Nonlinear Plant Control Using Neural Networks: A Design Procedure Through Linearization

by

Imran Ali Tasadduq

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

June, 1994

Order Number 1360417

Nonlinear plant control using neural networks: A design procedure through linearization

Tasadduq, Imran Ali, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1994

NONLINEAR PLANT CONTROL USING NEURAL NETWORKS: A DESIGN PROCEDURE THROUGH LINEARIZATION

BY

IMRAN ALI TASADDUQ

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

JUNE 1994

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN, SAUDI ARABIA

This thesis, written by

Imran Ali Tasadduq

under the direction of his Thesis Advisor, and approved by his Thesis committee, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

Thesis Committee:

[illegible]

Chairman (Dr. M.S.Ahmed)

[illegible]

Member (Dr. Davut Kavranoglu)

[illegible]

Member(Dr. L.Cheded)

[illegible]

Dr. K.S.Al-Sultan
Department Chairman

[illegible]

Dr. Ala H. Rabeh
Dean, College of Graduate Studies

Date: 12. 6. 04

الدراسات العليا

4

NONLINEAR PLANT CONTROL USING NEURAL NETWORKS: A DESIGN PROCEDURE THROUGH LINEARIZATION

IMRAN ALI TASADDUQ

SYSTEMS ENGINEERING

JUNE 1994

Dedicated to

My Mother

Acknowledgments

Praise be to the Lord of the World, the Almighty for having guided me at every stage of my life.

I have been fortunate to have the opportunity to work under the supervision of Dr. M. S. Ahmed. His guidance, valuable suggestions, continuous encouragement and extraordinary support throughout all stages of this research are highly appreciated. I would also like to place on record my appreciation for the cooperation and guidance extended by my committee members, Dr. Davut Kavranoglu and Dr. L. Cheded.

My thanks to the department chairman Dr. Khalid Al-Sultan for providing me every help and facility to carry out this research. I also thank my friend Mr. M. Farooq Anjum, who has been a source of great help during my M.S. I am grateful to all my friends, both from within and outside the department, who made my stay at KFUPM a pleasant and memorable one.

I recall with deep gratitude and respect the debt I owe to my mother, uncle Mr. Sirajuddin and brothers Rizwan and Noman for their unfathomable love, prays, support and encouragement throughout my studies. May Allah bless them and may they live long for me.

Contents

List of Figures

Abstract

Name: Imran Ali Tasadduq

Title: Nonlinear Plant Control Using Neural Networks: A Design Procedure Through Linearization

Major Field: Systems Engineering

Date of Degree: June 1994

A three stage procedure to design neural net controller for nonlinear plants is developed. The design is based upon linearization of the plant at each operating point. In stage one, the input-output data of the plant is used to train a neural net model of the plant. In stage two, the plant is controlled using a time varying linear controller based on the linearized plant model at each operating point. The neural net model of the plant identified in stage one is used to obtain the linearized models. In stage three, a neural net is trained to replace the time varying linear controller. Effect of noise has also been investigated. Simulation results of SISO as well as MIMO plants have been presented.

Master of Science Degree

King Fahd University of Petroleum and Minerals

Dhahran, Saudi Arabia

June, 1994

خلاصة الرسالة

اسم الطالب : عمران علي تصدق.
عنوان الرسالة : تحكم المنشأة الغير خطي بإستخدام الشبكات العصبية: طريقة تصميم من خلال التحويل الى الخطي.
التخصص : هندسة النظم والأساليب.
تاريخ التخرج : محرم ١٤١٥هـ.

تم تطوير طريقة ذات ثلاث مراحل لتصميم ضابط الشبكة العصبية للمنشآت الغيرخطية. يعتمد التصميم على جعل المنشأة خطية في كل نقطة تشغيل. في المرحلة الأولى، يتم استخدام بيانات الإدخال والإخراج للمنشأة لتدريب نموذج الشبكة العصبية للمنشأة. في المرحلة الثانية، يتم التحكم بالمنشأة عن طريق استخدام ضابط خطي متغير مع الوقت اعتماداً على النموذج الخطي للمنشأة عند كل نقطة تشغيل. يستخدم نموذج الشبكة العصبية للمنشأة والذي تُعرف عليه من خلال المرحلة الأولى للحصول على النماذج المحولة الى خطية. في المرحلة الثالثة، يتم تدريب الشبكة العصبية لتحل محل الضابط الخطي المتغير مع الوقت. تم أيضا بحث تأثير التشويش. تتضمن هذه الرسالة نتائج المحاكاة لمنشآت (SISO) و (MIMO).

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران-المملكة العربية السعودية
محرم ١٤١٥هـ

Nomenclature

α_i, β_i	parameters of the plant
y	actual plant output
u	actual plant input
y_d	desired output
e	error
η	training rate coefficient
J	cost function
$f(.)$	activation function
Γ	gradient of the neural network
w	Neural network weights

Chapter 1

INTRODUCTION

Control is a very common concept. This term is used to refer to certain specific human-machine interactions, as in driving an automobile, where it is necessary to control the vehicle if one is to arrive safely at a planned destination. Such systems are called manual controls. Automatic control involves machines only, as in room-temperature control, where a furnace is turned on and off depending on a thermostat reading to control the temperature in winter and a similarly-controlled air conditioner is used to control the temperature in summer. An extensive body of knowledge common to both manual and automatic control has evolved into the discipline of control systems design. The list of variables subject to control is vast, being virtually limited by only one's imagination. In mechanisms such as robots, control has been applied to, for example, position, speed and force. In the chemical industry, control is applied to fluid flow and liquid level, to gas flow and gas pressure, to chemical concentrations and to many other variables. Within the human body, blood pressure, blood sugar, cell carbon dioxide and eye-pupil diameter are only a few of the many variables controlled by automatic biological control mechanisms that can be studied by methods of feedback control. The manifestations of control are very widespread, both in nature and as a result of engineering design and the methods of analysis and design required of control engineers can be very useful indeed [1].

The best-developed aspect of mathematical systems theory treats systems defined by linear operators using well established techniques based on linear algebra, complex variable theory and the theory of linear ordinary differential equations. Since design techniques for dynamical systems are closely related to their stability properties and since necessary and sufficient conditions for the stability of linear time-invariant systems have been generated over the past decades, well-known design methods have been established for such systems [2].

Most control systems encountered in practice are nonlinear. Although it may be possible to represent systems which are perturbed over a restricted operating range, by a linear model, in general, nonlinear processes can only be adequately characterized by a nonlinear model. The identification and control of nonlinear systems is an interesting area and one in which much work remains to be done. While in the past three decades, major advances have been made in adaptive identification and control for identifying and controlling linear time-invariant plants with unknown parameters very few results exist in nonlinear systems theory which can be directly applied. Design procedures that simultaneously meet the requirements of stability, robustness and good dynamical response are currently not available in the case of the nonlinear systems. The reasons for this are the higher degree of complexity of, and lack of a general structure for nonlinear systems.

Artificial neural networks (ANN) show a great promise in the realm of nonlinear control problems. This stems from their theoretical ability to approximate any arbitrary nonlinear mappings. In the same way that transfer functions provide a generic representation for linear black box models, ANN's potentially provide a generic representation for nonlinear black box models. The great diversity of nonlinear systems is the primary reason why no systematic and generally applicable theory for nonlinear control design has yet evolved. A range of traditional methods for the analysis and synthesis of nonlinear controllers for specific classes of nonlinear systems exist: phase plane methods, linearization techniques and describing functions are but a few examples [3].

Artificial neural networks are biologically inspired; that is, they are composed of elements that perform in a manner that is analogous to the most elementary functions of a biological neuron. These elements are then organized in a way that may (or may not) be related to the anatomy of the brain. Despite this superficial resemblance, artificial neural networks exhibit a surprising number of the brain's characteristics. For example, they learn from experience, generalize from previous examples to new ones, and abstract essential characteristics from inputs containing irrelevant data [4].

The neural networks also have the capability of massive parallel processing in contrast to the sequential execution of the conventional digital computers. Neural networks also provide, in principle, significant fault tolerance, since damage to a few links will not impair the overall performance.

The hardware implementation of neural networks is currently a very active research area. Some manufacturers such as Intel have produced a neural net chip which is supposed to have more effective throughput (for the neural net calculations) than even the Cray supercomputer. Another company, Oxford Computer in Oxford, Connecticut, has developed an intelligent memory chip that can be used for neural network implementation.

An artificial neural network is made of single elements called neurons arranged in layers and these elements contribute to making the network learn over a period of time. The neurons arranged in different layers are fully interconnected with different connections having different weights that may be either positive or negative. The inputs to the network are fed into the neurons constituting the first layer, also called the input layer, which send these inputs onto the next layer. The outputs of the neurons in each layer are sent on to the neurons in the layer succeeding it. These are then multiplied by the values of the weights of the interconnections between the two neurons and algebraically summed. The resultant value is then used as an input to an "activation function" which is more often than not a nonlinear function. The output of this activation function serves as the node's output activation level. In this way a neural network can be used to model nonlinear relationships with an arbitrary accuracy. In order to learn, the neural network is subjected to training whereby a set of inputs is presented to it and some function of the error between its output and the desired output is then minimized, according to some minimization criterion, the most common one being the mean square one. This is called supervised training. In contrast, in the case of unsupervised training there is no target vector for the outputs, and hence, no comparisons to predetermined output responses are made. Rather, the network is taught to respond in a specified way when similar patterns are presented at the input.

In order to control nonlinear systems, one has to first identify the dynamic relationship associated with the system and then design a controller based on that model. Also, in many cases, because of various factors such as the effects of environment, wear, tear, age etc., the plant dynamics may change with time and the controller has to be updated accordingly and continuously. In the control literature, this strategy is known as adaptive control.

Given a set of inputs and desired outputs, an ANN can be "trained" to produce consistent responses. Its modifies its behavior in response to its environment. Once trained, a network's response can be, to a certain degree, insensitive to minor variations in its inputs [4].

Neural Networks have been considered for controlling nonlinear plants by various investigators. The design of both fixed as well as adaptive controllers have already been attempted. Most of the reported results are only applicable to inverse stable plants. All of them suffer from long training times of the neural networks. In the present research, we investigate the design of neural net controller through a linearization technique. This strategy allows the use of powerful design methods available in linear control systems. The design procedure allows control of inverse unstable plants too. The design principle is applied to both adaptive as well as fixed controllers. The use of improved training algorithms to speed up the training time of neural networks is also considered and demonstrated.

Organization

A survey of literature in the field of application of Neural Networks in control is given in chapter 2. A brief review of the structure and training of Neural Networks is given in chapter 3. The development of the three-stage design for control of nonlinear plants is presented in chapter 4. Simulation results are presented in chapter 5, followed by chapter 6 which includes contributions, conclusions and prospects for future research.

Chapter 2

LITERATURE SURVEY

Interest in artificial neural networks began in the early 1940s when pioneers, such as McCulloch, Pitts and Hebb, investigated networks based on neurons and attempted to formulate the adaptation laws which applied to such systems. During the 1950s and 1960s several basic architectures were developed and a background body of knowledge was built up from many diverse disciplines: biology, psychology, physiology, mathematics and engineering. General interest in the subject waned after the analysis of the perceptron by Minsky and Papert highlighting the limitations of several of the models. However, several groups did continue their research and by the mid 1980s the work of Hopfield and of Rumelhart gave renewed impetus to the area. Since then, the number of papers published, conferences organized and journals devoted exclusively to neural network research greatly increased and is continuing to do so[5].

Only a few years ago the area of neural networks in control systems was in its infancy of development. There were many hopes for the field, and fewer accomplishments. Over the past two years the field has been developing, but not by surprising leaps and bounds. Rather it has been evolving through steady progress. Certain views and approaches have now emerged to become accepted and popular. The field is also moving away from blind applications of large neural networks to applications to more specific problems [6].

Neural networks have several important characteristics which are of interest to control engineers:

- Modelling. Because of their ability to learn using data records for the particular system of interest, the major problem of developing a realistic system model is obviated.
- Non-linear systems. The networks possess the ability to 'learn' non-linear relationships with limited prior knowledge about the process structure. This is possibly the area in which they show the greatest promise.
- Multivariable systems. Neural networks, by their very nature, have many inputs and many outputs and so can be readily applied to multivariable systems.
- Parallel structure. The structure of neural networks is highly parallel in nature. This is likely to give rise to three benefits: very fast parallel processing, fault tolerance and robustness.

The great promise held out by these unique features is the main reason for the enormous interest which is currently being shown in this field.

An excellent survey of the importance of neural networks from a control systems perspective and also the future areas for research have been given in [3]. The main focus is on the promise of artificial neural networks in the realm of modelling, identification and control of nonlinear systems.

Fukuda and Shibata [7] have given a general overview of the theory and applications of NN for Industrial Control Systems. They have emphasized different Intelligent Control Systems like the Neuromorphic Control. A more practical approach towards the application of NN's in Industrial Control has been given in [8]. Tracking control of industrial drive systems has been carried out using NN's. Simulation results show that it is possible to achieve real time tracking of any arbitrarily prescribed trajectory with high degree of accuracy.

Kuperstein and Rubinstein [9] use visual information and learning to control a robot arm so that it grasps objects in space. An adaptive neural controller is developed and it is used to control a multijoint arm to reach objects. The controller can learn, unsupervised, to accurately grasp an elongated object arbitrarily positioned in space.

The classical control problem of the inverted pendulum is revisited in [10]. The novelty here is that, in addition to having no *a priori* knowledge of the dynamics of the cart and the pendulum, performance feedback is assumed to be unavailable at each step; it appears only as a positive signal when the pendulum falls or reaches the bounds of the track. Learning methods using reinforcement and temporal differences are used to build the necessary knowledge base and balance the pendulum for extended time periods.

Neural Networks also hold promise in some other branches of control like, IMC (Internal Model Control) [11] and Optimal and Predictive Control [12],[13]. In IMC, the trained NN forward model of the system is placed in parallel with the actual system. The difference between the system and model outputs is fed back which is processed by the NN controller trained as the inverse of the system. The implementation of IMC is limited to open-loop stable systems. In the realm of Optimal & Predictive control methods the receding horizon technique has been introduced as a natural, computationally feasible law. It has been proven that the method has desirable stability properties for nonlinear systems [12]. The application of NN as a potential aid in the development of a nonlinear predictive controller as well as an inferential estimator has been discussed in [13]. They have demonstrated that the neural estimator could provide a fast inference of a difficult-to-measure process output, from other easily measured variables. By using the estimates provided by the NN as feedback signals, good regulation is shown to be achieved.

Various researchers have used NN's for identification of inverse dynamics of non-linear systems. In [14], both generalized or direct inverse modelling as well as specialized inverse modelling are considered. In specialized inverse modelling, the input to the NN is the set point, while the error between the set point and the actual system output is propagated back through a trained forward model of the actual system. In [15], numerous examples have been given on the use of the inverse model for controlling systems. The inverse model is simply cascaded with the controlled system in order that the composed system results in an identity mapping between desired response and the control system output. Thus the network acts directly as the controller in such a configuration.

The Hopfield network may be utilized as a dynamical controller for a linear system [16]. Elements of variable structure control theory have been utilized to construct the controller and, as a result, the proposed controller is characterized by its robustness. Adaptation has been included in [17] and [18]. In this case, the network is included in the adaptation path and is used to minimize simultaneously the square error rates of all the states. Here, the output of the network represents the parameters of a linear model of the plant.

Neural Networks are also being used to adaptively control nonlinear plants. By learning the inverse dynamics of the system on-line, the NN is used to adaptively control the nonlinear plant in [19]. This method is based on the weak assumption that the states of the system are measurable. In contrast to this approach, a NN is trained as a forward model in [20]. This NN updates the controller to generate the desired control input. Simulation results show that it takes quite a long time for the system to follow the set point. In [21], a new neuron structure, called the Dynamic Neural Unit (DNU), has been proposed for adaptive control of nonlinear systems. However this approach suffers from its extremely complex structure.

The control of nonlinear plants have been considered in [22] and [2]. In [2], NN identification and control of nonlinear plants are discussed in detail. The dynamic back propagation technique has been introduced for training neural networks. Various methods of utilizing multilayer and recurrent networks for identification and control have also been given therein as well as the use of neural networks as explicit self-tuning controllers.

In [23], fast convergence algorithms have been proposed. These are ad hoc methods in which the weights of the plant emulator as well as those of the neural controller are updated more than once in each sampling period. Simulation results of one example show faster convergence of the proposed algorithms.

An attempt has been made at utilizing neural networks as implicit self tuning controllers in [24]. While this approach can also be used to control non-linear systems, only examples of linear systems were provided.

Feedback-error-learning of neural networks has been introduced in [25], [26] and [27]. They used the neural network as an add-on component to the conventional linear controller. Further, their approach trains the neural controller to assume the inverse dynamics of the plant. Therefore, the strategy cannot be used with inverse unstable plant.

In [28], ANN's haven been utilized in process engineering environment. Neural network models were used to provide estimates of biomass concentration in industrial fermentation systems and of top product composition of an industrial distillation tower. In particular, NN's were used for nonlinear predictive controller.

Recurrent neural networks have been used for direct adaptive control of nonlinear and non-minimum phase systems based on model predictive concepts [29]. Both off-line and on-line procedures have been developed. The strategy has been particularly applied for the water level control of a U-tube steam generator (UTSG).

Chen and Liu [30] have used multilayer artificial neural networks for adaptive control of nonlinear plants assuming them to be feedback-linearizable and in continuous-time system. The plant has been decomposed into two parts, one being controlled by NN's while the other is unobservable and is assumed to be stable. The control law is defined in terms of the neural network model to control the plant to track a reference command.

In [31], multilayer artificial neural networks have been used to construct the nonlinear learning control law for a class of unknown SISO nonlinear control systems. Weight training algorithm with a dead zone function has been discussed. Theorems have been provided to prove the convergence of the weight update law and stability of closed loop system is shown.

Chapter 3

NEURAL NETWORKS

This chapter is intended to provide a general introduction to neural networks. Some concepts of neural networks are introduced, with a background to biological aspects, and their attributes are described. Conventional and improved backpropagation algorithms for training neural networks are also presented.

3.1 Introduction

Artificial neural networks are a representation that attempts to mimic (albeit in an extremely simplistic manner) the functionality of the brain. For several decades, scientists have been trying to emulate the real neural structure of the brain, believing that the human process of learning might be reproduced by an algorithmic equivalent. Initially the principal motivation behind this research was the desire to achieve the sophisticated level of information processing that can be achieved by the brain. However, it is apparent that present research aims are not directed at emulating the sheer complexity of the brain. Generally, the methodology is used on a more modest scale to develop nonlinear models [13].

3.2 The Biological Neuron

A human brain contains over one hundred billion computing elements called neurons. Exceeding the stars in our Milky Way galaxy in number, these neurons communicate throughout the body by way of nerve fibers that make perhaps one hundred trillion connections called synapses. This network of neurons is responsible for all of the phenomena that we call thought, emotion and cognition, as well as for performing myriad of sensorimotor and autonomic functions. The exact manner in which this is accomplished is little understood, but much of the physiological structure has been mapped, and certain functional areas are gradually yielding to continued research.

The neuron is the fundamental building block of the nervous system. It is a cell similar to all cells in the body; however, certain critical specializations allow it to perform all of the computational and communication functions within the brain.

As shown in Figure 3.1, the neuron consists of three sections: the cell body, the dendrites and the axon, each with separate but complementary functions. Functionally, the dendrites receive the signals from other cells at connection points called synapses. From there, the signals are passed on to the cell body where they are essentially averaged with other such signals. If the average over a short time interval is sufficiently large, the cell "fires," producing a pulse down its axon that is passed on to succeeding cells. Despite its apparent simplicity, this computational function accounts for most of the known activity of the brain. Underlying it, however, is a complex electrochemical system.

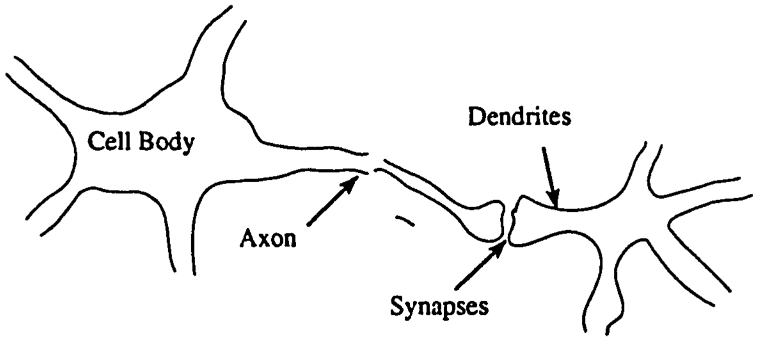


Figure 3.1: The biological neuron

3.3 Artificial Neuron

Artificial neural networks are made up of individual models of the biological neuron (artificial neurons or nodes) that are connected together to form a network. The neuron models that are used are typically much simplified versions of the actions of a real neuron. Information is stored in the network often in the form of different connection strengths, or weights, associated with the synapses in the artificial neuron models.

Conventional programming techniques are significantly better than humans at performing tasks requiring a high degree of numerical computation and repeatable steps that can be accurately pre-specified. However, humans still far exceed the performance of these methods in applications that are poorly defined, either because the problem is extremely complex or simply that exact solution rules are not known (e.g. speech and image recognition, plant monitoring and fault diagnosis). The above attributes of neural networks indicate their potential in solving these problems. Hence, the considerable interest in neural networks that has occurred in recent years is not only due to significant advances in computer processing power that has

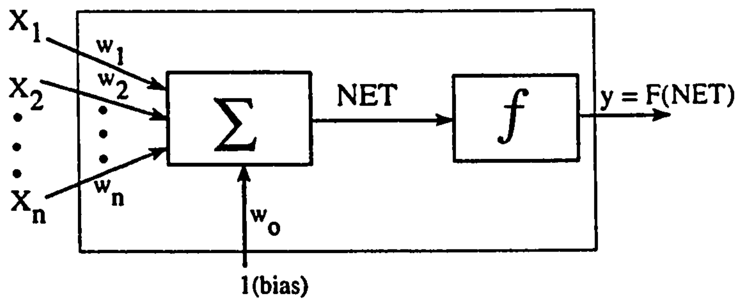


Figure 3.2: Artificial neural network

enabled their implementation, but also due to the diversity in their application areas.

The most commonly-used neuron model is depicted in Figure 3.2. Each neuron input $x_1, \ldots, x_n$, is weighted by the values $w_1, \ldots, w_n$. A bias, or offset, in the node is characterized by an additional constant input of 1 weighted by the value w_o. The output, y, is obtained by summing the weighted inputs to the neuron and passing the result through a non-linear activation function, $f(.)$:

$$\begin{aligned} NET &= \sum_{i=1}^{N} w_i x_i + w_o \\ and \quad y &= f(NET) = f\left(\sum_{i=1}^{N} w_i x_i + w_o\right) \end{aligned} \tag{3.1}$$

3.3.1 Activation Functions

Various types of nonlinear activation functions are possible. One of these may be a simple threshold function,

$$y = K(NET) \tag{3.2}$$

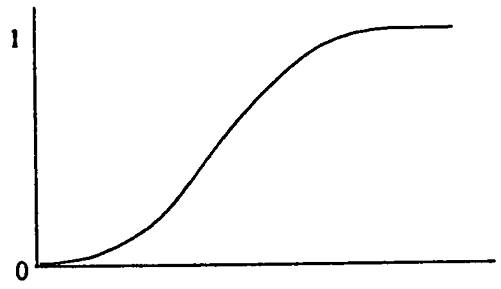


Figure 3.3: Sigmoidal logistic function

where K is a constant threshold function, i.e.,

$$y = 1 \; if \; NET > T$$

$$y = 0 \; otherwise$$

where T is a constant threshold value, or a function that more accurately simulates the nonlinear transfer characteristics of the biological neuron and permits more general network functions. If the activation block $f(.)$ compresses the range of NET, so that y never exceeds some low limits regardless of the value of NET, f is called a *squashing function.* The squashing function is often chosen to be the logistic function or "sigmoid" (meaning S-shaped) as shown in Figure 3.3. This function is expressed mathematically as,

$$y = \frac{1}{(1 + e^{-NET})} \tag{3.3}$$

Another commonly used activation function is the hyperbolic tangent. It is similar in shape to the logistic function and is often used by biologists as a mathematical model of nerve-cell activation. Used as an artificial neural network activation function, it is expressed by:

$$y = tanh(x)$$

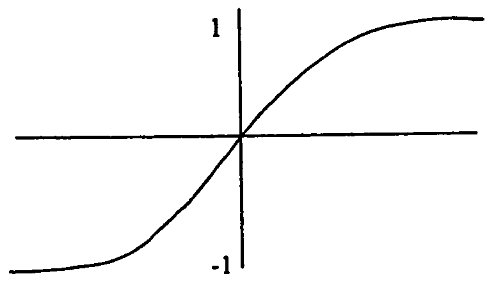


Figure 3.4: Hyperbolic tangent function

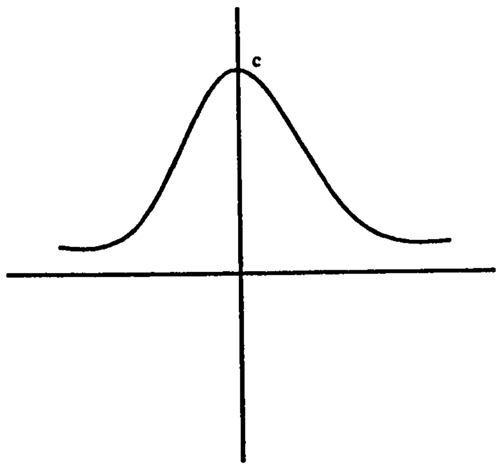


Figure 3.5: Gaussian function

$$or \quad y = \frac{1 - e^{-NET}}{1 + e^{-NET}} \tag{3.4}$$

Like the logistic function, the hyperbolic tangent function is S shaped, but is symmetrical about the origin, resulting in y having the value 0 when NET is 0 (see Figure 3.4) [4]. Another choice of the activation function can be the Gaussian function shown in Figure 3.5. Mathematically, it can be represented as,

$$y = ce^{-NET^2/\sigma^2}$$

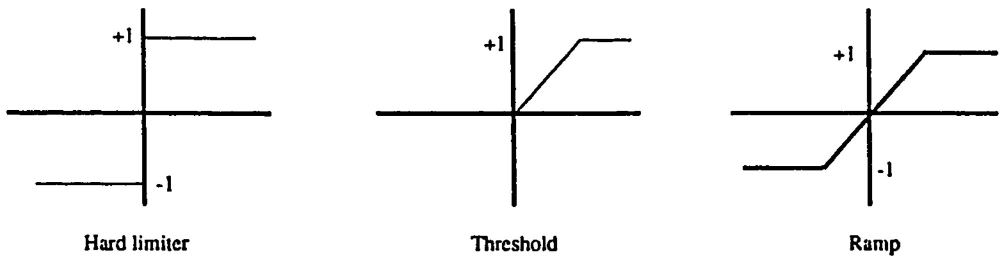


Figure 3.6: Various activation functions

where σ is the standard deviation of the function.

Some other functions which can be used in the activation block are shown in Figure 3.6.

3.3.2 Multilayer Feedforward Neural Networks

The neurons by themselves are not very powerful in terms of computation or representation but their interconnections allow them to encode complex relations between variables, hence endowing them with powerful processing capabilities. The connection of several layers gives the possibility of more complex nonlinear mapping between the inputs and the outputs. The most popular multilayer neural network architecture is shown in Figure 3.7. The network consists of an input layer, a number of hidden layers (typically only one or two hidden layers are used) and an output layer as shown in Figure 3.7. The output and hidden layers are made up of a number of nodes as described in section 3.3. However the input layer is essentially a direct link to the inputs of the first hidden layer and is included by convention. Tangent hyperbolic activation functions for the nodes in the hidden and output layers are the most common choice for control applications, although variants on this are also

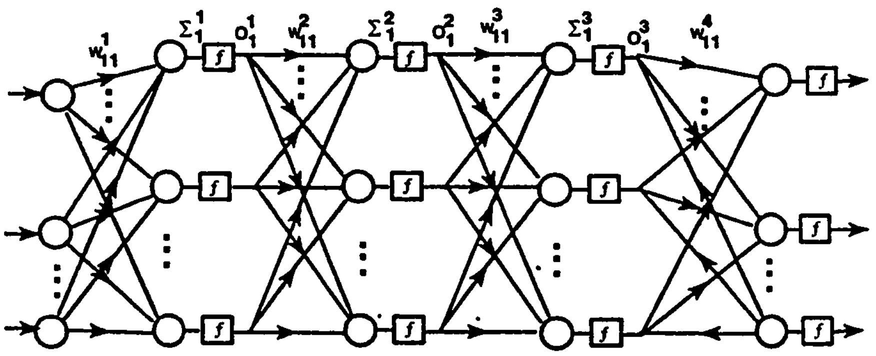


Figure 3.7: Multilayer neural network

possible. The outputs of each node in a layer are connected to the inputs of all of the nodes in the subsequent layer. Data flows through the network in the forward direction only, hence, making this type of network a feedforward one [5].

