NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

BACK-PROPAGATION NEURAL NETWORKS
IN ADAPTIVE CONTROL OF
UNKNOWN NONLINEAR SYSTEMS

by

Chin Hock Teo

December, 1991

Thesis Advisor: Roberto Cristi
Co-Advisor Ralph Hippenstiel

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION / AVAILABILITY OF REPORT Distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) EC	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
BACK-PROPAGATION NEURAL NETWORKS IN THE ADAPTIVE CONTROL OF UNKNOWN NONLINEAR SYSTEMS

12. PERSONAL AUTHOR(S)
Teo, Chin-Hock

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	1991, December	114

16 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
Back-propagation neural network, direct model reference control system, nonlinear systems, discrete-time models.

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The objective of this research is to develop a Back-propagation Neural Network (BNN) to control certain classes of unknown nonlinear systems and explore the network's capabilities. The structure of the Direct Model Reference Adaptive Controller (DMRAC) for Linear Time Invariant (LTI) systems with unknown parameters is first analyzed. This structure is then extended using a BNN for adaptive control of unknown nonlinear systems. The specific structure of the BNN DMRAC is developed for the control of four general classes of nonlinear systems modelled in discrete time. Experiments are conducted by placing a representative system from each class under the BNN's control. The conditions under which the BNN DMRAC can successfully control these systems are investigated. The design and training of the BNN are also studied.

The results of the experiments show that the BNN DMRAC works for the representative systems considered, while the conventional least-squares estimator DMRAC fails. Based on analysis and experimental findings, some general conditions required to ensure that this technique works are postulated and discussed. General guidelines used to achieve the stability of the BNN learning process and good learning convergence are also discussed.

To establish this as a general and significant control technique, further research is required to establish analytically, the conditions for stability of the controlled system, and to develop more specific rules and guidelines in the BNN design and training.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL R. Cristi	22b. TELEPHONE (*Include Area Code*) 408-646-2223 22c. OFFICE SYMBOL CODE ECCX

DD Form 1473, JUN 86 *Previous editions are obsolete.*
S/N 0102-LF-014-6603

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Approved for public release; distribution is unlimited.

Back-propagation Neural Networks
in Adaptive Control of
Unknown Nonlinear Systems

by

Chin Hock Teo
Major, Republic of Singapore Air Force
B.Eng.(Hon), National University of Singapore, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1991

Department of Electrical and Computer Engineering

ABSTRACT

The objective of this research is to develop a Back-propagation Neural Network (BNN) to control certain classes of unknown nonlinear systems and explore the network's capabilities. The structure of the Direct Model Reference Adaptive Controller (DMRAC) for Linear Time Invariant (LTI) systems with unknown parameters is first analyzed. This structure is then extended using a BNN for adaptive control of unknown nonlinear systems. The specific structure of the BNN DMRAC is developed for the control of four general classes of nonlinear systems modelled in discrete time. Experiments are conducted by placing a representative system from each class under the BNN's control. The conditions under which the BNN DMRAC can successfully control these systems are investigated. The design and training of the BNN are also studied.

The results of the experiments show that the BNN DMRAC works for the representative systems considered, while the conventional least-squares estimator DMRAC fails. Based on analysis and experimental findings, some general conditions required to ensure that this technique works are postulated and discussed. General guidelines used to achieve the stability of the BNN learning process and good learning convergence are also discussed.

To establish this as a general and significant control technique, further research is required to obtain analytically, the conditions for stability of the controlled system, and to develop more specific rules and guidelines in the BNN design and training.

TABLE OF CONTENTS

I. INTRODUCTION

A. OBJECTIVE

The objective of this thesis research is to develop a Back-propagation Neural Network (BNN) to control certain classes of unknown, nonlinear dynamical systems and to explore the network's capabilities. Discrete-time models, which readily describe many real world systems, are used to represent the unknown nonlinear systems for the purpose of analysis and simulation.

B. NEURAL NETWORKS IN ADAPTIVE CONTROL

Linear control theory is a very mature field. Since the beginning of this century, both necessary and sufficient conditions for the stability of Linear-Time-Invariant (LTI) systems have been established and rigorously proven. As the result, many powerful and well-established techniques (e.g. state-feedback) have been developed to design controllers for LTI systems which will achieve any desired system response or any specified robustness. In contrast, the conditions for stability of most nonlinear and time-varying systems can only be established, if at all possible, on a system-by-system basis.

Hence, general control design techniques, even just to achieve stability, are still not available for many classes of nonlinear systems.

From the fifties up to the late seventies, major advances were made in the identification and adaptive control of LTI systems with unknown or time varying dynamics [Ref.1]. Many adaptive control techniques, for which global stability is assured, have been developed by assuming the system to be LTI, and applying well-established results from linear systems theory and parameter estimation. However, the original targets of these techniques were actually systems with slowly varying parameters. These systems belong to a significant class of nonlinear systems. The controllers are also nonlinear systems by themselves. Nonetheless, limited advances have been made to address the adaptive control of more general classes of nonlinear systems.

Recently, the use of neural networks for parametric identification and adaptive control of certain general classes of nonlinear systems, based on the indirect adaptive control structure, has been suggested [Ref.2]. In that approach, a neural network is first trained to emulate an unknown, nonlinear Single-Input-Single-Output (SISO) system. Then the errors between the system output and a desired reference model output are back-propagated through the trained neural network emulator to obtain the contributing control input error. Based on a suitable minimization function of the control input error, a neural network controller is trained to control the system so that it behaves like the desired reference model. Simulation results have shown that neural network-based indirect adaptive control of large classes of nonlinear systems is not only feasible, but seems quite promising as a general technique.

The stated objective of this thesis research is to further explore, analyze and develop the neural network-based adaptive controller. Specifically, the use of neural networks as a direct adaptive controller for some general classes of nonlinear systems shall be considered. Unlike the indirect adaptive control approach, only one neural network, instead of two, shall be used to learn the unknown control structure and parameters directly. The same neural network estimator shall then be used as the controller. The development of a neural network estimator-controller is the key issue addressed in this thesis.

C. BASIC CONCEPTS OF ADAPTIVE CONTROL

Adaptive control has been applied to many areas such as robot manipulation, ship steering, aircraft control, chemical process control and bio-medical engineering. The applications are mainly aimed at handling parameter variations (slowly time-varying) and parameter uncertainties in the system under control.

In adaptive control, the basic idea is to combine an on-line parameter identification process with control system design calculation based on the estimated parameters and the required control law to implement the controller. The general structure of an adaptive control system is shown in Figure 1.

Consider the adaptive control of an unknown linear time invariant (LTI) system. One scheme is to parameterize the system, for example, by a linear state-space model

$\{A,B,C,D\}$ or a transfer function $H(\cdot)$ with unknown parameters. These parameters are then estimated on-line by a suitable estimator. Based on the estimated parameters,

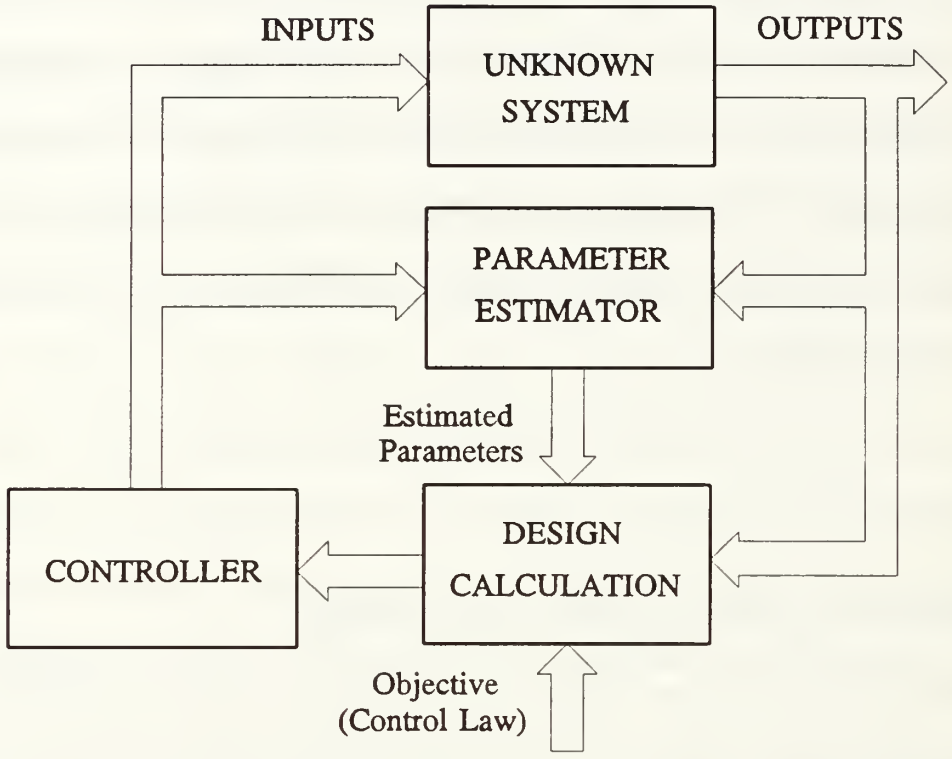

Figure 1. General Adaptive Control Structure.

appropriate design calculations can be performed on-line to implement the chosen control law. This class of algorithms is commonly referred to as indirect adaptive control. Figure 2 shows the structure of an indirect adaptive control system.

Alternatively, it may be possible to parameterize the unknown system directly in terms of the required control parameters (e.g. the state-feedback gains) to implement the chosen control law. In this case, the on-line estimator would generate the estimates of the unknown control parameters, and then uses them directly for the control. The need for design calculation on-line is therefore eliminated. This class of algorithms is called

direct adaptive control. The structure of a direct adaptive control system is shown in Figure 3.

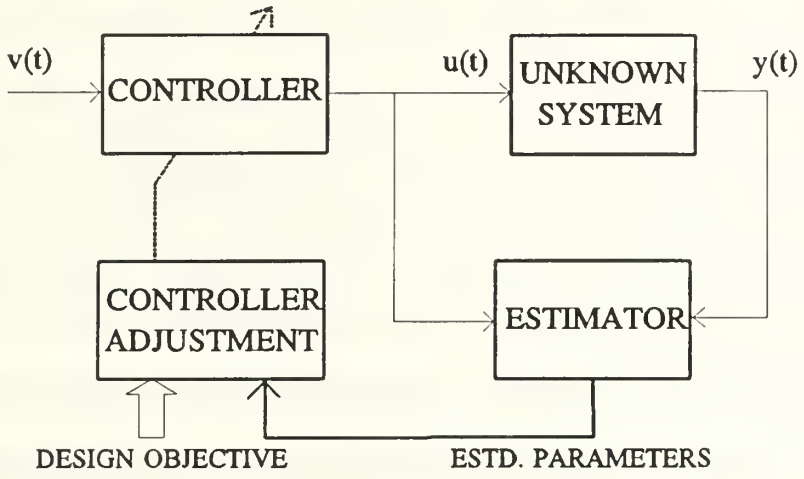

Figure 2. Indirect Adaptive Control Algorithm.

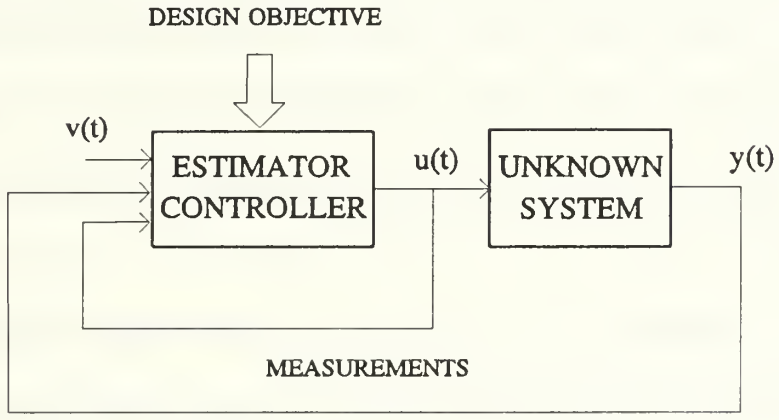

Figure 3. Direct Adaptive Control Algorithm.

Many different methods have also been used to specify the desired behavior or performance of a system under adaptive control. One very common scheme is the model reference adaptive control. The basic idea is to design the adaptive control system (be

it direct or indirect) so that the closed loop system behaves like the specified reference model.

We see that a key component of an adaptive controller is the parameter estimator. Many parameter estimation schemes have been devised and employed in adaptive control. However, it is important to note that most existing techniques generally require a linear parameterization of the system, i.e., parametric uncertainties must be expressed linearly in terms of a set of unknown parameters. Such parameterization of the system is usually in a form of a regression equation which is linear in the parameters. In linear systems, the regressor can usually be formed using only linear functions of the state measurements or observations from the systems, with the unknown parameters as coefficients. However, in nonlinear systems, nonlinear functions of the measurements or observations are generally required. Hence, to use current estimation techniques requires that these nonlinear functions are known. However, with unknown nonlinear systems, this will not be the case. Hence, the use of neural network as a generalized estimator is proposed in such a situation.

In order to develop a neural network-based direct model reference adaptive controller (DMRAC) for certain classes of unknown, nonlinear systems, the design of a DMRAC for unknown LTI systems shall be first reviewed and analyzed in detail. Based on the same control structure, the neural network shall be employed to extend the control to nonlinear systems.

D. ANALYSIS OF A DMRAC FOR UNKNOWN LTI SYSTEMS

Consider an LTI system described by an ARMA model,

$$A(q)y(t) = B(q)u(t) , \tag{1-1}$$

with $A(q)$ and $B(q)$ being polynomial operators[1] with unknown coefficients. $A(q)$ is assumed, without loss of generality, to be monic and degree$[A(q)] = n \geq$ degree$[B(q)] = m$. For the direct MRAC design, the following assumptions are required:

1. The upper bound on the system order (i.e. maximum degree of $A(q)$, n) is known.
2. The system has no hidden unstable modes and has a stable inverse.
3. The relative degree of $A(q)$ and $B(q)$ (i.e. n - m) is known.

The design objective is for the closed loop system to track a reference model

$$D(q)y(t) = v(t) , \tag{1-2}$$

where $D(q)$ is the monic characteristic polynomial operator of the desired system, and $v(t)$, an external input. Let the degree of $D(q)$ be r. It is well known that with linear state-feedback, all n poles of the closed loop system can be placed anywhere in the complex plane (provided the system is controllable). Hence to achieve model tracking by state-feedback, r out of the n poles of the closed loop system must be placed to match those of the reference model. The m unwanted zeros of the open loop system must also

[1] The argument q of the polynomials can be interpreted as the forward time-shift operator in discrete-time modelling or as the Laplace s-operator in continuous-time modelling.

be canceled by the remaining poles of the closed loop system. Hence r must be equal to $(n - m)$. For proper pole-zero cancellation, a stable inverse system is also required.

Very often, only $u(t)$ and $y(t)$ are accessible while the other states required for full-state feedback are not. Hence, in these cases, an observer is required. By employing a Luenberger observer or a steady-state Kalman filter, it can be shown that the combined observer-state-feedback system yields the following structure for the feedback controller,

$$u(t) = \frac{h(q)}{\alpha(q)}u(t) + \frac{k(q)}{\alpha(q)}y(t) + v(t) , \qquad (1\text{-}3)$$

where $\alpha(q)$ is the monic characteristic polynomial operator of the observer. It can be chosen arbitrarily, provided it is a stable system of degree n. Hence, degree$[A(q)] = n$ must be known. The polynomial operators $h(q)$ and $k(q)$ are the feedback polynomial operators and have parameters which are determined by the unknown system, the observer and the reference model characteristics. It can be shown that degree$[h(q)] \leq (n - 1)$ and degree$[k(q)] \leq (n - 1)$.

1. Controller Parameters

To obtain the unknown control parameters in terms the parameters of the system, the observer and the reference model, we introduce first the notion of the partial state $z(t)$ [Ref.3], in which we represent the system of equation (1-1) as

$$A(q)z(t) = u(t)$$
$$y(t) = B(q)z(t) \quad . \tag{1-4}$$

Combining equations (1-3) and (1-4), we can easily express the closed loop dynamic as

$$[\alpha(q)A(q) - h(q)A(q) - k(q)B(q)]z(t) = \alpha(q)v(t) \quad ,$$
$$y(t) = B(q)z(t) \quad . \tag{1-5}$$

To obtain the desired closed loop behavior, the equality

$$\alpha(q)A(q) - h(q)A(q) - k(q)B(q) = \frac{1}{b_1}\alpha(q)D(q)B(q) \tag{1-6}$$

must be satisfied so that the closed loop system has r of its poles coincide with those of the reference model. The remaining poles must cancel the open loop zeros, so that the closed loop dynamic is the same as the reference model's, apart from the scaling factor b_1 on the reference input $v(t)$.

Re-arranging equation (1-6), the following Diophantine equation is obtained [Ref.1:pp 508-510]

$$h(q)A(q) + k(q)B(q) = \alpha(q)\left[A(q) - \frac{1}{b_1}D(q)B(q)\right] \quad . \tag{1-7}$$

The left side of equation (1-7) is of degree $\leq (2n - 1)$, while $\alpha(q)A(q)$ is of degree $2n$. Hence the factor $1/b_1$ is needed to ensure that $(1/b_1)\alpha(q)D(q)B(q)$ is monic and thus eliminating the q^{2n} term on the right side of the equation. If $A(q)$ and $B(q)$ are relatively

co-prime (i.e. there is no pole-zero cancellation), then a unique solution for $h(q)$ and $k(q)$ is guaranteed to exist[2].

2. Parameter Estimation

Since $A(q)$ and $B(q)$ are unknown polynomial operators, an estimator is required to estimate the system parameters in order to implement the controller using the estimated parameters. In the following development, on-line estimation based on a particular regression form shall be used to recursively estimate the controller parameters directly.

Applying the polynomial operator in equation (1-7) to the partial state $z(t)$, the following regression equation,

$$\alpha(q)u(t) = h(q)u(t) + k(q)y(t) + \frac{1}{b_1}\alpha(q)D(q)y(t) \qquad (1\text{-}8)$$

is obtained. Then using q as the forward time shift operator and the filtered input and output signals defined by

$$\begin{aligned} q^{-n}\alpha(q)y^F(t) &= y(t) \\ q^{-n}\alpha(q)u^F(t) &= u(t) \ , \end{aligned} \qquad (1\text{-}9)$$

a more convenient form of the regression equation (1-8) is obtained in equation (1-10).

---

[2] The left hand side of equation (1-7) can be cast into a Sylvester matrix multiplied by the parameter vector consisting of the unknown coefficients of h(q) and k(q). The Sylvester matrix is non-singular if A(q) and B(q) are relatively co-prime [Ref.3:p.159] and a solution for the parameter vector is guaranteed in this case.

$$q^{-r}u(t) = q^{-(n+r)}h(q)u^F(t) + q^{-(n+r)}k(q)y^F(t) + \frac{1}{b_1}q^{-r}D(q)y(t) , \qquad (1\text{-}10)$$

or

$$q^{-r}u(t) = \Phi(t)^T \Theta_o ,$$

where

$$\Phi(t) = \begin{bmatrix} u^F(t-r-1) \\ u^F(t-r-2) \\ \cdots \\ u^F(t-r-n+1) \\ y^F(t-r-1) \\ y^F(t-r-2) \\ \cdots \\ y^F(t-r-n+1) \\ q^{-r}D(q)y(t) \end{bmatrix} \qquad \Theta_0 = \begin{bmatrix} h_1 \\ h_2 \\ .. \\ h_{(n-1)} \\ k_1 \\ k_2 \\ .. \\ k_{(n-1)} \\ \frac{1}{b_1} \end{bmatrix} . \qquad (1\text{-}11)$$

Equation (1-11) is a realizable linear in the parameter regression equation with a linear regressor $\Phi(t)$. In this form, many standard recursive estimation techniques can be used to estimate the unknown parameter vector Θ_o. The following estimate $\hat{\Theta}(t)$ of Θ_o is obtained by applying the recursive least-squares estimation technique[3] as follows:

$$\hat{\Theta}(t+1) = \hat{\Theta}(t) + \frac{P(t)\Phi(t)[u(t-r) - \Phi^T(t)\hat{\Theta}(t)]}{1 + \Phi^T(t)P(t)\Phi(t)} , \qquad (1\text{-}12)$$

---

[3] The value of **P**(0) to start the recursion is discussed in most texts on recursive least squares estimation.

$$P(t+1) = P(t) - \frac{P(t)\Phi(t)\Phi^T(t)P(t)}{1 + \Phi^T(t)P(t)\Phi(t)} . \tag{1-13}$$

Now the control equation (1-3) can be rewritten as

$$u(t) = q^{-n}h(q)u^F(t) + q^{-n}k(q)y^F(t) + \frac{1}{b_1}v(t) . \tag{1-14}$$

This can be further rearranged as

$$u(t) = \Phi_c(t)^T \Theta_o ,$$

where

$$\Phi_c(t) = \begin{bmatrix} u^F(t-1) \\ u^F(t-2) \\ \cdots \\ u^F(t-n+1) \\ y^F(t-1) \\ y^F(t-2) \\ \cdots \\ y^F(t-n+1) \\ v(t) \end{bmatrix} , \tag{1-15}$$

which is identical in structure to the regression equation (1-11). Notice that equation (1-15) has the same parameter vector Θ_o as equation (1-11). $\Phi_c(t)$ is also identical to $\Phi(t)$ except for the time shift q^f and the term $v(t)$ replacing $D(q)y(t)$. Therefore, with identical structure as the estimator, the controller can be directly implemented without the need for an intermediate control design calculation. In the control phase, the current estimate $\hat{\Theta}(t)$ of Θ_o is used to generate $u(t)$.

3. Summary

All the necessary steps from performance specification to the design of the DMRAC for unknown LTI systems have been developed. In summary, the design and implementation procedures are:

1. The observer characteristic polynomial $\alpha(q)$ of degree n and the desired reference model $1/D(q)$ of degree $(n - m)$ are first chosen.

2. The closed loop system output $y(t)$ is filtered by the inverse reference model to obtain $D(q)y(t)$. $y^F(t)$ and $u^F(t)$ are obtained by filtering the input and output signals, $u(t)$ and $y(t)$, respectively, by the observer $(1/[q^n\alpha(q)])$.

3. The vector $\Phi(t)$ is formed as shown in equation (1-11) and used as input to the parameter estimator. On-line estimation of the parameter vector Θ_o can be performed using equations (1-12) and (1-13).

4. The control signal $u(t)$ for the closed loop system is generated using equation (1-14) and the estimated parameter vector $\hat{\Theta}(t)$ (instead of Θ_o).

Figure 4 illustrates the estimation and control algorithm of the DMRAC. Note that the block $\hat{\Theta}(t)$ is a linear associative memory with recursive estimation updates to minimize mean square errors between $u(t)$ and $\hat{u}(t) = \Phi(t)^T \hat{\Theta}(t)$.

Appendix A contains a worked example of the design of a DMRAC for an unknown LTI systems. Software simulations are conducted to show how the DMRAC can be implemented and how it works. MATLAB[4] software environment is employed in all the software simulations conducted.

---

[4] MATLAB® is a registered trademark of The MathWorks, Inc.

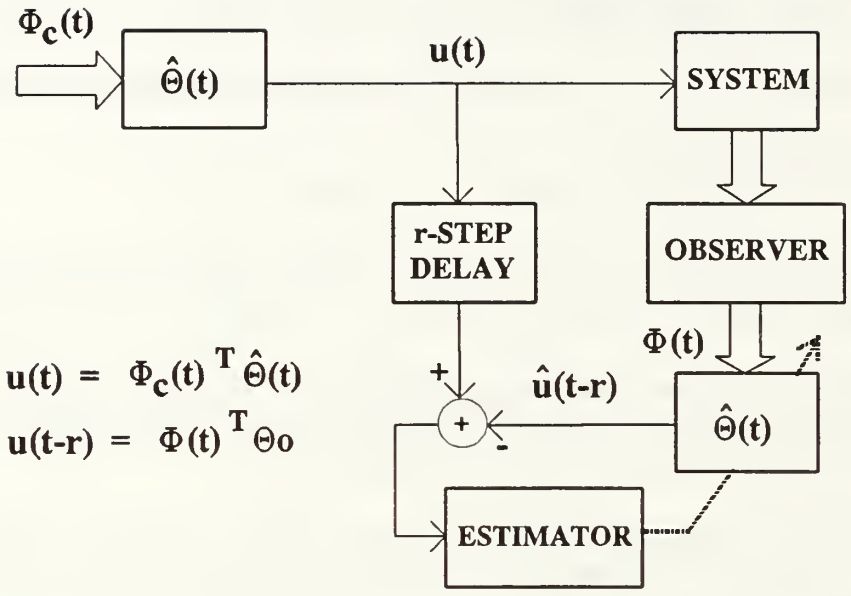

Figure 4: Estimation and Control Algorithm of the DMRAC

E. ADAPTIVE CONTROL OF NONLINEAR SYSTEMS

Many variations of the adaptive control technique analyzed above have been also developed to handle different assumptions about the unknown LTI systems [Ref.4]. Since most of these techniques deal with linear systems, simple linear functions of the measurements or observations, such as $y^F(t)$, $y^F(t-1)$,..., $u^F(t)$, $u^F(t-1)$,.... (assuming a SISO system) are always sufficient to form the regressor vector $\Phi(t)$ to give a regression equation which is linear in the parameters.

However, with nonlinear systems, the use of nonlinear functions of the measurements or observations in the regressor vector becomes almost always necessary

in order to keep the regression equation linear[5]. Therefore it is necessary to know the exact nature of these nonlinear transformations in order to form the linear regressor to allow the use of standard parameter estimation techniques. Chapter 5 of [Ref.5] provides more details on the use of standard parameter estimation techniques for nonlinear systems.

Since the nonlinear system to be controlled is assumed unknown, the appropriate nonlinear regressor required by the estimator is unknown. Therefore, the conventional approach in using standard parameter estimation technique such as least squares estimation cannot be used. It has been shown in [Ref.6] that a neural network can learn to emulate any continuous function. The idea then is to replace the linear associative memory of $\hat{\theta}(t)$ with a neural network. The neural network shall be taught to emulate the appropriate nonlinear controller in the same manner as the recursive least squares estimator is used in the DMRAC for LTI systems. The specific structure of the neural network-based direct model reference adaptive controller for certain classes of nonlinear systems is developed in the next chapter. The performance of the neural network as a DMRAC is investigated experimentally in Chapter III.

---

[5] This implicit requirement arises from the fact that existing estimation techniques generally require a linear parameterization of the system.

II. BNN DIRECT MODEL REFERENCE ADAPTIVE CONTROL

A. APPLICATIONS OF NEURAL NETWORKS

A large variety of artificial neural networks has been developed and employed in numerous applications [Ref.7]. Successful applications of artificial neural networks have been developed in such areas as pattern recognition, speech and natural language processing, image compression, functional optimization, and even financial and economic system modelling. Artificial neural networks have also been highly touted for control engineering applications with early experiments such as the self-learning broomstick balancer [Ref.8] and the recent neural network truck backer-upper [Ref.9].

