1992-06

Nonlinear adaptive control using backpropagating neural networks

Menke, Kurt William

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NONLINEAR ADAPTIVE CONTROL USING
BACKPROPAGATING NEURAL NETWORKS

by

Kurt William Menke

June, 1992

Thesis Advisor: Roberto Cristi

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY MULTIPLE SOURCES	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release: distribution is 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) Code 33	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)
NONLINEAR ADAPTIVE CONTROL USING BACKPROPAGATING NEURAL NETWORKS

12. PERSONAL AUTHOR(S)
Menke, Kurt W.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	June 1992	59

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			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Neural networks, Backpropagation, Adaptive Control System Design

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

The objective of this research is to develop a nonlinear regulator for an adaptive control system using backpropagating neural networks (BNN's) in conjunction with a linear quadratic regulator (LQR). The basic concepts of adaptive control and the structure of neural networks are discussed. These concepts are integrated and the nonlinear regulator is derived. Simulation is conducted on a representative nonlinear system with both the LQR and the nonlinear regulator. Training of the regulator and its performance under varying BNN parameter values are examined.

The simulation results show that the nonlinear regulator with BNN's exhibits superior performance compared to the LQR when the nonlinearities are large. The optimization of regulator performance with regard to BNN parameter values is discussed.

Further research is required in order to determine the general applicability of this regulator and to develop more specific guidelines for BNN parameters

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

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

Approved for public release; distribution is unlimited.

Nonlinear Adaptive Control Using
Backpropagating Neural Networks

by

Kurt William Menke
Lieutenant, United States Navy
B.S., United States Naval Academy

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June, 1992

Electrical and Computer Engineering

ABSTRACT

The objective of this research is to develop a nonlinear regulator for an adaptive control system using backpropagating neural networks (BNN's) in conjunction with a linear quadratic regulator (LQR). The basic concepts of adaptive control and the structure of neural networks are discussed. These concepts are integrated and the nonlinear regulator is derived. Simulation is conducted on a representative nonlinear system with both the LQR and the nonlinear regulator. Training of the regulator and its performance under varying BNN parameter values are examined.

The simulation results show that the nonlinear regulator with BNN's exhibits superior performance compared to the LQR when the nonlinearities are large. The optimization of regulator performance with regard to BNN parameter values is discussed.

Further research is required in order to determine the general applicability of this regulator and to develop more specific guidelines for BNN parameters.

TABLE OF CONTENTS

I. INTRODUCTION

A. OBJECTIVE

The objective of this thesis is to develop a nonlinear regulator for an adaptive control system using backpropagating neural networks (BNN's) in conjunction with a linear quadratic regulator (LQR). Discrete time models are used to represent the systems for simulation and analysis.

B. BASIC CONCEPTS OF ADAPTIVE CONTROL

There are inherent nonlinearities in most control systems due to elements such as environmental changes, minor system component failures, random time-varying parameters, and hard limits caused by physical constraints. These nonlinearities may be handled by an optimal control design if it is sufficiently robust, but with large uncertainties the controller has to sacrifice performance in favor of robustness and this might be unsatisfactory.

One method of handling these nonlinearities is to use an adaptive control system. An adaptive control system is one which continuously and automatically measures the dynamic characteristics of the plant, compares them with the desired attributes, and uses the difference to vary adjustable system parameters so that optimal performance can be maintained regardless of the nonlinearities encountered [Ref. 1: p. 792].

1. Performance Indexes

The basis of adaptive control rests on the premise that there is some performance of the system which is optimal. Optimal performance is defined by specifying a performance index to measure the closeness of the controlled system to its goal. There is an infinite number of possibilities in the choice of a performance index. The choice of a particular index depends on the system and the desired results. In most cases the choice of index involves a compromise of minimizing the system costs while maximizing the system performance.

The major drawback of performance indexes is while they specify the cost of system operation they do not give information about the transient response of the system [Ref. 1: p.793]. A system that operates optimally according to the performance index may have undesirable transient characteristics. The transient response of the system must be analyzed to validate the choice of weighting matrices used in the index.

2. Adaptive Control Systems

An adaptive control system may include the following three functions:

- Classification of the dynamic characteristics of the plant.
- Decision making based on the classification of the plant.
- Modification based upon the decisions made.

If the plant parameters are not exactly correct, then the initial classification, decision, and modification procedures will be insufficient to optimize the performance index. It is then necessary to continuously carry out these procedures throughout the period of operation. This constant redesign of the system identifies an adaptive control system [Ref. 1: p.793-794].

C. NEURAL NETWORKS IN ADAPTIVE CONTROL

While control theory for linear time-invariant (LTI) systems is a relatively mature field, nonlinear control systems generally must be designed on a system-to-system basis. Neural networks, with their inherent adaptability, have the potential for wide application in nonlinear control systems. The ability of a neural network to be trained suggests that a neural network may be able to successfully control nonlinear systems which have poor performance when regulated by linear time-invariant controllers. This thesis will investigate the use of neural networks in conjunction with LTI control methods to construct a nonlinear regulator [Ref. 2: p.410]. This regulator will be implemented on a system that may be modeled as LTI with an added nonlinearity which either makes control by LTI methods inefficient or unstable.

Two neural networks shall be used in the formulation of the regulator: one in parallel with the estimated plant

parameters to generate a better state vector estimate and one in parallel with the linear control to produce an improved control input to the system. Chapter II discusses a general neural network structure and the derivation of the nonlinear regulator. The performance of the regulator on a particular system is presented in Chapter III and conclusions are given in Chapter IV.

II. BNN NONLINEAR ADAPTIVE CONTROL

A. BASIC CONCEPTS OF NEURAL NETWORKS

Artificial neural networks are made up of elements which operate in a manner analogous to the most simple functions of biological neurons. These elements are linked in a fashion which may be similar to the connections within the human brain. Whether or not artificial neural networks actually are representative of the construction of the brain, they do show characteristics which are reminiscent of the human brain. An artificial neural network may be trained, it can recognize patterns, and it may apply its learning from past lessons to new data. These traits are quite limited, however they lend themselves to a wide field of applications.

1. The Artificial Neuron

The artificial neuron is meant to mimic the first-order characteristics of the biological neuron. A set of inputs is applied, each input representing the output of another neuron. Each input is multiplied by a corresponding linkweight and all of the weighted inputs are summed. This sum is the activation level of the neuron. Figure 1 shows a detailed representation of a single neuron.

In Figure 1 the inputs are labeled $x_1, x_2, \ldots, x_n$. These inputs are defined collectively as the $(n \times 1)$ column

vector x^{M-1}. Each input is multiplied by its corresponding linkweight a_1, a_2,...,a_n (row vector **a**, size $(1 \times N)$) and applied to the summation node [Ref. 3: pp.12-14]. The summation node produces a scalar output z_i^M which may be stated in vector notation:

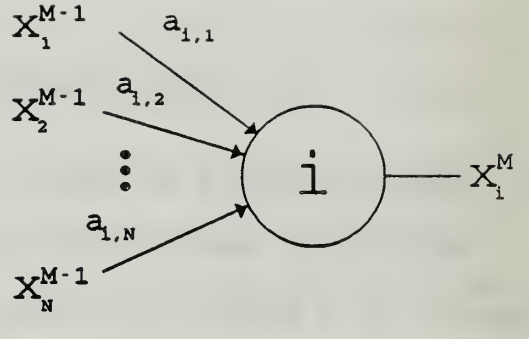

Figure 1. A Neuron

$$z_i^M = \mathbf{a}x^{M-1}. \qquad (2\text{-}1)$$

The output of the summation node, z_i^M, is further processed by an activation function f, to produce the neuron's output signal x_i^M:

$$x_i^M = f(z_i^M). \qquad (2\text{-}2)$$

There are numerous activation functions such as a simple threshold, a hard-limited linear function, the hyperbolic tangent, and the sigmoid. These functions are shown in Figure 2. The main purpose of the activation function is to serve as a nonlinear gain so that each neuron maps a wide range of inputs into a bounded output.

2. Single Layer Neural Networks

By themselves, neurons have limited capabilities, unless they are grouped together in layered networks. Figure

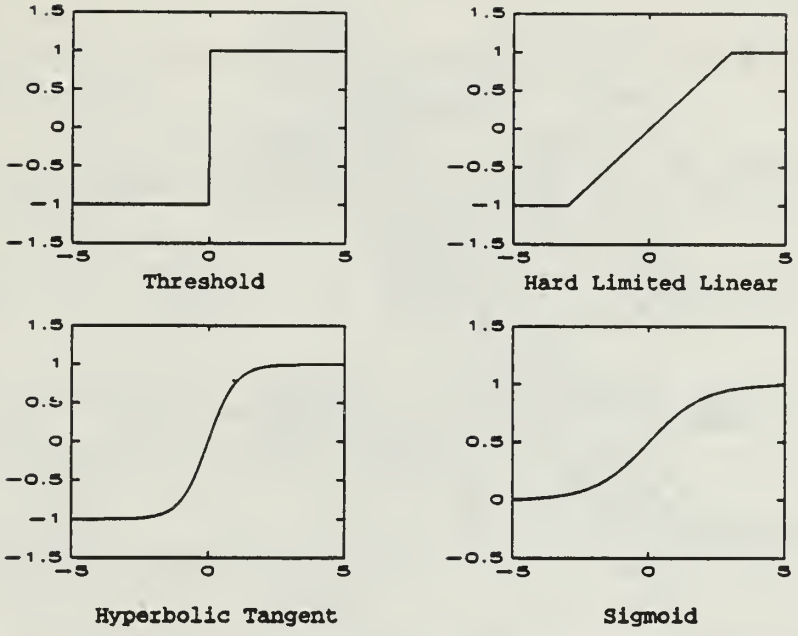

Figure 2. Various Activation Functions

3 shows a single layer neural network with its inputs on the far left of the figure and the outputs on the right. The single layer neural network only performs the vector multiplication; there is no activation function.