3.4 NN Training Using Backpropagation

The introduction of the backpropagation algorithm played a large part in the resurgence of interest in artificial neural networks. Backpropagation is a systematic method for training multilayer artificial neural networks. It has a mathematical foundation that is strong if not highly practical. Despite its limitations, backpropagation has dramatically expanded the range of problems to which artificial neural networks can be applied, and it has generated many successful demonstrations of its power.

The so-called backpropagation is a first order gradient scheme to minimize the error between the actual and target output. Let y_k and $y_{d,k}$ be the actual and desired outputs of the network respectively and e_k, the error, be the difference between them.

Backpropagation gives a systematic procedure to compute the gradient $\partial J/\partial w$ where,

$$J = \frac{1}{2}\|e_k\|^2 \tag{3.5}$$

$$= \frac{1}{2}\|y_{d,k} - y_k\|^2 \tag{3.6}$$

The aim of the algorithm is to minimize this cost function J. The well known delta rule for updating the weights is a first-order gradient descent algorithm moving in the negative gradient direction. Using the delta rule we have,

$$w_{k+1} = w_k - \eta \left[\frac{\partial^+ J}{\partial w}|_{w=w_k}\right]^T \tag{3.7}$$

where η, known as the "learning rate", is a small positive quantity chosen arbitrarily and $\partial^+ J/\partial w$ denotes the ordered derivative [32].

Using (3.6) (also see Figure 3.2),

$$\begin{aligned}\frac{\partial^+ J}{\partial w} &= -\frac{\partial^+ y_k}{\partial w}e_k \\ &= -\frac{\partial^+ y_k}{\partial NET}\frac{\partial NET}{\partial w}e_k \end{aligned} \tag{3.8}$$

Using the hyperbolic tangent given by (3.4) as the activation function, it can be sown that its derivative is simply given by

$$\frac{\partial^+ y_k}{\partial NET} = \frac{1}{2}(1 + y_k)(1 - y_k) \tag{3.9}$$

Also, from (3.1) we have,

$$\frac{\partial^+ NET}{\partial w} = x \tag{3.10}$$

Substituting (3.8), (3.9) and (3.10) in (3.7) we get,

$$w_{k+1} = w_k - \eta\frac{1}{2}(1 + y_k)(1 - y_k).x.e_k \tag{3.11}$$

Backpropagation algorithm can be applied to networks with any number of layers, equation (3.11) can be extended to update weights in other layers too[4]. But no systematic way is available for the choice of the learning rate "η". With a conservative choice of the learning rate, convergence to a local minima is guaranteed but at the cost of a convergence time that is usually painfully long.

3.4.1 The Training Procedure

The objective of training the network is to adjust the weights so that application of a set of inputs produces the desired set of outputs. For reasons of brevity, these input-output sets can be referred to as vectors. Training assumes that each input vector is paired with a target vector representing the desired output: together these two vectors are called a training pair. Usually, a network is trained over a number of training pairs.

Before starting the training process, all of the weights must be initialized to small random numbers. This ensures that the network is not saturated by large values of the weights, and prevents certain other training pathologies. For example, if all the weights start at equal values and the desired performance requires unequal values, the network will not learn.

The network is trained in a supervised fashion. This means that during training, both the network inputs and required (or target) outputs are used. A set of input and corresponding output data to be learned by the network, is collected. The training process consists of two steps: In the first step, an input pattern is applied to the network and an output is generated. This output is compared to the corresponding target output and an error is produced.

In the second step, the error is propagated back through the network, from output to input, and the network weights are adjusted in such a way as to minimize a cost function, typically the sum of the squared errors.

The procedure is repeated through all the data in the training set and numerous passes of the complete training data set are usually necessary before the cost function is reduced to a sufficient value. Backpropagation gives a systematic procedure to compute the gradient $\partial J/\partial w$. This is achieved by following a two-step procedure described below.

Step 1: (Forward Pass)

An input vector x is applied and an output vector y is produced, note that a 1 is included in x to represent the bias term. The input-target vector pair x and $y_{d,k}$ comes from the training set. The calculation is performed on x to produce the output vector y.

The weights between neurons can be considered to be a matrix w. The NET output vector N for a particular layer may be expressed as the inner product of x and w. In vector notation $\mathbf{N} = x^T w$. Applying the function F to the NET vector N, component by component, produces the output vector y. Thus, for a given layer, the following expression describes the calculation process:

$$y = F(x^T w) \tag{3.12}$$

The output vector of one layer is the input vector for the next, so calculating the outputs of the final layer requires the application of (3.12) to each layer, from the network's input to its output.

Step 2: (Reverse Pass)

Adjusting the weights of the output layer: In order to adjust the weights of the output layer, the error between the desired and actual outputs is multiplied by the derivative of the squashing function. If the hyperbolic tangent function is used as the activation function, $f'(x)$ is given by,

$$f'(x) = \frac{1}{2}(1 - f(x))(1 + f(x)) = \frac{1}{2}\left(1 - f^2(x)\right) \tag{3.13}$$

In this way one gets δ for the kth neuron in the ith layer as shown below:

$$\begin{aligned} \delta_i^k &= \frac{1}{2}(1 - y_k)(1 + y_k)e_k \\ \textit{where} \quad \delta_i^k &= \frac{\partial J}{\partial NET(i)} \end{aligned} \tag{3.14}$$

This δ is then multiplied by the output from the preceding jth layer to obtain,

$$\frac{\partial J}{\partial w_{pq}^k} = \delta_q^k y_p^j$$

The weight adjustment is then obtained as,

$$\Delta w_{pq}^k = \eta \frac{\partial J}{\partial w_{pq}^k} = \eta \delta_q^k y_p^j \tag{3.15}$$

Hence, the new value of the weight is given as,

$$w_{pq}^k(n+1) = w_{pq}^k(n) + \Delta w_{pq}^k \tag{3.16}$$

where,

$w_{pq}^k(n)$ = the value of a weight from neuron p in the kth hidden layer to neuron q in the output layer at step n (before adjustment)

$w_{pq}^k(n+1)$ = value of the weight at step $(n+1)$ (after adjustment)

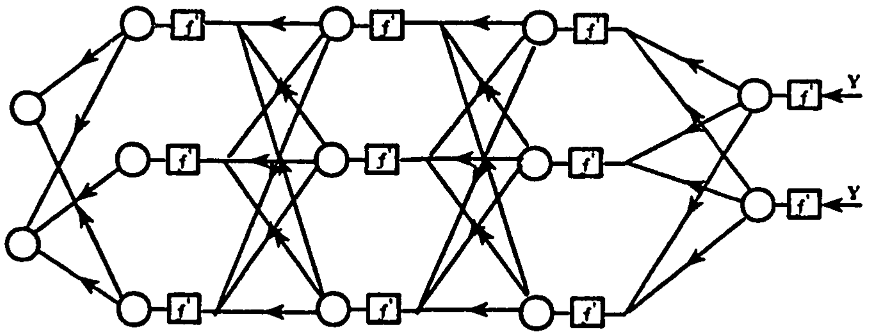


Figure 3.8: Backpropagation of error in a multilayer neural network

δ_q^k = the value of δ for neuron q in the output layer k

y_p^j = the value of y for neuron p in the hidden layer j.

Adjusting the weights of the hidden layers: To adjust the weights of hidden layers, the error from the output layer is propagated back to the hidden layer to obtain the appropriate partial derivatives via the chain rule. For each neuron this value is multiplied by the weights connecting the output layer to the hidden layer. Summation of all such quantities gives the desired δ. Mathematically,

$$\delta_p^j = \frac{1}{2}(1 - y_p^j)(1 + y_p^j)(\Sigma \delta_q^k w_{pq}^k) \tag{3.17}$$

Now the weights in the hidden layer can be adjusted by using (3.16) and (3.17).

3.5 Choice of Learning Rates

Training may be classified as *pattern learning* or *batch learning*. In pattern learning, the weights of the network are adjusted after the arrival of each pattern, whereas in batch learning, the weights are adjusted after a complete sweep of the entire training data. While pattern learning has the advantage of on-line implementation (hence applicable to adaptive control design), the batch learning has a better mathematical validity in the sense that it exactly implements the gradient descent method [33]. The pattern learning can be considered as an approximation of the batch gradient descent learning [34].

In this section, several algorithms for the computation of the learning rates which have been used in this research are presented . These algorithms have been developed by Ahmed ([35], [34] and [36]).

3.5.1 Pattern Learning

Noise-Free Case (Algorithm 1)

The value of the weights in w that minimizes the sum of squared error for all the training patterns, cannot be found analytically. Instead a numerical optimization procedure must be performed. In order to derive a simple gain sequence for the delta rule, some approximations need to be considered. We assume that there exists at least one value of $w = w^o$ such that

$$y(w^o, x_i) = y_d(x_i) \quad for\ i = 1, \ldots q. \tag{3.18}$$

where q is the total number of training patterns. This assumption is of course valid if the network is chosen to be large enough for the problem at hand.

Now the error in each iteration can be written as

$$e_k = y_d(x_i) - y(w_k, x_i) \tag{3.19}$$

We further assume that w_k lies in the vicinity of w^o, such that $y(w^o, x_i)$ can be approximated by a first order approximation about w_k as

$$y(w^o, x_i) = y(w_k, x_i) + \Gamma^T \tilde{w}_k \tag{3.20}$$

where

$$\tilde{w}_k \equiv w^o - w_k$$

$$\Gamma_k = -\left[\frac{\partial e}{\partial w}|_{w=w_k}\right]^T = \left[\frac{\partial y}{\partial w}|_{w=w_k}\right]^T$$

Now combining (3.18),(3.19) and (3.20), one gets,

$$e_k = \Gamma_k^T \tilde{w}_k \tag{3.21}$$

It may be emphasized here that many practically useful optimization algorithms such as Newton-Raphson and Gauss-Newton stem from lower order polynomial approximations of a nonlinear function under the same assumption.

Putting (3.14) in (3.15) and using the definition of Γ ,(3.21) and (3.16), one gets,

$$w_{k+1} = w_k + \eta_k \Gamma_k \Gamma_k^T \tilde{w}_k \tag{3.22}$$

Subtracting both sides of the above from w^o and premultiplying both sides by their corresponding transposes, one gets

$$\|\tilde{w}_{k+1}\|^2 = (1 - 2\eta_k \alpha + \eta_k^2 \alpha\beta)\|\tilde{w}_k\|^2 \tag{3.23}$$

where

$$0 \leq \alpha \leq \lambda_{max}[\Gamma_k \Gamma_k^T] \leq \gamma$$

$$0 \leq \beta \leq \lambda_{max}[\Gamma_k^T \Gamma_k] \leq \gamma$$

$$\gamma \equiv tr[\Gamma_k \Gamma_k^T] = tr[\Gamma_k^T \Gamma_k] = \|\Gamma_k\|^2$$

$\lambda_{max}[.]$, $\|.\|$ and tr[.] respectively denote the maximum eigenvalue, trace and the F-norm.

Now it may be verified from (3.22) that the following expression for η_k

$$\eta_k = \frac{\mu}{\gamma} \quad 0 \leq \mu \leq 2 \tag{3.24}$$

ensures that

$$\|\tilde{w}_{k+1}\|^2 \leq \|\tilde{w}_k\|^2$$

Substituting (3.23) in (3.22), one also gets

$$\|\tilde{w}_{k+1}\|^2 \leq \left[1 - (2-\mu)\mu\frac{\alpha}{\gamma}\right] \|\tilde{w}_k\|^2 \tag{3.25}$$

Further, differentiating the right and side of (3.24) with respect to μ and setting it to zero one gets the optimum value of μ as 1. However, in the initial stages when the linearization approximation will be poor, a more conservative expression for μ should be used which is,

$$\mu(\tau+1) = 1 - \mu_r[1 - \mu(\tau)]; \qquad \mu(0) \leq 1 \tag{3.26}$$

where $\mu(\tau)$ indicates the value of μ at the τth iteration and μ controls the rate of transition of $\mu(\tau)$ from $\mu(0)$ to 1. The argument τ may represent a pattern-iteration or a sweep-iteration. μ_r normally ranges between 0.9 and 1.

A further problem may arise when Γ_k approaches zero causing γ to be a very small number. Numerical problem may arise in the computation of η_k in (3.23) due to division by γ. To alleviate such problem, (3.23) requires a modification as

$$\eta_k = \frac{\mu}{(\epsilon + \gamma)} \qquad 0 \leq \mu \leq 2 \tag{3.27}$$

where ϵ is a small quantity ($\approx$0.001). Hence, the weights will be updated using,

$$w_{k+1} = w_k + \eta_k \Gamma_k e_k \tag{3.28}$$

Noisy Case (Algorithm 2)

The derivation in the previous section assumed that the training data is free from measurement noise. This led to equation (3.19) where the error vanishes as w_k

approaches the target value w^o. In the presence of noise $e(w^o, t)$ would approach $e^*(t)$ which generally will be different than zero.

An expression for pattern learning rate can also be derived for this case based on first order approximation of $\hat{y}(w_{k-1}, t)$. The steps involved consist of (a) the derivation of a scalar batch learning algorithm, where the terms including $e^*(t)$ are eliminated using the optimality condition and (b) the conversion of the batch learning algorithm into a pattern learning algorithm [36]. The resulting expression is obtained as

$$\eta_p^{-1}(t) = \lambda \eta_p^{-1}(t-1) + \|\Gamma_p(t)\|^2 \tag{3.29}$$

where η_p is the pattern learning rate, Γ_p is $-\partial e/\partial w$ and λ is a forgetting factor, typical value being 0.9~0.99. $\eta_p(0)$ is chosen as a small positive quantity (≈ 0.1).

3.5.2 Batch Learning

Scalar Learning Rate (Algorithm 3)

An expression for the scalar learning rate in batch update can be obtained following a similar approach to the one above. Assuming that w^k is in the vicinity of w^o, error linearization in the weight space gives the following equation,

$$e(t) = \Gamma_b^T(t)\tilde{w}_k \tag{3.30}$$

where,

$$\tilde{w}_{k+1} \equiv w_{k+1} - w^o$$

In the case of batch learning, the weights are updated as,

$$w_{k+1} = w_k - \eta_b \sum_{t=1}^{q} \left[\frac{de(t)}{dw_k}\right]^T = w_k + \eta_b \sum_{t=1}^{q} \Gamma_b(t) e(t) \tag{3.31}$$

where q is the number of patterns in a batch. Substituting (3.29) in (3.30), one gets

$$w_{k+1} = w_k + \eta_b \sum_{t=1}^{q} \Gamma_b(t)\Gamma_b^T(t)\tilde{w}_k \tag{3.32}$$

Subtracting both sides of the above from w^o and premultiplying both sides by their corresponding transposes, one gets

$$\|\tilde{w}_{k+1}\|^2 = (1 - 2\eta_b\alpha + \eta_b^2\alpha\beta)\|\tilde{w}_k\|^2 \tag{3.33}$$

where

$$0 \leq \lambda_{min}[H] \leq \alpha, \quad \beta \leq \lambda_{max}[H] \leq \gamma_b$$
$$H \equiv \sum_{t=1}^{q} \Gamma_b(t)\Gamma_b^T(t) \quad and \quad \gamma_b \equiv tr[H] = \sum_{t=1}^{q} \|\Gamma_b(t)\|^2 \tag{3.34}$$

The proof of (3.33) can be found in appendix B.

Following the derivation of (3.26) an expression for η_b may be obtained as

$$\eta_b = \frac{\mu}{(\epsilon + \gamma_b)} \qquad 0 \leq \mu \leq 2 \tag{3.35}$$

A comparison of (3.23) and (3.33) shows that the range of α and β in the latter is much wider. Since both learning rates are derived based on inequalities, the batch learning rate is relatively conservative. Another degree of conservativeness in (3.33) comes from the fact that γ_b is a conservative upper bound of λ_{max}[H] (i.e. generally λ_{max}[H] would never attain γ_b). It is therefore expected that the application of (3.35) for η_b will yield relatively slower convergence. A value of μ higher than 1 may be needed to obtain faster training.

Matrix Learning Rate (Algorithm 4)

It follows from the above discussion that a scalar learning rate for the batch training will be conservative causing slow convergence. The convergence however can be dramatically improved if a matrix learning rate is used. Assuming the learning rate η_b to be a matrix, subtracting both sides of (3.30) from w^o and premultiplying both sides by their corresponding transposes, one may get

$$\|\tilde{w}_{k+1}\|^2 = \left[I - 2\eta_b H + \eta_b H H \eta_b^T\right] \|\tilde{w}_k\|^2 \tag{3.36}$$

where H is defined following (3.33). Now it can be observed from (3.34) that if H is invertible, the following choice of the matrix learning rate

$$\eta_b = \mu H^{-1}; \quad 0 \leq \mu \leq 2 \tag{3.37}$$

ensures that

$$\|\tilde{w}_{k+1}\|^2 \leq \|\tilde{w}_k\|^2 \tag{3.38}$$

To handle cases of singular or near-singular H, heuristics may be incorporated giving the final expression of the matrix learning rate for the batch training as

$$\eta_b = \mu(\epsilon I + H)^{-1}; \quad 0 \leq \mu \leq 2 \tag{3.39}$$

Following the approach of (3.25), the optimum value of μ ensuring (3.36) can be shown to be 1.

The computation of the matrix learning rate involves solution of a set of linear equation. In a large network, the dimension of H may become very large making the matrix inversion (or solution of the linear system) prohibitive. Thus it may not be possible to use the above matrix learning rate in all the cases. The dimension of H is n_w x n_w where n_w is the total number of weights in the network. If the available computational facilities do not allow inversion of this matrix, one possibility is to use the scalar learning rate and accept a slower convergence as pointed out earlier.

Nevertheless, if a matrix learning rate is computable, extremely fast and accurate training of the network is possible as has been observed in this research.

Since these batch algorithms have been derived based on a minimization of the squared error ([34] and [36]), they both can be used in the presence or absence of noise, without any modifications.

Chapter 4

NEURAL NET BASED CONTROLLER DESIGN

This chapter describes the use of neural networks for the control of nonlinear plants. A three-stage design procedure for the control of nonlinear plants using neural networks is presented.

4.1 Neural Net Modelling of Dynamic Systems

In recent years a number of authors have addressed issues such as controllability, observability, feedback stabilization and observer design for nonlinear systems. In spite of such attempts, constructive procedures, similar to those available for linear systems, do not exist for nonlinear systems. Hence, the choice of identification and controller models for nonlinear plants is a formidable problem and successful identification and control has to depend upon several strong assumptions regarding the input-output behavior of the plant [2].

The ability of neural networks to approximate large classes of nonlinear functions sufficiently accurately made them prime candidates for representing dynamic nonlinear plants.

Nonlinear plants can be described using several representations. We present four models of discrete-time plants described by the following nonlinear difference equations [2].

Model I:

$$y_p(k+1) = \sum_{i=0}^{n-1} \alpha_i y_p(k-i) + g\left[u(k), u(k-1), \cdots, u(k-m+1)\right] \tag{4.1}$$

Model II:

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \cdots, y_p(k-n+1)\right] + \sum_{i=0}^{m-1} \beta_i u(k-i) \tag{4.2}$$

Model III:

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \cdots, y_p(k-n+1)\right] + g\left[u(k), u(k-1), \cdots, u(k-m+1)\right] \tag{4.3}$$

Model IV:

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \cdots, y_p(k-n+1); u(k), u(k-1), \cdots, u(k-m+1)\right] \tag{4.4}$$

where $[u(k), y_p(k)]$ represents the input-output pair of the SISO plant at time k, and $m \leq n$. The functions $f : \mathrm{R}^n \rightarrow \mathrm{R}$ in models II and III and $f : \mathrm{R}^{n+m} \rightarrow \mathrm{R}$ in models I and IV are assumed to be differentiable function of their arguments. In all of these four models, the output of the plant at the time $k+1$ depends both on its past n values $y_p(k-i)(i = 0, 1, \cdots, n-1)$ as well as the past m values of the input $u(k-j)(j = 0, 1, \cdots, m-1)$. The dependence on the past output values $y_p(k-1)$ is linear in Model I while in Model II the dependence on the past input values $u(k-j)$

is assumed to be linear. In Model III, the nonlinear dependence of $y_p(k+1)$ on $y_p(k-i)(i = 0, 1, \cdots, n-1)$ and $u(k-j)(j = 0, 1, \cdots, m-1)$ is assumed to be separable. It is evident that Model IV in which $y_p(k+1)$ is a nonlinear function of $y_p(k-i)(i = 0, 1, \cdots, n-1)$ and $u(k-j)(j = 0, 1, \cdots, m-1)$ subsumes Models I-III. In this research, we have used Model IV as the representation of our nonlinear plant.

In this section neural net modelling of dynamic nonlinear system is presented. Neural net modelling of dynamic relationships is needed in various stages of the proposed control design. Consider a dynamic nonlinear relationship described by Model IV, i.e., (equation (4.4)

$$y(t) = f\{\phi(t)\} \tag{4.5}$$

where $y(t)$ is the output and $\phi(t)$ is a $(n_\phi \text{ x } 1)$ vector composed of past outputs and inputs and $f(.)$ is a nonlinear function that represents the dynamic input-output relationship. A feedforward neural network may be trained to approximate the function f. Upon completion of training, the nonlinear relationship can then be expressed by the trained neural network.

The feedforward neural network is assumed to have multiple layers including one or more hidden layers. Units in each layer feed the units into the following layer through a set of weights and a nonlinear differentiable activation function. Each activation function may also have a bias term. The $(n_w \text{ x } 1)$ vector w is assumed to contain all the concatenated weights of the network. We further .denote the output of the feedforward neural network for a given weight w and input pattern $\phi(t)$ as

$$NN\{w, \phi(t)\} \tag{4.6}$$

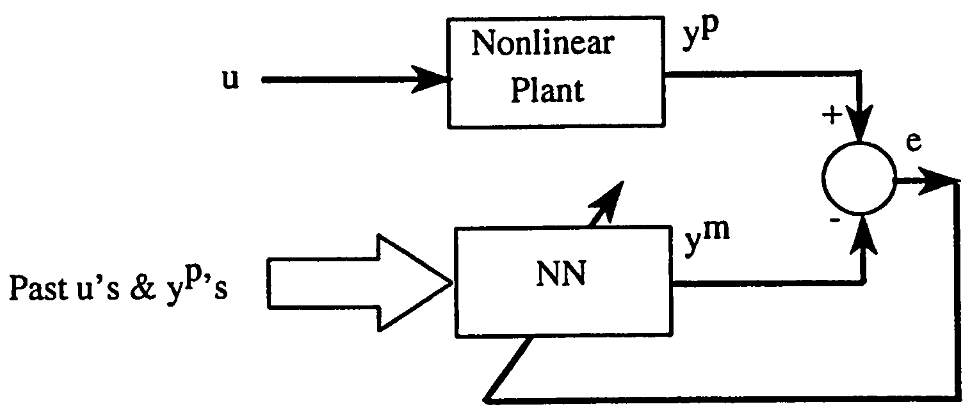


Figure 4.1: Identification using equation error method

The neural net modeling of a dynamic plant requires past values of input and output data as current inputs to the neural network. Based on whether the past outputs are taken from the actual plant or from the neural net model itself, the modelling of the plant is known as series-parallel modelling (equation error method) or parallel modelling (output error method) respectively. Both the configurations are described in Figures 4.1 and 4.2. However, series-parallel model is preferred over parallel model for identification of dynamic systems because of stability problems associated with the parallel model. If a continuous stability check is performed with the parallel model, it may make the algorithm stagnant. Also, parallel modelling is found to be less flexible.

4.2 Neural Net Based Controller

Control design based on linear plant models is well developed. A vast number of robust, reliable, easy to compute design techniques are available. They include

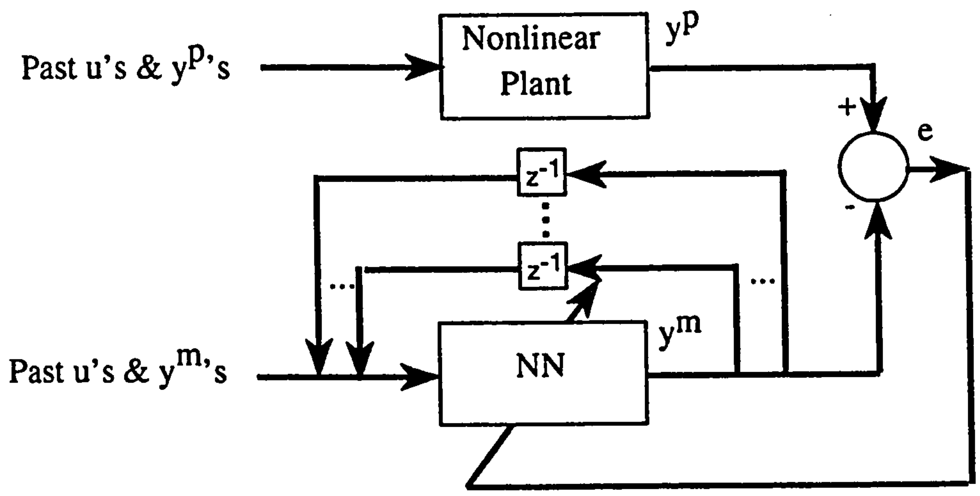


Figure 4.2: Identification using output error method

pole placement, optimal control, pole restriction and the likes. It is natural to make use of these techniques in nonlinear plants under suitable assumptions. In this section we propose a three-stage design technique that is primarily based upon plant linearization at each operating point.

4.2.1 Plant Identification (Stage 1)

The objective of this stage is plant identification, which may be used to obtain the linearized plant model at every operating point. If the plant model is already available from which a linearized model can be computed at each operating point, then this stage may be skipped.

For the identification experiment, it is assumed that the plant order is known and further that the plant is either open loop stable or stabilizable in closed loop. Moreover, the equation error method has been used for identification. Consider the discrete time nonlinear plant described by

$$y(t) = g[y(t-1), \dots, y(t-n), u(t-1), \dots, u(t-n)] \tag{4.7}$$

where n is the known order. Defining the equation error as

$$e(t) = y(t) - \hat{y}(t) \tag{4.8}$$

$$\textit{where} \quad \hat{y}(t) = NN\{w, y(t-1), \dots, y(t-n)|u(t-1), \dots, u(t-n)\} \tag{4.9}$$

a feedforward neural network may be trained to approximate the function g. The training can be performed off line using the series parallel scheme as shown in Figure 4.1. The batch algorithm presented in section 3.5.2 can also be applied for the offline identification.

The iterative training can be stopped either when the total training error reaches a bottom threshold or when training error ceases to decrease any further. Usually a neural network with one hidden layer is sufficient. However, the number of neurons in the hidden layer should be chosen through trial and error. Starting from a small number of neurons, the number can be gradually increased until an accepted training error is achieved.

Once the neural network is trained, it can be used to obtain a linearized model of the plant from the available input and output values. The linearized model is given as

$$\begin{aligned} A(q^{-1})y(t) &= B(q^{-1})u(t) \qquad (4.10) \\ \textit{where} \qquad A(q^{-1}) &= 1 + a_1 q^{-1} +, \dots, + a_n q^{-n} \\ B(q^{-1}) &= b_1 q^{-1} +, \dots, + b_n q^{-n} \end{aligned}$$

$$a_i \equiv -\frac{\partial^+ y(t)}{\partial y(t-i)} \quad , \quad b_i \equiv \frac{\partial^+ y(t)}{\partial u(t-i)}$$

and q^{-1} is a unit delay operator. The ordered partial derivatives [37] can be computed from the neural net model using one forward and a backward pass of the backpropagation. First, $y(t-i), u(t-i),\ i = 1, \ldots, n$ are forward passed. To obtain a particular row of partial derivatives a 1 is back propagated from the output terminal all the way back to the corresponding input terminal and a zero is backpropagated from all other output terminals. The implicit assumption is that $\hat{y}(w^o, t) \approx y(t)$.

