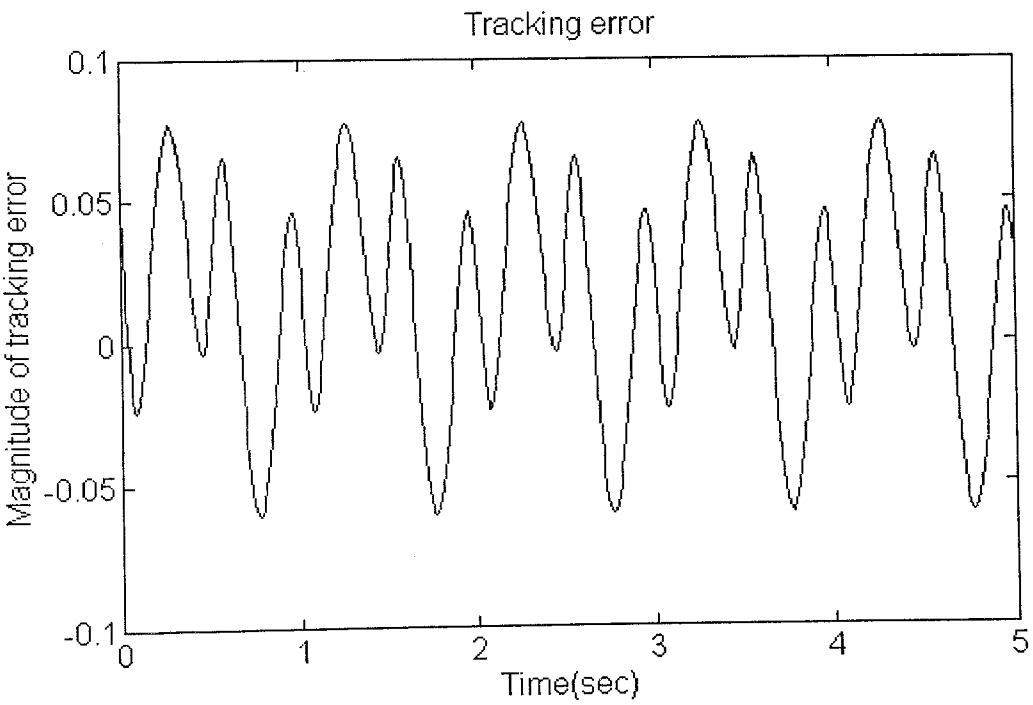


Figure 5.9(b) Tracking error for the second plant, the reference input is r(t) = sin(2πt)

Simulation 3.

Plant : $y(k+1) = 1.3679y(k) - 0.3679y(k-1) + 0.3679u(k) + 0.2642u(k-1)$ (5.3)

Identification model	: 4-10-1	MSE : 29e-6
Controller	: 3-16-4-1	MSE : 978e-6

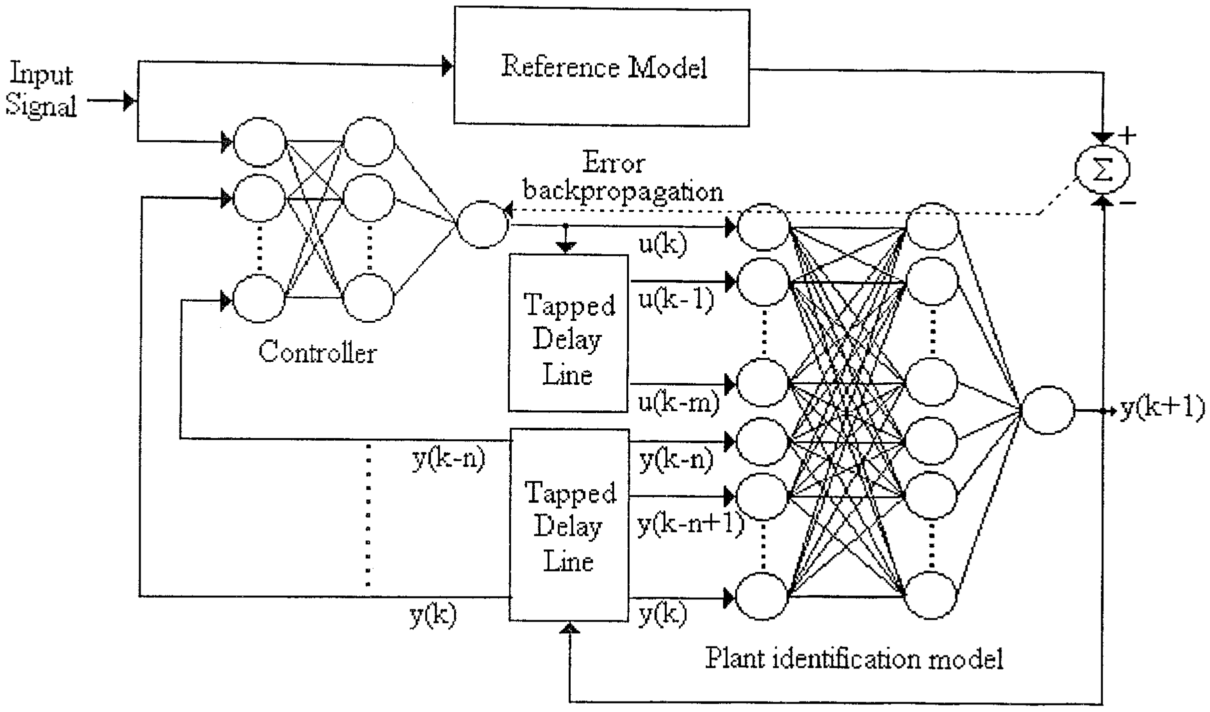


Figure 5.10 Controller training architecture for the third simulation

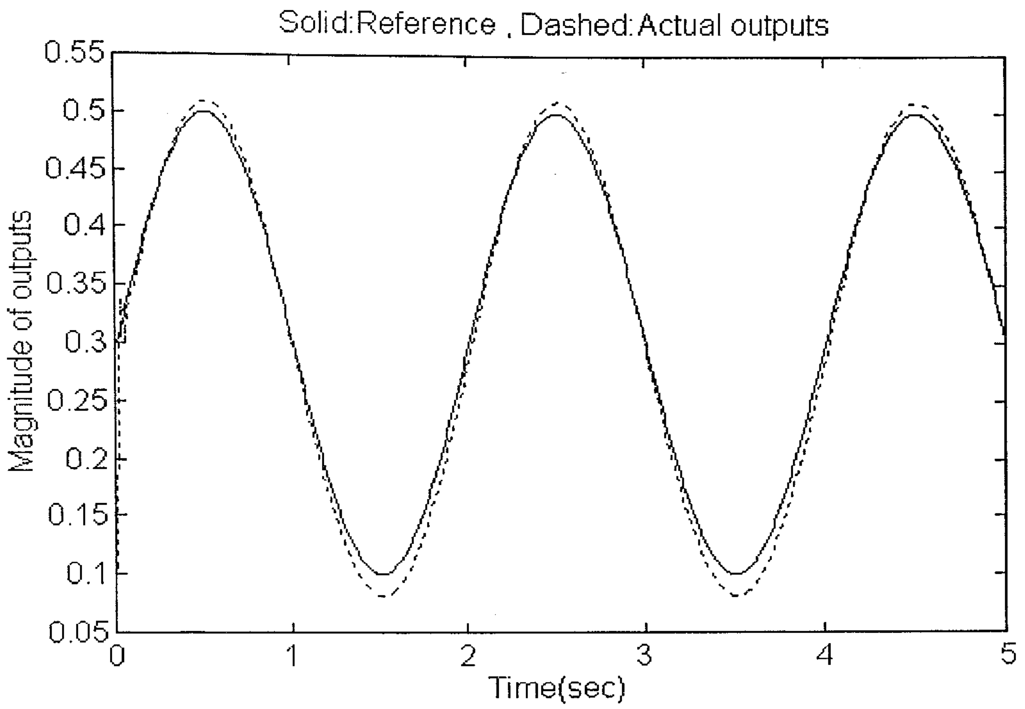


Figure 5.11(a) Reference and actual trajectories for the third plant, reference input is r(t) = 0.1+0.2*[1+sin(πt)]

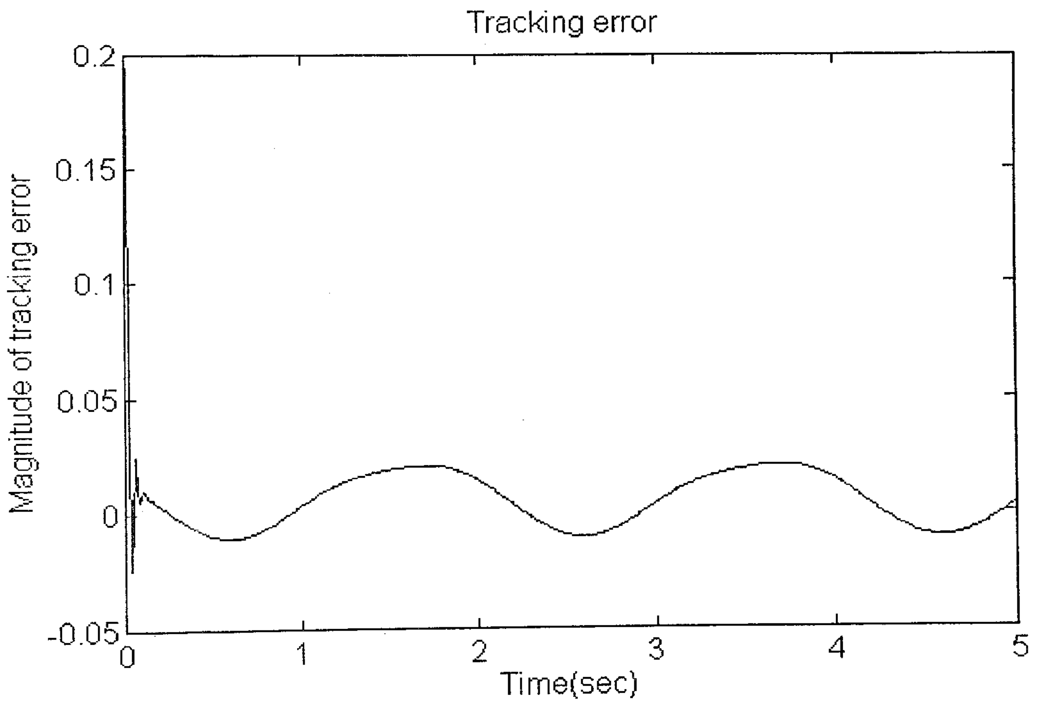


Figure 5.11(b) Tracking error for the third plant, the reference input is r(t)=0.1+0.2*[1+sin(πt)]

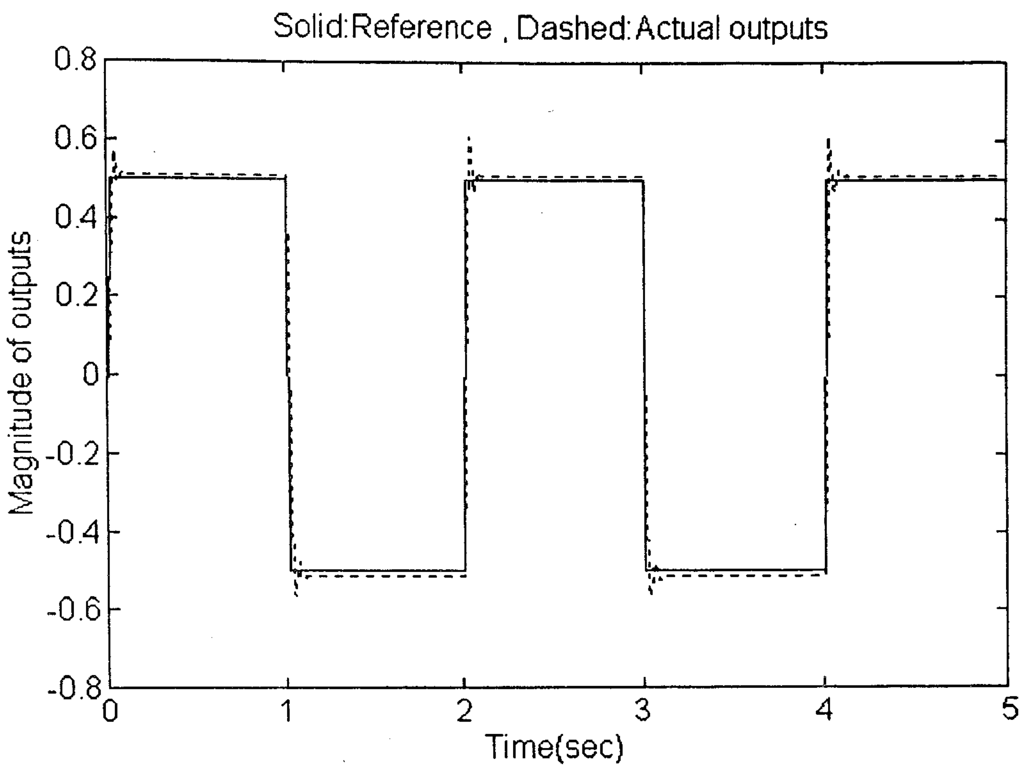


Figure 5.12(a) Reference and actual trajectories for the third plant, reference input is r(t) = 0.5*sgn[sin(πt)]

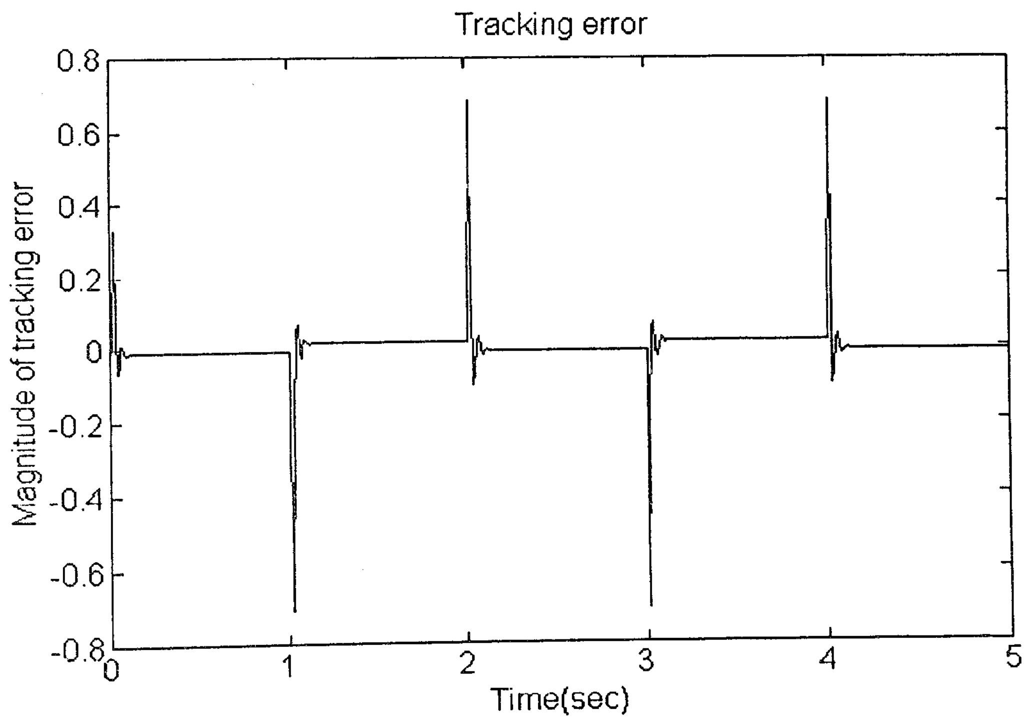


Figure 5.12(b) Tracking error for the third plant, the reference input is r(t) = 0.5*sgn[sin(πt)]

5.4 Performance Improvement by Adding a PI Controller

In this section, we discuss the performance improvement by adding a PI controller to the control system. The architecture is illustrated in Fig. 5.13. The need for a conventional control support arose because of the following two crucial facts:

1. The governing equations of the plant may be non-invertible. Consequently, the neural controller realizes the inverse dynamics roughly. Provided that the transfer relation of the plant is invertible, then a neural network can learn the inverse dynamics accurately and can be used as a controller. But, in general, proving the invertibility of the plant equations requires a highly involved analysis. In this respect, a PI controller can be added to improve the tracking performance.

2. Even in the case that the inverse dynamics of the plant exists, neural network will perform an approximation. Since we can not train the network with infinite number of training pairs with zero mean squared error, there will be an approximation error at the output of the neural identification model and the neural controller. Because of the controller training strategy, the model mismatch in the plant identification model will directly affect the performance of the neural controller.

These two important drawbacks will obviously result in considerable errors in tracking. In this respect, a conventional controller can be used to tolerate the tracking errors such that it does not dominate the control generated by the neurocontroller.

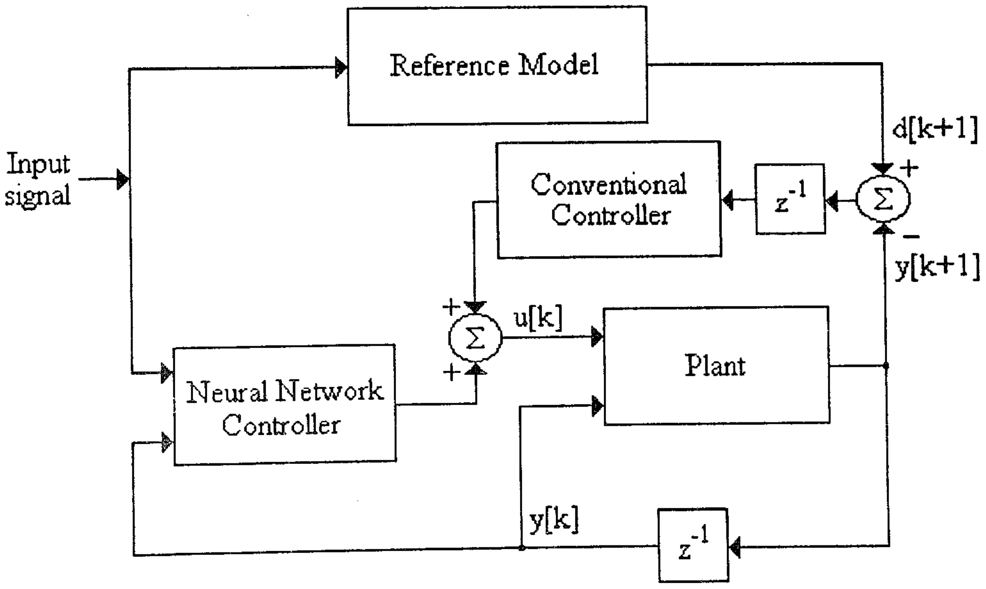


Figure 5.13 Architecture for using a conventional controller with the neurocontroller

We tested the conventional control support for the plant explained in the second simulation. A pulse train and a sinusoidal signal were used as the reference trajectories. A PI controller was used to compensate the errors in the output of the neurocontroller. The pulse transfer function of the digital PI controller is given in (5.4).

$$T(z) = K_P + \frac{K_I}{1-z^{-1}} \tag{5.4}$$

where K_P and K_I are position and integral constants respectively and for the second plant, which is given by (5.2), these constants were chosen to be $K_P = -0.1$ and $K_I = 0.2$.

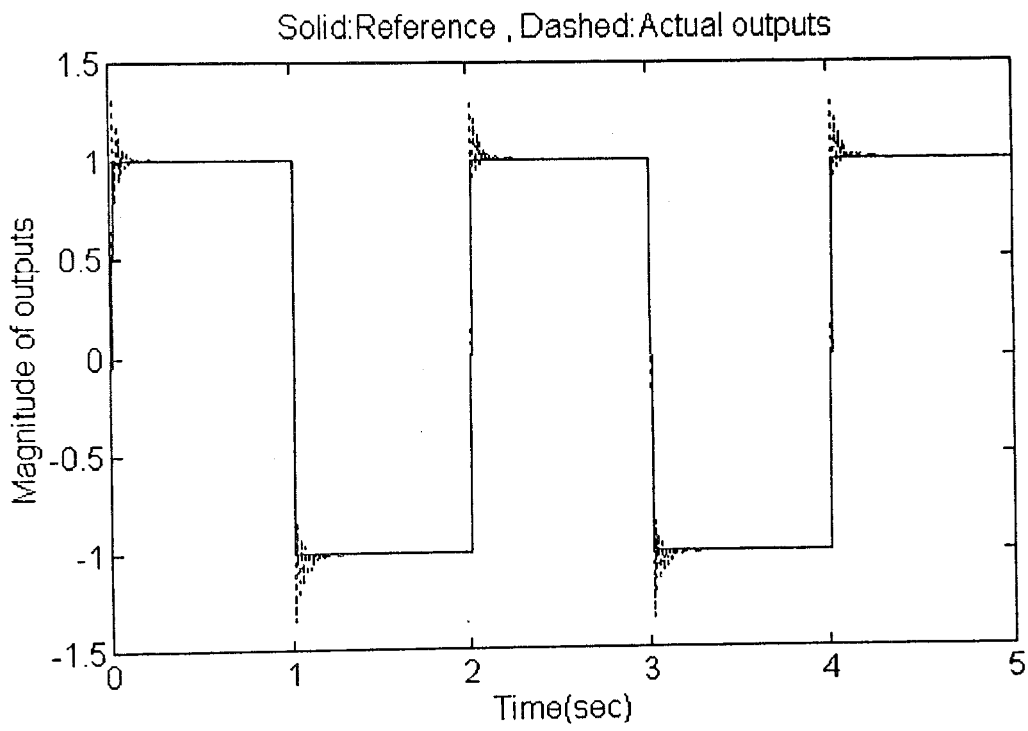


Figure 5.14(a) Reference and actual trajectories when a PI controller was used with the neural controller. Reference input is $r(t) = \text{sgn}[\sin(\pi t)]$

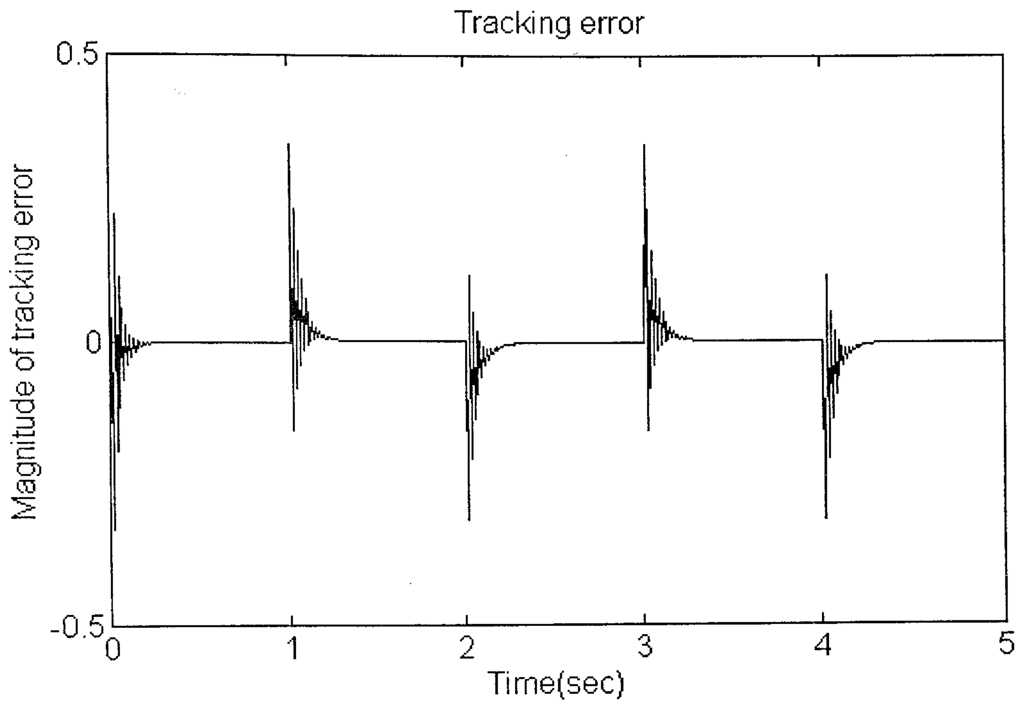


Figure 5.14(b) Tracking error when the PI controller is active. Reference input is r(t) = sgn[sin(πt)]

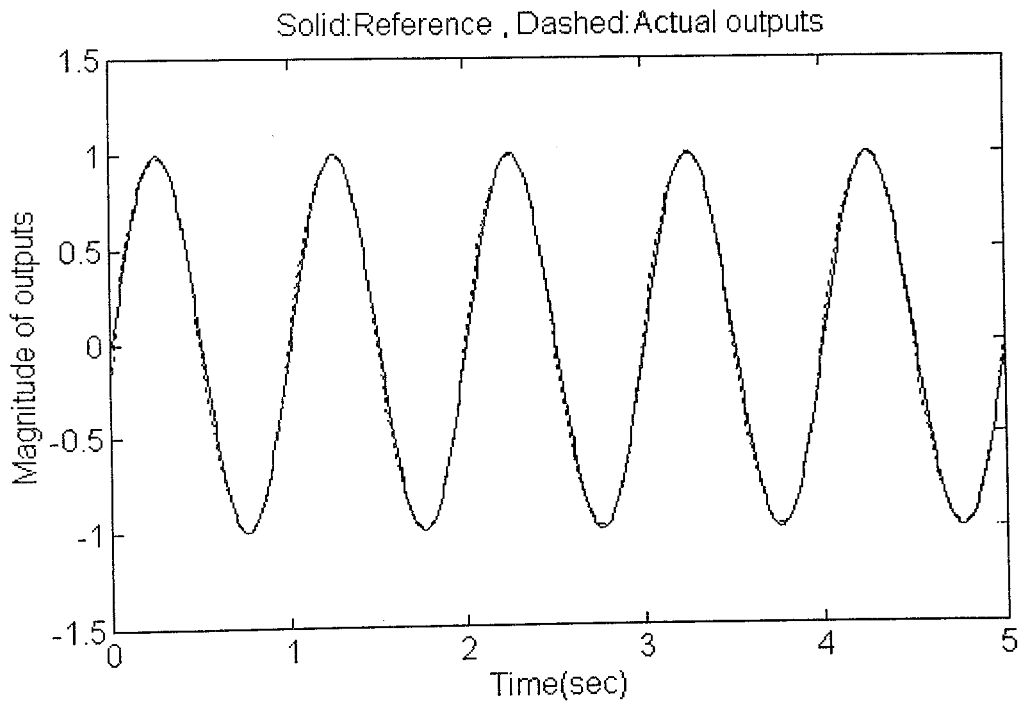


Figure 5.15(a) Reference and actual trajectories when a PI controller was used with the neural controller. Reference input is r(t) = sin(2πt)

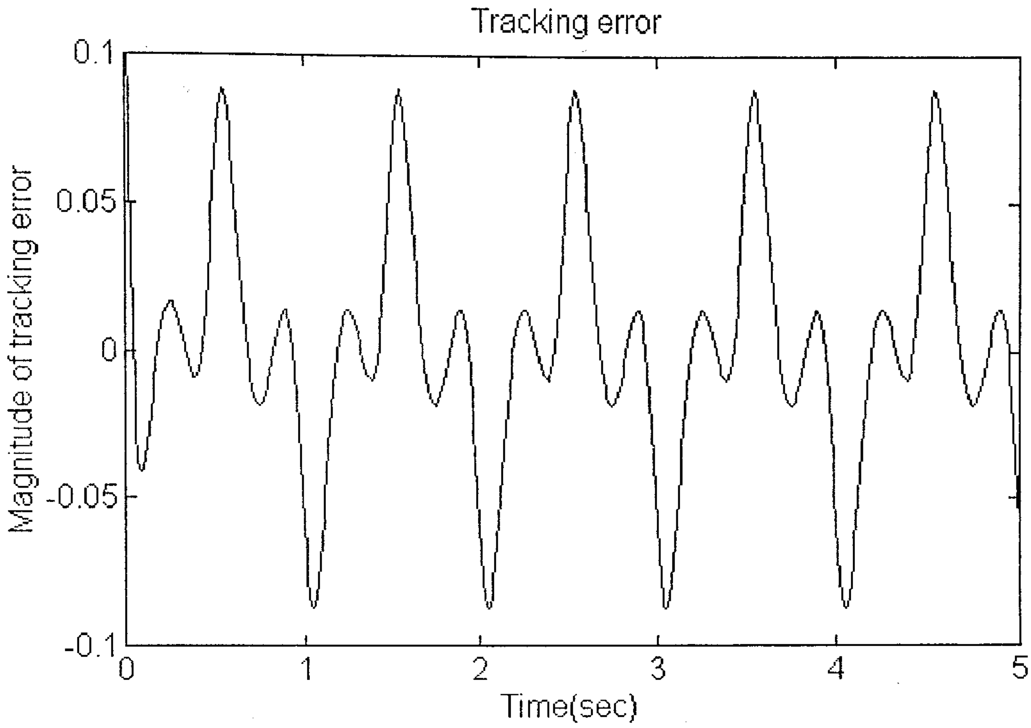


Figure 5.15(b) Tracking error when the PI controller is active. Reference input is $r(t) = \sin(2\pi t)$

As can be seen from Fig. 5.8(a), the actual plant output exhibits a steady state error when a pulse train is used as the reference input. Fig. 5.14(a) illustrates the output of the same simulation with PI control support. Our observations for the architecture shown in Fig. 5.13 revealed that the additional controller improved the steady state performance of the control system. For this test, our second reference was a sinusoidal signal. Fig. 5.9(a) illustrates the result of pure neurocontrol. In this simulation, we observed that the plant output could not reach the peak points of the reference signal. The same simulation was carried out with the PI control support, and it stipulated that the tracking ability is considerably improved around the maximum and the minimum levels of the reference signal.

Briefly, the architecture in Fig. 5.13 improved the tracking performance of the overall system from three points of view : firstly the steady state error is reduced because of the integral control action, secondly, the inaccurate tracking around the peak points of the reference signal is improved, lastly, for a single period of reference signal, the time that the plant output remains at a high error level is reduced.

5.4 Inverse Control Strategy and Application to a Bioreactor Plant

In this approach, the neural controller directly interpolates the inverse dynamics of the plant by using the training data obtained from the plant itself. Assume the governing equation of the plant is given by Equation (5.5)

$$x(k+1) = F\,[x(k), u(k)] \tag{5.5}$$

where $x \in R^m$ and $u \in R^n$, the plant performs a nonlinear mapping from R^{m+n} to R^m. Our objective is solving the plant equation for the input vector.

$$u(k) = G[x(k+1), x(k)] \tag{5.6}$$

If a controller realizes the nonlinear mapping G, then it is able to control the plant. From a systems theoretic point of view, the controller realizes the inverse dynamics of the plant. Given the state values and the desired next state, it generates the input which will cause the desired state transition. In this respect, neural networks come into the picture. As long as the states are observable, we can generate the training data for the neural controller so that it realizes the mapping G which is from R^{2m} to R^n.

We applied this control strategy to a bioreactor plant whose discretized state space representation is given in Equation (5.7) and Equation (5.8). The reader is referred to Appendix A for further details of the bioreactor benchmark problem.

$$c_1(k+1) = c_1(k) + \Delta\left(- c_1(k)\, r(k) + c_1(k)\left(1 - c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \right) \tag{5.7}$$

$$c_2(k+1) = c_2(k) + \Delta\left(-c_2(k)\, r(k) + c_1(k)\left(1 - c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \frac{1+\beta}{1+\beta - c_2(k)} \right) \tag{5.8}$$

The control objective of this problem is keeping the state variable c_1 at a desired level by changing the flow rate over time.