The circles on the left of Figure 3 serve only as distribution points for the inputs; they do not perform any calculations so they are not neurons. Each element of the input vector $\boldsymbol{x}$ is applied to each neuron. The linkweights may be considered as a matrix a with m rows and n columns where m is the number of outputs and n is the number of inputs. The number n is one greater than the dimension of the input vector $\boldsymbol{x}$ due to the bias term. Using equation (2-1), $\boldsymbol{z}$ is the ($m \times 1$) output vector:

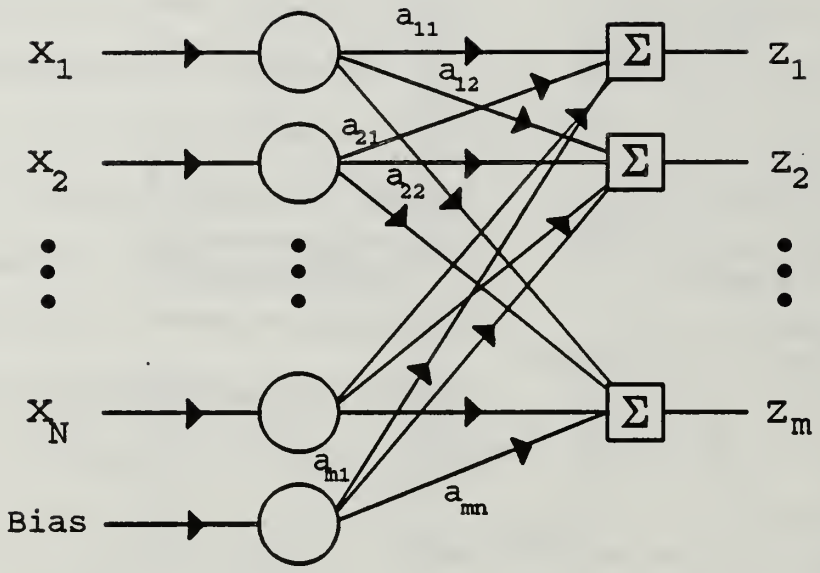

Figure 3. Single Layer Neural Network

$$z = ax. \tag{2-3}$$

The non-zero bias term is added as an input to each neuron. This bias input is also multiplied by its own linkweight.

3. Multiple Layer Neural Networks

Neural networks with multiple layers offer greater computational capabilities than single layer networks. These networks are formed by cascading a number of single layers with the outputs of one layer providing the inputs to the next layer. These middle layers are known as hidden layers. Figure 4 shows a multiple layer neural network. The key to providing the extra power of the multiple layer networks is

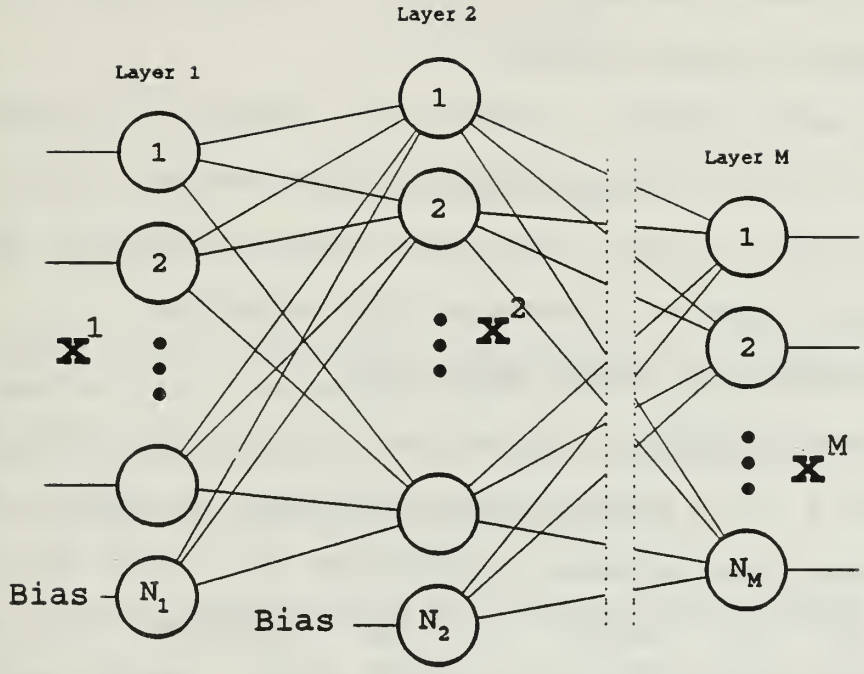

Figure 4. Multiple Layered Neural Network

that the activation function is included in the hidden layers of the network. Otherwise, the multiple layer network could be modeled by a single layer network which had a linkweight matrix equal to the product of the individual linkweight matrices [Ref. 3: p.19]. The output layer usually has the function $f(x) = x$, waiving the limits of the activation function.

4. Training of Neural Networks

A network is trained so that a set of inputs will produce a desired set of outputs. Training is accomplished by applying an input vector, computing the output vector, comparing it to the desired vector, and modifying the linkweights by a predetermined algorithm. As the network is

trained, the linkweights will converge to values which will produce the desired output vector.

There are several methods of training a neural network, one of which is the backpropagation routine. This is the algorithm used to train the neural networks used in this thesis. The next section discusses this technique.

5. Backpropagating Neural Networks

The backpropagating neural network is structured as shown in Figure 4. The signal flows from input to output. We assume no connections between the neurons of a layer and no feedback from any layer to the previous layers. Backpropagation refers to the method used to adjust the linkweights throughout the neural network.

The output of a neuron in the last layer (x_j^M) is used with a desired output (x_{tgt}) to produce an error signal (e_j^M). This error signal is multiplied by the first derivative of the activation function for that neuron. Mathematically,

$$\begin{aligned} \delta_{ij}^{M-1} &= f'(x_j^M)\,(x_j^M - x_{tgt}) \\ &= f'(x_j^M)\,(e_j^M)\,. \end{aligned} \qquad \textbf{(2-4)}$$

Then δ is multiplied by the output from the source neuron for the linkweight which is to be updated and in turn is multiplied by a scaling factor μ, the learning rate. This results in

$$\Delta_{ij}^{M-1} = \delta_{ij}^{M-1} x_i^{M-1} \qquad (2\text{-}5)$$

and

$$a_{ij,t+1}^{M-1} = a_{ij,t}^{M-1} - \mu \Delta_{ij}^{M-1}, \qquad (2\text{-}6)$$

where

- a_{ij}^{M-1} is the linkweight from neuron i in the layer $M-1$ to neuron j in the output layer M,
- δ_{ij}^{M-1} is the value of δ for the linkweight a_{ij}^{M-1},
- $t+1$ denotes the updated linkweight.

The linkweights in the hidden layers cannot be trained by this process, since there is no available target vector. Backpropagation trains the hidden layer linkweights by propagating the output error back through the network, adjusting the linkweights for each layer. Equations (2-4), (2-5), and (2-6) are still used for the hidden layers, but the target vector must be generated differently. The δ is calculated for each neuron in the output layer, as in Equation (2-4). The linkweights feeding the output layer are adjusted using Equations (2-5) and (2-6). The δ_{ij}^{M-1} is propagated back through the network to generate a value for δ for each neuron in the hidden layer. These values are used to adjust the weights of the preceding hidden layer, all the way back to the linkweights that act upon the inputs.

This is most easily shown in vector notation. The vector of δ's for the output layer is defined as D_{M-1} and the

set of linkweights for the output layer as the matrix A_{M-1}. To find D_{M-2}, multiply D_{M-1} by the transpose of the matrix A_{M-1}. Then multiply each component of this vector by the first derivative of the activation function for the corresponding neuron in the hidden layer. This yields the vector of δ's (D_{M-2}) for the hidden layer. Mathematically,

$$D_{M-2} = [D_{M-1} A_{M-1}^T] .* [f'(x^{M-1})], \qquad (2\text{-}7)$$

where .* denotes component-by-component multiplication of arrays [Ref. 3: pp.51-53].

B. INTEGRATION OF NEURAL NETWORKS AND ADAPTIVE CONTROL

As suggested earlier, the versatility of neural networks makes them prime candidates for nonlinear controllers. The vast majority of LTI systems are able to be controlled by linear quadratic regulators (LQRs). However, the LQR itself may not exhibit satisfactory performance in the presence of large uncertainties. That is, inaccuracies in the estimation of system parameters or nonlinearities in the system may cause the LQR to lose its optimality. In particular, nonlinear system dynamics can not be accounted for in LQR design.

The proposal here is to derive a nonlinear adaptive regulator which uses neural networks to compensate for the nonlinear dynamics. This regulator will include two backpropagating neural networks (BNN's): one for modeling and

one for control. The modeling BNN is used to find a more accurate state vector estimate than that found using the given system parameters. The control BNN modifies the control input by adding nonlinear terms to the LQR.

