(2)

Contract N00014-89-C-0137
Task Final Technical Report
Contract LIN 0003, 0004, 0005, 0006 (complete)
and LIN 0001, 0002 (partial)

Active Control of Complex Systems Via Dynamic (Recurrent) Neural Networks

David G. Ward
B. Eugene Parker, Jr., Ph.D.
Roger L. Barron

May 30, 1992

Prepared for:

DEPARTMENT OF THE NAVY
Office of the Chief of Naval Research
Applied Research and Technology Division
800 North Quincy Street
Arlington, Virginia 22217-5000

Prepared by:

BARRON ASSOCIATES, INC.
Route 1, Box 159
Stanardsville, Virginia 22973

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-018

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Same as report **This document has been approved for public release and sale; its distribution is unlimited.**
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 141-02/03-F	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Barron Associates, Inc. — 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of the Chief of Naval Research Applied Research and Technology Division
6c. ADDRESS (City, State, and ZIP Code) Route 1, Box 159 Stanardsville, VA 22973-9511	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-89-C-0137
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. — PROJECT NO. — TASK NO. — WORK UNIT ACCESSION

11. TITLE (Include Security Classification)

Active Control of Complex Systems Via Dynamic (Recurrent) Neural Networks

12. PERSONAL AUTHOR(S)
David G. Ward, B. Eugene Parker, Jr. Ph.D., and Roger L. Barron

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Task Final Report	FROM 4-15-89 TO 5-30-92	1992 May 30	

16. SUPPLEMENTARY NOTATION

Contract Line Item Nos. 0001-0006, inclusive

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Artificial Neural Networks — Estimation
Active Control — Classification
Dynamic Neural Networks — (Continued on reverse)

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

In this work the synthesis of artificial neural networks is examined from the perspective of statistical estimation of functions, and development of synthesis algorithms is centered on new tools for building dynamic (recurrent) neural networks that incorporate internal feedbacks and time delays. The *DynNet* algorithm is described; it learns the feedforward and feedback structure of a nonlinear dynamic neural network and optimizes the coefficients therein. Applications of the algorithm are presented for the following areas:

- time-series predictions related to an advanced turbopropulsion combustion process
- rapid predictions of the responses of a synchronous generator to changes in its input and load conditions
- predictions of the behavior of a deterministic chaotic process
- on-line, real-time, optimal two-point boundary-value guidance of an air-to-air missile

The report outlines the advantages of dynamic neural networks and probes the issues related to their synthesis and use.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) — 22c. OFFICE SYMBOL

3. DISTRIBUTION

ADDRESSEE	DODAAD CODE	NUMBER OF COPIES
Scientific Officer, OCNR	N00014	1
Administrative Contracting Officer, DCAS	S2101A	1
Director, Naval Research Laboratory, ATTN: Code 2627	N00173	1
Defense Technical Information Center Bldg. 5, Cameron Station Alexandria, VA 22314	S47031	12

18. SUBJECT TERMS

Recurrent Neural Networks
Statistical Estimation of Functions
Learning Algorithms
Modeling
Prediction
Chaotic Systems
Combustion Control
Synchronous Machines
Missile Guidance

FOREWORD

Under the terms of Contract N00014-89-C-0137 with the Office of the Chief of Naval Research, Barron Associates, Inc. (BAI) has, since April 15, 1989, engaged in study of the synthesis and use of artificial neural networks, particularly neural networks containing internal feedback pathways along with the customary feedforward connections. These are here called dynamic neural networks, and in the literature are generally referred to as recurrent neural networks. This work has been supported primarily under the OCNR Accelerated Research Initiative in Active Control, and the applications that here receive greatest attention relate to improved combustion control for turbopropulsion systems, a focus of the Active Control Initiative. Additionally, support has been provided by the Defense Advanced Research Projects Agency via the above contract to investigate applications of neural networks (principally dynamic neural networks) to improved modeling and prediction of the responses of a synchronous generator to changes in its input and load conditions.