4.2.2 Time Varying Controller Design (Stage 2)

The objective of this stage is to generate training data for the ultimate neural controller. However, this stage may also be converted into a completely adaptive scheme. If the nonlinear plant is known to be truly time-varying such adaptive scheme may be preferred instead of designing a fixed neural controller. The control scheme is shown in Figure 4.3. As shown in the figure the input and output of the plant are fed to the feedforward neural net model, which at every time instant provides the controller with a linearized model of the plant. The controller at every instant of time computes a linear control law using any of the standard design techniques and the linearized model. This results in a linear time varying controller. In our simulation study, we considered pole placement, optimal control and pole restriction control strategies. However, other strategies may well be used. The proposed control scheme may be made fully adaptive by incorporating further on line (pattern) training of the forward model by minimizing the error between $y(t)$ and $\hat{y}(t)$ as shown in Figure 4.4. The back propagation algorithm proposed in section 3.5.1 along with the learning rates derived can be used for this purpose.

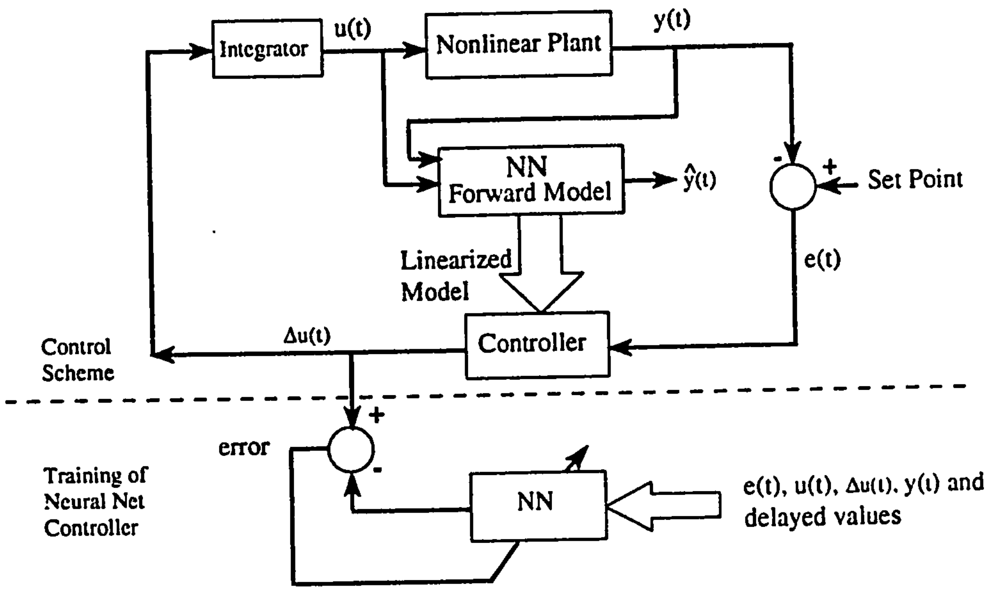


Figure 4.3: Plant control and neural net training using the linearized model

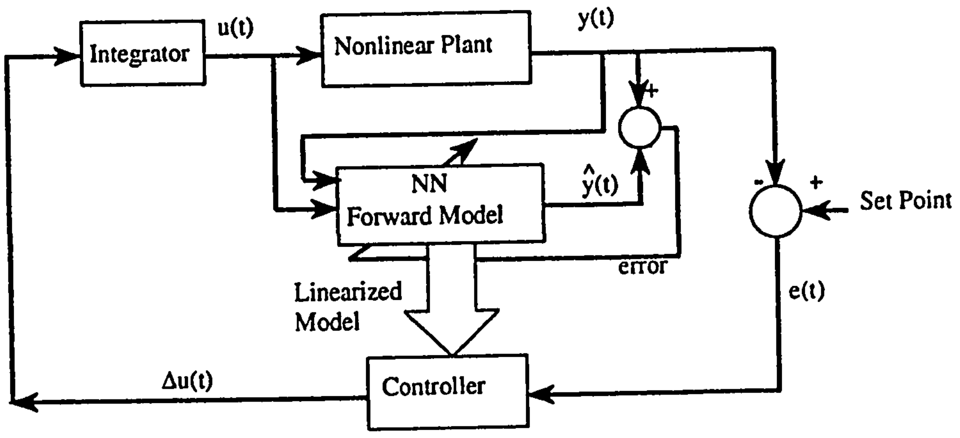


Figure 4.4: Adaptive control of the plant using the linearized model

On the other hand, if a fixed neural controller is desired, the data i.e. $y(t), u(t)$, $\Delta u(t)$ and $e(t)$ generated in this stage can be stored to train a neural controller. If the control scheme of this stage can be implemented in real time and the plant is accessible to implement the scheme, data may be collected from the real plant as shown in Figure 4.3. However, if this is not the case, the plant in Figure 4.3 may be substituted by its model.

Pole Placement Controller Design

In this method the pole positions of the closed loop system are stipulated beforehand. Once the system parameters are estimated, the controller parameters are computed so as to achieve the desired closed loop performance. The positions of the poles are decided upon based on various factors such as for example the desired transient performance and the system limitations.

Let the system be described by,

$$\begin{aligned} G &= \frac{B(q^{-1})}{A(q^{-1})} \qquad (4.11)\\ \textit{where} \quad A(q^{-1}) &= 1 + a_1 q^{-1} +, \ldots, + a_n q^{-n} \\ B(q^{-1}) &= b_1 q^{-1} +, \ldots, + b_n q^{-n} \end{aligned}$$

and a_i's and b_i's are computed using the backpropagation as described in section 4.2.1.

Assume that the linear controller is given by,

$$\begin{aligned} D &= \frac{S(q^{-1})}{R(q^{-1})} \qquad (4.12)\\ \textit{where} \quad S(q^{-1}) &= s_o + s_1 q^{-1} +, \ldots, + s_l q^{-l} \\ R(q^{-1}) &= 1 + r_1 q^{-1} +, \ldots, + r_k q^{-k} \end{aligned}$$

From equation (4.11) and (4.12), the closed loop transfer function can be found as,

$$G_c(q^{-1}) = \frac{B(q^{-1})S(q^{-1})}{A(q^{-1})R(q^{-1}) + B(q^{-1})S(q^{-1})} \tag{4.13}$$

Let the polynomial describing the desired closed loop poles be,

$$T(q^{-1}) = 1 + \alpha_1 q^{-1} +, \ldots, +\alpha_v q^{-v}$$

The controller polynomials for the desired closed loop poles can be obtained by solving the following identity

$$A(q^{-1})R(q^{-1}) + B(q^{-1})S(q^{-1}) = T(q^{-1}) \tag{4.14}$$

which is known as the Diophantine equation. The controller polynomials $R(q^{-1})$ and $S(q^{-1})$ can be obtained from the solution of the following linear system of equations [38]

$$\begin{pmatrix} 1 & 0 & \ldots & 0 & b_o & 0 & \ldots & 0 \\ a_1 & 1 & \ddots & \vdots & b_1 & b_o & \ddots & \vdots \\ a_2 & a_1 & \ddots & 0 & b_2 & b_1 & \ddots & 0 \\ \vdots & \vdots & \ddots & 1 & \vdots & \vdots & \ddots & b_o \\ a_n & \vdots & & a_1 & b_n & \vdots & & b_1 \\ 0 & a_n & & \vdots & 0 & b_n & & \vdots \\ \vdots & \ddots & \ddots & & \vdots & \ddots & \ddots & \\ 0 & \ldots & 0 & a_n & 0 & \ldots & 0 & b_n \end{pmatrix} \begin{pmatrix} r_1 \\ \vdots \\ r_k \\ s_o \\ \vdots \\ s_l \end{pmatrix} = \begin{pmatrix} \alpha_1 - a_1 \\ \vdots \\ \alpha_n - a_n \\ \alpha_{n+1} \\ \vdots \\ \alpha_{k+l+1} \end{pmatrix} \tag{4.15}$$

If an integrator is to be used with the controller, the above system of equations is modified to

$$\begin{pmatrix} 1 & 0 & \dots & 0 & b_1 & 0 & \dots & 0 \\ a_1 & 1 & \ddots & \vdots & b_2 & b_1 & \ddots & \vdots \\ a_2 & a_1 & \ddots & 0 & b_3 & b_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & 1 & \vdots & \vdots & \ddots & b_1 \\ a_n & \vdots & & a_1 & b_n & \vdots & & b_2 \\ 0 & a_n & & \vdots & 0 & b_n & & \vdots \\ \vdots & \ddots & \ddots & & \vdots & \ddots & \ddots & \\ 0 & \dots & 0 & a_n & 0 & \dots & 0 & b_n \\ 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} r_1 \\ \vdots \\ r_k \\ s_o \\ \vdots \\ s_l \end{pmatrix} = \begin{pmatrix} \alpha_1 - a_1 \\ \vdots \\ \alpha_n - a_n \\ \alpha_{n+1} \\ \vdots \\ \alpha_{k+l+1} \\ -1 \end{pmatrix} \tag{4.16}$$

Quadratic Optimal Controller Design

In this technique, the closed loop poles of the plant are determined such that they minimize the following performance index

$$J_c = \sum_{t=1}^{\infty} \left[y^2(t) + \rho u^2(t) \right] \tag{4.17}$$

The solution of which is obtained by solving a closed loop pole placement problem where the closed loop characteristic polynomial $P(z)$ is obtained by solving [39]

$$\rho A(z)A^*(z) + z^{-1}B(z)B^*(z) = \rho P(z)P^*(z) \tag{4.18}$$

where $A^*(z)$ is the reciprocal polynomial of $A(z)$ and ρ is used to penalize the control input. The closed loop characteristic polynomial can be obtained through transformation to state space and solving a steady state Riccati equation. An alternative method is to find a polynomial $P(z)$ through *Spectral Factorization* which will satisfy equation (4.18) directly. A feedback that gives the desired closed loop poles can then be determined by pole placement strategy using the procedure stated above.

Solution through Riccati equation: Consider the following linear discrete-time plant

$$x(k+1) = \overline{A}x(k) + \overline{B}u(k) \tag{4.19}$$

$$y(k) = \overline{C}x(k) \tag{4.20}$$

where $x(k)$, $u(k)$ and $y(k)$ are the state, control input and measured output respectively. $\overline{A}, \overline{B}$ and $\overline{C}$ are the system matrices.

The objective is to find suitable values of the feedback matrices $F(k)$ such hat the control input given by

$$u(k) = -F(k)x(k) \tag{4.21}$$

minimizes the loss function

$$J = \sum_{k=0}^{\infty} \left[x^T(k)Q_1x(k) + u^T(k)Q_2u(k)\right] \tag{4.22}$$

where Q_1 and Q_2 are symmetric and positive definite with, $Q_1 = C^TC$ and $Q_2 = \rho$.

Control design: The state feedback law [40]

$$u(k) = -\left(R+\overline{B}^T P\overline{B}\right)^{-1}\overline{B}^T P\overline{A}x(k) \tag{4.23}$$

$$= -Fx(k) \tag{4.24}$$

ensures that the value of J defined above is a minimum, where P is a positive definite symmetric solution of the algebraic Riccati equation

$$P = \overline{A}^T P\overline{A} + H^TH - \overline{A}^T P\overline{B}\left(R+\overline{B}^T P\overline{B}\right)^{-1}\overline{B}^T P\overline{A} \tag{4.25}$$

where R is an arbitrary positive definite matrix and H is a matrix such that the pair $(\overline{A}, H)$ is observable. The desired characteristic polynomial is obtained as

$$Det(zI - \overline{A} + \overline{B}F)$$

Controller Design Based on Pole Restriction Principle

In this approach, the user defines the allowable region for the closed loop poles instead of their exact location. In the optimal control approach, the control design specification cannot be made directly in terms of the overshoot and settling time. On the other hand, although such specification can be adopted in pole placement control, the input amplitude is often quite large because of stringent pole specification. Using pole restriction, the user has the flexibility of placing the closed loop poles within a specified disc inside the z-plane. For example, in the pole restriction principle, the performance specifications at the output may assumed to be

- damping coefficient of at least 0.5 (which corresponds to at most 16% overshoot) and
- a time constant of at most 5 samples (which corresponds to a 2% settling time of at most 20 samples).

The damping coefficient constraint confines the closed loop poles within the heart-shaped region shown in Figure 4.5, while the time constant confines the closed loop poles within the dotted circle, which together require that all the closed loop poles must lie within the hatched region. The hatched region is approximated by the inner solid circle in the z-plane as shown in Figure 4.5 [40]. The modified design, therefore, would be based upon overshoot and settling time requirements, but the input amplitude will not be too large because of the less stringent requirement on the closed loop pole locations. The design rule consists of finding the characteristic equation of the closed loop system by solving [40]

$$\rho A(z')A^{*}(z') + z^{-1}B(z')B^{*}(z') = \rho P(z)P^{*}(z) \qquad (4.26)$$

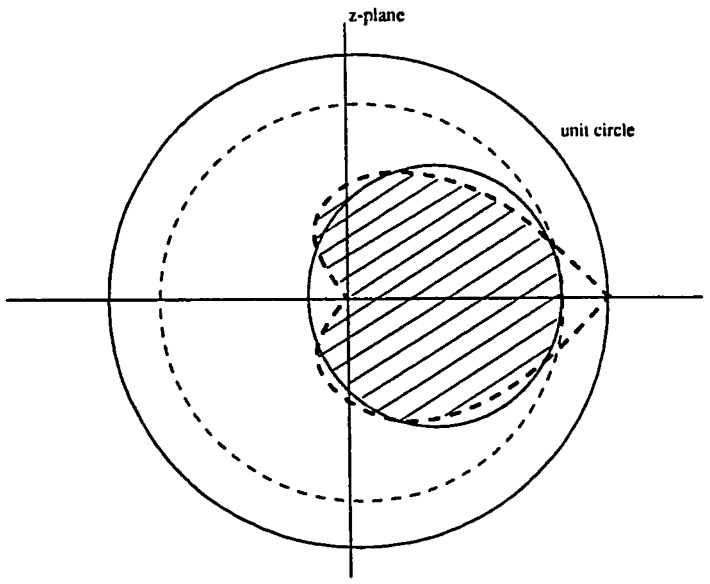


Figure 4.5: Pole restriction region approximated by a circle

where $z' = (z - \alpha)/\beta$ and $\rho > 0$. α and β specify the center and radius of the specified circle that approximates the hatched region (see Figure 4.5. Equation (4.26) can also be solved through transformation to state space and a steady state Riccati equation.

Solution through Riccati equation: Consider the discrete time plant defined by equations (4.19) and (4.20). The objective here is to find suitable values of the feedback matrix $F(k)$ such that the control input given by equation (4.21) confines all the closed loop poles inside a disc in the z-plane with center α and radius β.

Control design: The state feedback law [40]

$$
\begin{aligned}
u(k) &= -\left(\beta^2 R + \overline{B}^T P \overline{B}\right)^{-1} \overline{B}^T P \left(\overline{A} - \alpha I\right) x(k) && (4.27)\\
&= -Fx(k) && (4.28)
\end{aligned}
$$

ensures that all the closed loop poles of the system given by equation (4.19) lie inside a disc in the z-plane having the center at α and radius β, where P is a positive definite symmetric solution of the algebraic Riccati equation

$$P = \frac{(\overline{A}-\alpha I)^T}{\beta} P \frac{(\overline{A}-\alpha I)}{\beta} + H^T H - \frac{(\overline{A}-\alpha I)^T}{\beta} P\overline{B}(\beta^2 R + \overline{B}^T P \overline{B})^{-1} \overline{B}^T P \frac{(\overline{A}-\alpha I)}{\beta} \tag{4.29}$$

where R is an arbitrary positive definite matrix and H is a matrix such that the pair $(\overline{A}, H)$ is observable. The design steps involved can be summarized as follows:

- From the linearized model, get the numerator polynomial $B(z)$ and denominator polynomial $A(z)$ of the plant.
- Transform the system into a state space form.
- Substitute $\overline{B}$ and $\overline{A}$ by $\overline{B}/\beta$ and $\overline{A} - \alpha I$ by choosing appropriate α and β.
- Solve the Riccati equation (4.25) and transform back to the input/output representation to get the desired closed loop polynomial.

4.2.3 Training of the Neural Controller (Stage 3)

In this stage, the data from stage 2 is used to train a specialized-structure neural network to act as the final controller. The trained network is required to replicate the time-varying controller of the previous stage. The inputs to this network include the input-output of the time varying controller i.e. $e(t)$ and $\Delta u(t)$. However, the neural controller must also know the operating point in order to figure out the appropriate control law. The information about the operating point is buried in the plant input-output. Therefore the neural controller must receive $y(t)$ and $u(t)$ as inputs too. The neural controller may be expressed as

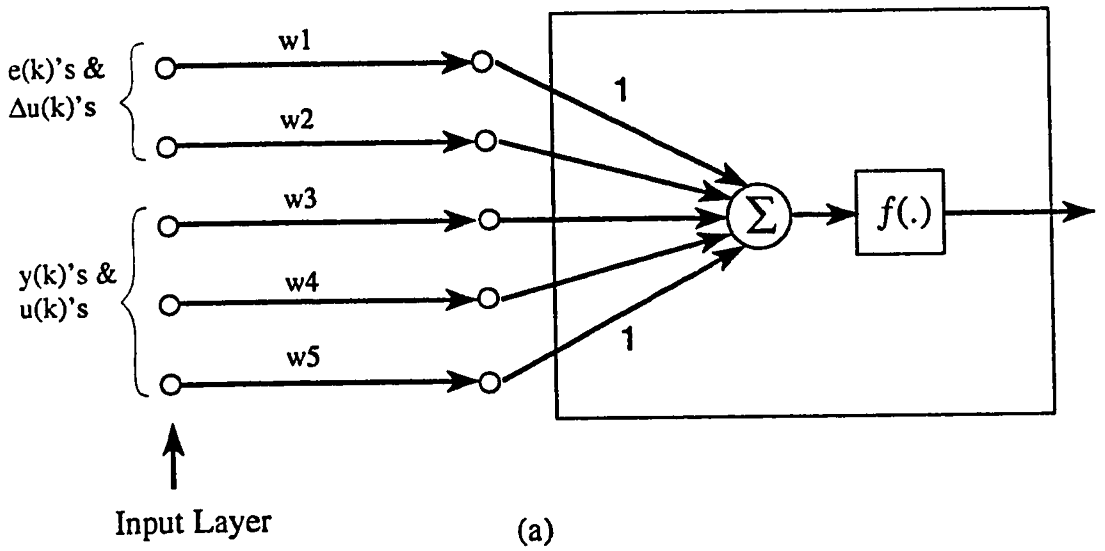
$$\Delta u(t) = h\{e(t),\ldots,e(t-m)|\Delta u(t-1),\ldots,\Delta u(t-m)|y(t),\ldots,y(t-n)|u(t-1),\ldots,u(t-n)\} \tag{4.30}$$

where m is the controller order, n is the plant order and $h(.)$ is a nonlinear function.

Once the data become available, a neural network is then used to approximate the function $h(.)$. However, for proper set point following, it is also needed that a zero steady-state error $e(t)$ in the set point produces a zero value of $\Delta u(t)$. Although, this has been guaranteed in the time varying controller in stage 2 due to its linear structure, the function $h(.)$ may not ensure that unless some restriction is applied to its structure.

We propose a specialized neural net structure for this purpose. This consists of two modifications, (a) the network contains no bias terms and (b) the neurons in the first hidden layers are replaced by special structure neurons as shown in Figure 4.6(b). A comparison of Figure 4.6(a) and 4.6(b) reveals that although zero values of $e(k)'s$ and $\Delta u(k)'s$ do not ensure a zero neuron output in the conventional neuron structure, the same ensures a zero neuron output into the modified network. This special neuron structure in the first hidden layer and absence of bias terms in the neural controller ensures proper set point tracking.

The training of the neural network can be done using the batch algorithm proposed in section 3.5.2 and in a similar way to the identification of the forward model in stage 1. However, both the forward and backward passes must consider the specialized structure of Figure 4.6(b). Usually a network with one hidden layer would be sufficient. The number of neurons in the hidden layer can be gradually increased until an accepted training error is achieved. After successful training of the neural



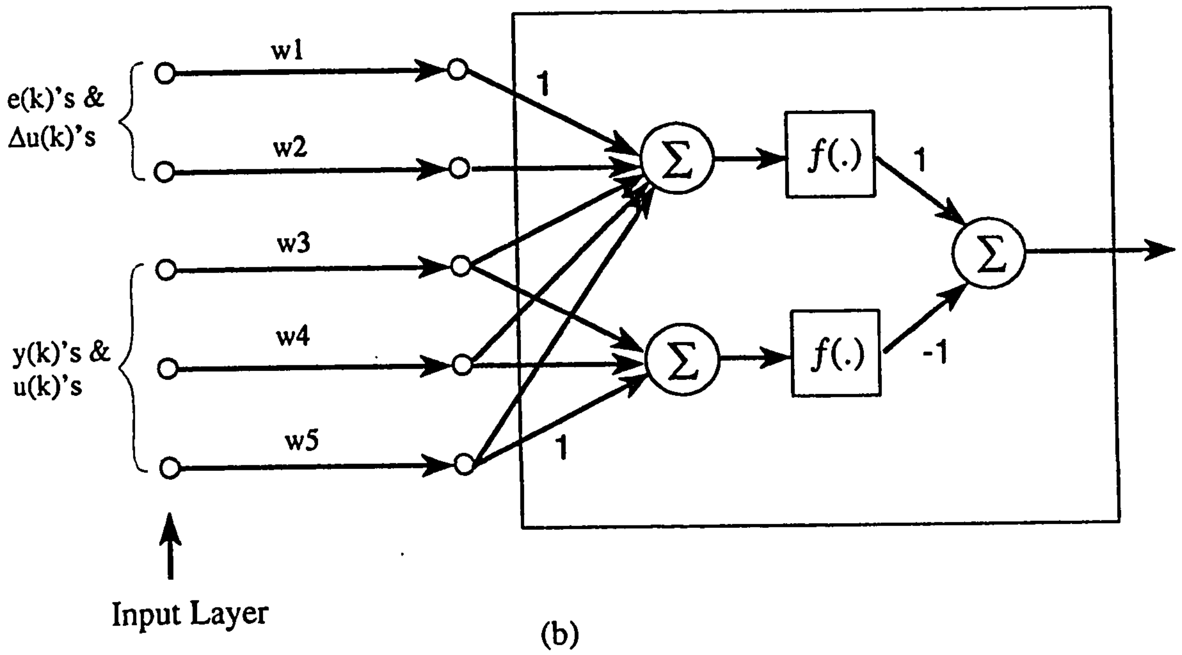


Figure 4.6: (a) Neuron structure in conventional NN (b) Neuron structure in the first hidden layer of the NN controller

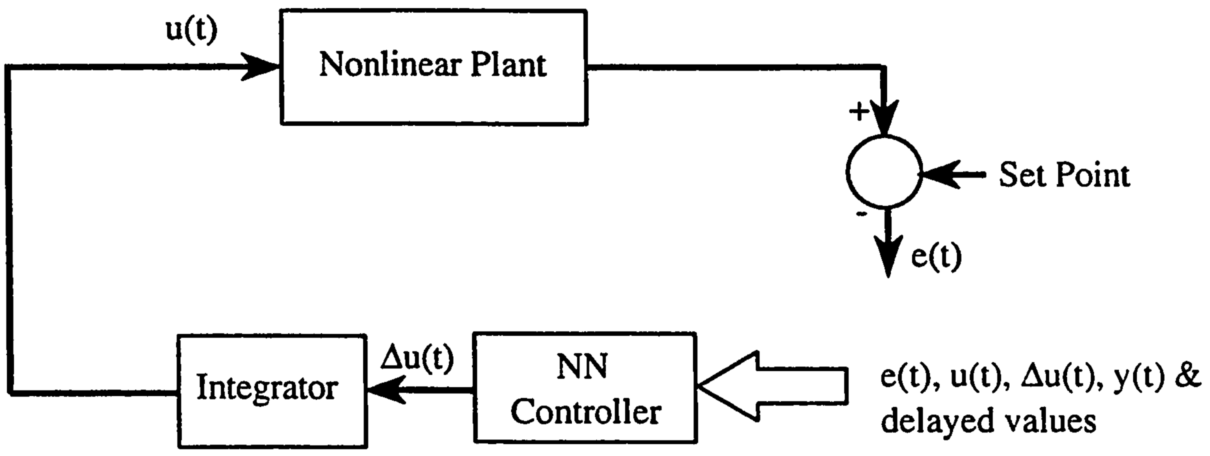


Figure 4.7: Plant control using neural network as the controller

controller, it can be implemented to control the nonlinear plant as shown in Figure 4.7.

4.3 Neural Net Based Controller For Multi-input Multi-output Plants

The three-stage design technique can also be applied to multi-input multi-output (MIMO) plants. The plant identification stage (stage 1) is the same as that for single-input single-output systems except that now the neural network model has as many neurons in the output layer as the number of outputs of the actual plant. However, the control of MIMO plants needs some special considerations as described below.

4.3.1 Pole Placement Controller Design For MIMO Plants (Stage 2)

Once the neural network is trained, it can be used to obtain a linearized model for given input and output values. The linearized model for a MIMO plant is given as

$$\begin{aligned} D(q^{-1})Y(t) &= N(q^{-1})U(t) \\ \text{or} \quad Y(t) = G(q^{-1})U(t) & \\ \text{where} \quad G(q-1) &= D^{-1}(q^{-1})N(q^{-1}) \\ \text{and} \quad D(q^{-1}) &= D_o + D_1 q^{-1} +, \ldots, +D_n q^{-n} \\ N(q^{-1}) &= N_o + N_1 q^{-1} +, \ldots, +N_n q^{-n} \end{aligned} \tag{4.31}$$

$D_i's$ and $N_i's$ are now matrices obtained through linearization.

Assume that the linear controller is given by

$$\begin{aligned} C &= N_c(q^{-1})D_c^{-1}(q^{-1}) \\ \text{where} \quad D_c(q^{-1}) &= D_{co} + D_{c1} q^{-1} +, \ldots, +D_{cn} q^{-m} \\ N_c(q^{-1}) &= N_{co} + N_{c1} q^{-1} +, \ldots, +N_{cm} q^{-m} \end{aligned} \tag{4.32}$$

$N'_{ci}s$ and $D'_{ci}s$ are all matrices.