A neural network usually consists of a large number of simple processing elements, known as neurons. Each neuron has a number of inputs, each associated with a synaptic weight as shown in Figure 5. It usually performs only very simple mathematical operations:

- each input (including a fixed bias) to the neuron is multiplied by the associated synaptic weight.
- the results of the multiplications for all the inputs are summed.
- the summand is then mapped to the output of the neuron through a nonlinear function $\Gamma[\cdot]$. Typically, $\Gamma[\cdot]$ is a monotonically increasing function (e.g. $\tanh[\cdot]$).

The first two operations is actually a scalar dot-product between the inputs and

the associated synaptic weight vector of the neuron. The neurons are often interconnected in layers, in a predefined manner.

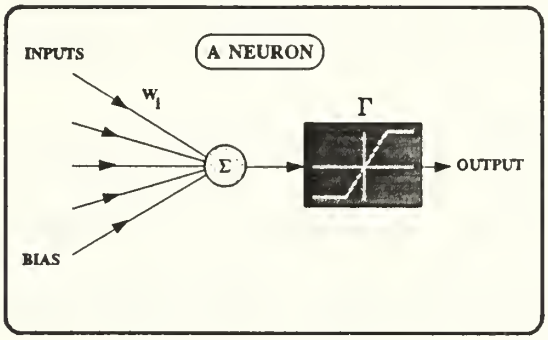

Figure 5: A Neuron.

The most distinctive and appealing feature of many neural networks is that they learn by examples. Learning in the context of artificial neural network, is achieved through adapting the synaptic weights of the neurons. The synaptic weights then serve as a form of associative memory mapping the inputs of the neural network to its outputs. Based only on pre-assigned learning rules, the neural network can hence derives its functionality through learning by examples rather than through the traditional programming approach employed in traditional von Neumann machines. Hence, neural networks provide an approach that is closer to human perception and recognition than most other information processing approaches in use today.

Currently, the most popular and commonly used neural networks for control system design is the Back-propagation Neural Network (BNN). Its popularity stems from the fact that the BNN implements a learning procedure, known commonly as the generalized delta rule [Ref.10], that allows it to learn to emulate a very large class of nonlinear functions.

The structure of the BNN will be discussed in detail in the next section. This is followed by a description of the four general classes of SISO nonlinear systems

considered for control by a BNN DMRAC. Finally, the structure of a BNN DMRAC for each class of these nonlinear systems is established.

B. ANALYSIS OF BACK-PROPAGATION NEURAL NETWORK

A back-propagation neural network is a multi-layer, feed-forward network which has an input layer, an output layer and at least one hidden layer. Neurons are found in the output and hidden layer(s) while the input layer has only input connections feeding the neurons in the first hidden layer. Figure 6 shows a multi-layer back-propagation

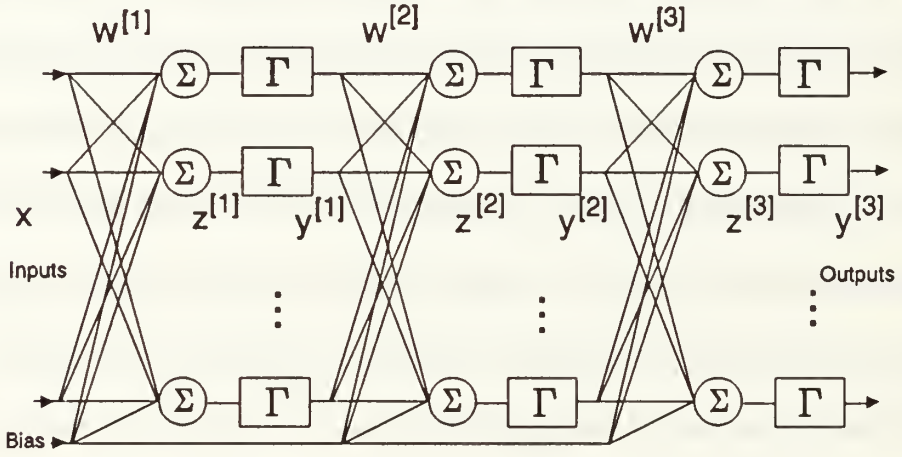

Figure 6. A Back-propagation Neural Network

network with two hidden layers. In the back-propagation neural network, the signal flows from input to output layers. There is no feedback or even interconnection between neurons in the same layer. There is usually also a bias input for each neuron with an associated non-zero synaptic weight.

To describe mathematically the learning process the BNN uses, we first define $w_{jk}^{[i]}$ as the connection weight for the path from the j^{th} neuron in $(i-1)^{th}$ layer to the k^{th} neuron in i^{th} layer. Also define x_j as the j^{th} input to the neural network. Then the BNN in Figure 6 can be represented mathematically as

$$y^{[3]} = \Gamma[W^{[3]}\Gamma[W^{[2]}\Gamma[W^{[1]}x]]] \ , \tag{2-1}$$

where $x = \{x_j\}$ is the vector of all the inputs to the BNN. $W^{[i]} = \{w_{jk}^{[i]}\}$ the synaptic weight matrix of the i^{th} layer formed from columns of synaptic weights associated with the inputs of each neuron. $y^{[i]} = \{y_k^{[i]}\}$ is the vector of all outputs in the i^{th} layer. Next we define also $z_j^{[i]}$ as the summation of weighted inputs of the j^{th} neuron. In the learning process, the BNN adjusts the synaptic weights $W^{[i]}$ for all i, to minimize a suitable function of the error between the output $y^{[N]}$ and a desired output $yd = \{yd_k\}$ for a N-layer BNN. The most common error function used is

$$E = \frac{1}{2} \sum_{All\ k} (yd_k - y_k)^2 \ , \tag{2-2}$$

where k is the index spanning all the output neurons. This minimization is performed for each set of input vector given to the BNN. Other forms of error functions, including the sum of the absolute errors, can also be used.

The BNN implements a modification of the gradient descent algorithm (also known as the least-mean-squares method, LMS) to update each synaptic weight at time $t + 1$ with

$$\left(\Delta w_{jk}^{[i]}\right)_{t+1} = -\mu * \left(\frac{\partial E}{\partial w_{jk}^{[i]}}\right) + \nu * \left(\Delta w_{jk}^{[i]}\right)_t , \tag{2-3}$$

where μ and ν are scalars representing the learning rate and the momentum rate. The learning rate is equivalent to the step-size parameter in the conventional LMS algorithm. Like the LMS algorithm, too large a learning rate often leads to instability of the learning system while too small a value would result in a very slow learning process. The use of a momentum term has been found to speed up the learning process considerably. It allows a larger learning rate yet avoids the point of instability. For the i^{th} layer,

$$\begin{aligned} \frac{\partial E}{\partial w_{jk}^{[i]}} &= \left(\frac{\partial E}{\partial z_j^{[i]}}\right) * \left(\frac{\partial z_j^{[i]}}{\partial w_{jk}^{[i]}}\right) \\ &= -e_j^{[i]} * x_k^{[i]} , \end{aligned} \tag{2-4}$$

where the local error vector $e_j^{[i]}$ is given by

$$e_j^{[i]} = \Gamma'\left(z_j^{[i]}\right) * \sum_{All\ k} e_k^{[i+1]} * w_{jk}^{[i+1]} . \tag{2-5}$$

Equation (2-4) is a direct application of the chain rule in differential calculus. At the output layer (say, N^{th} layer),

$$e_j^{[N]} = \Gamma'\left(z_j^{[N]}\right) * \left(yd_k - y_k^{[N]}\right) . \tag{2-6}$$

Once $e^{[N]} = \{e_j^{[N]}\}$ at the output layer is obtained, then $e^{[N-1]}$, $e^{[N-2]}$,... can be recursively computed using equations (2-4) and (2-5). The weights can be updated using equation (2-3). Note that different learning rates and momentum rates can be used in different layers.

The equations (2-3) through (2-5) describe mathematically the error back-propagation mechanism from which the BNN derived its name. Figure 7 describes the learning process diagrammatically. $\Gamma'(\cdot)$ is the first derivative function of $\Gamma(\cdot)$ and π represent the term-by-term products of the two sets of inputs.

As proposed earlier, a direct model reference adaptive controller shall be built by replacing the linear associative memory block of the DMRAC (see Figure 4) with a BNN. As an initial proof of the concept, an experiment was conducted by replacing the least-squares estimator with a BNN directly in the DMRAC for unknown LTI systems. The results of this experiment are shown in Appendix B together with the programs

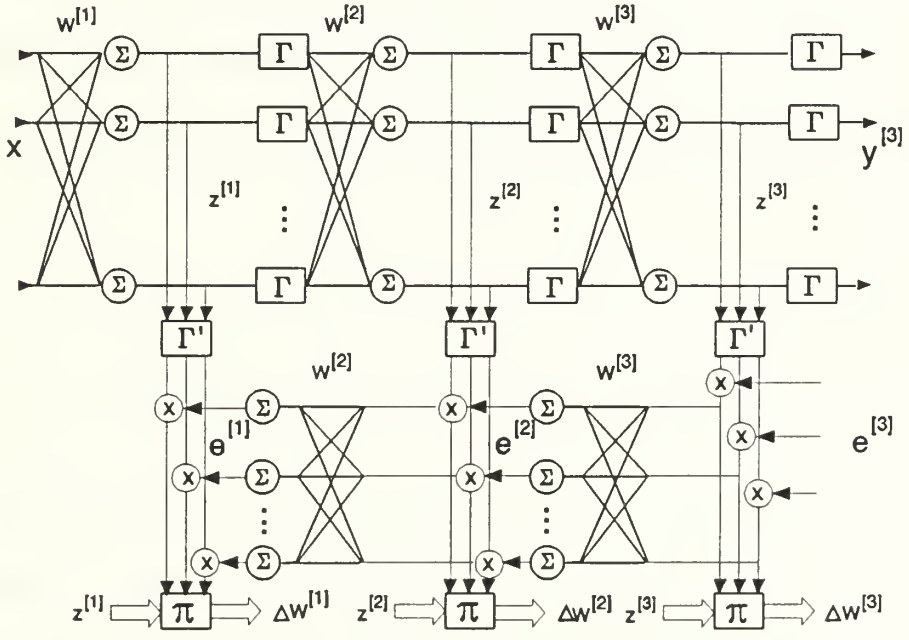

Figure 7. Back-propagation Neural Network Learning

developed for the simulations. The results indicate that the BNN in the DMRAC structure

can control an unknown LTI system as well as the DMRAC based on recursive least-squares estimator.

C. NONLINEAR SYSTEMS FOR BNN DMRAC

Four important classes of unknown nonlinear SISO systems are considered for direct adaptive control using the BNN. They are modelled in discrete-time for analysis and simulation. These are the system models used in [Ref. 2] for which BNN indirect adaptive control has been successfully demonstrated. They are important because many real world systems are readily described by these models [Ref. 5]. Mathematical models are first introduced to describe these systems so that the structure of the BNN DMRAC can be developed analytically. The four models are:

(1) Model 1:

$$y(t+1) = \sum_{k=0}^{n-1} a_k y(t-k) + g\,[(u(t),u(t-1),..,u(t-m+1)] \qquad (2\text{-}7)$$

In this model, the external input $u(t)$ is subjected to a nonlinear mapping $g[\cdot]$. The result then acts as the system input. These auto-regressive systems are indeed very common. For example, large mechanical systems, hard nonlinearities such as input saturation, dead-zones or backlash are readily described by this model.

(2) Model 2:

$$y(t+1) = f[\, y(t),y(t-1),..,y(t-n+1)] + \sum_{k=0}^{m-1} b_k u(t-k) \qquad (2\text{-}8)$$

In the second model, the auto-regressive variables of the difference equation describing the model are subjected to a nonlinear functional mapping. Again this class of systems is very common. As an example, the action of viscous drag on an underwater vehicle can be modelled by an equation of this form.

(3) Model 3:

$$y(t+1) = f[\, y(t),y(t-1),..,y(t-n+1)] + g\,[u(t),u(t-1),..,u(t-m+1)] \qquad (2\text{-}9)$$

Here both the input and the auto-regressive variables are subjected to nonlinear functional mapping. However the nonlinear mapping of the input and the auto-regressive variables remain separate. Again, it is not difficult to find real world systems that are closely described by this model. For example, an underwater vehicles subjected to input saturation and viscous drag could be conveniently modelled in discrete-time by a difference equation of this form.

(4) Model 4:

$$y(t+1) = h\,[\, y(t),y(t-1),..,y(t-n+1),u(t),u(t-1),..,u(t-m+1)] \qquad (2\text{-}10)$$

In this model, a single nonlinear functional mapping applies to the external input as well as the autoregressive variables of the difference equation. An example of this class of systems is the bilinear system.

D. DEVELOPMENT OF THE BNN DMRAC

Consider the class of systems described by Model 1. By replacing $g[u(t)]$ with $w(t)$, an equivalent linear system

$$y(t+1) = \sum_{k=0}^{n-1} a_k y(t-k) + w(t) \qquad (2\text{-}11)$$

is obtained. This has a form similar to equation (1-1). Therefore, the development of the DMRAC for this equivalent linear system would be exactly as in Chapter I, Section D. The regression equation for the estimator will be identical to equation (1-11) with $u^F(t)$ replaced by $w^F(t) = q^{-n}\alpha(q)w(t)$, where $w(t) = g[u(t)]$.

Unfortunately, since $g[\cdot]$ is unknown, there is a problem in forming the regressor which shall be used as the input to a standard estimator. If a BNN can be taught to emulate the nonlinear mapping of $w^F(t) = g[u^F(t)]$, then the regressor can be formed. In addition, if it can also be taught to perform the parameter estimation simultaneously, then we will have a BNN DMRAC. Hence one approach is to replace the least-squares estimator with a BNN. The BNN can be trained using the input vector $\Phi(t)$ and the desired output $q^{-r}u(t)$ of equation (1-11). The BNN is expected to learn the functional

mapping of $g[\cdot]$ while performing the parameter estimation simultaneously. The same BNN can then be used as the controller, generating $u(t)$ given the input vector $\Phi_c(t)$ as in equation (1-15).

Next consider the class of systems described by Model 2. If a BNN can emulate the nonlinear mapping of the following control equation

$$u(t) = -\frac{1}{b_0}\left(f\left[y(t), y(t-1), .., y(t-n+1) \right] + \sum_{k=1}^{m-1} b_k u(t-k) - r(t) \right), \qquad (2\text{-}12)$$

then the system will track $r(t)$[6]. As long as a BNN can be taught to emulate this nonlinear mapping given the direct measurements $y(t)$, $y(t-1)$,..., $u(t)$, $u(t-1)$,... and $r(t)$, a controller can be realized. A regression equation suitable for parameter estimation can be obtained by replacing $r(t)$ with $y(t)$ in equation (2-12) provided the inverse mapping exists. So a BNN shall be employed to learn the nonlinear mapping of equation (2-12) using this regression form, and to act as the controller.

In many cases, the direct state measurements forming the inputs are not always accessible in the actual system. For example, under continuous-time modelling, these measurements may be derivatives of some physical measurements such as velocity or angular acceleration and are usually not accessible as measurements. Hence observations such as $y^F(t)$, $y^F(t-1)$,..., $u^F(t)$, $u^F(t-1)$,... shall be used as inputs to the BNN controller instead of the direct measurements. In addition, $v(t)$ (the reference input for the model

---

[6] This is equivalent to model tracking, if r(t) is the output of the model system given a reference input v(t), i.e. D(q)r(t) = v(t).

reference system) instead of $r(t)$ shall be used. Likewise the same state observations and $D(q)y(t)$ (since $v(t)$ is used) shall then be used to form the input vector for the BNN learning. This keeps the approach completely identical to that of the previous case.

For systems described by Model 3 and Model 4, the required forms of control $u(t)$ for model tracking are

$$g[\, u(t),u(t-1),....,u(t-m+1)] = -f[\, y(t),y(t-1),...,y(t-n+1)] + r(t) \;, \qquad (2\text{-}13)$$

$$h[\, y(t),y(t-1),...,y(t-n+1),u(t),u(t-1),...,u(t-m+1)] \;=\; r(t) \qquad (2\text{-}14)$$

respectively, provided the inverses of $g[\cdot]$ and $h[\cdot]$ exist and are unique. These can be implemented as long as the BNN can be taught to emulate these nonlinear mapping of $f[\cdot]$ and the inverses of $g[\cdot]$ and $h[\cdot]$, so that they will generate the appropriate $u(t)$ given the measurements $y(t)$, $y(t-1)$,.... $u(t-1)$, $u(t-2)$,... and $r(t)$. Since a suitable regression equation in each case can be obtained by replacing $r(t)$ with $y(t)$, the BNN can be taught using these regression equations.

To keep the approach consistent with the previous cases, observations such as $y^F(t)$, $y^F(t-1)$,..., $u^F(t)$, $u^F(t-1)$,... shall be used as inputs to the BNN for both learning and control. In the control phase, $v(t)$ instead of $r(t)$ shall be used. In the learning phase, $D(q)y(t)$ shall replace $v(t)$. Therefore in all cases, the structure in Figure 8 can be employed.

With least-squares estimator DMRAC, the parameter estimation is carried out on-line, usually starting with arbitrary states (normally zero) for all the parameters. It can be shown that the LTI system under control will be stable even under such conditions.

However, for the BNN DMRAC, this cannot be fully assured. Due to the nonlinearities of the system and the BNN, there is yet no general means or conditions to assure the stability of the controlled system when starting with an untrained BNN. Also the output

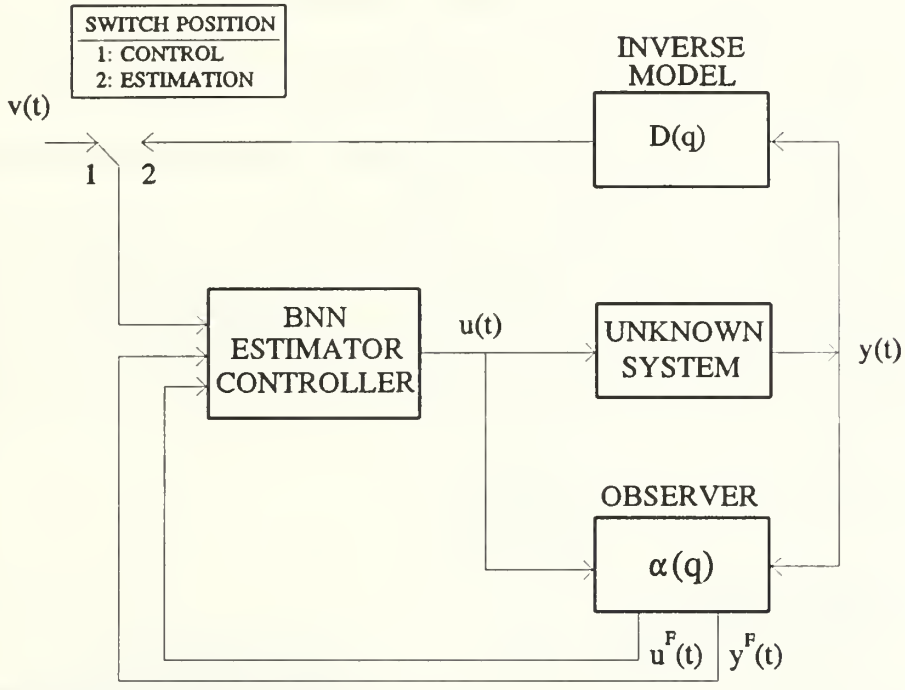

Figure 8. Structure of the BNN DMRAC

of the BNN has a saturation limit due to the use of a saturating nonlinear function in each neuron. Hence, it is very likely that by applying an untrained BNN directly to control the unknown nonlinear system instability in some cases will result. Hence, the training for the BNN is broken into two phases: off-line and on-line.

1. Off-line Learning

The off-line training phase uses an arbitrary input *u(t)* to drive the system and produce the output *y(t)*. From these measurements, $\Phi(t)$ in equation (1-10) is formed and

used as input vector to the BNN under training. The desired output given this input is $q^{-r}u(t)$ and shall be compared to the actual BNN output to obtain the output error. This is then used for BNN learning as described in Section A. The procedure for off-line training can be rationalized as follows: Assume that there exist a controller when driven by $[u^F(t), u^F(t-1),\ldots, y^F(t), y^F(t-1),\ldots v(t)]^T$ generates the required $u(t)$ so that the controlled system tracks the specified reference model. The input vector for the estimator and the desired output vector will then be $[q^{-r}u^F(t), q^{-r}u^F(t-1),\ldots, q^{-r}y^F(t), q^{-r}y^F(t-1),\ldots, q^{-r}D(q)y(t)]^T$ and $q^{-r}u(t)$, respectively. We note that $v(t)$ is not required in this input vector. Hence there is no need to know $v(t)$ directly. As a corollary, $u(t)$ can then be chosen arbitrarily.

2. On-line Learning

Having trained the BNN off-line, the on-line training can then be conducted. On-line training for the BNN is identical to the procedures used for on-line recursive least squares estimation. The BNN continues to learn (by updating its weights at each time step) based on input vector $\Phi(t)$ and desired output $q^{-r}u(t)$ as in the off-line learning. Upon each update, the BNN is used as the controller with $\Phi_c(t)$ as the input vector. The control signal generated is then used to drive the system. Figure 9 illustrates both the off-line learning and the on-line learning and control algorithms discussed.

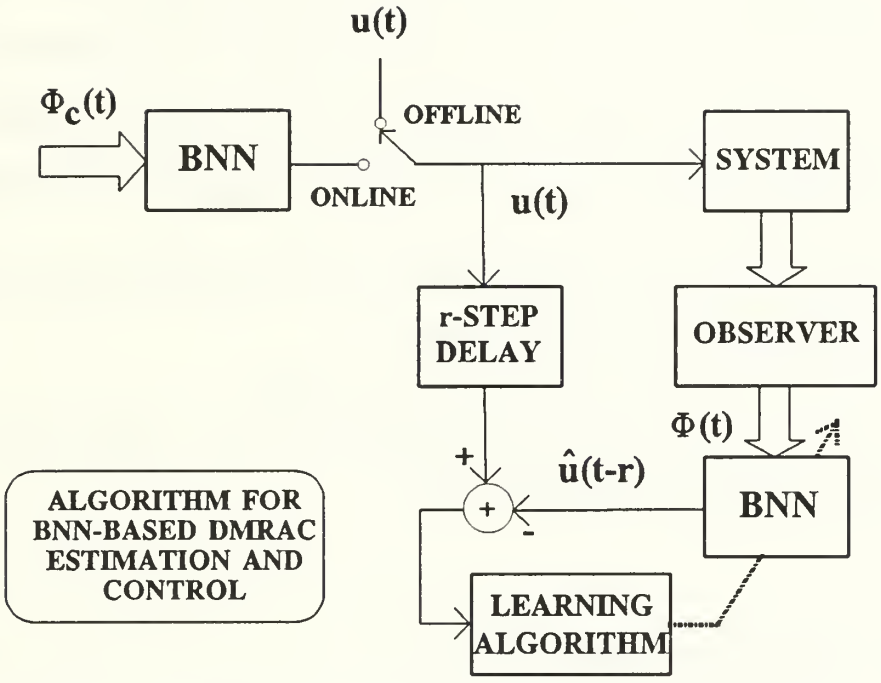

Figure 9: Off-line and On-line Learning Algorithm

III. SIMULATIONS, RESULTS AND DISCUSSIONS

A. EXPERIMENTING WITH THE BNN DMRAC

In this section, the results of the experiments using the BNN DMRAC on various nonlinear SISO systems, are presented. Four experiments were conducted using software simulations, covering the control of the four classes of unknown nonlinear systems considered in Chapter II. The main purpose of these experiments is to see under what conditions the proposed BNN DMRAC works. The software simulation programs used in these experiments are listed in Appendix C.

In the next section, some important general observations regarding the controller, its design and implementation are discussed.

1. Experiment 1: System Model 1

In the first experiment, a nonlinear system in the class described by Model 1 is to be controlled by a BNN DMRAC. Its discrete-time model is governed by the difference equation

$$y(t+1)=a_1y(t)+a_2y(t-1)+g[u(t)] \ . \qquad (3\text{-}1)$$

The parameters $a_1 = 0.3$, $a_2 = 0.6$ and the nonlinear function $g[x] = x^3 + 0.3x^2 - 0.4x$ are assumed unknown to the controller. As discussed earlier, by replacing $g[u(t)]$ by $w(t)$,

an equivalent linear system is obtained: $A_{eq}(q)y(t) = B_{eq}(q)w(t)$, where the polynomial operators $A_{eq}(q) = q^2 - 0.3q - 0.6$, and $B_{eq}(q) = q$, and q is the forward time-shift operator. The degree of $A_{eq}(q)$ is $n = 2$ while that of $B_{eq}(q)$ is $m = 1$. Assuming that these are known, the following observer,

$$\alpha(q) \equiv (q-0.1)(q-0.05) \ , \tag{3-2}$$

of degree $n = 2$, was chosen for the design. Since $(n - m) = 1$, the reference model of degree 1 was specified as $(q - 0.8)y(t) = v(t)$.

Using a BNN estimator, the input vector at time, t for BNN learning is given by

$$\Phi(t) = \begin{bmatrix} u^F(t-1) \\ u^F(t-2) \\ y^F(t-1) \\ y^F(t-2) \\ (1-0.8q^{-1})y(t) \end{bmatrix} , \tag{3-3}$$

where $u^F(t) = q^{-2}\alpha(q)u(t-1)$ and $y^F(t) = q^{-2}\alpha(q)y(t-1)$. The desired BNN output is $u(t-1)$. According to the suggested training procedures, the BNN was first trained off-line. The training set generated from an training input $u(t)$ and the resulting open-loop system output $y(t)$, are shown in Figure 10. The training set consisted of 200 data points each for the input and output measurements, $u(t)$ and $y(t)$. $u(t)$ is a sum of sinusoids with different magnitudes and phases, each with a small random varying phase component. From the data set, 200 sets of input vectors in the form of equation (3-3) were obtained. The magnitude of $u(t)$ was adjusted such that $u(t)$ is always within the range ± 0.8 while the norm of each input vector $\Phi(t)$ for BNN training was kept less than 1. The input

vector with the associated desired output *u(t-1)* were then presented in a random order to the BNN for learning. After 50 passes through the entire set of input vectors (i.e.