1. Linear Quadratic Regulator

The LQR is a state feedback controller which minimizes a performance index known as the cost function. Consider the LTI system:

$$\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t, \tag{2-8}$$

where $\boldsymbol{x}_t = (x_{1,t}, x_{2,t}, \ldots, x_{n,t})^T$ is the n-dimensional state vector, $\boldsymbol{u}_t = (u_{1,t}, u_{2,t}, \ldots, u_{m,t})^T$ is the m-dimensional control vector, and A and B are the system parameters of dimensions $(n \times n)$ and $(n \times m)$, respectively. The LQR is a regulator which minimizes the cost function:

$$E = \frac{1}{2}\sum_{t=1}^{\infty} (\boldsymbol{x}_t^T Q \boldsymbol{x}_t + \boldsymbol{u}_t^T R \boldsymbol{u}_t) \tag{2-9}$$

where Q is an $(n \times n)$ symmetric and positive semi-definite matrix and R is an $(m \times m)$ positive definite matrix. The optimal control u_t is defined by

$$\boldsymbol{u}_t = -K\boldsymbol{x}_t \,. \tag{2-10}$$

where

$$K = [B^T PB + R]^{-1} B^T PA. \qquad (2\text{-}11)$$

K is the $(m \times n)$ optimal feedback gain matrix and the $(n \times n)$ matrix P is the solution to the algebraic Riccati equation [Ref. 4: p.320]:

$$P = A^T PA + Q - A^T PB[B^T PB + R]^{-1} B^T PA. \qquad (2\text{-}12)$$

Obviously, the LQR does not take nonlinear system dynamics into consideration, but it does give a good baseline for comparison to the nonlinear regulator derived in the next section.

2. Backpropagating Neural Network Design

The system with the nonlinear regulator is shown in Figure 5. The modeling and control BNNs are labeled BNNM and BNNC, respectively. The modeling BNN is connected in parallel with the estimated plant parameters, labeled A_e and B_e. The control BNN is connected in parallel with the LQR, labeled K. The block labeled D denotes a unit time delay.

The nonlinear plant will be modeled as a linear time-invariant system with a nonlinearity added:

$$\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t + \boldsymbol{v}_t \qquad (2\text{-}13)$$

where $\boldsymbol{v}_t$ is the n-dimensional nonlinear function. First, a linear estimate of the state equation is made:

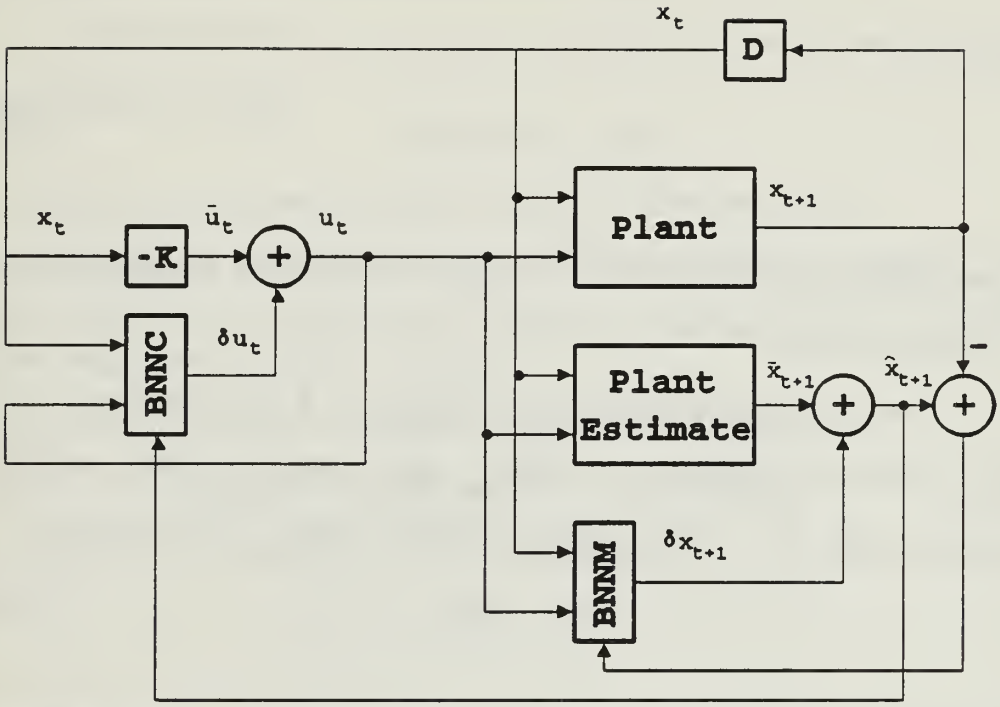

Figure 5. Regulator Block Diagram

$$\bar{\boldsymbol{x}}_{t+1} = A_e \boldsymbol{x}_t + B_e \boldsymbol{u}_t. \qquad (2\text{-}14)$$

The estimates of the system parameters may be computed by any number of methods. This linear estimate will deviate from the actual system output due to the nonlinear system dynamics and the inaccuracies in parameter estimation.

In order to compute a more accurate estimate of $\boldsymbol{x}_{t+1}$, the modeling BNN is used to produce an adjustment to $\bar{\boldsymbol{x}}_{t+1}$. The modeling BNN has the $(m + n)$-dimensional input vector $(\boldsymbol{u}_t, \boldsymbol{x}_t)$ and the n-dimensional output vector $\delta\boldsymbol{x}_{t+1}$. Therefore, the new estimate of $\boldsymbol{x}_{t+1}$ is:

$$\hat{\boldsymbol{x}}_{t+1} = \bar{\boldsymbol{x}}_{t+1} + \delta x_{t+1} \qquad (2\text{-}15)$$

where

$$\delta x_{t+1} = g(\boldsymbol{u}_t, \boldsymbol{x}_t) . \qquad (2\text{-}16)$$

The modeling BNN is trained so that the error between $\hat{\boldsymbol{x}}_{t+1}$ and $\boldsymbol{x}_{t+1}$ is minimized.

The optimal feedback gain matrix K is computed from A_e and B_e found in equation (2-14) and user-defined weighting matrices Q and R. The linear control input is computed:

$$\bar{\boldsymbol{u}}_t = -K\boldsymbol{x}_t . \qquad (2\text{-}17)$$

As discussed earlier, this controller is no longer optimal due to the nonlinear system dynamics and the parameter inaccuracies. The control BNN is used in order to optimize the control input. The control BNN has the n-dimensional input vector ($\boldsymbol{x}_t$) and the m-dimensional output vector $\delta \boldsymbol{u}_t$. The control input $\boldsymbol{u}_t$ is generated by

$$\boldsymbol{u}_t = \bar{\boldsymbol{u}}_t + \delta u_t \qquad (2\text{-}18)$$

where

$$\delta u_t = h(\boldsymbol{x}_t) . \qquad (2\text{-}19)$$

The control BNN is trained so that the control input $\boldsymbol{u}_t$ minimizes the cost function of equation (2-9).

3. BNN Training Algorithms

The following notation is used for both the modeling and the control BNN:

- n refers to a particular layer of the network;
- the ith node in the nth layer is designated *node(n, i)*;
- M is the total number of layers in the network, including the input and output layers;
- N_n is the number of nodes in the nth layer;
- $x^n_{i,t}$ is the output of the *node(n, i)* at time t;
- a^n_{ij} is the linkweight from the *node(n, j)* to the *node(n + 1, i)*.

The functional representation of the *node(n, i)* is found by:

$$z^n_{i,t} = \sum_{j=1}^{N_{n-1}} a^{n-1}_{i,j} x^{n-1}_{j,t} \qquad (2\text{-}20)$$

or

$$\mathbf{z}^n = a^{n-1}\mathbf{x}^{n-1} \qquad (2\text{-}21)$$

where a^{n-1} is the $(N_n \times N_{n-1})$-dimensional linkweight matrix and

$$x^n_{i,t} = f(z^n_{i,t}) \qquad (2\text{-}22)$$

where $x^n_{i,t}$ is the ith element of the vector input to layer n. The function $f(\bullet)$ is chosen to be the sigmoid function

$$f(x) = \frac{1}{1+e^{-x}} \qquad (2\text{-}23)$$

for the hidden layers and

$$f(x) = x \qquad (2\text{-}24)$$

for the output layer. In accord with this notation, the dimensions $N_I = m + n$ and $N_M = n$ in the modeling BNN, and $N_I = n$ and $N_M = m$ in the control BNN where n and m are the number of columns in the system matrices A and B, respectively.

a. Modeling BNN

In deriving the modeling BNN, the first step is to determine the error function which will be used for training. Since the goal is to produce an accurate estimate of $\boldsymbol{x}_{t+1}$, the error function is chosen to minimize the difference between $\hat{\boldsymbol{x}}_{t+1}$ and $\boldsymbol{x}_{t+1}$:

$$E^M_{t+1} = \frac{1}{2}(\hat{\boldsymbol{x}}_{t+1} - \boldsymbol{x}_{t+1})^T(\hat{\boldsymbol{x}}_{t+1} - \boldsymbol{x}_{t+1}). \qquad (2\text{-}25)$$

The linkweights are updated using a version of the gradient descent algorithm:

$$a^n_{i,j,t+1} = a^n_{i,j,t} - \mu_M \frac{\partial E^M_{t+1}}{\partial a^n_{i,j}} \qquad (2\text{-}26)$$

where μ_M is a step size parameter which controls the learning rate of the linkweights. The partial derivative of the cost function is computed as follows:

$$\begin{aligned}\frac{\partial E_{t+1}^{M}}{\partial a_{i,j}^{n}} &= \frac{\partial}{\partial a_{i,j}^{n}}\left[\frac{1}{2}(\hat{\boldsymbol{x}}_{t+1}-\boldsymbol{x}_{t+1})^{T}(\hat{\boldsymbol{x}}_{t+1}-\boldsymbol{x}_{t+1})\right] \\ &= (\hat{\boldsymbol{x}}_{t+1}-\boldsymbol{x}_{t+1})^{T}\frac{\partial}{\partial a_{i,j}^{n}}(\hat{\boldsymbol{x}}_{t+1}-\boldsymbol{x}_{t+1}) \\ &= (\hat{\boldsymbol{x}}_{t+1}-\boldsymbol{x}_{t+1})^{T}\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial a_{i,j}^{n}}\end{aligned} \tag{2-27}$$

where

$$\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial a_{i,j}^{n}} = \frac{\partial}{\partial a_{i,j}^{n}}(\bar{\boldsymbol{x}}_{t+1}+\delta x_{t+1}) . \tag{2-28}$$

Since $\bar{x}_{t+1}$ is independent of the linkweights:

$$\begin{aligned}\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial a_{i,j}^{n}} &= \frac{\partial}{\partial a_{i,j}^{n}}\delta x_{t+1} \\ &= \frac{\partial}{\partial a_{i,j}^{n}} g(\boldsymbol{u}_t, \boldsymbol{x}_t) .\end{aligned} \tag{2-29}$$

The function g is the mapping function for the layers of the network described earlier.