Let G_f be the transfer matrix of the overall system. Then we have [41] [1]

$$G_f(q^{-1}) = [I + G(q^{-1})C(q^{-1})]^{-1} G(q^{-1})C(q^{-1}) \tag{4.33}$$

Substituting equations (4.31) and (4.32) in equation (4.33),

$$G_f(q^{-1}) \;=\; [I + D^{-1}(q^{-1})N(q^{-1})N_c(q^{-1})D_c^{-1}(q^{-1})]^{-1} \times$$

[1] In [41], all the equations are given in terms of continuous time systems. We have converted them for discrete time systems.

$$\begin{aligned} & D^{-1}(q^{-1})N(q^{-1})N_c(q^{-1})D_c^{-1}(q^{-1}) \\ = \; & D_c(q^{-1})[D(q^{-1})D_c(q^{-1}) + N(q^{-1})N_c(q^{-1})]^{-1} \times \\ & N(q^{-1})N_c(q^{-1})D_c^{-1}(q^{-1}) \qquad (4.34) \end{aligned}$$

$$Define \qquad D_f(q^{-1}) = D(q^{-1})D_c(q^{-1}) + N(q^{-1})N_c(q^{-1})$$

where D_f is the polynomial matrix describing the desired closed loop poles, then G_f becomes

$$\begin{aligned} G_f &= D_c(q^{-1})D_f^{-1}(q^{-1})[D_f(q^{-1}) - D(q^{-1})D_c(q^{-1})]D_c(q^{-1}) \\ &= I - D_c(q^{-1})D_f^{-1}(q^{-1})D(q^{-1}) \qquad (4.35) \end{aligned}$$

The design hinges on solving the Diophantine equation given by (4.35). The controller polynomial matrices $D_c(q^{-1})$ and $N_c(q^{-1})$ can be obtained from the solution of the following linear system of equations

$$[D_{co}\ N_{co} \vdots D_{c1}\ N_{c1} \vdots \ \ldots \ \vdots D_{cm}\ N_{cm}]S_m = [F_o\ F_1\ F_2\ \ldots\ F_{m+n}] \qquad (4.36)$$

where

$$\mathbf{S_m} = \begin{pmatrix} D_o & D_1 & \ldots & D_n & 0 & 0 & \ldots & 0 \\ N_o & N_1 & \ldots & N_n & 0 & 0 & \ldots & 0 \\ -- & -- & -- & -- & -- & -- & -- & -- \\ 0 & D_o & \ldots & D_{n-1} & D_n & 0 & \ldots & 0 \\ 0 & N_o & \ldots & N_{n-1} & N_n & 0 & \ldots & 0 \\ -- & -- & -- & -- & -- & -- & -- & -- \\ & & \vdots & & & & & \\ -- & -- & -- & -- & -- & -- & -- & -- \\ 0 & \ldots & 0 & D_o & \ldots & \ldots & D_{n-1} & D_n \\ 0 & \ldots & 0 & N_o & \ldots & \ldots & N_{n-1} & N_n \end{pmatrix} \qquad (4.37)$$

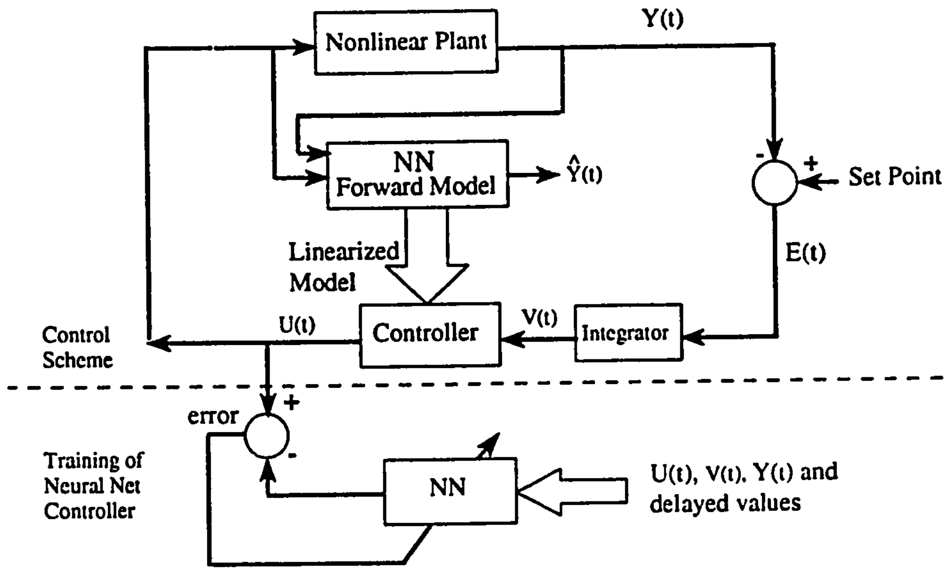


Figure 4.8: Control of a MIMO plant and neural net training using the linearized model

and $F_i's$ are the coefficient matrices of the desired closed loop polynomial matrix D_f. The control scheme is shown in Figure 4.8. Unlike the SISO plants, the integrator has been placed before the controller. This is to ensure that the input to the controller is constant only when all the errors are zero.

4.3.2 Training of the Neural Controller for MIMO Plants (Stage 3)

In this stage, the data from stage 2 is used to train a neural network to act as the final controller. The trained network is required to replicate the time-varying controller of the previous stage. The inputs to this network include the input-output of the time varying controller i.e. $\Delta E(t)$ and $U(t)$. However, the neural controller must also know the operating point in order to figure out the appropriate control

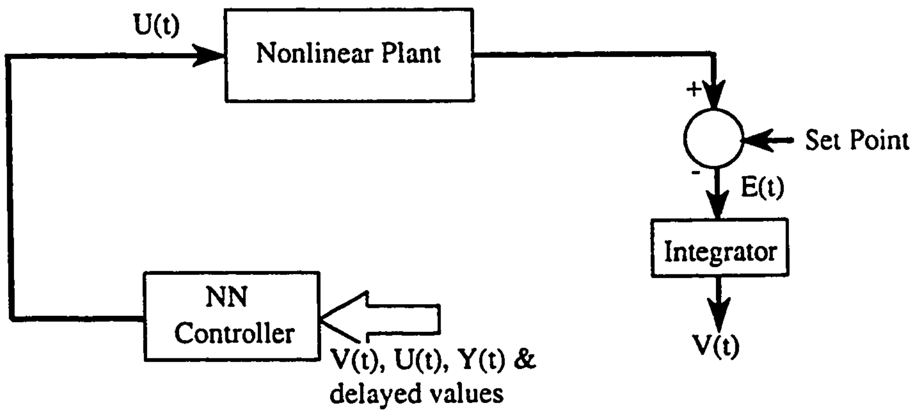


Figure 4.9: Control of a MIMO plant using neural network controller

law. The information about the operating point is buried in the plant input-output. Therefore the neural controller must receive $Y(t)$ and its delayed values as inputs too. The neural controller may be expressed as

$$U(t) = h\{V(t),\ldots,V(t-m-p)|U(t-1),\ldots,U(t-m-p)|Y(t),\ldots,Y(t-n)\} \quad (4.38)$$

where m is the controller order, n is the plant order, p is the number of outputs of the plant and $h(.)$ is a nonlinear function. Once data is available, the neural network may be used to assume the function $h(.)$. The training of the neural network can be done using the batch algorithm proposed in section 3.5.2.

However, since the required neural network size is going to be very large, it is better to do some preliminary sweeps of pattern learning until the error ceases to change. Starting from these values of the weights, batch learning can then be used. This has a twofold advantage. First the computation time may be reduced. Secondly, and more importantly, because the pattern learning algorithm presented in section 3.5.1 is known to be capable of bypassing any local minima [34], this preliminary training is likely to bring the neural weights closer to the global minima. The batch learning of section 3.5.2 can then be used to improve on the accuracy of training (fine tuning). The batch learning with matrix learning rate, being a second order method, is capable of fast convergence to the nearest minima. Usually a network with one hidden layer is sufficient. The number of neurons in the hidden layer can be gradually increased until an accepted training error is achieved. After successful training of the neural controller. it can be implemented to control the nonlinear plant as shown in Figure 4.9.

Chapter 5

SIMULATION RESULTS

The proposed design approach has been applied to various nonlinear plants. The results obtained are presented in this chapter. In order to investigate the effect of noise on the proposed strategy, simulation results for a noisy plant are also reported.

5.1 Introduction

We have used the Model IV representation of nonlinear plants presented in section 4.1. In order to carry out the identification, we have assumed that the model order is known and the plant is either open loop stable or stabilized in closed loop.

Both for the plant and controller representations, two layer neural networks (i.e. one hidden layer) have been found to be adequate.

All the simulations have been carried out using MATLAB on PC i486 (33 MHz).

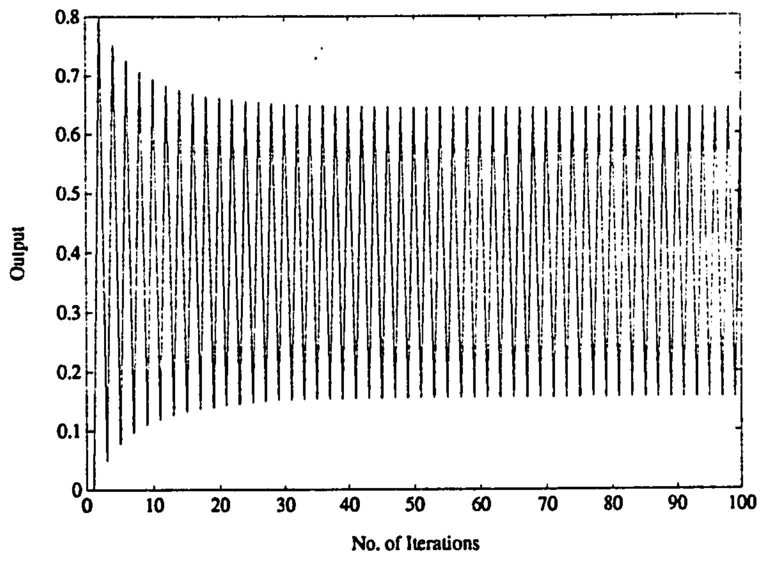


Figure 5.1: Plant response with step input of amplitude 0.8

5.2 A First Order Nonlinear Plant

The plant is assumed to be described by the following equation

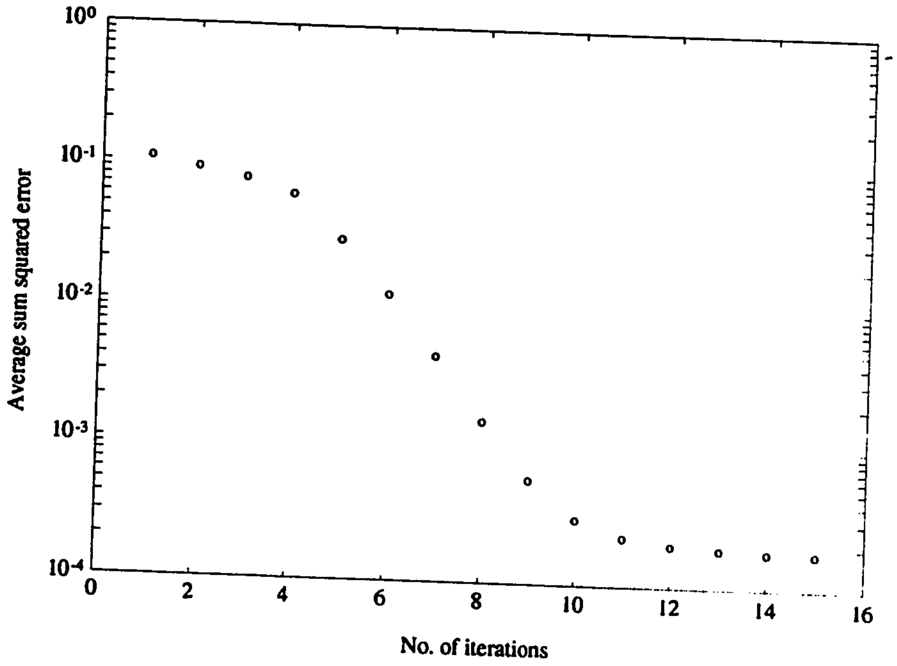
$$y(t) = \frac{-0.9y(t-1) + u(t-1)}{1 + y^2(t-1)} \tag{5.1}$$

The model is taken from [42]. Figure 5.1 shows the open loop step response with $u(t) = 0.8$. The step response of the plant is very oscillatory for low amplitude input and shows limit cycle oscillation for $u(t) \geq 0.7$ as shown in the above figure. A neural net with one hidden layer of three neurons has been trained to model the plant. Batch training has been performed along with the matrix learning rate given in section 3.5.2. The parameters ϵ, $\mu(0)$, μ_r and μ_∞ in equations (3.26) and (3.27) have been set to 0.1, 0.1, 0.9 and 1.0 respectively. The training error as a function of the sweep iteration is displayed in Figure 5.2. The rapid convergence of the prediction error (pe) algorithm can be observed from this figure. The figure also shows a comparison of the NN model output and the desired plant output. It can be seen that it is very difficult to differentiate between these two outputs. The forward neural net model has been used to compute the linearized model. Control of the plant based on this model and the linear time varying controller using the configuration of Figure 4.3 has been implemented. Figure 5.3 shows a typical output and input when pole placement control is used. The closed loop pole has been placed at 0.675. The controller parameters are calculated by solving the diophantine equation [39] at every time step. It can be observed that satisfactory control and set point tracking are achieved. The transient response however does not reflect a first order response with the closed loop pole at 0.675. This is to be expected as the plant is nonlinear and the control is achieved by successive linearization.

In order to train a neural controller to replicate the time varying controller, 1000 training data have been generated. For these training data, the set point is varied between ± 0.45 in a random fashion as shown in Figure 5.4. In stage 3, these data are used to train the neural controller having no bias term and specialized neuron structure in the first layer as shown in Figure 4.6(b). The matrix learning rate of section 3.5.2 has been used, with ϵ, $\mu(0)$, μ_r and μ_∞ being 0.1, 0.1, 0.9 and 1 respectively. One hidden layer with four neurons is observed to adequately capture the controller characteristics. Since both n and m are equal to 1, the neural controller has a structure (see equation(4.30))

$$\Delta u(t) = h\{e(t), e(t-1), \Delta u(t-1), y(t-1), u(t-1)\} \tag{5.2}$$

The training error as a function of sweep iterations is shown in Figure 5.5 along with a plot of the desired control input (Δu) and the neural net controller output. After



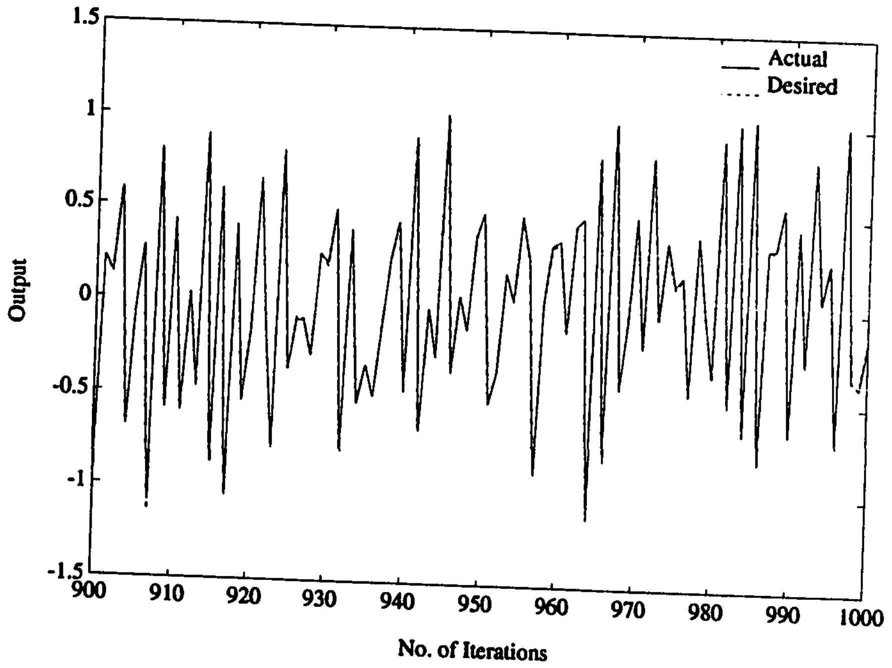
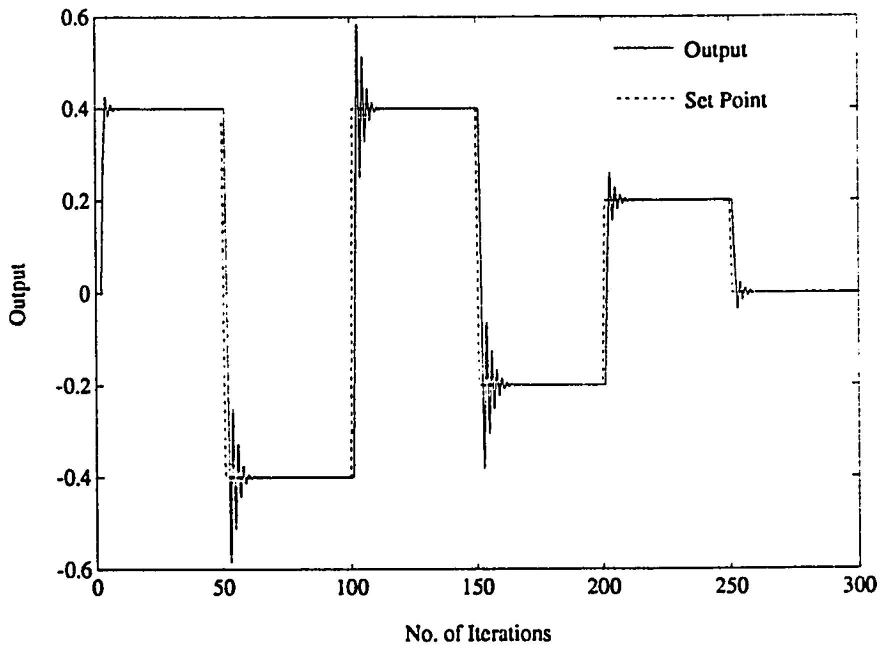


Figure 5.2: Identification of a first order nonlinear plant



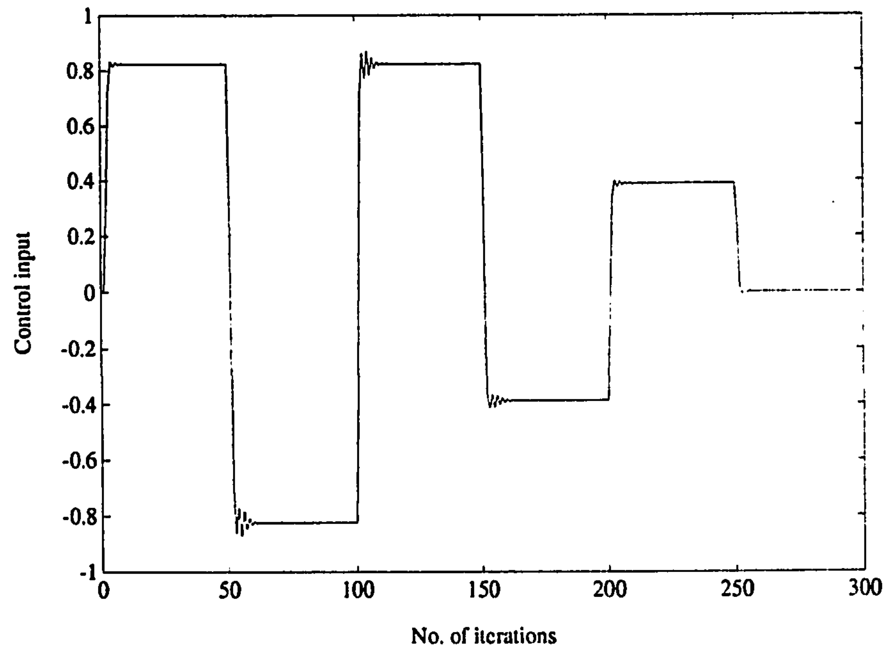


Figure 5.3: Control of a first order plant using time varying linear controller and pole placement principle

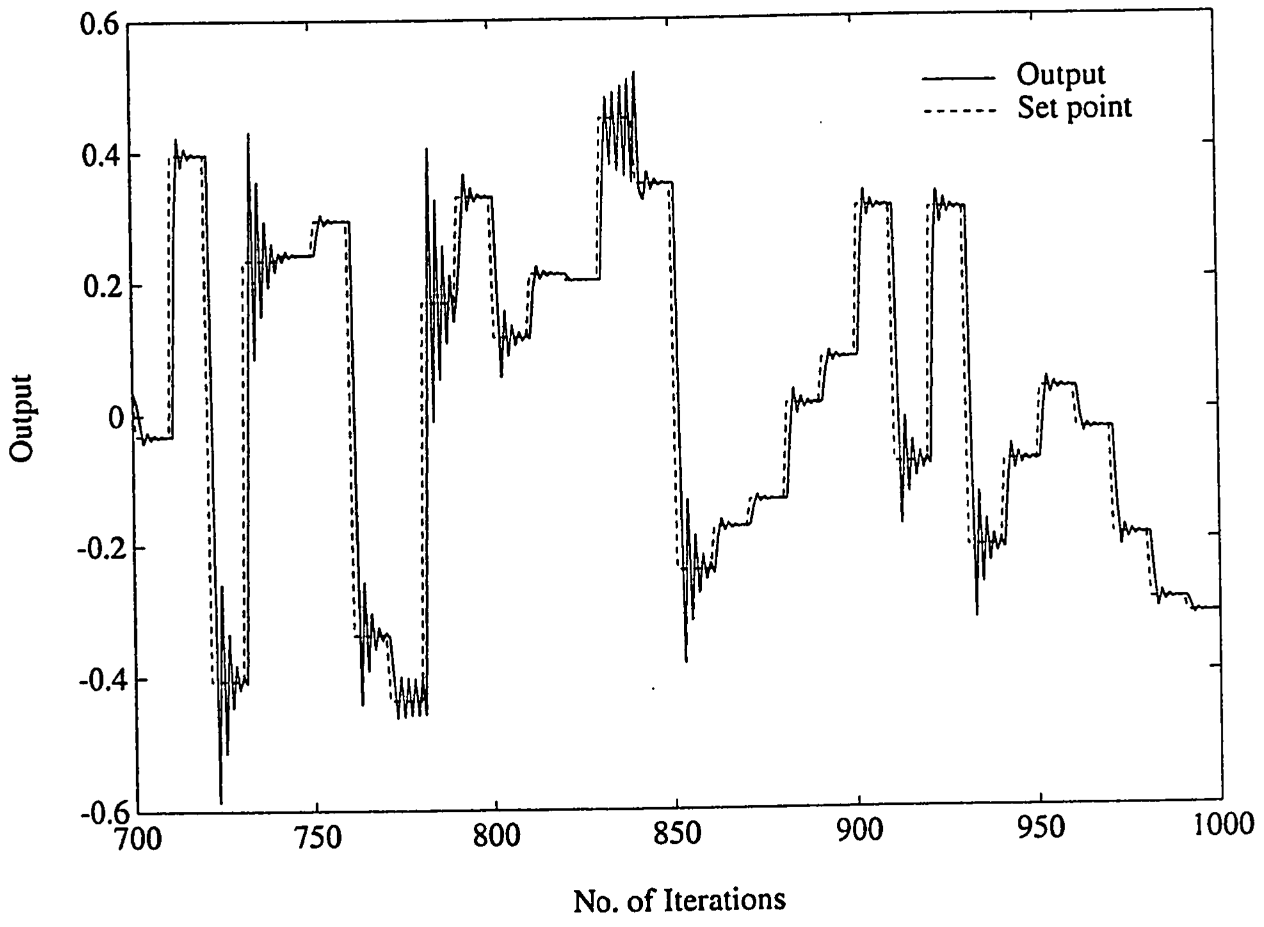
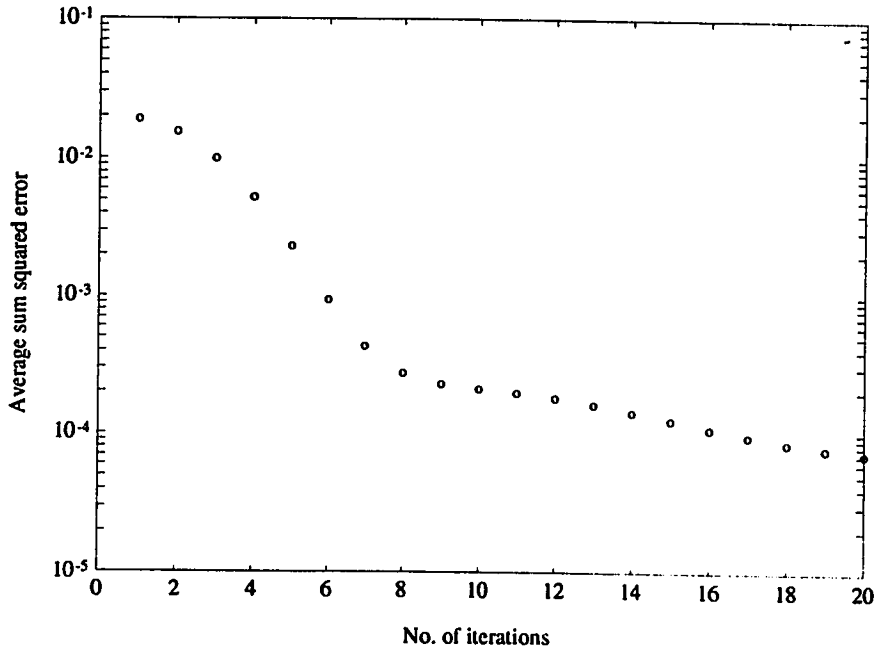


Figure 5.4: Training data for the neural controller for the first order plant



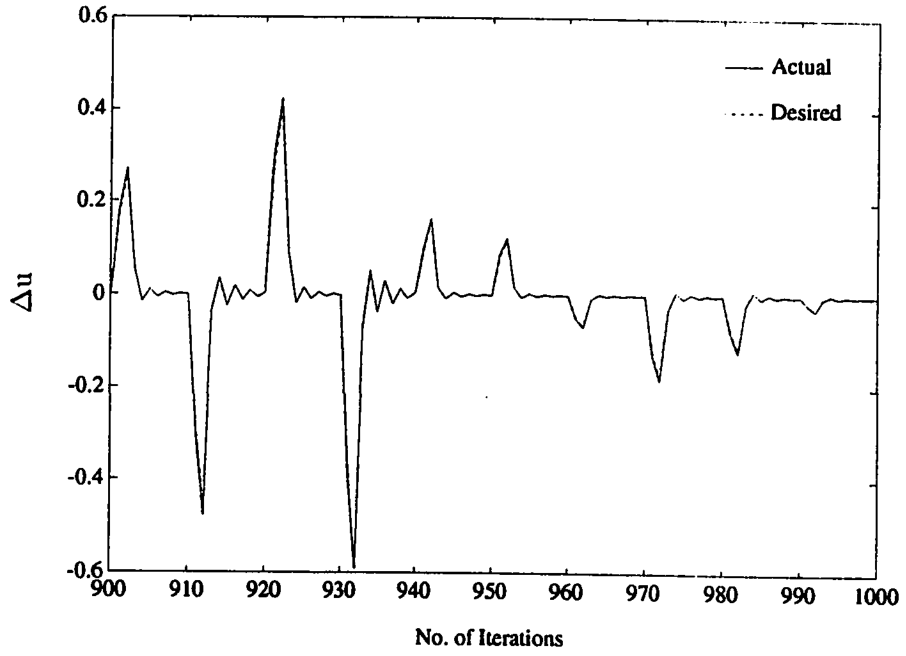
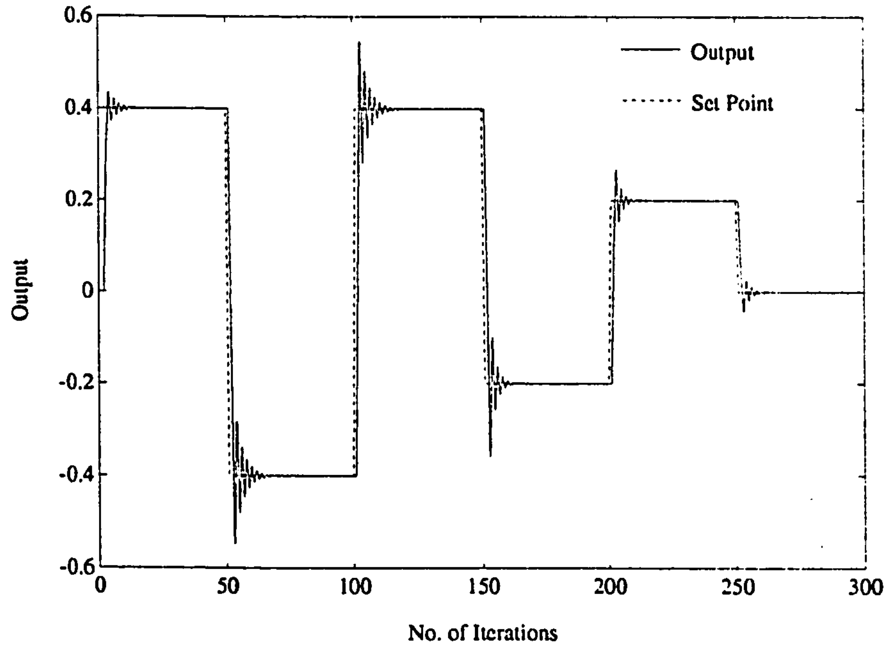