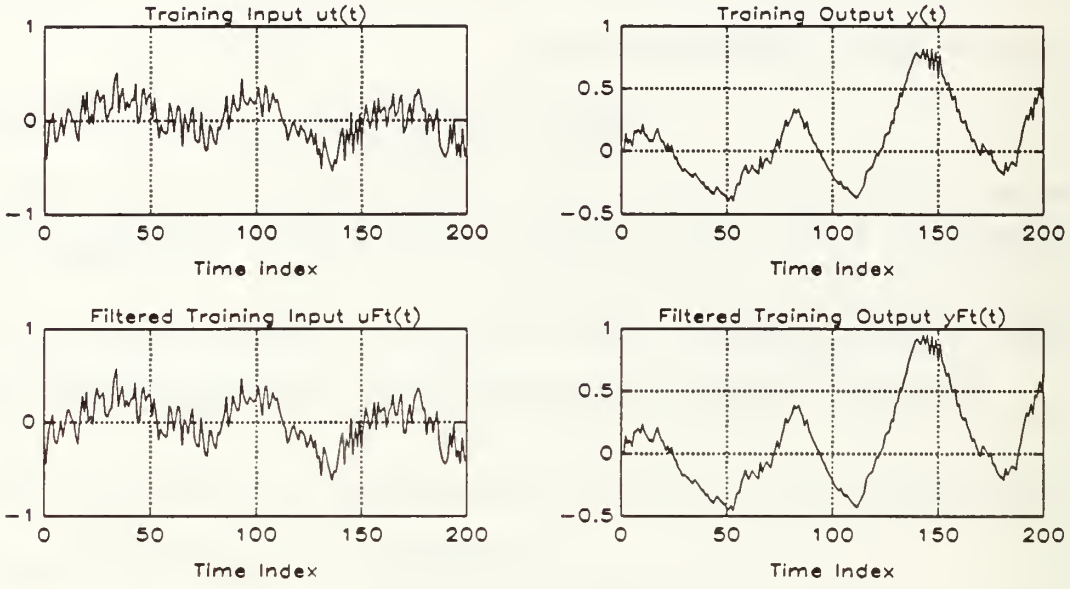

Figure 10: Off-line Training Data. System Model 1.

10,000 training examples), the BNN was tested with a different set of input *u(t)* and output *y(t)*, as shown in Figure 11. In the test, the BNN estimator was fed the input vector of equation (3-3) formed with the new data set. The BNN output $\hat{u}(t)$ is then compared to *u(t)*. No learning takes place during testing. The result is shown in Figure 12. As shown, $\hat{u}(t)$ was almost identical to *u(t)*, indicating that the BNN had been adequately trained.

The BNN was next placed on-line to control the system. During the on-line control mode, the BNN recursively learns to adapt to the required control structure and parameters. During the learning phase. the BNN estimator was fed with the input vector $\Phi(t)$ as in equation (3-3) and the desired output *u(t-1)*. During the control phase, the input

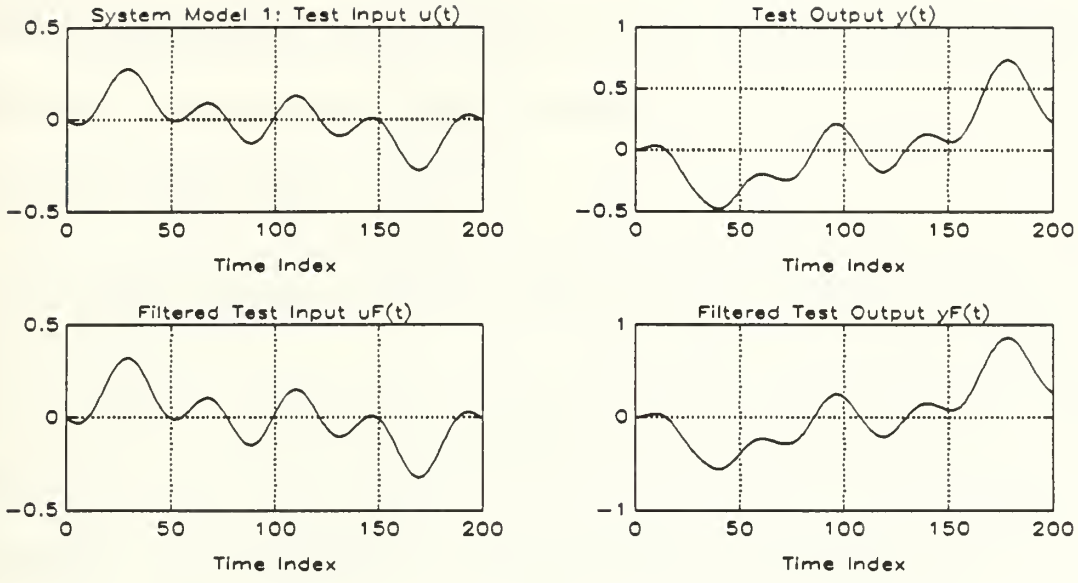

Figure 11: Test Data for Off-line Training of the BNN DMRAC.

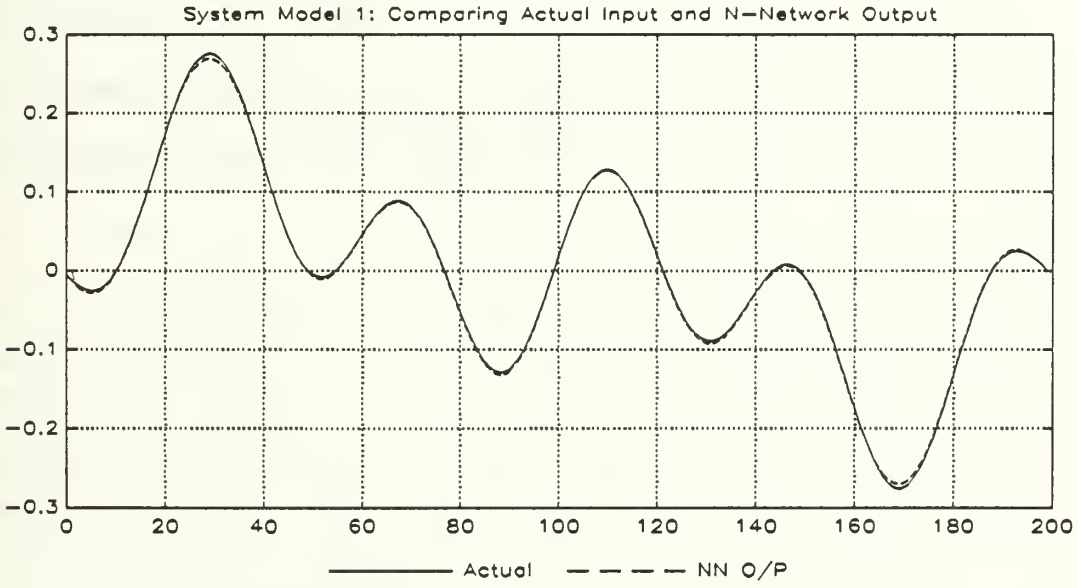

Figure 12: Test Result for Off-line trained BNN DMRAC.

vector to the BNN was replaced by

$$\Phi_c(t) = \begin{bmatrix} u^F(t) \\ u^F(t-1) \\ y^F(t) \\ y^F(t-1) \\ v(t) \end{bmatrix} \tag{3-4}$$

to generate *u(t)*. The result of one such experiment is shown in Figure 13. In this

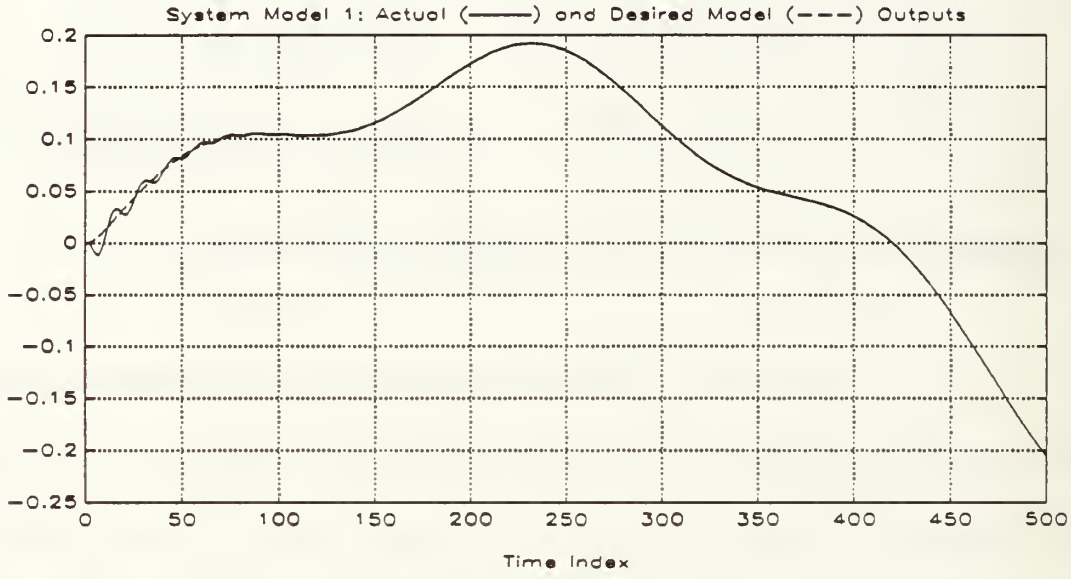

Figure 13. On-line Control. System Model 1.

experiment, a 5000-point reference input *v(t)* was used. The output of the controlled system are compared to that of the reference model in the figure (only the first 500 points are shown), which had been given the same reference input *v(t)*. It can be seen that the unknown nonlinear system has been successfully controlled by the BNN DMRAC.

On the other hand, a DMRAC designed with a least-squares estimator, assuming the unknown system is LTI, failed to work for this system. Hence, the BNN DMRAC actually offers a viable method to control such an unknown nonlinear system while the conventional technique failed.

Other experiments conducted showed that the BNN DMRAC performs its control function reasonably well for various types of inputs. However, for inputs with high frequency components (with respect to the sampling rate), the controlled system became quite oscillatory. It could not track the reference model well in this situation. In addition, the controller would also sometimes saturate during the start of on-line control (and learning) and therefore fails to control the system. It is postulated that the solution to this problem is to increase the sampling rate and/or to increase the number of neurons and hidden layers used in the BNN. This arises from the observations that the BNN can usually emulate a 'smoother' function with lesser number of neurons and lesser training. There seems to be a Nyquist-like relationship between the 'smoothness' of the nonlinear function the BNN seeks to emulate and the number of neurons it requires. Off-line training with more appropriate training data (i.e. training signals containing similar frequency characteristics as the actual signals experienced by the controlled system), and adjusting the learning parameters also helps to improve the tracking performance and avoid the saturation. Unfortunately, there is still no general rule to help select the most appropriate learning parameters. Hence a great deal of experimentation is usually required.

2. Experiment 2: System Model 2

In the next experiment, a nonlinear system in the class described by Model 2 is used. It is governed by

$$y(t+1) = \frac{y(t)y(t-1)[y(t)+2.5]}{1+y(t)^2 + y(t-1)^2} + u(t) . \qquad (3\text{-}5)$$

The nonlinear mapping of the auto-regressive variables in the difference equation is unknown to the controller. It can be seen that the following control signal

$$u(t) = -\left(\frac{y(t)y(t-1)[y(t)+2.5]}{1+y(t)^2 + y(t-1)^2}\right) + r(t) \qquad (3\text{-}6)$$

will allow $y(t)$ to track r(t) and hence, achieve model following if $D(q)r(t) = v(t)$. A regression equation for the controlled system is obtained using equation (3-6) with $r(t)$ replaced by $y(t)$. For a consistent approach, observations $u^F(t)$, $y^F(t)$ and input $v(t)$ shall be used instead of $u(t)$ $y(t)$ and $r(t)$ respectively. The input to the BNN estimator shall be the regressor vector $\Phi(t)$ of equation (1-11). The desired output for the BNN estimator is $q^r u(t)$. $\Phi_c(t)$ in equation (1-14) is the input vector during the control phase. The order of the equivalent $A(q)$, $n = 2$ and the order of equivalent $B(q)$, $m = 1$ were assumed known. The observer $\alpha(q)$ in equation (3-2) was again chosen. The same reference model of degree (2 - 1) was also used. Therefore, the BNN estimator input vector at time t is given by equation (3-3). The same procedures for off-line training, testing and on-line control-plus-learning were employed. The BNN estimator was first trained off-line with 50 passes through a 200-point training set. To test the off-line trained BNN, another set of data generated from a different $u(t)$ and $y(t)$ was used. The input vectors formed from

this data set were fed into the BNN to generate the output $\hat{u}(t)$. This was compared to actual $u(t)$. The test result is shown in Figure 14. Again, $\hat{u}(t)$ was almost identical to $u(t)$,

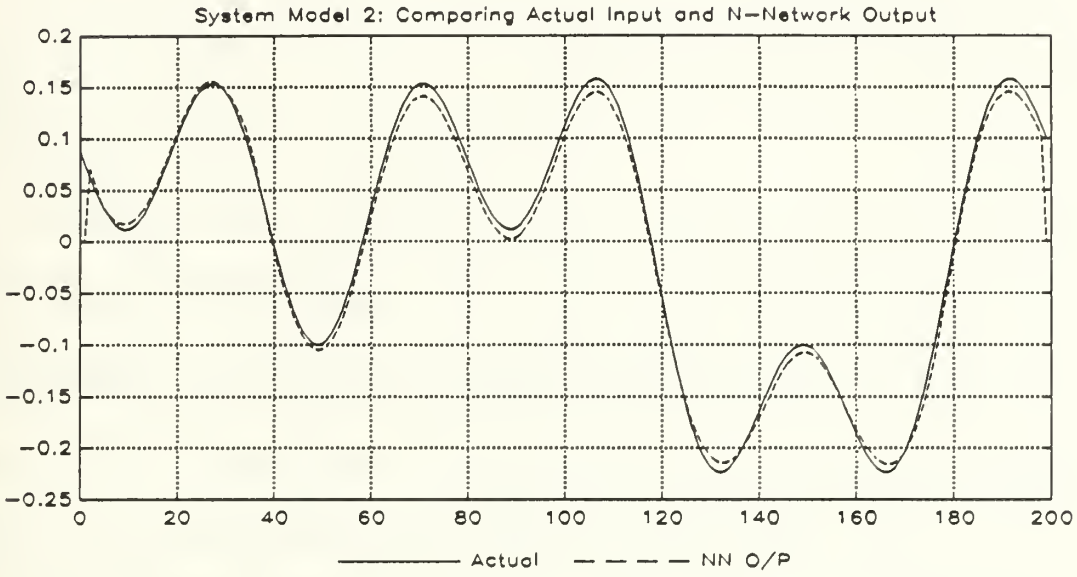

Figure 14: Test Result for Off-line Trained BNN-DMRAC.

indicating that the BNN was adequately trained.

Finally, the BNN was placed on-line to control, and learn the control structure and estimate the parameters simultaneously. The result of one experiment is shown in Figure 15. As shown, the system with the BNN DMRAC successfully tracked the model reference system very closely. A number of other experiments were conducted and the results show that the BNN DMRAC consistently performs its function well. To optimize the performance of the control system, many different learning parameters and training data were tried. However, once a trained BNN works, it tracks the reference model very well for inputs with similar characteristics.

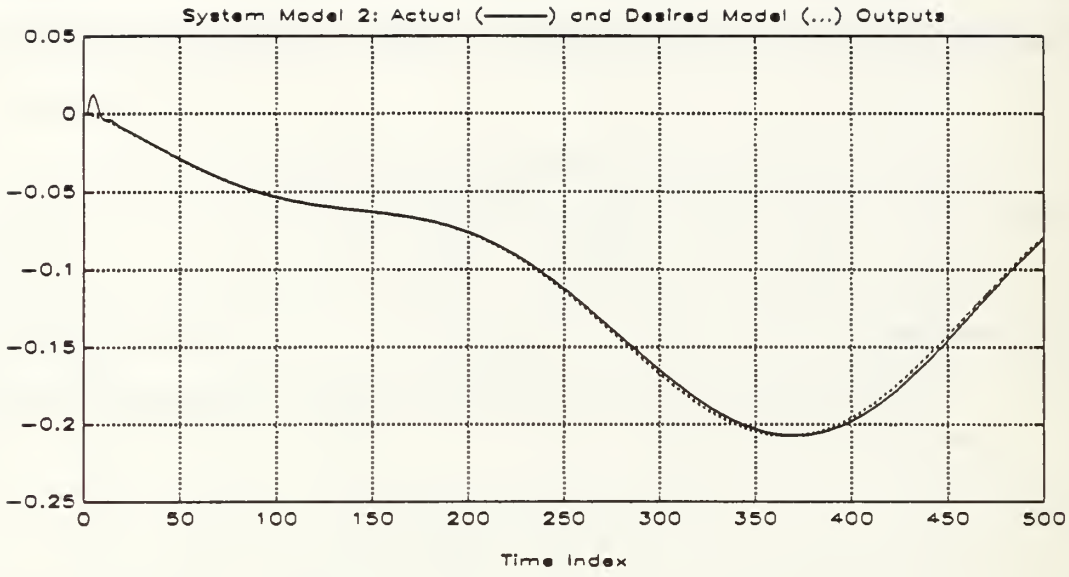

Figure 15: On-line Control. System Model 2.

3. Experiment 3: System Model 3

In this experiment, the nonlinear system is governed by:

$$y(t+1) = \frac{y(t)}{1+y(t)^2} + u(t)^3 \ . \tag{3-7}$$

Again, the nonlinearities associated with the auto-regressive variable ($y(t)$ only) and the input $u(t)$ are assumed unknown to the controller. The following control signal will allow $y(t)$ to track $r(t)$ and achieve model following if $D(q)r(t) = v(t)$:

$$u(t) = 3\sqrt{\left(\frac{-y(t)}{1+y(t)^2} + r(t)\right)}. \qquad (3\text{-}8)$$

One suitable regression equation for the controlled system is equation (3-8) with $r(t)$ replaced by $y(t)$. Again for consistency, observations $u^F(t)$, $y^F(t)$ and the input $v(t)$ shall be used instead of $u(t)$, $y(t)$ and $r(t)$, respectively.

In this design, instead of using the maximum order of the system $n = 1$, it was assumed that $n = 2$. The degree of $B(q)$ was assumed to be 1 even though it is of degree zero. The BNN estimator input vector at time t is again the identical to the one given in equation (3-3). The desired output is $u(t-1)$. The observer $\alpha(q)$ was chosen as $\alpha(q) = (q - 0.03)(q - 0.2)$. The reference model chosen was as $(q - 0.6)y(t) = v(t)$.

The same training and testing procedures as in first simulation were adopted. Again the BNN estimator was first trained off-line with 50 passes through a training set. To test the trained BNN, another set of data generated from a different $u(t)$ and $y(t)$ were used. The test result shown in Figure 16 allows for the comparison of $u(t)$ and $\hat{u}(t)$. Again the estimate $\hat{u}(t)$ tracks $u(t)$ very well.

Next, the BNN was placed on-line to control the system. The result of one such experiment is shown in Figure 17. In this simulation, even though a higher order was assumed for the unknown system, the BNN controller managed to perform quite well. Simulations with different inputs were conducted and the results again showed that the BNN DMRAC performs reasonably well for inputs with similar characteristics.

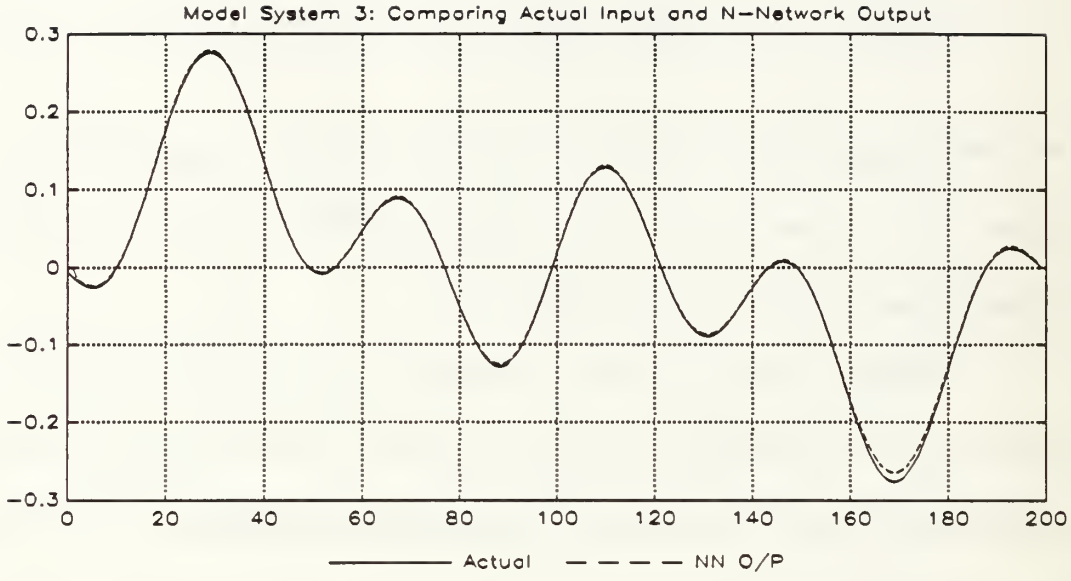

Figure 16: Test Result for Off-line trained BNN DMRAC.

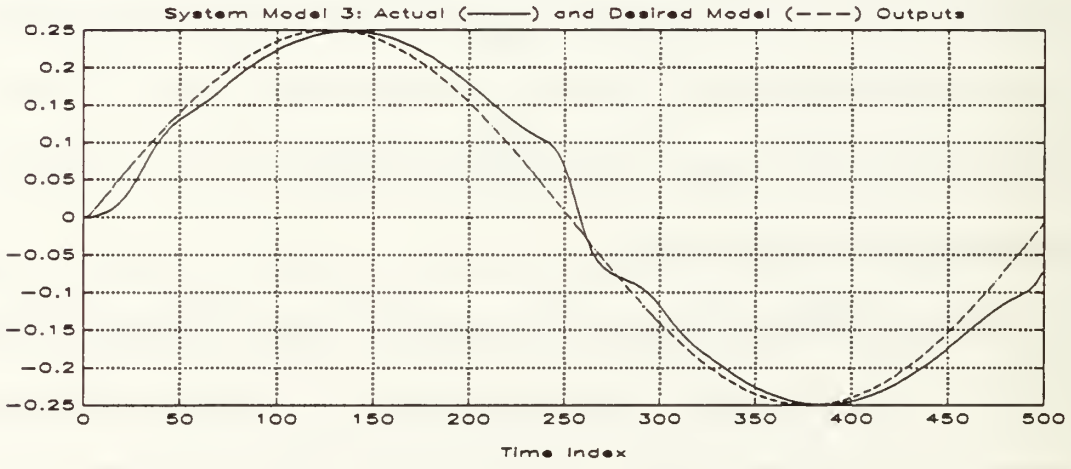

Figure 17: On-line Control. System Model 3.

4. Experiment 4: System Model 4

In the final experiment, the nonlinear system is governed by:

$$y(t+1) = \frac{y(t)y(t-1)y(t-2)u(t-1)[y(t-2)+1] + u(t)}{1+y(t-1)^2 + y(t-2)^2} \tag{3-9}$$

The nonlinear mapping of the difference equation is unknown to the controller. The following control signal will however allow $y(t)$ to track $r(t)$:

$$u(t) = w(t)\left(\frac{-(y(t)y(t-1)y(t-2)u(t-1)[y(t-2)+1]}{w(t)} + r(t)\right), \tag{3-10}$$

$$\textit{where} \quad w(t) = 1 + y(t-1)^2 + y(t-2)^2 .$$

Again, a regression equation for the controlled system is equation (3-10) with $r(t)$ replaced by $y(t)$. For consistency, observations $u^F(t)$, $y^F(t)$ and input $v(t)$ will be used instead of $u(t)$, $y(t)$ and $r(t)$. The maximum order of the system $n = 3$ is assumed to be known. The observer $\alpha(q) = q(q - 0.03)(q - 0.05)$ was chosen. Also the degree of $B(q)$ is assumed to be 2. Hence the reference model $(q - 0.75)y(t) = v(t)$ of degree (3 - 2) was chosen. The BNN input vector at time t is

$$\Phi(t) = \begin{bmatrix} u^F(t-1) \\ u^F(t-2) \\ u^F(t-3) \\ y^F(t-1) \\ y^F(t-2) \\ y^F(t-3) \\ (1-0.75q^{-1})y(t) \end{bmatrix}, \tag{3-11}$$

where $u^F(t) = q^{-3}\alpha(q)u(t-1)$ and $y^F(t) = q^{-3}\alpha(q)y(t-1)$. The desired output is $u(t-1)$. The same training and testing procedures as in first experiment were adopted. The BNN estimator was first trained off-line and tested as before. The result is shown in Figure 18.

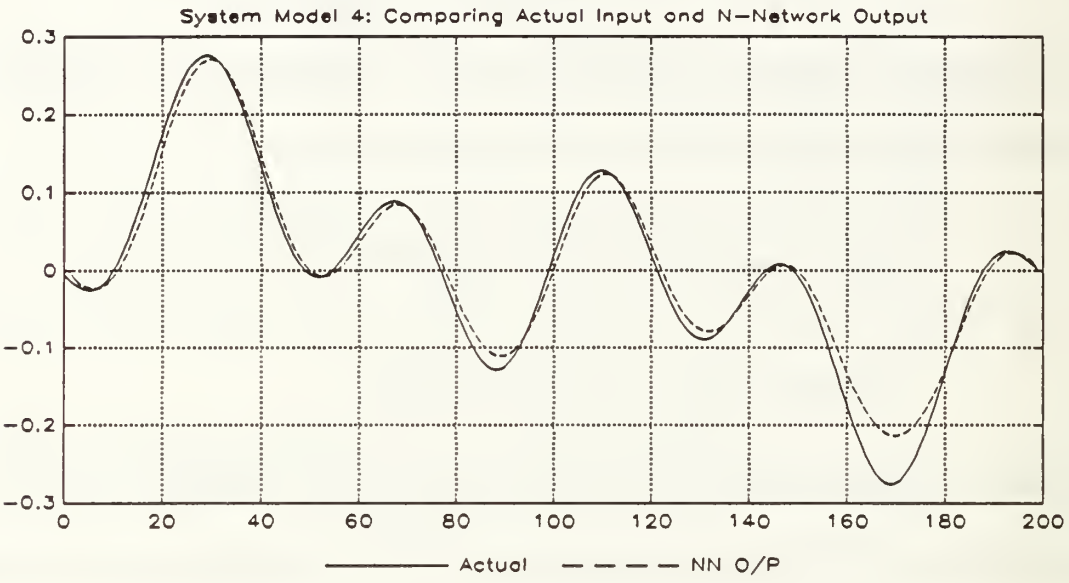

Figure 18: Test Result for Off-line trained BNN DMRAC.

Then with on-line control, the result of one such experiment is shown in Figure 19. Even though the BNN in this case did not seem to be adequately trained off-line, it was sufficient for the on-line control to work. Other experiments with various inputs again showed that the BNN DMRAC performs reasonably well for this class of systems. However, if the reference input $v(t)$ has high frequency components, the BNN could not track the reference model. This problem and possible solutions have been discussed earlier.