This report is the final technical report on BAI's work under the Active Control Initiative and the synchronous generator task. This report fulfills the requirements of Line Item Nos. 0003 and 0004, and the funded portion of the work scope under Line Item Nos. 0005 and 0006 of the subject contract. This report also partially fulfills the requirements of Line Item Nos. 0001 and 0002. Under modifications to the contract, BAI has also performed and is continuing research in several areas involving static and dynamic neural networks. These areas, which include transient signal processing, acoustic radiation, and multivariable control, will be documented in future technical reports for OCNR.

Many individuals in the Government, in industry, and in academia have contributed importantly to the work reported. The OCNR Scientific Officers for this work have been Dr. Robert J. Hansen (now with The Pennsylvania State University), Dr. Eric W. Hendricks, and Mr. James G. Smith. This work would not have been accomplished without their guidance, support, and encouragement, for which the authors are deeply appreciative. Also, the support and direction of the Submarine Technology Program of DARPA under Captain Theodore L. Rice, USN, is gratefully acknowledged, as is the help of Mr. David Young of the Planning Research Corporation DARPA Office.

The authors thank Drs. Ephraim Gutmark and Klaus W. Schadow of the Naval Air Warfare Center, Weapons Division, China Lake, CA, for their participation in the experimental phases of the work reported herein in Section 4 and Appendix E, and other collaborative work being reported separately by them. Likewise, the authors thank Professors Craig T. Bowman and J. David Powell of the Mechanical Engineering Department, Stanford University, for collaborative work being reported separately by them.

Within BAI, the work of Messrs. Dean W. Abbott, Richard L. Cellucci, and Todd M. Nigro has been of considerable benefit to earlier phases of this research. Dr. Andrew R. Barron, Associate Professor of Statistics and of Electrical and Computer Engineering, University of Illinois, has provided invaluable advice on information-theoretic aspects of neural network synthesis. The authors also thank Dr. H. Vincent Poor, Professor of Electrical Engineering, Princeton Unviersity, for his critique of the manuscript. David G. Ward has written Sections 2 and 3 of this report. Dr. B. Eugene Parker has prepared Section 4. All authors of this report are staff members of Barron Associates, Inc. Roger L. Barron served as Principal Investigator for BAI.

Furthermore, the authors express their appreciation to G. Clark Smith of the BAI staff for assistance in preparing Appendix A, and to Susan W. Reynolds of the BAI staff for the word processing involved in preparation of this report.

The continuing research by BAI under the subject contract is now under the direction of OCNR Code 122 Scientific Officers Dr. Albert J. Tucker, Dr. Eric W. Hendricks, and Commander Dan Forkel, USN.

ABSTRACT

In this work the synthesis of artificial neural networks is examined from the perspective of statistical estimation of functions, and development of synthesis algorithms is centered on new tools for building dynamic (recurrent) neural networks that incorporate internal feedbacks and time delays. The *DynNet* algorithm is described; it learns the feedforward and feedback structure of a nonlinear dynamic neural network and optimizes the coefficients therein. Applications of the algorithm are presented for the following areas:

- time-series predictions related to an advanced turbopropulsion combustion process
- rapid predictions of the responses of a synchronous generator to changes in its input and load conditions
- predictions of the behavior of a deterministic chaotic process
- on-line, real-time, optimal two-point boundary-value guidance of an air-to-air missile

The report outlines the advantages of dynamic neural networks and probes the issues related to their synthesis and use.