Figure 5.5: Training of neural controller for the first order plant



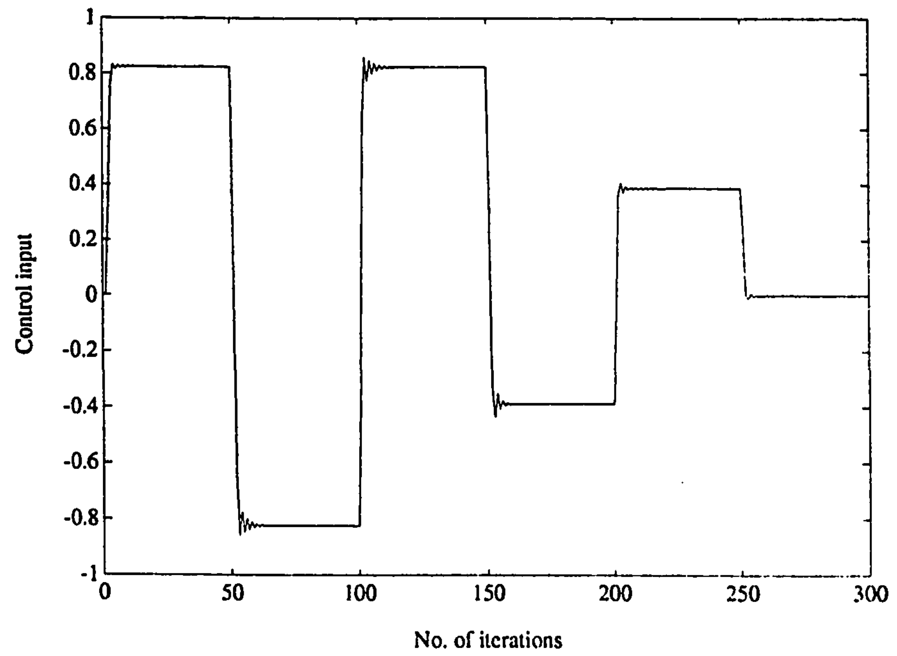
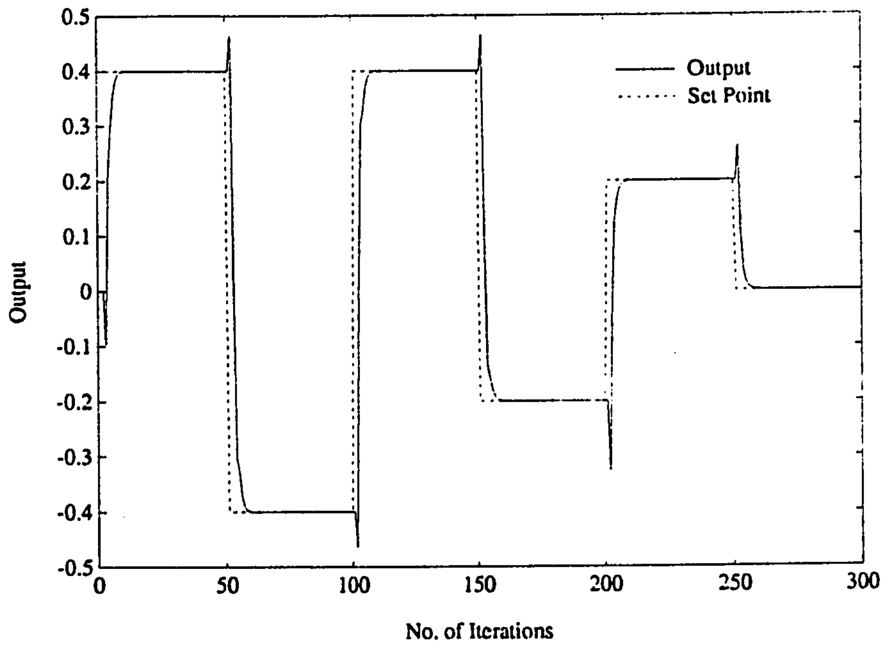


Figure 5.6: Control of the first order plant using the neural net controller and pole placement principle



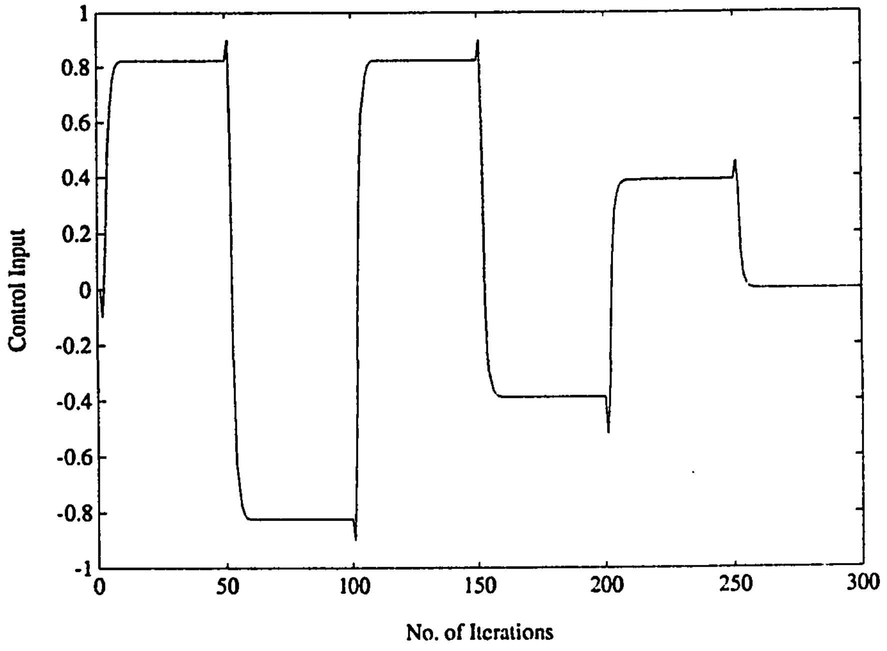
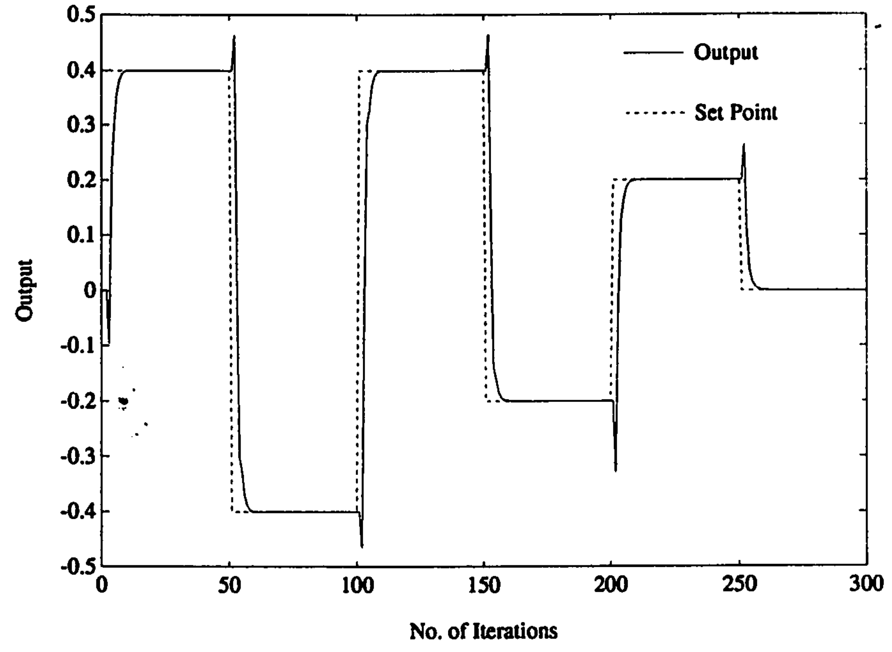


Figure 5.7: Quadratic optimal control of the first order plant using time varying linear controller



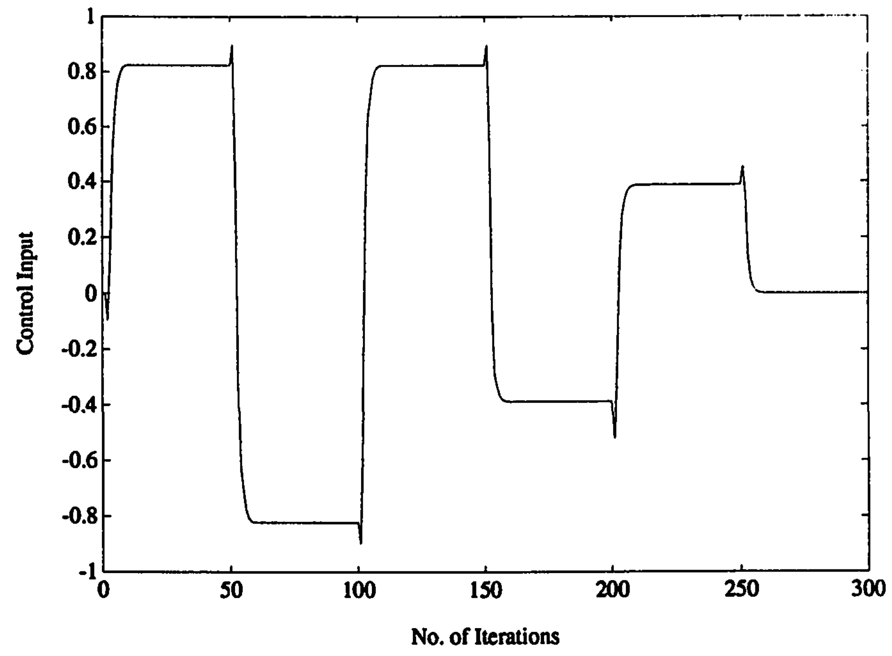
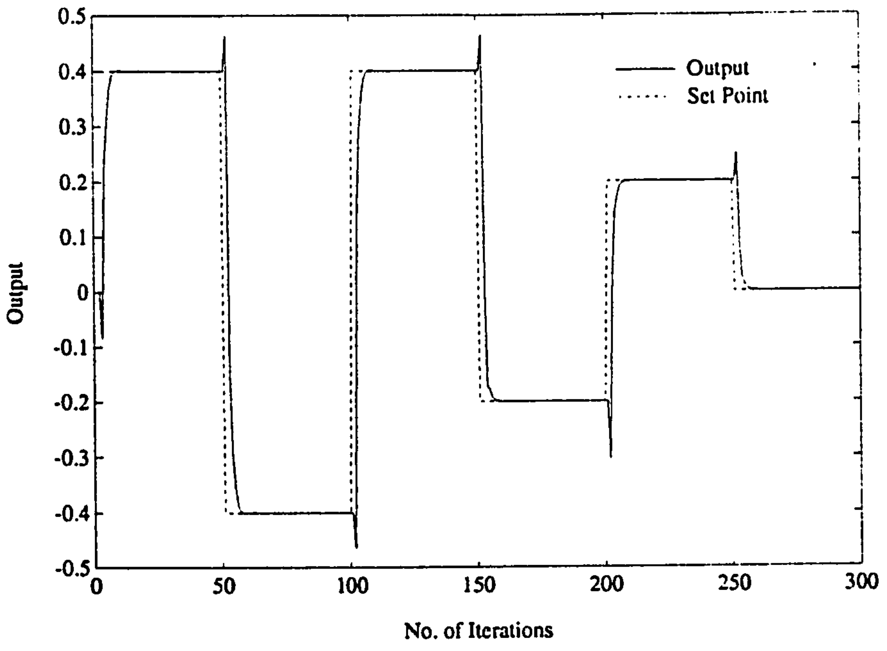


Figure 5.8: Quadratic optimal control of the first order plant using neural net controller



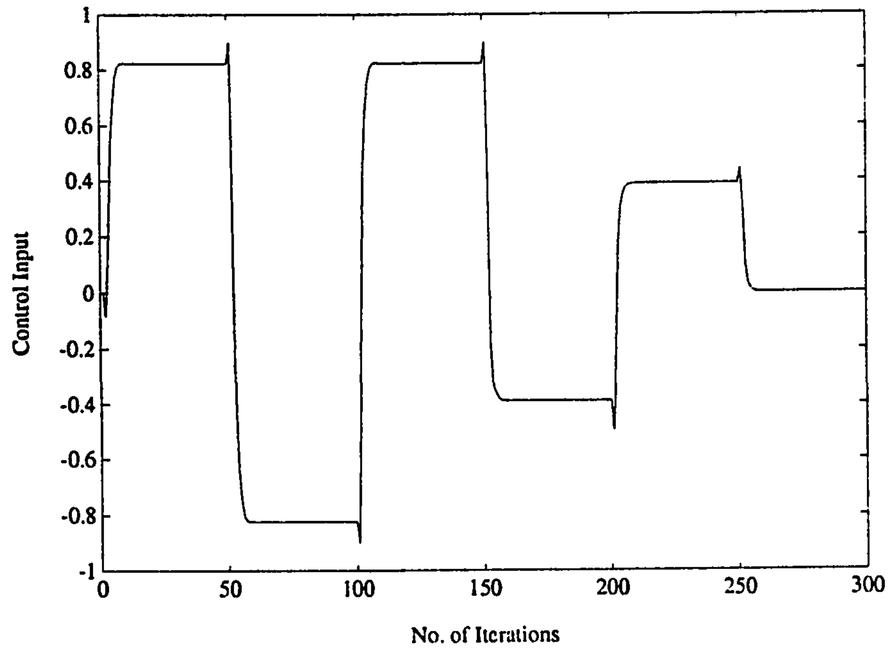
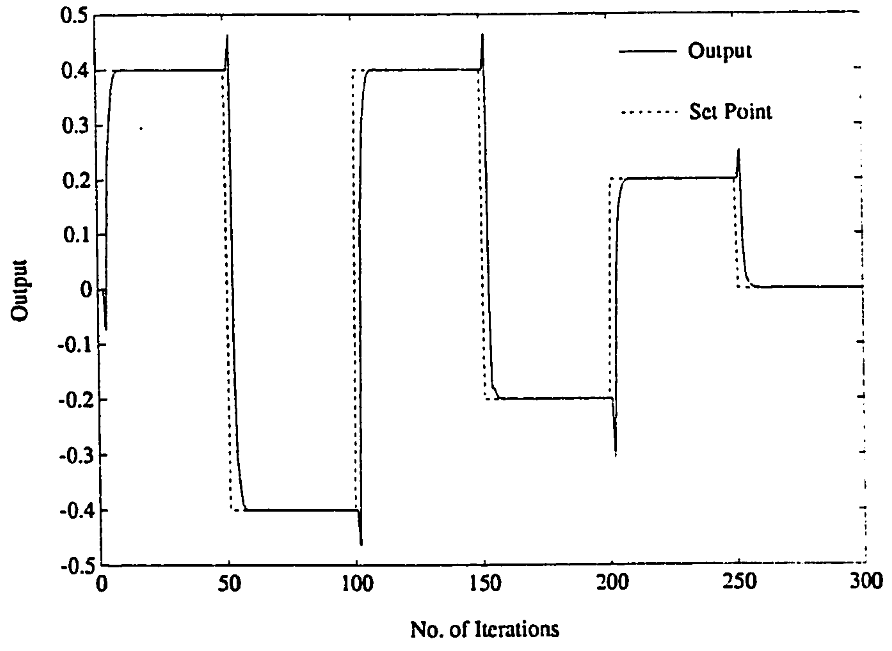


Figure 5.9: Control of the first order plant using the time varying linear controller and pole restriction principle



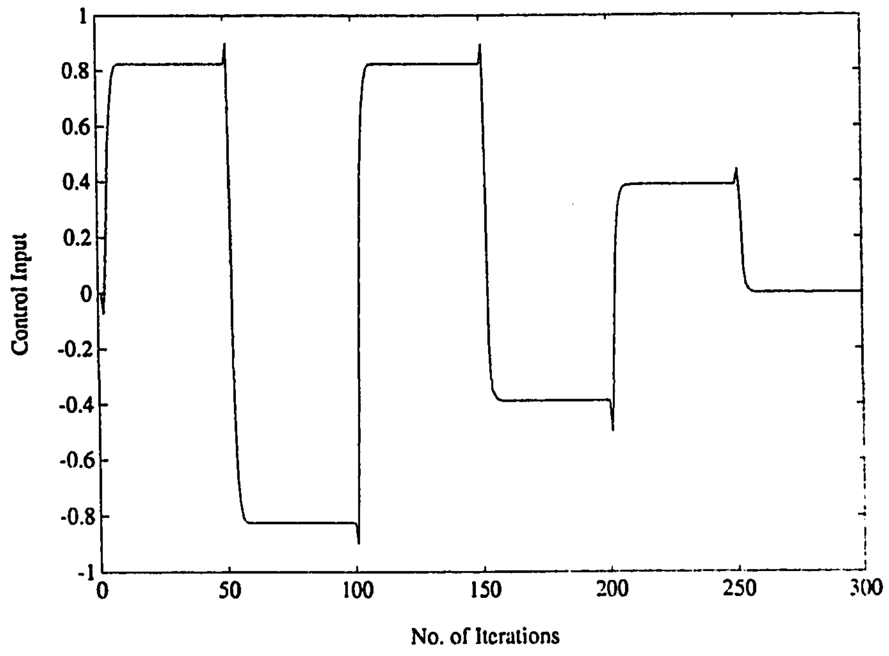
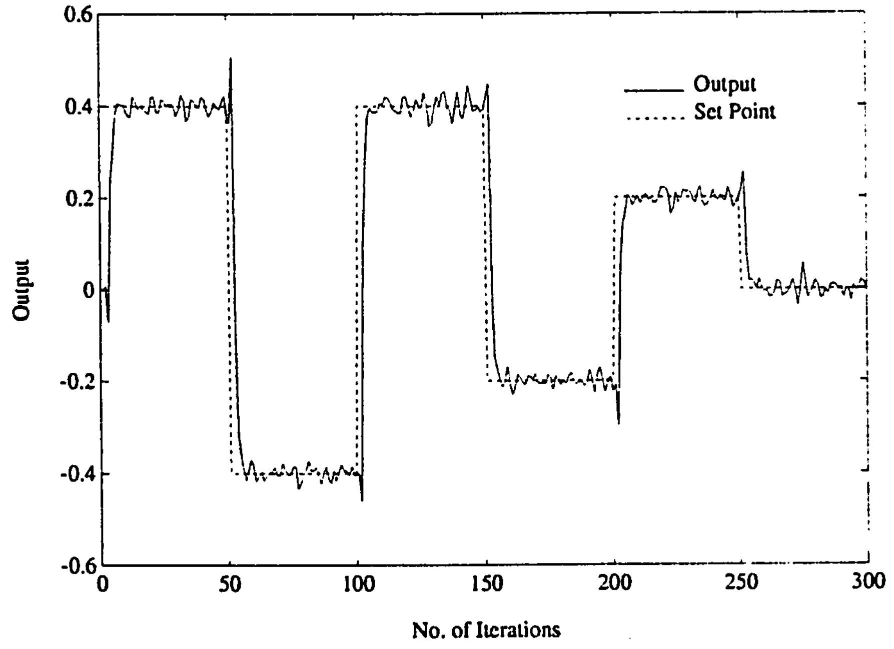


Figure 5.10: Control of the first order plant using the neural net controller and pole restriction principle



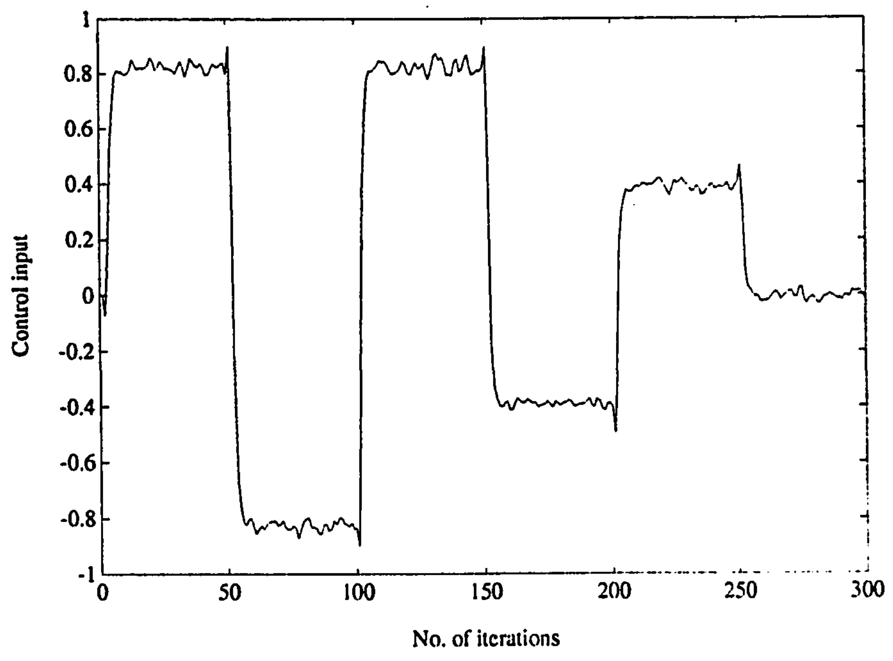
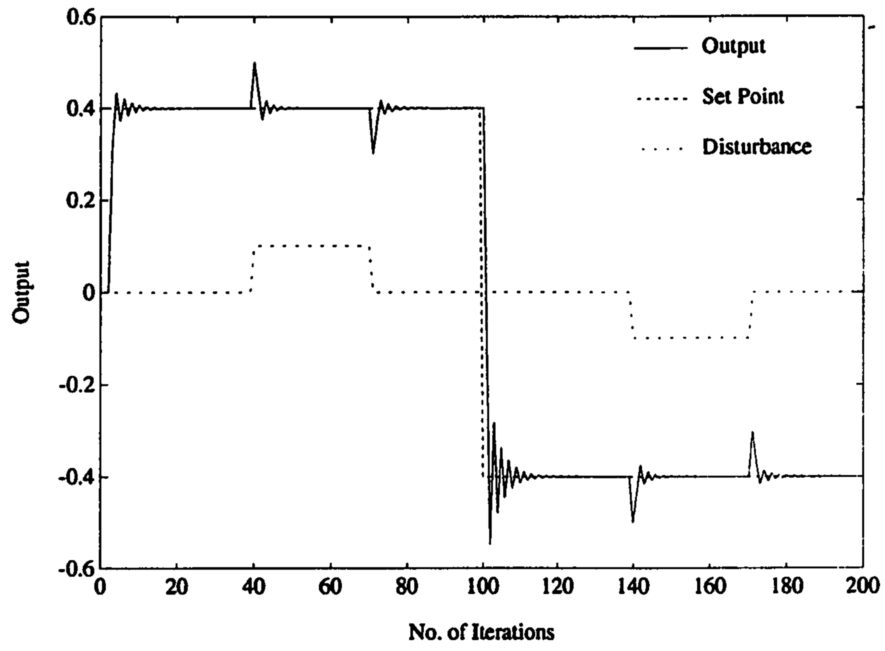


Figure 5.11: Control of noisy plant using the neural net controller and pole restriction principle



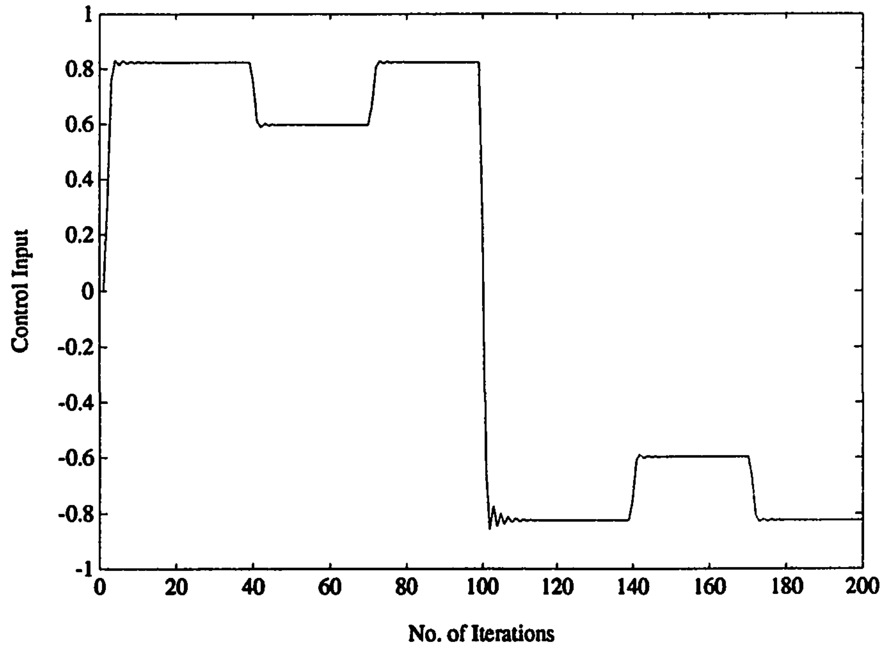


Figure 5.12: Control of the first order plant with step disturbance using the neural net controller and pole placement principle

successful training of the neural controller, the latter has been used to control the plant through the configuration of Figure 4.4. Figure 5.6 shows the plant output and input. A comparison of Figures 5.3 and 5.6 shows the capability of the neural controller in replicating the time-varying controller behavior. Figures 5.7 and 5.8 show the plots when optimal control, instead of pole placement, has been used as the control strategy with $\rho = 1$. Figures 5.9 and 5.10 show the simulation results with pole restriction control. Here ρ, α and β (see equation(4.33)) have been chosen as 1, 0.3 and 0.5 respectively. This corresponds to a damping coefficient (ζ) of at least 0.5 and a time constant (τ) of at most 5 samples. From Figures 5.8 and 5.10, it can be seen that with optimal and pole restriction control the outputs attain the set point rapidly without much oscillations. This is not the case with pole placement control as seen from Figure 5.3. The similarity in set point tracking capability between the optimal control and pole restriction control of the plant, can also be observed from these figures.

5.2.1 Effect of Noise

In order to investigate the effect of noise on the proposed strategy, some simulation has also been conducted by adding noise to the plant output. Noise has also been added to the training data for identification (stage 1), during training data generation in stage 2 and during plant control using the neural controller (stage 3). The standard deviation of the noise has been 0.01. Batch learning rate of section 3.5.2 has been used for identification with $\mu(0)$, μ_∞ and r being 0.1, 1 and 0.9 respectively. Figure 5.11 shows the output of the noisy plant. Other than the presence of noise, the control performance looks satisfactory.

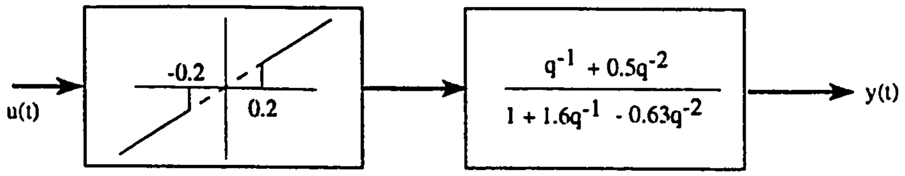


Figure 5.13: The Hammerstein model

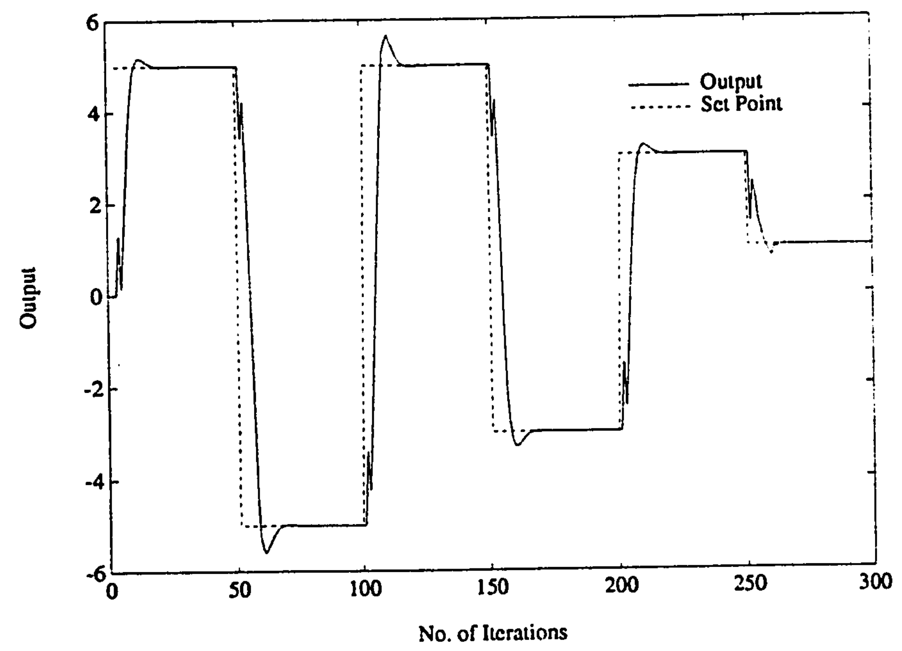
We also investigated the effect of deterministic disturbance by adding a step disturbance of 0.1 and -0.1 to the plant output. This disturbance was added at the 40th and 140th iterations and removed at the 70th and 170th iteration respectively. It can be seen from Figure 5.12 that the neural net controller quickly recovers the effect of this disturbance.