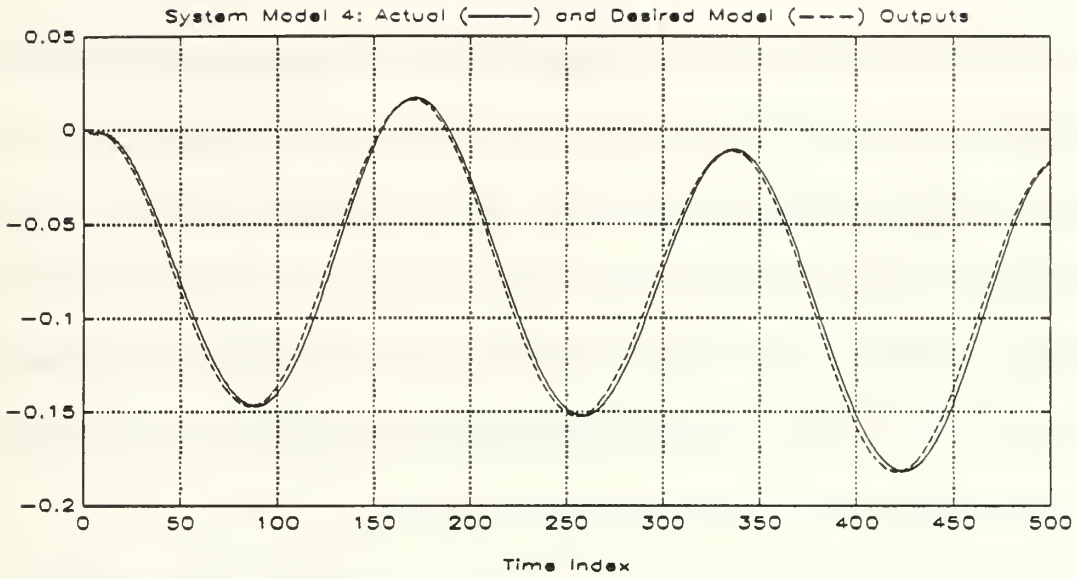

Figure 19: On-line Control. System Model 4.

B. OBSERVATIONS AND DISCUSSIONS

In this section, some important observations on the controller, the design and implementation in the experiments are discussed. In general, we observed that the BNN DMRAC developed in Chapter II works well under certain conditions. These are described in detail in this section. Although, the generality of these conditions cannot yet be fully established due to the lack of analysis techniques for these nonlinear systems and neural network, they have served to developed fairly good controllers for the experiments. Often the BNN MRAC works very well with sufficient training and careful tuning of the learning parameters.

1. Failure of Least-Squares Estimator DMRAC

In each case, a DMRAC was designed with a least-squares estimator assuming the unknown system is LTI. Except for system in the second experiment, all the linear system DMRAC failed to work. Therefore, the BNN DMRAC is an effective technique in controlling these unknown nonlinear systems which the conventional adaptive control technique cannot handle. The BNN can be viewed as a generalized estimator which performs the nonlinear estimation and hence allows the adaptive control technique to be extended to cover large classes of nonlinear systems.

2. Some General Requirements for the System

Two conditions required for success of the BNN DMRAC are postulated from the analysis and the experiments. Like the DMRAC designed for unknown LTI systems, the BNN DMRAC will only work with systems which have a minimum phase property, or equivalently[7], the stability of the systems given by the regression equations. It is seen that estimator and the BNN have to learn from the regression system. If it is unstable, it is obvious that effective learning cannot take place, in particular, with a BNN.

Another important condition governing the inverse of the nonlinear system is also postulated: the regression system must be unique. Only then will the BNN be able to learn the mapping consistently.

---

[7] Minimum phase property applies only to linear systems.

3. Assumed Orders of the System

The control system still worked when higher orders were assumed for the system in the controller design. This was illustrated in experiment 3. A higher order controller (and observer) would however entail more inputs, and hence a larger BNN. A larger BNN typically requires more training, hence a longer training period. On the other hand, in situations where the orders of the system are uncertain, sufficiently high orders can always be chosen such that they exceed the actual system orders.

4. Stability of Open Loop System

Since off-line training requires the system to be operated in the open-loop, this procedure cannot be recommended for open-loop unstable systems. There is another reason why the technique should not be used for an open-loop unstable system. With unknown and unstable LTI systems, it is still possible to design a stabilizing DMRAC. However, the control effort required may be very large in order to stabilize the system (especially during starting-up). In the BNN, the output is limited to the range ± 1, since the hyperbolic tangent function is employed in each neuron. So, this may limit the required control input needed to achieve stabilization. To overcome this problem, a linear function for the output neuron was used, thus avoiding the saturation limit of the BNN output. However, with linear output neuron, it was found that the stability of the learning algorithm became difficult to maintain. More analysis and experimenting will be required to explore this approach.

5. System Input and Output Scaling

Since the BNN output is limited to the range $\pm$ 1, the desired output of the BNN under training must be limited to the same range. Therefore scaling of the training input *u(t)* for the system (i.e. the BNN output) during off-line or on-line training is always required so that the limits are not exceeded. Consider next the scaling of the BNN input vector $\Phi(t)$. In the LMS algorithm, it has been shown [Ref.11] that the step-size parameter must be in the range between 0 and $2/\lambda_{max}$, where λ_{max} is the largest eigenvalue of the correlation matrix of the input vector, to ensure stability of the LMS learning algorithm. By appropriately scaling the input vector, it is possible to use the range between 0 and 1 for the step-size parameter. Drawing a parallel for the BNN training, it is necessary that the input vector be scaled to a certain range in order to keep the learning system stable and the learning rate μ, in the range 0 to 1.

In all the experiments, the input *u(t)* used to generate data for off-line training was always scaled so that it spanned the range ± 0.8. In addition, the norm of the BNN input vector ($\Phi(t)$) for learning was limited to < 1. If not, then the input *u(t)* was further scaled so that the last condition could be met. This seemed to prevent the BNN from learning instability in all the experiments, at least, as long as the learning rate μ is kept between 0 to 1.

The typical operating ranges of a system may however be higher than the operating ranges used in these simulations. However, this problem can be overcome by scaling if the normal operating ranges of such a system are known.

C. BNN DESIGN AND TRAINING

The issues discussed so far relate primarily to the control system design and the required conditions under which the BNN DMRAC can be successfully applied. Although the input and output scaling issues have been addressed, there are still many important questions related specifically to the design and training of the BNN. This section discusses some of these issues: the design parameters of the BNN used as a DMRAC, the adequacy of training and the training regimes. The implementation of the BNN software simulator is first discussed to provide more detailed background for subsequent discussions.

1. Implementation of the BNN Software Simulator

For this thesis research, a software BNN simulator was developed. Until neural network hardware systems or neurocomputers become commonly (and economically) available, most researchers will work with software simulators for neural networks. It is fairly simple to emulate the actions of a neuron and an entire neural network in software. The software approach offers the full flexibility for development, allowing the user to exercise and experiment freely with the various features of the neural network. The main drawback of this approach is the slowness of the simulator during the learning process.

The BNN simulator used is built from a collection of functions developed in the form of a MATLAB toolbox. The processing required in the BNN can be easily represented by vector and matrix operations. Hence MATLAB, a high-level programming

environment with built-in matrix operators, is ideally suited as a development platform for the BNN software simulator. Another advantage in using MATLAB is that it comes with a Control Toolbox which fully supports the simulations of discrete-time system.

The neural network toolbox developed consists of the following functions:

- NET2 and NET3. These functions set up the data structure for a 2 and 3 layer back-propagation neural network. It takes in a parameter describing the number of inputs and neurons in the hidden and output layers. It also takes one other parameter specifying the spreading range of the biased input (this feature shall be explained later).

- RECALL2 and RECALL3. These are functions which calculate the output vectors for a 2-layered and 3-layered neural network given an input vector.

- LEARN2 and LEARN3. These are learning functions for the 2-layered and 3-layered neural networks respectively. By presenting a desired output vector and the actual neural network output vector, these function updates the synaptic weights according to the learning rules described in Section B of Chapter II.

- BKPROP2 and BKPROP3. These functions back-propagate the output errors to the input layer. They are not used in the simulations conducted in this thesis research.

- Other miscellaneous functions including SHUFFLE which randomly shuffles a set of indices for use in the BNN learning procedure.

The source codes for the toolbox implementations of the functions used in this thesis research are provided in Appendix D. These MATLAB programs are self-documented so that they can be used without any other documentation.

2. Design of the BNN

In using a BNN as a DMRAC, the following design parameters must be determined or specified:

- The number of inputs: This is simply determined by the order n assumed for the nonlinear system. The number of inputs required is $(2n - 1)$, which is the size of the input vector $\Phi(t)$ or $\Phi_e(t)$ in equation (1-11) or equation (1-15).

- The number of layers: There is no hard and fast rule in determining the number of layers for the BNN. In general, we found that a 2-layer network is adequate for the control of simple low order models such as those used in the simulations. With more layers, considerably more training is required for the BNN during the off-line learning phase. Therefore, excessive layers should be avoided wherever possible.

- The number of neurons: The number of neurons in the output layer is of course determined by the number of outputs. Since the research here deals with SISO systems, only one output neuron is required in all cases. The choice of the number of neurons required in each hidden layer is another grey area. In our experiments, using 2-layer BNN, the number of hidden neurons taken to be 3-4 times the number of inputs seemed to work adequately.

- The nonlinear mapping: The choice of nonlinear mapping for the neuron depends on applications. In control system design, the control input is commonly in the range $\pm R$, where R a real number. This requires the output neurons of the controller to match this range. Therefore an odd symmetric function is suitable for the neuron. In the BNN simulator used, the hyperbolic tangent function ($\tanh[\cdot]$) is used in all neurons.

3. Adequacy of Training

In off-line training of the neural networks, it is important to determine the adequacy of training in order to decide if the training can be terminated. There are several ways to ensure that a BNN has been trained adequately. One way is to calculate the mean square output errors over a reasonable time window. Since there is no absolute level (except that it should be 'small') for this value to indicate if the BNN is trained sufficiently, the mean square error at different training cycles can computed and compared. When the error becomes almost constant, assuming that the training set has been carefully chosen and the BNN has been adequately designed, then the learning cycle

can be considered as done. The BNN is said to have converged. Another way is to check the singular value decomposition (SVD) of the synaptic weight matrices $W^{[1]}$, $W^{[2]}$, ... When the SVD of these matrices remain almost constant, the learning cycle can usually be considered as done.

4. Neural Network Training

The most important factors affecting the BNN learning are

1. the ranges of the inputs and desired output values used in training,
2. the characteristics and the size of the training set, and the number of passes through the training sets, and
3. the learning regime.

The first item has already been discussed. The second item concerns itself mainly with how well the BNN converges. In general, the larger the training set, the better the convergence will be. Here, the amount of training time to be expended or the number of available training examples collected limits of the size of the training set. What is more important though is the characteristics of the training examples in the training set. Drawing a parallel from adaptive control theory, the condition of "persistent excitation" must exist in a training set. Loosely speaking, the training set must excites all the system modes under normal operating conditions. This then allows the BNN to learn to emulate the required regression form thoroughly. The off-line training for all simulations conducted here used $u(t)$ with 200 to 500 data-points to generate the training set. $u(t)$ is

a sum of at least 3 sinusoids with different magnitudes and phases, each with a small random varying phase component. In terms of persistency of excitation, this seemed adequate. The use of uniformly distributed random white noise sequences were also quite found to be adequate. The small training set also seemed adequate in all the experiments when each training example was repeatedly presented (at different times and in random order, of course) to the BNN. This randomized presentation sequence has been observed to help the BNN learning to converge much faster. In most cases, 20 to 30 repeated passes through the entire 200-point training sets were adequate to allow the trained BNN to produces $\hat{u}(t)$ that was very close to $u(t)$.

The selection of learning rates and momentum rates as a function of training cycles for different layers of the BNN, make up the training regime. The training regimes is still an important area for more research. This is the area where a lot of experimentation is required due to the lack of strict rules and guidelines to ensure (1) stability of the learning process and (2) good and fast convergence. The training regime determines strongly how fast the learning process proceeds. The learning rates and momentum rates can be separately assigned for each layer. For simplicity, they were usually kept identical in our experiments. The following two rules of thumb generally used by many neural network researchers were adopted in the learning process. One, the learning rate should be decreased as the number of training cycles increase. Two, larger momentum rate should be used in the early phases of training and then lowered for the final phase of training. In the experiments, learning rates typically > 0.8 and momentum rates > 0.4 were chosen for the first 5,000 training cycles in off-line training. Then they were

normally set to < 0.6 for the rest of the training cycles. During on-line learning, the momentum term was usually set to < 0.2 because it is expected that the off-line trained BNN would require only minor adjustment in its synaptic weights with further on-line training. The techniques used in checking for convergence of off-line learning can also be used to dynamically adjust the learning rates during on-line training. For example, the average rate of change of the SVD's of the weight matrices can be used as a guide in setting the learning schedule: when the average rate is, say half of the initial value, a smaller learning rate is switched in.

There is a lot of experimenting involved in the selection of both the learning rates and momentum rates. Hence, the training process will benefit greatly with the development of stricter rules regarding the selection of these parameters.

5. Setting the Bias Inputs

Each neuron has a bias input. In the BNN software simulator, this input can set to zero. However, a BNN with zero bias inputs for all the neurons can only output zero when its input vector is $\mathbf{0}$. Therefore, this BNN can only emulate functions which the property $f(0) = \mathbf{0}$. This form of neural network is usually not sufficiently general for use as a DMRAC for nonlinear systems. Hence fixed non-zero bias inputs are used. They are fixed by spreading the synaptic weights associated with the bias inputs across a range, $\pm R$. R is normally selected to between 1 to 2. These weights are kept unchanged even during learning while the bias inputs are set to 1. This feature has been incorporated into the BNN simulator by specifying the spread range R during the initialization of the

network. The bias of the output neuron (since there is only one output neuron in our BNN DMRAC) is kept at a fixed value $+R$.

Alternatively, fixed bias inputs can be used while allowing the associated synaptic weights to vary in the learning process. However, it was found that considerably more off-line training were needed for convergence using this approach. Convergence was also difficult to achieve and the mean square errors of the BNN output and the desired output tended to change erratically between different passes during the on-line training with different reference inputs.

IV. CONCLUSIONS

A. SUMMARY

Starting with the development of a direct model reference adaptive controller for LTI systems with unknown parameters, the basic structure for a neural network-based adaptive controller was advocated. The DMRAC for LTI systems was extended to nonlinear systems by training a BNN to emulate a suitable nonlinear regression form that describes the system under consideration.

The control of four general classes of unknown nonlinear systems, modelled in discrete-time, using the BNN DMRAC was considered. The specific structure for the BNN DMRAC of these four classes of systems was developed.

B. IMPORTANT RESULTS

Experiments in BNN-based adaptive control were conducted using four specific examples of nonlinear systems, belonging to four different classes of systems. The main observations from these experiments are summarized below:

(1) The results indicated that BNN DMRAC works well in the control of these unknown nonlinear systems. It was also seen that in most cases, the standard least-squares estimator DMRAC designed using LTI assumption failed to work for these systems.

(2) The design approach is quite specific as far as the controller structure is concerned. The general conditions for successful application of BNN-based DMRAC can easily be satisfied.

(3) Off-line training of the BNN is required. The amount of off-line training required is quite insignificant. The system is required to be open-loop stable.

(4) The performance of a trained control system depends somewhat on the inputs used. For inputs with high frequency contents with respect to the sampling rate, the controlled system tends to become quite oscillatory and does not track the reference model well. Some solutions were proposed for this problem.

(5) The BNN training does requires significant attention and experimentation. No firm rules are yet available for the training regimes, the scaling of the inputs and output, and the use of bias inputs. Any breakthrough in the development of general analysis techniques to help establish conditions for stability of both the BNN learning system and the closed loop system will significantly boost the usefulness of this technique.

The general requirements for the unknown nonlinear systems to ensure the success of the BNN DMRAC are postulated from the analysis and observations made in the experiments:

- A suitable control structure for which a BNN can emulate must exist. For the BNN to be able to emulate this nonlinear controller, the functional mapping of the controller must be continuous (such that a BNN can emulate this controller).

- In addition, the system should be equivalently minimum phase, in the sense that the regression form which the BNN learns to emulate must be stable and unique.

- The open-loop nonlinear system should be inherently stable for the off-line training.

- The orders (n and m) of the system, or at least their lower bounds, must be known.

C. FURTHER RESEARCH AND DEVELOPMENT

In this thesis research, the emphasis was to develop a structure for direct adaptive control of certain classes for unknown nonlinear systems using the BNN. The results of the experiments showed clearly that the BNN DMRAC in the proposed form can work, at least for systems similar to those considered in the experiments. Some general conditions required of the nonlinear systems have also been postulated. The general guidelines used to keep the learning algorithm stable and the convergence fast, worked well in the experiments. However, more exact conditions governing the successful employment of the BNN DMRAC, the stability of the closed system, and stricter rules on the choices of the parameters in the BNN design (e.g. the number of hidden layers, number of neurons, etc.) should be established.

1. Stability Conditions for the BNN DMRAC

Development of sufficient conditions to establish the stability of the BNN DMRAC controlled system is the most critical aspect required for the acceptance of this technique for real-world applications. However, it will definitely entail much more research since there are currently no established analytical tools available for use in these nonlinear systems and neural networks. Furthermore, the stability of the closed loop control system is affected not only by the open loop nonlinear system and the BNN design, but also the types and operating range of the input and the training regimes employed.

2. Design of the BNN

In the design of the BNN, the selection of the number of layers, neurons, the type of nonlinear transformation, etc., is still very much an art. This area is definitely needs further research. Stricter rules governing the choices of number of layers, the number of neurons in each layers, the use of bias inputs and even the appropriate choice of nonlinear function for the neuron should be generated. The correct design should allow the BNN to fully emulate the appropriate unknown nonlinear functions, such that it will work with the full range of the required control input.

3. BNN Learning

The selection of appropriate training data and the learning schedule, which determine the goodness and speed of convergence are important aspects of this technique. Stricter rules governing the characteristics of training data and the establishment of a learning schedule will greatly benefit this technique. Specifically, rules should be developed to determine the required 'persistency' in the excitation of the training data. Also the learning schedule should be related to the convergence rate.

APPENDIX A. DMRAC DESIGN FOR UNKNOWN LTI SYSTEMS

A direct model-reference adaptive controller was designed and implemented to control an unknown linear-time-invariant system. The design approach and the structure of the controller was analyzed in Section D of Chapter I. The recursive least-square estimation technique shall be used for control parameter estimation.

The unknown LTI system to be controlled is described by the difference equation:

$$y(t) - 0.2y(t-1) + 0.9y(t-2) = 3u(t-1) . \qquad \text{(A-1)}$$

Defining q as the forward time shift operator, then

$$A(q) = q^2 - 0.2q + 0.9 ,$$

$$B(q) = 3q .$$

This system has two poles at Ps_1 and Ps_2 obtained by setting $1 - 0.2z^{-1} + 0.9z^{-2} = 0$ which gives $Ps_{1,2} = 0.1 \pm 0.9434i$ and a zero at $Zs_1 = 0$ by setting $3z^{-1} = 0$. Suppose the closed loop system is required to track the reference model

$$y_m(t) - 0.8y_m(t-1) = v(t-1) . \qquad \text{(A-2)}$$

$D(q) = (q - 0.8)$. It has one pole, $Pm_1 = 0.8$.

If the system is known, then by state-feedback, one pole of the closed loop system must be placed at Pm_1. The remaining pole must therefore be placed at Zs_1, so that it is

cancelled by the closed loop zero. Hence the desired characteristics polynomial of the closed loop system shall be

$$p^*(q) = (q - Pm_1)(q - Zs_1) = (q - 0.8)q \ . \tag{A-3}$$

The state-space equation of the system is:

$$\begin{pmatrix} x(t+1) \\ x(t) \end{pmatrix} = \begin{pmatrix} 0.2 & -0.9 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x(t) \\ x(t-1) \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{A-4}$$

and

$$y(t) = (1 \quad 0) \begin{pmatrix} x(t) \\ x(t-1) \end{pmatrix}, \tag{A-5}$$

or more compactly,

$$x(t+1) = \Phi \cdot x(t) + \Gamma \cdot u(t)$$

$$y(t) = C \cdot x(t) \ .$$

First assume all states are accessible. Using state-feedback, $u(t) = -L \cdot x(t) + v(t)$, L the feedback gain, the closed loop system is

$$\begin{aligned} x(t+1) &= [\Phi - \Gamma L]x(t) + v(t) \\ y(t) &= C\ x(t) \ . \end{aligned} \tag{A-6}$$

L can be obtained by equating the characteristic polynomial of the closed loop system function derived from (A-6) to the desired characteristic polynomial in equation (A-3) giving

$$L = [-0.6 \ -0.9].$$

Suppose only $u(t)$ and $y(t)$ are accessible while the states $\boldsymbol{x}(t)$ are not, then an observer is needed. The estimated states $\hat{\boldsymbol{x}}(t)$ is then used for state feedback

$$u(t) = \boldsymbol{L}\,\hat{\boldsymbol{x}}(t) + v(t) \ . \tag{A-7}$$

A Luenberger or steady-state Kalman filter observer is given by

$$\hat{\boldsymbol{x}}(t+1) = \Phi\,\hat{\boldsymbol{x}}(t) + \Gamma\,u(t) + \boldsymbol{K}\,[y(t) - \boldsymbol{C}\,\hat{\boldsymbol{x}}(t)] \ , \tag{A-8}$$

where $\boldsymbol{K}$ is the observer gain. In the Luenberger observer, the observer gain $\boldsymbol{K}_l$ can be obtained by first designating the observer pole locations, $Po_{1,2}$ (often chosen so that the observer dynamics is approximately four times faster than the system, if that is known). Then $\boldsymbol{K}_l$ is obtained by equating the characteristics polynomial of $(\Phi - \boldsymbol{K}_l \cdot \boldsymbol{C})$ to $(q\text{-}Po_1)(q\text{-}Po_2)$. Choosing $Po_{1,2} = |\,Ps_{1,2}\,|^4\,\underline{/Ps}_{1,2}$, the resulting observer gain is $\boldsymbol{K}_l = [0.0097\ 0.0903]'$.

For the Kalman filter, the covariance matrices $\boldsymbol{Q}$ and R of the state disturbances and output measurement noise must first be specified. Then $\boldsymbol{K}_k$, the steady-state Kalman filter gain is obtained by solving the arithmetic Riccati equation. Choosing $\boldsymbol{Q} = \boldsymbol{I}$ and $R = 1$, $\boldsymbol{K}_k = [0.0033\ -0.3492]^T$.

From equations (A-7) and (A-8), the following equation is obtained:

$$\begin{aligned} u(t+1) = {} & \boldsymbol{L}\,[q\boldsymbol{I} - \Phi + \boldsymbol{K}\,\boldsymbol{C}]^{-1}\Gamma\,u(t) + \\ & \boldsymbol{L}\,[q\boldsymbol{I} - \Phi + \boldsymbol{K}\,\boldsymbol{C}]^{-1}\boldsymbol{K}\,y(t) + v(t) \ . \end{aligned} \tag{A-9}$$

This controller hence has the following structure:

$$u(t) = \frac{h(q)}{\alpha(q)}u(t) + \frac{k(q)}{\alpha(q)}y(t) + v(t) \ , \qquad \text{(A-10)}$$

where $\alpha(q)$ is the characteristic polynomial of the observer. It can be chosen using the Luenberger or steady-state Kalman filter observers.

Using the partial state $z(t)$, equation (A-1) can be written as follows:

$$A(q)z(t) = u(t) \ ,$$

$$y(t) = B(q)z(t) \ .$$

Combining the above with (A-10), we have equation (1-5). To obtain the required closed loop behaviour, we set

$$\alpha(q)A(q) - h(q)A(q) - k(q)B(q) = \alpha(q)p^*(q) \ , \qquad \text{(A-11)}$$

so that $p^*(q)y(t) = B(q)v(t)$ becomes the reference model by choosing $p^*(q) = 1/b_1$ and $D(q)B(q) = (z - Pm_1)(z - Zs_1)$.

Re-arranging equation (A-11), we get

$$h(q)A(q) + k(q)B(q) = \alpha(q)\ [\ A(q) - p^*(q)] \ . \qquad \text{(A-12)}$$

Equation (A-12) is the Diophantine equation. Since $A(q)$ and $B(q)$ are relatively co-prime (i.e. no pole-zero cancellation), a solution for $h(q)$ and $k(q)$ is guaranteed. $h(q)$ and $k(q)$ are solved by forming the Sylvseter matrix using the two different observers, the Luenberger and the Kalman filter observers. The two characteristics polynomials $\alpha_l(q)$ and $\alpha_k(q)$ are:

$$\alpha_l(q) = q^2 - 0.1708q + 0.6561 \ ,$$

$$\alpha_k(q) = q^2 - 0.19q + 1.8427 \ .$$

Solving the equation (A-12), the results in Table A-1 are obtained for the unknown polynomials $h(q) = h_1q + h_2$ and $k(q) = k_1q + k_2$.

	Luenberger observer	Kalman observer
h1	0.6	0.6
h2	0.6561	1.8427
k1	0.0871	-0.3123
k2	-0.0563	0.25 4

TABLE A.1: Coefficients of h(q) and k(q)

The complete computer solution for $h(q)$ and $k(q)$ are logged below below:

System y(t) - 0.2y(t-1) + 0.9y(t-2) = 3u(t-1)

Aq = 1.0000 -0.2000 0.9000

Bq = 3 0

Poles of the system

Poles =
0.1000 + 0.9434i
0.1000 - 0.9434i

The state-space representation of the system

Phi =
0.2000 -0.9000
1.0000 0

Gma =
1
0

C =
3 0

D =
0

(a) Estimated State-Feedback Approaches

Luenberger Observer Poles

ObsPoles =
0.0854 + 0.8055i
0.0854 - 0.8055i

The Luenberger observer gain Kl

Kl =
0.0097
0.0903

Kalman filter observer: Choose the noise covariance matrices

Q =
1 0
0 1

R =
1

The steady-state Kalman observer gain Kk

Kk =
0.0033
-0.3492

```
Desired system (Reference model): D(q)ym(t) = v(t)

D =
   1.0000   -0.8000

Desired Closed Loop Poles

DsrPoles =
   0.8000
        0

System Feedback Gain L
place: ndigits= 16

L =
  -0.6000   -0.9000

(b) Diophantine Approach
Form the Sylvester matrix

Ms =
   1.0000         0         0         0
  -0.2000    1.0000    3.0000         0
   0.9000   -0.2000         0    3.0000
        0    0.9000         0         0


Desired system behaviour: p*(q)

Pstar =
   1.0000   -0.8000         0

Observer characteristic polynomials: α(q)
For the Luenberger Observer

Alphal =
   1.0000   -0.1708    0.6561

Fl =
   0.6000
   0.7975
   0.2400
   0.5905

For the Kalman Observer

KObsPoles =
  0.0950 + 1.3541i
  0.0950 - 1.3541i


Alphak =
   1.0000   -0.1900    1.8427


Fk =
   0.6000
   0.7860
   0.9346
   1.6585

Controller parameters: h(q) and k(q)
For the Luenberger Observer

hkl =
   0.6000
   0.6561
   0.0871
  -0.0563

For the Kalman Observer

hkk =
   0.6000
   1.8427
  -0.3123
   0.2544
```

Since $A(q)$ and $B(q)$ are actually unknown, an estimator is required. Applying the polynomial in equation (A-12) to z(t), the partial state, the following system is obtained:

$$\alpha(q)u(t) = h(q)u(t) + k(q)y(t) + \frac{\alpha(q)D(q)}{b_1}y(t) \; . \qquad \text{(A-13)}$$

In this case, equation (A-13) can be written as

$$u(t) = \frac{h_1 + h_2 q^{-1}}{\alpha_1 + \alpha_2 q^{-1} + \alpha_3 q^{-2}} q^{-1}u(t) + \frac{k_1 + k_2 q^{-1}}{\alpha_1 + \alpha_2 q^{-1} + \alpha_3 q^{-2}} q^{-1}y(t) + \frac{D(q)}{b_1}y(t) \; . \qquad \text{(A-14)}$$

Setting,

$$(\alpha_1 + \alpha_2 q^{-1} + \alpha_3)y^F(t) = y(t-1),$$

$$(\alpha_1 + \alpha_2 q^{-1} + \alpha_3)u^F(t) = u(t-1),$$

equation (A-14) becomes

$$u(t-1) = \begin{pmatrix} y^F(t-1) & y^F(t-2) & u^F(t-1) & u^F(t-2) & q^{-1}D(q)y(t) \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ k_1 \\ k_2 \\ 1/b_1 \end{pmatrix} \; . \qquad \text{(A-15)}$$

Based on the above regression equation (A-15) above and using least-squares estimation technique, the recursive estimate for $\boldsymbol{\Theta}$ the unknown parameter vector in equation (A-15)

can be obtained using equations (1-11) and (1-12). Then using the estimated parameter vector, control can be effected using equation (A-10).