TABLE OF CONTENTS

List of Figures

List of Tables

1. INTRODUCTION AND SUMMARY

Artificial neural networks (ANNs), although envisioned in fictional literature long before, became a subject of serious scientific study in the early 1940s with the publication by McCulloch and Pitts of their work to describe the functioning of neurons and simple aggregates of neurons in terms of mathematical logic [McCulloch and Pitts, 1943].† The McCulloch-Pitts *formal neuron* was a mathematical construct particularly suitable for descriptions and analyses of feedforward networks of neurons, that is, networks in which the propagation of information (encoded as neuronal impulses) was unidirectional from input to output. Emphasis was thus placed from the beginning of ANN research on the type of networks that produce *static*, i.e., feedforward, transformations. A static ANN transformation is not explicitly dependent on its own prior outputs, but is subject, of course, to on-going changes brought about by learning processes. As research in neurodynamics proceeded, the concept of reverberatory information flows received attention [Hebb, 1949]. Reverberatory neuronal networks have internal connections that provide feedback pathways conducting information from the outputs of specific neurons to the inputs of the same or other neurons stimulated at the same or earlier stages in a reverberatory sequence of neuronal firings. It was recognized by Hebb and others that reverberatory loops could provide a basis for temporary retention of information within a neural network; could exhibit, in general, coupled oscillatory modes of learned responses evoked by input patterns; and could organize, adapt, and maintain complex, learned psychomotor functions in living organisms.

Despite the appeal of reverberatory artificial neural networks as a possible means for the control of complex engineering systems, the ANN research community has grappled for 50 years with the technical issues that surround synthesis and use of static neural networks. This concentration on static transformations has been prudent, partly because it was several decades before crucial interdisciplinary connections would be completed between the rigorous theory and algorithms of statistical modeling and the developing field of artificial neural networks [see A.R. Barron and R.L. Barron, 1988]. Many fruitful engineering applications of ANNs had been realized by 1980 [R.L. Barron, et al., 1984], but these early applications generally proceeded more from intuition than solid theoretical foundations. Since approximately 1980, statistical learning theory has found its way into the ANN domain, and in this fertile territory has taken strong root, as discussed in Section 2 of this report. The advent of statistical learning theory within the ANN community is now having a profound influence on development of procedures for training and using artificial neural networks.

---

† References cited in body of this report appear on pages 95–101.

In the opinion of the authors, a major area of further advances during the 1990s will be that of reverberatory artificial neural networks, used particularly to process time-series data (signals) and to control complex, high-order, multivariable processes. The now-perceived advantages of reverberatory ANNs, here called *dynamic* ANNs, but often referred to in the literature as *recurrent* ANNs, are:

1. Judging from the experience obtained with dynamic networks in this project, fewer degrees of freedom are required internal to the ANN for a given modeling/prediction/control application using dynamic networks than when using static networks. This translates into greater robustness, reduced network training time, reduced size of the training database, and reduced on-line computational burden.
2. The analyst can be relieved of many design decisions concerning the optimum strategies for sampling and holding prior measures of the states of controlled processes, because dynamic ANNs can be synthesized via algorithms that automatically learn what information is to be retained and for how long.
3. Dynamic network synthesis algorithms can automatically blend data from multiple time series, using just the appropriate phase and magnitude and the optimum linear and nonlinear interaction terms.
4. Dynamic networks can compute a time-series estimate (output) with *or without* time-varying inputs. Dynamic networks can thus compute predictions that are not limited to a small number of discrete forecast horizons. Viewed more generally, estimated time series can be computed with or without time-varying input patterns.
5. Dynamic networks can readily constitute infinite impulse response (IIR) filters — not available with static (purely feedforward) ANNs — in addition to providing finite impulse response (FIR) solutions. This has important consequences for signal processing and control.
6. Information-theoretic modeling criteria and network synthesis procedures developed for static ANN syntheses have applicability (with suitable extensions) to dynamic networks. The ANN lessons of the first 50 years therefore have carry-over to dynamic ANNs.
7. Dynamic networks can provide more general solutions that embrace those obtained via static networks and reduce, automatically, to static networks where such are optimal.
8. As with static networks, off-line learning and on-line adaptation and learning are realizable.