5.3 A Second Order Hammerstein Model

A second-order Hammerstein Model shown in Figure 5.13 is considered. The nonlinearity represents a typical dead zone. The training of the forward model for this plant has been done using one hidden layer with seven neurons. The three control design methods, namely pole placement, optimal control and pole restriction, have been simulated using the time-varying controller. All the neural controllers have been trained using 2000 data points and a set point varying randomly between ± 6 in a similar way as in section 5.1. The control of the plant by the time varying linear controller and neural controller based on pole placement principle is shown in Figures 5.14 and 5.15 respectively. The closed loop poles have been specified as $0.6290 \pm j0.2316$. This corresponds to a damping of 0.75 and time constant of 2.5 samples. The satisfactory control and set point tracking can be observed from these figures. The control of the plant using the optimal control principle was found to be unsatisfactory as shown in Figure 5.16 when ρ was chosen as 1. The primary reason for this has been bad location of the closed loop poles as specified by the optimal control law. With ρ=1 and the plant output level being around 6, the pole locations were -0.4077 and 0.3129. Although such pole location could be satisfactory for a certain linearized model, due to estimation error and/or drift in the operating point it may not be suitable for the plant. In order to overcome this problem, we applied pole restriction control, where the closed loop poles are restricted to lie within a region specified by a circle with ρ, α and β being 1, 0.3 and 0.5 respectively. This corresponds to ζ (at least) = 0.5 and τ (at most) = 5 samples. The training data used for the neural net controller are shown in Figure 5.17. A comparison of Figures 5.16 and 5.18 shows that pole restriction principle can overcome the problems associated with optimal control. A comparison of Figures 5.15 and 5.18 also shows the superior performance of the pole restriction principle. It can be shown that the former scheme has superior robustness properties compared to the latter one [40]. When the plant is adaptively controlled using the strategy of Figure 4.4 and the pole restriction principle, the output and input have been as shown in Figure 5.19. A comparison of Figure 5.18 and Figure 5.19 shows slightly better performance of adaptive control strategy.

5.4 A Second Order Weiner Model

A second order Weiner Model shown in Figure 5.20 is considered. The training of the forward model for this plant has been done using one hidden layer with seven neurons. Plant control using the linearized model and the time varying controller



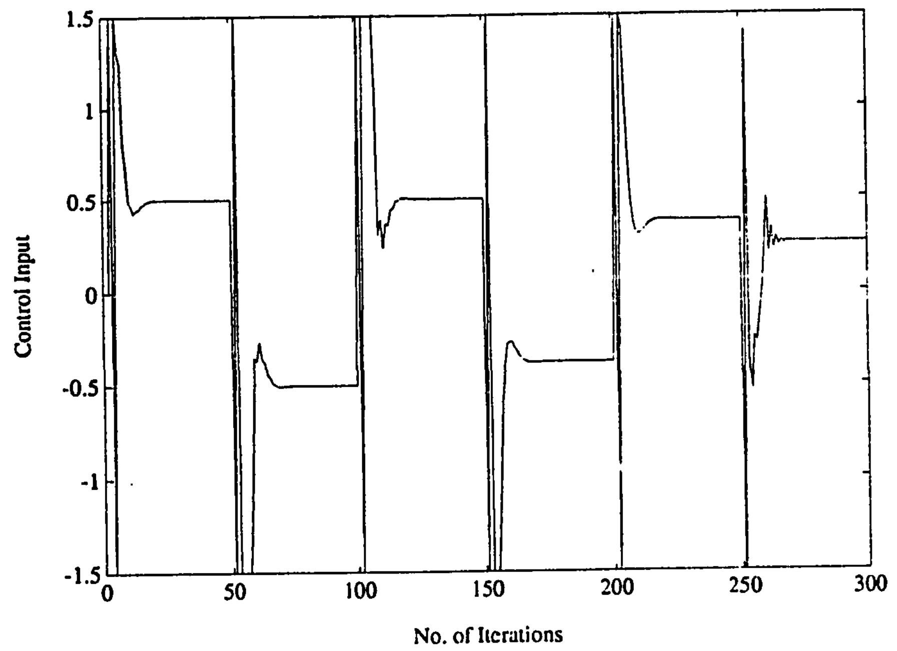
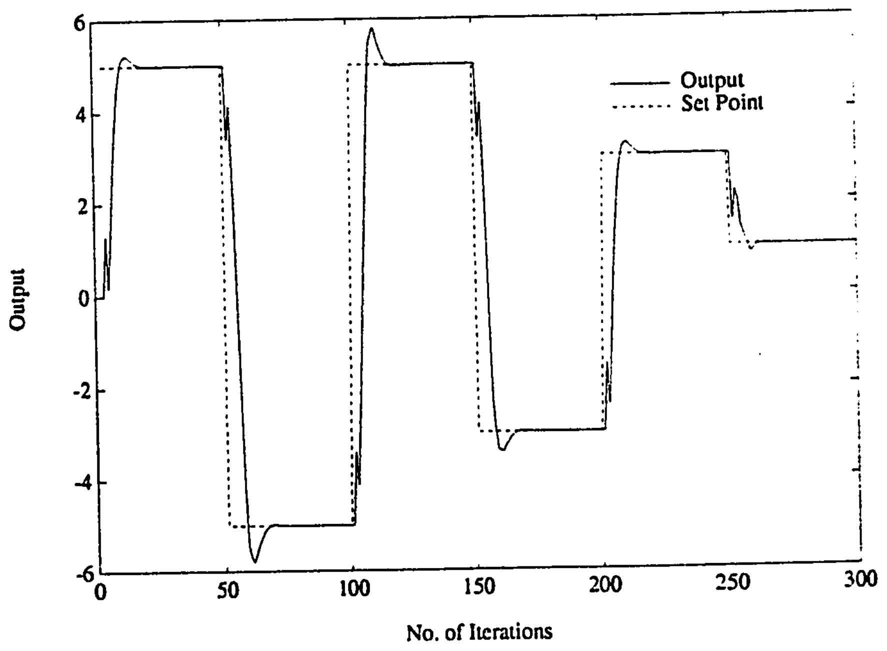


Figure 5.14: Control of the second order Hammerstein model using time varying linear controller and pole placement principle



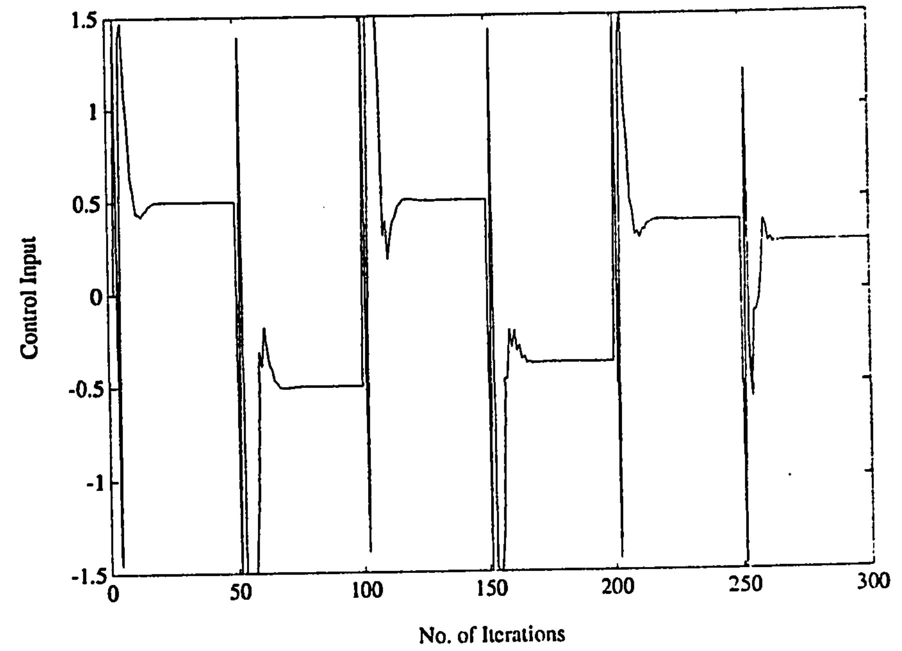
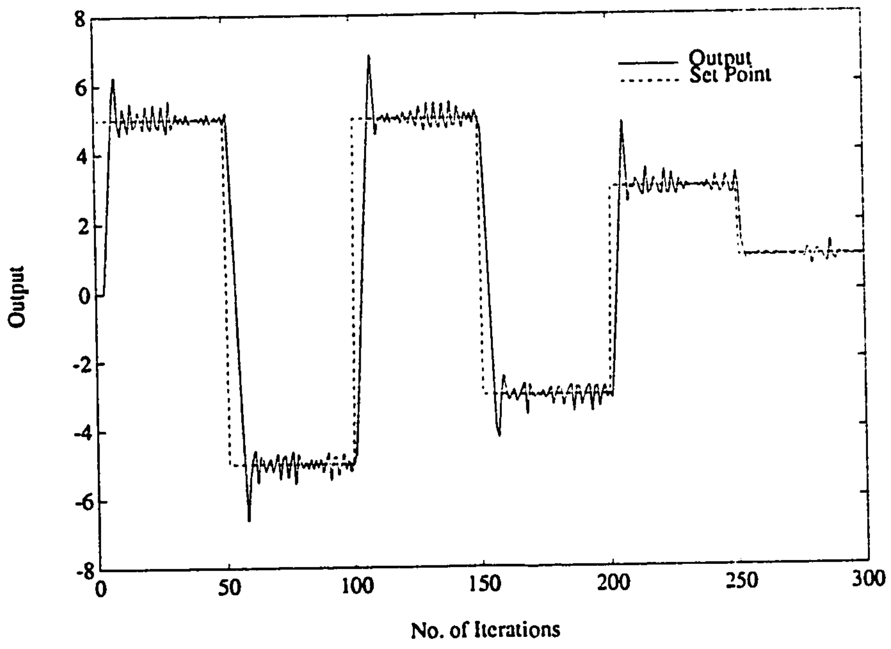


Figure 5.15: Control of the second order Hammerstein model using the neural net controller and pole placement principle



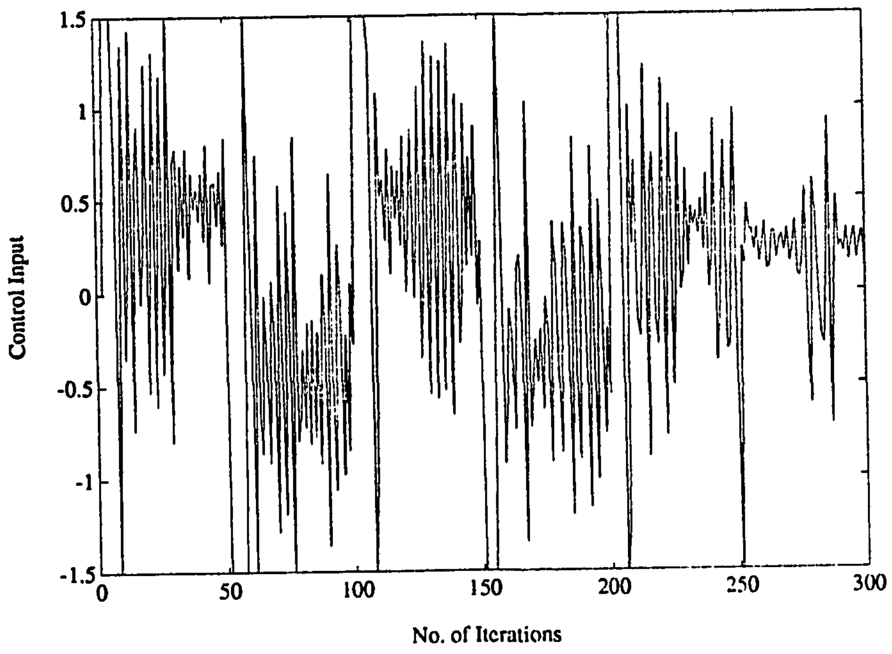


Figure 5.16: Quadratic optimal control of the second order Hammerstein model using the time varying linear controller

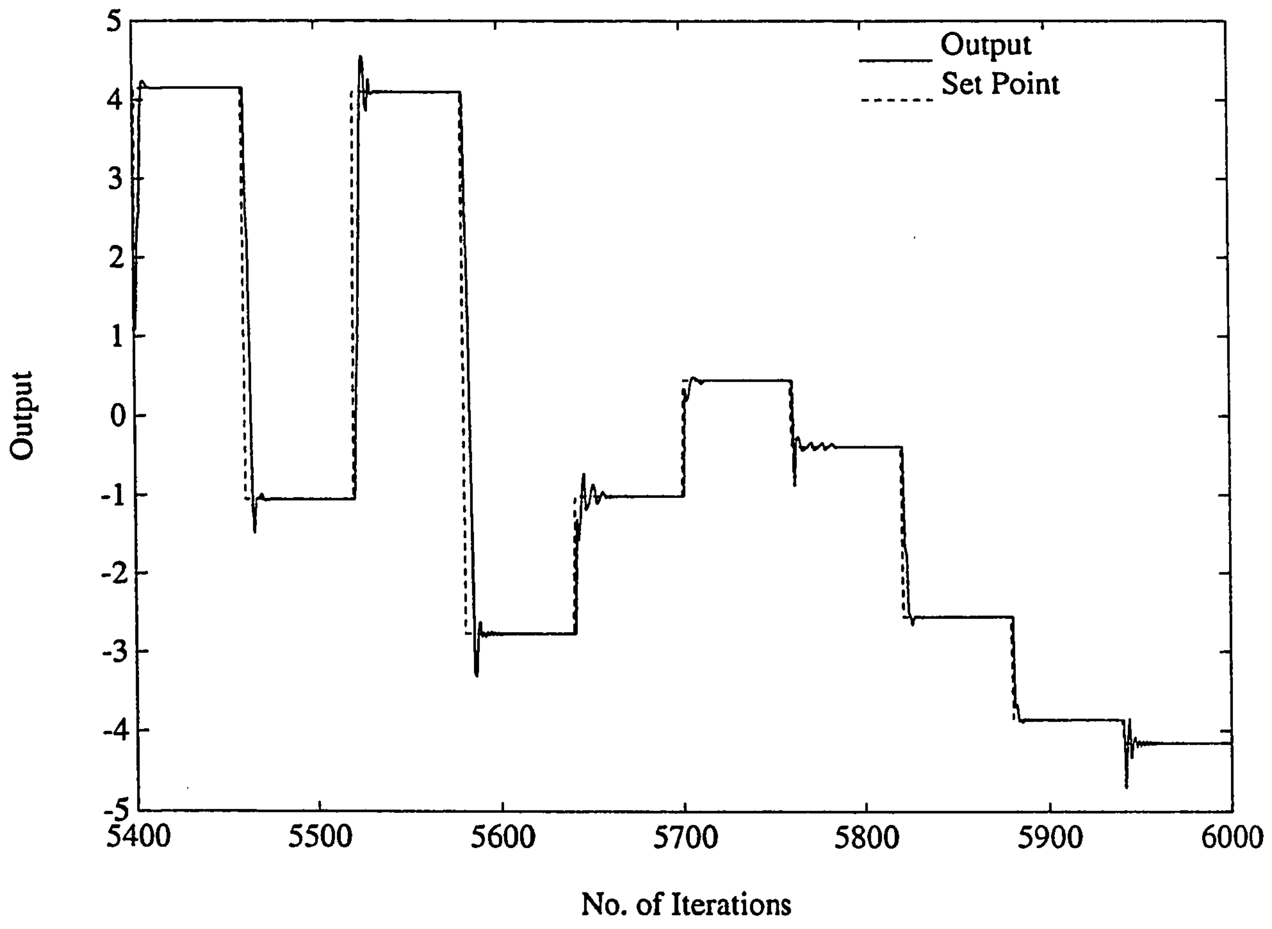
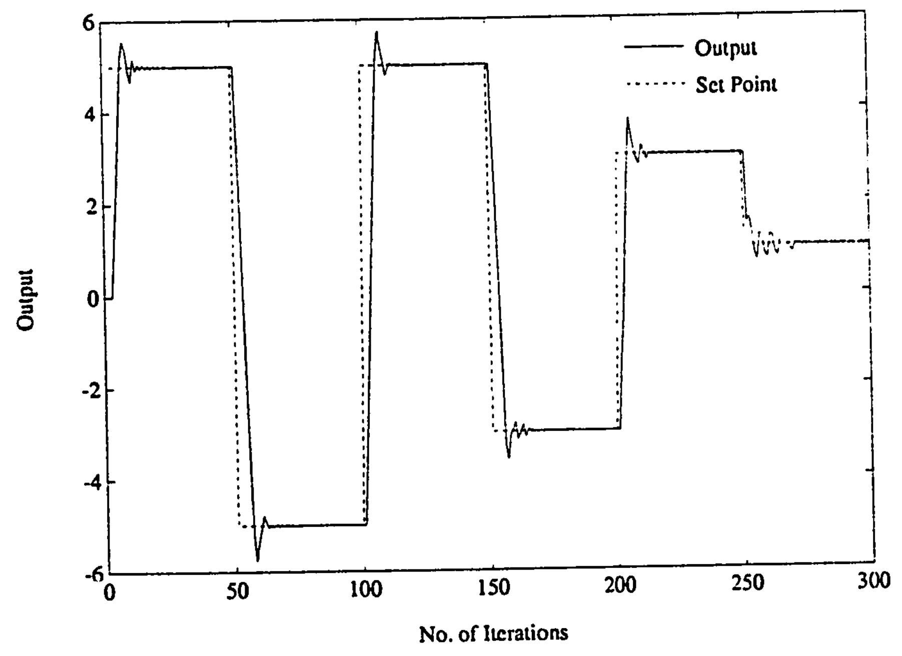


Figure 5.17: Training data for the neural controller for the second order Hammerstein model



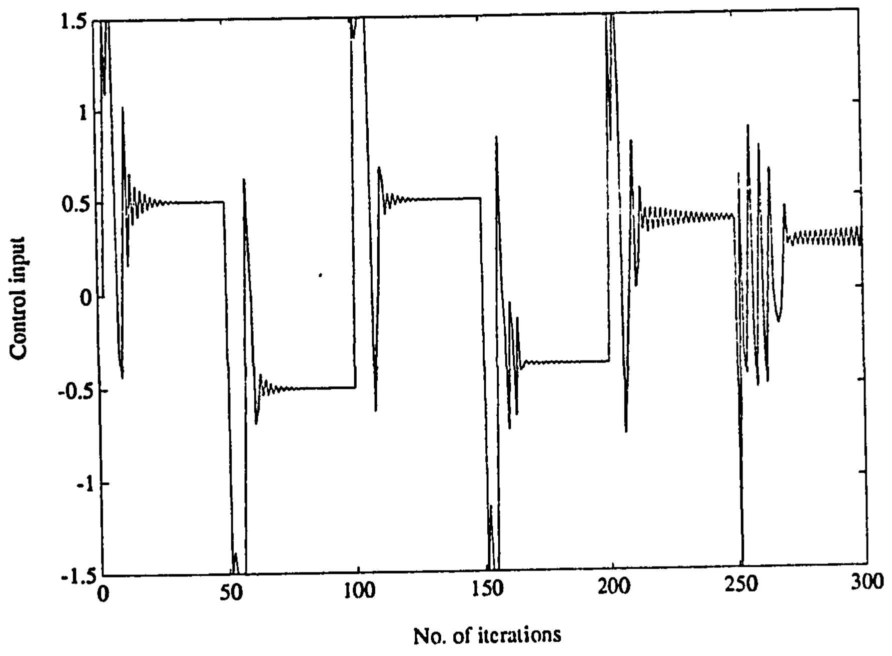
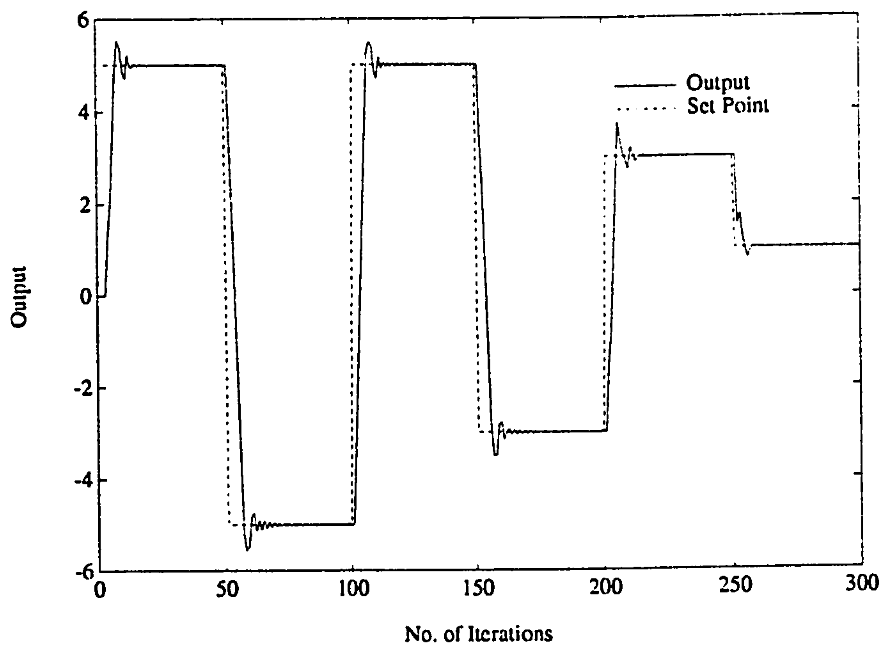


Figure 5.18: Control of the second order Hammerstein model using the neural net controller and pole restriction principle



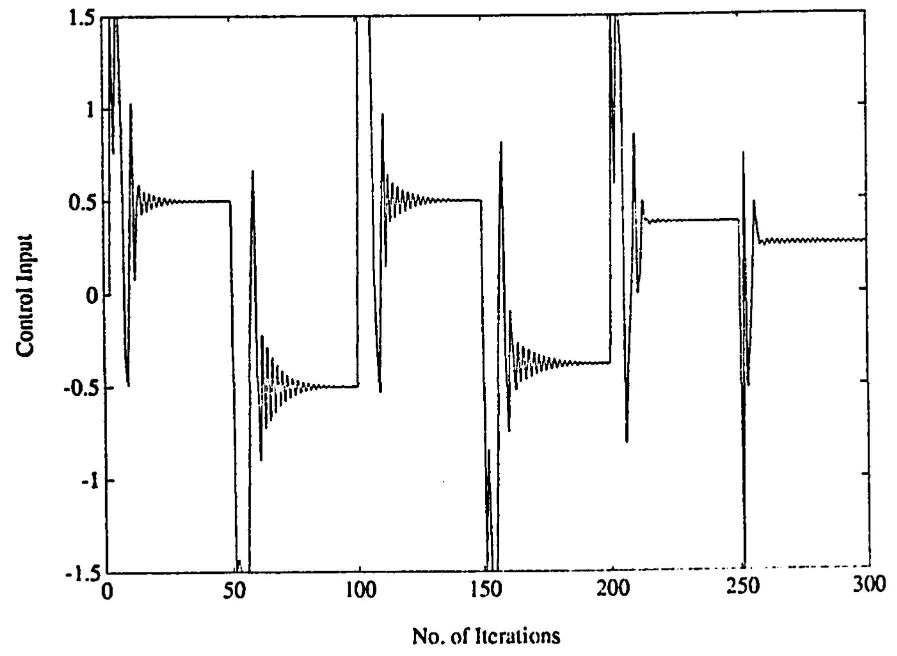


Figure 5.19: Adaptive control of the second order Hammerstein model using the time varying linear controller and pole restriction principle

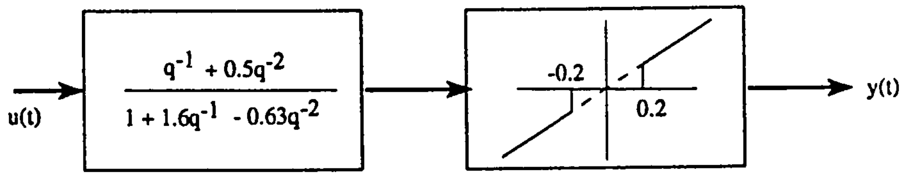


Figure 5.20: The Weiner model

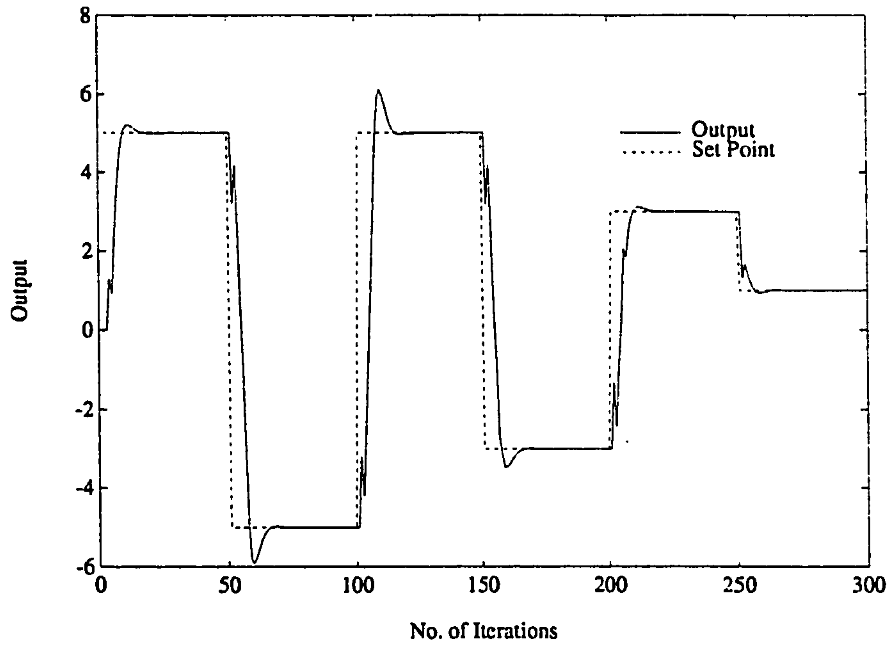
based on pole placement and pole restriction have been implemented. The neural controllers have been trained using 2000 data points where the set point was randomly varied between ± 6. The control of the plant by time varying linear controller and the neural controller based on pole placement principle is shown in Figures 5.21 and 5.22. The closed loop poles have been specified as $0.6290 \pm j0.2316$. This corresponds to a damping ratio of 0.75 and a time constant of 2.5 samples. The satisfactory control and set point tracking performance of this scheme can be observed from these figures. Figures 5.23 and 5.24 show the output and input of the plant when the pole restriction principle was used. The parameters ρ, α and β (see equation (4.33)) for this case have been 1, 0.3 and 0.5 respectively.