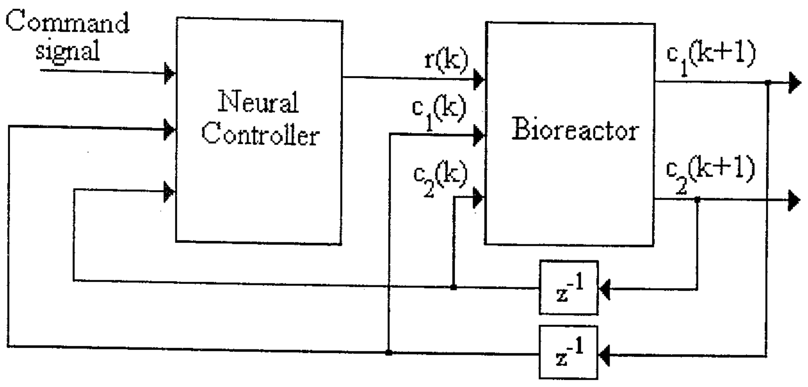


Figure 5.16 Control system architecture for bioreactor plant

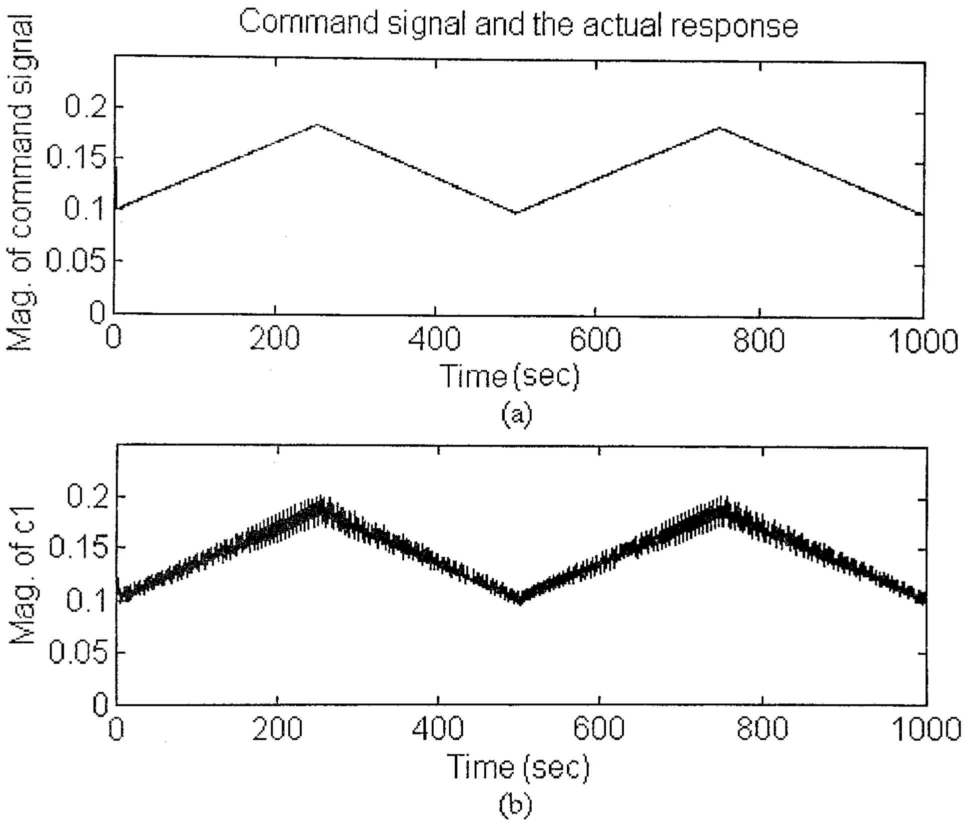


Figure 5.17 (a) Command signal **(b)** Actual c_1 trajectory

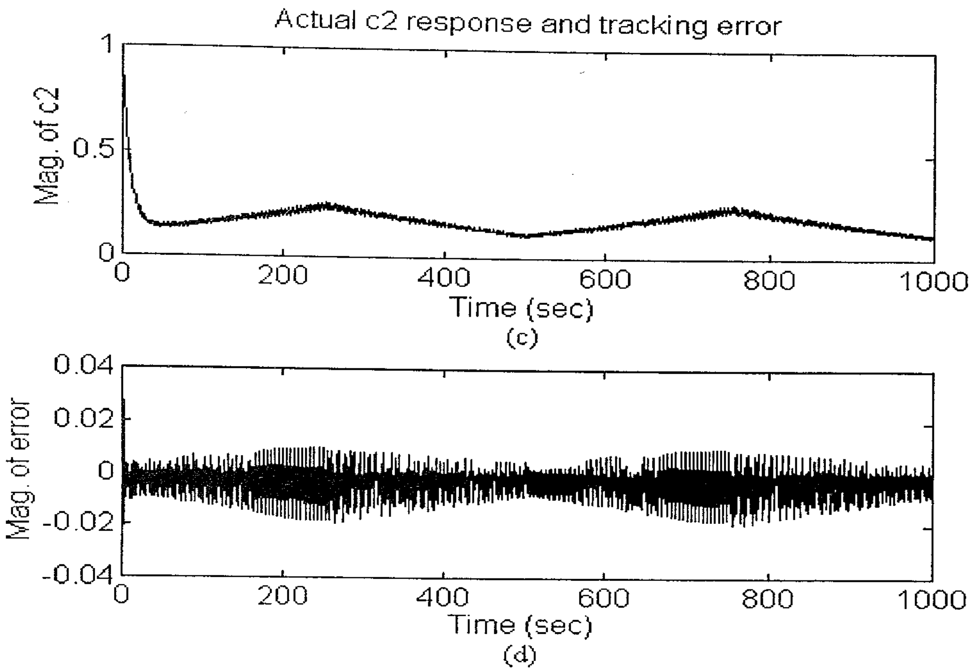


Figure 5.17 (c) Actual c_2 trajectory **(d)** The difference between command signal and actual c_1 value

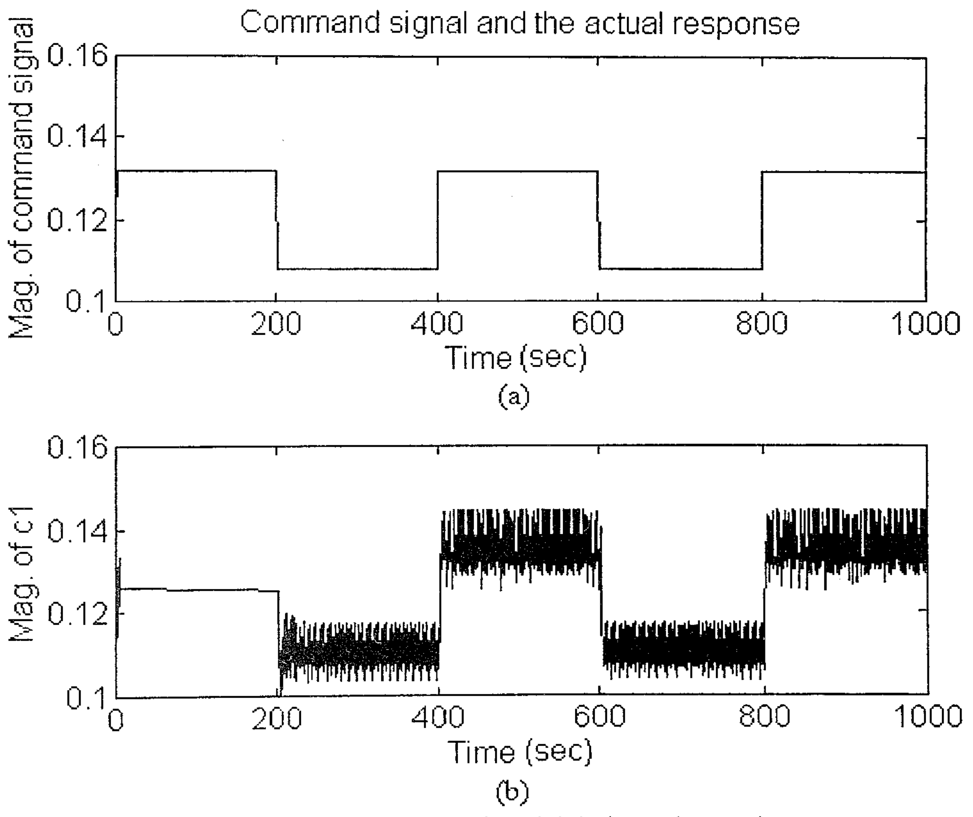


Figure 5.18 (a) Command signal **(b)** Actual c_1 trajectory

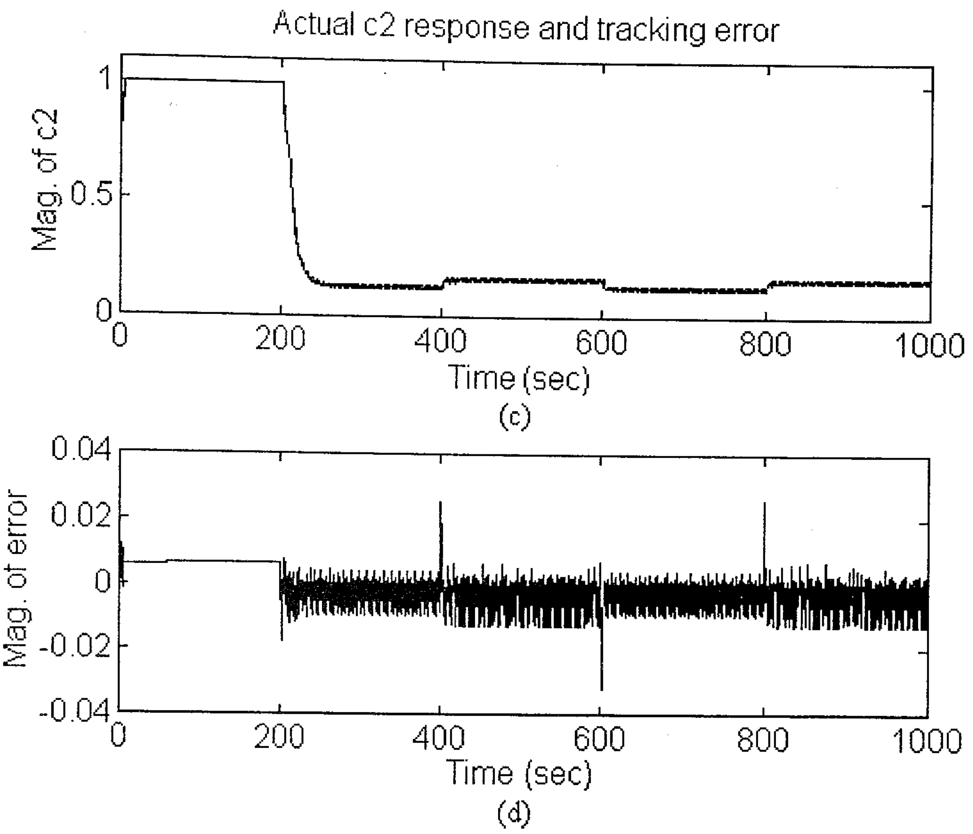


Figure 5.18 (c) Actual c_2 trajectory **(d)** The difference between command signal and actual c_1 value

5.5 Performance Assessment

We trained the neural controller with 670 training pairs until the mean squared error decreased to 1e-4. Besides, the neural network has the topology 3-5-5-1 with linear neurons in the input layer and sigmoidal neurons in the hidden layers and the output layer. Specifically, output neuron has the nonlinearity $F(x)=2/(1+\exp(-x))$ because of the maximum flow rate constraint of the problem.

Our simulations for the bioreactor benchmark problem revealed that, the neural controller is able to control the plant roughly, because there are extremely sharp deviations in the outputs. As mentioned earlier, the inversion of the plant by means of this method is not an efficient way of controller design for this kind of a highly nonlinear plant. Nevertheless, the method is applicable to various types of plants whose inverses can be realized by inverse control strategy.

6. SELF TUNING ADAPTIVE CONTROL USING BACKPROPAGATION NEURAL NETWORKS

Self-tuning controllers automatically adjust, or tune, themselves to achieve prespecified performance objectives related to the system to be controlled. Use of neural networks facilitates dealing with unknown nonlinearities. Additionally, unmodeled dynamics of the plant can be disregarded by utilizing a self-tuning control scheme.

In this chapter, we explain the concept of self tuning adaptive control using neural networks. The approach that we considered was introduced by Chen [4], and it is applicable only to the single-input-single-output and feedback linearizable systems.

6.1 Control Strategy

The governing equation of the plant under control is given by Equation (6.1),

$$\begin{aligned} y(k+1) = {} & f(\, y(k), y(k-1), \ldots, y(k-p), u(k-1), u(k-2), \ldots, u(k-p)\,) + \\ & g(\, y(k), y(k-1), \ldots, y(k-p), u(k-1), u(k-2), \ldots, u(k-p)\,)\, u(k) \end{aligned} \tag{6.1}$$

where y(k) and u(k) denote the output of the system at time k and the input of the system at time k respectively. The algorithm presumes that the function g(.) is a nonzero function. From Equation (6.1) the control at time k can directly be evaluated if the functions f(.) and g(.) are exactly known. This control is given by Equation (6.2).

$$u(k) = -\frac{f(.)}{g(.)} + \frac{d_{k+1}}{g(.)} \tag{6.2}$$

where d_{k+1} is the desired next output of the plant. In what follows, Chen [4] proposed that the functions f (.) and g(.) can be realized by neural networks and the control at time k can be evaluated by using the estimates of these functions as follows;

$$u(k) = -\frac{\hat{f}(.)}{\hat{g}(.)} + \frac{d_{k+1}}{\hat{g}(.)} \tag{6.3}$$

Several past values of the plant inputs and the outputs are provided to the neural network by means of tapped delay lines (TDL), then, the neural network evaluates the estimated values of the functions f(.) and g(.). By the use of these estimates and the target output, as stated in Equation (6.3), the control at time k is evaluated. This control obviously forces the plant to track the target output. Based on the tracking error, the weights and the biases of the neural network are updated using backpropagation technique. The control scheme is illustrated in Fig. 6.1.

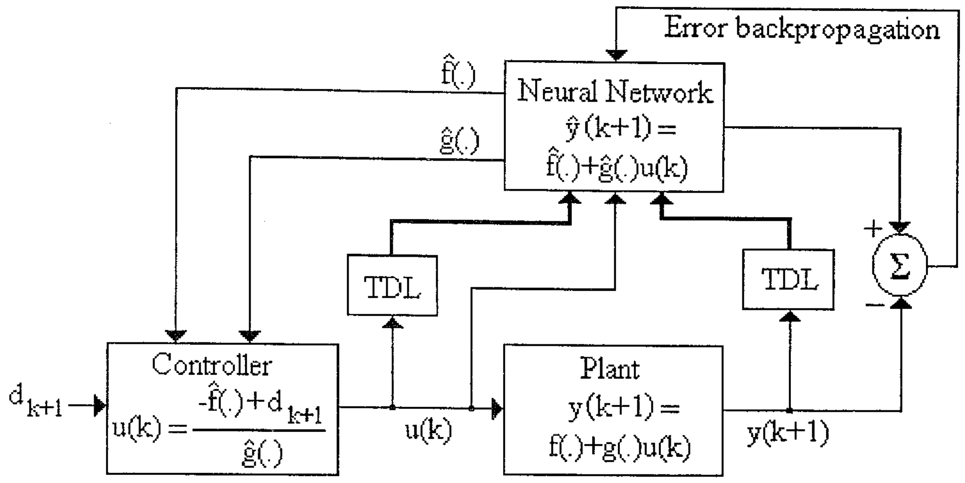


Figure 6.1 Self-tuning control scheme with neural networks

The neural network in Fig. 6.1 is comprised of two sub-networks realizing the functions f(.) and g(.) separately. In our simulations, we restricted our plant model to which is given by Equation (6.4). For this case, structure of the neural network is illustrated in Fig. 6.2.

$$y(k+1) = f(y(k)) + g(y(k))\,u(k) \tag{6.4}$$

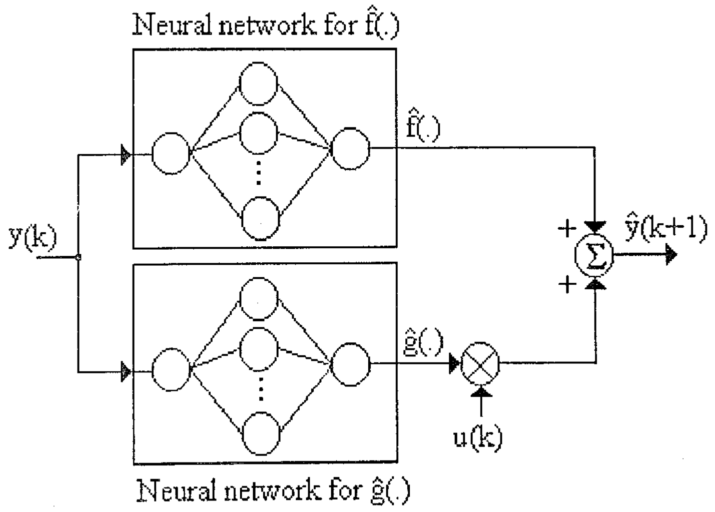

Neural network for $\hat{f}(.)$
$\hat{f}(.)$
y(k)
+
Σ
+
$\hat{y}(k+1)$
$\hat{g}(.)$
u(k)
Neural network for $\hat{g}(.)$

Figure 6.2 Structure of the neural network

In Fig. 6.2, both of these sub-networks have linear neurons in the input and in the output layers, and, neurons possessing sigmoidal nonlinearities in the hidden layers. The performance criterion for this approach is defined as follows;

$$e_{k+1} = d_{k+1} - y(k+1) \tag{6.5}$$

$$E_k = \frac{1}{2} e_{k+1}^2 \tag{6.6}$$

Control objective is based on the minimization of this performance measure. Chen [4] proposed the following update rule for the weights and biases of the neural network.

$$w_f(k+1) = w_f(k) - \eta(k) \frac{\operatorname{sgn}(g(y(k)))}{\hat{g}(y(k), W_F(k))} \frac{\partial \hat{f}(y(k), W_F(k))}{\partial w_f(k)} e_{k+1} \tag{6.7}$$

$$w_g(k+1) = w_g(k) - \mu(k) \frac{\operatorname{sgn}(g(y(k)))}{\hat{g}(y(k), W_G(k))} \frac{\partial \hat{g}(y(k), W_G(k))}{\partial w_g(k)} u(k)\, e_{k+1} \tag{6.8}$$

$$\eta(k) = \mu(k) \frac{\sum_j \left(\frac{\partial \hat{g}(y(k), W_G(k))}{\partial w_g(k)} \right)^2}{\sum_i \left(\frac{\partial \hat{f}(y(k), W_F(k))}{\partial w_f(k)} \right)^2} \tag{6.9}$$

Algorithmically, this control scheme can be summarized as follows:

1. Initialize all of the weights and biases to small random values
2. Initialize the plant output and the control value (we chosen $y(0) = 0.1$, $u(0) = 0.1$)
3. Evaluate neural network outputs
4. Evaluate the control $u(0)$
5. Evaluate the plant output
6. Evaluate the tracking error
7. Assign delta values to the output neurons of the sub-networks
8. Evaluate hidden layer deltas
9. Update weights and biases
10. Present the next desired output d_{k+1} and go to step #3.

6.2 Simulation Results

Simulation 1.

Plant : $y(k+1) = 0.8\sin(2\pi y(k)) + e^{-y^2(k)}u(k)$ (6.10)

Desired trajectory : $r(t) = \begin{cases} \sin(2\pi t/50) & \text{if } 0 \le t \le 50 \\ \operatorname{sgn}(\sin(2\pi t/50)) & \text{if } 50 \le t \le 100 \\ 0 & \text{otherwise} \end{cases}$ (6.11)

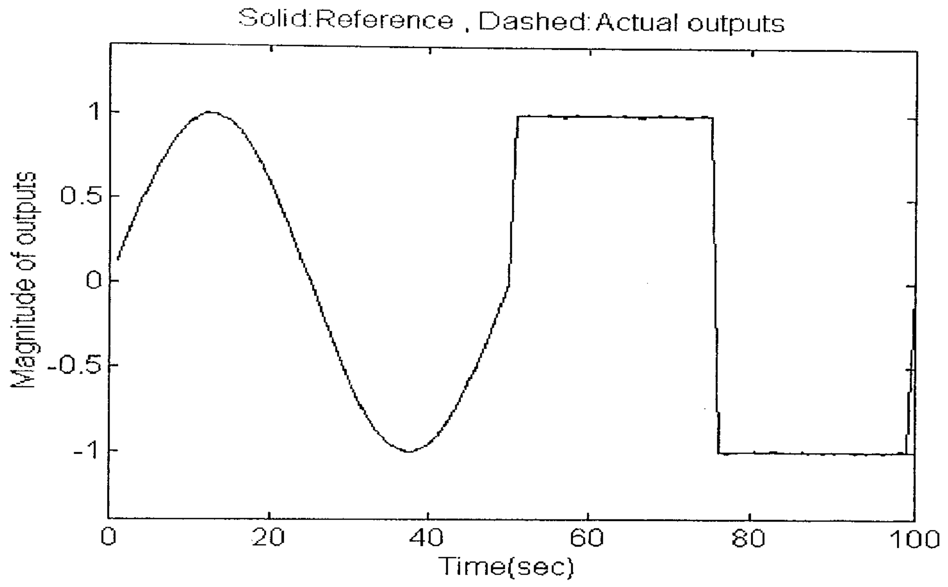


Figure 6.3(a) Output graph for the first plant

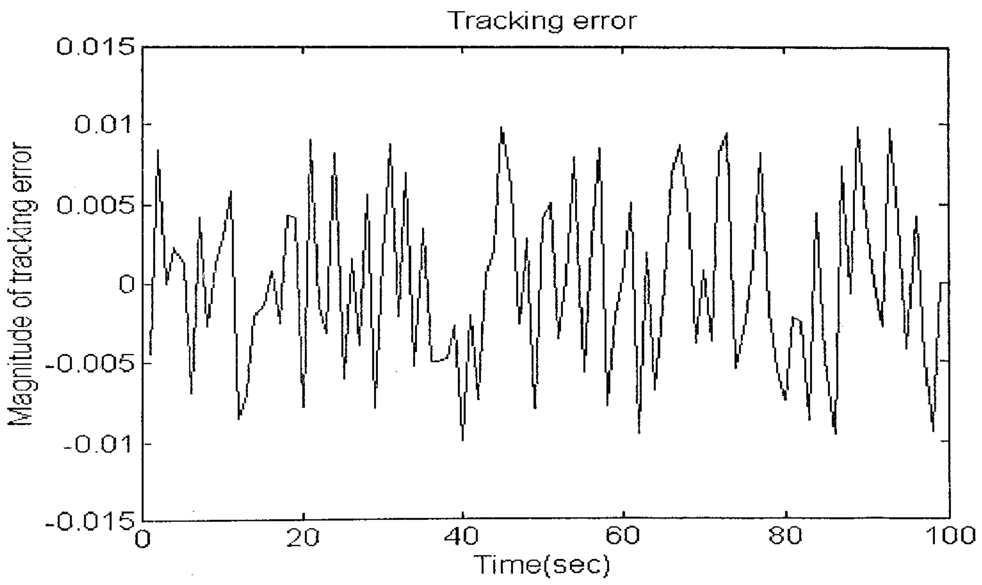


Figure 6.3(b) Tracking error graph for the first plant

Simulation 2.

Plant : $$y(k+1) = \frac{1}{1+y^2(k)} + y(k)u(k) \tag{6.12}$$

Desired trajectory : $$r(t) = \sin(2\pi t/100) \tag{6.13}$$

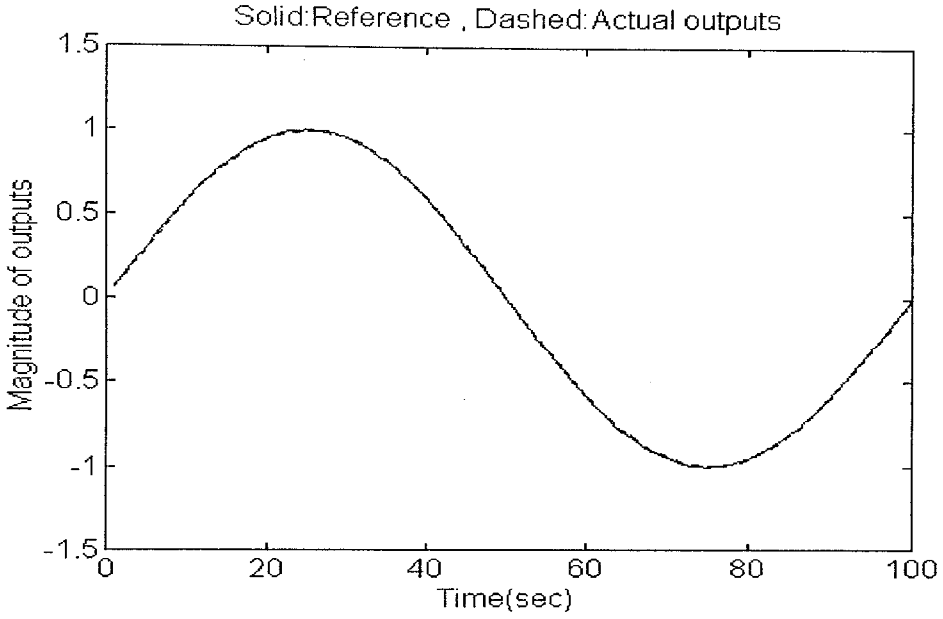


Figure 6.4(a) Output graph for the second plant

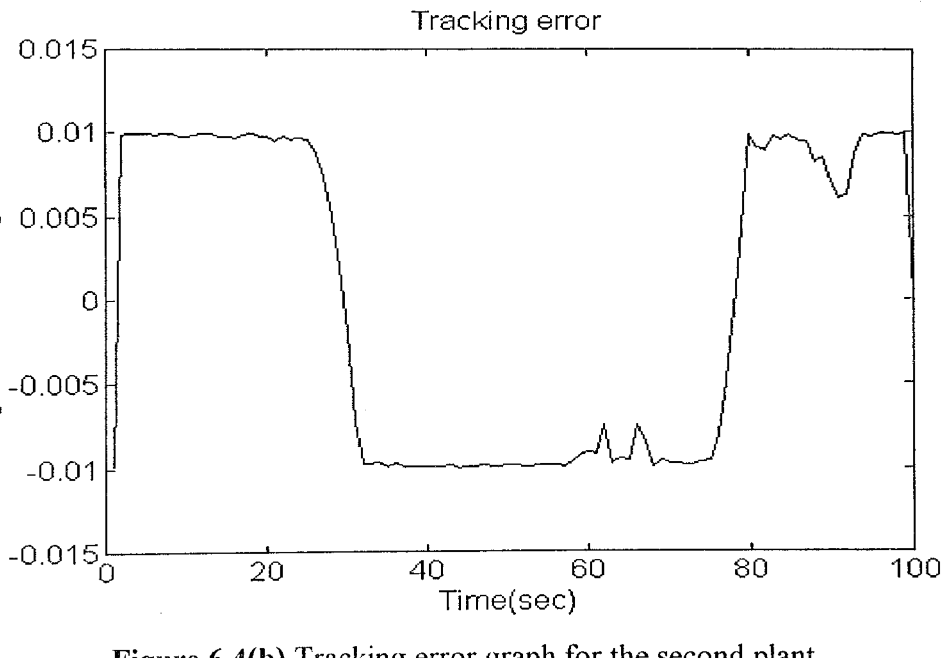


Figure 6.4(b) Tracking error graph for the second plant

Simulation 3.

Plant : $y(k+1) = \sin(y(k))\sin(10y(k)) + 0.08u(k)$ (6.14)

Desired trajectory : $r(t) = \operatorname{sgn}(\sin(2\pi t/10))$ (6.15)

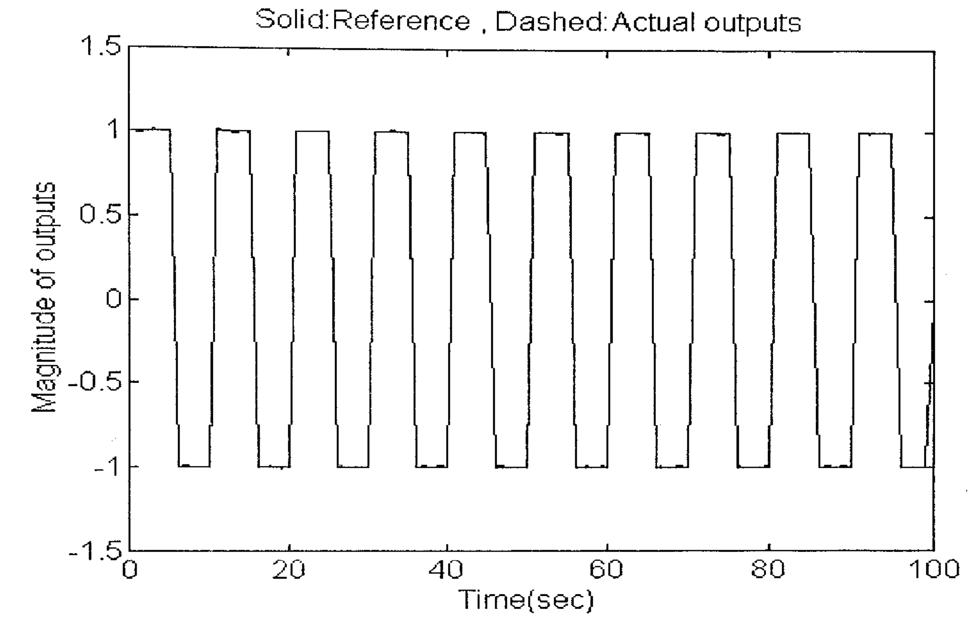


Figure 6.5(a) Output graph for the third plant

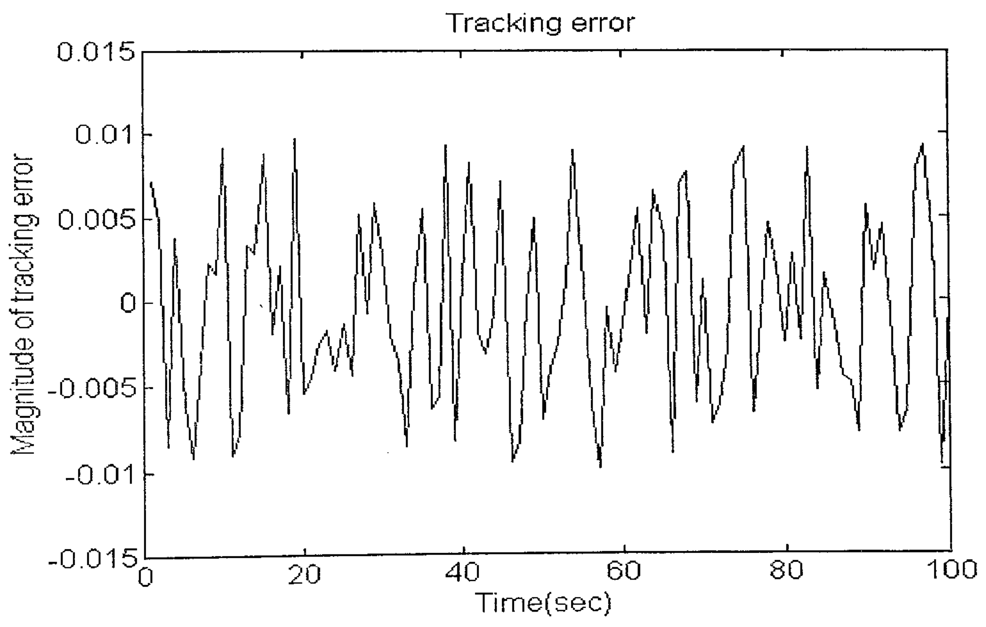


Figure 6.5(b) Tracking error graph for the third plant

6.3 Performance Assessment

The method which was proposed by Chen [4] showed an admissible performance in the sense of tracking ability. Secondly, the approach does not require a detailed plant model, consequently model uncertainties can directly be compensated by neural adaptation.

There are also some important drawbacks of the strategy. The control scheme is applicable solely to the plant model described by Equation (6.1). Besides, the sign of the function g(.) is assumed to be known. Apart from all of these, there is no closed loop stability result. This last drawback, perhaps the most important one, cannot be explained because of the fact that there is no concrete study examining the stability of neural controllers and neural network based control systems.

Briefly, the control strategy is able to tolerate model mismatches but it is not mathematically tractable in the sense of stability analysis.

7. MODEL REFERENCE ADAPTIVE CONTROL USING NEURAL NETWORKS

Model reference adaptive control (MRAC) is another technique that we consider in this thesis. This control scheme generates some controls so that the plant output follows the output of a prespecified stable reference model. In many control problems, the parameters of the plant are not known exactly. In order to compensate the errors stemming from the changes in the parameters of the plant, controller must adapt itself. In this respect, neural networks are proposed as the parameter estimators that utilize the available information dealing with the system and performance objectives. Additionally, as time passes, there must be a convergence on the vector of control parameters. In this chapter, we applied the method to various types of plant models. Narendra and Parthasarathy [5] stipulated a concrete study dealing with MRAC. In our simulations we also duplicated some parts of their work.

7.1 Control Strategy

Model reference adaptive control technique is applicable to wide variety of linear and nonlinear systems. The strategy evaluates some control inputs so that the plant output tracks a stable reference model output. There are two approaches in the control strategy: direct adaptive control and indirect adaptive control. These approaches are illustrated in Fig. 7.1 and Fig 7.2.

Direct adaptive control scheme utilizes the instantaneous tracking error, which is denoted by e_c, in parameter updating. Several past control inputs and plant outputs are provided by tapped delay lines. Narendra [5] in his work points out that the parameters of the controller are directly adjusted to reduce some norm of the output error.

Indirect adaptive control scheme employs an additional plant identification model which provides the information about the nonlinear functions in the governing equations of the plant. The identification is carried out on-line.

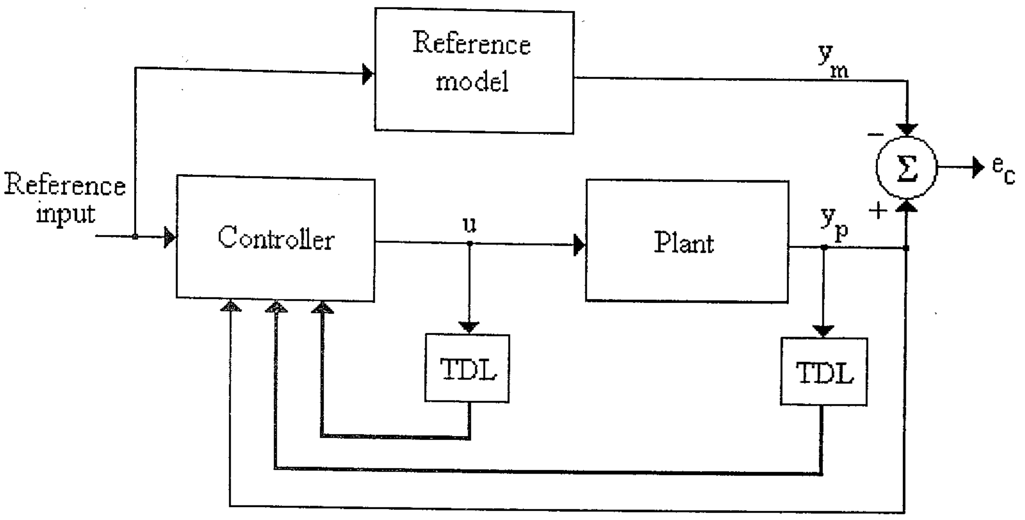


Figure 7.1 Direct adaptive control scheme

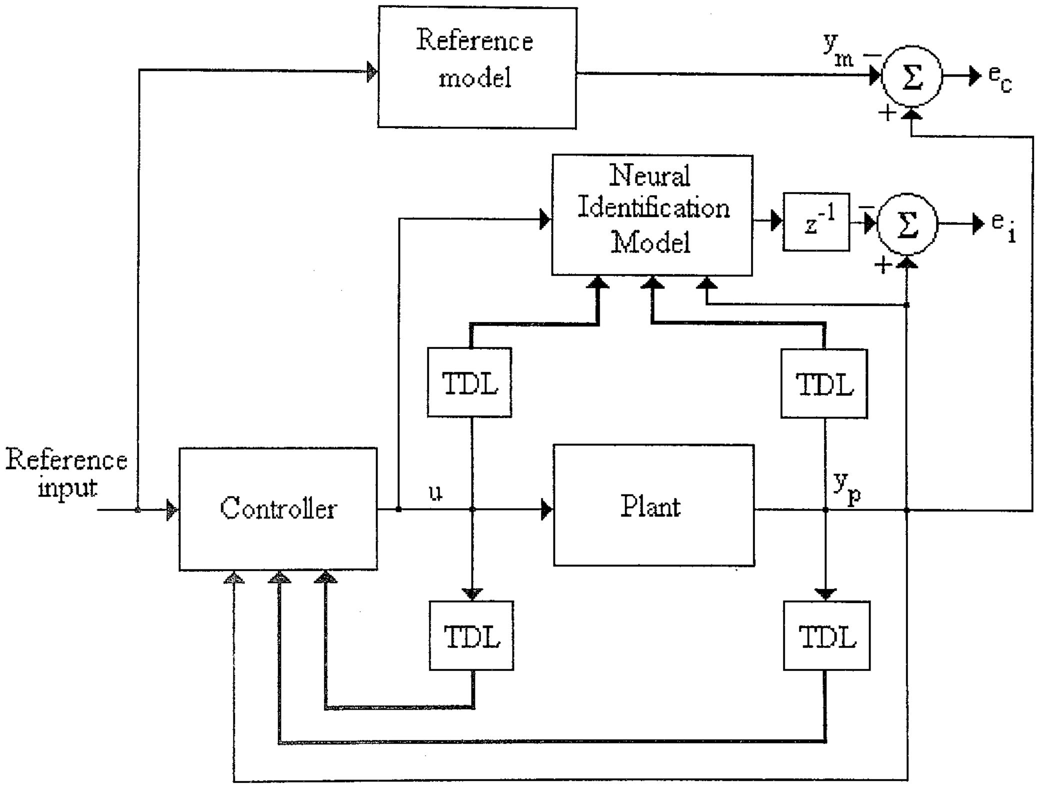


Figure 7.2 Indirect adaptive control scheme

7.2 Simulation Results

In the first simulation we applied the control strategy to the plant defined by Equation (7.1) and the stable reference model defined by Equation (7.2). We applied the reference input which is given by Equation (7.3).