Among the first work in dynamic networks was that of Rumelhart, Hinton, and Williams, who described *recurrent* (dynamic) backpropagation (BP) network [1986]. A generalization of the recurrent BP network is presented by Hecht-Nielsen

[1989]. A recent paper by Kuan and Hornik [1992] describes some of the issues concerning recurrent BP networks as follows:

> "A fundamental difference between feedforward and recurrent networks is the presence of internal feedbacks R_t, which are also a function of network connection weights θ. In a recurrent network, the standard BP type of algorithms are *not* valid because they fail to take into account the parameter dependence of R_t ... R_t is a very complex function of θ. This motivates the study of the recurrent BP algorithm (Kuan, Hornik, and White (1990)).
>
> "The recurrent BP algorithm shares all the advantages and disadvantages of the standard BP algorithm; in particular, it converges rather slowly. By adding a Newton direction to the recurrent BP algorithm we can obtain a computationally efficient algorithm."

Other investigators have (in static and dynamic network synthesis algorithms) optimized the entire set of coefficients in a pre-structured network. The algorithms they have employed have, for the most part, been inherently unimodal (subject to trapping on local minima). In the present formulation, the authors have sought to provide computationally efficient *structure learning and weight optimization* synthesis algorithms for dynamic networks, including capabilities for global optimization of the weights. We have also endeavored to gain preliminary understanding of the capabilities and limitations of the new algorithms, largely through investigation of several applications arising in complex prediction and control problems. It is doubtful that the algorithms presented are equally suitable for all applications, but they have shown themselves to be highly effective in the cases considered.

Section 2 of this report presents a perspective on ANN synthesis based upon the statistical estimation of functions. This thinking is relevant to static and dynamic networks and is a product of the research by A.R. Barron [see references]. Section 3 introduces the computer algorithms *Dyn3* and *DynNet*, developed in this project for the purpose of dynamic ANN syntheses. Application to time-series predictions for an advanced turbopropulsion combustion process and for a synchronous generator are detailed in Section 4. Algorithms for static ANN estimation and classification function learning (refined during this project) are described in Appendix A, while Appendix B outlines a numerical optimization algorithm that has been useful in some of the syntheses of dynamic networks. Appendix C summarizes results of a brief study of the use of ANNs for prediction of the behavior of a deterministic chaotic process. Appendix D presents an application study of the synthesis and simulated performance of a dynamic ANN studied for possible on-line, real-time implementation of an optimal, two-point boundary-value guidance law for a tactical air-to-air missile. It is shown that the dynamic ANN guidance law provides substantial improvements relative to a static ANN guidance law for this application in terms of simplicity, robustness, noise sensitivity, and guidance accuracy. Appendix E provides further details of the combustion-process time-series prediction application discussed in Section 4, drawing analogies to classical Wiener-Volterra filter design, showing how ANN methods provide a

valuable generalization of the classical methods and presenting comparative numerical results for dynamic and static ANNs. The dynamic artificial neural network emerges as clearly superior to the static embodiment, which itself offers compelling benefits vis-a-vis classical solutions.

Section 5 presents conclusions and recommendations, and Section 6 provides the list of references for this report.

2. FUNCTION ESTIMATION PRINCIPLES

2.1 Introduction

To construct an effective neural network for any purpose, particularly including active control, one must make several decisions regarding the fundamental structure of the network and the algorithms that will be used for network generation. To make these decisions properly it is helpful to understand network structure and network training in the broader context of generalized function estimation. This section outlines important principles for both static and dynamic function estimation that underlie all artificial neural networks (ANNs), including the dynamic polynomial neural networks (DPNNs) covered in Section 3.