5.5 A 2-input 2-output Plant

The plant is assumed to be described by the following equations

$$\begin{aligned} y_1(k) &= y_1(k-1) - 0.07y_1^3(k-1) - 0.07\left[y_1(k-1)+1\right]y_2(k-1) + \\ &\quad 1.05u_1(k-1) \\ y_2(k) &= y_2(k-1) + \frac{1.05u_2(k-1) - 0.7y_2(k-1) - 0.3325y_1^2(k-1)}{1+0.1y_1(k-1)} \end{aligned} \tag{5.3}$$

The model is originally taken from [43] in continuous time and is discretized using a sampling time of 0.7. Since for the identification stage, it is required that the



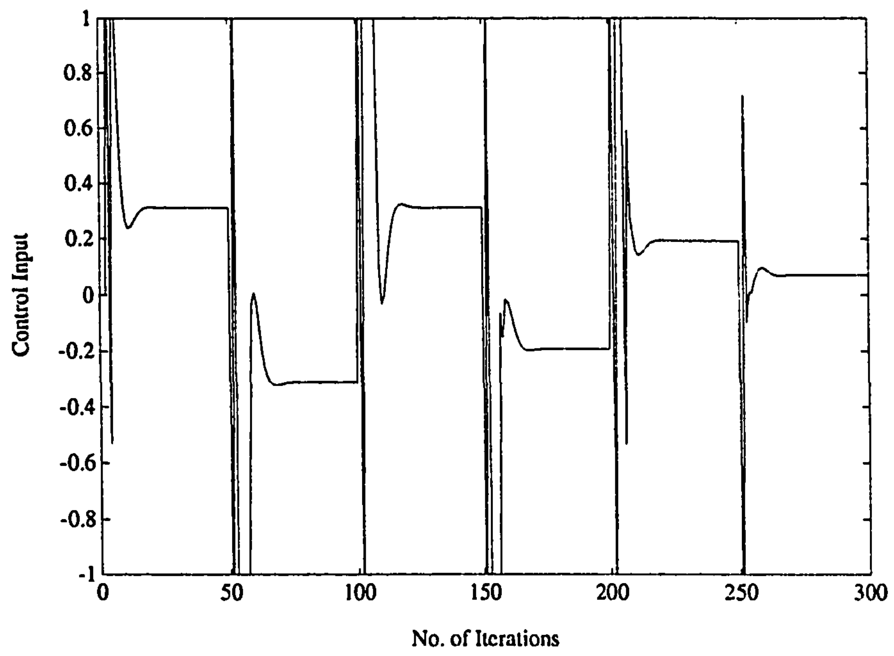
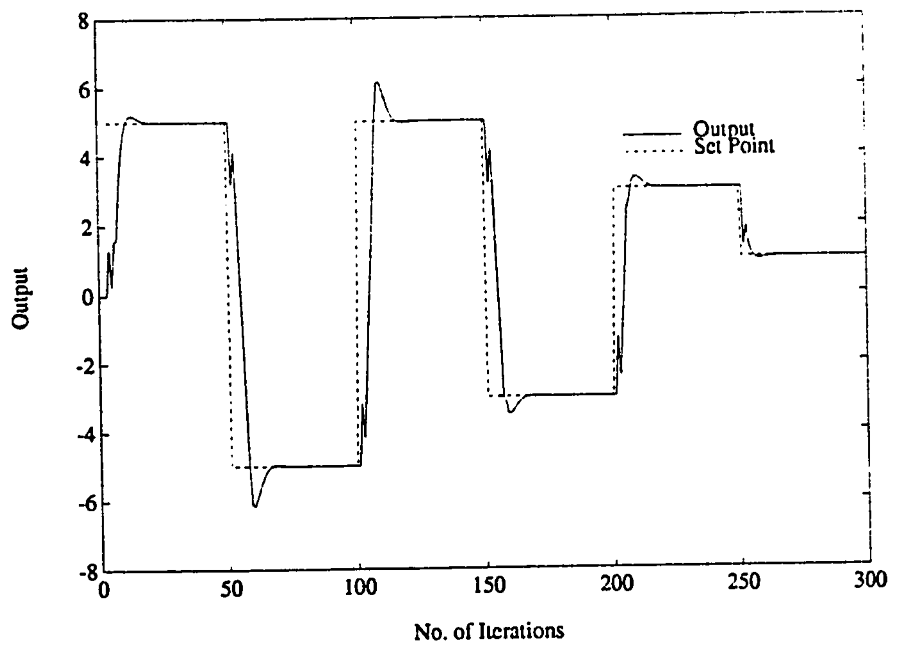


Figure 5.21: Control of the second order Weiner model using time varying linear controller and pole placement principle



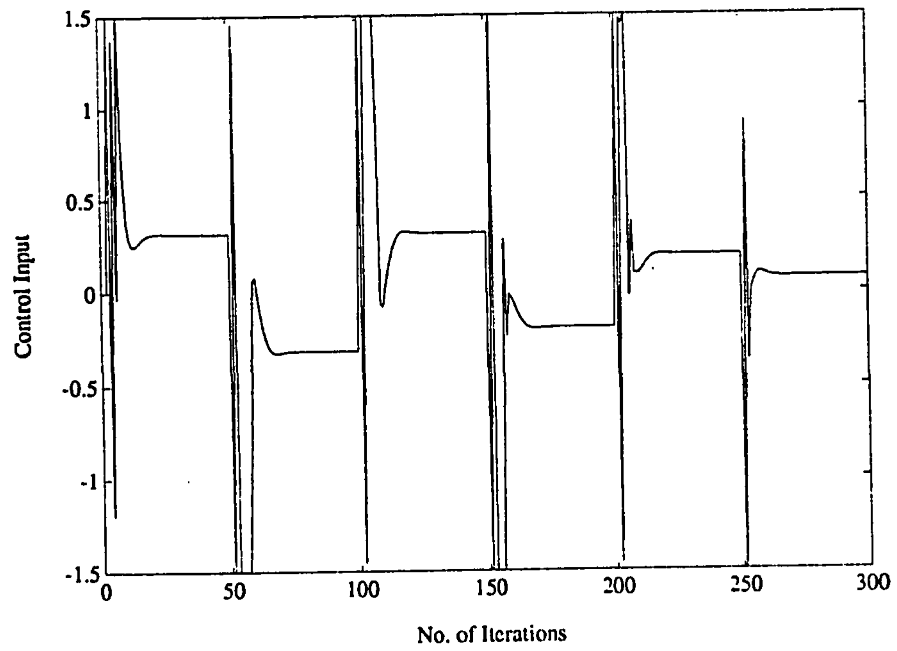
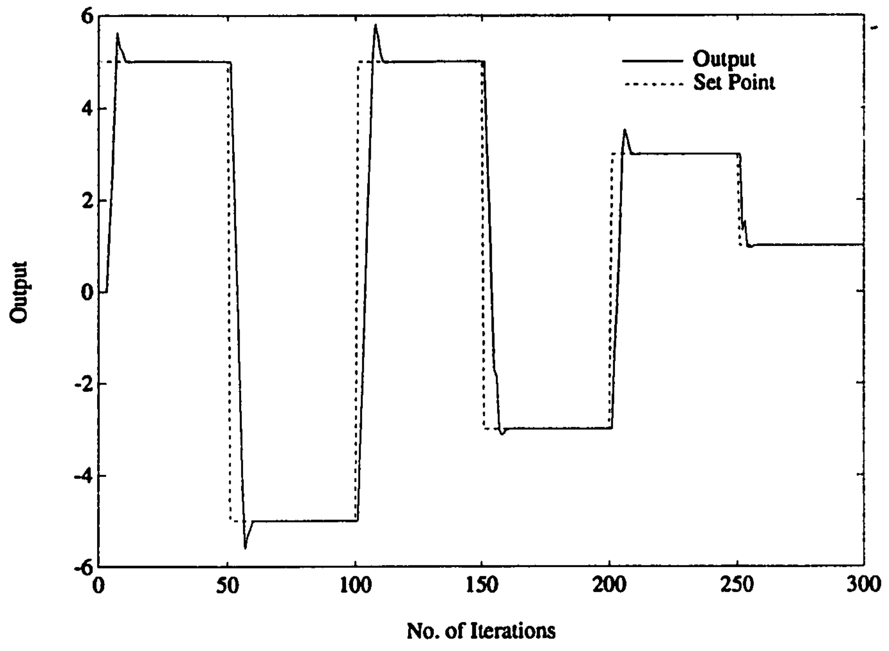


Figure 5.22: Control of the second order Weiner model using the neural net controller and pole placement principle



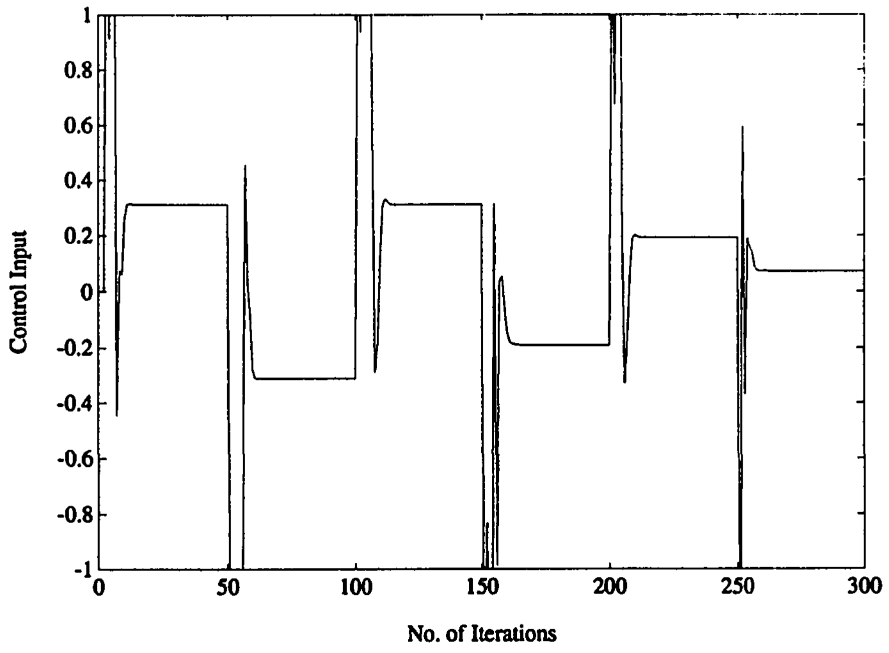
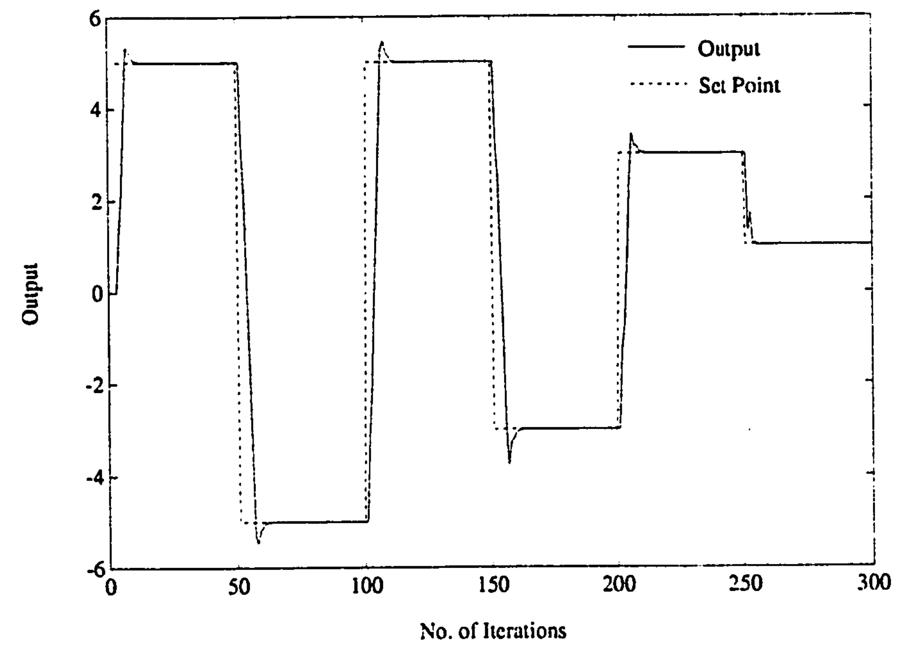


Figure 5.23: Control of the second order Weiner model using time varying linear controller and pole restriction principle



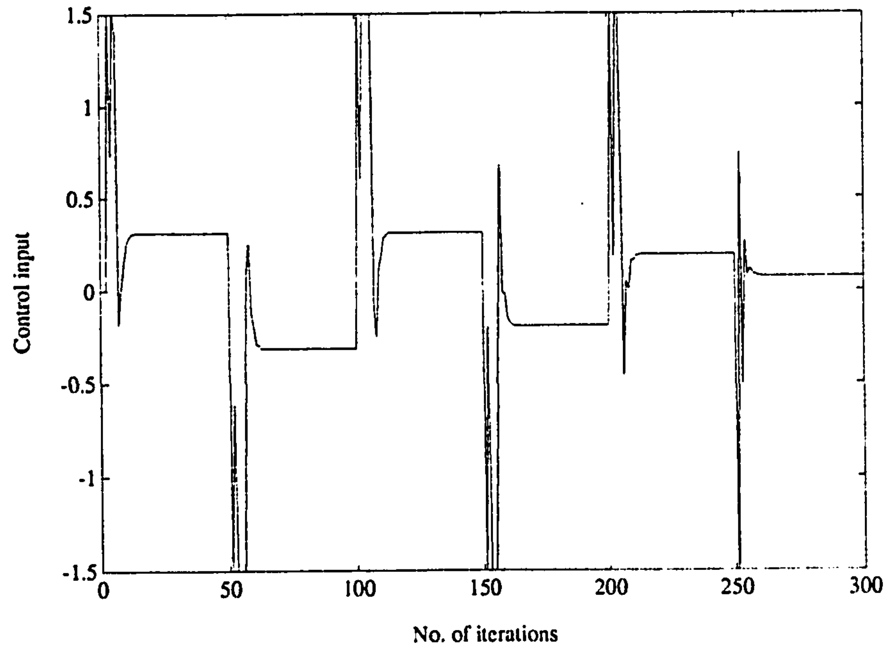


Figure 5.24: Control of the second order Weiner model using the neural net controller and pole restriction principle

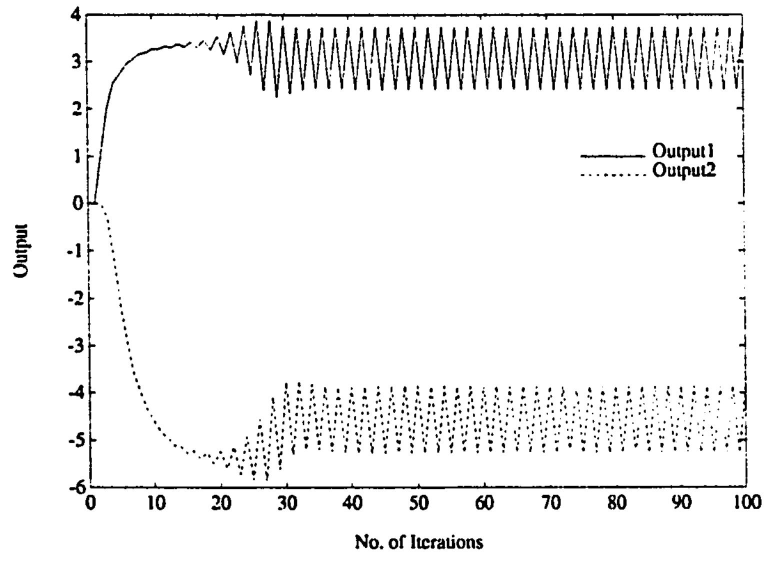


Figure 5.25: Response of the MIMO plant for input 1 as the unit step

plant be open loop stable and the continuous-time plant is originally unstable, a parameter has been modified to stabilize the plant. Figures 5.25 and 5.26 show the open loop response of the system for unit step input. The response when input 1 is a unit step and input 2 is zero is very oscillatory and shows limit cycles. A neural net consisting of one hidden layer with 7 neurons was trained to model the plant. Batch training was performed along with the matrix learning rate given in section 3.5.2. The learning parameters ϵ, $\mu(0)$, μ_r and μ_∞ in equation (3.26) were given the values 0.1, 0.1, 0.9 and 1.0 respectively.

The forward neural net model was used to compute the linearized model. Control of the plant based on this model and the linear time varying controller as shown in Figure 4.3 were implemented. Figures 5.27 and 5.28 show typical outputs and inputs when pole placement control is used. The closed loop poles were placed at

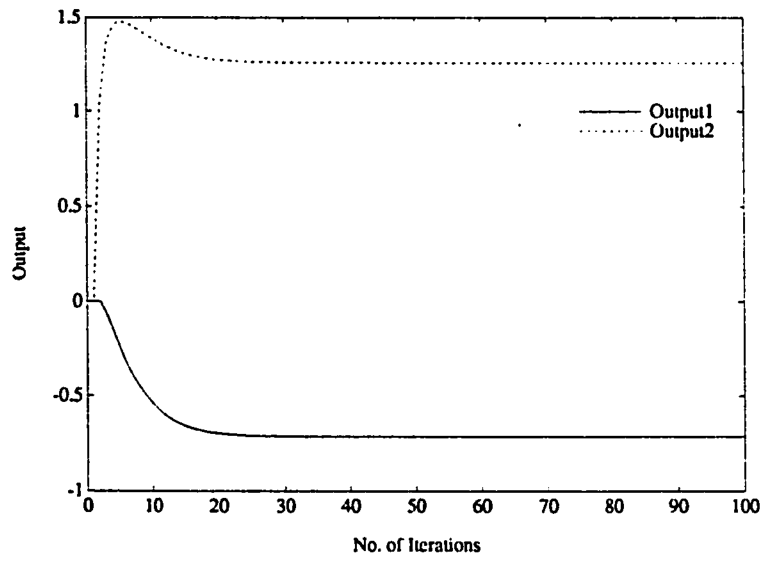
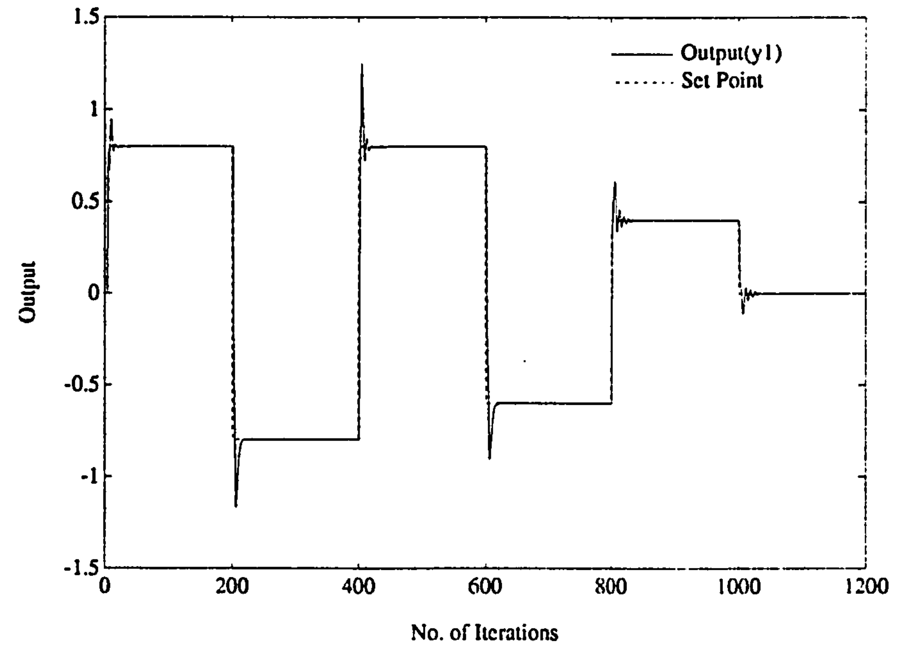


Figure 5.26: Response of the MIMO plant for input 2 as the unit step

0.7, 0.36 and 0. The controller parameters are calculated by solving the diophantine equation [39] at every time step. It can be observed from figures 5.27 and 5.28 that both satisfactory control and set point tracking are achieved.

In order to train a neural controller to replicate the time varying controller 6400 training data were generated. For this training data, the set point was varied between ± 1.1 in a random fashion. In stage 3, this data is used to train the neural controller. One hidden layer with 10 neurons was used to replicate the linear time varying controller. Due to the large size of the neural network, the training time is expected to be quite high. In order to reduce the training time and avoid local minima, few preliminary sweeps of pattern learning were done before using the matrix learning rate of section 3.5.2 with ϵ, $\mu(0)$, μ_r and μ_∞ being 0.1, 0.1, 0.9 and 1 respectively. But still, it took more than 72 hours of i486 PC time, for the neural



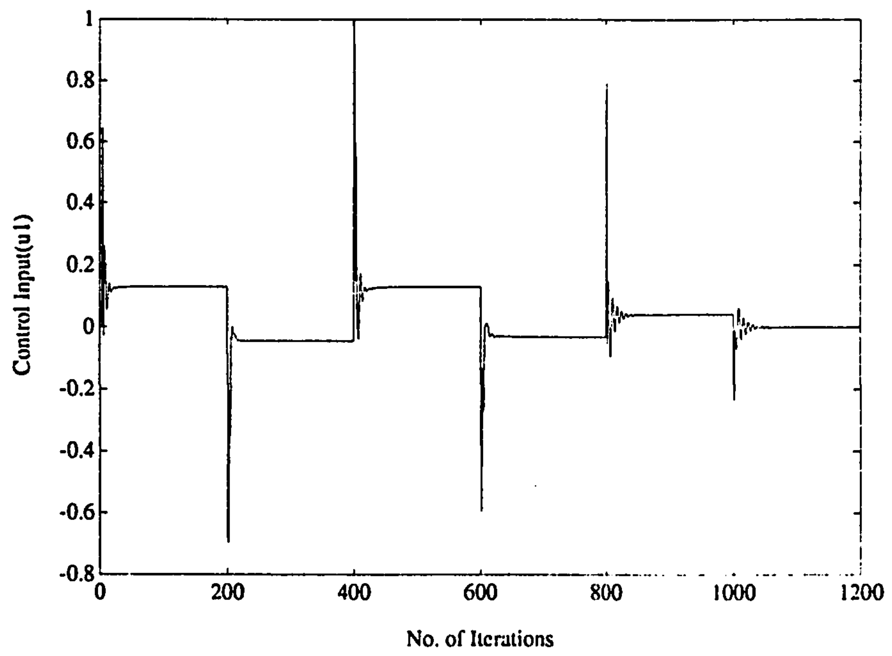
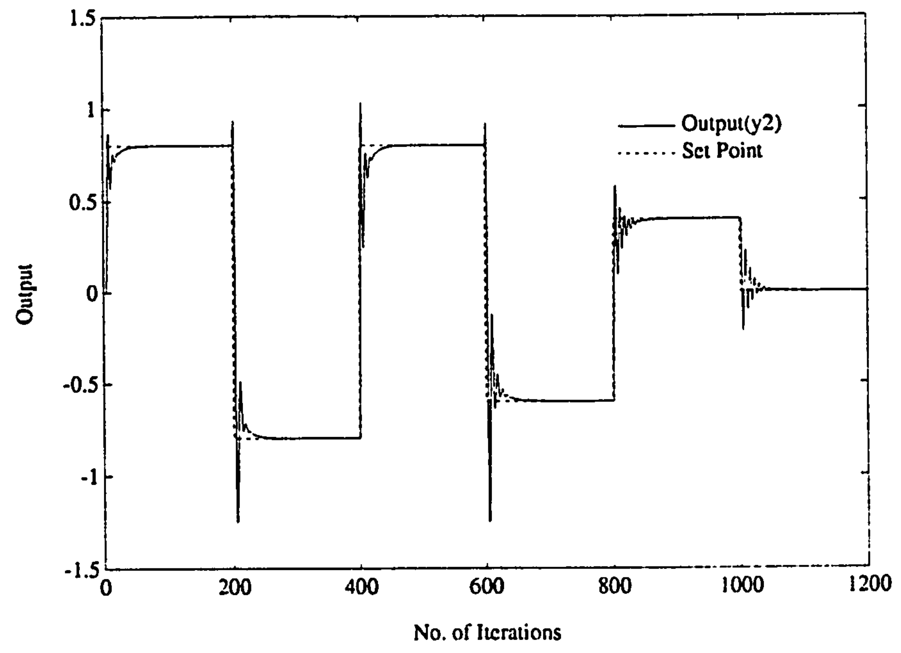


Figure 5.27: Control of a 2-input 2-output plant using time varying linear controller and pole placement principle (Output 1)



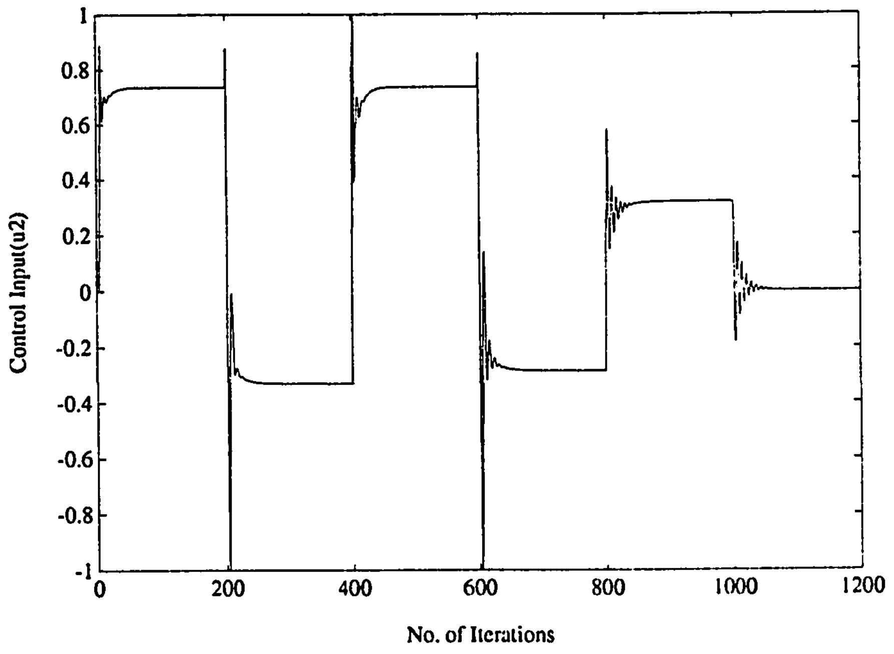


Figure 5.28: Control of a 2-input 2-output plant using time varying linear controller and pole placement principle (Output 2)

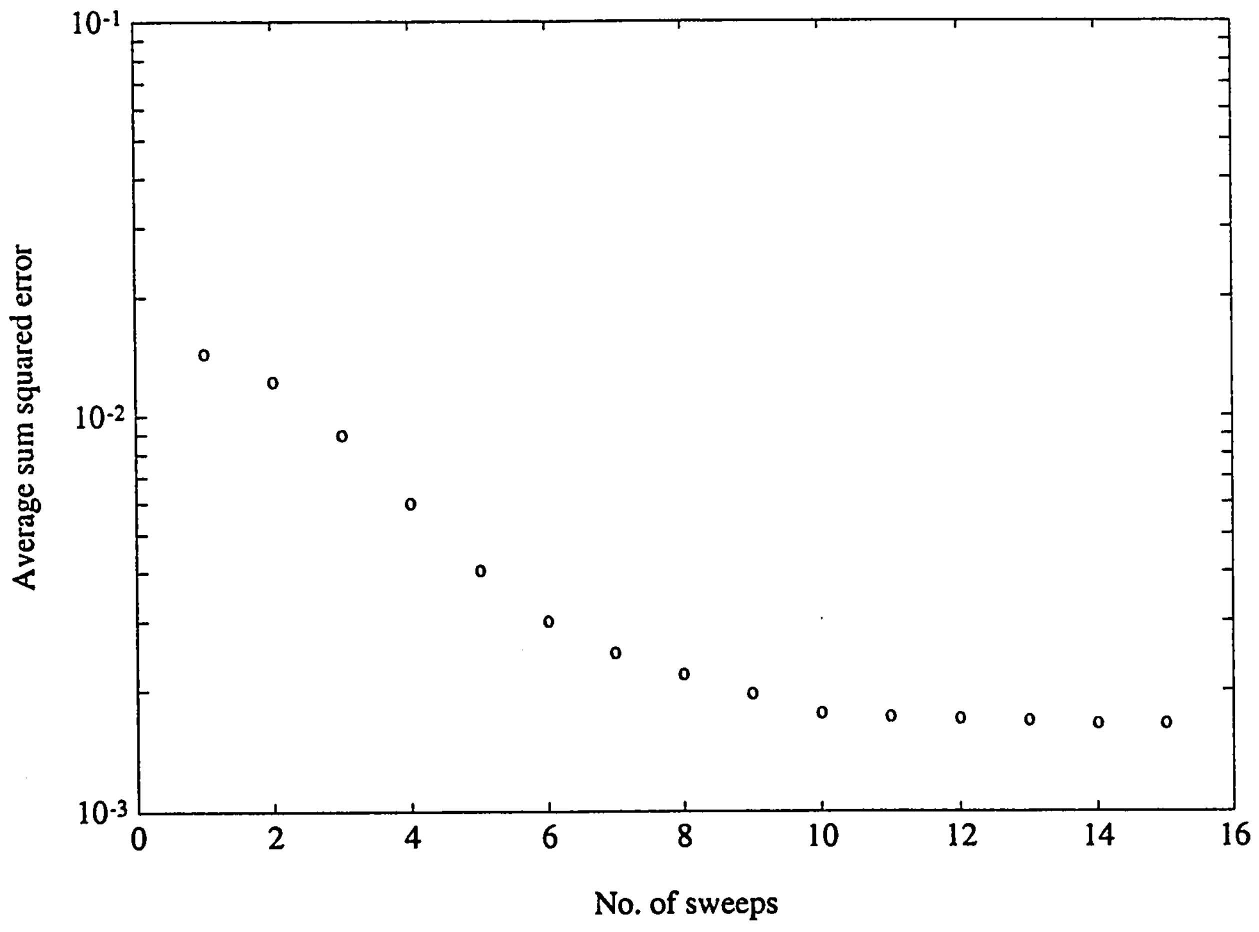
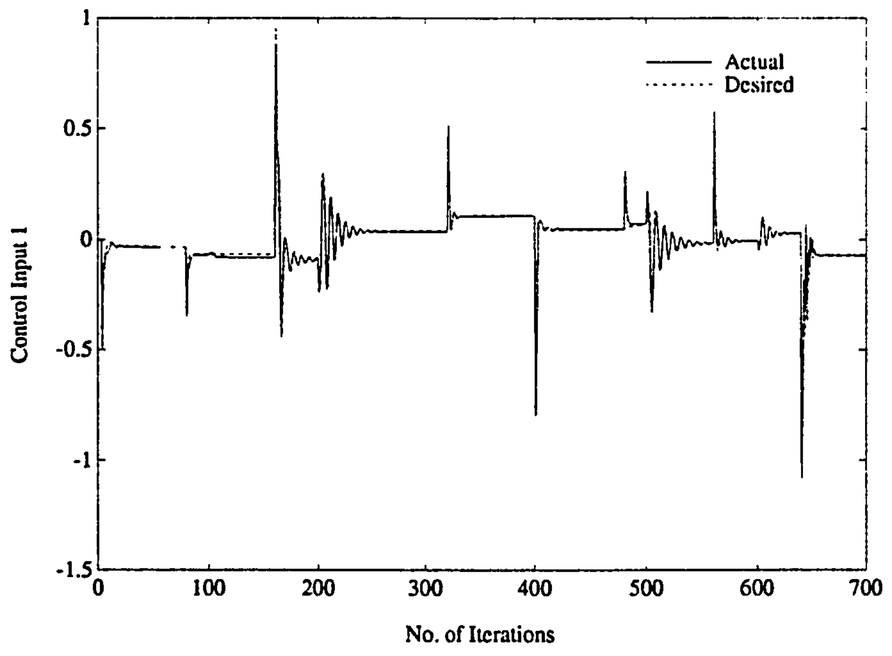


Figure 5.29: Average sum squared error during training of the neural controller

network to adequately capture the controller characteristics. Figure 5.29 shows the average sum squared error while Figure 5.30 shows the desired control inputs and neural net outputs after successful training. After successful training of the neural controller, the latter was used to control the plant through the configuration of Figure 4.9. Figures 5.31 and 5.32 show the plant outputs and inputs. A comparison of Figures 5.27, 5.31 and 5.28, 5.32 shows the capability of the neural controller in replicating the time varying controller behavior.