Plant : $$y_p(k+1) = \frac{y_p(k)y_p(k-1)\left(y_p(k)+2.5\right)}{1+y_p^2(k)+y_p^2(k-1)} + u(k) \qquad (7.1)$$

Model : $$y_m(k+1) = 0.6y_m(k) + 0.2y_m(k-1) + r(k) \qquad (7.2)$$

Reference input : $$r(t) = 0.5\sin(0.07t) \qquad (7.3)$$

The plant belongs to the second representation which is explained in the fourth chapter and defined by Narendra [5]. Our aim is to derive a control law so that the plant behaves as the model. We can consider the plant equation as $y_p(k+1) = f(y_p(k),y_p(k-1)) + u(k)$. As long as the relation f is known accurately, the control at time k can be solved from the equation of the plant as follows;

$$u(k) = y_p(k+1) - f(y_p(k),y_p(k-1)) \qquad (7.4)$$

Under the assumption that the plant output follows the reference model output y_m , $y_p(k+1)$ can be replaced with $y_m(k+1)$.

$$u(k) = 0.6y_p(k) + 0.2y_p(k-1) + r(k) - f(y_p(k),y_p(k-1)) \qquad (7.5)$$

If this u(k) is written into Equation (7.1), plant equation turns out to be as follows;

$$y_p(k+1) = 0.6y_p(k) + 0.2y_p(k-1) + r(k) \qquad (7.6)$$

From this point, we have to prove that if the control given by Equation (7.5) is applied to the plant, the error between the reference model output and the plant output must go to zero in the limiting case. In this respect we introduce the neural networks into the control strategy. If a neural estimator provides the value of the function f(.), then this value can be used to evaluate the control at time k which is stated in Equation (7.5). Let $N(y_p(k),y_p(k-1))$ denote the approximate value of $f(y_p(k),y_p(k-1))$;

$$y_p(k+1) - y_m(k+1) = f(y_p(k),y_p(k-1)) + u(k) - 0.6y_m(k) - 0.2y_m(k-1) - r(k) \tag{7.7}$$

If u(k) in Equation (7.7) is evaluated by using Equation (7.8) we end up with Equation (7.9).

$$u(k) = 0.6y_p(k) + 0.2y_p(k-1) + r(k) - N(y_p(k),y_p(k-1)) \tag{7.8}$$

$$e_c(k+1) = 0.6e_c(k) + 0.2e_c(k-1) + f(y_p(k),y_p(k-1)) - N(y_p(k),y_p(k-1)) \tag{7.9}$$

$$e_c(k+1) = y_p(k+1) - y_m(k+1) \tag{7.10}$$

In Equation (7.9), the forcing term comes from the difference between the actual f value and the estimated f value. As long as the neural network performs a precise approximation, effect of this term can be neglected.

In this simulation we used a neural network with the structure 2-8-4-1. Input and output layer neurons are linear and hidden layers' neurons are sigmoidal. The network was trained until the mean squared error decreased to 45e-5. We used Levenberg-Marquardt method for the neural network training.

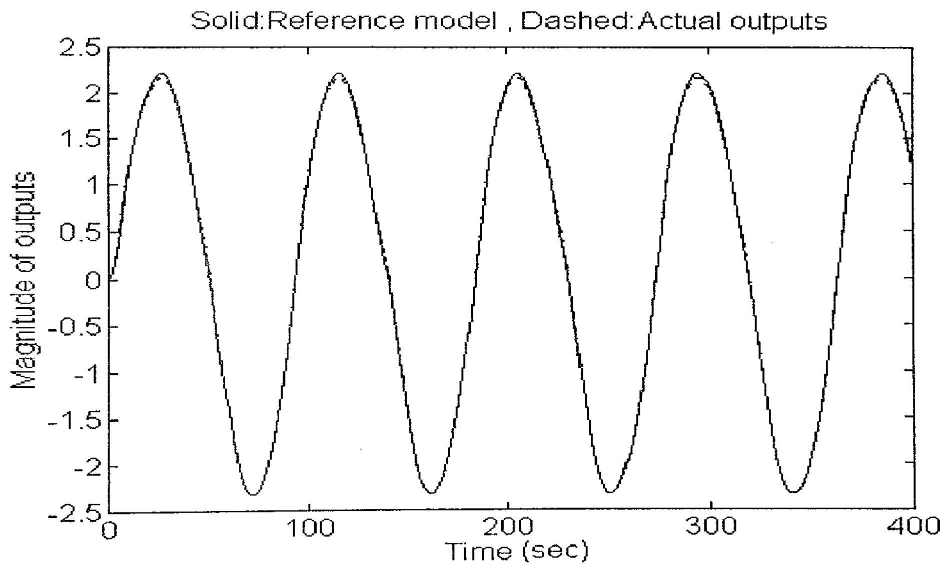


Figure 7.3(a) Reference model and the actual plant outputs which are defined by Equations (7.2) and (7.1) respectively (Reference input is sinusoidal)

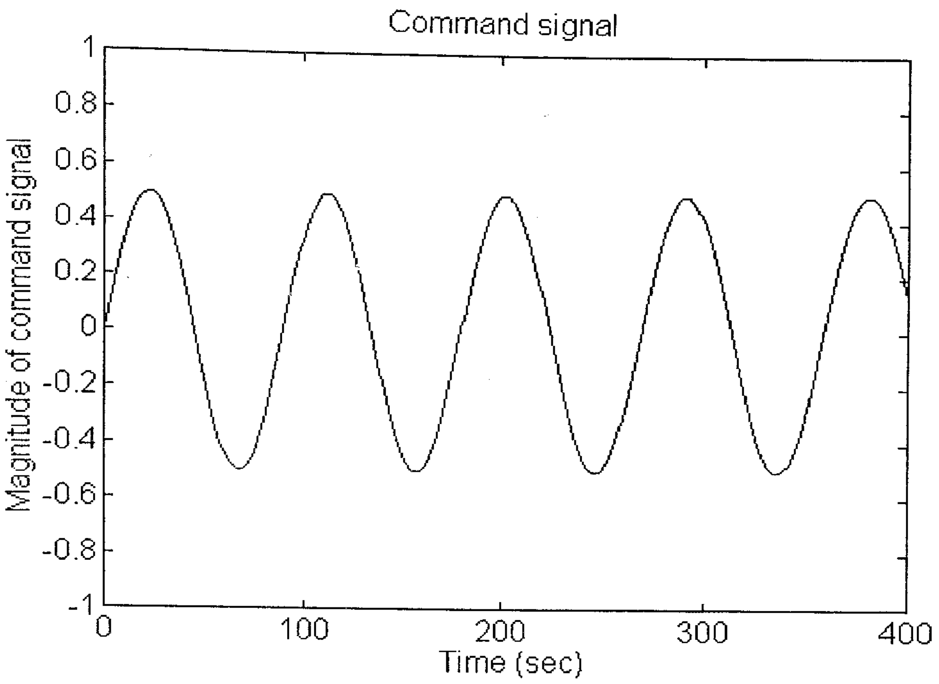


Figure 7.3(b) Applied reference input to the system

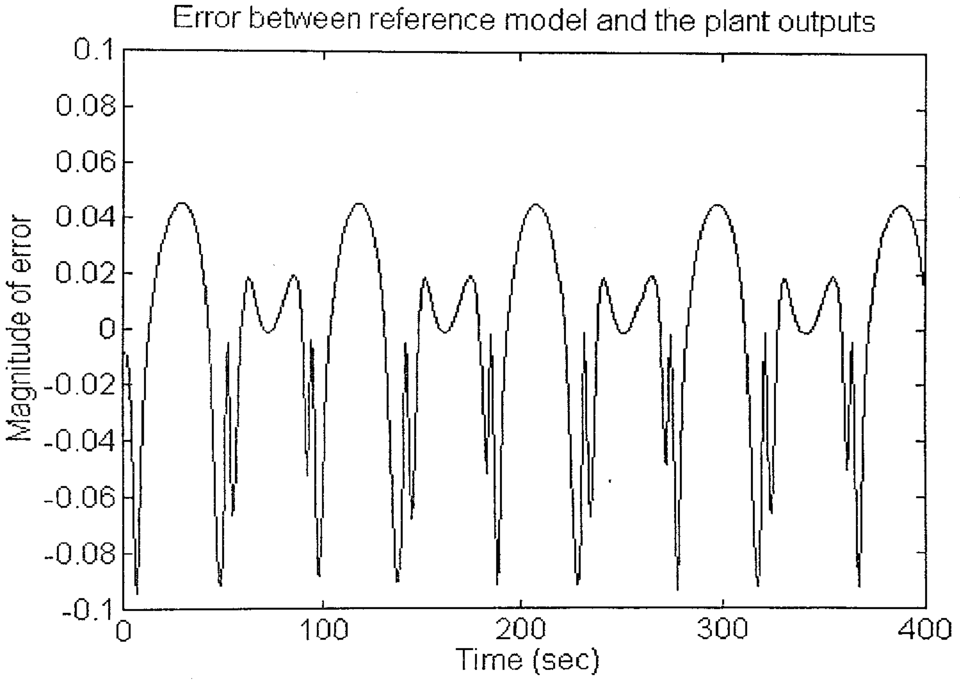


Figure 7.3(c) Error graph for the first reference input defined by Equation (7.3) and illustrated in Fig. 7.3(b)

In the second simulation of the same plant, we applied the reference input, which is defined by Equation (7.11), to the system.

Reference input : $r(t) = 0.5\mathrm{sgn}(\sin(2\pi t/200))$ (7.11)

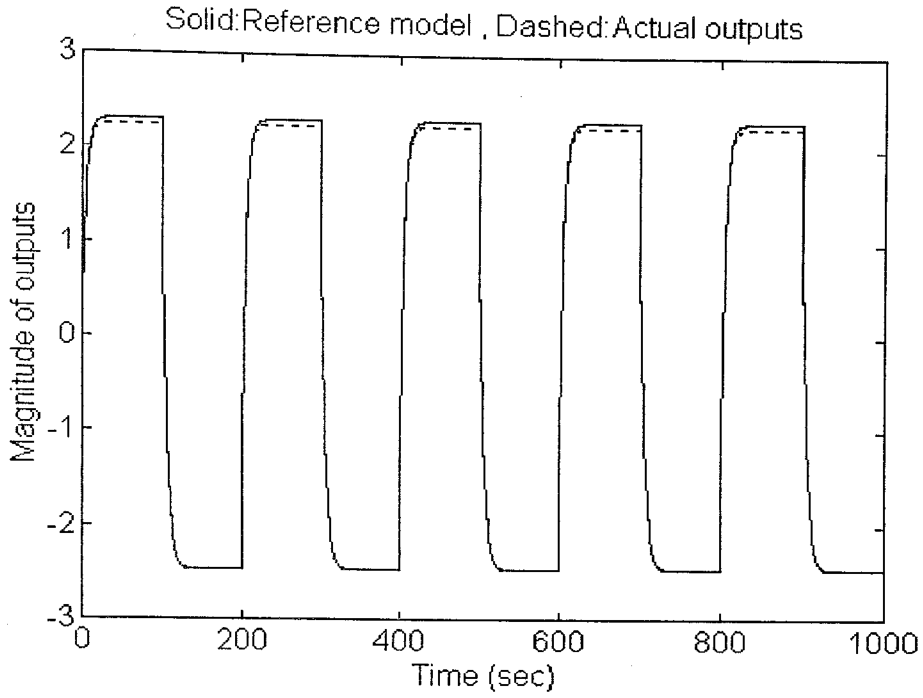


Figure 7.4(a) Reference model and the actual plant outputs which are defined by Equations (7.2) and (7.1) respectively (Reference input is a pulse train)

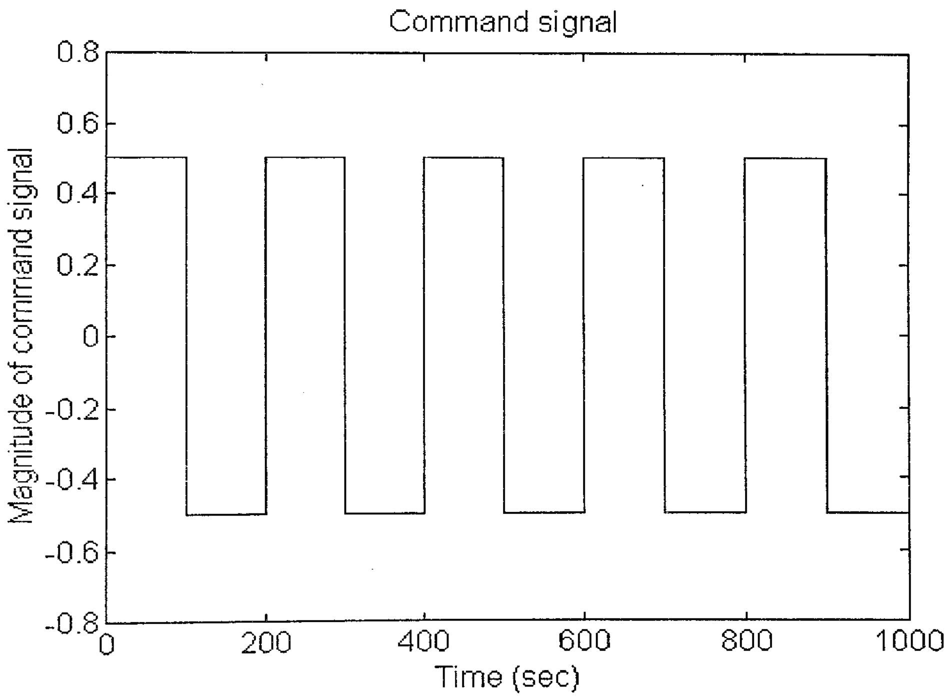


Figure 7.4(b) Applied reference input to the system

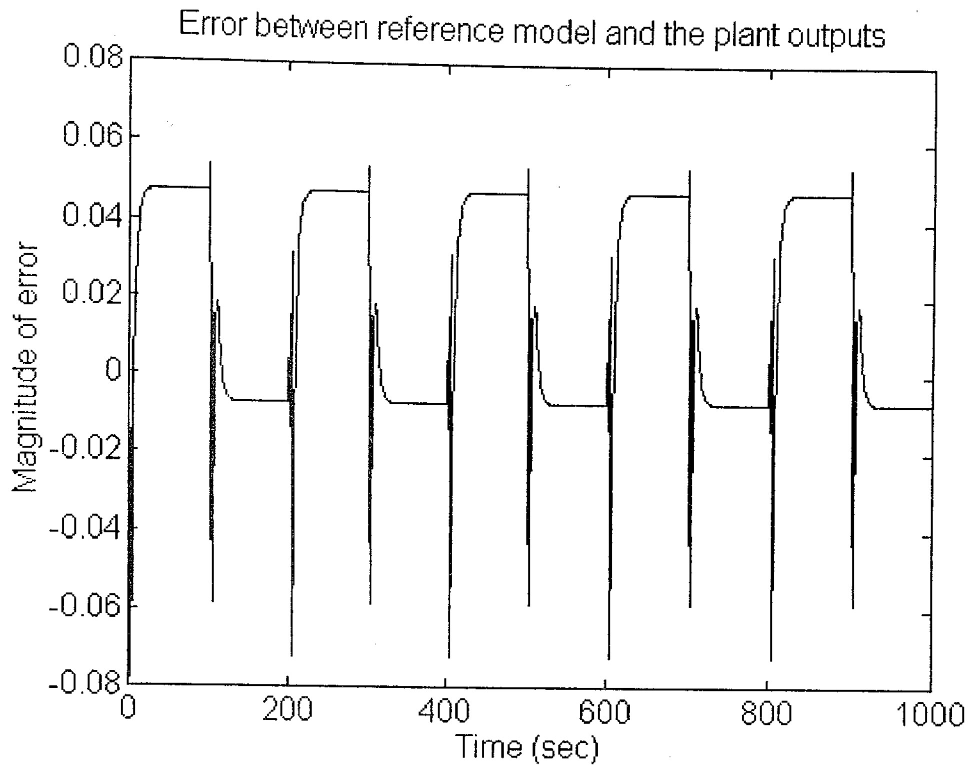


Figure 7.4(c) Error graph for the second reference input defined by Equation (7.11) and illustrated in Fig. 7.4(b)

We carried out the third simulation with a MIMO plant which is used in [5]. If the same analysis explained for the first plant is applied to the plant defined by Equation (7.12), again, a stable control law is obtained.

Plant
$$\begin{bmatrix} y_{p1}(k+1) \\ y_{p2}(k+1) \end{bmatrix} = \begin{bmatrix} \dfrac{y_{p1}(k)}{1+y_{p2}^2(k)} \\ \dfrac{y_{p1}(k)y_{p2}(k)}{1+y_{p2}^2(k)} \end{bmatrix} + \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \tag{7.12}$$

Model
$$\begin{bmatrix} y_{m1}(k+1) \\ y_{m2}(k+1) \end{bmatrix} = \begin{bmatrix} 0.6 & 0.2 \\ 0.1 & -0.8 \end{bmatrix} \begin{bmatrix} y_{m1}(k) \\ y_{m2}(k) \end{bmatrix} + \begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix} \tag{7.13}$$

Reference input 1 : $r_1(t) = \sin(2\pi t/200)$ (7.14)

Reference input 2 : $r_2(t) = \cos(2\pi t/200)$ (7.15)

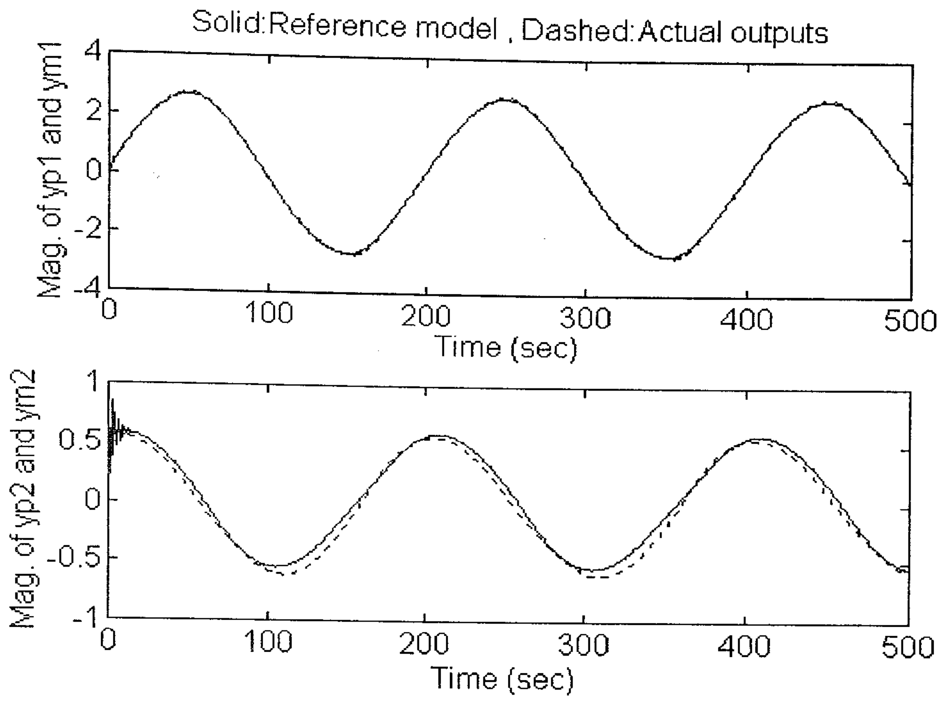


Figure 7.5(a) Reference model and the actual plant outputs which are defined by Equations (7.13) and (7.12) respectively (Reference inputs are sinusoidal)

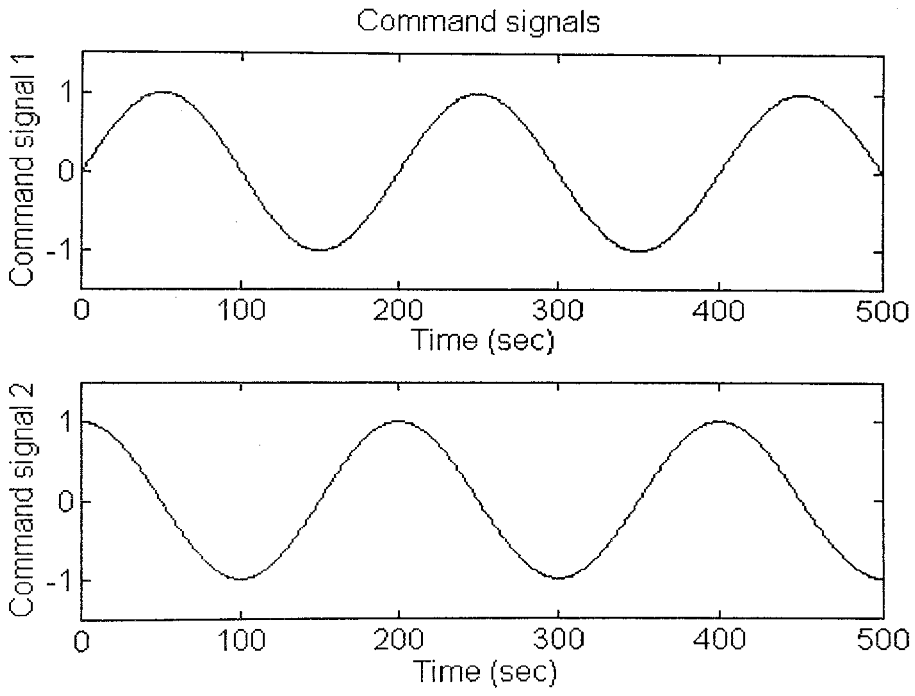


Figure 7.5(b) Applied reference inputs to the system

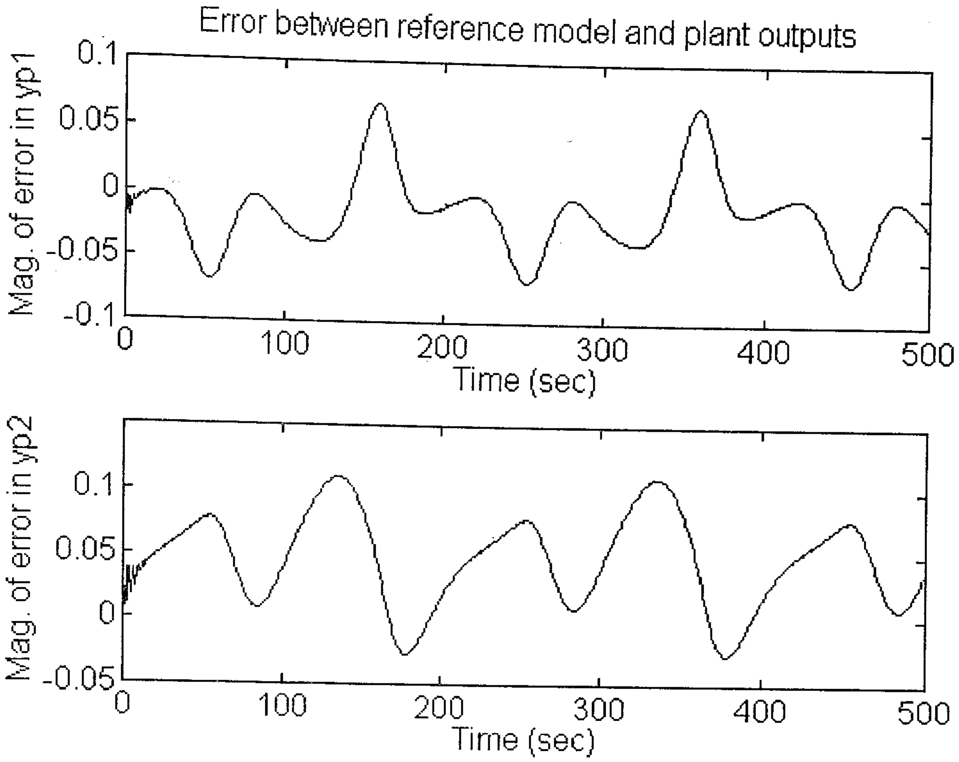


Figure 7.5(c) Error graph for the first reference input group defined by Equations (7.14) and (7.15) and are illustrated in Fig. 7.5(b)

In the fourth simulation reference inputs are changed to pulse trains which are given by Equations (7.16) and (7.17).

Reference input 1 : $r_1(t) = \mathrm{sgn}(\sin(2\pi t/200))$ (7.16)

Reference input 1 : $r_2(t) = 2\mathrm{sgn}(\sin(2\pi t/200))$ (7.17)

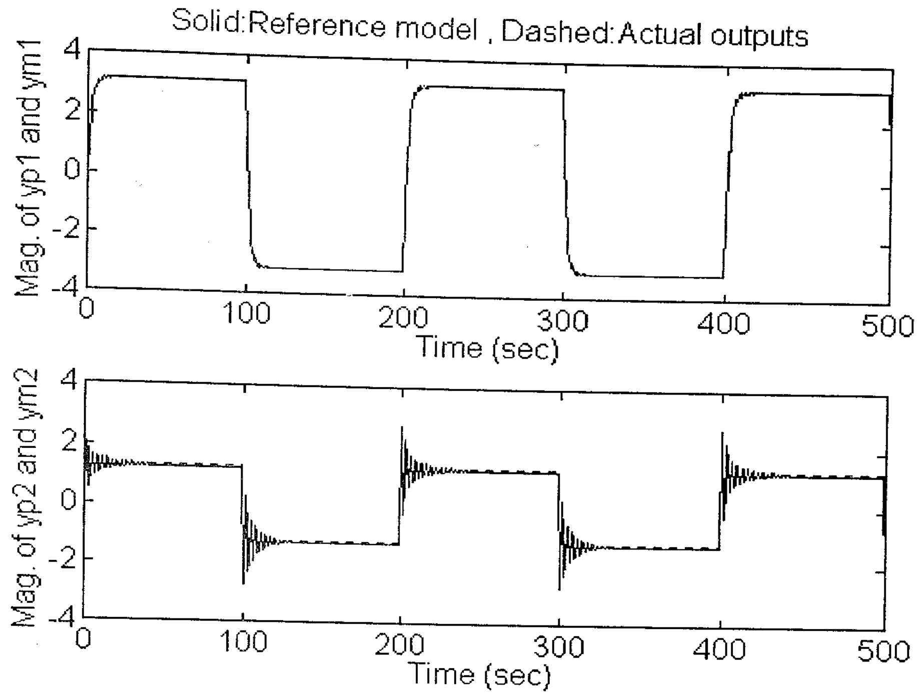


Figure 7.6(a) Reference model and the actual plant outputs which are defined by Equations (7.13) and (7.12) respectively (Reference inputs are pulse trains)

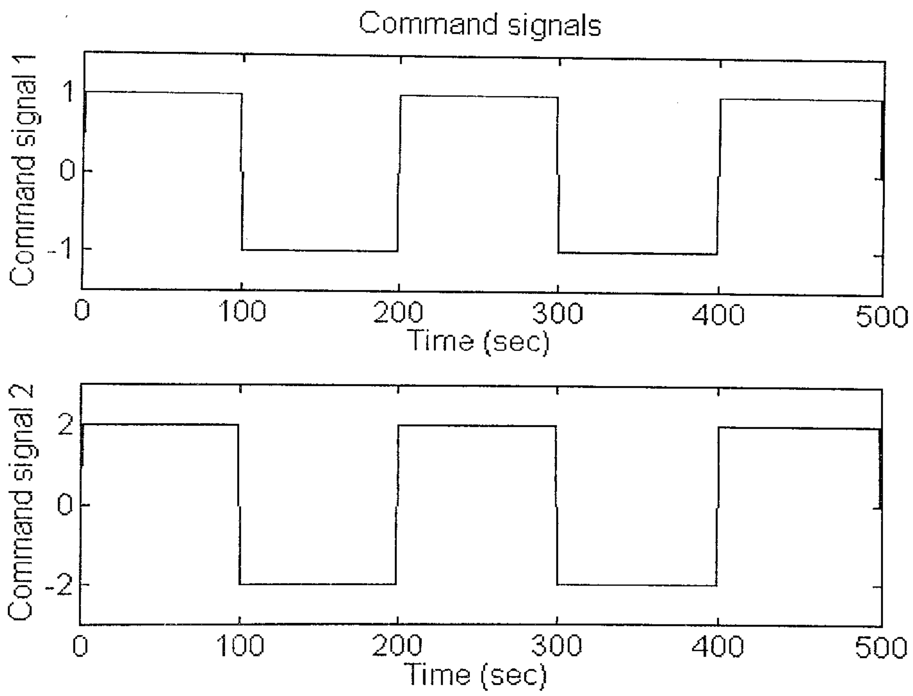


Figure 7.6(b) Applied reference inputs to the system

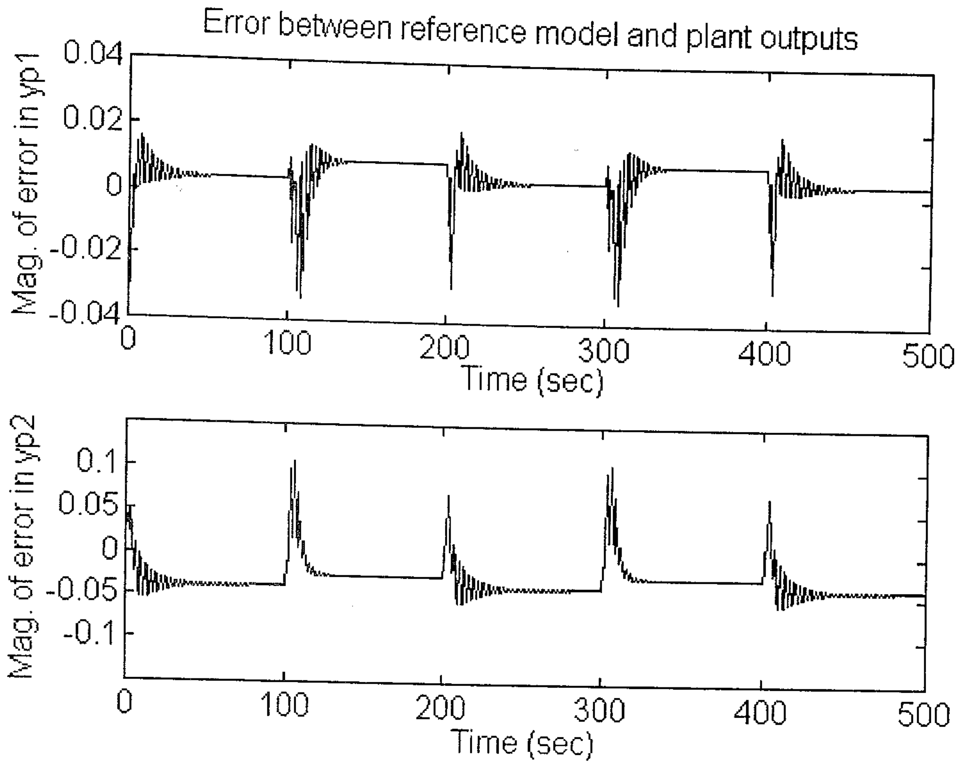


Figure 7.6(c) Error graph for the first reference input group defined by Equations (7.16) and (7.17) and are illustrated in Fig. 7.6(b)

Our fifth and sixth simulations are comprised of controlling the bioreactor model which is given in Appendix A with the MRAC strategy. For this plant we will derive the control law once more. The continuous time differential equations of the bioreactor model are given by Equations (7.18) and (7.19).

$$\dot{c}_1(t) = -c_1(t)w(t) + c_1(t)\big(1-c_2(t)\big)\, e^{\frac{c_2(t)}{\gamma}} \tag{7.18}$$

$$\dot{c}_2(t) = -c_2(t)w(t) + c_1(t)\big(1-c_2(t)\big)\, e^{\frac{c_2(t)}{\gamma}} \frac{1+\beta}{1+\beta - c_2(t)} \tag{7.19}$$

In Equations (7.18) and (7.19) β and γ are constants. Let's drop time parentheses and define $f(c_1,c_2)$ and $g(c_2)$ and rewrite these equations as follows;

$$\dot{c}_1 = -c_1 w + f(c_1,c_2) \tag{7.20}$$

$$\dot{c}_2 = -c_2 w + f(c_1,c_2)g(c_2) \tag{7.21}$$

where

$$f(c_1,c_2) = c_1(1-c_2)\,e^{\frac{c_2}{\gamma}} \tag{7.22}$$

$$g(c_2) = \frac{1+\beta}{1+\beta - c_2} \tag{7.23}$$

From this point of view, we will construct the reference model and explain the philosophy that lies behind this choice. Firstly, $c_{1m}(t)$ must follow command signal r(t); when $c_{1m}(t) = r(t)$ at a moment t_0 (i.e. if c_{1m} catches r) for all $t > t_0$ Equation (7.24) is satisfied.

$$\dot{c}_{1m}(t) = -c_{1m}(t) + r(t) \tag{7.24}$$

Secondly, since c_{2m} is a free-state variable, the conditions imposed by c_{2m} on the system must be redundant, equivalently c_{2m} must be function of c_{1m}. Hence, at the differential level c_{2m} is linearly dependent on c_{1m}.

$$c_{2m}(t) = c_{2m}(c_{1m}(t)) \tag{7.25}$$

$$\dot{c}_{2m}(t) = \frac{\partial c_{2m}(t)}{\partial c_{1m}(t)}\,\dot{c}_{1m}(t) = \frac{\partial c_{2m}(t)}{\partial c_{1m}(t)}\left(-c_{1m}(t) + r(t)\right) \tag{7.26}$$

If Equation (7.26) is arranged;

$$\dot{c}_{2m}(t) = -\frac{\partial c_{2m}(t)}{\partial c_{1m}(t)}\,c_{1m}(t) + \frac{\partial c_{2m}(t)}{\partial c_{1m}(t)}\,r(t) \tag{7.27}$$

Now, we interpret the terms in Equation (7.27) as follows;

$$-\frac{\partial c_{2m}}{\partial c_{1m}}\,c_{1m} = -c_{2m} \tag{7.28}$$

$$\frac{\partial c_{2m}}{\partial c_{1m}}\,r = g(c_{2m})\,r \tag{7.29}$$

Note that, in our model $g(c_{2m})$ is defined as follows;

$$g(c_{2m}) = \frac{c_{2m}}{c_{1m}} = \frac{1+\beta}{1+\beta-c_{2m}} \tag{7.30}$$

This analysis imposes the model defined by Equations (7.31) and (7.32);

$$\dot{c}_{1m}(t) = -c_{1m}(t) + r(t) \tag{7.31}$$

$$\dot{c}_{2m}(t) = -c_{2m}(t) + g\left(c_{2m}(t)\right)r(t) \tag{7.32}$$

Dynamics of the bioreactor plant is characterized by Equations (7.33) and (7.34).

$$\dot{c}_1(t) = -c_1(t)w(t) + f\left(c_1(t), c_2(t)\right) \tag{7.33}$$

$$\dot{c}_2(t) = -c_2(t)w(t) + f\left(c_1(t), c_2(t)\right)g\left(c_2(t)\right) \tag{7.34}$$

where,

$$f\left(c_1(t), c_2(t)\right) = c_1(t)\left(1-c_2(t)\right) e^{\frac{c_2(t)}{\gamma}} \tag{7.35}$$

$$g\left(c_2(t)\right) = \frac{1+\beta}{1+\beta-c_2(t)} \tag{7.36}$$

Equations (7.33) and (7.34) can be rewritten as follows;

$$\dot{c}_1(t) = -c_1(t)\left(w(t) - \frac{f\left(c_1(t), c_2(t)\right)}{c_1(t)}\right) \tag{7.37}$$

$$\dot{c}_2(t) = -c_2(t)\left(w(t) - \frac{f\left(c_1(t), c_2(t)\right) g\left(c_2(t)\right)}{c_2(t)}\right) \tag{7.38}$$

Since $c_1(t)$ and $c_2(t)$ are nonzero, Equations (7.37) and (7.38) are valid.