Therefore,

$$\frac{\partial E_{t+1}^{M}}{\partial a_{i,j}^{n}} = (\hat{\boldsymbol{x}}_{t+1}-\boldsymbol{x}_{t+1})^{T}\frac{\partial}{\partial a_{i,j}^{n}} g(\boldsymbol{u}_t, \boldsymbol{x}_t) . \tag{2-30}$$

The computation of $(\partial g/\partial a)$ is found next [Ref. 2: p.412]:

$$\frac{\partial}{\partial a_{i,j}^{n}} g(\boldsymbol{u}_t, \boldsymbol{x}_t) = \Delta_{i,t}^{n} x_{j,t}^{n} \tag{2-31}$$

where

$$\Delta_{i,t}^{n} = f'(z_{i,t}^{n+1}) \sum_{k=1}^{N_{n+2}} a_{k,i}^{n+1} \Delta_{k,t}^{n+1}. \tag{2-32}$$

The function $f'(x)$ denotes the first derivative of $f(x)$ at x. The initial condition on Δ is

$$\Delta_{i,t}^{M-1} = [0,\ldots,0,\overset{i}{1},0,\ldots,0]^T. \tag{2-33}$$

Therefore, the linkweight training is accomplished as:

$$a_{i,j,t+1}^{n} = a_{i,j,t}^{n} - \mu_M (\hat{\boldsymbol{x}}_{t+1} - \boldsymbol{x}_{t+1})^T \Delta_{i,t}^{n} x_{j,t}^{n} \tag{2-34}$$

b. Control BNN

Finding the error function for the control BNN is more involved than for the modeling BNN. If the plant was an LTI system, the control input $\boldsymbol{u}_t$ which minimizes

$$E_t^C = \frac{1}{2} (\boldsymbol{x}_{t+1}^T P \boldsymbol{x}_{t+1} + \boldsymbol{u}_t^T R \boldsymbol{u}_t) \tag{2-35}$$

is given by equations (2-10) through (2-12) [Ref. 2: p.413]. This implies that the LQR generates a control input which minimizes equation (2-35) for each time t. This control input

is suboptimal even if the system is LTI. The nonlinearities in the system make the LQR even less optimal. The control BNN is connected in parallel with the LQR to adjust for control errors caused by parameter inaccuracies and nonlinear system dynamics. Therefore an error function must be chosen which makes the output of the control BNN equal zero if the plant parameters are exactly expressed by A_e and B_e and the nonlinearities are set equal to zero. Equation (2-35) is a good choice for the error function at time t, but x_{t+1} is not available at time t. It is replaced by $\hat{x}_{t+1}$ which is found with the modeling BNN. The error function for the control BNN is now

$$E_t^C = \frac{1}{2}\left(\hat{x}_{t+1}^T P \hat{x}_{t+1} + u_t^T R u_t\right). \qquad (2\text{-}36)$$

The linkweight for the control BNN is defined as b^n_{ij} in order to differentiate it from the modeling BNN linkweight, a^n_{ij}. The control linkweights are updated by the gradient descent algorithm:

$$b^n_{i,j,t+1} = b^n_{i,j,t} - \mu_C \frac{\partial E^C_{t+1}}{\partial b^n_{i,j}} \qquad (2\text{-}37)$$

where μ_C is the learning rate parameter.

Equation (2-19) defines the nonlinear function for the control BNN. This is used to calculate the training algorithm as follows:

$$\frac{\partial E^{C}_{t+1}}{\partial b^{n}_{i,j}} = \frac{\partial}{\partial b^{n}_{i,j}} \left[\frac{1}{2} \left(\hat{\boldsymbol{x}}^{T}_{t+1} P \hat{\boldsymbol{x}}_{t+1} + \boldsymbol{u}^{T}_{t} R \boldsymbol{u}_{t} \right) \right] = \hat{\boldsymbol{x}}^{T}_{t+1} P \frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial b^{n}_{i,j}} + \boldsymbol{u}^{T}_{t} R \frac{\partial \boldsymbol{u}_{t}}{\partial b^{n}_{i,j}} . \tag{2-38}$$

First, $(\partial \hat{\boldsymbol{x}}_{t+1}/\partial b)$ will be derived followed by the derivation of $(\partial \boldsymbol{u}_{t}/\partial b)$. Substituting equation (2-15) into $(\partial \hat{\boldsymbol{x}}_{t+1}/\partial b)$ gives:

$$\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial b^{n}_{i,j}} = \frac{\partial}{\partial b^{n}_{i,j}} \left(\bar{\boldsymbol{x}}_{t+1} + \delta x_{t+1} \right) . \tag{2-39}$$

Next, equations (2-14) and (2-18) are used:

$$\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial b^{n}_{i,j}} = \frac{\partial}{\partial b^{n}_{i,j}} \left(A_{e} \boldsymbol{x}_{t} + B_{e} \boldsymbol{u}_{t} + \delta x_{t+1} \right) = \frac{\partial}{\partial b^{n}_{i,j}} \left(A_{e} \boldsymbol{x}_{t} + B_{e} (\bar{\boldsymbol{u}}_{t} + \delta u_{t}) + \delta x_{t+1} \right) \tag{2-40}$$

but $\boldsymbol{x}_{t}$ and $\bar{\boldsymbol{u}}_{t}$ are independent of b, therefore

$$\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial b^{n}_{i,j}} = \frac{\partial}{\partial b^{n}_{i,j}} \left(B_{e} \delta u_{t} + \delta x_{t+1} \right) . \tag{2-41}$$

Substituting in equations (2-19) and (2-16) results in

$$\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial b^{n}_{i,j}} = B_{e} \frac{\partial h(\boldsymbol{x}_{t})}{\partial b^{n}_{i,j}} + \frac{\partial g(\boldsymbol{u}_{t}, \boldsymbol{x}_{t})}{\partial b^{n}_{i,j}} . \tag{2-42}$$

The second partial derivative in equation (2-42) is the partial derivative of a function of u_t which is itself dependent upon b, therefore the derivative must be split as follows:

$$\frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial b_{i,j}^{n}} = B_e \frac{\partial h(\boldsymbol{x}_t)}{\partial b_{i,j}^{n}} + \frac{\partial g(\boldsymbol{u}_t, \boldsymbol{x}_t)}{\partial \boldsymbol{u}_t} \frac{\partial \boldsymbol{u}_t}{\partial b_{i,j}^{n}}. \tag{2-43}$$

But the partial derivative of u_t may be further simplified by:

$$\begin{aligned} \frac{\partial \boldsymbol{u}_t}{\partial b_{i,j}^{n}} &= \frac{\partial (\bar{\boldsymbol{u}}_t + \delta u_t)}{\partial b_{i,j}^{n}} \\ &= \frac{\partial (\delta u_t)}{\partial b_{i,j}^{n}} \\ &= \frac{\partial h(\boldsymbol{x}_t)}{\partial b_{i,j}^{n}}. \end{aligned} \tag{2-44}$$

Substituting equation (2-44) into equation (2-43):

$$\begin{aligned} \frac{\partial \hat{\boldsymbol{x}}_{t+1}}{\partial b_{i,j}^{n}} &= B_e \frac{\partial h(\boldsymbol{x}_t)}{\partial b_{i,j}^{n}} + \frac{\partial g(\boldsymbol{u}_t, \boldsymbol{x}_t)}{\partial \boldsymbol{u}_t} \frac{\partial h(\boldsymbol{x}_t)}{\partial b_{i,j}^{n}} \\ &= \left(B_e + \frac{\partial g(\boldsymbol{u}_t, \boldsymbol{x}_t)}{\partial \boldsymbol{u}_t} \right) \frac{\partial h(\boldsymbol{x}_t)}{\partial b_{i,j}^{n}}. \end{aligned} \tag{2-45}$$

Using equation (2-31), $(\partial \boldsymbol{h}(\boldsymbol{x}_t)/\partial b)$ may be computed in the same manner as $(\partial \boldsymbol{g}(\boldsymbol{u}_t, \boldsymbol{x}_t)/\partial a)$. The $(n \times m)$ matrix $(\partial \boldsymbol{g}(\boldsymbol{u}_t, \boldsymbol{x}_t)/\partial \boldsymbol{u}_t)$ is found by [Ref. 2: p.413]:

$$\frac{\partial g(\boldsymbol{u}_t, \boldsymbol{x}_t)}{\partial \boldsymbol{u}_t} = \sum_{i=1}^{N_{M-1}} \Delta_{i,t}^{M-2} \sum_{j=1}^{N_{M-2}} a_{i,j}^{M-2}. \tag{2-46}$$

The control linkweight training algorithm may be summarized as:

$$b^n_{i,j,t+1} = b^n_{i,j,t} - \mu_c \left[\hat{x}^T_{t+1} P \left(B_e + \frac{\partial g(u_t, x_t)}{\partial u_t} \right) + u^T_t R \right] \Delta^n_{i,t} x^n_{j,t}. \quad \text{(2-47)}$$

The training algorithm for both BNNs may be summarized as follows:

- observe the state vector x_t at time t,
- update the linkweight matrices a, using equation (2-34),
- compute the control input u_t with equations (2-17), (2-18) and (2-19),
- calculate the state vector estimate $\hat{x}_{t+1}$ using equations (2-14), (2-15), and (2-16),
- update the linkweight matrices b, using equation (2-47).