Using MATLAB, the DMRAC with a least squares estimator developed above is implemented. The programs for all the experiments conducted with the DMRAC are attached at the end of this Appendix.

Experimental Results and Comments:

Figures A.1 to A.6 show that the adaptive controller managed to recursively converge very quickly to the correct values as that obtained in solving the Diophantine equation (assuming the plant is known). The observer characteristics polynomial $\alpha_l(q)$ chosen in the first two cases is based on the Luenberger observer. In the next two cases, the characteristics polynomial $\alpha_k(q)$ is based on the Kalman filter. The convergence is quite independent of the input *v(t)*. Many different frequencies were tried. Figure A.1 and A.3 shows the use of a high frequency *v(t)* while Figure A.2 and A.4 show the use of a low frequency *v(t)*. As the system is linear-time invariant, we can stop the adaptation after a while. The result of model following control is close to perfect even without further adaptation.

Using a sinusoidal *v(t)* instead of the square wave, Figure A.5 and A.6 shows almost identical results for the control parameters after a few iterations. Convergence of the parameters does take a little longer in these cases.

P(0) has been chosen in the recursive least squares algorithm to be 1000***I**. With smaller values of *P(0)*, slower convergence of the parameters is observed.

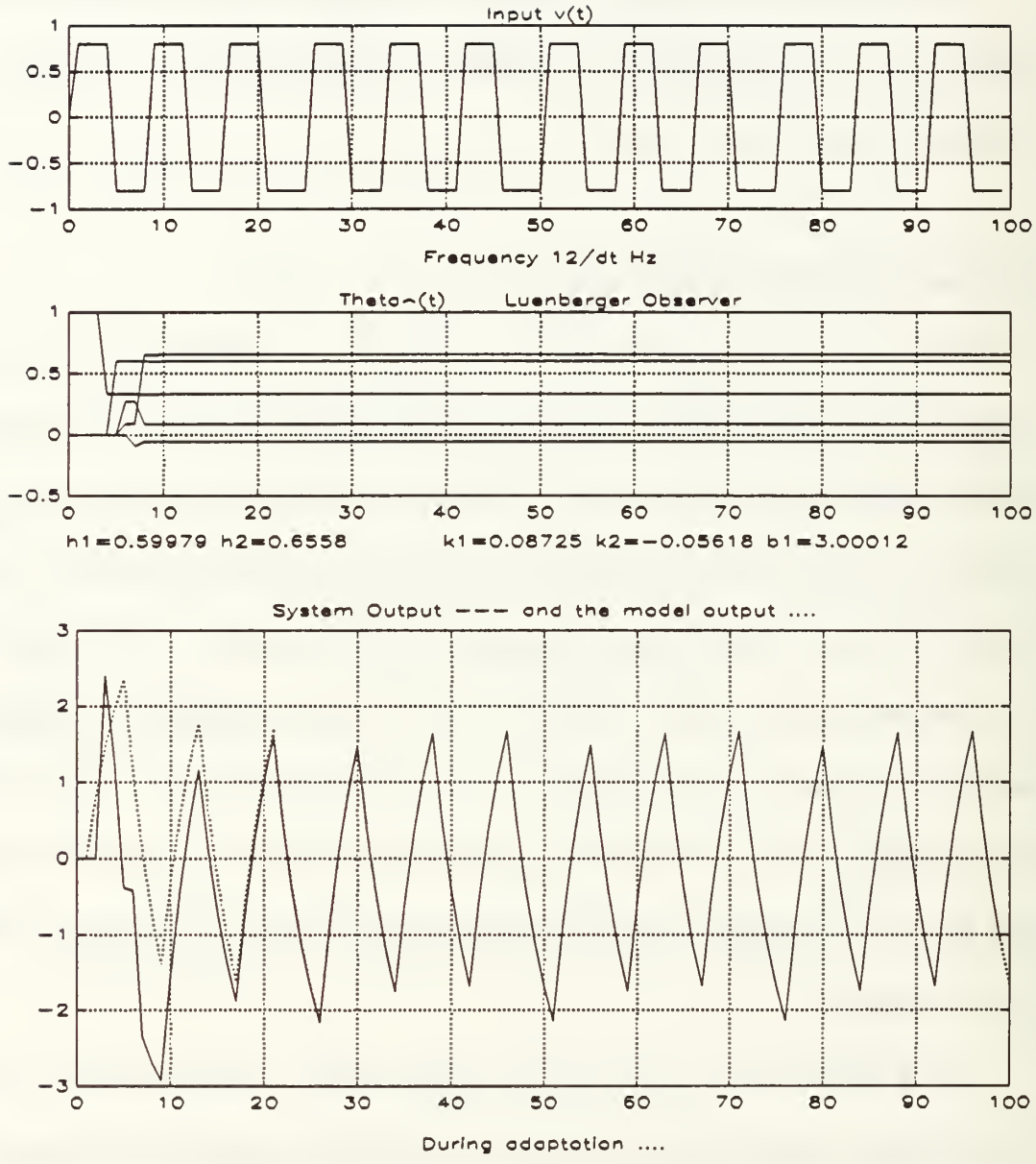

Figure A-1: Least-Squares Estimator DMRAC Experiment 1

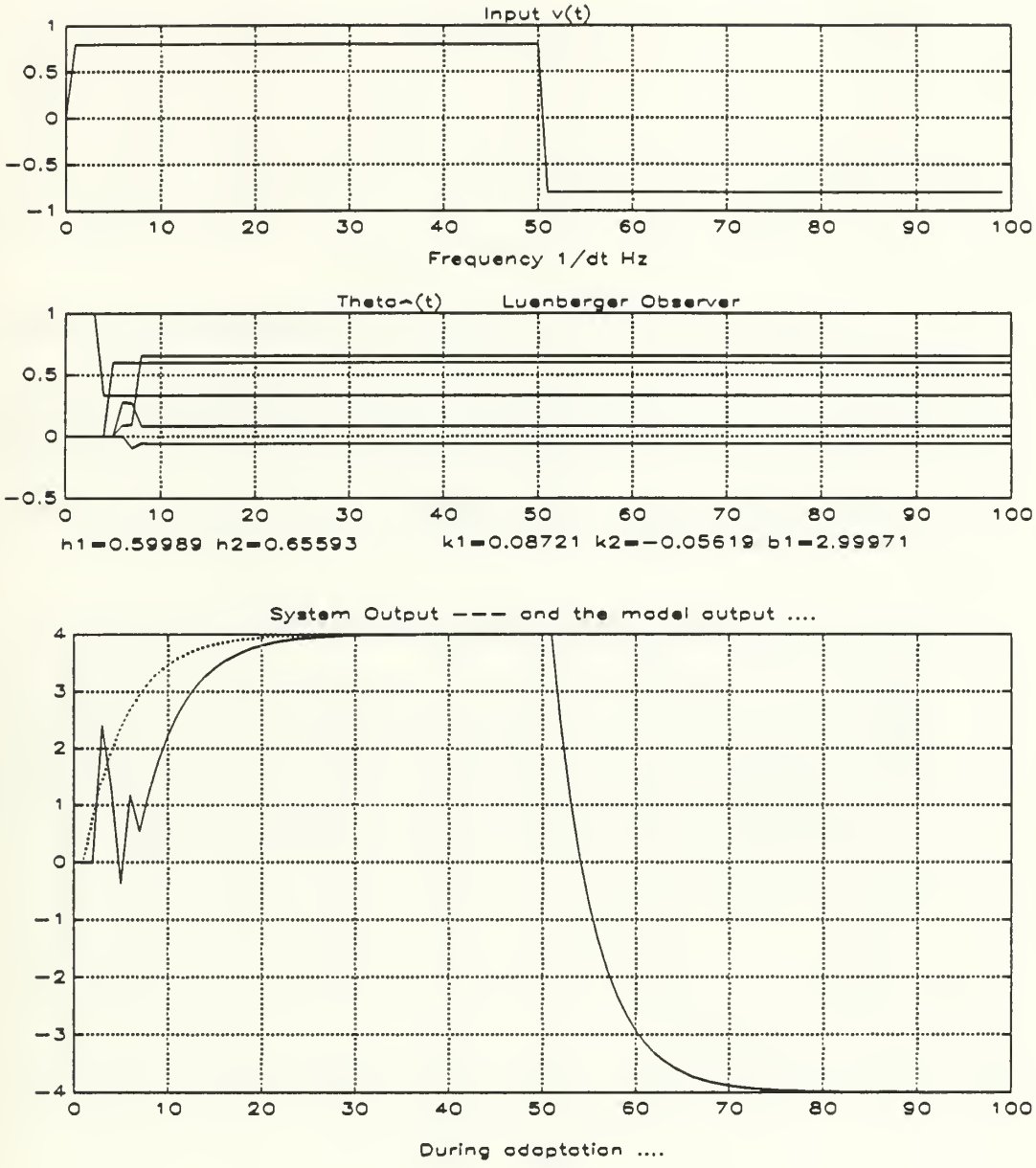

Figure A-2: Least-Squares Estimator DMRAC Experiment 2

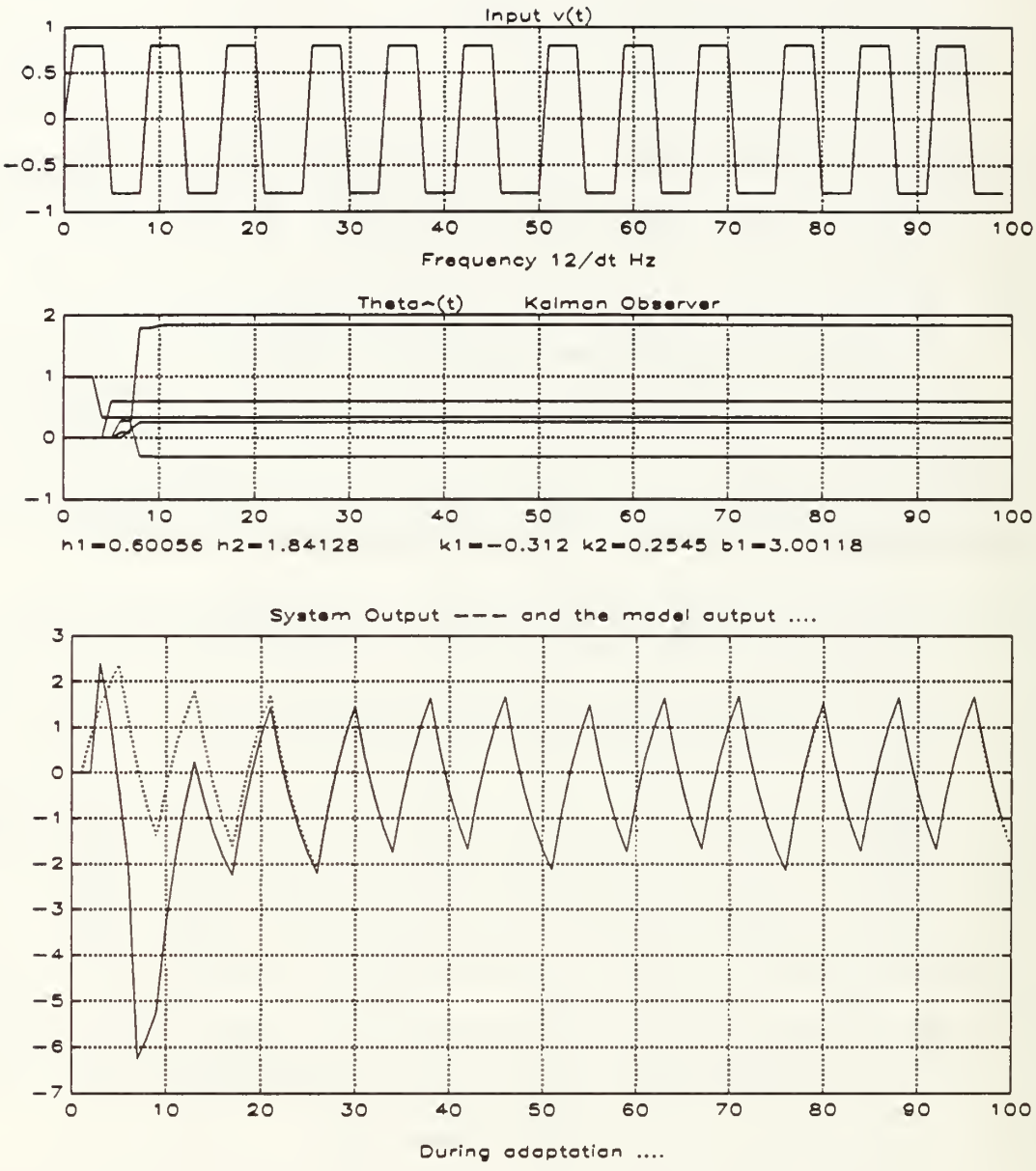

Figure A-3: Least-Squares Estimator DMRAC Experiment 3

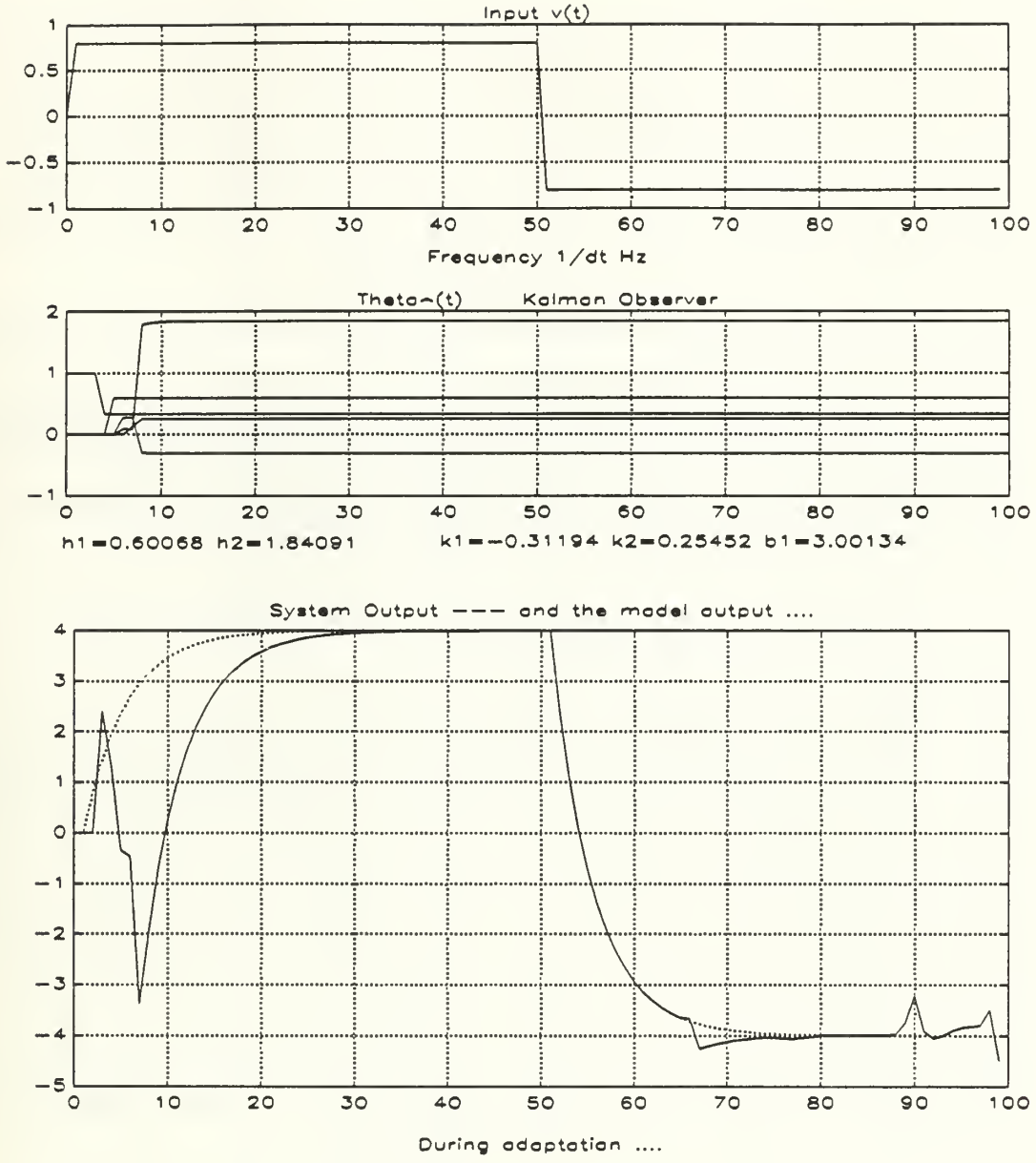

Figure A-4: Least-Squares Estimator DMRAC Experiment 4

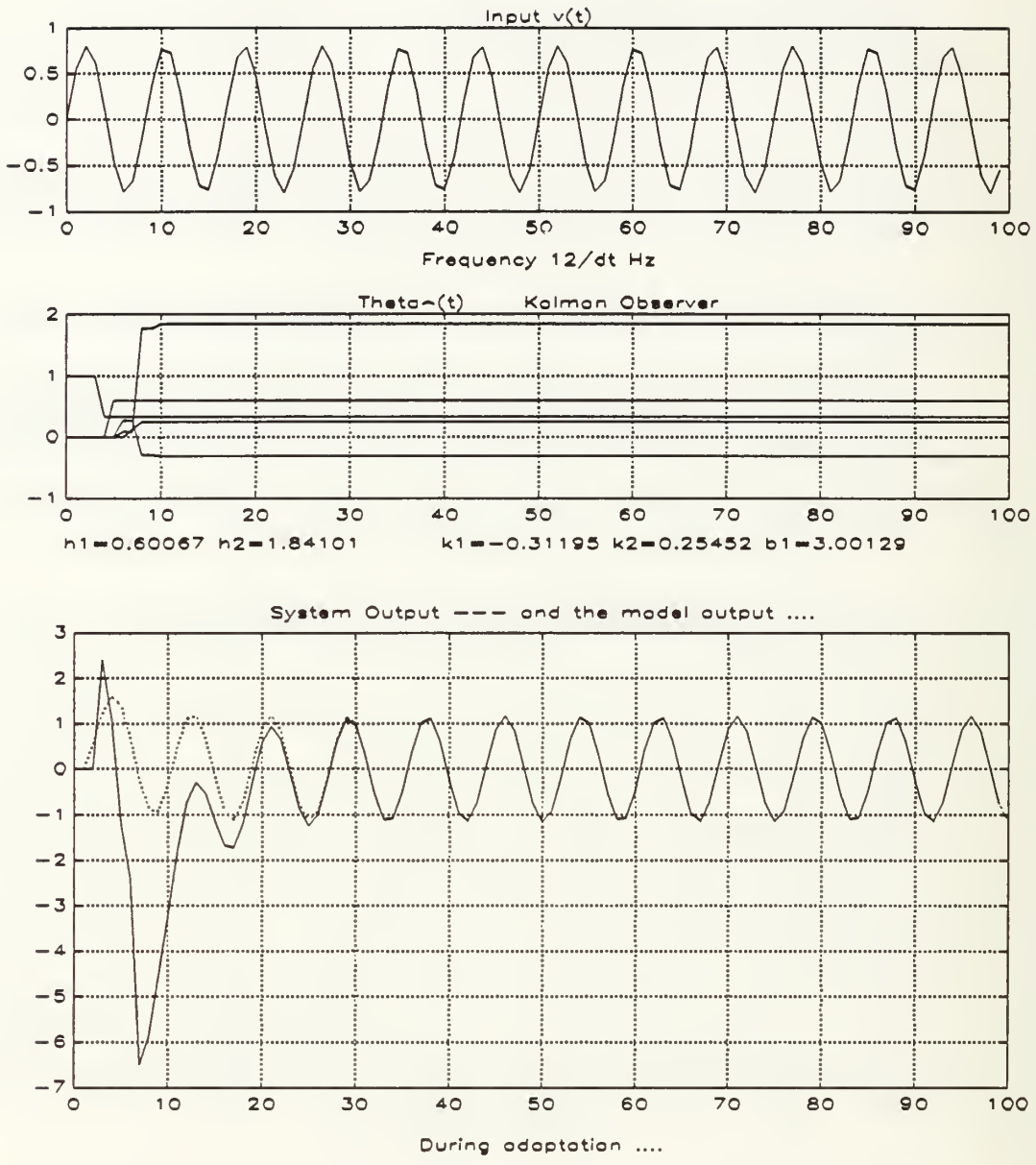

Figure A-5: Least-Squares Estimator DMRAC Experiment 5

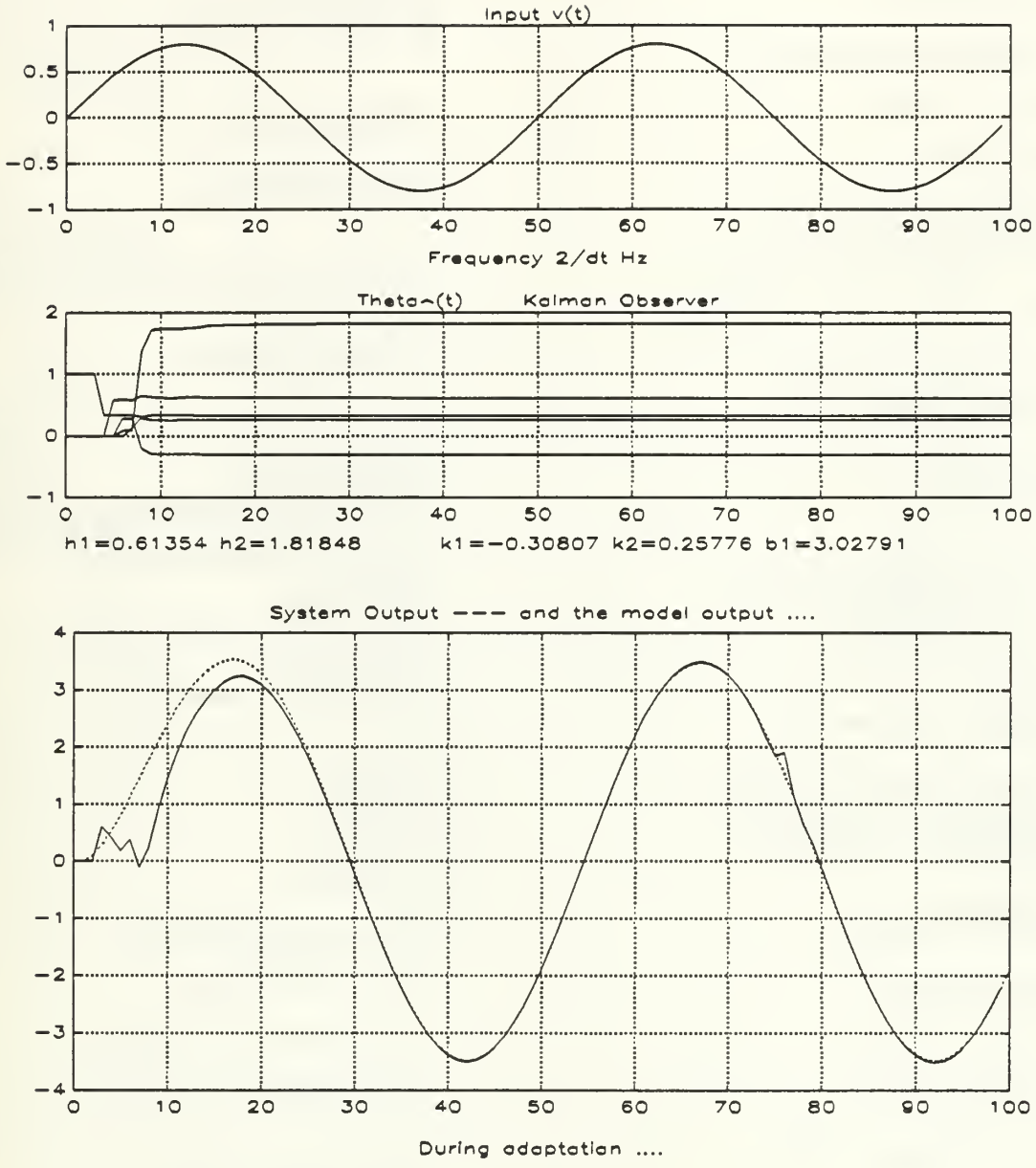

Figure A-6: Least-Squares Estimator DMRAC Experiment 6

```
% System y(t) - 0.2y(t-1) + 0.9y(t-2) = 3u(t-1)
%
Aq = [1 -0.2 0.9]
Bq = [3 0]

disp('Poles of the system');
Poles = roots(Aq)

% The state-space representation of the system
%
[Phi,Gma,C,D] = tf2ss(Bq,Aq)

% Controller Design
%
disp('(a) Estimated State-Feedback Approaches');
disp('Luenberger Observer Poles');
% Choose the observer poles to be four time faster
%
MagPoles = abs(Poles).^4; ArgPoles = angle(Poles);
ObsPoles=MagPoles.*(cos(ArgPoles)+i*sin(ArgPoles))


disp('The Luenberger observer gain Kl');
Kl = acker(Phi',C',ObsPoles)'

% Kalman filter observer: Choose the noise
% covariance matrices
%
Q = [1 0; 0 1], R = [1]

disp('The steady-state Kalman observer gain Kk');
Kk = inv(Phi) * dlqe(Phi,[1],C,Q,R)

% State-feedback gain using estimate
% state-variables
%
disp('Reference model: D(q)ym(t)=v(t)');
Dq = [1 -0.8]

% Desired poles for the feedback system. One pole
% equal to pole of the reference model, the other
% to cancel the zero (z = -Bq(2)/Bq(1)) of
% the original system.
%
disp('Desired Closed Loop Poles');
DsrPoles = [-D(2); -Bq(2)/Bq(1)]

disp('System Feedback Gain L');
L = place(Phi,Gma,DsrPoles)

disp('(b) Diophantine Approach');
disp('Form the Sylvester matrix');
Ms = sylvest(Aq,Bq)

disp('Desired system behaviour: p*(q)');
Pstar = conv(Dq,Bq./Bq(1))

disp('Observer characteristic polynomials: α(q)');
disp('For the Luenberger Observer');
Alphal = conv([1 -ObsPoles(1)],[1 -ObsPoles(2)])
Fl = conv(Alphal,(Aq - Pstar))';
Fl = Fl(2:length(Fl))

disp('For the Kalman Observer');
KObsPoles = eig(Phi - Kk*C)
Alphak = conv([1 -KObsPoles(1)],[1 -KObsPoles(2)])
Fk = conv(Alphak,(Aq - Pstar))';
Fk = Fk(2:length(Fk))

disp('Controller parameters: h(q) and k(q)');
disp('For the Luenberger Observer');
hkl = inv(Ms) * Fl
disp('For the Kalman Observer');
hkk = inv(Ms) * Fk


% Number of unknowns h1,h2,...,k1,k2,...,1/b1
Nu = 2*(length(Aq) - 1) + 1

% Adaptive Controller Design
disp('Time horizon for simulation');
Nt = 1000

disp('Generating input data');
Tt = (0:Nt-1)/Nt;

if InpType == 'Square',
  v = 0.8*sign(sin(2*pi*Fr*Tt));
else
  v = 0.8*sin(2*pi*Fr*Tt);
end;
clg; subplot(211); plot(0:Nt-1,v); grid;
title('Input v(t)'); pause(1);
xlabel(sprintf('Frequency %g/dt Hz', Fr));

disp('Observer characteristics polynomial α(q)');
if Obs == 'L'
  Alpha = Alphal
  ObStr = 'Theta^(t)    Luenberger Observer'
else
  Alpha = Alphak
  ObStr = 'Theta^(t)    Kalman Observer'
end;

u = zeros(1,Nt);
uf = zeros(1,Nt);
y = zeros(1,Nt);
yf = zeros(1,Nt);
yd = zeros(1,Nt);
P0 = 1e3*eye(Nu);
P = P0;
disp('Setup the parameter vector Θ^(t)');
ThetaHat = zeros(Nu,Nt);
ThetaHat(Nu,1:Nt) = ones(1:Nt);

clc; disp('Simulation begins ... Please wait.');
for indx = 3:Nt,
  % Update the plant
  y(indx) = -Aq(2:3)* [y(indx-1);y(indx-2)] + ...
Bq(1)*u(indx-1);

  % Filter u(t) and y(t)
  uf(indx) = -Alpha(2:3) * [uf(indx-1); ...
          uf(indx-2)] + u(indx-1);
  yf(indx) = -Alpha(2:3) * [yf(indx-1); ...
          yf(indx-2)] + y(indx-1);
  yd(indx) = Dq*[y(indx); y(indx-1)];


  % Form the regression vector
  fi = [uf(indx-1); uf(indx-2); yf(indx-1); ...
      yf(indx-2); yd(indx)];
```

```
    % Adaptive update of the parameter estimates
    tmp = 1 + fi'*P*fi;
    ThetaHat(:,indx) = ThetaHat(:,indx-1) + ...
      P*fi*(u(indx-1)-fi'*ThetaHat(:,indx-1))/tmp;
    P = P - P*fi*fi'*P/tmp;

    % Update control action
    u(indx) = [uf(indx); uf(indx-1); yf(indx); ...
               yf(indx-1); v(indx)]' ...
               * ThetaHat(:,indx);
end;

plot(ThetaHat'); grid;
title(ObStr);
ThetaHat0 = mean(ThetaHat(:,0.9*Nt:Nt)')'
gtext(sprintf('h1 = %g h2 = %g',
      ThetaHat0(1),ThetaHat0(2)));
gtext(sprintf('k1 = %g k2 = %g b1 = %g',
      ThetaHat0(3),ThetaHat0(4),1/ThetaHat0(5)));


function [Ms] = Sylvest(A, B);

N = length(A);
M = length(B);
Ms = zeros(2*(N-1));
for indx=1:(N-1),
   Ms(indx:(indx+N-1),indx) = A(:);
   Ms((2*N-indx-M):(2*N-1-indx),(2*N-1-indx)) = B(:);
end;
```

APPENDIX B. BNN DMRAC FOR UNKNOWN LTI SYSTEMS

In this experiment, the least-squares estimator of the DMRAC developed in Appendix A is replaced directly with a BNN. Simulations results are then presented.