Proposition

There exists a function $F(c_1, c_2, w(c_1, c_2, r))$ such that,

$$\dot{c}_1(t) = -c_1(t)\, F\left(c_1(t)\, ,\, c_2(t)\, ,\, w\left(c_1(t)\, ,\, c_2(t)\, ,\, r(t)\right)\right) + r(t) \tag{7.39}$$

$$\dot{c}_2(t) = -c_2(t)\,F\big(c_1(t), c_2(t), w(c_1(t), c_2(t), r(t))\big) + g(c_2(t))\,r(t) \tag{7.40}$$

Proof

We know that the variable w is the control input and is a function of state variables $(c_1(t),c_2(t))$ and command signal r(t).

$$-c_1(t)\,F\big(c_1(t), c_2(t), w(c_1, c_2, r)\big) + r(t) = -c_1(t)\,w(c_1, c_2, r) + f(c_1(t),c_2(t)) \tag{7.41}$$

$$F\big(c_1(t), c_2(t), w(c_1, c_2, r)\big) = w(c_1, c_2, r) + \frac{r(t)}{c_1(t)} - \frac{f(c_1(t),c_2(t))}{c_1(t)} \tag{7.42}$$

Since the control input w depends continuously and differentially on r(t) with the condition $\partial w/\partial r \neq 0$ for all t, by the inverse function theorem $r = r(c_1,c_2,w)$. Hence, Equation (7.42) is trivially true. Putting F in Equation (7.40), we obtain;

$$\dot{c}_2(t) = -c_2(t)w - \frac{c_2(t)}{c_1(t)}\,r(c_1,c_2,w) - \frac{c_2(t)}{c_1(t)}\,f(c_1,c_2) + g(c_2)\,r(c_1,c_2,w) \tag{7.43}$$

$$\dot{c}_2(t) = -c_2(t)w - r(c_1,c_2,w)\left(g(c_2) - \frac{c_2(t)}{c_1(t)}\right) - \frac{c_2(t)}{c_1(t)}\,f(c_1,c_2) \tag{7.44}$$

Now, we claim that, each choice of control input with a given command signal can be considered as a constant value of the function F, namely, $F(c_1,c_2,w(c_1,c_2,r(t))) = K$ where K is a constant. This is equivalent to say that $w = w(c_1,c_2,r(t))$ by implicit function theorem. However, this imposes three conditions as stated in Equations (7.45), (7.46) and (7.47).

$$\frac{\partial F}{\partial w} \neq 0 \tag{7.45}$$

$$\frac{\partial F}{\partial c_1} \neq 0 \tag{7.46}$$

$$\frac{\partial F}{\partial c_2} \neq 0 \tag{7.47}$$

Since $F(c_1,c_2,w(c_1,c_2,r(t))) = K$ the derivative of F with respect to t must be equal to zero.

$$\frac{\partial F}{\partial c_1}\dot{c}_1 + \frac{\partial F}{\partial c_2}\dot{c}_2 + \frac{\partial F}{\partial w}\dot{w} = 0 \tag{7.48}$$

$$\frac{\partial F}{\partial c_1}\left(-c_1 w + f\right) + \frac{\partial F}{\partial c_2}\left(-c_2 w + f g\right) + \frac{\partial F}{\partial w}\dot{w} = 0 \tag{7.49}$$

Among many other solutions (restricting F by further conditions) Equations (7.50), (7.51) and (7.52) can be proposed as a possible solution set.

$$c_1 \frac{\partial F}{\partial c_1} + c_2 \frac{\partial F}{\partial c_2} = 0 \tag{7.50}$$

$$\frac{\partial F}{\partial c_1} + g \frac{\partial F}{\partial c_2} = 0 \tag{7.51}$$

$$\frac{\partial F}{\partial w}\dot{w} = 0 \tag{7.52}$$

In this case, we can rewrite the solution set in a matrix form.

$$\begin{bmatrix} c_1 & c_2 & 0 \\ 1 & g(c_2) & 0 \\ 0 & 0 & \dot{w} \end{bmatrix} \begin{bmatrix} \partial F / \partial c_1 \\ \partial F / \partial c_2 \\ \partial F / \partial w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.53}$$

Let the matrix in the left hand side of Equation (7.53) is denoted by A. If A is invertible, this means that $\partial F/\partial w = 0$ which contradicts with our assumption. Then det A = 0.

$$\dot{w}\left(c_1(t)\, g(c_2(t)) - c_2(t)\right) = 0 \tag{7.54}$$

Now, either $\partial w/\partial t = 0$ or $c_1 g(c_2) - c_2 = 0$. If $\partial w/\partial t = 0$, the question reduces to,

$$\begin{bmatrix} c_1 & c_2 \\ 1 & g(c_2) \end{bmatrix} \begin{bmatrix} \partial F / \partial c_1 \\ \partial F / \partial c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{7.55}$$

If $c_1 g(c_2) - c_2$ is nonzero then $\partial F/\partial c_1 = 0$ and $\partial F/\partial c_2 = 0$ which contradict with our assumption. Then we conclude with;

$$c_1(t)\, g\big(c_2(t)\big) - c_2(t) = 0 \Rightarrow g\big(c_2(t)\big) = \frac{c_2(t)}{c_1(t)} \tag{7.56}$$

Now the equations of the plant reduce to,

$$\dot{c}_1(t) = -c_1(t)\, F\Big(c_1(t)\,,\, c_2(t)\,,\, w\big(c_1(t)\,,\, c_2(t)\,,\, r(t)\big)\Big) + r(t) \tag{7.57}$$

$$\dot{c}_2(t) = -c_2(t)\, F\Big(c_1(t)\,,\, c_2(t)\,,\, w\big(c_1(t)\,,\, c_2(t)\,,\, r(t)\big)\Big) + \frac{c_2(t)}{c_1(t)}\, r(t) \tag{7.58}$$

In order to equate our reference model to the plant, we must choose w such that $F(c_1,c_2,w(c_1,c_2,r(t))) = 1$. This implies that,

$$w\big(c_1(t)\,,\, c_2(t)\,,\, r(t)\big) + \frac{r(t)}{c_1(t)} - \frac{f\big(c_1(t), c_2(t)\big)}{c_1(t)} = 1 \tag{7.59}$$

If Equation (7.59) is arranged, we find Equation (7.60)

$$w\big(c_1(t), c_2(t), r(t)\big) = \frac{f\big(c_1(t), c_2(t)\big) + c_1(t) - r(t)}{c_1(t)} \tag{7.60}$$

Equation (7.60) imposes that $c_{1m}(t) = c_1(t)$ and $c_1(t)$ follows r(t) by the construction of our model. In what follows, we must show that the errors between the reference model outputs and the actual plant outputs tend to zero in the limiting case. Let the neural network realize the function $f(c_1,c_2)$, then Equation (7.60) becomes,

$$w = \frac{\hat{f}(c_1, c_2) + c_1 - r}{c_1} \tag{7.61}$$

If this control is written into Equations (7.20) and (7.21), we find Equations (7.62) and (7.63).

$$\dot{c}_1 = -c_1 + \Big(f(c_1,c_2) - \hat{f}(c_1,c_2)\Big) + r \tag{7.62}$$

$$\dot{c}_2 = -c_2 + g(c_2)\, r - \frac{g(c_2)}{f(c_1,c_2)} \Big(f(c_1,c_2) - \hat{f}(c_1,c_2)\Big) \tag{7.63}$$

From the theorems explained in the second chapter, the function $f(c_1,c_2)$ can be realized by neural networks such that the error in the neural network output remains within a prespecified level. As long as the neural network realizes the function $f(c_1,c_2)$ precisely, we can neglect the difference between the network output and the actual value of the function. Below, we explain the effect of this difference.

$$\dot{c}_{1m} - \dot{c}_1 = -(c_{1m} - c_1) - \varepsilon(t) \tag{7.64}$$

$$\dot{c}_{2m} - \dot{c}_2 = -(c_{2m} - c_2) - \frac{g(c_2)}{f(c_1,c_2)}\varepsilon(t) \tag{7.65}$$

$$\dot{e}_1 = -e_1 - \varepsilon(t) \tag{7.66}$$

$$\dot{e}_2 = -e_2 - \frac{g(c_2)}{f(c_1,c_2)}\varepsilon(t) \tag{7.67}$$

Equations (7.66) and (7.67) reveal that the error model has the roots on the left half s-plane, moreover, the error in c_1 is forced to track $\varepsilon(t)$ and the error in c_2 is forced to track $(g/f)\varepsilon(t)$. Therefore, our inferences dealing with the accuracy of the neural network stipulated that the bioreactor is forced to track the reference model. In the remainder of this section, we give the simulation results for the bioreactor control problem.

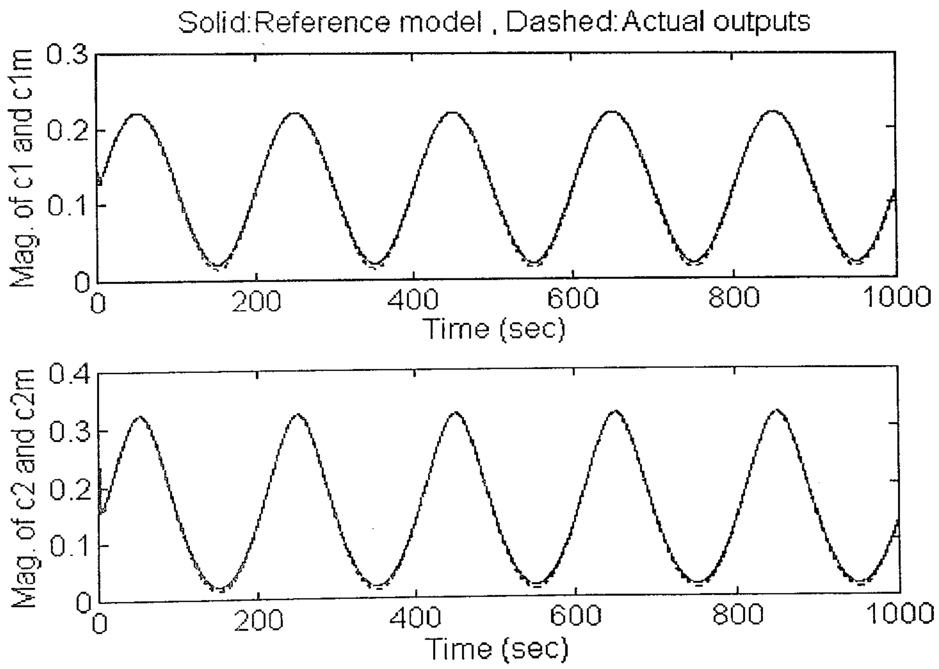


Figure 7.7(a) Cell and nutrient concentration graphs for the reference model and for the actual plant. Command signal is $r(t) = 0.1207 + 0.1\sin(2\pi t/200)$

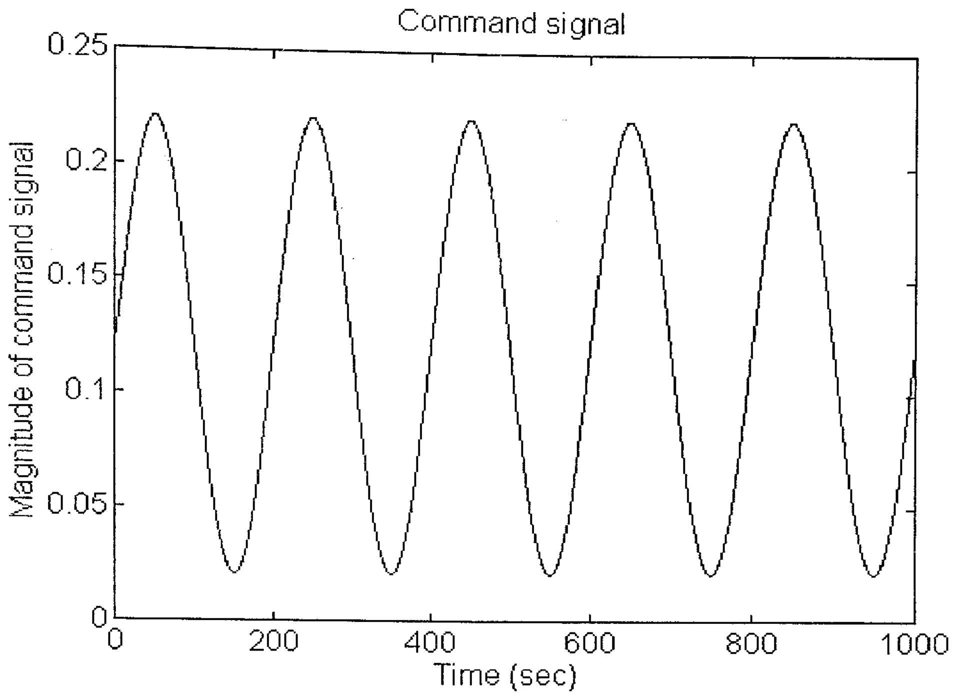


Figure 7.7(b) Command signal graph, r(t) = 0.1207 + 0.1sin(2πt/200)

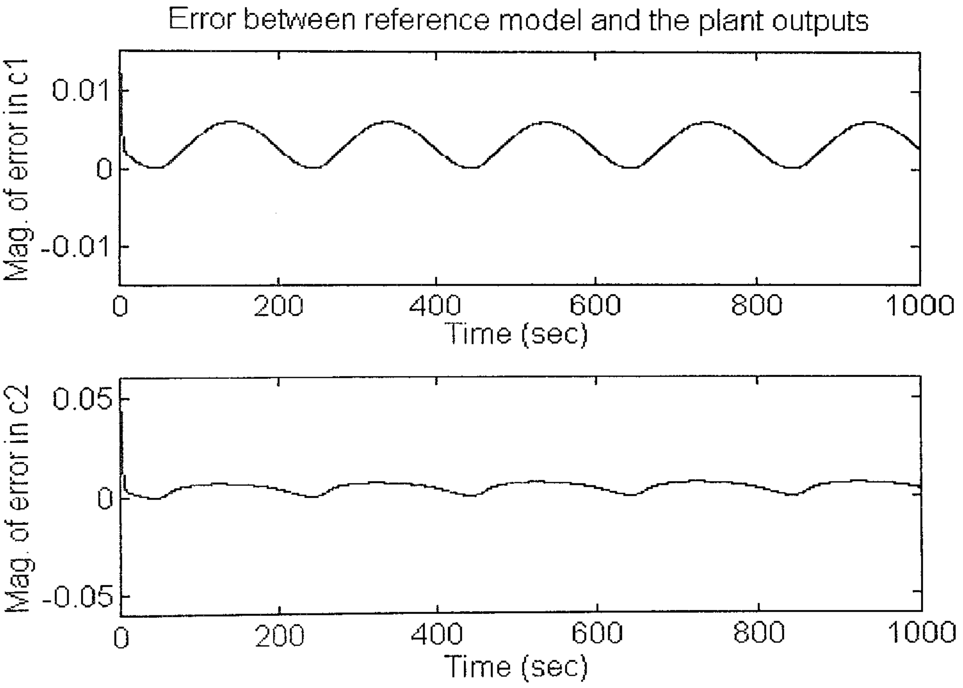


Figure 7.7(c) Error graph for the cell and nutrient concentration, command signal is r(t) = 0.1207 + 0.1sin(2πt/200)

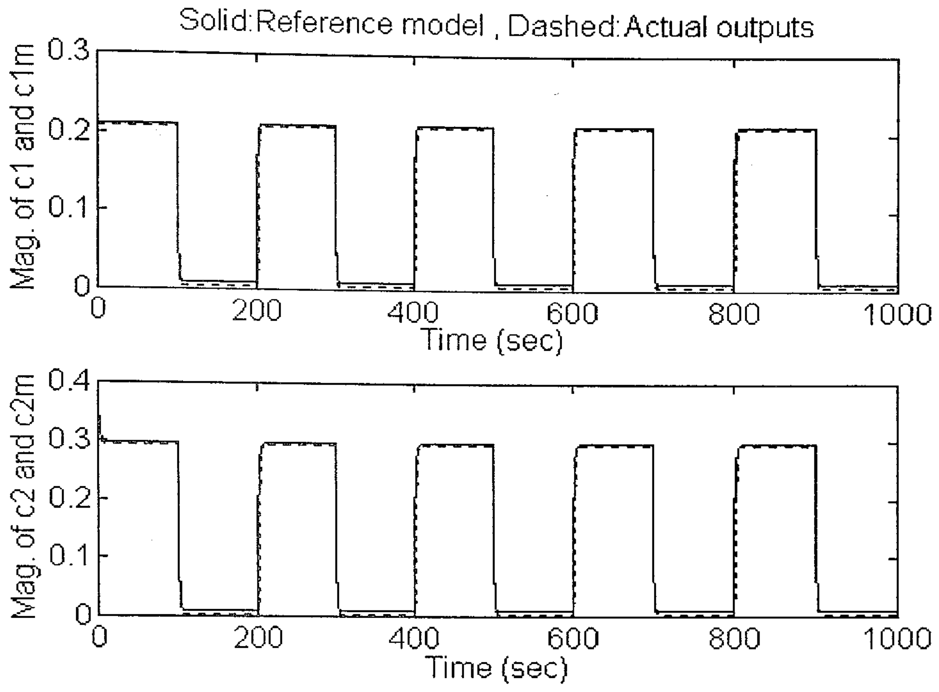


Figure 7.8(a) Cell and nutrient concentration graphs for the reference model and for the actual plant. Command signal is r(t) = 0.11 + 0.1sgn(sin(2πt/200))

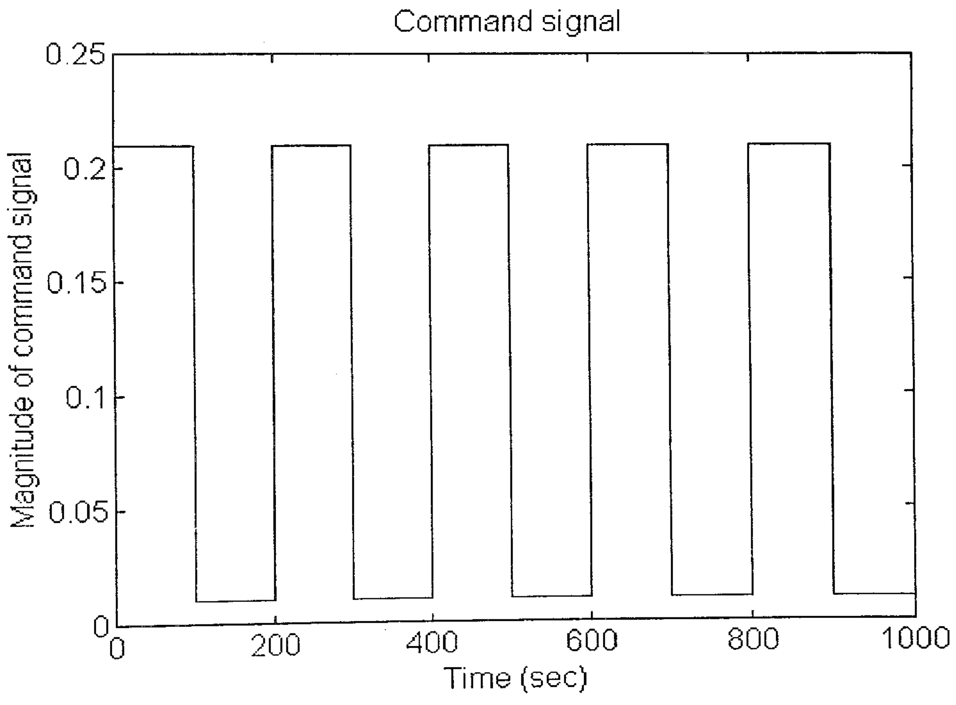


Figure 7.8(b) Command signal graph, r(t) = 0.11 + 0.1sgn(sin(2πt/200))

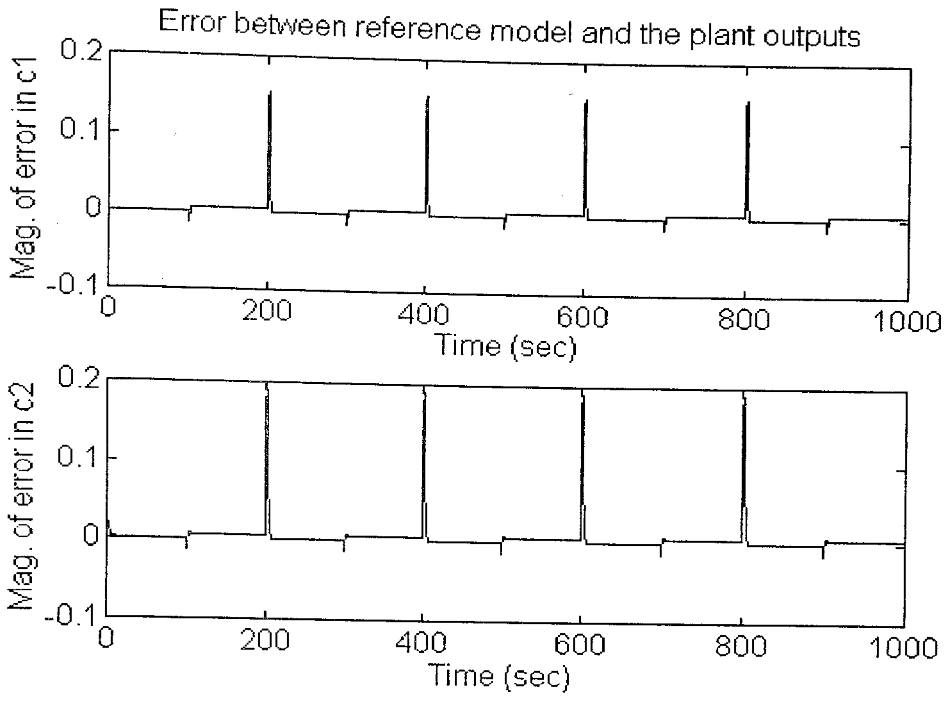


Figure 7.8(c) Error graph for the cell and nutrient concentration, command signal is r(t) = 0.11 + 0.1sgn(sin(2πt/200))

7.3 Performance Assessment

MRAC technique is one of the well-known and widely used method for controlling nonlinear dynamical systems. Recent studies showed that the method can be combined with the neural networks. In our simulations, we considered the control problems from this point of view.

In the first two simulations, the plants and the models were extracted from [5] and each one of them was simulated for two different kinds of reference signals. These two simulations showed that the performance of this approach is good enough in the sense of tracking ability. In the third simulation, the bioreactor plant was controlled by means of MRAC technique. The results for this case revealed that the performance of this approach is considerably better than the method explained in the fifth chapter. MRAC showed a smooth tracking of reference model outputs whereas in the inverse control strategy, we had observed sharp deviations in the bioreactor outputs.

The vital problem of the approach is the stability of the controller. As mentioned earlier, there is no concrete work explaining the stability of neural controllers and the stability of update dynamics.

In brief, if the reference model is stable, and if we can show the difference between reference model output and the actual plant output tends to zero in the limiting case, also, if we can prove the convergent behavior of the parameters of the control strategy, then, MRAC technique yields an admissible performance in the sense of output tracking ability.

8. SELF-LEARNING CONTROL USING NEURAL NETWORKS

In this chapter, we elaborated the self-learning control scheme which was originally proposed by Nguyen and Widrow [6]. The method differs from those explained earlier in that this scheme is goal directed. The controller is trained to keep the plant output at a desired and previously defined level. The approach utilizes the well-known backpropagation method in the training of the controller. Nguyen and Widrow applied the method to a truck backer-upper plant. We tested the performance of the method by applying it to different types of plants and the bioreactor benchmark problem.

8.1 Control Strategy

The first step of the control procedure is obtaining the neural identification model of the plant. In Fig. 8.1, identification model, which is also called plant emulator, is represented by E boxes. If the identification model represents the plant dynamics accurately, it can be used in the architecture shown in Fig. 8.1. This architecture implies that after K time steps, plant output could be pulled from its initial state y_0 to the desired state y_d. K is determined by the designer and represents the length of the chain-like structure. The objective of the strategy is minimization of the cost function defined by Equation (8.1).

$$J = E\left(\| y_d - y_K \|^2 \right) \tag{8.1}$$

The minimization of Equation (8.1) is carried out through evaluating the output error at the Kth step and propagating it back through the structure illustrated in Fig. 8.1. Utilizing the plant emulator makes the training process easy. The details of the error backpropagation have been scrutinized in the fifth chapter of this study. An important point in the weight updating is the succession that appears in the controller training structure. In fact, the weight change at the Kth stage influences the (K-1)th stage. This obviously requires the saving of each individual weight change evaluated at the corresponding stage. Nguyen and Widrow [6], in their work, say that the weight changes could be applied immediately as they are evaluated because they are the accumulated effects that improve the performance of the controller and this does not affect the final performance significantly. Having trained the neural controller, it is installed to the control system as shown in Fig. 8.2.

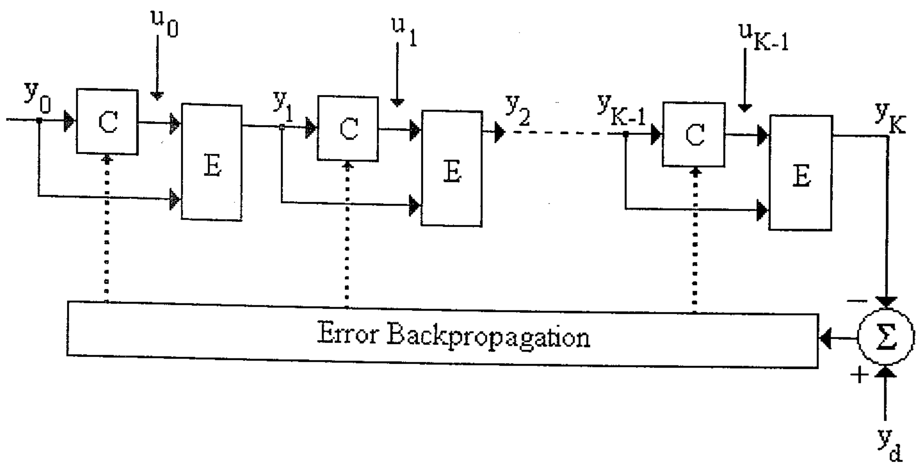


Figure 8.1 Controller training architecture for the self-learning control scheme

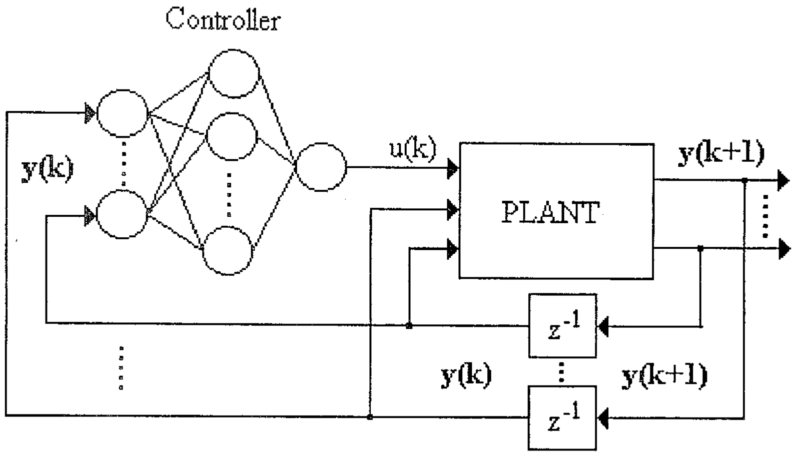


Figure 8.2 Control system structure with neural controller

8.2 Simulation Results

In the simulation results given in this section, we used the neural controller and neural identification blocks that have the neurons possessing linear activation functions in the input and the output layers, and the neurons possessing sigmoidal activation functions in the hidden layers.

Simulation 1.

Plant : $y(k+1) = \tanh(y(k)) + u(k)$ (8.2)

Identification model	: 2-6-1	MSE : 1e-6
Controller	: 1-8-1	MSE : 1e-6

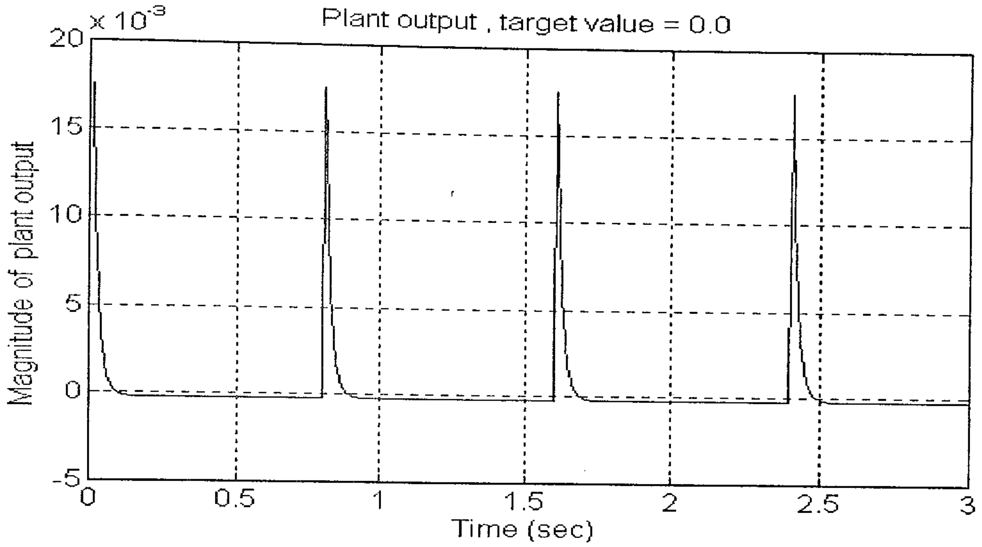


Figure 8.3(a) Output graph for the plant defined by Equation (8.2). The desired level of the output is 0.0 and the plant was initiated at every 0.8 seconds randomly.

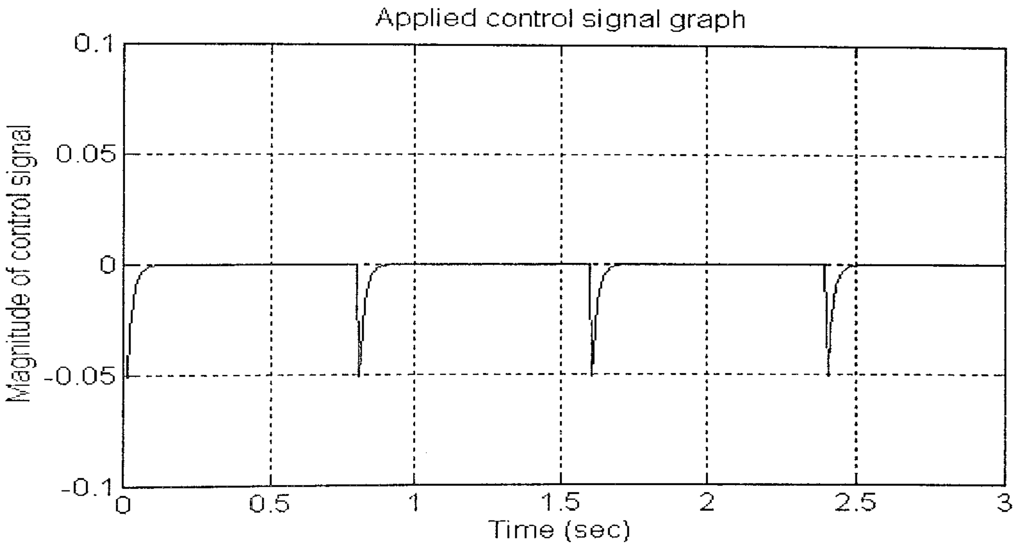


Figure 8.3(b) The control signal that was applied to the plant defined by Equation (8.2) and that was generated by the neural controller

Simulation 2.