This is repeated at each successive time t, as the new state vector becomes available.

III. SIMULATION RESULTS AND DISCUSSION

A. NONLINEAR SYSTEM DESIGN

In order to test the LQR and the nonlinear regulator, a nonlinear system was required to be modeled. This system was modeled with a linear part which was then acted upon by a nonlinear function. The linear portion was artificially estimated to provide a more realistic case study. The linear system used as a baseline was first presented as a transfer function in continuous time, transformed to a state space representation, and then converted to discrete time.

The chosen transfer function was unstable with poles at +2.00 and -1.00, a zero at +0.10, and a gain of 10:

$$\frac{Y(s)}{U(s)} = \frac{10(s-0.1)}{(s+1)(s-2)}. \qquad (3\text{-}1)$$

The equivalent of this transfer function in state space format is [Ref. 1: pp.675-677]:

$$\dot{\boldsymbol{y}} = \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \boldsymbol{y} + \begin{bmatrix} 10 \\ 9 \end{bmatrix} u \qquad (3\text{-}2)$$
$$= A_c \boldsymbol{y} + B_c u.$$

This equation is converted to discrete time using a truncated infinite series to calculate the state transition matrix and input matrix [Ref. 4: pp.123-127] with a sample time of 0.1 seconds. The discrete time equation is:

$$\begin{aligned} \boldsymbol{y}_{t+1} &= \begin{bmatrix} 1.0104 & 0.1055 \\ 0.2110 & 1.1159 \end{bmatrix} \boldsymbol{y}_t + \begin{bmatrix} 1.0500 \\ 1.0533 \end{bmatrix} u_t \\ &= A\boldsymbol{y}_t + Bu_t. \end{aligned} \tag{3-3}$$

Estimated values (A_e and B_e) were found for the purpose of illustrating the relative robustness of the LQR and the nonlinear regulator. These values were found by taking the members of A and B and calculating values which differed from the actual numbers by ±10-20 percent.

The nonlinear part of the system equation was modeled as follows:

$$\begin{aligned} x_{1,t+1} &= y_{1,t+1} + \epsilon(1 - \exp(y_{2,t+1})) \\ x_{2,t+1} &= y_{2,t+1} + \epsilon(1 - \exp(y_{1,t+1})). \end{aligned} \tag{3-4}$$

The value ϵ is a control on the level of nonlinearity.

B. LQR SIMULATION RESULTS

The LQR is calculated using A_e and B_e. The weighting matrices Q and R are set at:

$$Q = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad and \quad R = 1. \tag{3-5}$$

The solutions to equations (2-11) and (2-12) are:

$$K = [0.1775 \;\; 0.5057] \quad and \quad P = \begin{bmatrix} 0.0771 & 0.1060 \\ 0.1060 & 1.3859 \end{bmatrix}. \qquad (3\text{-}6)$$

Figures 6 and 7 illustrate the performance of the LQR to a unit impulse. Figure 6 shows the system response for ϵ = 0.05. The state variables quickly converge to zero. Figure 7 shows the response to the same input for ϵ = 0.20. It can be seen that the LQR is not sufficiently robust to cause the state variables to converge to zero. The plant converges to a nonzero equilibrium value.

C. NONLINEAR REGULATOR SIMULATION RESULTS

Both BNN's used here are three-layered neural networks with 30 nodes in the hidden layer. In the modeling BNN, N_1 = $m + n$ = 3, N_2 = 30, and N_3 = n = 2. In the control BNN, N_1 = n = 2, N_2 = 30, and N_3 = m = 1. The learning rate for both BNN's was set at 0.05. The initial condition on the linkweight matrices was to fill them with small random numbers (normal distribution, standard deviation = 0.1).

Figure 8 plots simulation results for the nonlinear plant with ϵ = 0.05. The results are quite similar to those of Figure 6. For this level of nonlinearity, the nonlinear regulator causes the state variables to converge to zero

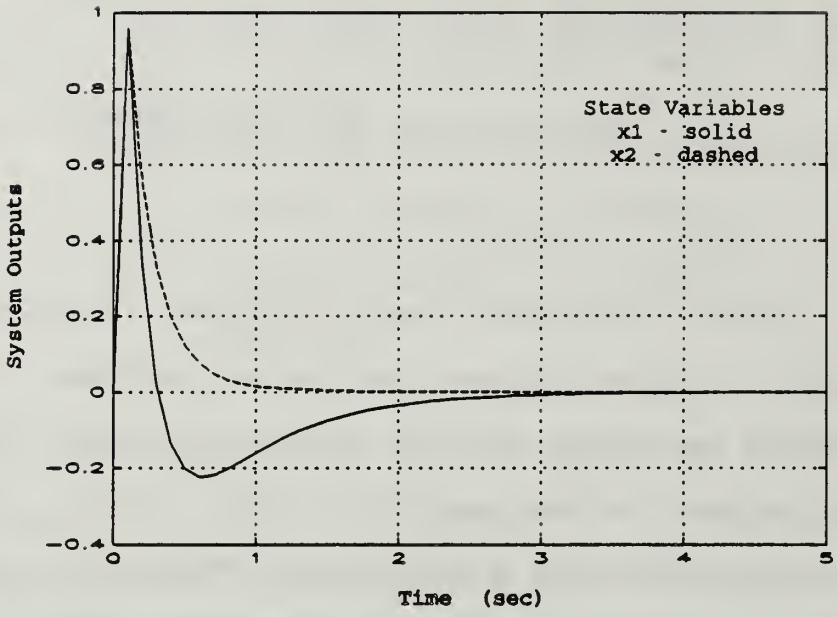

Figure 6. LQR Result for $\epsilon = 0.05$

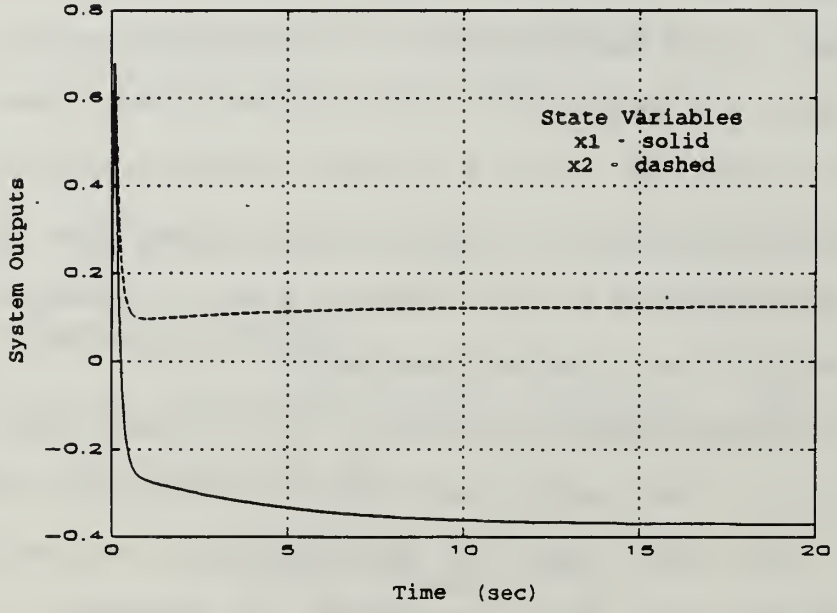

Figure 7. LQR Result for $\epsilon = 0.20$

faster than the LQR, but the response is slightly more oscillatory.

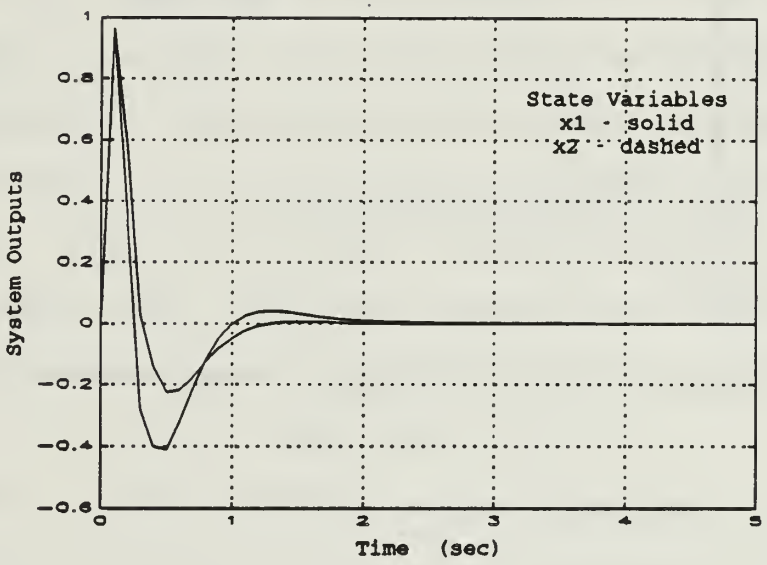

Figure 8. Nonlinear Regulator Result for $\epsilon = 0.05$

Figure 9 shows simulation results for the nonlinear plant with $\epsilon = 0.20$. Comparing these results to those of Figure 7 shows that the nonlinear regulator reduces the state variables to zero in a case where the LQR is unable to do so. While the LQR stabilizes the system for relatively large factors of ϵ, it is of limited utility once the state variables no longer converge to zero. The state variables converge to zero using the LQR up to $\epsilon = 0.18$ while the nonlinear regulator continues to drive the state variables to zero up to $\epsilon = 0.21$. The performance of the nonlinear regulator is better than the LQR for $\epsilon \geq 0.06$.