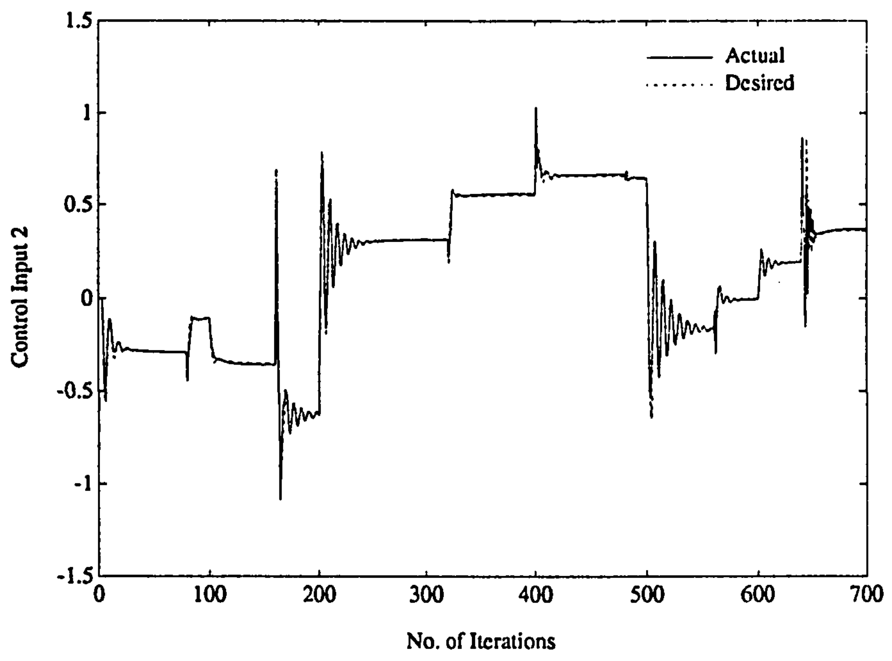
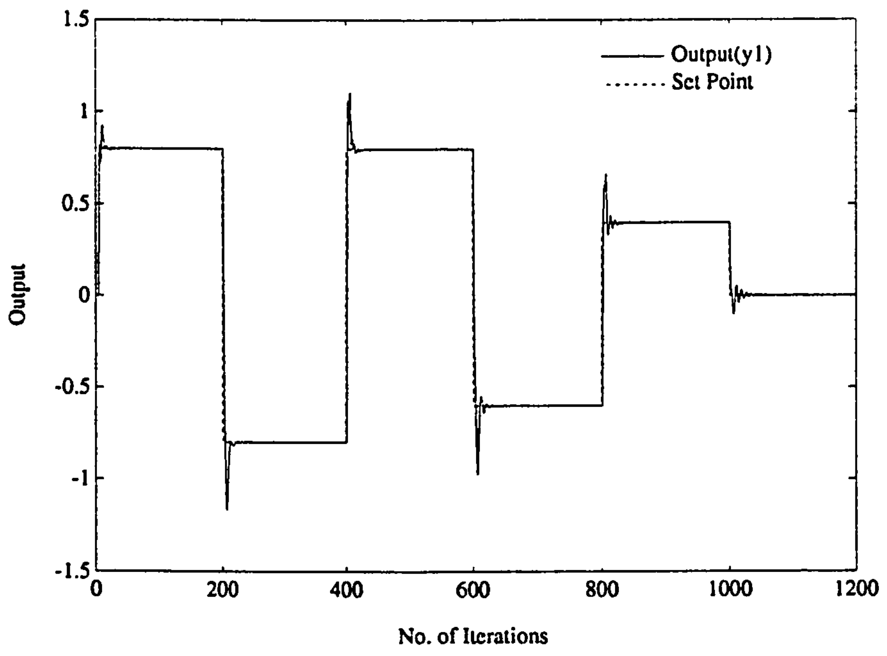


Figure 5.30: Desired control inputs and outputs of the neural net trained as the controller



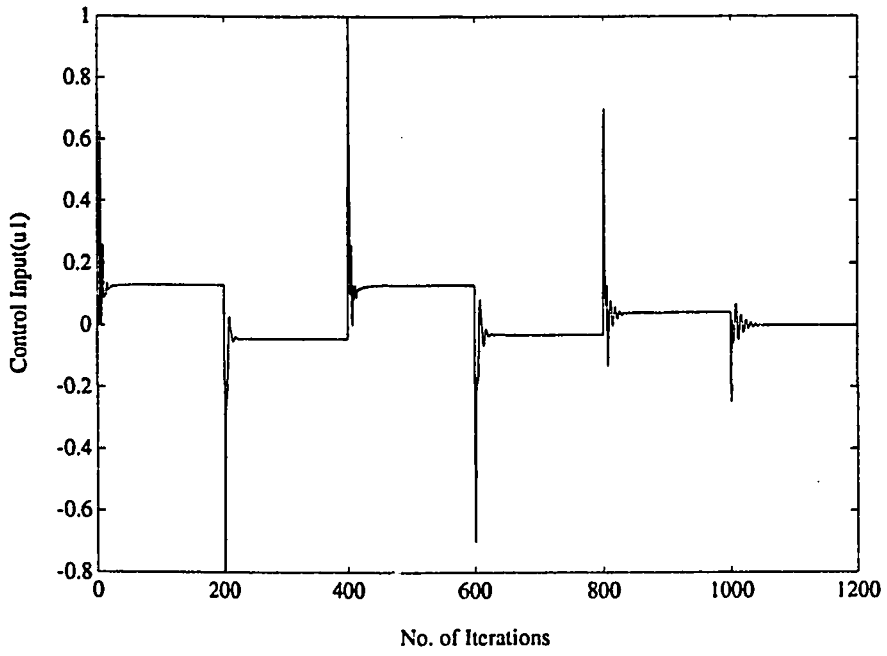
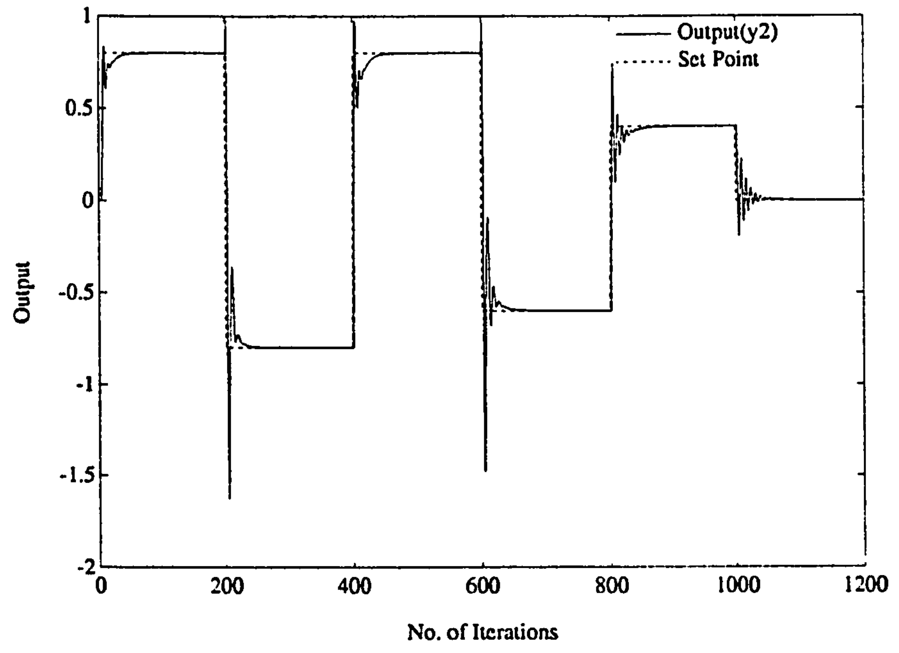


Figure 5.31: Control of a 2-input 2-output plant using neural net controller (Output 1)



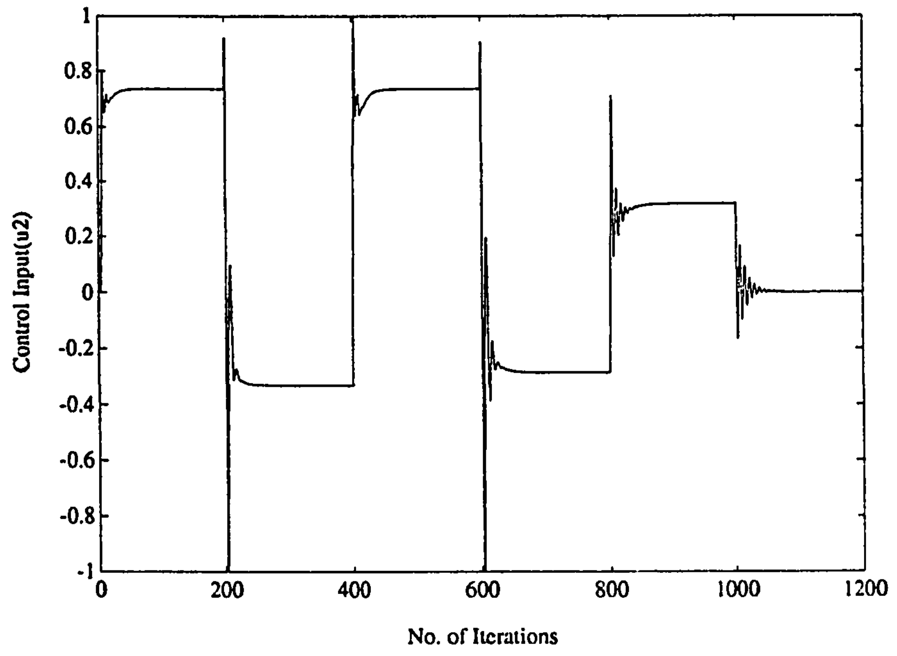


Figure 5.32: Control of a 2-input 2-output plant using neural net controller (Output 2)

Chapter 6

CONTRIBUTIONS, CONCLUSIONS AND RECOMMENDATIONS

In this work, we have concentrated upon the methods of utilizing neural networks to control nonlinear plants. The design principle is based upon successive linearization. Both adaptive control and fixed control have been considered. The strategy also works well for noisy plants. The control performance of SISO as well as MIMO plants using three control schemes, pole placement, quadratic optimal control and pole restriction were found to be satisfactory. The contributions, conclusions and recommendations for future works are described below.

6.1 Contributions

1. Linearization of nonlinear plant utilizing neural net model and an improved backpropagation algorithm was proposed.

2. Nonlinear controller design method using the neural network and employing the plant linearization strategy was proposed and studied.

3. Use of pole-placement, pole restriction and quadratic optimal control strategies in the design of time varying linear controller were studied. Both adaptive and fixed control strategies were developed.

4. A specialized neural net structure was proposed and implemented to act as the controller in order to replace the linear time varying controller.

5. A new scheme was proposed for the training of large neural networks, that avoids local minima and as well provide small learning error by applying two different kinds of learning rate.

6. Simulation studies were carried out to establish the feasibility of the proposed approaches. The plants considered were both of the SISO and MIMO types. Further, both deterministic as well as noisy plants were considered.

6.2 Conclusions

Based on our limited simulation study, the following may be concluded.

1. Neural networks can be effectively used to obtain linearization of nonlinear plants.

2. Techniques from linear systems theory can be effectively utilized to control the nonlinear plants.

3. Nonlinear plant of unknown structure can be effectively described by a dynamic neural network. Further, a state dependent time-varying controller can be represented arbitrarily well by Neural Networks.

4. In the proposed control strategy, pole-placement and pole restriction principles provided better control compared to optimal control design.

5. The proposed strategy is applicable to SISO as well as MIMO nonlinear plants, and deterministic as well as noisy plants.

6.3 Recommendations

1. Other control strategies in linear control theory such H_∞, IMC and predictive control can also be accommodated in the proposed strategy. They will however significantly increase the computational requirement.

2. Instead of designing the controller through linearization, the neural controller can be directly trained to optimize some desired performance criterion. Such approach may provide faster and less oscillatory system response.

3. A few neural net chips are becoming available in the market. The proposed algorithms can be applied for controlling real-time nonlinear plants employing the neural net chips.

4. The time-varying linearization approach proposed here can also be extended to other fields for example, nonlinear signal processing.

Appendix A

In this appendix expressions for the scalar pattern (on-line) learning rates for the noisy case are derived. When these expressions are used in conjunction with the back propagation algorithm given by (3.28), the training algorithm becomes considerably simpler compared to the RPE algorithm.

Consider minimization of a modified criterion function

$$J(w,t) = \sum_{s=1}^{t} \lambda^{t-s} \|\epsilon(w,s)\|^2 = \sum_{s=1}^{t} \lambda^{t-s} \|y(s) - \hat{y}(s)\|^2 \tag{A.1}$$

where $\hat{y}(t)$ is the predicted neural network output. λ (≤ 1) is known as a forgetting factor the purpose of which is to gradually discount the effect of past data. The batch updating algorithm developed in the last section with this modified criterion function becomes (see (3.31), (3.33) and (3.35))

$$w(t) \quad = \quad w(0) + \eta_b(t)F(t) \tag{A.2}$$

$$\text{where} \quad F(t) = \sum_{s=1}^{t} \lambda^{t-s} \Gamma_b(s) e_b(s) \quad , \quad \eta_p^{-1}(t) = \sum_{s=1}^{t} \lambda^{t-s} \|\Gamma_b(s)\|^2 \tag{A.3}$$

$$\Gamma_b(s) \equiv \left[\frac{d\hat{y}(s)}{dw(0)}\right]^T \quad \text{and} \quad e_b(s) \equiv y(s) - \hat{y}(w(0), s) \tag{A.4}$$

The argument t in w indicates the use of data for $s = 1, \ldots, t$. μ is set to 1 as it's effect can be absorbed in the choice of λ. It can also be verified that

$$F(t) = \lambda F(t-1) + \Gamma_b(t)e_b(t) \tag{A.5}$$

$$\text{and} \quad \eta_p^{-1} = \lambda \eta_p^{-1}(t-1) + \|\Gamma_b(t)\|^2 \tag{A.6}$$

Using (A.2), (A.5) and (A.6), subtracting $w(t-1)$ from $w(t)$ one gets

$$w(t) = w(t-1) + \left\{\eta_p^{-1}(t-1)\Gamma_b(t)e_b(t) - \|\Gamma_b(t)\|^2 F(t-1)\right\}\eta_p(t)\eta_p(t-1) \tag{A.7}$$

Now define (and also compare with (A.3) and (A.4))

$$\Gamma_p(t) \equiv \left[\frac{d\hat{y}(t)}{dw(t-1)}\right]^T \quad \text{and} \quad e_p(t) \equiv y(t) - \hat{y}(w(t-1),t) \tag{A.8}$$

and use a first order approximation to $\hat{y}(t)$ in order to obtain

$$\Gamma_b(t) = \Gamma_p(t) \tag{A.9}$$

$$\text{and} \quad e_b(t) = e_p(t) - \Gamma_p^T(t)[w(0) - w(t-1)] \tag{A.10}$$

$$= e_p(t) + \eta_b(t-1)\Gamma_p^T(t)F(t-1) \tag{A.11}$$

The last expression follows from (A.2). Using (A.9) and (A.10) in (A.7) one gets

$$w(t) = w(t-1) + \eta_p(t)\Gamma_p(t)e_p(t) - \left[\Gamma_p^T(t)\Gamma_p(t)I - \Gamma_p(t)\Gamma_p^T(t)\right]\eta_p(t)\eta_p(t-1)F(t-1) \tag{A.12}$$

The last term in the right hand side makes the algorithm computationally some what unattractive. Since the bracketed term is at least positive semidefinite, dropping this term is equivalent to put less weight on the current data, which may well be compensated through proper choice of the weighting factor λ. Our simulation experiment confirmed this conjecture [36]. Thus ignoring the last term in (A.12), and using (A.6) and (A.9) one gets the pattern learning algorithm as

$$w(t) = w(t-1) + \eta_p(t)\Gamma_p(t)e_p(t) \tag{A.13}$$

$$\text{and} \quad \eta_p^{-1} = \lambda \eta_p^{-1}(t-1) + \|\Gamma_p(t)\|^2$$

$\eta_p(0)$ can be taken as a small positive quantity (≈ 0.1). Suitable value of λ lies between 0.9 and 0.99. $\square$

Appendix B

This appendix contains the following lemma which has been used in deriving (3.33).

Lemma: Let x be an arbitrary vector and A be a real symmetric positive definite matrix of compatible dimension such that

$$0 \leq \lambda_{min} \leq \frac{x^T A x}{x^T x} \leq \lambda_{max} \qquad for\ x \neq 0 \tag{B.1}$$

$$then \qquad \lambda_{min} \leq \frac{x^T A^2 x}{x^T A x} \leq \lambda_{max} \tag{B.2}$$

Proof: Since A is symmetric and real it can be written in terms of its diagonal form as

$$A = C^T \Lambda C \tag{B.3}$$

where Λ is a diagonal matrix with all eigenvalues of A as diagonal elements and C is a real orthogonal matrix. Thus

$$\frac{x^T A^2 x}{x^T A x} = \frac{x^T C^T \Lambda^2 C x}{x^T C^T \Lambda C x} = \frac{y^T \Lambda^2 y}{y^T \Lambda y} = \frac{\sum \lambda_i^2 y_i^2}{\sum \lambda_i y_i^2} \tag{B.4}$$

where $\lambda_i's$ are the eigenvalues of Λ which are real and positive. Now (B.2) directly follows from (B.4). □

Bibliography

[1] Franklin Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems.* Addison-Wesley Publishing Company.

[2] Narendra K. S. and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[3] Hunt K. J. et al. Neural networks for control systems–a survey. *Automatica*, 28(6):1083–1112, 1992.

[4] Wasserman P. *Neural Computing.* Van Nostrand Reinhold, New York, 1988.

[5] Page G. F., J. B. Gomm, and D. Williams. *Application of neural networks to modelling and control.* Chapman and Hall, London, 1993.

[6] Antsaklis Panos J. Neural networks in control systems. *IEEE Control Systems Magazine*, 12(2):8–10, 1992.

[7] Fukuda Toshio and Takanori Shibata. Theory and applications of neural networks for industrial control systems. *IEEE Transactions on Industrial Electronics*, 39(6):472–489, 1992.

[8] Tai Heng-Ming, Juli Wang, and Kaveh Ashenayi. A neural network based tracking control system. *IEEE Transactions on Industrial Electronics*, 39(6):504–510, 1982.

[9] Kuperstein M. and J. Rubinstein. Implementation of an adaptive neural controller for sensory motor coordination. *IEEE Control Systems Magazine*, 9(3):25–30, 1989.

[10] Anderson C. W. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine,*, 9(3):31–37, 1989.

[11] Hunt K. J. and D. Sbarbaro. Neural networks for nonlinear internal model control. *Proc. IEE Pt. D*, 138:431–438, 1991.

[12] Keerthi S. S. and E. G. Gilbert. Moving horizon approximations for a general class of optimal nonlinear infinite-horizon discrete-time systems. *Proc. 20th Annual Conf. Information Science and Systems, Princeton University*, pages 301–306, 1986.

[13] Willis M. J., G. A. Montague, C. Di Massimo, M. T. Tham, and A. J. Morris. Arificial neural networks in process estimation and control. *Automatica*, 28(6):1181–1187, 1992.

[14] Psaltis et al. A multilayered neural network controller. *IEEE Control Systems Magazine*, pages 17–21, 1988.

[15] Miller W. T., R. S. Sutton, and P. J. Werbos. *Neural Networks for Control.* MIT Press, Cambridge, MA., 1990.

[16] Zak S. H. Robust tracking control of dynamic systems with neural networks. *IEEE Int. Joint Conf. on Neural Networks, IJCNN'90*, pages 563–566, 1990.

[17] Chi S. R., R. Shoureshi, and M. Tenorio. Neural networks for system identification. *IEEE Control Systems Magazine* ,, 10:31–34, 1990.

[18] Demircioglu H. and P. J. Gawthrop. 'continuous-time relay self-tuning control. *Int. J. Control*, 47:1061–1080, 1988.

[19] Li Weiping and Jean-Jacques E. Slotine. Neural network control of unknown nonlinear systems. *Proc. American Control Conference, Pittsburgh, PA.*, pages 1137–1141, 1989.

[20] Chen F. C. Backpropagation neural networks for nonlinear self-tuning adaptive control. *IEEE Control systems magazine*, 10(2):44–48, 1990.

[21] Gupta M. M. and D. H. Rao. Dynamic neural units with applications to the control of unknown nonlinear systems. *Journal of Intelligent and Fuzzy Systems,*, 1(1):73–92, 1993.

[22] Zhu Q. M., Prof. K. Warwick, and Prof. J. L. Douce. Adaptive general predictive controller for nonlinear systems. *IEE Proceedings, D.*, 138(1):33–40, 1991.

[23] Tanomaru J. and S. Omatu. Process control by on-line trained neural controllers. *IEEE Transactions on Industrial Electronics*, 39(6):511–521, 1992.

[24] Watanabe K., T. Fukuda, and S. G. Tzafestas. An adaptive control for carma systems using linear neural networks. *International Journal of Control*, 56(2):483–497, 1992.

[25] Kawato M., K. Furukawa, and R. Suzuki. A hierarchical neural-network model for control and voluntary movement. *Biological Cybernetics*, 57:169-185, 1987.

[26] Miyamoto H., M. Kawato, T. Setoyama, and R. Suzuki. Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks*, 1:251-265, 1988.

[27] Gomi H. and M. Kawato. Neural network control for a closed-loop system using feedback-error-learning. *Neural Networks*, 6:933-946, 1993.

[28] Willis M. J., C. Di Massimo, G. A. Montague, M. T. Tham, and A. J. Morris. Arificial neural networks in process engineering. *IEE Proceedings-D*, 138(3):256-266, 1991.

[29] Parthasarathy Sanjay, Alexander G. Parlos, and Amir F. Atiya. Direct adaptive control of process systems. *Proceedings of 1992 American Control Conference*, pages 63-65, 1992.

[30] Chen Fu-Chuang and Chen-Chung Liu. Adapively controlling nonlinear continuous-time systems using neural networks. *Proceedings of 1992 American Control Conference*, pages 46-50, 1992.

[31] Jin L., P. N. Nikiforuk, and M. M. Gupta. Adaptive tracking of siso nonlinear systems using multilayered neural networks. *Proceedings of 1992 American Control Conference*, pages 56-60, 1992.

[32] Werbos P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339-356, 1988.

[33] Rumelhart D. E., G. E. Hinton, and R. J. Williams. Learning internal representation by back propagation. *Parallel Distributed Processing: exploration in the microstructure of cognition*, 1, 1986.

[34] Ahmed M. S. Neural net based identification of dynamic plants with variable learning rates. *submitted to IEEE Transactions on Industrial Electronics*, 1993.

[35] Ahmed M. S. Improved backpropagation algorithms for faster convergence. *submitted to IEEE Transactions on Neural Networks*, 1993.

[36] Ahmed M. S. Neural net identification of noisy dynamic plants. *submitted to Automatica*, 1993.

[37] Webros P. J. Backpropagation through time: what it does and how to do it. *Proc. of IEEE*, 78:1550–1560.

[38] Åström Karl J. and Björn Wittenmark. *Adaptive Control.* Addison-Wesley Publishing Company, 1989.

[39] Åström Karl J. and Björn Wittenmark. *Computer-controlled systems.* Addison-Wesley Publishing Company, 1990.

[40] Ahmed M. S. State space based adaptive control with pole restriction. *IEE Proceedings, D., to appear*, 1993.

[41] Chen Chi-Tsong. *Linear System Theory and Design.* Saunders College Publishing, Harcourt Brace Jovanovich College Publishers, Florida, 1984.

[42] Qin Si-Zhao, Hong-Te Su, and Thomas J. McAvoy. Comparison of four neural net learning methods for dynamic system identification. *IEEE Transactions on Neural Networks*, 3(1):122–130, 1992.

[43] Yaniv Oded. Robust design of mimo feedback systems having an uncertain nonlinear plant. *International Journal of Control*, 53(6):1283–1294, 1991.