Plant : $y(k+1) = \dfrac{1}{1+y^2(k)} + y(k)u(k)$ (8.3)

Identification model	: 2-4-1	MSE : 3e-6
Controller	: 1-6-1	MSE : 1e-6

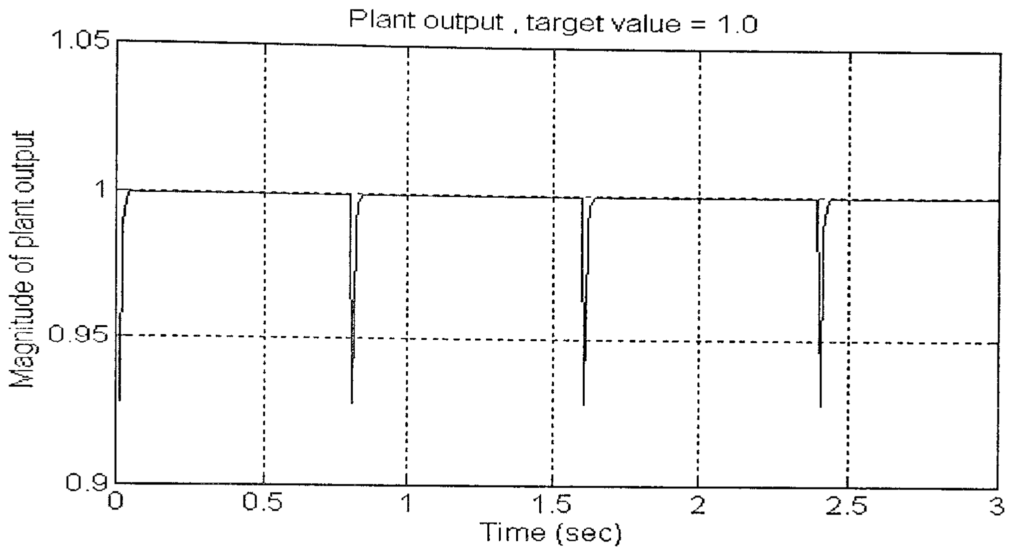


Figure 8.4(a) Output graph for the plant defined by Equation (8.3). The desired level of the output is 1.0 and the plant was initiated at every 0.8 seconds randomly.

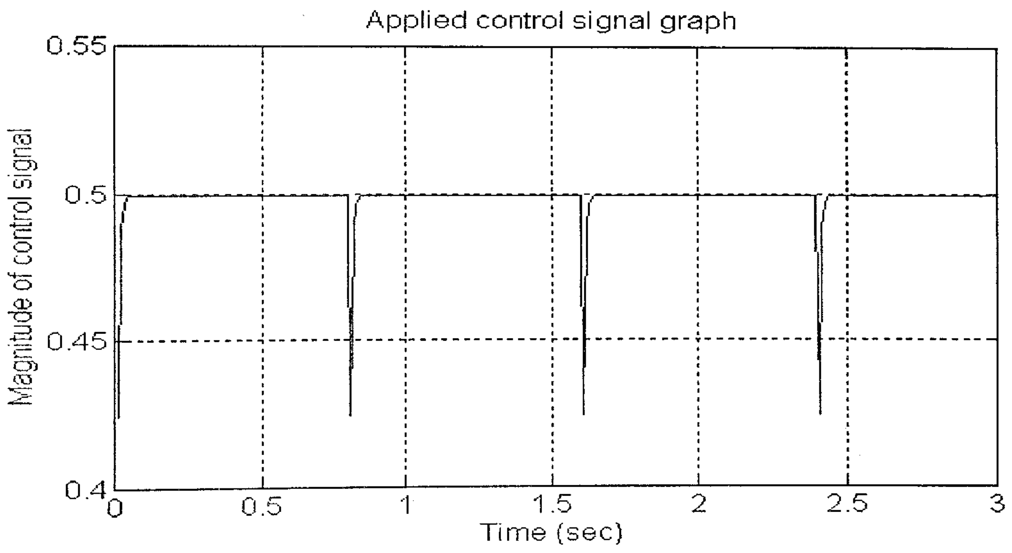


Figure 8.4(b) The control signal that was applied to the plant defined by Equation (8.3) and that was generated by the neural controller

Simulation 3.

Plant : Bioreactor model with coupled nonlinear difference eqns.

$$c_1(k+1) = c_1(k) + \Delta\left(- c_1(k)\, r(k) + c_1(k)\left(1-c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \right) \tag{8.4}$$

$$c_2(k+1) = c_2(k) + \Delta\left(-c_2(k)\, r(k) + c_1(k)\left(1-c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \frac{1+\beta}{1+\beta-c_2(k)} \right) \tag{8.5}$$

Identification model	: 3-25-16-2	MSE : 88e-6
Controller	: 2-8-4-1	MSE : 1e-6

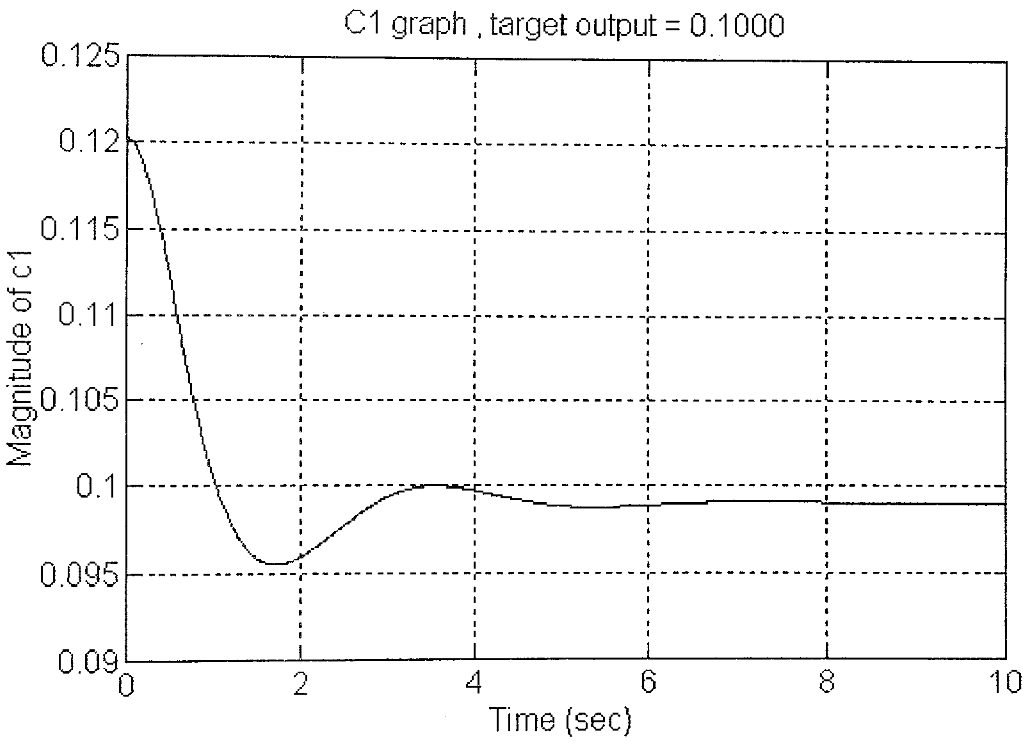


Figure 8.5(a) The time evolution of cell concentration (c_1). The desired level was defined to be 0.1000

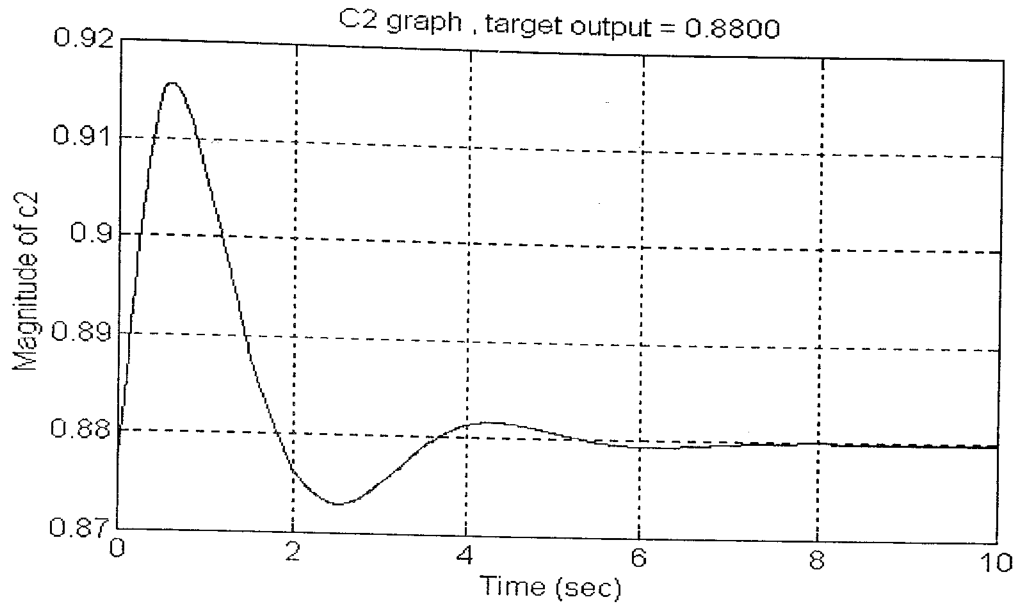


Figure 8.5(b) The time evolution of nutrient concentration (c_2). The desired level was defined to be 0.8800

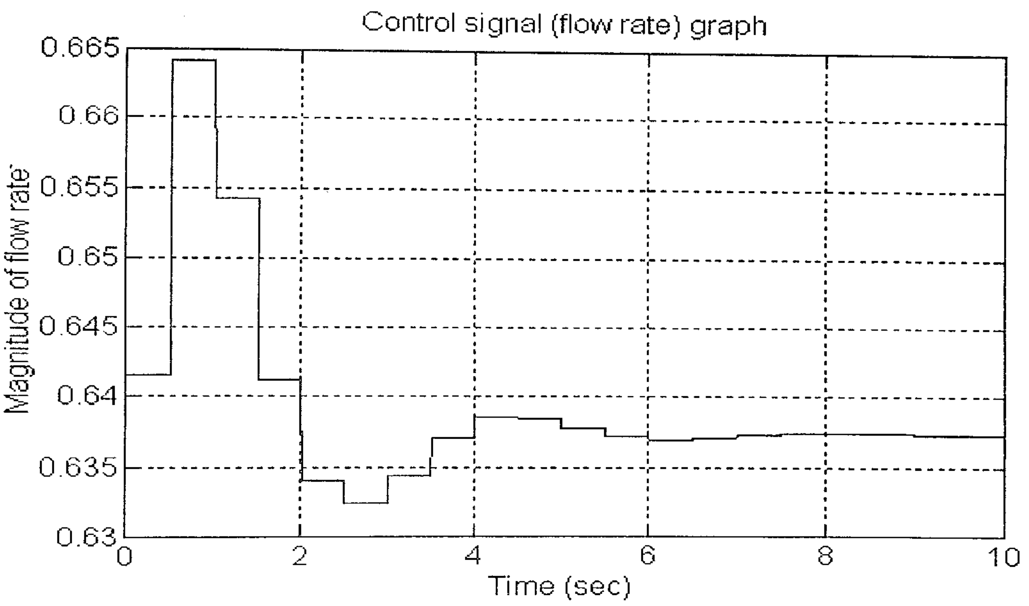


Figure 8.5(c) The control signal (flow rate) that was applied to the bioreactor defined by Equations (8.4) and (8.5) and that was generated by the neural controller

8.3 Performance Assessment

In our simulations, we chosen the plants that were used in the strategies explained earlier. In this section, we consider the control strategy from the point of goal direction rather than following a user defined trajectory.

In the first simulation, we used the plant which was used in the fifth chapter of this study. The simulation results in Fig. 8.3(a) showed that the neural controller which was trained with the self-learning control scheme is able to bring the plant output to its desired level very accurately. The plant used in the second simulation is taken from the sixth chapter. The same accuracy was observed, also, for this simulation. Our third simulation is controlling the bioreactor plant which is explained in Appendix A in detail. The desired cell (c_1) and nutrient (c_2) concentrations were defined to be 0.1000 and 0.8800 respectively. As can be seen form Figs. 8.5(a), 8.5(b) and 8.5(c), the method shows an admissible performance when these results are compared with those obtained in the fifth chapter. Firstly, we observed a smooth change in the plant outputs and a fast response. If the Figs. 5.18(a) through 5.18(d) are taken into consideration, we had observed sharp deviations in the tracking even in the regions that the desired trajectory is constant. The method that we introduced in this chapter resulted in better performance in the sense of steady state performance and in the sense of controlling the plant in a few seconds. The stability of the neural controller is again an unanswered problem for this approach.

9. DYNAMICAL NEURAL UNITS FOR CONTROL OF NONLINEAR DYNAMICAL SYSTEMS

In this chapter, we opened a highly promising way of designing neural network based controllers. The method changes the classical neuron model to a second order system with synaptic and somatic parts and adopts this new neuronal model. Additionally, the adaptation is carried out on the coefficients of this second order block and the on the slope of its nonlinear activation function. The authors who proposed this method called the new neuron model as Dynamical Neural Unit (DNU) and separated the learning process into two different adaptation mechanisms. The strategy is comprised of synaptic adaptation and somatic adaptation. In the former, parameters of the linear second order dynamical system are adjusted while in the latter the gain of the nonlinear activation function is adjusted.

The authors Gupta and Rao [7] point out that the resulting neural controller which is trained using the proposed method possesses a higher degree of mathematical tractability in the sense of stability analysis.

9.1 Control Strategy

The topology of a single dynamical neural unit consists delay elements, feedforward and feedback synaptic weights and a nonlinear somatic operator. The architecture of the DNU model is illustrated in Fig 9.1.

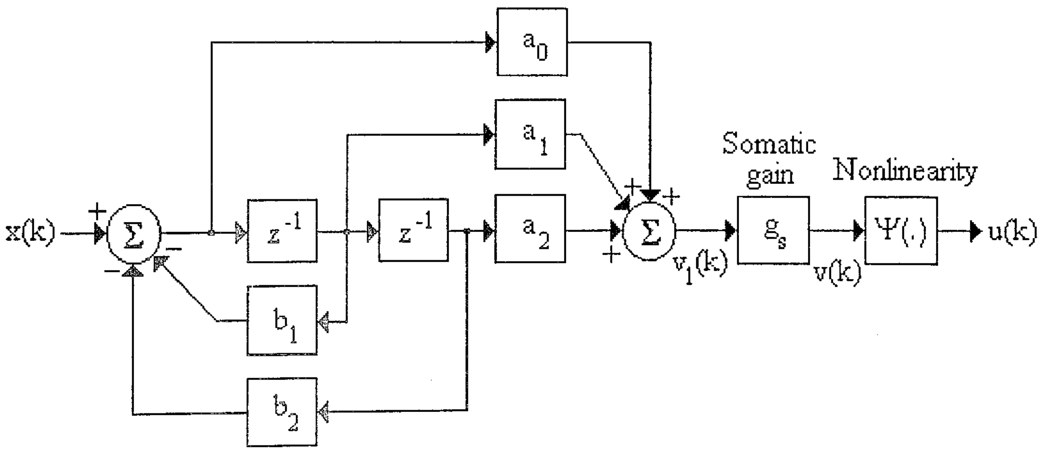


Figure 9.1 Structure of a single dynamical neural unit

The difference equation which describes the behavior of the second order dynamical structure is given in Equation (9.1) in which $v_1(k), x(k) \in R^1$. Similarly, the pulse transfer function of this part can be given by Equation (9.2).

$$v_1(k) = -b_1v_1(k-1) - b_2v_1(k-2) + a_0x(k) + a_1x(k-1) + a_2x(k-2) \tag{9.1}$$

$$T(z) = \frac{a_0 + a_1z^{-1} + a_2z^{-2}}{1 + b_1z^{-1} + b_2z^{-2}} \tag{9.2}$$

The output of the DNU can then be evaluated as follows;

$$v(k) = g_sv_1(k) \tag{9.3}$$

$$u(k) = \Psi(v(k)) = \tanh(g_sv_1(k)) \tag{9.4}$$

The objective is based on the minimization of the instantaneous error evaluated at the output of the system. The cost function which is to be minimized is defined by Equation (9.6) in which E denotes the expectation operation. If w denotes the parameters of a single DNU, the update rule can be given by;

$$w_{k+1} = w_k - \mu \frac{\partial J}{\partial w} \tag{9.5}$$

where;

$$J = \frac{1}{2} E\left(e^2(k)\right) \tag{9.6}$$

$$e(k) = y_d(k) - u(k) \tag{9.7}$$

$$\frac{\partial J}{\partial w} = \frac{1}{2} E\left(\frac{\partial\left[y_d(k)-u(k)\right]^2}{\partial w}\right) \tag{9.8}$$

$$\frac{\partial J}{\partial w} = E\left(e(k)\left[-\frac{\partial u(k)}{\partial w}\right]\right) = E\left(e(k)\left[-\frac{\partial \Psi(v)}{\partial w}\right]\right) \tag{9.9}$$

$$\frac{\partial J}{\partial w} = E\left(-e(k)\left[\frac{\partial \Psi(v)}{\partial v}\,\frac{\partial v}{\partial w}\right]\right) \tag{9.10}$$

$$\frac{\partial v}{\partial a_i} = g_s x(k-i) \quad i = 0,1,2 \tag{9.11}$$

$$\frac{\partial v}{\partial b_j} = -g_s v_1(k-j) \quad j = 1,2 \tag{9.12}$$

$$\frac{\partial v}{\partial g_s} = v_1 \tag{9.13}$$

More compactly, the parameter update rule is given by Equations (9.14) through (9.16) where i = 0,1,2 and j = 1,2 ;

$$\Delta a_i(k+1) = \mu_{a_i} E\left(e(k)\, g_s^2(k)\, \mathrm{sech}^2[v_1(k)]\, x(k-i)\right) \tag{9.14}$$

$$\Delta b_j(k+1) = -\mu_{b_j} E\left(e(k)\, g_s^2(k)\, \mathrm{sech}^2[v_1(k)]\, v_1(k-j)\right) \tag{9.15}$$

$$\Delta g_s(k+1) = g_s(k)\mu_{g_s} E\left(e(k)\, \mathrm{sech}^2[v_1(k)]\, v_1(k)\right) \tag{9.16}$$

In the parameter update equations, the coefficients μ_{ai}, μ_{bj} and μ_{gs}, denote the step size for the corresponding parameter and chosen to be constant throughout a simulation.

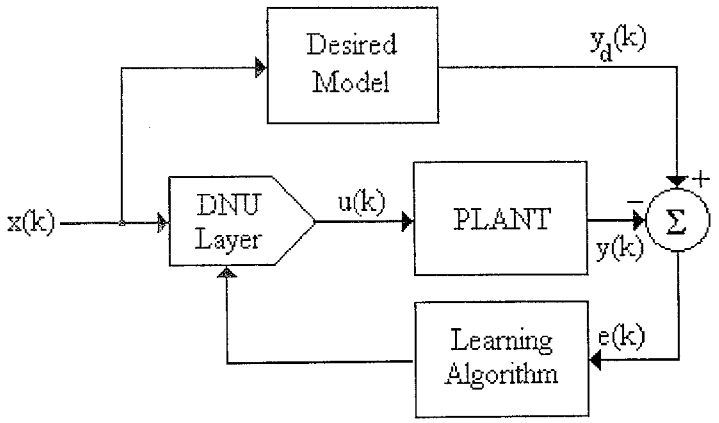


Figure 9.2 Control system architecture with DNU based controller network

In Fig. 9.2, DNU layer includes the desired number of individual DNU blocks whose inputs are connected together and whose outputs are added to form the control u(k). Depending on the magnitude of the output error, the algorithm updates the feedforward and feedback weights and the gain of the nonlinear activation functions of each dynamical neural unit in the DNU layer. The derivation of the algorithm is given in [7] in detail.

9.2 Simulation Results

Simulation 1

Plant :

$$y(k+1) = \begin{cases} \dfrac{\sin\big(y(k) + u(k)\big)}{\sqrt{1+y^2(k)+y^2(k-1)+y^2(k-2)+u^2(k-1)+u^2(k-2)}} + u(k) & t < 15 \text{ sec} \\[2ex] \dfrac{\sin\big(y(k)\big)\sin\big(y(k-1)y(k-2) - u(k-1)u(k-2)\big)}{\sqrt{1+y^2(k)+y^2(k-1)+y^2(k-2)}} + u(k) & t \geq 15 \text{ sec} \end{cases}$$

(9.17)

Input : $0.8\sin\left(\dfrac{\pi t}{2}\right)$ (9.18)

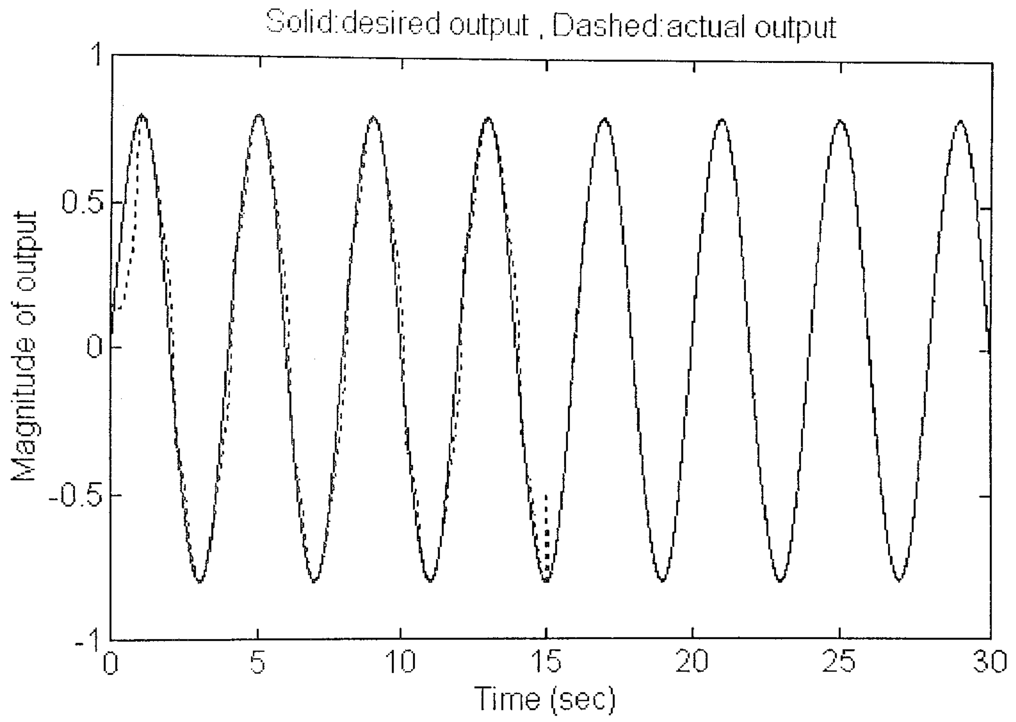


Figure 9.3(a) Output of the plant defined by Equation (9.17) in response to the input defined by Equation (9.18)

At time t = 15, plant is changed as described by Equation (9.17) so that we tested the robustness of the DNU based controller. We observed a spike at the output of the plant at t = 15. It is clear that the adaptation mechanism compensated the changes in the plant parameters quickly. The effect of this change can also be seen from Figs. 9.3(b) through

9.3(e). In this simulation a single DNU block adequately controlled the plant. In Figs. 9.4(a) through 9.4(c), we carried out the same simulation with the addition of Gaussian random noise to the measured state y(k).

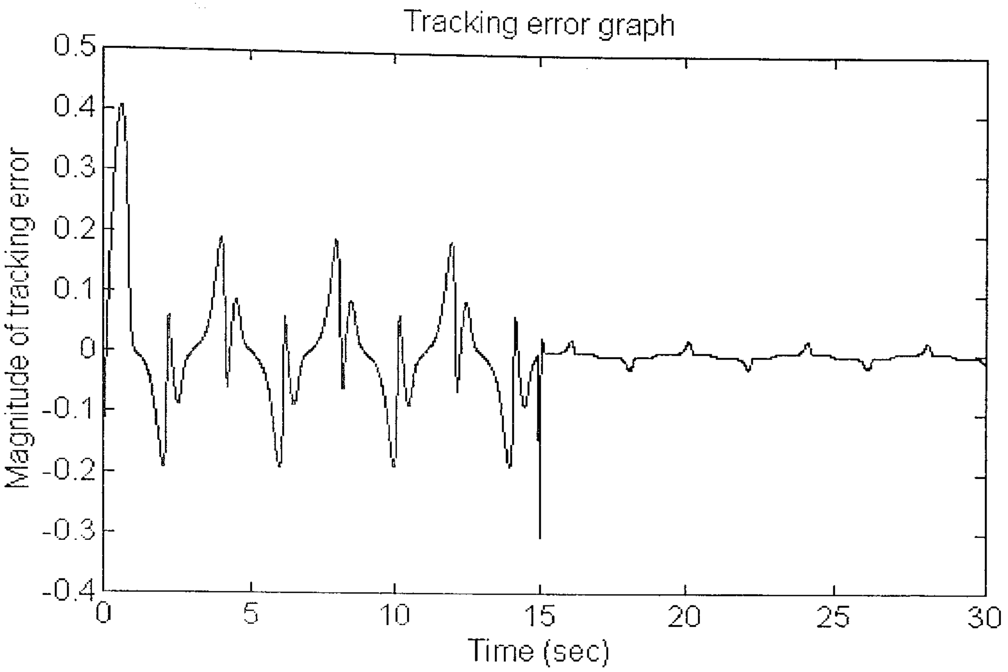


Figure 9.3(b) Output error for the plant defined by Equation (9.17) in response to the input defined by Equation (9.18)

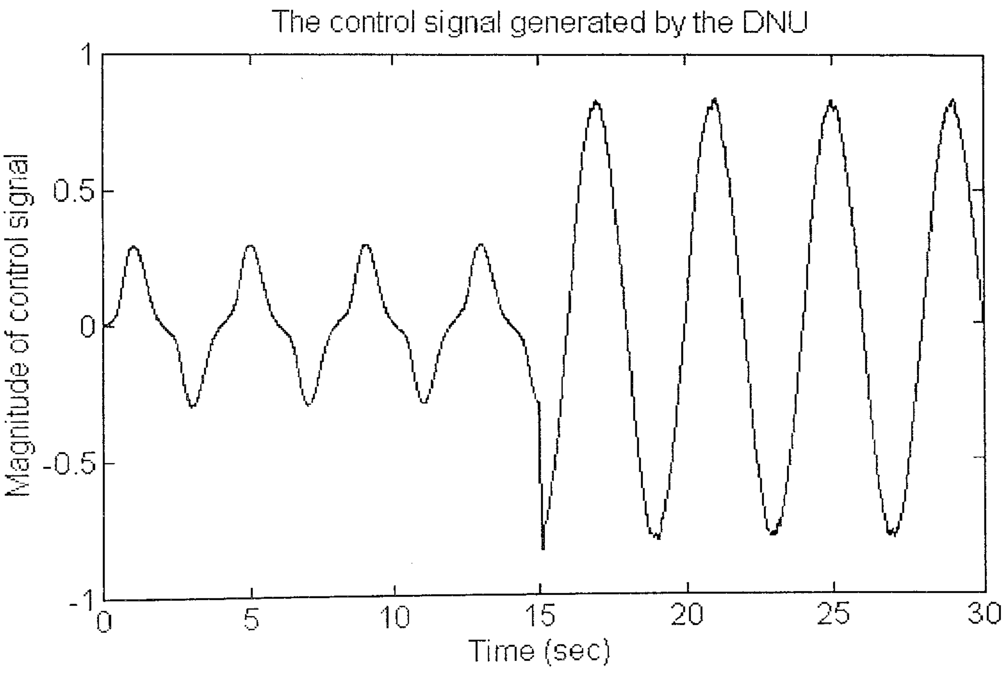


Figure 9.3(c) Control signal applied to the plant defined by Equation (9.17) and generated by DNU

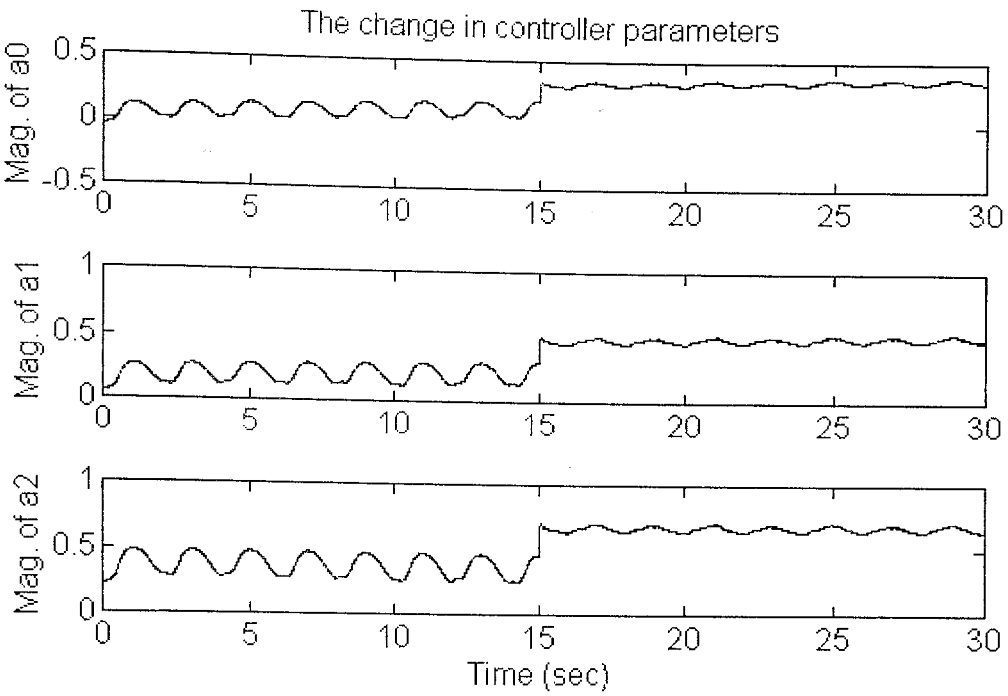


Figure 9.3(d) The changes observed at the feedforward synaptic weights of the DNU

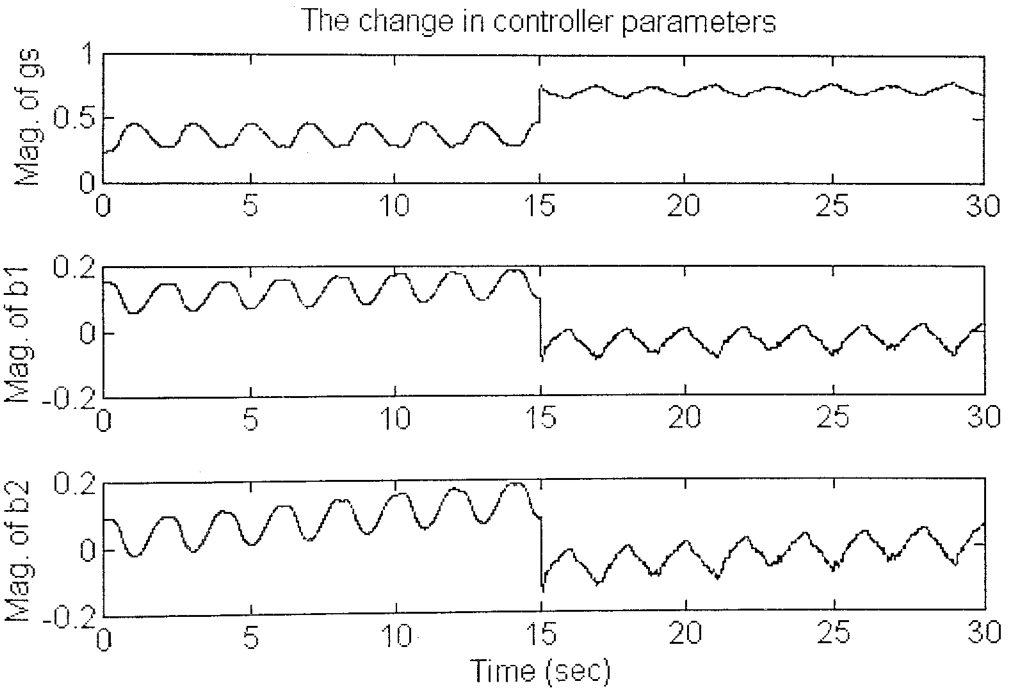


Figure 9.3(e) The changes observed at the somatic gain and the feedback synaptic weights of the DNU

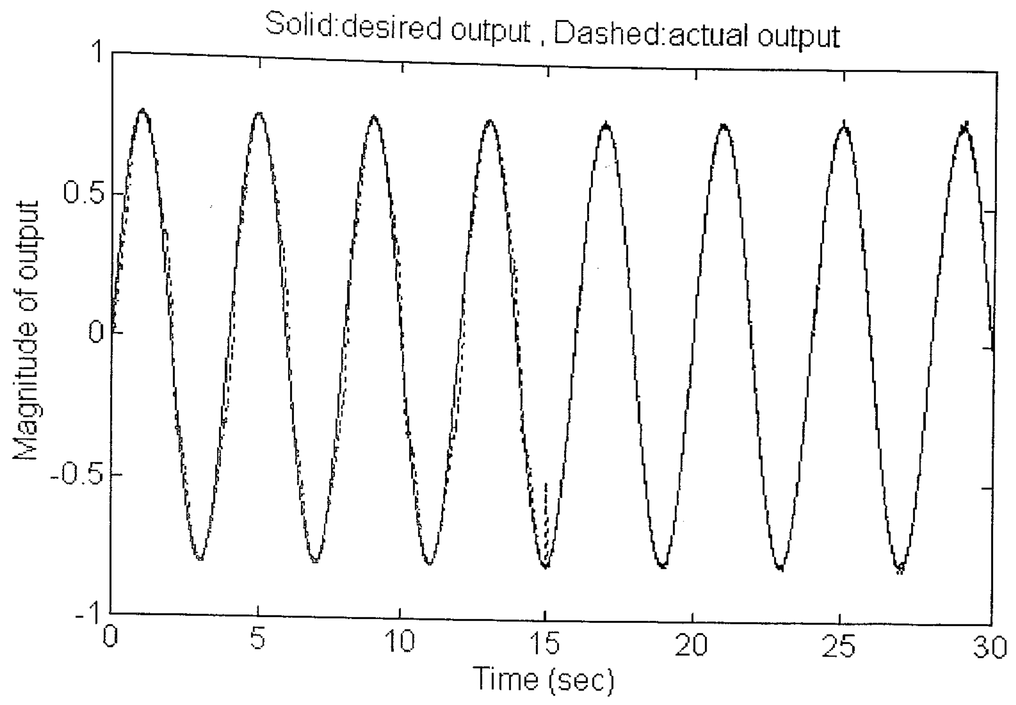


Figure 9.4(a) Output of the plant defined by Equation (9.17) in response to the input defined by Equation (9.18) 10% Gaussian random noise is added to the measured state y(k)