The unknown LTI system is given by equation (A-1) in Appendix A. The same desired reference model are employed. Therefore, the same regressor vector $\Phi(t)$ in equation (A-15) is used as the input for the neural network during training. Hence, the BNN must have 5 inputs. 20 neurons are employed in the hidden layer. Only a single neuron is needed in the output to produce the control input.

The BNN is first off-line trained, tested to see if it is adequately trained, and then put online for simultaneous learning and control of the system. These steps are accomplished by the programs attached: OFFLIN.M, TSTLIN.M and ONLIN.M. Similar procedures for off-line training, testing and on-line training are used subsequently for the BNN DMRAC for unknown nonlinear systems. They are discussed in great detail in the Chapter III and shall not be repeated here.

The results of various experiments are shown in Figures B-1 to B-5. It can be clearly seen that the BNN DMRAC performs as well as the least-squares estimator.

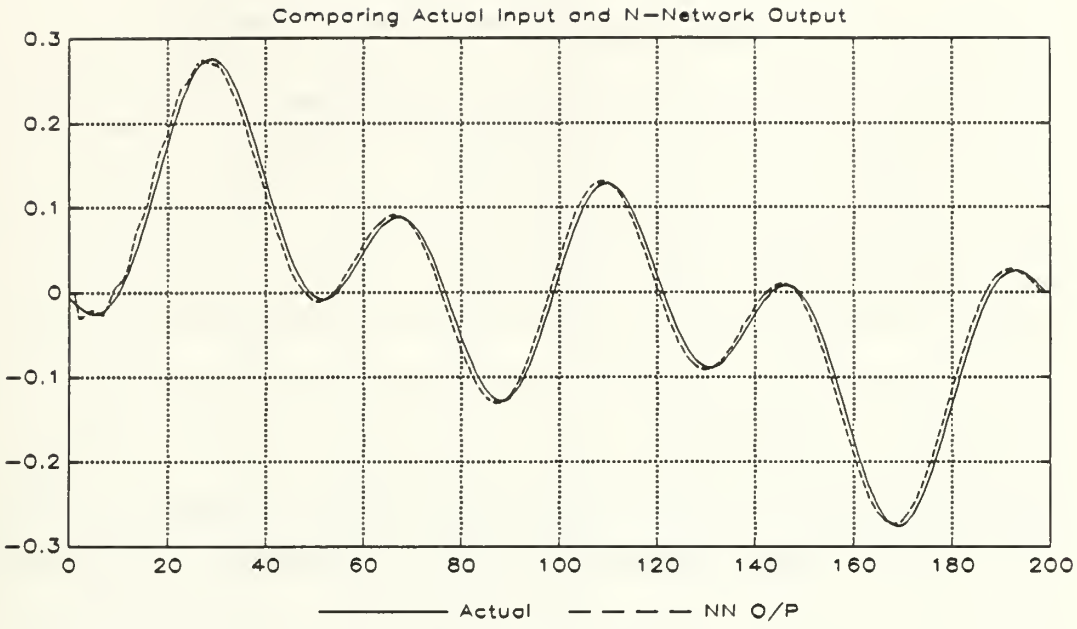

Figure B-1. After Offline Training.

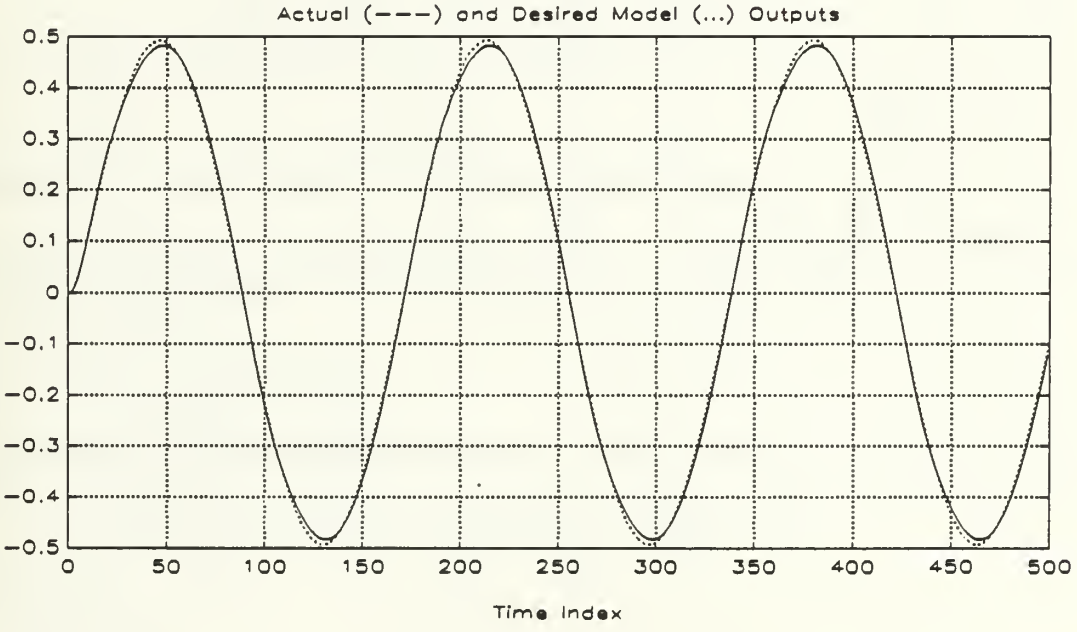

Figure B-2. On-line Control With A Single Sinusoid Input.

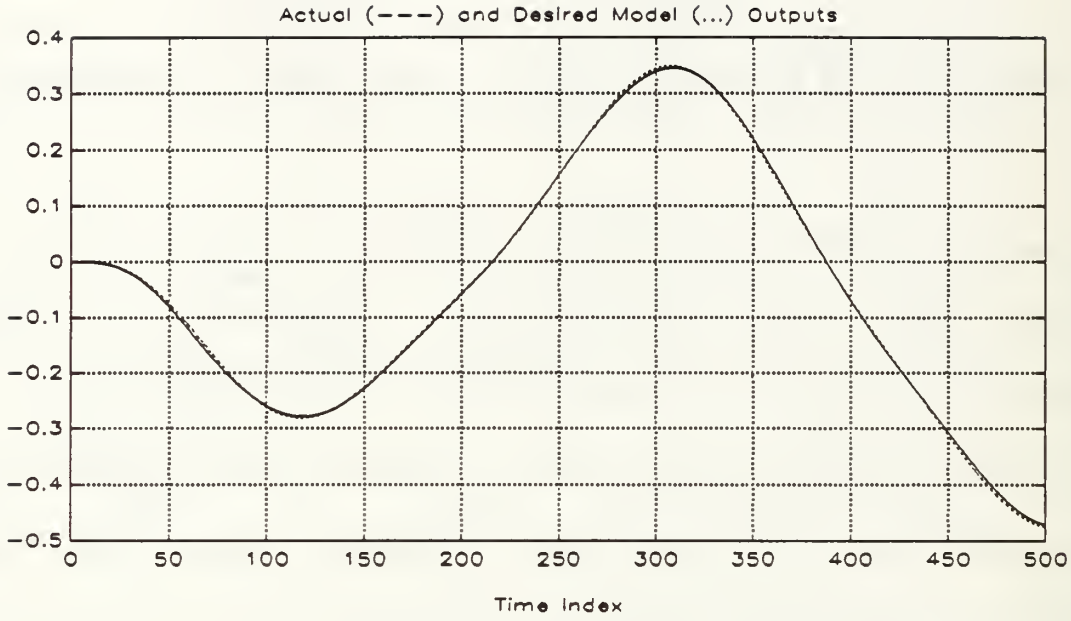

Figure B-3. On-line Control With Sum of Sinusoids Input.

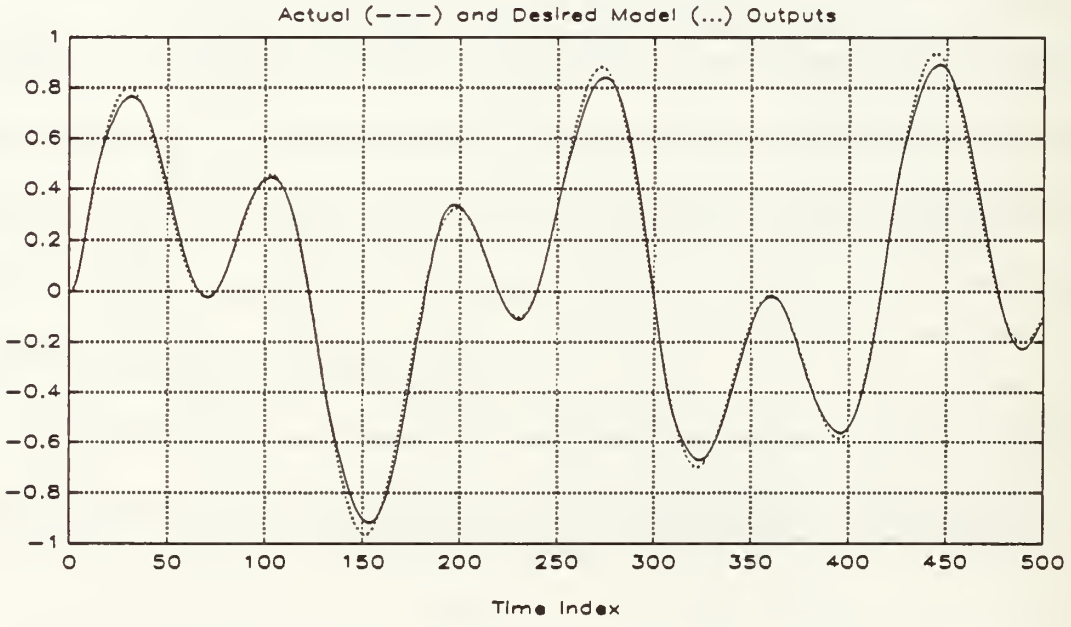

Figure B-4. On-line Control With High Frequency Sum of Sinusoids Input.

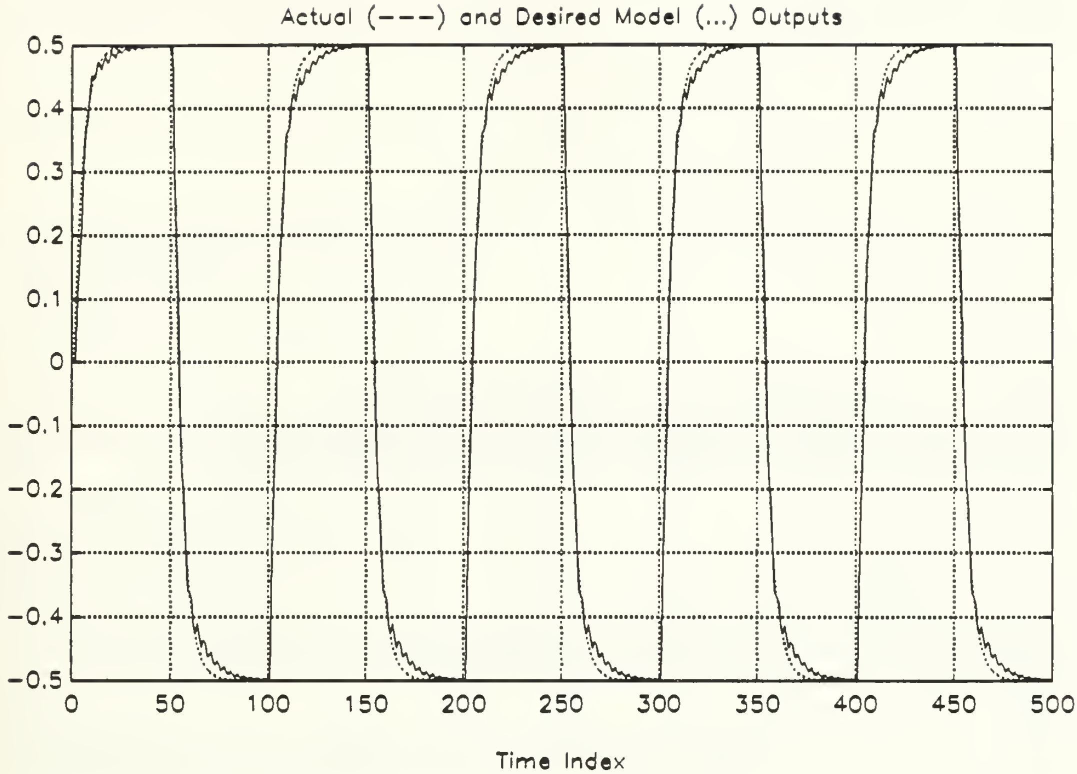

Figure B-5. On-line Control With A Square Wave Input.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%OFFLIN.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Neural Network Identification and Control of an
%    Unknown LTI System.
%
%    Offline training for Identifier-Controller.
%
%    Written by: Teo, Chin-Hock   15 Sept 91
%%%%%%%%%%%%%%%%%%%%%%%%%%OFFLIN.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Offline training. Generating Training Data
Nt = 500; Tt=(1:Nt)/Nt; ut=0.2*(sin(2*pi*3*Tt)+sin(2*pi*5*Tt)-cos(2*pi*4*Tt)+1);

yt=zeros(1,Nt);
for indx=3:Nt
% Simulate the system
    yt(indx) = 0.2*yt(indx-1) - 0.9*yt(indx-2) + 3*ut(indx-1);
end;

disp('Choose the observer characteristic polynomial');
% Assuming that the system is 2nd order.
```

```
%
Alpha = [1 -0.15 0.005];

disp('Generate filtered signals uF & yF ... ');
% Using the observer as the filter
%
uft = filter(1, Alpha, ut(:)); yft = filter(1, Alpha, yt(:));

disp('The desired reference model');
% Assume that a first order reference model can be tracked.
%
Dq = [1 -0.8]; ydt = filter(Dq, 1, yt(:));

% Plotting the training data
%
clg; subplot(221);
plot(0:(Nt-1),ut); title('Training Input ut(t)'); xlabel('Time Index'); grid;
plot(0:(Nt-1),yt); title('Training Output y(t)'); xlabel('Time Index'); grid;
plot(0:(Nt-1),uft); title('Filtered Training Input uFt(t)'); xlabel('Time Index'); grid;
plot(0:(Nt-1),yft); title('Filtered Training Output yFt(t)'); xlabel('Time Index'); grid;

% Create Neural Network
%
First = input('Create a new neural network ? (Y)es  (N)o : ', 's');
if First == 'Y' | First == 'y',
% Creating the neural network called IdCtrlr, with Clayer(1) inputs and two hidden layer of
% Clayer(2) neurons and an output layer with Clayer(3) neurons.
%
  Clayer = [5, 20, 1]; [IdCtrlr,W1,W2,dW1,dW2] = net2f(Clayer,2);
else
% Continue training the net.
%
  disp('Loading trained net ..... '); load netlin;
end;

% Choose learning parameters
%
Learn1 = 0.5; Learn2 = 0.2; Moment1 = 0.5; Moment2 = 0.4;
Lpar = [Learn1, Learn2, Moment1, Moment2];

% Set Bias = 0 for no bias. Always set Gain = 1.
%
Bias = 0; Gain = 1; Npar = [Bias, Gain];

% Index to output neuron
%
OpIndx = sum(Clayer);

% Estimator Neural Network Learning
%
disp('Neural Network Training ...'); Lnum = 20
for indx=1:Lnum
% Randomly shuffle the order of presentation of data points.
 disp('Shuffling training data ... '); Rindx = shuffle(Nt-2) + 2; indx,
   for indx1=1:(Nt-2)
```

```
        IdCtrlr(1) = yft(Rindx(indx1)-1); IdCtrlr(2) = yft(Rindx(indx1)-2);
        IdCtrlr(3) = uft(Rindx(indx1)-1); IdCtrlr(4) = uft(Rindx(indx1)-2);
        IdCtrlr(5) = ydt(Rindx(indx1)); [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar);
              DoVec = [ut(Rindx(indx1)-1)]; [W1,W2,dW1,dW2] =
learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
    end;
end;
save netlin IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq



%%%%%%%%%%%%%%%%%%%%%%TSTLIN.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Neural Network Identification and Control of an
%    Unknown LTI System.
%
%    Testing the offline trained Identifier-Controller.
%
%    Written by: Teo, Chin-Hock  15 Sep 91
%%%%%%%%%%%%%%%%%%%%%%TSTLIN.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% To test the trained net: A input u(t) is fed into the 'unknown'
% system to generate a set of
% output data. The generated data are then used to feed the trained
% neural network to produce u^(t).
%
% Load the trained neural network
%
load netlin
disp('Generating test input ...'); Nv = 200;
u = 0.1 .*(sin(2*pi*(1:Nv)/Nv)+sin(2*pi*(1:Nv).*2/Nv)-sin(2*pi*(1:Nv).*5/Nv));
%u = 0.1 * sign(sin(2*pi*5*(1:Nv)/Nv));

disp('Generating test output ...'); y=zeros(1,Nv);
for indx=3:Nv,
%
% Simulate the system
    y(indx) = 0.2*y(indx-1) - 0.9*y(indx-2) + 3*u(indx-1);
end;

% Filtered signals using the observer as the filter
%
uf = filter(1, Alpha, u(:)); yf = filter(1, Alpha, y(:));

% Desired output
%
yd = filter(Dq, 1, y(:));

% Plot the test data
%
clg; subplot(221);
plot(0:(Nv-1),u); title('System Model 1: Test Input u(t)'); xlabel('Time Index'); grid;
plot(0:(Nv-1),y); title('Test Output y(t)'); xlabel('Time Index'); grid;
plot(0:(Nv-1),uf); title('Filtered Test Input uF(t)'); xlabel('Time Index'); grid;
plot(0:(Nv-1),yf); title('Filtered Test Output yF(t)'); xlabel('Time Index'); grid;
uhat=zeros(1,Nv);
```

```
% Identifier Recalling
for indx=4:Nv
   IdCtrlr(1) = yf(indx-1); IdCtrlr(2) = yf(indx-2);
   IdCtrlr(3) = uf(indx-1); IdCtrlr(4) = uf(indx-2);
   IdCtrlr(5) = yd(indx);
   [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); uhat(indx-1) = IdCtrlr(Oplndx);
end;

% Plot the result comparing u(t) to u^(t)
clg;subplot(111); plot(0:(Nv-1),u(1:Nv),0:(Nv-1),uhat(1:Nv),'--');
title('Comparing Actual Input and N-Network Output'); xlabel('_____ Actual   _ _ _ _ NN O/P'); grid;
!del lin1.met
meta lin1



%%%%%%%%%%%%%%%%%%%%%%% ONLIN.M %%%%%%%%%%%%%%%%%%%%%%%%%
%    Neural Network Identification and Control of an
%    Unknown Nonlinear Dynamical System Type 1.
%
%    Online training for Identifier-Controller.
%
%    Written by: Teo, Chin-Hock  11 Oct 91
%%%%%%%%%%%%%%%%%%%%%%% ONLIN.M %%%%%%%%%%%%%%%%%%%%%%%%%
% Load the trained net
%
load netlin
%
% Learning parameters for online learning
%
Learn = [0.8 0.8]; Moment = [0.2 0.2]; Lpar = [Learn Moment];
% Leave Npar unchanged

disp('Generating the reference signal ... ');
Ns = 500; Ts=(0:Ns-1)/Ns;
%
% Keep Ref small so the ym is between ± 1
Ref = 0.06*(0.8*sin(pi*Ts)+0.6*cos(pi*3*Ts)-0.6- ...
     0.5*sin(0.2*pi*14*Ts)+0.1*sin(2*pi*3.7*Ts));
%Ref = [zeros(1,Ns/5), 0.65*ones(1,4*Ns/5)];
%Ref = 0.1 * sin(2*pi*6*Ts) + 0.1 *sin(2*pi*2.5*Ts);
%Ref = 0.1*sign(sin(2*pi*5*Ts));
%Ref = 0.08*sin(2*pi*14*Ts);


% Reference model output
%
ym = dlsim(1,Dq,Ref);
clg; subplot(211);
plot(0:Ns-1,Ref); title('System Model 1: Reference Signal v(t)'); xlabel('Time Index'); grid;
plot(0:Ns-1,ym); title('Desired Reference Model Output ym(t)'); xlabel('Time Index'); grid;

% Initial Conditions
%
```

```
ys=zeros(1,Ns); us=zeros(1,Ns);
ufs=zeros(1,Ns); yfs=zeros(1,Ns);

Onlin = input('(0) No Learning (1) Online Learning : ');
if Onlin == 1
  disp('Online Control and Learning ... ');
else
  disp('Online Control ... ');
end;
for indx=3:Ns-1
% Generate the control signal us
%
  IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
  IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
  IdCtrlr(5) = Ref(indx);
  [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); us(indx) = IdCtrlr(OpIndx);

% Update the plant
%
    ys(indx+1) = 0.2*ys(indx) - 0.9*ys(indx-1) + 3*us(indx);

% Filter u(indx) and y(indx) with the observer filter
%
  ufs(indx+1) = -Alpha(2:3)*[ufs(indx); ufs(indx-1)] + us(indx);
  yfs(indx+1) = -Alpha(2:3)*[yfs(indx); yfs(indx-1)] + ys(indx);
  yds(indx+1) = Dq*[ys(indx+1); ys(indx)];

% Identifier on-line learning
%
  if Onlin == 1
    IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
    IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
    IdCtrlr(5) = yds(indx+1);
    [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = us(indx);
    [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
  end;
end;

clg; plot(0:Ns-1, ys); grid; xlabel('Time Index');
title('Actual (--) and Desired Model (...) Outputs'); hold; plot(ym,'g:'); hold off
!del lin2.met
meta lin2
pause;

if Onlin == 1,
  tstlin; pause;
  Ch = input('Do you wish to save the online trained net: (Y) or (N) ? ', 's');
  if Ch == 'Y' | Ch == 'y',
    save netlin IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq
  end;
end;
```

APPENDIX C. SIMULATIONS PROGRAMS

```
% Experiment # 1
%%%%%%%%%%%%%%%%%%%% OFFTRG1F.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Neural Network Identification and Control of an
%    Unknown Nonlinear Dynamical System Type 1.
%
%    Offline training for Identifier-Controller.
%
%    Written by: Teo, Chin-Hock   11 Oct
%%%%%%%%%%%%%%%%%%%% OFFTRG1F.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Keep input, ut(t) between ± 1
%
disp('Generate training input ... '); Nt = 200; rand('uniform');
u1 = 0.2*sin(2*pi.*(0:Nt-1).*1/Nt + 0.1*pi.*(rand(1,Nt) - 0.5));
u2 = 0.4*cos(2*pi.*(0:Nt-1).*3/Nt + 0.05*pi.*(rand(1,Nt) - 0.5));
u3 = 0.1*sin(2*pi.*(0:Nt-1).*7/Nt + 0.02*pi.*(rand(1,Nt) - 0.5));
u4 = 1.0*(rand(1,Nt) - 0.5);
ut = 0.5*(u1 - u2 + u3 - u4);
%
% The unknown parameters of the nonlinear dynamical system here.
%
a1 = 0.3; a2 = 0.6; b0 = 1; b1 = 0.3; b2 = -0.4;
%
% The generating outputs of the unknown nonlinear dynamical
% system here.
%
disp('Generate training output ... '); yt=zeros(1,Nt);
for indx=2:Nt,
   yt(indx+1) = a1*yt(indx) + a2*yt(indx-1) + b0*ut(indx)^3 + b1*ut(indx)^2 + b2*ut(indx);
end;

disp('Choose the observer characteristic polynomial');
% Assuming that the system is 2nd order.
%
ObsPoles = [0.1; 0.05]; Alpha = conv([1 -ObsPoles(1)],[1 -ObsPoles(2)])

disp('Generate filtered signals uF & yF ... ');
% Using the observer as the filter
%
uft = filter(1, Alpha, ut(:)); yft = filter(1, Alpha, yt(:));

disp('The desired reference model');
% Assume that a first order reference model can be tracked.
%
Dq = [1 -0.8]; ydt = filter(Dq, 1, yt(:));