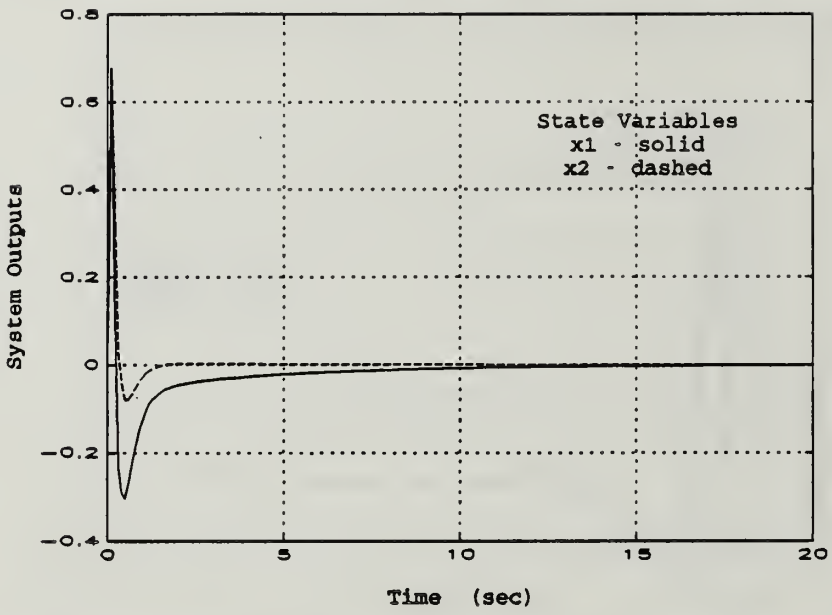

Figure 9. Nonlinear Regulator Result for $\epsilon = 0.20$

1. BNN Parameter Variation

The next area of investigation was to vary some of the parameters of the neural network to examine the effects upon the regulator. This was accomplished by holding all parameters constant with the exception of the one in question. The performance index W is found by summing the absolute values of the state variables ($x_{1,t}$ and $x_{2,t}$) for the specified time range. The performance index W was calculated for each instance, again for a unit impulse over a range of 20 seconds.

One of the major variances in the performance of neural networks is caused by the number of nodes in the hidden layer of the network. This number (N_2) was varied from 10 to

40. The results are plotted in Figure 10. The straight line (without point markers) shows the performance of the LQR with no neural network. Figure 10 shows that adding more nodes to the hidden layer improves the performance of the regulator. However, when ϵ is small, the LQR yields better performance than the nonlinear regulator with any number of nodes. As ϵ increases above 0.15, the performance degrades and the number of nodes has less of an effect. There are limits upon the number of nodes used. When the regulator was tried with 50 nodes in the hidden layer, the system became unstable.

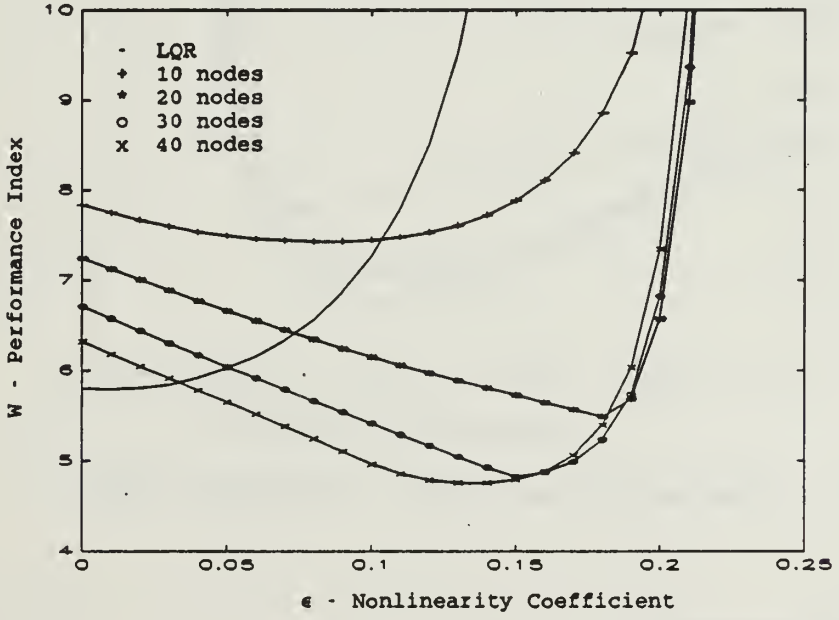

Figure 10. Variation in *W* due to Number of Nodes

The learning rate was also varied to determine its effect upon regulator performance. This step size parameter

had an effect on W similar to that of varying N_2. Both the modeling and the control BNN's had their learning rate changed to the same value. Figure 11 shows the effect of this upon W. An interesting feature of Figure 11 is that while increasing the learning rate caused better performance, this was only true up to $\mu = 0.05$. When μ was increased beyond this, performance was degraded. Increasing μ to 0.10 caused the system to become unstable.

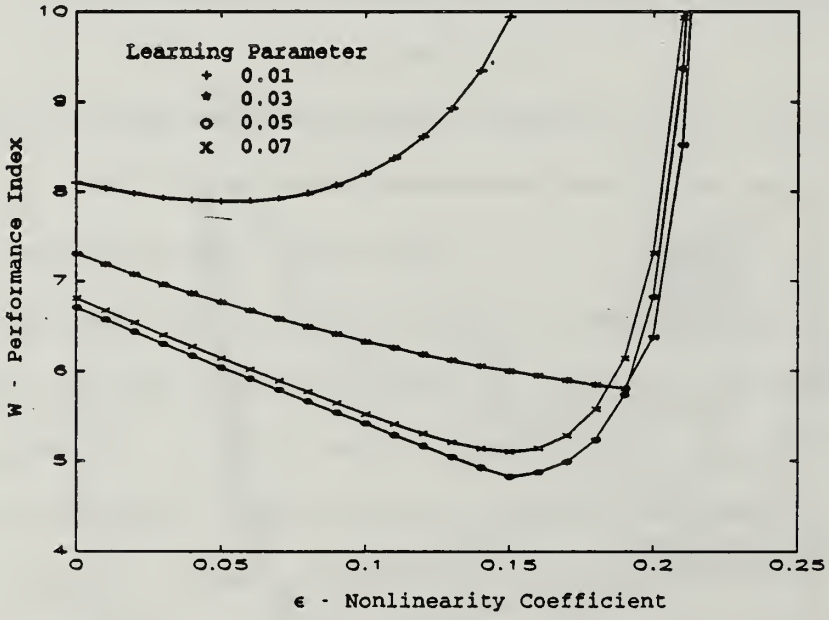

Figure 11. Variation in W due to Learning Rate

2. BNN Training

One of the advantages of this regulator design is that there is very little training required in order to achieve

convergence of the linkweight values. All simulation runs were completed with a single unit impulse training run. One of the major drawbacks to the majority of neural network applications is the amount of training inputs required for convergence [Ref. 3: p.88]. The apparent reason for this rapid convergence is that the BNN's are not providing all of the state vector estimate and control input: they are only adding a correction to the linear state vector estimate and the LQR control input.

IV. CONCLUSIONS

A. SUMMARY

Starting with the rationale for developing an adaptive regulator for nonlinear systems, the development of a nonlinear regulator which used backpropagating neural networks in conjunction with a linear quadratic regulator in an adaptive control system was proposed. The regulator was derived and applied to control a representative nonlinear system.

B. SIGNIFICANT RESULTS

Simulations of the BNN-based nonlinear regulator were conducted on a representative nonlinear system. The main observations from these simulations are summarized below:

- The results show that the nonlinear regulator works well in the control of a nonlinear system. It was also seen that using a LQR on the system works if the nonlinearity is small.
- There are many variables involved in the regulator design which must be optimized by trial and error. Definitive rules to govern the selection of these variables would significantly decrease the time to arrive at an appropriate controller.
- The amount of training required for the regulator is minimal. One pulse is sufficient to train the network.
- This regulator is unproven for all nonlinear systems. Its utility may be limited to those which may be modeled as a linear system with an added nonlinear component.

C. FURTHER RESEARCH

The emphasis in this thesis was to develop a BNN-based nonlinear regulator that would use the *a priori* knowledge of system parameters to find a controller that could function more efficiently than one which did not utilize this knowledge. The neural networks used in this research only contained one hidden layer. While this was sufficient, the rules used in deriving the BNN's for the regulator could be used to design neural networks with numerous hidden layers. Guidelines should be developed which would govern the choice of the number of layers used in the neural networks, the number of nodes necessary in the hidden layers, the learning parameters, and the weighting matrices for the error functions.