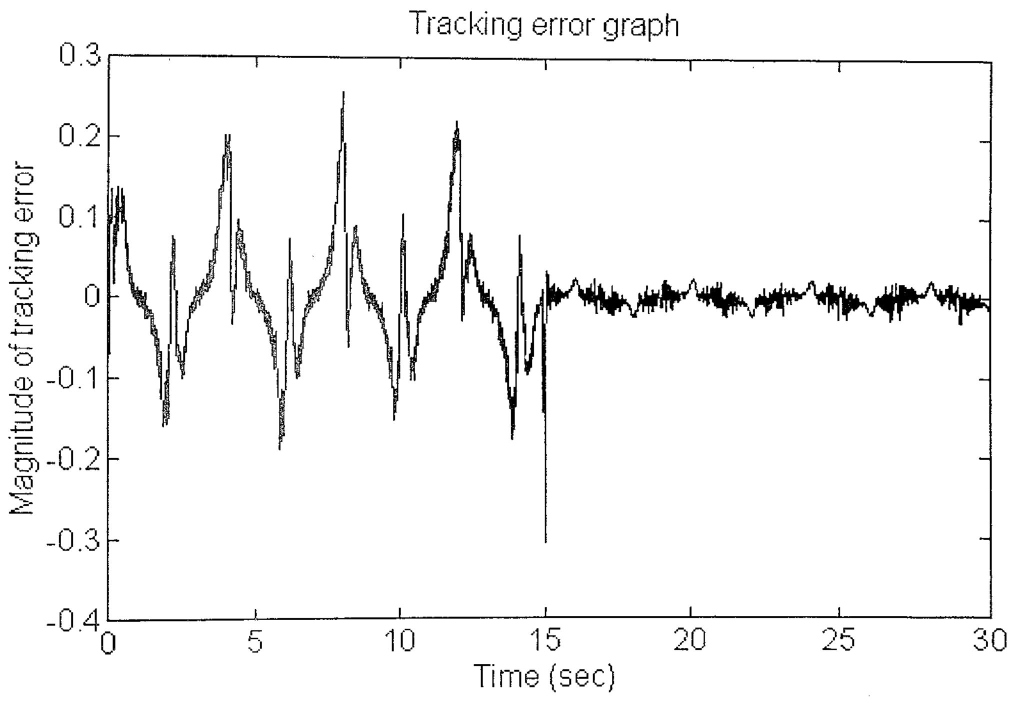


Figure 9.4(b) Output error for the plant defined by Equation (9.17) in response to the input defined by Equation (9.18), 10% Gaussian random noise is added to the measured state y(k)

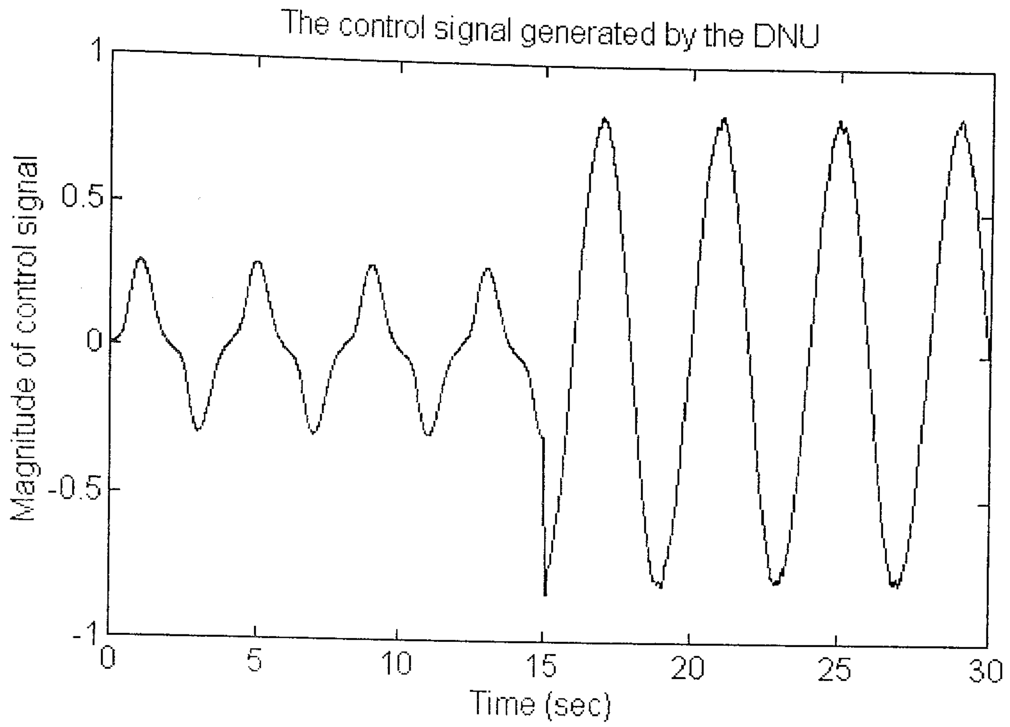


Figure 9.4(c) Control signal applied to the plant defined by Equation (9.17) and generated by DNU 10% Gaussian random noise is added to the measured state y(k)

Simulation 2

$$\text{Plant : } y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{9.19}$$

$$\text{Input : } \sin\left(\frac{\pi t}{2}\right) \tag{9.20}$$

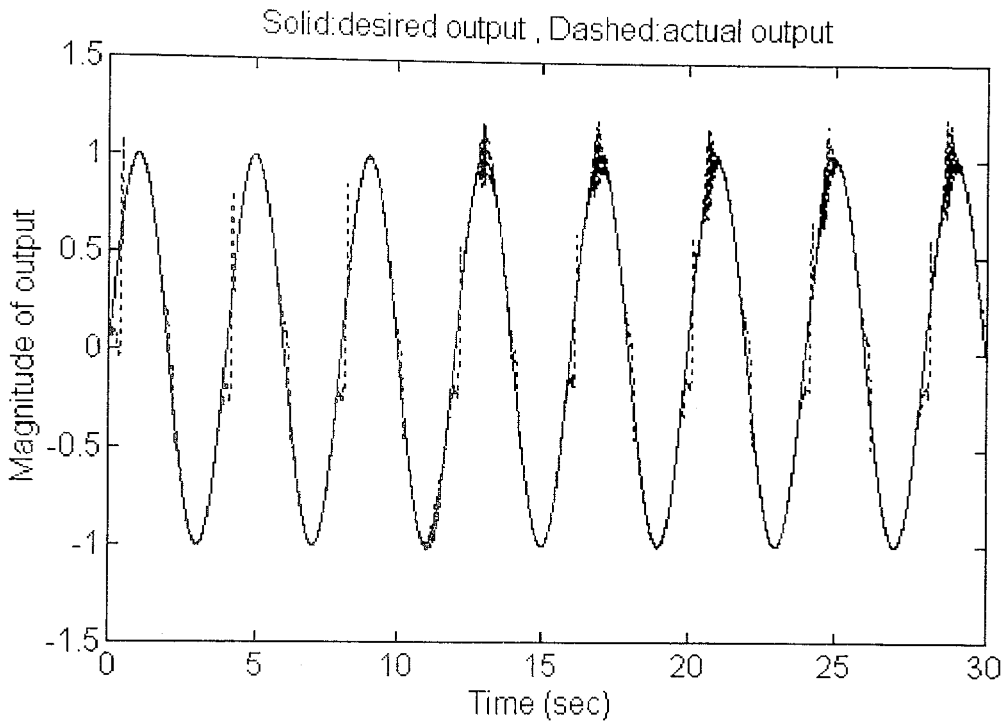


Figure 9.5(a) Output of the plant defined by Equation (9.19) in response to the input defined by Equation (9.20), 10% Gaussian random noise is added to the measured state y(k)

This simulation also concerns the 10% additive Gaussian random noise which affects the state y(k). Additionally, the DNU layer is comprised of a single DNU block. As can be seen from Fig. 9.5(a), as time progresses an overshoot appears in the plant output at the maximum values of the input signal. For this plant, the algorithm could not compensate this undesired behavior which is observed at the output.

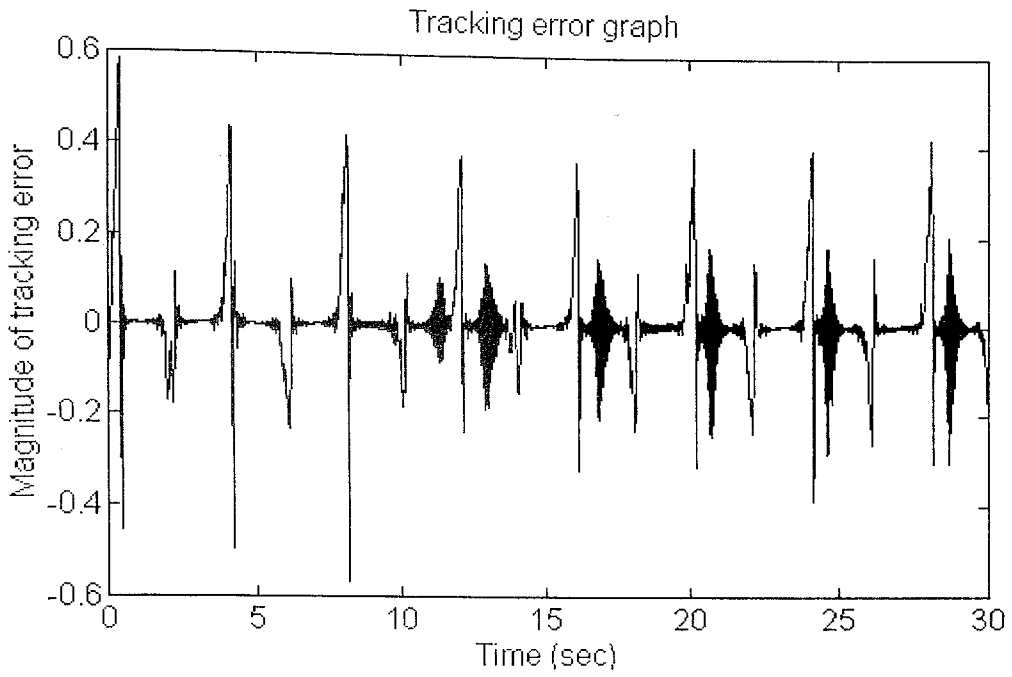


Figure 9.5(b) Output error for the plant defined by Equation (9.19) in response to the input defined by Equation (9.20), 10% Gaussian random noise is added to the measured state y(k)

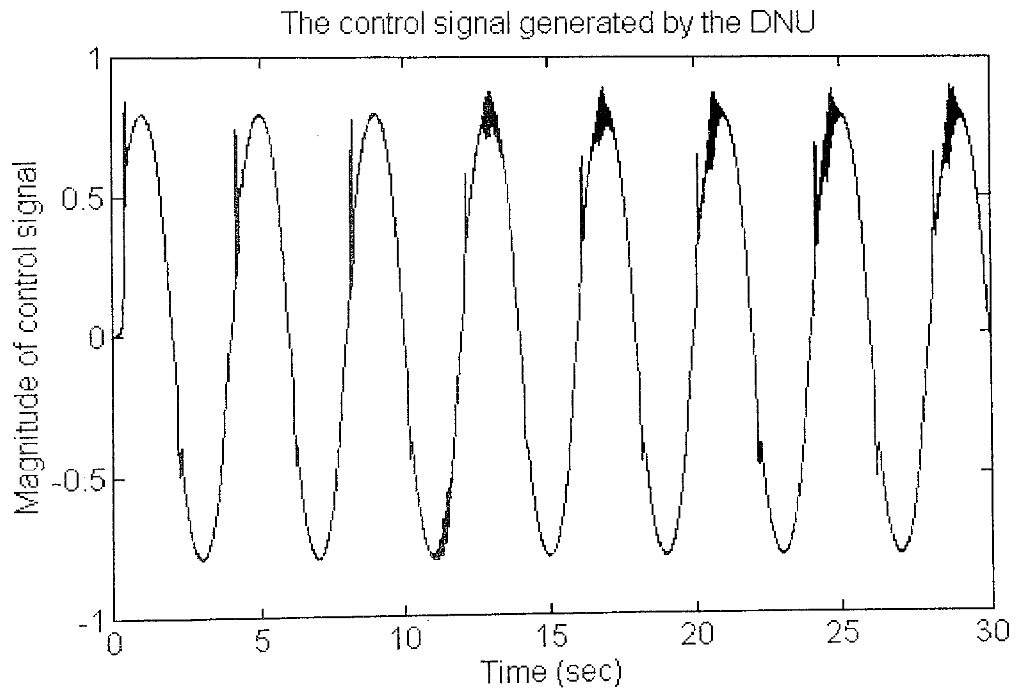


Figure 9.5(c) Control signal applied to the plant defined by Equation (9.19) and generated by DNU, 10% Gaussian random noise is added to the measured state y(k)

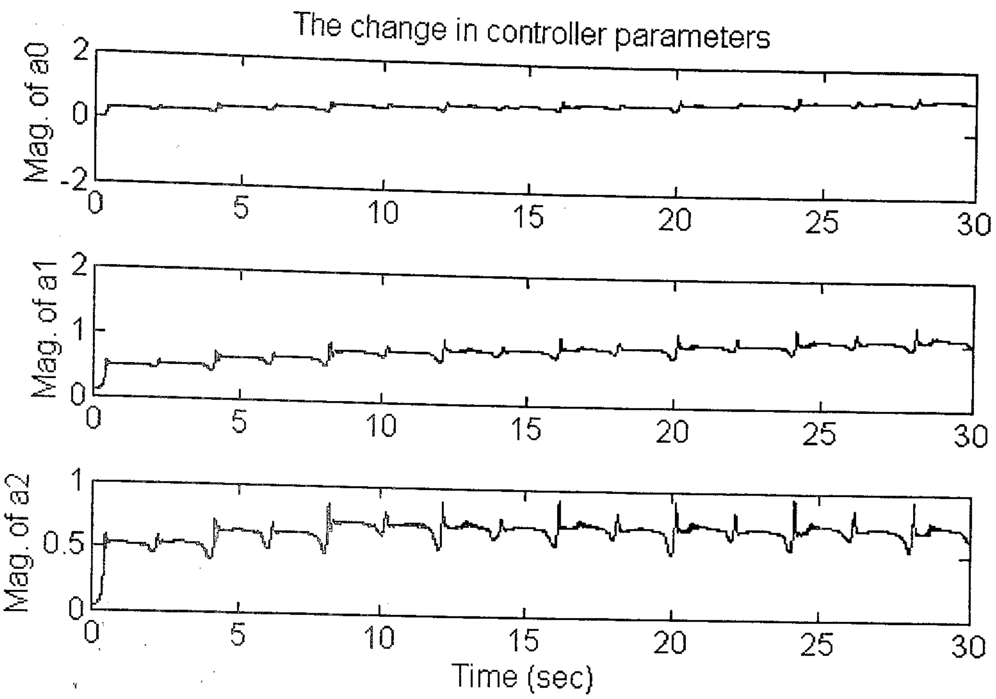


Figure 9.5(d) The changes observed at the feedforward synaptic weights of the DNU, 10% Gaussian random noise is added to the measured state y(k)

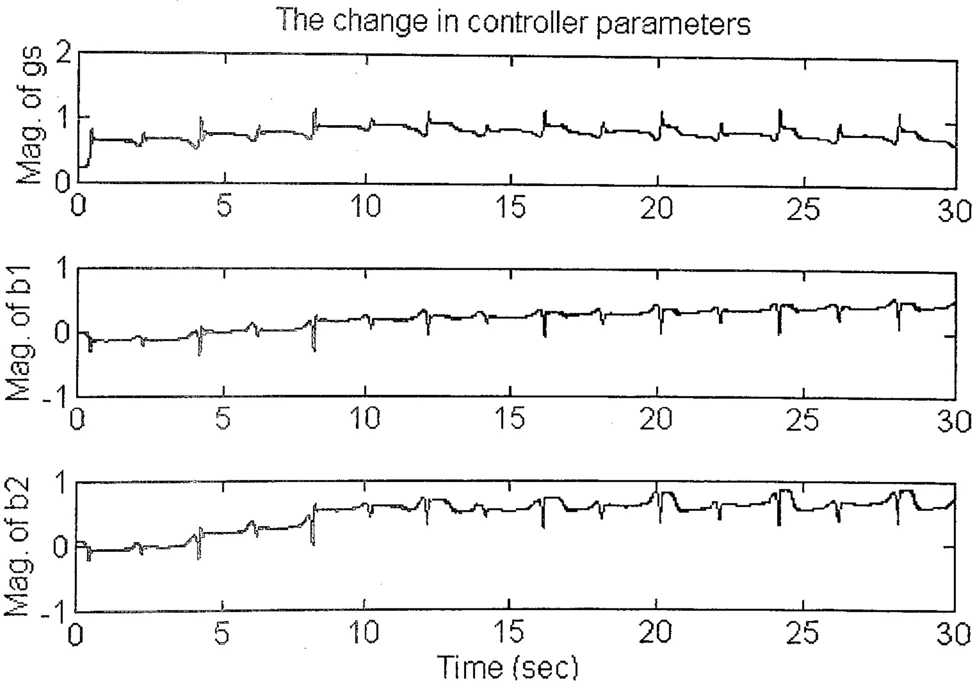


Figure 9.5(e) The changes observed at the somatic gain and the feedback synaptic weights of the DNU, 10% Gaussian random noise is added to the measured state y(k)

Simulation 3

$$\text{Plant} : y(k+1) = \tanh(y(k)) + u(k) \tag{9.21}$$

$$\text{Input} : \sin\left(\frac{\pi t}{2}\right) \tag{9.22}$$

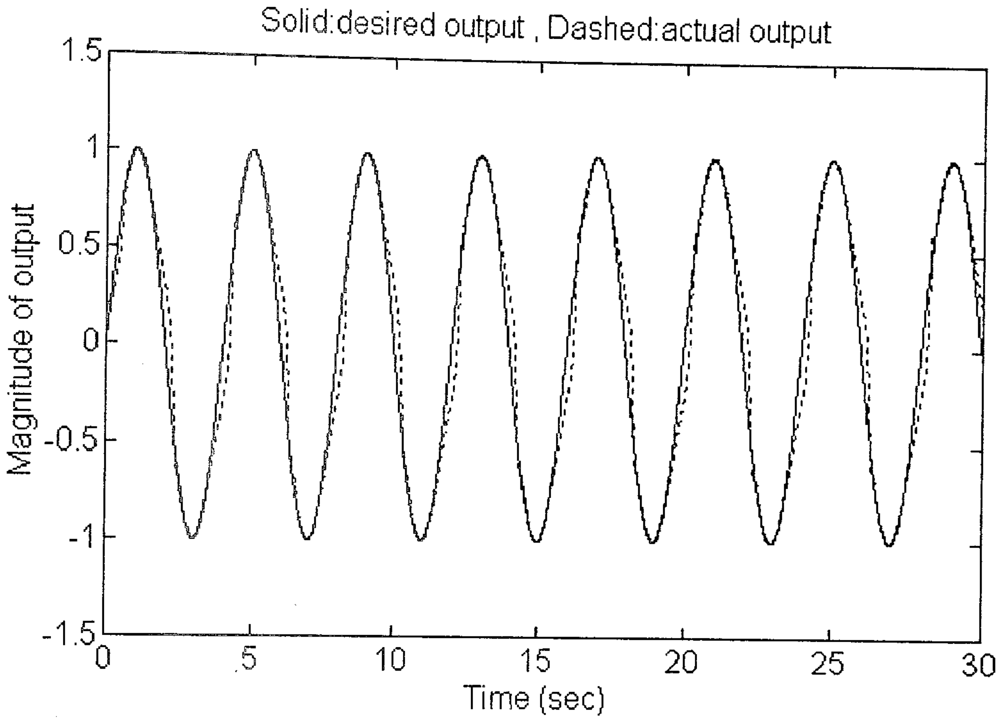


Figure 9.6(a) Output of the plant defined by Equation (9.21) in response to the input defined by Equation (9.22)

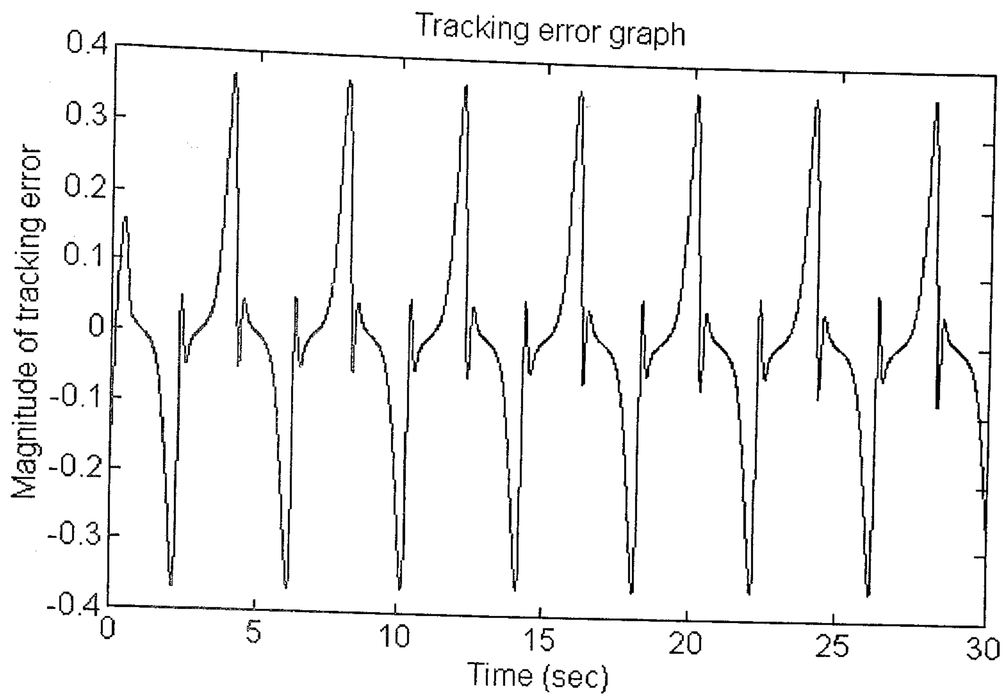


Figure 9.6(b) Output error for the plant defined by Equation (9.21) in response to the input defined by Equation (9.22)

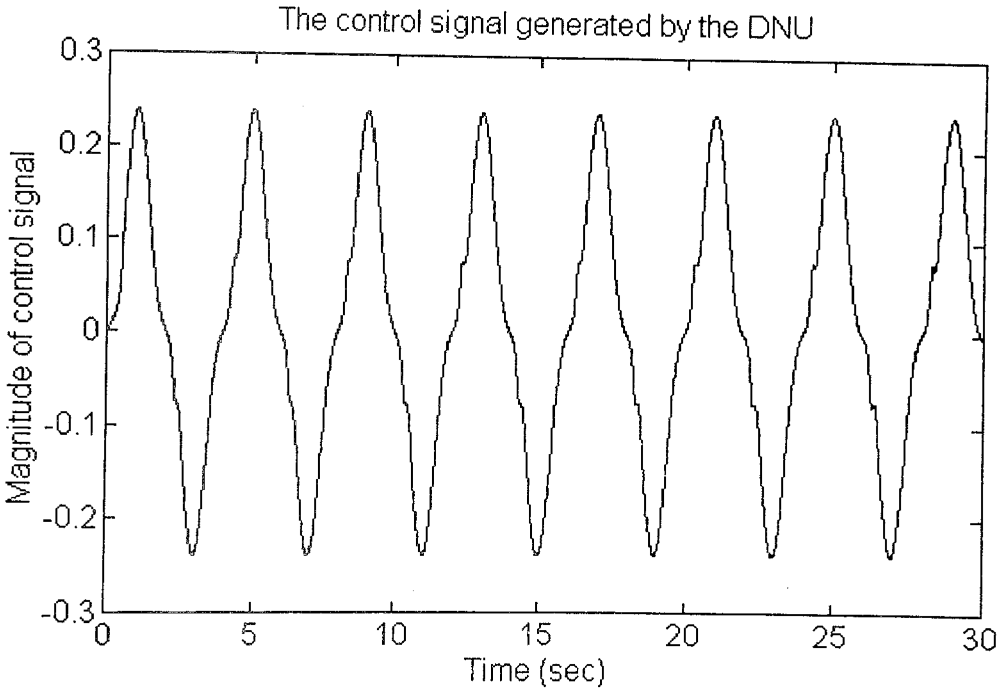


Figure 9.6(c) Control signal applied to the plant defined by Equation (9.21) and generated by DNU

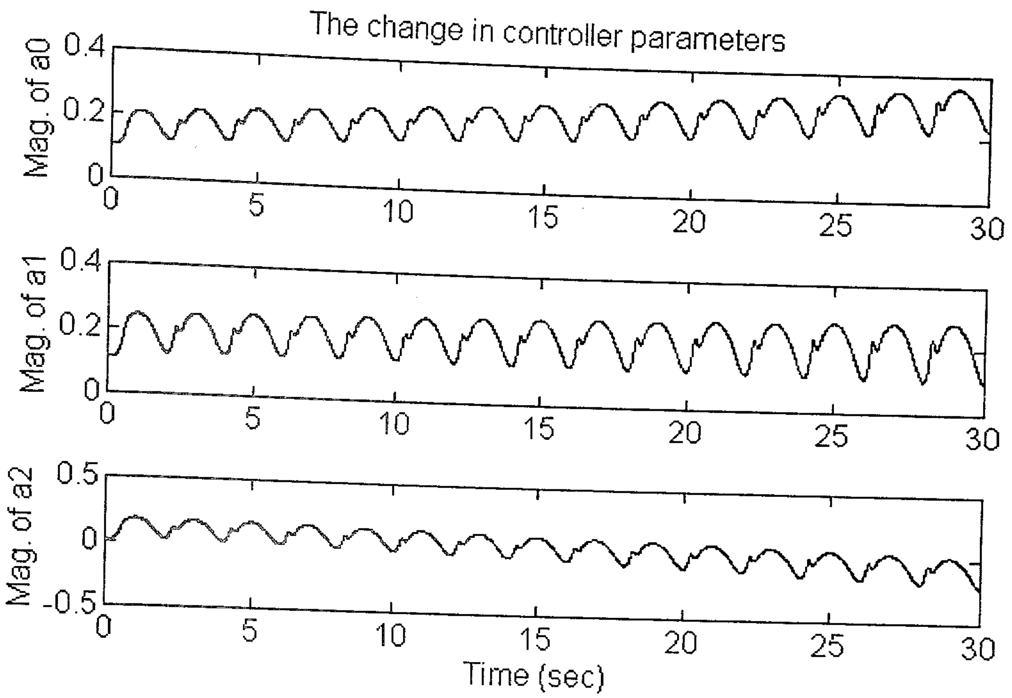


Figure 9.6(d) The changes observed at the feedforward synaptic weights of the DNU

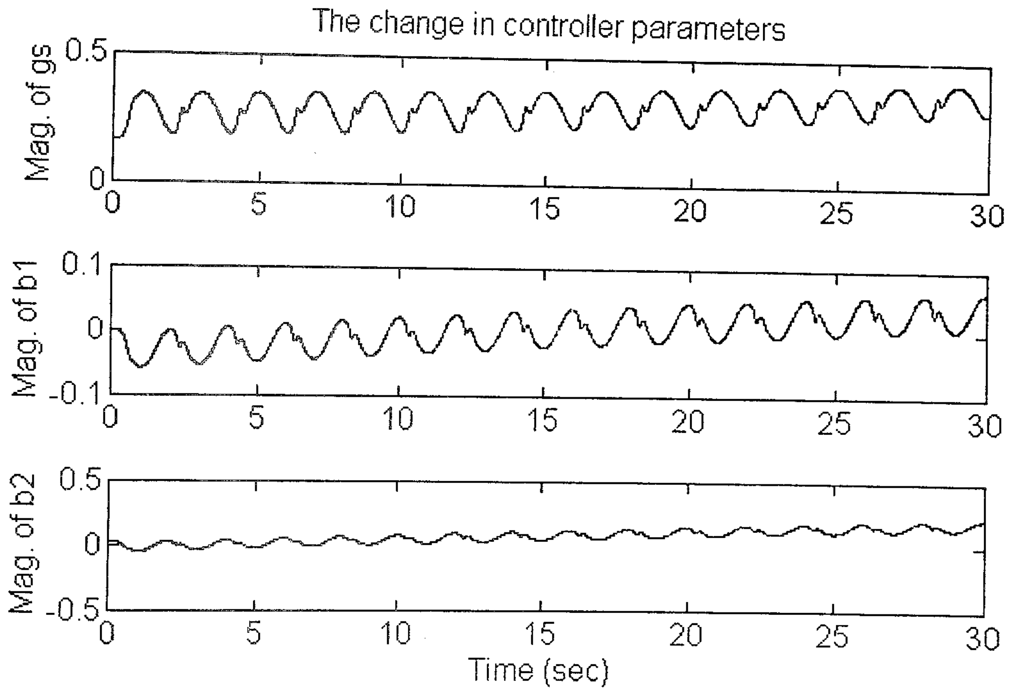


Figure 9.6(e) The changes observed at the somatic gain and the feedback synaptic weights of the DNU

Simulation 4

Plant :

$$y(k+1) = \begin{cases} \dfrac{\sin(0.9y(k) + u(k))}{1 + 0.8\cos(y(k) + u(k))} & 0 \le t < 7.5 \text{ and } 15 \le t < 22.5 \\ \dfrac{u^2(k-1)}{\sqrt{1 + y^2(k-1) + y^2(k-2)}} + \sin(y(k) + u(k)u(k-1)u(k-2)) + u(k) & 7.5 \le t < 15 \text{ and } 22.5 \le t \le 30 \end{cases} \tag{9.23}$$

$$\text{Input : } \sin\left(\frac{\pi t}{2}\right) \tag{9.24}$$

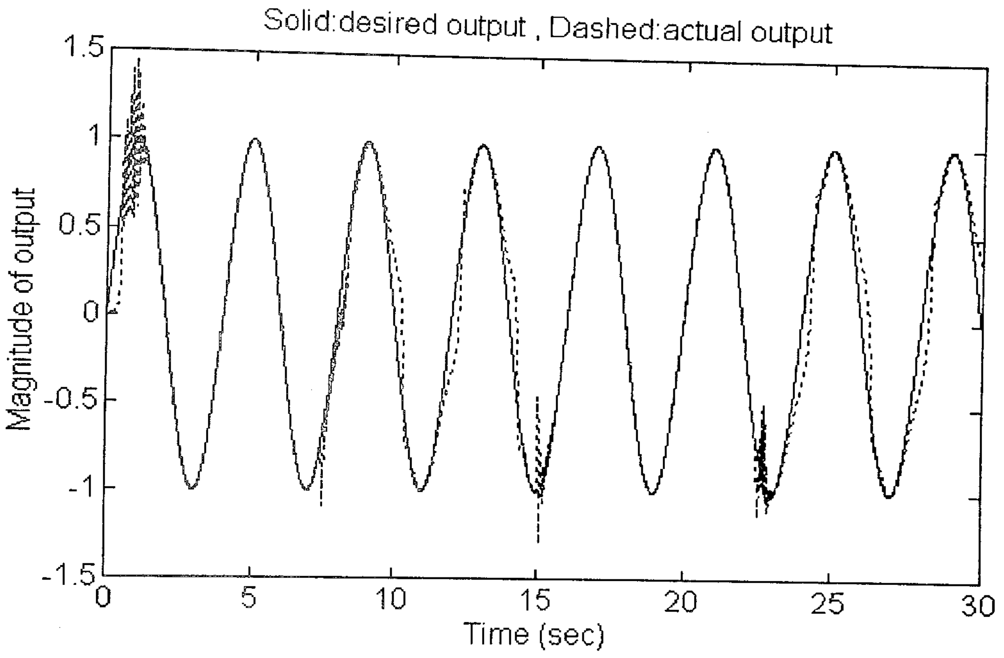


Figure 9.7(a) Output of the plant defined by Equation (9.23) in response to the input defined by Equation (9.24)

The success of the DNU based controller in terms of the parameter adaptation and fast response is apparent in this simulation too. The nonlinearity of the plant under control was regularly switched between two different models and the dynamical neural controller adapted itself quickly. This can also be seen from Fig. 9.7(b). The error trend for $t \in [0,7.5)$ is similar to $t \in [15,22.5)$, the same appearance is valid for the intervals $t \in [7.5,15)$ and

$t \in [22.5,30]$.The effect of this switching shows itself as spikes occurring at the corresponding instants.