% Plotting the training data
%
clg; subplot(221);
plot(0:Nt-1,ut); title('Training Input ut(t)'); xlabel('Time Index'); grid;
```

```
plot(0:Nt,yt); title('Training Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nt-1,uft); title('Filtered Training Input uFt(t)'); xlabel('Time Index'); grid;
plot(0:Nt,yft); title('Filtered Training Output yFt(t)'); xlabel('Time Index'); grid;
!del ex11f.met
meta ex11f

% Create Neural Network
%
First = input('Create a new neural network ? (Y)es  (N)o : ', 's');
if First == 'Y' | First == 'y',
% Creating the neural network called IdCtrlr, with Clayer(1) inputs and two hidden layer of
% Clayer(2) neurons and an output layer with Clayer(3) neurons.
%
  Clayer = [5, 15, 1]; [IdCtrlr,W1,W2,dW1,dW2] = net2f(Clayer,2);
else
% Continue training the net.
%
  disp('Loading trained net ..... '); load netex1x;
end;

% Choose learning parameters
%
Learn1 = 0.5; Learn2 = 0.2; Moment1 = 0.5; Moment2 = 0.4;
Lpar = [Learn1, Learn2, Moment1, Moment2];

% Set Bias = 0 for no bias. Always set Gain = 1.
%
Bias = 1; Gain = 1; Npar = [Bias, Gain];

% Index to output neuron
%
OpIndx = sum(Clayer);

% Estimator Neural Network Learning
%
disp('Neural Network Training ...'); Lnum = 50
for indx=1:Lnum
% Randomly shuffle the order of presentation of data points.
    disp('Shuffling training data ...  '); Rindx = shuffle(Nt-2) + 2; indx,
    for indx1=1:(Nt-2)
      IdCtrlr(1) = yft(Rindx(indx1)-1); IdCtrlr(2) = yft(Rindx(indx1)-2);
      IdCtrlr(3) = uft(Rindx(indx1)-1); IdCtrlr(4) = uft(Rindx(indx1)-2);
      IdCtrlr(5) = ydt(Rindx(indx1));
      [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = [ut(Rindx(indx1)-1)];
      [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
    end;
end;
save netex1f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx a1 a2 b0 b1 b2 Alpha Dq
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%TSTRG1F.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Neural Network Identification and Control of an
%    Unknown Nonlinear Dynamical System Type 1.
%
%    Testing the offline trained Identifier-Controller.
%
%    Written by: Teo, Chin-Hock  11 Oct 91
%%%%%%%%%%%%%%%%%%%%%%%%%TSTRG1F.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% To test the trained net: A input u(t) is fed into the 'unknown' system to generate a set of
% output data. The generated data are then used to feed the trained neural network to produce u^(t).
%
% Load the trained neural network
%
load netex1f
disp('Generating test input ...'); Nv = 200;
u = 0.1 .*(sin(2*pi*(1:Nv)/Nv)+sin(2*pi*(1:Nv).*2/Nv)-sin(2*pi*(1:Nv).*5/Nv));
%u = 0.1 * sign(sin(2*pi*5*(1:Nv)/Nv));

disp('Generating test output ...'); y=zeros(1,Nv);
for indx=2:Nv,
% Unknown plant
   y(indx+1) = a1*y(indx) + a2*y(indx-1) + b0*u(indx)^3 + b1*u(indx)^2 + b2*u(indx);
end;

% Filtered signals
% Using the observer as the filter
%
uf = filter(1, Alpha, u(:)); yf = filter(1, Alpha, y(:));

% Desired output
%
yd = filter(Dq, 1, y(:));

% Plot the test data
%
clg; subplot(221);
plot(0:Nv-1,u); title('System Model 1: Test Input u(t)'); xlabel('Time Index'); grid;
plot(0:Nv,y); title('Test Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nv-1,uf); title('Filtered Test Input uF(t)'); xlabel('Time Index'); grid;
plot(0:Nv,yf); title('Filtered Test Output yF(t)'); xlabel('Time Index'); grid;
!del ex12f.met
meta ex12f

uhat=zeros(1,Nv);
% Identifier Recalling
for indx=4:Nv
   IdCtrlr(1) = yf(indx-1); IdCtrlr(2) = yf(indx-2);
   IdCtrlr(3) = uf(indx-1); IdCtrlr(4) = uf(indx-2);
   IdCtrlr(5) = yd(indx);
   [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); uhat(indx-1) = IdCtrlr(OpIndx);
end;

% Plot the result comparing u(t) to u^(t)
clg;subplot(111); plot(0:Nv-1,u(1:Nv),0:Nv-1,uhat(1:Nv),'--');
title('System Model 1: Comparing Actual Input and N-Network Output');
```

```
xlabel('_____ Actual  _ _ _ _ NN O/P'); grid;
!del ex13f.mct
meta ex13f
%%%%%%%%%%%%%%%%%%%%%%%%ONTRG1F.M%%%%%%%%%%%%%%%%%%%%%%%%
%    Neural Network Identification and Control of an
%    Unknown Nonlinear Dynamical System Type 1.
%
%    Online training for Identifier-Controller.
%
%    Written by: Teo, Chin-Hock  11 Oct 91
%%%%%%%%%%%%%%%%%%%%%%%%ONTRG1F.M%%%%%%%%%%%%%%%%%%%%%%%%
% Load the trained net
%
load netex1f
%
% Learning parameters for online learning
%
Learn = [0.4 0.4]; Moment = [0 0]; Lpar = [Learn Moment];
% Leave Npar unchanged

disp('Generating the reference signal ... ');
Ns = 5000; Ts=(0:Ns-1)/Ns;
%
% Keep Ref small so the ym is between ± 1
%
Ref = 0.06*(0.5*sin(2*pi*Ts)+cos(2*pi*7*Ts)-0.3*sin(2*pi*14*Ts));
%Ref = [zeros(1,Ns/5), 0.1*ones(1,4*Ns/5)];
%Ref = 0.1 * sin(2*pi*3*Ts);
%Ref = 0.1*sign(sin(2*pi*5*Ts));
%Ref = 0.08*sin(2*pi*14*Ts);


% Reference model output
%
ym = dlsim(1,Dq,Ref); clg; subplot(211);
plot(0:Ns-1,Ref); title('System Model 1: Reference Signal v(t)'); xlabel('Time Index'); grid;
plot(0:Ns-1,ym); title('Desired Reference Model Output ym(t)'); xlabel('Time Index'); grid;
!del cx14f.met
meta ex14f

% Initial Conditions
%
ys=zeros(1,Ns); us=zeros(1,Ns); ufs=zeros(1,Ns); yfs=zeros(1,Ns);

Onlin = input('(0) No Learning (1) Online Learning : ');
if Onlin == 1
   disp('Online Control and Learning ... ');
else
   disp('Online Control ... ');
end;
for indx=3:Ns-1
% Generate the control signal us
%
   IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
   IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
```

```
  IdCtrlr(5) = Ref(indx);
  [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); us(indx) = IdCtrlr(OpIndx);

% Update the plant
%
  ys(indx+1) = a1*ys(indx) + a2*ys(indx-1) + b0*us(indx)^3 + b1*us(indx)^2 + b2*us(indx);
% Filter u(indx) and y(indx) with the observer filter
%
  ufs(indx+1) = -Alpha(2:3)*[ufs(indx); ufs(indx-1)] + us(indx);
  yfs(indx+1) = -Alpha(2:3)*[yfs(indx); yfs(indx-1)] + ys(indx);
  yds(indx+1) = Dq*[ys(indx+1); ys(indx)];

% Identifier on-line learning
%
  if Onlin == 1
    IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
    IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
    IdCtrlr(5) = yds(indx+1);
    [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = us(indx);
    [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
  end;
end;

clg; plot(0:Ns-1, ys); grid; xlabel('Time Index');
title('System Model 1: Actual (___) and Desired Model (---) Outputs'); hold; plot(ym,'g--'); hold off
!del ex15f.met
meta ex15f
pause;

if Onlin == 1,
  tstrg1f; pause;
  Ch = input('Do you wish to save the online trained net: (Y) or (N) ? ', 's');
  if Ch == 'Y' | Ch == 'y',
    save netex1f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx a1 a2 b0 b1 b2 Alpha Dq
  end;
end;



% Experiment # 2
%%%%%%%%%%%%%%%%%%%%%%% OFFTRG2F.M %%%%%%%%%%%%%%%%%%%%%%%%%%%
% Keep input, ut(t) between ± 1
disp('Generate training input ... '); Nt = 200; rand('uniform');
u1 = 0.2*sin(2*pi.*(0:Nt-1).*1/Nt + 0.1*pi.*(rand(1,Nt) - 0.5));
u2 = 0.4*cos(2*pi.*(0:Nt-1).*3/Nt + 0.05*pi.*(rand(1,Nt) - 0.5));
u3 = 0.1*sin(2*pi.*(0:Nt-1).*7/Nt + 0.02*pi.*(rand(1,Nt) - 0.5));
u4 = 1.0*(rand(1,Nt) - 0.5);
ut = 0.2*(u1 - u2 + u3 - u4);

% The generating outputs of the unknown nonlinear dynamical
% system here.

disp('Generate training output ... '); yt=zeros(1,Nt);
for indx=2:Nt,
  yt(indx+1) = (yt(indx)*yt(indx-1)*(yt(indx)+2.5))/(1+yt(indx)^2+yt(indx-1)^2) + ut(indx);
```

```
end;

disp('Choose the observer characteristic polynomial');
% Assuming that the system is 2nd order.
%
ObsPoles = [0.1; 0.05]; Alpha = conv([1 -ObsPoles(1)],[1 -ObsPoles(2)])

disp('Generate filtered signals uF & yF ... ');
% Using the observer as the filter
%
uft = filter(1, Alpha, ut(:)); yft = filter(1, Alpha, yt(:));

disp('The desired reference model');
% Assume that a first order reference model can be tracked.
%
Dq = [1 -0.8]; ydt = filter(Dq, 1, yt(:));

% Plotting the training data
%
clg; subplot(221);
plot(0:Nt-1,ut); title('System Model 2: Training Input ut(t)'); xlabel('Time Index'); grid;
plot(0:Nt,yt); title('Training Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nt-1,uft); title('Filtered Training Input uFt(t)'); xlabel('Time Index'); grid;
plot(0:Nt,yft); title('Filtered Training Output yFt(t)'); xlabel('Time Index'); grid;
!del ex21f.met
meta ex21f

% Create Neural Network
%
First = input('Create a new neural network ? (Y)es  (N)o : ', 's');
if First == 'Y' | First == 'y',
% Creating the neural network called IdCtrlr, with Clayer(1) inputs and one hidden layer
% of Clayer(2) neurons and an output layer with Clayer(3) neurons.
%
  Clayer = [5, 15, 1]; [IdCtrlr,W1,W2,dW1,dW2] = net2f(Clayer,1);
else
  disp('Loading trained net ..... '); load netex2f;
end;

% Choose learning parameters
%
Learn = [0.6 0.4]; Moment = [0.4 0.4]; Lpar = [Learn Moment];

% Set Bias = 0 for no bias. Set Gain = 1.
%
Bias = 1; Gain = 1; Npar = [Bias, Gain];

% Index to output neuron
%
OpIndx = sum(Clayer);

% Estimator Neural Network Learning
%
disp('Neural Network Training ...'); Lnum = 50
for indx=1:Lnum
```

```
% Randomly shuffle the order of presentation of data points.
    disp('Shuffling training data ...  '); Rindx = shuffle(Nt-2) + 2; indx,
    for indx1=1:(Nt-2)
      IdCtrlr(1) = yft(Rindx(indx1)-1); IdCtrlr(2) = yft(Rindx(indx1)-2);
      IdCtrlr(3) = uft(Rindx(indx1)-1); IdCtrlr(4) = uft(Rindx(indx1)-2);
      IdCtrlr(5) = ydt(Rindx(indx1));
      [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = [ut(Rindx(indx1)-1)];
      [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
    end;
end;
save netex2f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq

%%%%%%%%%%%%%%%%%%%%%%TSTRG2F.M%%%%%%%%%%%%%%%%%%%%%%%%%%
% To test the trained net: A input u(t) is fed into the 'unknown' system to generate a set of
% output data. The generated data are then used to feed the trained neural network to produce u^(t).
%
% Load the trained neural network
%
load netex2f
%
% Keep input, ut(t) between ± 1
%
disp('Generating test input ...'); Nv = 200;
u = 0.1 .*(sin(2*pi*(1:Nv)/Nv)+sin(2*pi*(1:Nv).*2/Nv)-sin(2*pi*(1:Nv).*5/Nv));
%u = 0.1 * sign(sin(2*pi*5*(1:Nv)/Nv));

disp('Generating test output ...'); y=zeros(1,Nv);
for indx=2:Nv,
% Unknown plant
%
  y(indx+1) = (y(indx)*y(indx-1)*(y(indx)+2.5))/(1+y(indx)^2+y(indx-1)^2)+u(indx);
end;

% Filtered signals
% Using the observer as the filter
%
uf = filter(1, Alpha, u(:)); yf = filter(1, Alpha, y(:));

% Desired output
yd = filter(Dq, 1, y(:));

% Plot the test data
%
clg; subplot(221);
plot(0:Nv-1,u); title('System Model 2: Test Input u(t)'); xlabel('Time Index'); grid;
plot(0:Nv,y); title('Test Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nv-1,uf); title('Filtered Test Input uF(t)'); xlabel('Time Index'); grid;
plot(0:Nv,yf); title('Filtered Test Output yF(t)'); xlabel('Time Index'); grid;
!del ex22f.met
meta ex22f

uhat=zeros(1,Nv);
% Identifier Recalling
for indx=4:Nv
  IdCtrlr(1) = yf(indx-1); IdCtrlr(2) = yf(indx-2);
```

```
   IdCtrlr(3) = uf(indx-1); IdCtrlr(4) = uf(indx-2);
   IdCtrlr(5) = yd(indx);
   [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); uhat(indx-1) = IdCtrlr(Oplndx);
end;

clg;subplot(111); plot(0:Nv-1,u(1:Nv),0:Nv-1,uhat(1:Nv),'--');
title('System Model 2: Comparing Actual Input and N-Network Output');
xlabel('_____ Actual  _ _ _ _ NN O/P'); grid;
!del ex23f.met
meta ex23f

%%%%%%%%%%%%%%%%%%%%%%%%ONTRG2F.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the trained net
%
load netex2f
%
% Learning parameters for online learning
%
Learn = [0.3 0.3]; Moment = [0 0]; Lpar = [Learn Moment];
% Leave Npar unchanged

disp('Generating the reference signal ... ');
Ns = 5000; Ts=(0:Ns-1)/Ns;
%
% Keep Ref small so the ym is between ± 1
%
Ref = 0.02*(0.5*sin(2*pi*Ts)+cos(2*pi*7*Ts)-1)+0.3*sin(2*pi*17*Ts));
%Ref = [zeros(1,Ns/5), 0.1*ones(1,4*Ns/5)];
%Ref = 0.1 * sin(2*pi*3*Ts);
%Ref = 0.1*sign(sin(2*pi*5*Ts));
%Ref = zeros(1,Ns); %Ref(1:10) = 0.5*ones(1,10);
%Ref = 0.5*ones(1,Ns);

% Reference model output
ym = dlsim(1,Dq,Ref); clg; subplot(211);
plot(0:Ns-1,Ref); title('Model System 2: Reference Signal v(t)'); xlabel('Time Index'); grid;
plot(0:Ns-1,ym); title('Desired Reference Model Output ym(t)'); xlabel('Time Index'); grid;
!del del ex24f.met
meta ex24f

% Initial Conditions
ys=zeros(1,Ns); us=zeros(1,Ns); ufs=zeros(1,Ns); yfs=zeros(1,Ns);

Onlin = input('(0) No Learning (1) Online Learning : ');
if Onlin == 1
   disp('Online Control and Learning ... ');
else
   disp('Online Control ... ');
end;
for indx=3:Ns-1
% Generate the control signal us
%
   IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
   IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
   IdCtrlr(5) = Ref(indx);
```

```
  [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); us(indx) = IdCtrlr(OpIndx);

% Update the plant
%
  ys(indx+1) = (ys(indx)*ys(indx-1)*(ys(indx)+2.5))/(1+ys(indx)^2+ys(indx-1)^2)+us(indx);

% Filter u(indx) and y(indx) with the observer filter
%
  ufs(indx+1) = -Alpha(2:3)*[ufs(indx); ufs(indx-1)] + us(indx);
  yfs(indx+1) = -Alpha(2:3)*[yfs(indx); yfs(indx-1)] + ys(indx);
  yds(indx+1) = Dq*[ys(indx+1); ys(indx)];

% Identifier on-line learning
%
  if Onlin == 1
    IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
    IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
    IdCtrlr(5) = yds(indx+1);
    [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = us(indx);
    [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
  end;
end;

clg; plot(0:Ns-1, ys); grid; xlabel('Time Index');
title('System Model 2: Actual (___) and Desired Model (...) Outputs'); hold; plot(ym,':'); hold off
!del ex25f.met
meta ex25f

if Onlin == 1,
  tstrg2f; pause;
  Ch = input('Do you wish to save the online trained net: (Y) or (N) ? ', 's');
  if Ch == 'Y' | Ch == 'y',
    save netex2f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq
  end;
end;




% Experiment # 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%OFFTRG3.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Keep input, ut(t) between ± 1
%
disp('Generate training input ... '); Nt = 200; rand('uniform');
u1 = 0.2*sin(2*pi.*(0:Nt-1).*1/Nt + 0.1*pi.*(rand(1,Nt) - 0.5));
u2 = 0.4*cos(2*pi.*(0:Nt-1).*3/Nt + 0.05*pi.*(rand(1,Nt) - 0.5));
u3 = 0.1*sin(2*pi.*(0:Nt-1).*7/Nt + 0.02*pi.*(rand(1,Nt) - 0.5));
u4 = 1.0*(rand(1,Nt) - 0.5);
ut = 0.2*(u1 - u2 + u3 - u4);
%
% The generating outputs of the unknown nonlinear dynamical
% system here.
%
disp('Generate training output ... '); yt=zeros(1,Nt);
for indx=2:Nt,
```

```
  yt(indx+1) = yt(indx)/(1 + yt(indx-1)^2) + ut(indx)^3;
end;

disp('Choose the observer characteristic polynomial');
% Assuming that the system is 2nd order.
%
ObsPoles = [0.03; 0.2]; Alpha = conv([1 -ObsPoles(1)],[1 -ObsPoles(2)])

disp('Generate filtered signals uF & yF ... ');
% Using the observer as the filter
%
uft = filter(1, Alpha, ut(:)); yft = filter(1, Alpha, yt(:));

disp('The desired reference model');
% Assume a zero order reference model can be tracked
%
Dq = [1 -0.6]; ydt = filter(Dq, 1, yt(:));

% Plotting the training data
%
clg; subplot(221);
plot(0:Nt-1,ut); title('Model System 3: Training Input ut(t)'); xlabel('Time Index'); grid;
plot(0:Nt,yt); title('Training Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nt-1,uft); title('Filtered Training Input uFt(t)'); xlabel('Time Index'); grid;
plot(0:Nt,yft); title('Filtered Training Output yFt(t)'); xlabel('Time Index'); grid;
!del ex31.met
meta ex31

% Create the Neural Network
%
First = input('Create a new neural network ? (Y)es  (N)o : ', 's');
if First == 'Y' | First == 'y',
% Creating the neural network called IdCtrlr, with Clayer(1) inputs and one hidden layer of
% Clayer(2) neurons and an output layer with Clayer(3) neurons.
%
  R = 1; Clayer = [5, 15, 1]; [IdCtrlr,W1,W2,dW1,dW2] = net2f(Clayer,R);
else
% Continue training the net.
%
  disp('Loading trained net ..... '); load netex3;
end;

% Choose learning parameters
%
Learn1 = 0.5; Learn2 = 0.7; Moment1 = 0.4; Moment2 = 0.4; Lpar = [Learn1, Learn2, Moment1, Moment2];

% Set Bias = 0 for no bias. Always set Gain = 1.
%
Bias = 1; Gain = 1; Npar = [Bias, Gain];

% Index to output neuron
%
OpIndx = sum(Clayer);

% Estimator Neural Network Learning
```

```
%
disp('Neural Network Training ...'); Lnum = 50
for indx=1:Lnum
% Randomly shuffle the order of presentation of data points.
    disp('Shuffling training data ...  '); Rindx = shuffle(Nt-2) + 2; indx,
    for indx1=1:(Nt-2)
      IdCtrlr(1) = yft(Rindx(indx1)-1); IdCtrlr(2) = yft(Rindx(indx1)-2);
      IdCtrlr(3) = uft(Rindx(indx1)-1); IdCtrlr(4) = uft(Rindx(indx1)-2);
      IdCtrlr(5) = ydt(Rindx(indx1));
      [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = [ut(Rindx(indx1)-1)];
      [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrir,W1,W2,dW1,dW2,Npar);
    end;
end;
save netex3f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq

%%%%%%%%%%%%%%%%%%%%%%%%TSTRG3.M%%%%%%%%%%%%%%%%%%%%%%%%
% To test the trained net: A input u(t) is fed into the 'unknown' system to generate a set of
% output data. The generated data are then used to feed the trained neural network to produce u^(t).
%
% Load the trained neural network
%
load netex3f
disp('Generating test input ...'); Nv = 200;
u = 0.1 .*(sin(2*pi*(1:Nv)/Nv)+sin(2*pi*(1:Nv).*2/Nv)-sin(2*pi*(1:Nv).*5/Nv));
%u = 0.1 * sign(sin(2*pi*5*(1:Nv)/Nv));

disp('Generating test output ...'); y=zeros(1,Nv);
for indx=2:Nv,
% Unknown plant
%
  y(indx+1) = y(indx)/(1 + y(indx-1)^2) + u(indx)^3;
end;

% Filtered signals
% Using the observer as the filter
%
uf = filter(1, Alpha, u(:)); yf = filter(1, Alpha, y(:));

% Desired output
%
yd = filter(Dq, 1, y(:));

% Plot the test data
%
clg; subplot(221);
plot(0:Nv-1,u); title('Model System 3: Test Input u(t)'); xlabel('Time Index'); grid;
plot(0:Nv,y); title('Test Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nv-1,uf); title('Filtered Test Input uF(t)'); xlabel('Time Index'); grid;
plot(0:Nv,yf); title('Filtered Test Output yF(t)'); xlabel('Time Index'); grid;
!del ex32f.met
meta ex32f

uhat=zeros(1,Nv);
% Identifier Recalling
for indx=4:Nv
```

```
  IdCtrlr(1) = yf(indx-1); IdCtrlr(2) = yf(indx-2);
  IdCtrlr(3) = uf(indx-1); IdCtrlr(4) = uf(indx-2);
  IdCtrlr(5) = yd(indx);
  [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); uhat(indx-1) = IdCtrlr(OpIndx);
end;

% Plot the result comparing u(t) to u^(t)
clg;subplot(111); plot(0:Nv-1,u(1:Nv),0:Nv-1,uhat(1:Nv),'-');
title('Model System 3: Comparing Actual Input and N-Network Output');
xlabel('_____ Actual  _ _ _ _ NN O/P'); grid;
!del ex33f.met
meta ex33f

%%%%%%%%%%%%%%%%%%%%%%%%ONTRG3F.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the trained net
%
load netex3f
%
% Learning parameters for online learning
%
Learn1 = 0.2; Learn2 = 0.2; Moment1 = 0; Moment2 = 0; Lpar = [Learn1, Learn2, Moment1, Moment2];
% Leave Npar unchanged

disp('Generating the reference signal ... ');
Ns = 5000; Ts=(0:Ns-1)/Ns;
Ref = 0.2*(0.5*sin(2*pi*Ts)+cos(2*pi*3*Ts))- 0.3*sin(2*pi*11*Ts));
%Ref = [zeros(1,Ns/5), 0.1*ones(1,4*Ns/5)];
%Ref = 0.1 * sin(2*pi*3*Ts);
%Ref = 0.1*sign(sin(2*pi*5*Ts));
%Ref = zeros(1,Ns); %Ref(1:10) = 0.5*ones(1,10);
%Ref = 0.5*ones(1,Ns);

% Reference model output
%
ym = dlsim(1,Dq,Ref); clg; subplot(211);
plot(0:Ns-1,Ref); title('Model System 3: Reference Signal v(t)'); xlabel('Time Index'); grid;
plot(0:Ns-1,ym); title('Desired Reference Model Output ym(t)'); xlabel('Time Index'); grid;
!del ex34.met
meta ex34

% Initial Conditions
%
ys=zeros(1,Ns); us=zeros(1,Ns); ufs=zeros(1,Ns); yfs=zeros(1,Ns);

Onlin = input('(0) No Learning (1) Online Learning : ');
if Onlin == 1
  disp('Online Control and Learning ... ');
else
  disp('Online Control ... ');
end;
for indx=3:Ns-1
% Generate the control signal us
%
  IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
  IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
```

```
  IdCtrlr(5) = Ref(indx);
  [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); us(indx) = IdCtrlr(OpIndx);

% Update the plant
%
  ys(indx+1) = ys(indx)/(1 + ys(indx-1)^2) + us(indx)^3;

% Filter u(indx) and y(indx) with the observer filter
%
  ufs(indx+1) = -Alpha(2:3)*[ufs(indx); ufs(indx-1)] + us(indx);
  yfs(indx+1) = -Alpha(2:3)*[yfs(indx); yfs(indx-1)] + ys(indx);
  yds(indx+1) = Dq*[ys(indx+1); ys(indx)];

% Identifier on-line learning
%
  if Onlin == 1
    IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
    IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
    IdCtrlr(5) = yds(indx+1);
    [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = us(indx);
    [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
  end;
end;

clg; plot(0:Ns-1, ys); grid; xlabel('Time Index');
title('System Model 3: Actual (___) and Desired Model (---) Outputs'); hold; plot(ym,'g--'); hold off
!del ex35f.met
meta ex35f

if Onlin == 1,
  tstrg3f; pause;
  Ch = input('Do you wish to save the online trained net: (Y) or (N) ? ', 's');
  if Ch == 'Y' | Ch == 'y',
    save netex3f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq
  end;
end;




% Experiment # 4
%%%%%%%%%%%%%%%%%%%%%%%% OFFTRG4F.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Keep input, ut(t) between ± 1
%
disp('Generate training input ... '); Nt = 200; rand('uniform');
u1 = 0.2*sin(2*pi.*(0:Nt-1).*1/Nt + 0.1*pi.*(rand(1,Nt) - 0.5));
u2 = 0.4*cos(2*pi.*(0:Nt-1).*3/Nt + 0.05*pi.*(rand(1,Nt) - 0.5));
u3 = 0.1*sin(2*pi.*(0:Nt-1).*7/Nt + 0.02*pi.*(rand(1,Nt) - 0.5));
u4 = 1.0*(rand(1,Nt) - 0.5);
ut = 0.2*(u1 - u2 + u3 - u4);