APPENDIX A: NONLINEAR REGULATOR SIMULATION

A. SYSTEM DERIVATION

A nonlinear regulator using backpropagating neural networks in conjunction with a linear quadratic regulator was designed in Section B of Chapter II. This appendix details the procedure used to arrive at the simulation results of Chapter II and includes the software written for the research.

The representative nonlinear system chosen for examination has the continuous domain transfer function:

$$\frac{Y(s)}{U(s)} = \frac{10(s-0.1)}{(s+1)(s-2)} = \frac{10s-1}{s^2-s-2}. \qquad \text{(A-1)}$$

This equation must be transformed to state space format. This is accomplished using the derivation shown below [Ref. 1: p.675-677].

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^n + b_1 s^{n-1} + \cdots + b_{n-1} s + b_n}{s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n} \qquad \text{(A-2)}$$

This may be rearranged and transformed to an nth-order system of linear differential equations:

$$y^{(n)} + a_1 y^{(n-1)} + \cdots + a_n y = b_0 u^{(n)} + b_1 u^{(n-1)} + \cdots + b_n u \qquad \text{(A-3)}$$

where $y^{(n)}$ and $u^{(n)}$ are the nth order derivatives of y and u. The state variable is chosen:

$$\begin{aligned} x_1 &= y \\ \dot{x}_1 &= x_2 \\ \dot{x}_2 &= x_3 \\ &\vdots \\ \dot{x}^n &= -a_n x_1 - a_{n-1} x_2 - \cdots - a_1 x_n + b_0 u^{(n)} + b_1 u^{(n-1)} + \cdots + b_n u. \end{aligned} \tag{A-4}$$

However, $x_1 = y$ may not yield a unique solution due to the derivatives of the forcing function. The state variable are redefined as

$$\begin{aligned} x_1 &= y - \beta_0 u \\ x_2 &= \dot{y} - \beta_0 \dot{u} - \beta_1 u = \dot{x}_1 - \beta_1 u \\ x_3 &= \ddot{y} - \beta_0 \ddot{u} - \beta_1 \dot{u} - \beta_2 u = \dot{x}_2 - \beta_2 u \\ &\vdots \\ x_n &= y^{(n-1)} - \beta_0 u^{(n-1)} - \cdots - \beta_{n-1} u = \dot{x}_{n-1} - \beta_{n-1} u \end{aligned} \tag{A-5}$$

where $\beta_0, \beta_1, \ldots, \beta_n$ are determined from

$$\begin{aligned} \beta_0 &= b_0 \\ \beta_1 &= b_1 - a_1 \beta_0 \\ \beta_2 &= b_2 - a_1 \beta_1 - a_2 \beta_0 \\ &\vdots \\ \beta_n &= b_n - a_1 \beta_{n-1} - \cdots - a_{n-1} \beta_1 - a_n \beta_0. \end{aligned} \tag{A-6}$$

With this choice of state variables, the following state and output equations are found:

$$\dot{\boldsymbol{x}} = A_c\boldsymbol{x} + B_c u$$
$$y = C_c\boldsymbol{x} + D_c u \qquad \text{(A-7)}$$

where

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}, \qquad A_c = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix}$$

$$B_c = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}, \qquad C_c = [1 \;\; 0 \;\; \cdots \;\; 0], \qquad D_c = \beta_0 = b_0 . \qquad \text{(A-8)}$$

B. SIMULATION SOFTWARE

```
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%
%                               LQR.M
%
%    This program computes a linear quadratic regulator for the
%    stated system and simulates the response to a unit
%    impulse.  No neural network adjustment is provided.
%    This is a stand-alone program.
%    LT Kurt Menke, 10 June 1992
%
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%
%
clear; clg
%
%
%    The first part of this program converts the system
%    from continuous time transfer function to discrete
%    domain state space equations.
%
%
%    The numbers for num and den are for the
%    representative system of Chapter III.  This is set up
%    for a second order system with a single input.
%    It may be adapted for any order system using the equations
%    in Appendix A.
num = [0 10 -1];
den = [1 -1 -2];
%
%
%    This transformation to state space follows the
%    format of Appendix A.
a1 = den(1,2);
a2 = den(1,3);
b0 = num(1,1);
b1 = num(1,2);
b2 = num(1,3);
beta0 = b0;
beta1 = b1 - a1*beta0;
beta2 = b2 -a1*beta1 -a2*beta0;
%
%
```

```
%   Continuous state space equations
Ac = [ 0   1;
     -a2 -a1];
Bc = [beta1; beta2];
Cc = [1  0];
Dc = beta0;
%
%disp('Sample time for continuous')
dt = input('to discrete conversion?  ');
%
%
%   Discrete state space equations
[A, B] = c2d(Ac,Bc,dt);
%
%
[na,ma] = size(A);
[nb,mb] = size(B);
rand('uniform')
rand('seed',0)
%
%
%   Ar, Br, As, and Bs are used to compute the "estimate" of
%   A and B used in the computation of the LQR.
%   This ensures that Ae and Be are within +/- 10-20%
%   of A and B
Ar = 0.1 .* rand(na,ma) + 0.1;
Br = 0.1 .* rand(nb,mb) + 0.1;
%
%
%   As and Bs are arbitrary signs which make the particular
%   value of Ar/Br either +10-20% or -10-20%
As = [-1 1; 1 -1];
Bs = [-1; 1];
Ar = As .* Ar;
Br = Bs .* Br;
%
%
%   Ae and Be are the estimates of A and B
Ae = (Ar .* A) + A;
Be = (Br .* B) + B;
%
%
%   Weighting matrices
Q = [0 0; 0 1];
R = 1;
%
%
%   Computes the LQR gain matrix (K) and the solution to the
%   algebraic Riccati equation (P).
[K,P] = dlqr(Ae,Be,Q,R);
```

```
%
%
%   e is the coefficient which controls the amount of
%   nonlinearity seen by the system.
e = input('Nonlinearity coefficient?  ');
%
%%  Recommended run time:  10 or 20 seconds.
time = input('Amount of system run time?  ');
%
%
%   Initial values
Nt   = time/dt + 1;
xhat = zeros(na,Nt);
y    = zeros(na,Nt);
x    = zeros(na,Nt);
u    = [1 zeros(1,Nt-1)];  % Impulse input at time 0.
%
%
for t = 2:Nt
  y(:,t)    = A*x(:,t-1) + B*u(:,t-1);
  %
  % State vector
  x(1,t)    = y(1,t) + e*(1-exp(y(2,t)));
  x(2,t)    = y(2,t) + e*(1-exp(y(1,t)));
  %
  % Control input
  u(:,t)    = -K*x(:,t);
  %
  % State vector estimate
  xhat(:,t+1) = Ae*x(:,t) + Be*u(:,t);
end;
%
%
%   Performance calculation
W1 = 0;
W2 = 0;
for t = 1:Nt
  W1 = W1 + abs(x(1,t));
  W2 = W2 + abs(x(2,t));
end;
W = W1 + W2;
%
%
%   Plot of state vector response to unit impulse
t=0:dt:time;
plot(t,x(1,:),t,x(2,:)),grid;
text(0.8,0.8,num2str(W),'sc');
title('Linear Quadratic Regulator');
xlabel('Time (sec)');
ylabel('System Output');
```

```
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%
%                              NLREG.M
%
%    This program computes a nonlinear regulator for the stated
%    system and simulates the response to a unit impulse.  The
%    regulator uses two neural networks:  BNNC and BNNM.  This
%    is a stand-alone program.
%    LT Kurt Menke, 10 June 1992
%
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%
%
clear; clg;
%
%
%    The first part of this program converts the system
%    from continuous time transfer function to discrete
%    domain state space equations.
%
%
%    The numbers for num and den are for the
%    representative system of Chapter III.  This is set up
%    for a second order system with a single input.
%    It may be adapted for any order system using the equations
%    in Appendix A.
num = 10 * [0 1 -.1];
den = conv([1 -2],[1 1]);
%
%
%    This transformation to state space follows the
%    format of Appendix A.
a1 = den(1,2);
a2 = den(1,3);
b0 = num(1,1);
b1 = num(1,2);
b2 = num(1,3);
beta0 = b0;
beta1 = b1 - a1*beta0;
beta2 = b2 -a1*beta1 -a2*beta0;
%
%
%    Continuous state space equations
Ac = [ 0   1;
      -a2 -a1];
Bc = [beta1; beta2];
Cc = [1 0];
Dc = beta0;
%
%
```

```
disp('Sample time for continuous')
dt = input('to discrete conversion?  ');
%
%
%   Discrete state space equations
[A, B] = c2d(Ac,Bc,dt);
%
%
[na,ma] = size(A);
[nb,mb] = size(B);
rand('uniform')
rand('seed',0)
%
%
%   Ar, Br, As, and Bs are used to compute the "estimate" of
%   A and B used in the computation of the LQR.
%   This ensures that Ae and Be are within +/- 10-20%
%   of A and B
Ar = 0.1 .* rand(na,ma) + 0.1;
Br = 0.1 .* rand(nb,mb) + 0.1;
%
%
%   As and Bs are arbitrary signs which make the particular
%   value of Ar/Br either +10-20% or -10-20%
As = [-1 1;1 -1];
Bs = [-1;1];
Ar = As .* Ar;
Br = Bs .* Br;
%
%
%   Ae and Be are the estimates of A and B
Ae = (Ar .* A) + A;
Be = (Br .* B) + B;
%
%
%   Weighting matrices
Q = [0 0; 0 1];
R = 1;
%
%
%   Computes the LQR gain matrix (K) and the solution to the
%   algebraic Riccati equation (P).
[K,P] = dlqr(Ae,Be,Q,R);
%
%
%   Learning rate parameters.  LearnM is for BNNM and LearnC
%   is for BNNC.
LearnM = 0.05;
LearnC = 0.05;
%
%
```

```
%   e is the coefficient which controls the amount of
%   nonlinearity seen by the system.
e = input('Nonlinearity coefficient?  ');
%
%
%   Recommended run time:  10 or 20 seconds.
time = input('Amount of system run time?  ');
%
%
disp('Number of nodes in hidden layer')
Nm = input('of modeling net (BNNM)?  ');
%
%
disp('Number of nodes in hidden layer')
Nc = input('of control net (BNNC)?  ');
%
%
%   Initialization of the linkweight matrices
[a1,a2] = netinitm(na,Nm,na+2);
[b1,b2] = netinitm(mb,Nc,nb+1);
%
%
%   Bias value for neural networks
Bias = 1;
%
%
%   Initial values
Nt = time/dt + 1;
x    = zeros(na,Nt);
y    = zeros(na,Nt);
xbar = zeros(na,Nt+1);
xhat = zeros(na,Nt+1);
xdel = zeros(na,Nt+1);
ex   = zeros(na,Nt);
u    = [1 zeros(mb,Nt-1)];  % Impulse input at time 0
%
%
%   Training run
for t = 2:Nt
  y(:,t)       = A*x(:,t-1) + B*u(:,t-1);
  % State vector
  x(1,t)       = y(1,t) + e*(1-exp(y(2,t)));
  x(2,t)       = y(2,t) + e*(1-exp(y(1,t)));
  % Error vector
  ex(:,t)      = x(:,t)-xhat(:,t);
  % Training of linkweights for BNNM
  [a1,a2] = bpm(a1,a2,[x(:,t);u(:,t);Bias],ex(:,t),Learnm);
  % Linear control input
  ubar(:,t)    = -K*x(:,t);
  % BNNC output
  udel(:,t)    = netm([x(:,t); Bias], b1, b2);
```

```
  % Control input
  u(:,t)       = ubar(:,t) + udel(:,t);
  % Linear state vector estimate
  xbar(:,t+1) = Ae*x(:,t) + Be*u(:,t);
  % BNNM output
  xdel(:,t+1) = netm([x(:,t); u(:,t); Bias], a1, a2);
  % State vector estimate
  xhat(:,t+1) = xbar(:,t+1) + xdel(:,t+1);
  % Training of linkweights for BNNC
  [b1,b2]     = bpc(a1,a2,b1,b2,x(:,t),u(:,t),xhat(:,t+1),...
                    Bias,P,Be,R,Learn);
end;
%
%
%   Resetting initial values
x    = zeros(na,Nt);
y    = zeros(na,Nt);
xbar = zeros(na,Nt+1);
xhat = zeros(na,Nt+1);
xdel = zeros(na,Nt+1);
ex   = zeros(na,Nt);
ubar = zeros(mb,Nt);
udel = zeros(mb,Nt);
u    = [1 zeros(mb,Nt-1)];  % Impulse input at time 0
%
%
%   Simulation run
for t = 2:Nt
  y(:,t)       = A*x(:,t-1) + B*u(:,t-1);
  % State vector
  x(1,t)       = y(1,t) + e*(1-exp(y(2,t)));
  x(2,t)       = y(2,t) + e*(1-exp(y(1,t)));
  % Error vector
  ex(:,t)      = x(:,t)-xhat(:,t);
  % Training of linkweights for BNNM
  [a1,a2] = bpm(a1,a2,[x(:,t);u(:,t);Bias],ex(:,t),Learnm);
  % Linear control input
  ubar(:,t)    = -K*x(:,t);
  % BNNC output
  udel(:,t)    = netm([x(:,t); Bias], b1, b2);
  % Control input
  u(:,t)       = ubar(:,t) + udel(:,t);
  % Linear state vector estimate
  xbar(:,t+1) = Ae*x(:,t) + Be*u(:,t);
  % BNNM output
  xdel(:,t+1) = netm([x(:,t); u(:,t); Bias], a1, a2);
  % State vector estimate
  xhat(:,t+1) = xbar(:,t+1) + xdel(:,t+1);
  % Training of linkweights for BNNC
  [b1,b2]     = bpc(a1,a2,b1,b2,x(:,t),u(:,t),xhat(:,t+1),...
                    Bias,P,Be,R,Learn);
```

```
end;
%
%
%   Performance calculation
W1 = 0;
W2 = 0;
for t = 1:Nt
  W1 = W1 + abs(x(1,t));
  W2 = W2 + abs(x(2,t));
end;
W = W1 + W2;
%
%
%   Plot of state vector response to unit impulse
t=0:dt:time;
plot(t,x(1,:),t,x(2,:)),grid;
text(0.8,0.8,num2str(W),'sc');
title('Linear Quadratic Regulator');
xlabel('Time (sec)');
ylabel('System Output');
```