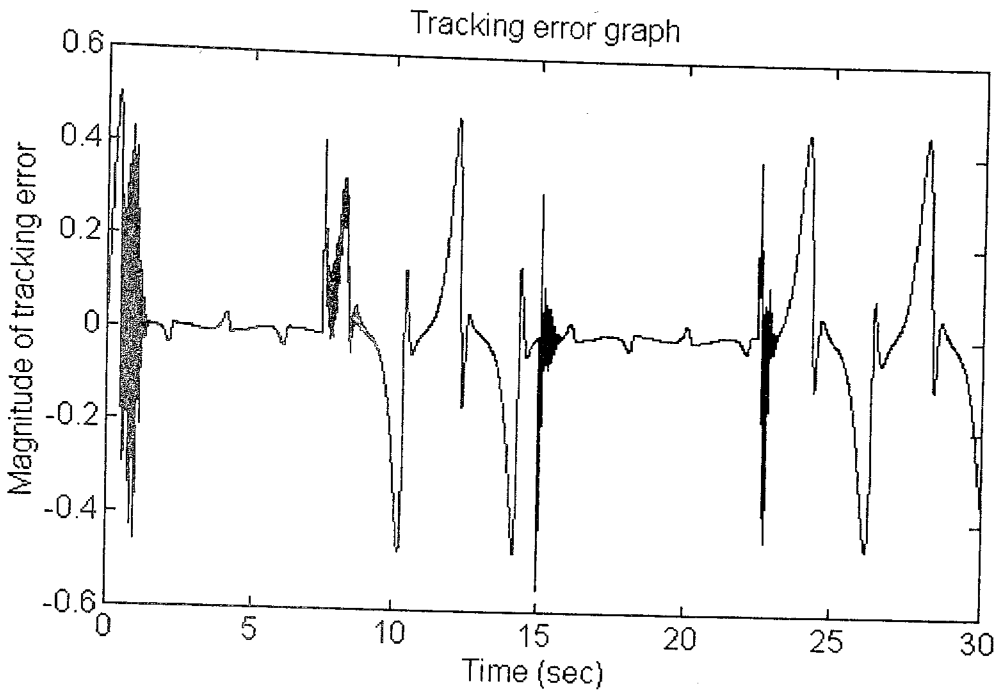


Figure 9.7(b) Output error for the plant defined by Equation (9.23) in response to the input defined by Equation (9.24)

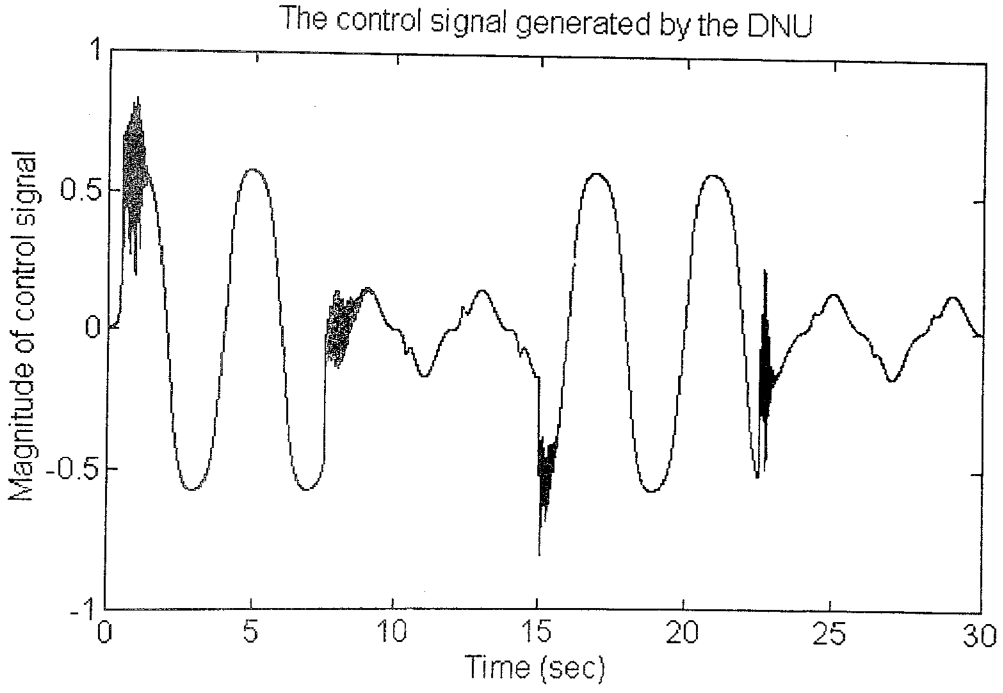


Figure 9.7(c) Control signal applied to the plant defined by Equation (9.23) and generated by DNU

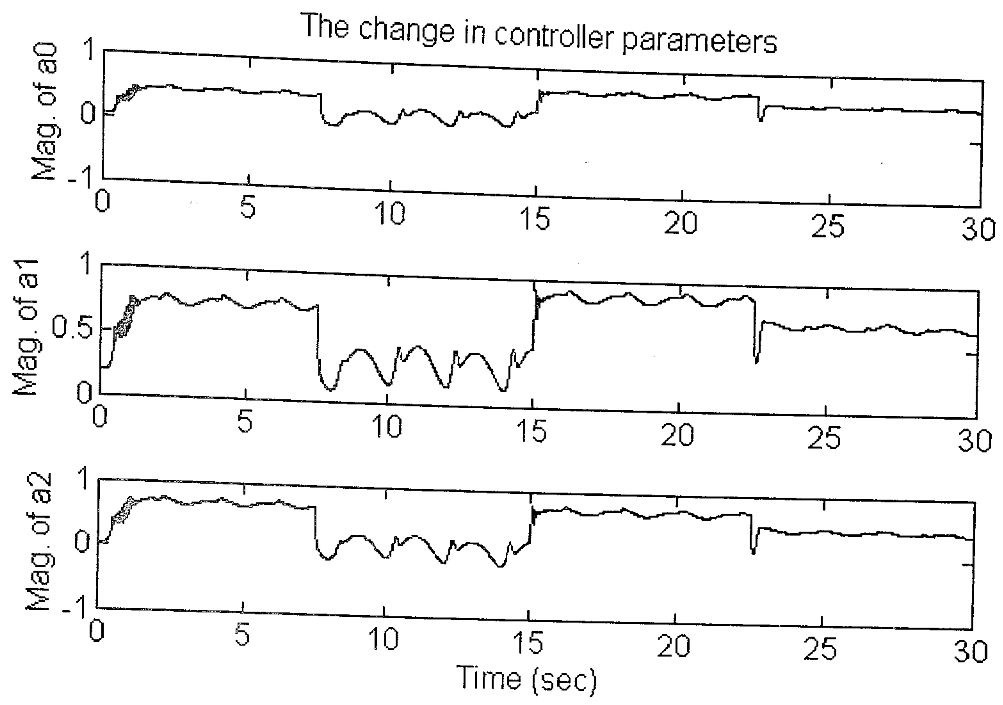


Figure 9.7(d) The changes observed at the feedforward synaptic weights of the DNU

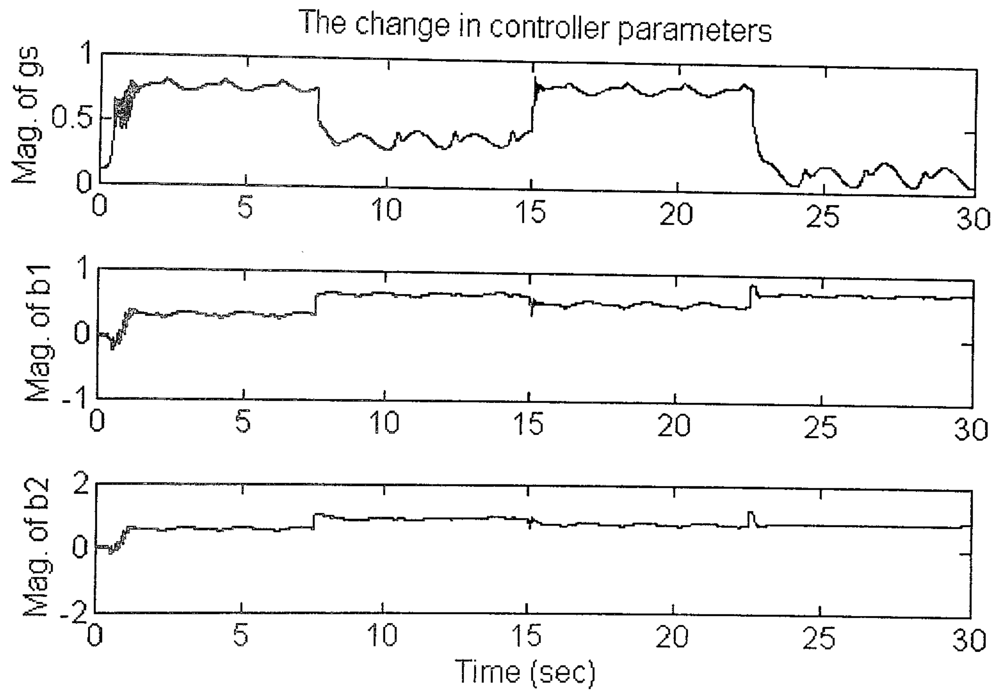


Figure 9.7(e) The changes observed at the somatic gain and the feedback synaptic weights of the DNU

Simulation 5

$$\text{Plant} : y(k+1) = \frac{\sin(0.9y(k) + u(k))}{1+0.8\cos(y(k)+u(k))} \tag{9.25}$$

$$\text{Input} : \begin{cases} \sin\left(\dfrac{\pi t}{2}\right) & 0 \le t < 8 \\ 0.5\,\mathrm{sgn}\left(\sin\left(\dfrac{2\pi t}{4}\right)\right) & 8 \le t < 16 \\ 2.1\cos(t)\exp(-0.09t) & 16 \le t < 30 \end{cases} \tag{9.26}$$

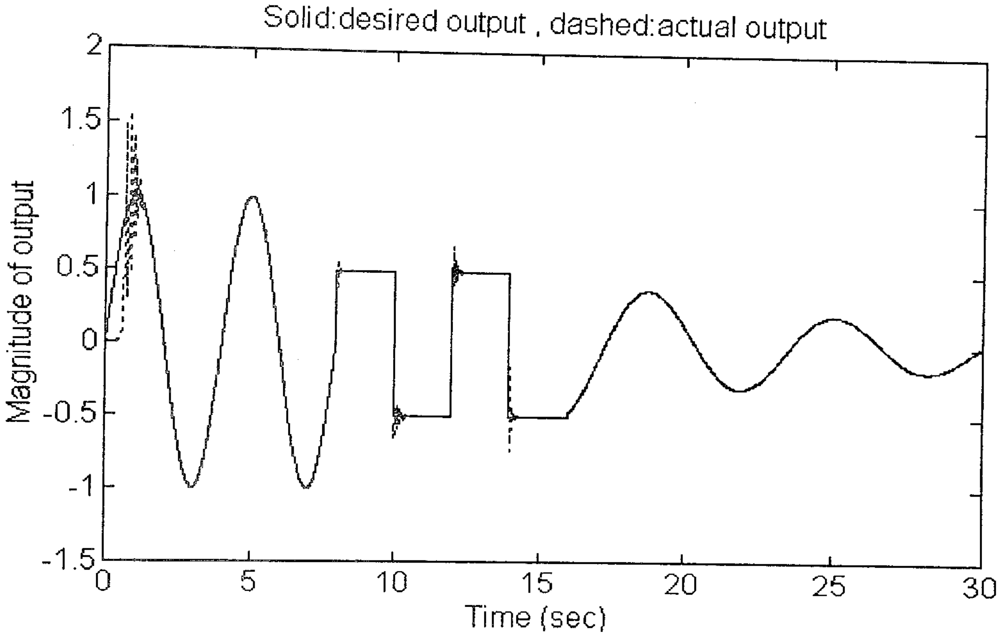


Figure 9.8(a) Output of the plant defined by Equation (9.25) in response to the input defined by Equation (9.26)

In this simulation, we tested the plant defined by Equation (9.25) for different input types. Our aim was to demonstrate the performance of the dynamical neural unit under the condition that considerable variations occur in the input signal. As can be seen from Fig. 9.8(a), a single DNU is able to control the plant even in the occurrence of large variations in the input signal.

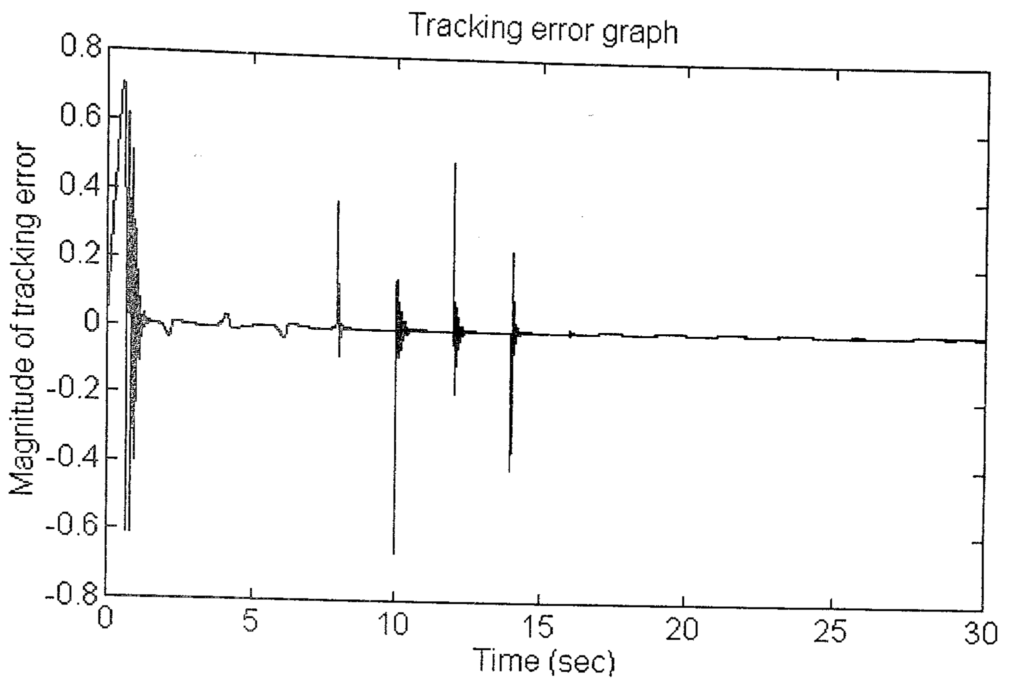


Figure 9.8(b) Output error for the plant defined by Equation (9.25) in response to the input defined by Equation (9.26)

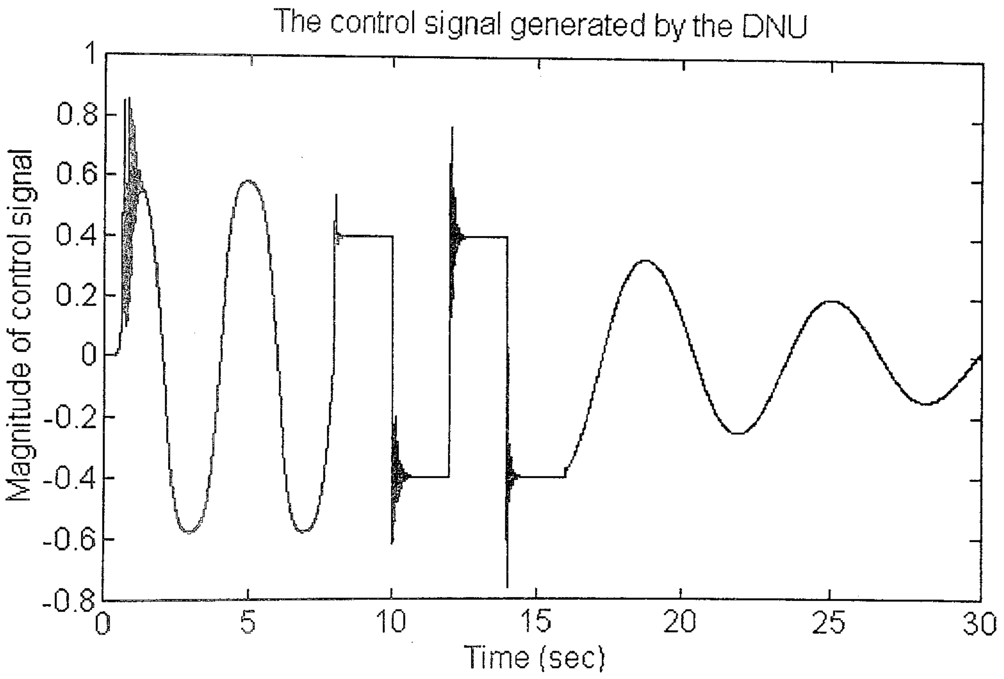


Figure 9.8(c) Control signal applied to the plant defined by Equation (9.25) and generated by DNU

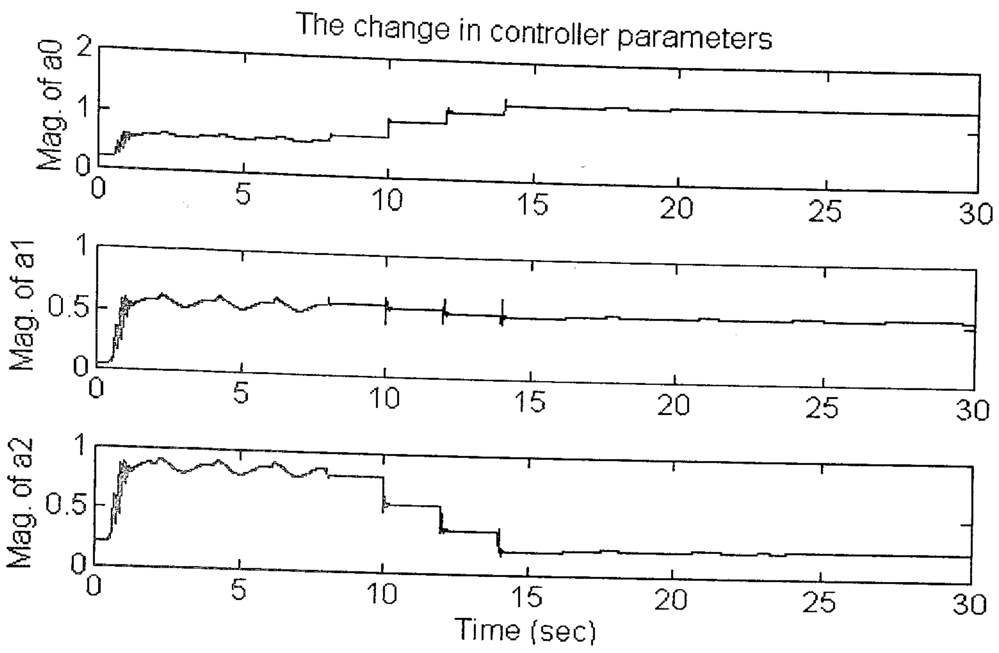


Figure 9.8(d) The changes observed at the feedforward synaptic weights of the DNU

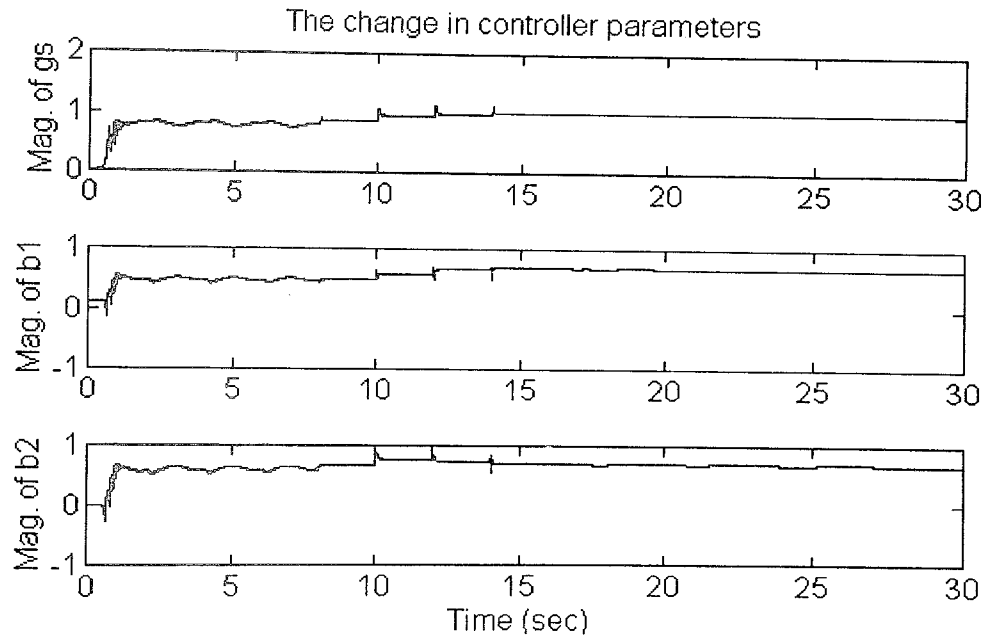


Figure 9.8(e) The changes observed at the somatic gain and the feedback synaptic weights of the DNU

Simulation 6

Plant : Bioreactor model with coupled nonlinear difference equations (Defined in Appendix. A)

$$c_1(k+1) = c_1(k) + \Delta\left(- c_1(k)\,r(k) + c_1(k)\left(1-c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \right) \tag{9.27}$$

$$c_2(k+1) = c_2(k) + \Delta\left(-c_2(k)\,r(k) + c_1(k)\left(1 - c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \frac{1+\beta}{1+\beta-c_2(k)} \right) \tag{9.28}$$

$$\text{Input} : 0.1 + 0.05\sin\left(\frac{2\pi t}{50}\right) \tag{9.29}$$

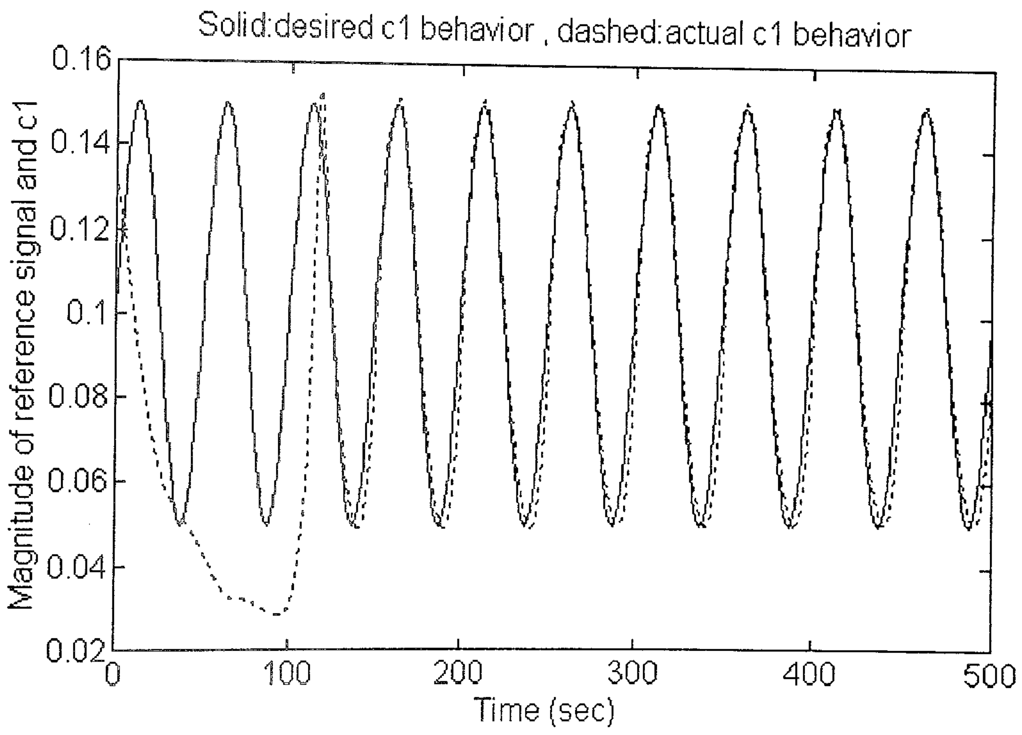


Figure 9.9(a) $c_1(t)$ Output of the bioreactor plant defined by Equations (9.27) and (9.28) in response to the input defined by Equation (9.29)

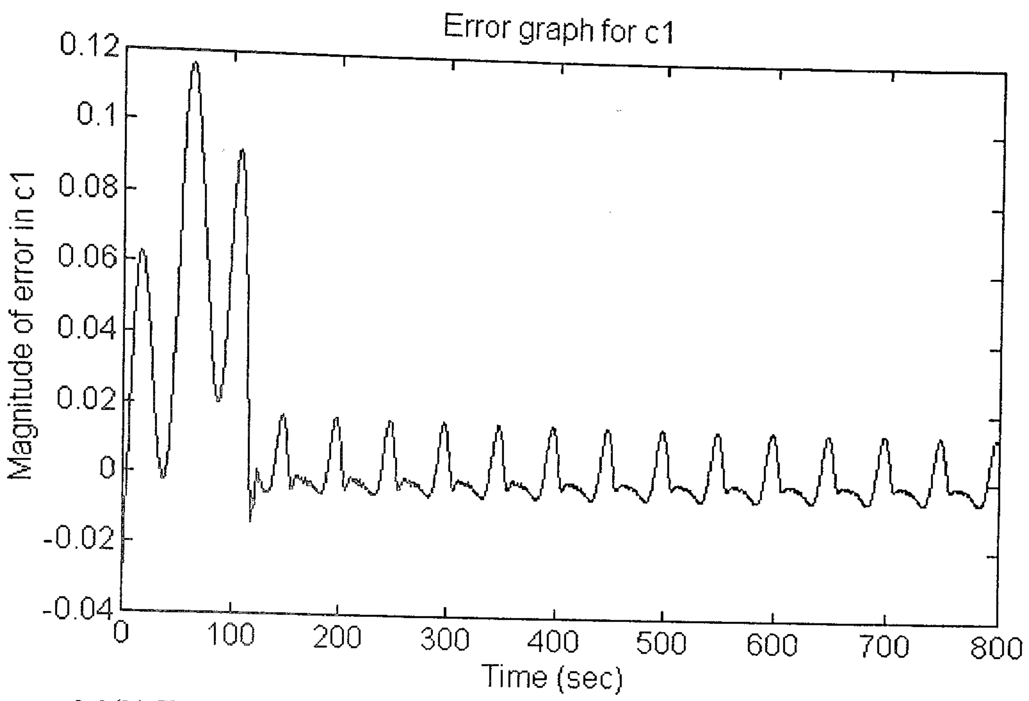


Figure 9.9(b) Tracking error in $c_1(t)$ for the plant defined by Equations (9.27) and (9.28) in response to the input defined by Equation (9.29)

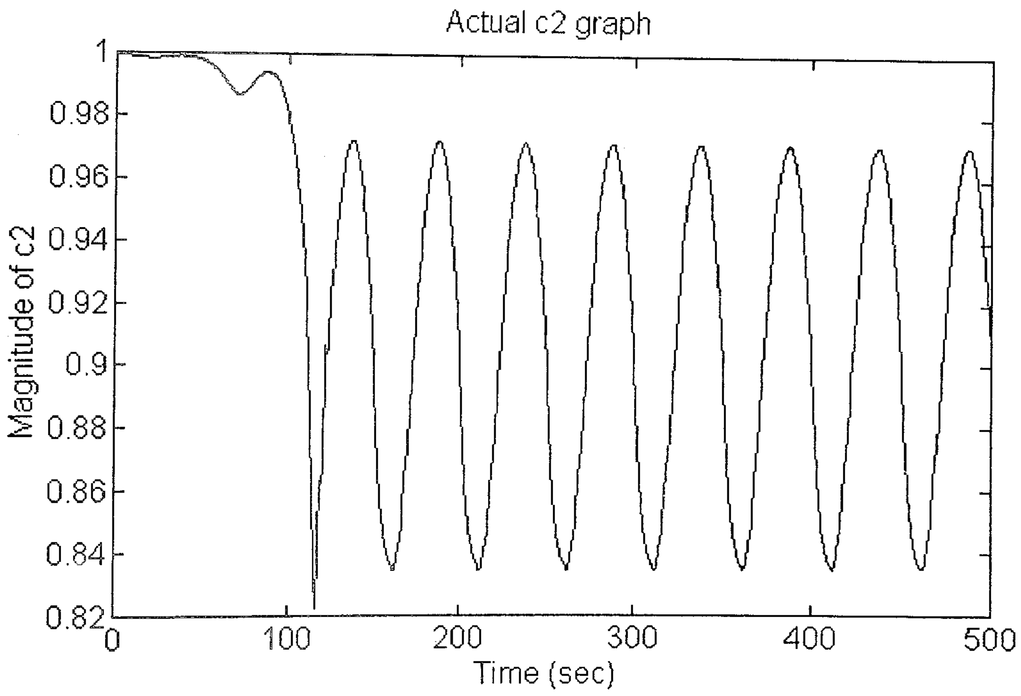


Figure 9.9(c) $c_2(t)$ Output of the bioreactor plant defined by Equations (9.27) and (9.28) in response to the input defined by Equation (9.29)

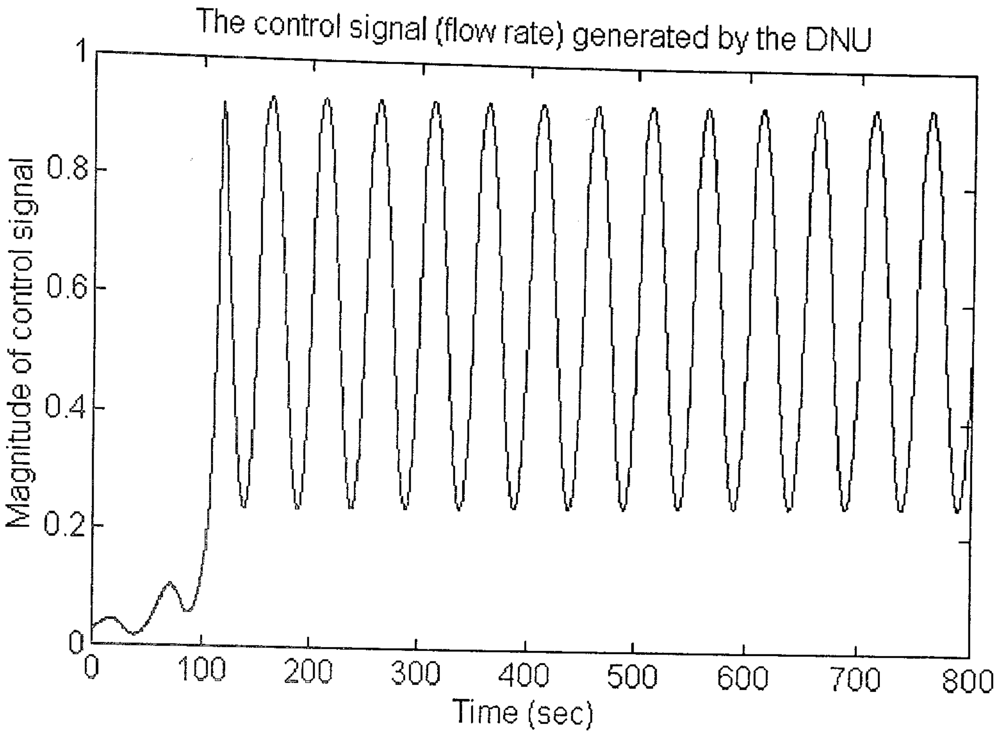


Figure 9.9(d) Control signal applied to the plant defined by Equations (9.27) and (9.28) and generated by DNU

Simulation 7

Plant : Bioreactor model with coupled nonlinear difference equations (Defined in Appendix. A)

Input : $0.05 + 0.01 \sum_{i=1}^{8} u(t-200i)$, u(t) is the unit step function (9.30)

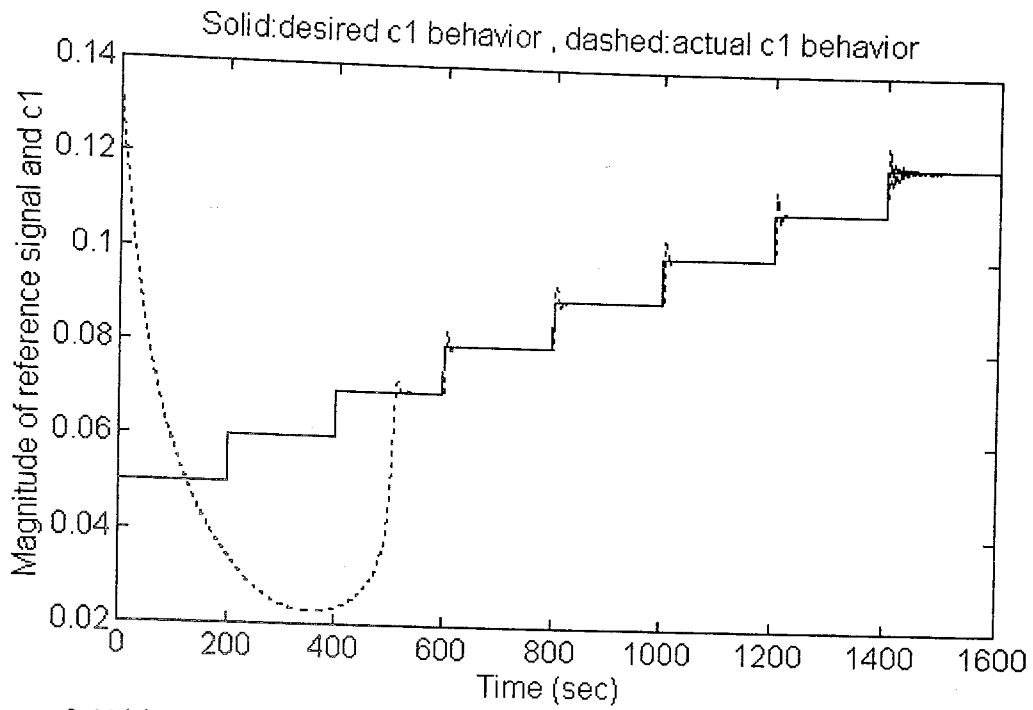


Figure 9.10(a) $c_1(t)$ Output of the bioreactor plant defined by Equations (9.27) and (9.28) in response to the input defined by Equation (9.30)

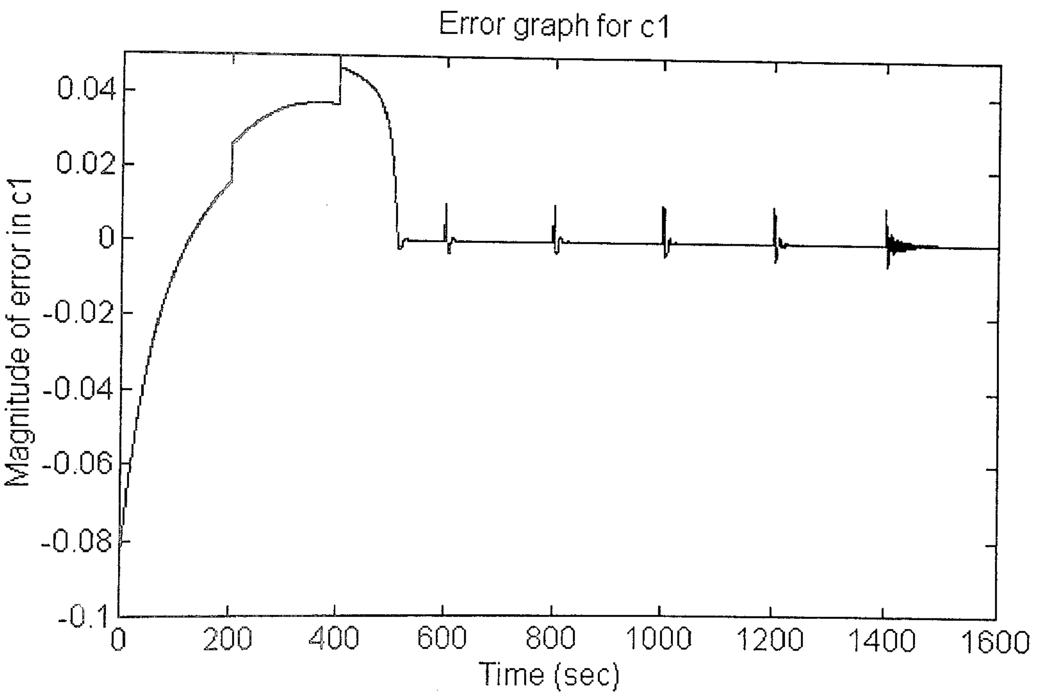


Figure 9.10(b) Tracking error in $c_1(t)$ for the plant defined by Equations (9.20) and (9.21) in response to the input defined by Equation (9.23)