% The generating outputs of the unknown nonlinear dynamical system here.
%
disp('Generate training output ... '); yt=zeros(1,Nt);
for indx=3:Nt,
```

```
   yt(indx+1)=(yt(indx)*yt(indx-1)*yt(indx-2)*ut(indx-1)*(yt(indx-2)-1)+...
            ut(indx))/(1+yt(indx-1)^2+yt(indx-2)^2);
end;

disp('Choose the observer characteristic polynomial');
% Assuming that the system is 2nd order.
%
ObsPoles = [0.03; 0.05; 0]; Alpha = conv(conv([1 -ObsPoles(1)],[1 -ObsPoles(2)]),[1 -ObsPoles(3)])

disp('Generate filtered signals uF & yF ... ');
% Using the observer as the filter
%
uft = filter(1, Alpha, ut(:)); yft = filter(1, Alpha, yt(:));

disp('The desired reference model');
% Assume that a first order reference model can be tracked.
%
Dq = [1 -0.75]; ydt = filter(Dq, 1, yt(:));

% Plotting the training data
%
clg; subplot(221);
plot(0:Nt-1,ut); title('System Model 4: Training Input ut(t)'); xlabel('Time Index'); grid;
plot(0:Nt,yt); title('Training Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nt-1,uft); title('Filtered Training Input uFt(t)'); xlabel('Time Index'); grid;
plot(0:Nt,yft); title('Filtered Training Output yFt(t)'); xlabel('Time Index'); grid;
!del ex41f.met
meta ex41f

% Create Neural Network
%
First = input('Create a new neural network ? (Y)es  (N)o : ', 's');
if First == 'Y' | First == 'y',
% Creating the neural network called IdCtrlr, with Clayer(1) inputs and one hidden layer of
% Clayer(2) neurons and an output layer with Clayer(3) neurons.
%
  Clayer = [7, 21, 1]; [IdCtrlr,W1,W2,dW1,dW2] = net2f(Clayer,1);
else
% Continue training the net.
%
  disp('Loading trained net ..... '); load netex4;
end;

% Choose learning parameters
%
Learn = [0.6 0.6]; Moment = [0.4 0.4]; Lpar = [Learn Moment];

% Set Bias = 0 for no bias. Always set Gain = 1.
%
Bias = 1; Gain = 1; Npar = [Bias, Gain];

% Index to output neuron
%
OpIndx = sum(Clayer);
```

```
% Estimator Neural Network Learning
%
disp('Neural Network Training ...'); Lnum = 50
for indx=1:Lnum
% Randomly shuffle the order of presentation of data points.
    disp('Shuffling training data ... '); Rindx = shuffle(Nt-3) + 3; indx,
    for indx1=1:(Nt-3)
      IdCtrlr(1) = yft(Rindx(indx1)-1); IdCtrlr(2) = yft(Rindx(indx1)-2);
      IdCtrlr(3) = yft(Rindx(indx1)-3);
      IdCtrlr(4) = uft(Rindx(indx1)-1); IdCtrlr(5) = uft(Rindx(indx1)-2);
      IdCtrlr(6) = uft(Rindx(indx1)-3);
      IdCtrlr(7) = ydt(Rindx(indx1));
      [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = [ut(Rindx(indx1)-1)];
      [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
    end;
end;
save netex4f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq

%%%%%%%%%%%%%%%%%%%% TSTRG4F.M %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% To test the trained net: A input u(t) is fed into the 'unknown' system to generate a set of
% output data. The generated data are then used to feed the trained neural network to produce u^(t).
%
% Load the trained neural network
%
load netex4f
disp('Generating test input ...'); Nv = 200;
u = 0.1 .*(sin(2*pi*(1:Nv)/Nv)+sin(2*pi*(1:Nv).*2/Nv)-sin(2*pi*(1:Nv).*5/Nv));
%u = 0.1 * sign(sin(2*pi*5*(1:Nv)/Nv));

disp('Generating test output ...'); y=zeros(1,Nv);
for indx=3:Nv,
% Unknown plant
%
  y(indx+1) = (y(indx)*y(indx-1)*y(indx-2)*u(indx-1)*(y(indx-2)-1)+u(indx))/...
              (1+y(indx-1)^2+y(indx-2)^2);
end;

% Filtered signals using the observer as the filter
%
uf = filter(1, Alpha, u(:)); yf = filter(1, Alpha, y(:));

% Desired output
%
yd = filter(Dq, 1, y(:));

% Plotting the test data
%
clg; subplot(221);
plot(0:Nv-1,u); title('System Model 4: Test Input u(t)'); xlabel('Time Index'); grid;
plot(0:Nv,y); title('Test Output y(t)'); xlabel('Time Index'); grid;
plot(0:Nv-1,uf); title('Filtered Test Input uF(t)'); xlabel('Time Index'); grid;
plot(0:Nv,yf); title('Filtered Test Output yF(t)'); xlabel('Time Index'); grid;
!del ex42f.met
meta ex42f
```

```
uhat=zeros(1,Nv);
% Identifier Recalling
for indx=4:Nv
   IdCtrlr(1) = yf(indx-1); IdCtrlr(2) = yf(indx-2);
   IdCtrlr(3) = uf(indx-1); IdCtrlr(4) = uf(indx-2);
   IdCtrlr(5) = yd(indx);
   [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); uhat(indx-1) = IdCtrlr(OpIndx);
end;

% Plot the result comparing u(t) to u^(t)
clg;subplot(111); plot(0:Nv-1,u(1:Nv),0:Nv-1,uhat(1:Nv),'-');
title('System Model 4: Comparing Actual Input and N-Network Output');
xlabel('_____ Actual  _ _ _ _ NN O/P'); grid;
!del ex43f.met
meta ex43f

%%%%%%%%%%%%%%%%%%%%%%%%ONTRG4F.M%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the trained net
%
load netex4f
%
% Learning parameters for online learning
%
Learn = [0.4 0.2]; Moment = [0 0]; Lpar = [Learn Moment];
% Leave Npar unchanged

disp('Generating the reference signal ... '); Ns = 3000; Ts=(0:Ns-1)/Ns;
%
% Keep Ref small so the ym is between ± 1
%
Ref = 0.02*(0.5*sin(2*pi*Ts)+cos(2*pi*3*Ts)-1 - 0.3*sin(2*pi*11*Ts));
%Ref = [zeros(1,Ns/5), 0.1*ones(1,4*Ns/5)];
%Ref = 0.1 * sin(2*pi*3*Ts);
%Ref = 0.1*sign(sin(2*pi*5*Ts));
%Ref = zeros(1,Ns); %Ref(1:10) = 0.5*ones(1,10);
%Ref = 0.5*ones(1,Ns);

% Reference model output
%
ym = dlsim(1,Dq,Ref); clg; subplot(211);
plot(0:Ns-1,Ref); title('Reference Signal v(t)'); xlabel('Time Index'); grid;
plot(0:Ns-1,ym); title('Desired Reference Model Output ym(t)'); xlabel('Time Index'); grid;
!del ex44f.met
meta ex44f

% Initial Conditions
ys=zeros(1,Ns); us=zeros(1,Ns); ufs=zeros(1,Ns); yfs=zeros(1,Ns);

Onlin = input('(0) No Learning (1) Online Learning : ');
if Onlin == 1
   disp('Online Control and Learning ... ');
else
   disp('Online Control ... ');
end;
for indx=3:Ns-1
```

```
% Generate the control signal us
%
  IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
  IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
  IdCtrlr(5) = Ref(indx);
  [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); us(indx) = IdCtrlr(OpIndx);

% Update the plant
%
  ys(indx+1) = (ys(indx)*ys(indx-1)*ys(indx-2)*us(indx-1)* ...
            (ys(indx-2)-1)+us(indx))/(1 + ys(indx-1)^2 + ys(indx-2)^2);

% Filter u(indx) and y(indx) with the observer filter
%
  ufs(indx+1) = -Alpha(2:3)*[ufs(indx); ufs(indx-1)] + us(indx);
  yfs(indx+1) = -Alpha(2:3)*[yfs(indx); yfs(indx-1)] + ys(indx);
  yds(indx+1) = Dq*[ys(indx+1); ys(indx)];

% Identifier on-line learning
%
  if Onlin == 1
    IdCtrlr(1) = yfs(indx); IdCtrlr(2) = yfs(indx-1);
    IdCtrlr(3) = ufs(indx); IdCtrlr(4) = ufs(indx-1);
    IdCtrlr(5) = yds(indx+1);
    [IdCtrlr] = recall2f(Clayer,IdCtrlr,W1,W2,Npar); DoVec = us(indx);
    [W1,W2,dW1,dW2] = learn2f(Lpar,DoVec,Clayer,IdCtrlr,W1,W2,dW1,dW2,Npar);
  end;
end;

clg; plot(0:Ns-1, ys); grid; xlabel('Time Index');
title('System Model 4: Actual (___) and Desired Model (--) Outputs'); hold; plot(ym,'g--'); hold off
!del ex45f.met
meta ex45f
pause;

if Onlin == 1,
  tstrg4f; pause;
  Ch = input('Do you wish to save the online trained net: (Y) or (N) ? ', 's');
  if Ch == 'Y' | Ch == 'y',
    save netex4f IdCtrlr Clayer W1 W2 dW1 dW2 Npar OpIndx Alpha Dq
  end;
end;
```

APPENDIX D. BNN SOFTWARE SIMULATOR

```
function [Neurons,W1,W2,dW1,dW2]=net2f(Layer,R);
% function [Neurons,W1,W2,dW1,dW2]=net2f(Layer,R);
% This function generates the global data structure for a two-layer (excluding input connections) back-
% propagating neural network.
%
% The number of inputs, neurons in the hidden layer and output layer are specified by the vector Layer
% in Layer(1), Layer(2) and Layer(3) respectively.
%
% Returns
%   Neurons:
%       Array for storing the network inputs and the outputs of all neurons.
%   W1: Weights of input connections to neurons in hidden layer 1.
%   W2: Weights of input connections to neurons in output layer.
%       Weight elements are random numbers between -Range to Range (default 0.1). W1 and W2 include
%       one weight element for a biased input of 1 for each neuron.
%   dW1, dW2:
%       Working arrays to store the previous weight changes for W1 and W2 elements respectively
%       (used in the momentum term in the learning law).
%
% Teo Chin Hock. NPS.
% Date: 9 Oct 91.
% Version: 1.02
NInput=Layer(1); Nh1=Layer(2); NOutput=Layer(3);
% Total number of neurons;
NTotal = NInput + Nh1 + NOutput;
%
% Inputs/neuron outputs are assigned to the layer in the
% following order:
%                              Neuron/Input #
%                             ----------------------------
%       Input Connections     | 1 .... NInput
%       Hidden Layer Neurons  | (NInput+1)....(NInput+Nh1)
%       Output Layer Neurons  | (NInput+Nh1+1)....(NTotal)
%
% Zero all inputs/neuron outputs
Neurons = zeros(NTotal, 1);
%
% Initialise the weights to random numbers within Range
Range = 0.1; rand('uniform');
W1 = 2*Range*rand(Nh1, NInput+1) - Range; W2 = 2*Range*rand(NOutput, Nh1+1) - Range;
[M,N]=size(W1);
if M == 1,
  W1(M,N) = 0;
else
  Tmp = (0:M-1) + 0.5 - M/2; W1(:,N) = (2*R/M)*Tmp(:);
end;
[M,N]=size(W2);
if M == 1,
  W2(M,N) = R;
```

```
else
  Tmp = (0:M-1) + 0.5 - M/2; W2(:,N) = (2*R/M)*Tmp(:);
end;
dW1 = zeros(Nh1, NInput+1); dW2 = zeros(NOutput, Nh1+1);
%
disp(sprintf( '*** 2-Layer Back-Propagating Neural Network Created ***\n' ))
disp(sprintf( 'NInput  - Number of Input: %g', NInput))
disp(sprintf( 'Nh1     - Number of Neurons in Hidden Layer #1: %g', Nh1))
disp(sprintf( 'NOutput - Number of Output Neurons: %g', NOutput))
disp(sprintf( 'Wgts    - Connection Weights for Inputs to all Neurons are: ' ))
W1 = W1
W2 = W2
return;


function [Neurons,W1,W2,W3,dW1,dW2,dW3] = net3f(Layer,R);
% function [Neurons,W1,W2,W3,dW1,dW2,dW3] = net3f(Layer,R);
% This function generates the global data structure for a 3-layer (excluding input connections)
% back-propagating neural network. Bias weightings are set and evenly spaced between -R:R.
%
% Use this to create the backpropagating neural network to be used with recall3f.m and learn3f.m.
%
% The number of inputs, neurons in the hidden layers and output layer are specified by the vector
% Layer in Layer(1), Layer(2), Layer(3) and Layer(4) respectively.
%
% Returns
%   Neurons:
%       Array for storing the network inputs and the outputs of all neurons.
%   W1: Weights of input connections to neurons in hidden layer 1.
%   W2: Weights of input connections to neurons in hidden layer 3.
%   W3: Weights of input connections to neurons in output layer.
%       Weight elements are random numbers between -Range to Range (default 0.1). W1, W2 and W3 include
%       one weight element for a biased input of 1 for each neuron.
%   dW1, dW2, dW3:
%       Working arrays to store the previous weight changes for W1, W2 and W3 elements respectively
%       (used in the momentum term in the learning law).
%
% Teo Chin Hock. NPS.
% Date: 9 Oct 91.
NInput=Layer(1); Nh1=Layer(2); Nh2=Layer(3); NOutput=Layer(4);
% Total number of neurons;
NTotal = NInput + Nh1 + Nh2 + NOutput;
%
% Inputs/neuron outputs are assigned to the layer in the
% following order:
%                              Neuron/Input #
%                             -------------------------
%       Input Connections      | 1 .... NInput
%       Hidden Layer #1 Neurons | (NInput+1).....(NInput+Nh1)
%       Hidden Layer #2 Neurons | (NInput+Nh1+1)..(NInput+Nh1+Nh2)
%       Output Layer Neurons    | (NInput+Nh1+Nh2+1)..(NTotal)
%
% Zero all inputs/neuron outputs
Neurons = zeros(NTotal, 1);
%
```

```
% Initialise the weights to random numbers within Range
Range = 0.1; rand('uniform');
W1 = 2*Range*rand(Nh1, NInput+1) - Range;
W2 = 2*Range*rand(Nh2, Nh1+1) - Range;
W3 = 2*Range*rand(NOutput, Nh2+1) - Range;
[M,N]=size(W1);
if M == 1,
  W1(M,N) = 0;
else
  Tmp = (0:M-1) + 0.5 - M/2; W1(:,N) = (2*R/M)*Tmp(:);
end;
[M,N]=size(W2);
if M == 1,
  W2(M,N) = R;
else
  Tmp = (0:M-1) + 0.5 - M/2; W2(:,N) = (2*R/M)*Tmp(:);
end;
[M,N]=size(W3);
if M == 1,
  W3(M,N) = R;
else
  Tmp = (0:M-1) + 0.5 - M/2; W3(:,N) = (2*R/M)*Tmp(:);
end;
dW1 = zeros(Nh1, NInput+1); dW2 = zeros(Nh2, Nh1+1); dW3 = zeros(NOutput, Nh2+1);
%
disp(sprintf( '*** 3-Layer Back-Propagating Neural Network Created ***\n' ))
disp(sprintf( 'NInput  - Number of Input: %g', NInput))
disp(sprintf( 'Nh1     - Number of Neurons in Hidden Layer #1: %g', Nh1))
disp(sprintf( 'Nh2     - Number of Neurons in Hidden Layer #2: %g', Nh2))
disp(sprintf( 'NOutput - Number of Output Neurons: %g', NOutput))
disp(sprintf( 'Wgts    - Connection Weights for Inputs to all Neurons are: ' ))
W1 = W1
W2 = W2
W3 = W3
return;


function [W1,W2,dW1,dW2] = learn2f(P,DoVec,L,Nrons,W1,W2,dW1,dW2,Npar)
% function [W1,W2,dW1,dW2] = learn2f(P,DoVec,L,Nrons,W1,W2,dW1,dW2,Npar)
% This function facilitates back-propagation learning for the 2-layer neural network. The nonlinear
% mapping in each neuron is tanh(·). The bias weightings are fixed and evenly spaced between -R:R
% set using net2f.
%
% Requires:
%    P(arameters):  P(1,2) = Learning Rate, P(3,4) = Momemtum Rate
%    DoVec: The desired output column vector [d1 ; d2; ....; dNOutput]
%    N(eurons): Neuron outputs given the current input vector
%    L(ayer): L(1) = NInput, L(2) = Nh1, L(3) = NOutput
%    W1, W2: Connection weights
%    dW1, dW2: Previous changes in connection weights
%    Npar: Npar(1) = Bias on/off, Npar(2) = Gain = 1 (Not Used)
%
% Returns:
%    W1, W2: Updated connection weights
%    dW1, dW2: Work arrays for latest weight changes
```

```
% Teo Chin Hock.
% Date: 9 Oct 91.
% Version: 1.02
%
NTotal = length(Nrons); NL1 = L(1) + 1; NL2 = NL1 + L(2);
% Calculate the output error vector
ErrVec2 = DoVec - Nrons(NL2:NTotal);
% delta for the output layer.
delta2 = ErrVec2 .* (1 - Nrons(NL2:NTotal) .* Nrons(NL2:NTotal));
dW2 = P(2) .* (delta2 * [Nrons(NL1:(NL2-1)); Npar(1)]') + (P(4) .* dW2);
%
% delta for the hidden layer.
ErrVec1 = W2(:,1:L(2))' * delta2;
delta1 = ErrVec1 .* (1 - Nrons(NL1:(NL2-1)) .* Nrons(NL1:(NL2-1)));
dW1 = P(1) .* (delta1 * [Nrons(1:(NL1-1)); Npar(1)]') + (P(3) .* dW1);
%
% Updating the weights except the bias weighting
W1(:,1:L(1)) = W1(:,1:L(1)) + dW1(:,1:L(1)); W2(:,1:L(2)) = W2(:,1:L(2)) + dW2(:,1:L(2));
%
return;


function [W1,W2,W3,dW1,dW2,dW3]=learn3f(P,DoVec,L,Nrons,W1,W2,W3,dW1,dW2,dW3,Npar)
%function [W1,W2,W3,dW1,dW2,dW3]=learn3f(P,DoVec,L,Nrons,W1,W2,W3,dW1,dW2,dW3,Npar)
% This function facilitates back-propagation learning for the 3-layer neural network. The nonlinear
% mapping in each neuron is tanh(·). The bias weightings are fixed and evenly spaced between
% -R:R set using net3f.
%
% Requires:
%    P(arameters): Layer#1: P(1) = Learning Rate, P(4) = Momemtum Rate
%                  Layer#2: P(2) = Learning Rate, P(5) = Momemtum Rate
%                  Layer#3: P(3) = Learning Rate, P(6) = Momemtum Rate
%    DoVec: The desired output column vector [d1 ; d2; ....; dNOutput]
%    N(eurons): Neuron outputs given the current input vector
%    L(ayer): L(1) = NInput, L(2) = Nh1, L(3) = Nh2, L(4) = NOutput
%    W1, W2, W3: Connection weights
%    dW1, dW2, dW3: Previous changes in connection weights
%    Npar(ameters): Npar(1) = Bias, Npar(2) = Gain = 1 (Not used)
%
% Returns:
%    W1, W2, W3: Updated connection weights
%    dW1, dW2, dW3: Work arrays for latest weight changes
%
% Teo Chin Hock. NPS.
% Date: 9 Oct 91.
% Version: 1.0
NTotal = length(Nrons); NL1 = L(1) + 1; NL2 = NL1 + L(2); NL3 = NL2 + L(3);
% Calculate the output error vector
ErrVec3 = DoVec - Nrons(NL3:NTotal);
% delta for the output layer.
delta3 = ErrVec3 .* (1 - Nrons(NL3:NTotal) .* Nrons(NL3:NTotal));
dW3 = P(3) .* (delta3 * [Nrons(NL2:(NL3-1)); 1]') + (P(6) .* dW3);
%
% delta for the hidden layer #2.
ErrVec2 = W3(:,1:L(3))' * delta3;
```

```
delta2 = ErrVec2 .* (1 - Nrons(NL2:(NL3-1)) .* Nrons(NL2:(NL3-1)));
dW2 = P(2) .* (delta2 * [Nrons(NL1:(NL2-1)); 1]') + (P(5) .* dW2);
%
% delta for the hidden layer #1.
ErrVec1 = W2(:,1:L(2))' * delta2;
delta1 = ErrVec1 .* (1 - Nrons(NL1:(NL2-1)) .* Nrons(NL1:(NL2-1)));
dW1 = P(1) .* (delta1 * [Nrons(1:(NL1-1)); 1]') + (P(4) .* dW1);
%
% Updating the weights
W1(:,1:L(1)) = W1(:,1:L(1)) + dW1(:,1:L(1));
W2(:,1:L(2)) = W2(:,1:L(2)) + dW2(:,1:L(2));
W3(:,1:L(3)) = W3(:,1:L(3)) + dW3(:,1:L(3));
%
return;


function [Neurons]=recall2f(Layer,Neurons,W1,W2,Npar);
% function [Neurons]=recall2f(Layer,Neurons,W1,W2,Npar);
% Function to facilitate recall of the back-propagation neural network once. The nonlinear mapping
% in each neuron is tanh(·). The bias weightings are set and evenly spaced
% between -R:R using net3f.
%
% Type help learn2f for explanation of all parameters.
% Teo Chin Hock. NPS.
% Date: 9 Oct 91.
% Version: 1.02
NL1 = Layer(1) + 1; NL2 = NL1 + Layer(2); NTotal = sum(Layer);
% Calculate the outputs for first layer of the neurons
Summ = W1 * [Neurons(1:(NL1-1)); Npar(1)];
Neurons(NL1:(NL2-1)) = mtanh(Summ);
% Calculate the outputs for second layer of the neurons
Summ = W2 * [Neurons(NL1:(NL2-1)); Npar(1)];
Neurons(NL2:NTotal) = mtanh(Summ);
return;


function [Neurons]=recall3f(Layer,Neurons,W1,W2,W3,Npar);
% function [Neurons]=recall3f(Layer,Neurons,W1,W2,W3,Npar);
% Function to facilitate recall of the back-propagation neural network once. The nonlinear mapping
% in each neuron is tanh(·). The bias weightings are set and evenly spaced
% between -R:R using net3f.
%
% Type help learn3f for explanation of all parameters.
% Teo Chin Hock. NPS.
% Date: 9 Oct 91.
% Version: 1.0
NL1 = Layer(1) + 1; NL2 = NL1 + Layer(2); NL3 = NL2 + Layer(3); NTotal = sum(Layer);
% Calculate the outputs for first layer of the neurons
Summ = W1 * [Neurons(1:(NL1-1)); Npar(1)];
Neurons(NL1:(NL2-1)) = mtanh(Summ);
% Calculate the outputs for second layer of the neurons
Summ = W2 * [Neurons(NL1:(NL2-1)); Npar(1)];
Neurons(NL2:(NL3-1)) = mtanh(Summ);

% Calculate the outputs for output layer of the neurons
```

```
Summ = W3 * [Neurons(NL2:(NL3-1)); Npar(1)];
Neurons(NL3:NTotal) = mtanh(Summ);
return;


function [RIndx]=shuffle(Nelem);
% function [RIndx]=shuffle(Nelem)
% Returns a randomly shuffled index vector, RIndx, with Nelem elements. RIndx contains indices
% (from 1 to Nelem) randomly ordered.
% Teo Chin Hock. NPS.
% Date: 18 April 91.
rand('uniform'); WIndx= zeros(2,Nelem);
for i=1:Nelem,
  n = fix(Nelem*rand(1)) + 1;
  while WIndx(2,n) > 0,
    n = n + 1;
    if n > Nelem,
      n = 1;
    end;
  end;
  WIndx(1,n) = i; WIndx(2,n) = 1;
end;
RIndx = WIndx(1,:);
return;


function [t]=mtanh(d);
% A correct version of tanh().
% Written by Teo Chin Hock.
% NPS 29 July 1991.
%
dSign = sign(d);
t = (1 - exp(-2 .* abs(d))) ./ (1 + exp(-2 .* abs(d)));
t = dSign .* t;
return;
```

LIST OF REFERENCES

1 Åström, K.J. and Wittenmark, B., *Adaptive Control*, Addison-Wesley Publishing Company, 1989.

2 Narendra, K.S. and Parthasarathy, K., "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, Mar. 1990.

3 Thomas Kailath, *Linear Systems*, Prentice-Hall International, Inc., 1980.

4 Goodwin, G.C. and Sin, K.S., *Adaptive Filtering, Prediction and Control*, Prentice Hall, Inc., 1984.

5 Lennart Ljung, *"System Identification: Theory for the User"*, Prentice Hall, Inc., 1987.

6 K. Hornik, M. Stinchcombe, and H. White, "*Multi-layer feed-forward networks are universal approximators*," Dept. of Economics, UCSD, discussion paper, June, 1988.

7. NeuralWare, Inc., *Neural Computing*, documentation for the Neural Professional II Plus Neural Network Simulation Software, 1991.

8. Tolat, V.V., and Widrow, B., "An adaptive broom balancer with visual inputs," *Proceedings of the International Conference on Neural Networks,* II, 641-647, IEEE Press, New York, July 1988.

9. Nyugen, D. and Widrow, B., "The Truck Backer-Upper: An Example of Self-Learning in Neural Networks," *IJCNN-89*, Conference Record, July 1989.

10. Rumelhart, D.E., Hinton, G.E., and Williams, R.J., "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pp 318-362, MIT Press, Cambridge, MA, 1986.

11. Haykin, S., *"Adaptive Filter Theory,"* Prentice-Hall Inc., New Jersey, 1986.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5000	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Dr. A. J. Healey, Code ME Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
5.	Dr. Roberto Cristi, Code EC/Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Dr. Ralph Hippenstiel, Code EC/Hi Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
7.	Chief Defence Scientist MINDEF Singapore MINDEF Building, Gombak Drive, S2366 Republic of Singapore	1
8.	Head, Air Logistics HQ-RSAF, MINDEF MINDEF Building, Gombak Drive, S2366 Republic of Singapore	1
9.	Director Defence Science Organisation 20 Science Park Drive, S0511 Republic of Singapore	1

10. Director 1
Defence Materials Organisation
LEO Building, Paya Lebar Airport, S1953
Republic of Singapore

11. Mr. Paul Heckman, Code 943 1
Naval Ocean Systems Center
San Diego, California 92152

12. Mr. Robert Wilson 1
Head, Systems Engineering Branch
David Taylor Research Center
Carderock, Bethesda, Maryland 20084-5000

13. Mr. Dan Steiger 1
Marine Systems Group
Naval Research Laboratory
Washington, D.C. 20032

Thesis
T2755 Teo
c.1 Back-propagation neural networks in adaptive control of unknown nonlinear systems.