The following programs are the subroutines required to run LQR.M and NLREG.M.

```
function [a1,a2] = netinitm(ny,M,nu)
%
%   Routine for initializing the neural net with
%   small random numbers.
%
%   Function call:  [a1,a2] = netinitm(ny,M,nu)
%
%   where ny = number of outputs
%          M = number of nodes in the hidden layer
%         nu = number of inputs
%
%   LT Kurt Menke, 10 June 1992
%
rand('normal')
rand('seed',0)
a1 = 0.1*rand(M,nu);
a2 = 0.1*rand(ny,M);
return
```

```
function x3 = netm(x1,a1,a2)
%
%   Routine for calculating the output of a neural net.
%
%   Function call:  x3 = net(x1,a1,a2)
%
%   where x1 = neural net input vector
%         a1 = linkweight matrix from input to hidden layer
%         a2 = linkweight matrix from hidden layer to output
%
%   LT Kurt Menke, 10 June 1992
%
x2 = sigmoid(a1*x1);
x3 = a2*x2;
return
```

```
function y = sigmoid(x)
%
%   Routine for calculating the sigmoid of a vector input
%
%   Function call:  y = sigmoid(x)
%
%   where x = vector input
%         y = vector output
%
%   LT Kurt Menke, 10 June 1992
%
y = 1 ./ (1+exp(-x));
return


function y = dsig(x)
%
%   Routine for calculating the first derivative of the
%       sigmoid of a vector input
%
%   Function call:  y = dsig(x)
%
%   where x = vector input
%         y = vector output
%
%   LT Kurt Menke, 10 June 1992
%
temp = exp(-x);
y = temp./ (1 + 2*temp + temp.*temp);
return;
```

```
function [a1,a2] = bpm(a1,a2,x1,ex,mu)
%
%   Routine for updating the linkweight matrices for BNNM.
%
%   Function call:  [a1,a2] = bpm(a1,a2,x1,ex,mu)
%
%   where a1 = linkweight matrix from input to hidden layer
%         a2 = linkweight matrix from hidden layer to output
%         x1 = neural net input vector
%         ex = error vector for linkweight adjustment
%         mu = learning rate for BNNM
%
%   LT Kurt Menke, 10 June 1992
%
z2 = a1*x1;
x2 = sigmoid(z2);
z3 = a2*x2;
%
parE_a1 = -diag(dsig(z2))*a2'*ex*x1';
parE_a2 = -ex*x2';
%
a1 = a1 - mu.*parE_a1;
a2 = a2 - mu.*parE_a2;
return
```

```
function [b1,b2] = bpc(a1,a2,b1,b2,x,u,xhat,Bias,P,B,R,mu)
%
%   Routine for updating the linkweight matrices for BNNC.
%
%   Function call:  [b1,b2] = bpm(a1,a2,b1,b2,x,u,xhat,...
%                                 Bias,P,B,R,mu)
%
%   where a1 = BNNM linkweights from input to hidden layer
%         a2 = BNNM linkweights from hidden layer to output
%          x = neural net input vector
%          u = control input
%       xhat = state vector estimate
%       Bias = bias value for neural net
%          P = matrix solution to algebraic Riccati equation
%          B = system input matrix
%          R = weighting factor
%         mu = learning rate for BNNC
%
%   LT Kurt Menke, 10 June 1992
%
x1a = [x; u; Bias];
z2a = a1*x1a;
x2a = sigmoid(z2a);
z3a = a2*x2a;
%
x1b = [x; Bias];
z2b = b1*x1b;
x2b = sigmoid(z2b);
z3b = b2*x2b;
%
parh_b2 = x2b';
parh_b1 = diag(dsig(z2b))*b2'*x1b';
%
del1a = a2*diag(dsig(z2a));
parg_h = del1a*(sum(a1'))';
parE_c = xhat'*P*(B+parg_h) + u'*R;
%
deb2 = parE_c.*parh_b2;
deb1 = parE_c.*parh_b1;
%
b1 = b1 - mu.*deb1;
b2 = b2 - mu.*deb2;
return
```

LIST OF REFERENCES

1. Ogata, K., *Modern Control Engineering*, Prentice-Hall, Inc., 1970.

2. Iiguni, Y., Sakai, H., and Tokumaru, H., "A Nonlinear Regulator Design in the Presence of System Uncertainties Using Multilayered Neural Networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 4, July 1991.

3. Wasserman, P.D., *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, 1989.

4. Phillips, C.L., and Nagle, H.T., *Digital Control System Analysis and Design*, Prentice-Hall,Inc., 1984.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Dr. Roberto Cristi, Code EC/Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
5.	LT Kurt Menke Naval Submarine School Code 80 SOAC Class 92070 Box 700 Groton, Connecticut 06349-5700	1