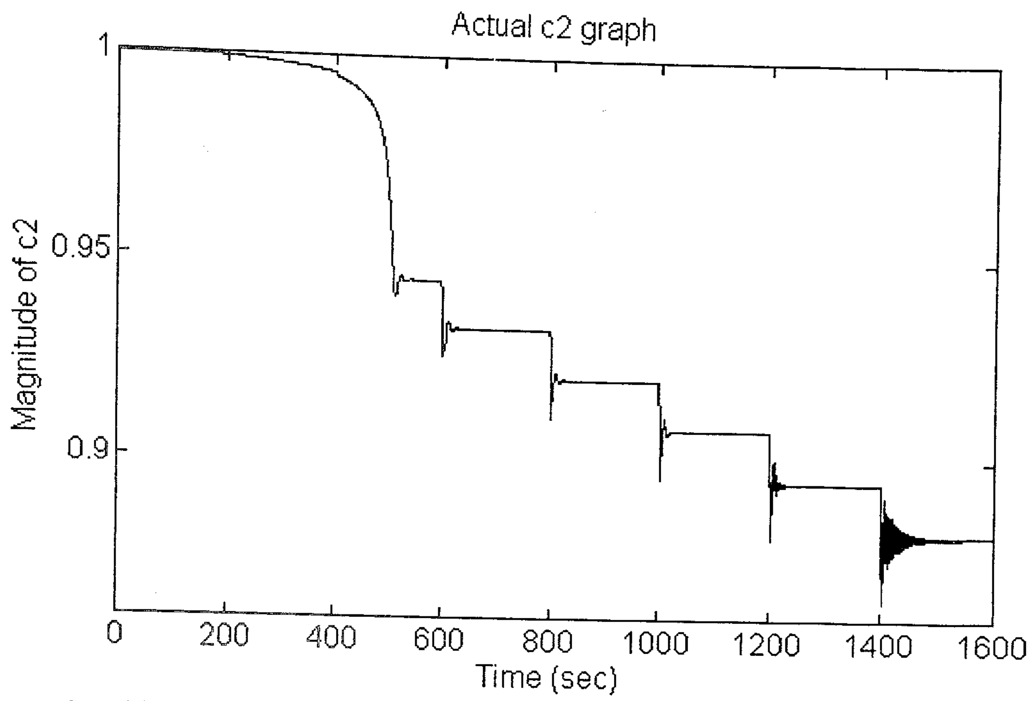


Figure 9.10(c) $c_2(t)$ Output of the bioreactor plant defined by Equations (9.27) and (9.28) in response to the input defined by Equation (9.30)

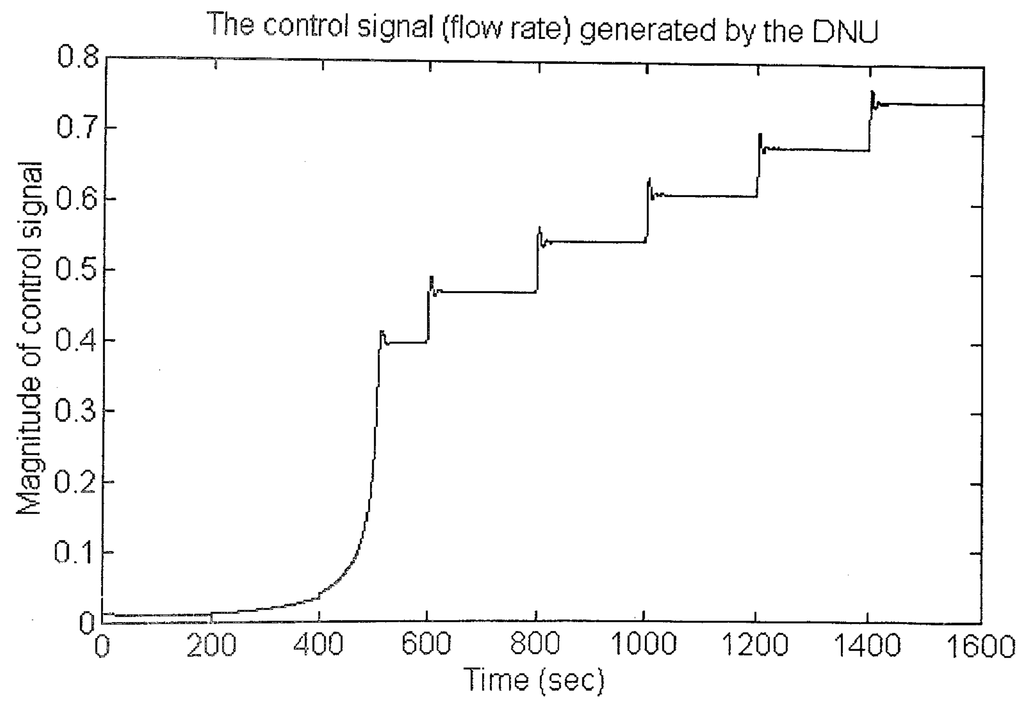


Figure 9.10(d) Control signal applied to the plant defined by Equations (9.27) and (9.28) and generated by DNU

Simulation 8

Plant : Bioreactor model with coupled nonlinear difference equations (Defined in Appendix. A)

Input : 0.1207 (desired level is constant)

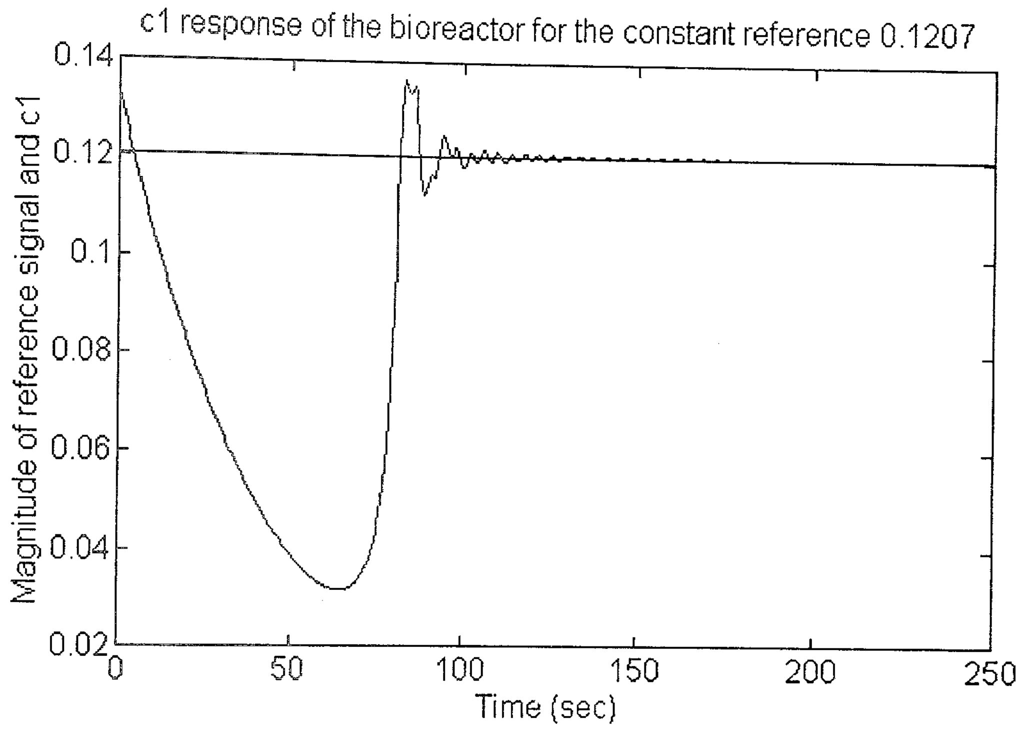


Figure 9.11(a) $c_1(t)$ Output of the bioreactor plant defined by Equations (9.27) and (9.28) in response to the constant input 0.1207

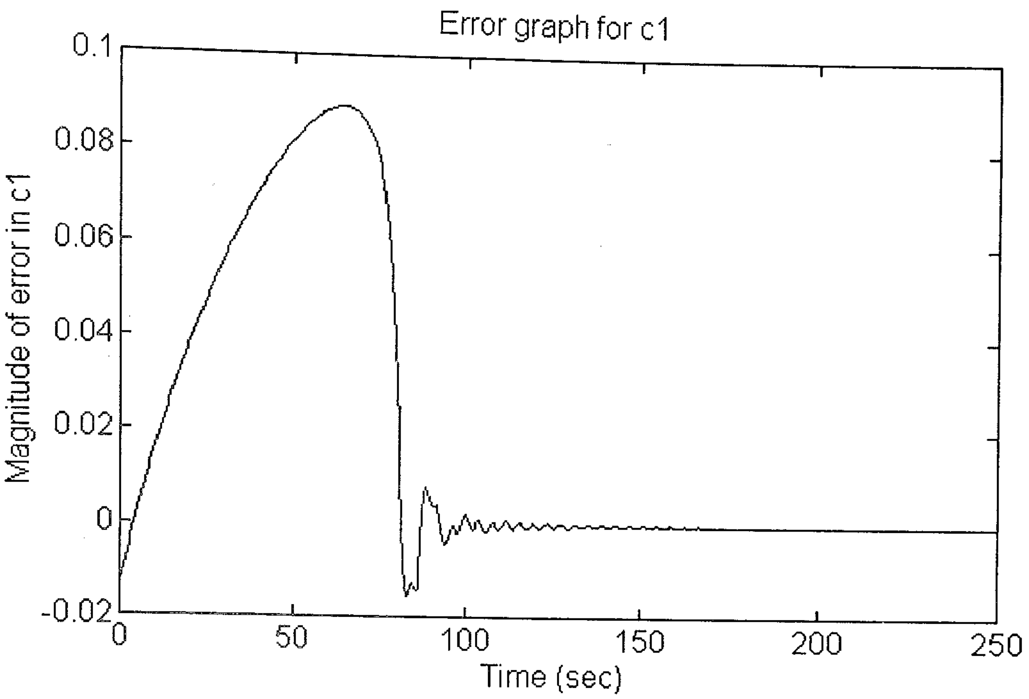


Figure 9.11(b) Tracking error in $c_1(t)$ for the plant defined by Equations (9.27) and (9.28) in response to the constant input 0.1207

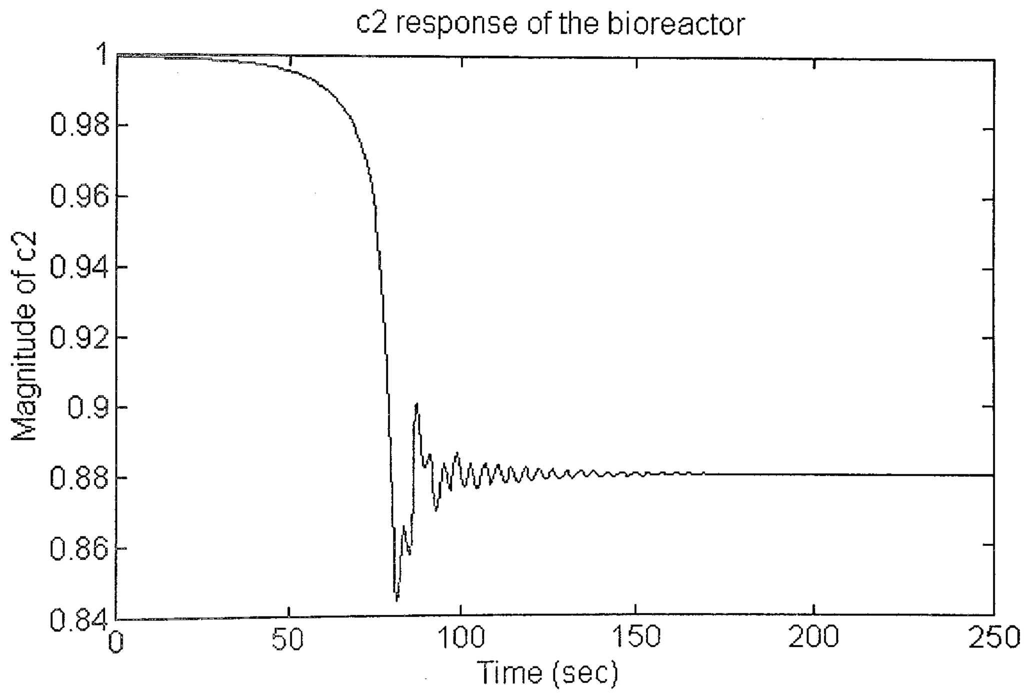


Figure 9.11(c) $c_2(t)$ Output of the bioreactor plant defined by Equations (9.27) and (9.28) in response to the constant input 0.1207

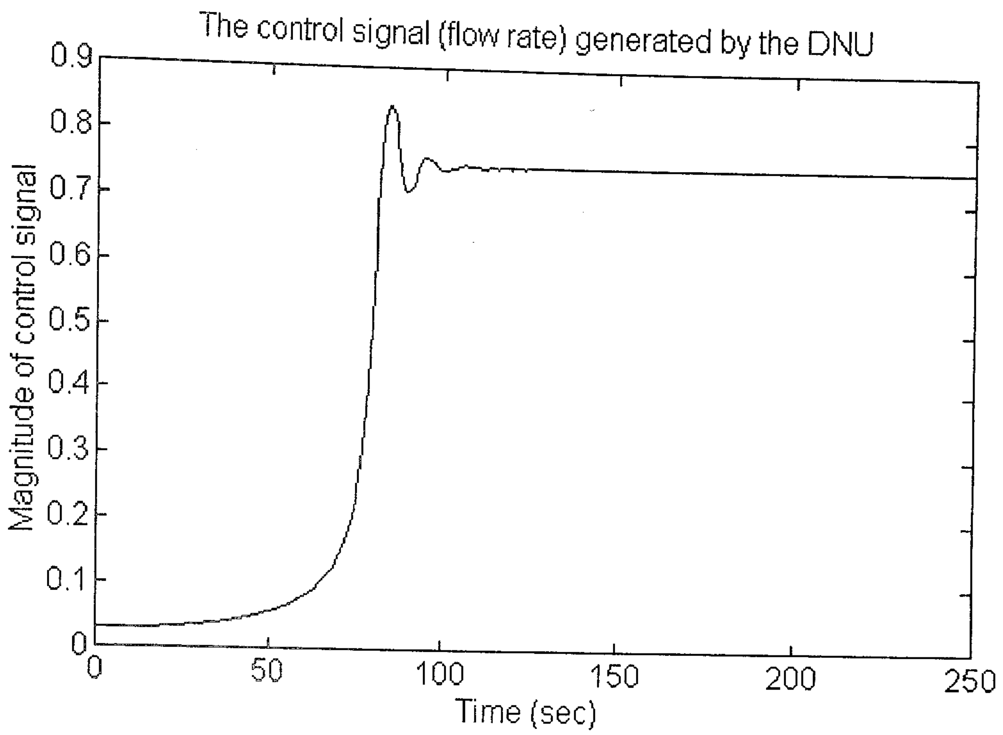


Figure 9.11(d) Control signal applied to the plant defined by Equations (9.27) and (9.28) and generated by DNU

9.3 Performance Assessment

In this chapter, we discussed the dynamical neuron model for the control of nonlinear systems. Our assessment perspective includes three performance metrics. The first one is the occurrence of wild changes in the nonlinearity of the plant models. The aim was to demonstrate the adaptation capability of the DNU based controller training algorithm. In the first and fourth simulations we tested this aspect of the method. Our simulations showed that the resulting DNU based controller is robust under parametric perturbations. The second performance metric is the level of noise rejection property. The first and second simulations deal with this topic. As can be seen from Fig. 9.3(a) and Fig. 9.4(a), even in the case that the observed state value is corrupted by the 10% Gaussian random noise additively, the performance of the controller is again at an admissible level. The third performance metric is the response to the variations in the input signal. In the fifth simulation, we considered this case and applied different types of inputs successively. Our simulation results stipulated that the controller is able to adapt itself such that the input signal is followed as quickly and as perfectly as possible. The most important property of the approach is that the stability of the controller can be explained since it is comprised of a second order dynamical system and a sigmoidal logistic function. In the Tables 9.1 through 9.9 we give the positions of the zeroes and the poles of the resulting DNU based controller in the z-plane. Stable poles or zeroes are represented by (S), unstable ones are represented by (U). In conjunction with all of these, for the first five simulations, a single DNU adequately controlled the plant. In the bioreactor control problem we used ten DNU blocks in the DNU layer.

TABLE 9.1 Resulting controller poles and zeroes for the first simulation

	Zero 1	Zero 2	Pole 1	Pole 2
$0 \le t < 15$	-0.6460+j0.9918 (U)	-0.6460+j0.9918 (U)	0.0439+j0.1494 (S)	0.0439-j0.1494 (S)
$15 \le t \le 30$	-0.6297+j0.9823 (U)	-0.6297-j0.9823 (U)	0.2789 (S)	-0.1179 (S)

TABLE 9.2 Resulting controller poles and zeroes for the second part of the first simulation (Noise addition)

	Zero 1	Zero 2	Pole 1	Pole 2
$0 \le t < 15$	-1.0962+j1.6709 (U)	-1.0962-j1.6709 (U)	-0.0639+j0.2901 (S)	-0.0639-j0.2901 (S)
$15 \le t \le 30$	-0.7156+j1.2030 (U)	-0.7156-j1.2030 (U)	0.1812 (S)	-0.1533 (S)

TABLE 9.3 Resulting controller poles and zeroes for the second simulation (Noise addition)

	Zero 1	Zero 2	Pole 1	Pole 2
$0 \le t \le 30$	-0.6203+j0.7805 (S)	-0.6203-j0.7805 (S)	-0.1115+j0.6904 (S)	-0.1115-j0.6904 (S)

TABLE 9.4 Resulting controller poles and zeroes for the third simulation

	Zero 1	Zero 2	Pole 1	Pole 2
$0 \le t \le 30$	-0.3194 (S)	-0.5591 (S)	-0.0160+j0.2826 (S)	-0.0160-j0.2826 (S)

TABLE 9.5 Resulting controller poles and zeroes for the fourth simulation

	Zero 1	Zero 2	Pole 1	Pole 2
$0 \le t < 7.5$	-0.8149+j0.9077 (U)	-0.8149-j0.9077 (U)	-0.1379+j0.7137 (S)	-0.1379-j0.7137 (S)
$7.5 \le t \le 15$	-0.3161 (S)	-1.3629 (U)	-0.3291+j0.9121 (S)	-0.3291-j0.9121 (S)

TABLE 9.6 Resulting controller poles and zeroes for the fifth simulation

	Zero 1	Zero 2	Pole 1	Pole 2
$0 \leq t < 8$	- 0.4681+j1.0966 (U)	- 0.4681-j1.0966 (U)	-0.2171+j0.6967 (S)	-0.2171-j0.6967 (S)
$8 \leq t < 16$	-0.2742+j0.6334 (S)	-0.2742-j0.6334 (S)	-0.3036+j0.7902 (S)	-0.3036+j0.7902 (S)
$16 \leq t \leq 30$	-0.2087+j0.3506 (S)	-0.2087-j0.3506 (S)	-0.3417+j0.7558 (S)	-0.3417+j0.7558 (S)

In brief, the method introduced in this chapter exhibited high performance from several performance metrics, additionally, the stability of the resulting controller can be analyzed easily when it is compared with the classical neuron model based controllers.

10. DISCUSSION AND CONCLUSIONS

In this thesis, we considered the artificial neural networks from the control engineering perspective. We simulated various types of plant models with the architectures and control strategies explained throughout the study.

The backpropagation learning method and the Levenberg-Marquardt optimization technique were examined as training methods. We first tested the performance of these two methods on the well-known XOR problem. Next, the varieties, namely momentum term and adaptive learning rate, that are introduced to the backpropagation algorithm, were studied. Nevertheless, none of these varieties could make the backpropagation method as successful as the Levenberg-Marquardt technique. The success of this method is attributed to the fact that it employs both the first and the second order derivatives of the performance function and goes to the minima rapidly. These two methods are utilized in the identification and control of the nonlinear dynamical systems whose numerous simulation results are presented in this study.

It is well known that the concept of identification plays an important role in the systems and control area. This, mainly, stems from the fact that we generally want to obtain an approximate model of the system, such that the resulting model is mathematically tractable and it closely matches the system dynamics. Neural networks were proven to be successful identifiers, because they can learn any kind of nonlinear mapping with any degree of accuracy. From this point of view, neural networks can identify a system so that the identification model could be used to devise a controller. Training range is another important point in the neural identification procedure. Neural networks can realize the system's dynamics solely in the range of interest. This range, without loss of generality, includes the permissible operating points.

Neural network controller design is a natural consequence of neural identification process. In this study, we elaborated the controller training by means of error backpropagation. This approach needs an emulator, which is the neural identification model of the system to be controlled. The reason that we use an emulator instead of the actual plant is that, the output error can easily be backpropagated through the plant emulator. The necessary condition is that the plant emulator must realize the actual plant's behavior in the range of interest. Different plant models were simulated. It has been found that if the equations of the plant model are simple, the tracking performance is at an admissible level. As the complexity of the system equations increases, training takes a long time. Consequently, the method yields a neurocontroller that cannot fulfill the performance objectives as well as we desired. We proposed a PI control support in order to improve the performance of the overall control system. The main reason that lies behind our proposal was that the neurocontroller caused a tracking error even in the case where the reference signal is extremely smooth. It is well known that this kind of an error can be reduced by the use of a conventional controller that includes integral action. The simulation results revealed that the tracking ability could be improved by means of this method. Nevertheless, the overall performance, again, is not as satisfactory as we expected. Because the proportional and integral gain constants were chosen experimentally.

Another way of controller training is the inversion of the plant dynamics. If the state transition knowledge and the control that causes this transition are known, this approach can be applicable. In some sense, this implies the observability of the state variables. We applied the method to a bioreactor plant and ended up with unsatisfactory results. The main problem in this approach is that the invertibility or non-invertibility of the plant dynamics, especially for a real system, cannot be foreseen easily. Therefore, if the plant possesses a non-invertible dynamics, this method will only yield a poor approximate of the inverse dynamics and the resulting neurocontroller cannot fulfill the control objectives satisfactorily. As we expected, bioreactor control trial showed sharp deviations around the desired operating points because of the nonlinearities and long time delays introduced to the problem and the method's inadequacy in the inversion of the plant equations.

The next strategy is the self-tuning adaptive control using neural networks. The method is applicable to a fixed type of plant models. Additionally, the method presumes that an extra knowledge dealing with the sign of the nonlinearity that appears in the plant model. The strategy operates on-line and is able to compensate model mismatches. The tracking performance is relatively better than the error backpropagation based controller training method.

Model reference adaptive control (MRAC) is also considered in this work. Specifically, the method was designed so that the plant under control follows the output of a stable reference model. We mainly used the ideas proposed by Narendra and Parthasarathy [5] and Narendra and Boskovic [12]. The main conclusions about this control strategy can be summarized as follows : Firstly, neural controller provides the value of the nonlinear function that exist in the plant dynamics. Secondly, the error equation has to be evaluated and it has to be proven that the output error tends to zero in the limiting case. This implies that the plant output follows the output of a reference model. The method was firstly applied to a SISO plant and a MIMO plant which were used in [5] and the results were duplicated. Our last simulation for MRAC strategy was the control of the bioreactor plant. A detailed analysis was carried out for the bioreactor control problem. All of our simulation results showed that the method did better than the strategies explained earlier in the sense of output tracking ability. It should be designated that we used the direct adaptive control scheme in our MRAC simulations.

The next control strategy is the self-learning control scheme. The most remarkable feature of this approach is its goal directed behavior. During the normal operation, no external reference or command signal is provided to the control system. The controller is trained so that, it brings the plant outputs to their desired values regardless of initial conditions. Desired plant outputs are defined before training and the controller is trained for these values. Nguyen and Widrow [6] proposed the controller training architecture and they tested the method on a truck backer-upper example. We applied the method to different plants and the bioreactor plant. Our simulation results, especially the bioreactor control problem, showed that the resulting neural controller is able to achieve performance specifications such as fast response and small tracking error. The controller training structure is comprised of a chain-like architecture and utilizes the error backpropagation through the components of this architecture. In our simulations, we trained the neural controller by initializing the plant outputs to many positions which are possible initial outputs that lie in the state space. We did the same initialization during the normal operation. Plant output, after a while, settled down to a fixed value which is close to the desired value of the plant output. In order to see the performance of the controller, in the normal operation, we initialized the plant output at some certain instants. Our observations for this case stipulated that the resulting neural controller is able to tolerate some failures that occur in the normal operation.

The last control strategy differs from previous ones in that, it employs a different neuron model which is explained in detail by the theory of discrete time control systems. The method employs a dynamical neuron model which is comprised of a second order discrete pulse transfer function and a nonlinear activation function. The training methodology adopts the simple gradient descent rule in determining the coefficients of the second order system and the slope of the nonlinear activation function. Gupta and Rao [7] separated the parameter adaptation into two main parts: the first part includes the adaptation of synaptic feedforward and feedback gains which are the coefficients of the second order block. The second part is the adaptation of the somatic gain which is the slope of the nonlinear activation function. The method is applied to many plant models with different reference trajectories. Our numerous simulation results revealed that the method is more successful than the other methods considered in this thesis. For this comparison, output tracking ability, adaptation capability under some considerable perturbations such as observation noise, changes in the plant parameters and changes in the nonlinearity were used as performance measures. The most important property of the method is that the resulting neurocontroller is mathematically tractable in the sense of stability analysis. Since the dynamical part of the neuron model is a second order discrete pulse transfer function, the stability concepts form discrete time control theory can be used in explaining the stability of the controller.

A common problem of the methods explained so far but the last one, is that there is no concrete study that explains the stability of neural controllers. Still, this constitutes a serious problem that needs to be explained when artificial neural networks are of interest. Another important question that is to be answered is the size of the neural network, and the value of learning parameters such as convergence rate, momentum coefficient etc. Which parameter set is optimal and which network configuration does the best for our desires are still unanswered questions as well as the stability problem.

The conclusions obtained in our simulations are given compactly in Table 10.1.

TABLE 10.1 Comparison of neural network based control algorithms

	Error BP	Plant Inversion	Self-Tuning Control	MRAC	Self-Learning Control	Dynamical Neural Unit
Tracking Performance	LOW	LOW	MIDDLE	HIGH	HIGH	HIGH
Applicability to different plants	MIDDLE	LOW	LOW	MIDDLE	HIGH	HIGH
Robustness under perturbations	LOW	LOW	LOW	HIGH	HIGH	HIGH
Mathematical tractability for stability analysis	NO	NO	NO	NO	NO	YES
Noise reduction	-	-	-	-	-	HIGH
Capability of Fault Tolerance	-	-	-	-	MIDDLE	HIGH

As we explained throughout this study, artificial neural networks as systems which are capable of *learning*, opened a new horizon in the area systems and control. If the nature that lies behind the machine learning is explained adequately, more complex tasks can be achieved. Although the results are not perfect, the subject itself keeps its mystery in the sense of machine learning. In the future, new neuron models can be devised so that the drawbacks of the classical neuron model of the artificial neural networks can be overcome and more versatile controllers can be designed.

Appendix A

Definition of the Bioreactor Benchmark Problem

The bioreactor is a tank containing water, nutrients, and biological cells as shown in Fig. A.1. Nutrients and cells are introduced into the tank where the cells mix with nutrients. The state of this process is characterized by the number of cells $c_1(t)$ and the amount of nutrients $c_2(t)$. The volume in the tank is maintained at a constant level by removing tank contents at a rate equal to the incoming rate which is denoted by $r(t)$. This rate is called the flow rate and is the variable by which the bioreactor is controlled. The bioreactor control problem is to maintain the amount of cells at a desired level.

The state variables can take values between zero and one, the flow rate can take values between zero and two. In [13], the stable state of the process is defined to be $c_1 = 0.1207$, $c_2 = 0.8801$, and $r = 0.7500$; initialization of the state variables and the flow rate is made by considering these values such that the initial value of any parameter is within plus or minus ten percent of its stable value. Moreover, initial values exhibit uniform distribution over their interval and chosen randomly.

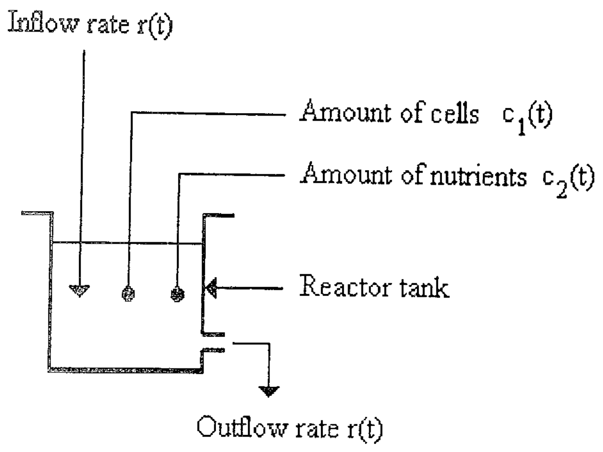


Figure A.1

The bioreactor is a tank of a liquid mixture of cells and nutrients.
The objective is to control the amount of cells by adjusting the flow rate.

Continuous-time equations of the plant dynamics are given by;

$$\dot{c}_1(t) = -c_1(t)r(t) + c_1(t)\left(1-c_2(t)\right)e^{\frac{c_2(t)}{\gamma}} \tag{A1.1}$$

$$\dot{c}_2(t) = -c_2(t)r(t) + c_1(t)\left(1-c_2(t)\right)e^{\frac{c_2(t)}{\gamma}} \frac{1+\beta}{1+\beta - c_2(t)} \tag{A1.2}$$

If these equations are discretized by the use of first order approximation, we obtain;

$$c_1(k+1) = c_1(k) + \Delta\left(- c_1(k)\, r(k) + c_1(k)\left(1-c_2(k)\right) e^{\frac{c_2(k)}{\gamma}}\right) \tag{A1.3}$$

$$c_2(k+1) = c_2(k) + \Delta\left(-c_2(k)\, r(k) + c_1(k)\left(1 -c_2(k)\right) e^{\frac{c_2(k)}{\gamma}} \frac{1+\beta}{1+\beta-c_2(k)}\right) \tag{A1.4}$$

where $\Delta = 0.01$ sampling interval, $\beta = 0.02$ growth rate parameter, $\gamma = 0.48$ nutrient inhibition parameter. Controller inputs are the state variables. The control interval is defined to be 50Δ. The output of the controller is the flow rate r(t). The objective is to achieve and maintain a desired cell amount, by altering the flow rate throughout a learning trial.

The bioreactor is a challenging problem for neural network controllers for several reasons. Although the task involves few variables and is easily simulated, its nonlinearity makes it difficult to control. For example, small changes in parameters' values can cause the bioreactor to become unstable. The issues of delay, nonlinearity, and instability can be studied with the bioreactor control problem. Significant delays exist between changes in flow rate and the response in cell concentration. Nonlinearities in the bioreactor's dynamics present a challenge to networks for learning nonlinear models. Additionally, uncontrolled equations exhibit limit cycles. Neural networks that learn to compensate for deficiencies in the performance of the conventional controllers for this task can be tested. This is also a proper problem for investigating combinations of methods for predicting future states with controllers that learn to avoid unstable regions of the state space.

REFERENCES

1. Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proceedings of the National Academy of Sciences, pp. 2554-2558, April 1982.
2. Miller, W. T., Sutton, R. S., Werbos, P. J., *Neural Networks for Control,* MIT Press, 1991.
3. Psaltis, D., Sideris, A., Yamamura, A. A., "A Multilayered Neural Network Controller," *IEEE Control Systems Magazine*, pp. 17-20, April 1988.
4. Chen, F. C., "Back-Propagation Neural Networks for Nonlinear Self-Tuning Adaptive Control," *IEEE Control Systems Magazine*, pp. 44-47, April 1990.
5. Narendra, K. S., Parthasarathy, K., "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp. 4-27, March 1990.
6. Nguyen, D. H., Widrow, B., "Neural Networks for Self-Learning Control Systems," *IEEE Control Systems Magazine*, pp. 18-23, April 1990.
7. Gupta, M. M., Rao, D. H., "Dynamical Neural Units with Applications to the Control of Unknown Nonlinear Systems", *Journal of Intelligent and Fuzzy Systems*, Vol.1, No.1, pp. 73-92, 1993.
8. Funahashi, K., "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks*, Vol. 2, pp 183-192, 1989
9. Hornik, K., "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol. 2, pp 359-366, 1989
10. Narendra, K.S., Levin, A. U., "Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization," *IEEE Transactions on Neural Networks*, Vol. 4, No. 2, pp. 192-206, March 1993.
11. Hagan, M. T., Menhaj, M. B., "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, pp. 989-993, November 1994.

12. Boskovic, J. D., Narendra, K. S., "Comparison of Linear, Nonlinear and Neural-Network Based Adaptive Controllers for a Class of Fed-batch Fermentation Process," *Automatica*, Vol. 31, No.6, pp. 817-840, 1995.

13. Ungar, L. H., "A Bioreactor Benchmark for Adaptive-Network Based Process Control," in W. T. Miller, R. S. Sutton, P. J. Werbos, *Neural Networks for Control*, MIT Press, 1991

REFERENCES NOT CITED

1. Antsaklis, P. J., "Neural Networks in Control Systems," *IEEE Control Systems Magazine*, pp. 8-10, April 1992.
2. Narendra, K. S., Mukhopadhyay, S., "Intelligent Control Using Neural Networks," *IEEE Control Systems Magazine*, pp. 11-18, April 1992.
3. Guez, A., Eilbert, J. L., Kam, M., "Neural Network Architecture for Control," *IEEE Control Systems Magazine*, pp. 22-25, April 1988.
4. Narendra, K. S., Balakrishnan, J., Ciliz, M. K., "Adaptation and Learning Using Multiple Models, Switching, and Tuning," *IEEE Control Systems Magazine*, pp. 37-51, June 1995.
5. Hush, D. R., Horne, B. G., "Progress in Supervised Learning," *IEEE Signal Processing Magazine*, pp. 7-39, January 1993.
6. Hunt, K. J., Sbarbaro, D., Zbikowski, R., Gawthrop, P. J., "Neural Networks for Control Systems - A Survey," *Automatica*, Vol. 28, No.6, pp. 1083-1112, 1992.
7. Hertz, J., Krogh, A., Palmer, R. G., *Introduction to the Theory of Neural Computation*, Addison-Wesley, California, 1991.
8. Antsaklis, P. J., Passino K. M., *An Introduction to Intelligent and Autonomous Control*, Kluwer Academic Publishers, 1993.
9. Abulafya, N., "Neural Networks for System Identification and Control," M.S. Thesis, Imperial College of Science, Technology and Medicine, September 1995
10. Fadıloğlu, A., "Neural Network Control of Nonlinear Systems," M.S. Thesis, Boğaziçi University, 1992
11. Topçuoğlu, H. R., "Comparison of Multilayer Perceptron Algorithms," M.S. Thesis, Boğaziçi University, 1993
12. Astrom, K. J., Wittenmark, B., *Adaptive Control*, Addison-Wesley, 1989
13. Ogata, K., *Discrete-Time Control Systems*, Prentice-Hall, 1987