2.2 Network Structure

An artificial neural network is typically composed of nodal elements that perform a transformation between input and output data vectors. Sets of nodal elements are connected in a specific way to comprise layers; the layers in turn are connected to create the entire network. Figure 2.1 shows the structural heirarchy:

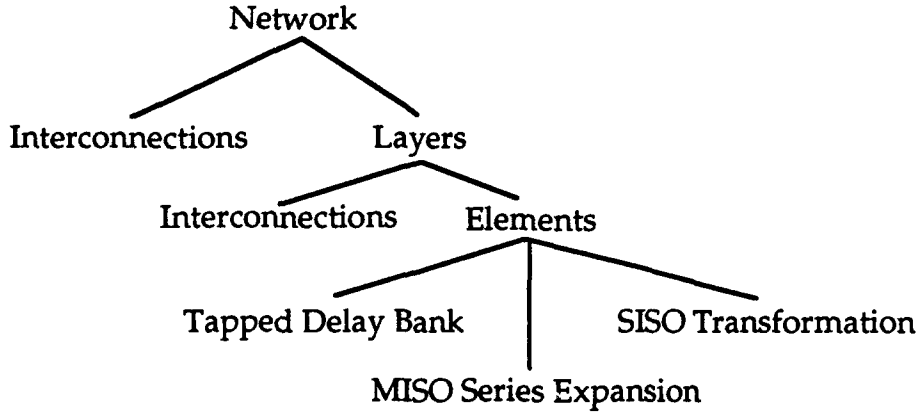


Figure 2.1: Artificial Neural Network Structural Heirarchy

2.2.1 Network Inputs and Outputs

On the highest level, an artificial neural network is a transformation which, when interrogated, produces an output vector, $\underline{s}$, in response to a given input vector, $\underline{x}$. In the case of static networks, the output vector is a single-point transformation of the input data:

$$\underline{s}_i = f(\underline{x}_i, \underline{\theta}) \qquad 2:1$$

where $\underline{\theta}$ is the set of network parameters. Dynamic networks contain internal feedbacks and time delays, and produce a transformation of the form

$$\underline{s}_i = \underline{g}[\underline{x}_i, \ldots, \underline{x}_{i-m}, \underline{s}_{i-1}, \ldots, \underline{s}_{i-n}, \underline{\theta}] \qquad 2:2$$

Neural networks are typically imbedded in systems and are trained to produce a desired effect on the system response. We define the training database as:

$$(\underline{x}_i, \underline{y}_i); \qquad i = 1, 2, 3, \ldots, N \qquad 2:3$$

where N is the number of data vectors in the training database and $\underline{x}_i$ and $\underline{y}_i$ are the *measured* inputs and system responses for the *i*th observation.

Often the network output is written as $\underline{\hat{y}}$ instead of $\underline{s}$; however, this invokes the interpretation that the network output is an estimate of the system response recorded in the training database. For modeling and inverse modeling such is certainly the case, but there are numerous instances in which the network output is not intended to be the best estimate of the database response vector.

In certain control applications, for example, it is not the network output, but a transformation of the network output, that is fitted to the response values recorded in the training database. Fig. 2.2 illustrates a multiple-input, multiple-output (MIMO) network controller. In this figure the network is adapted on-line because the network itself is part of the overall input-output transfer function. The desired network response is the one which, when passed as input into the plant, produces over time the minimum absolute error between plant output and the reference signal.

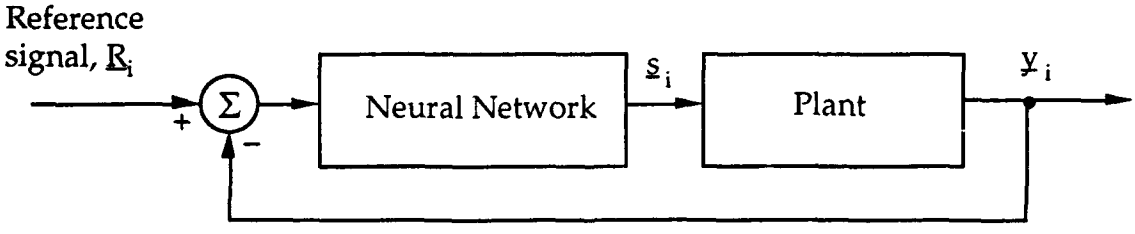
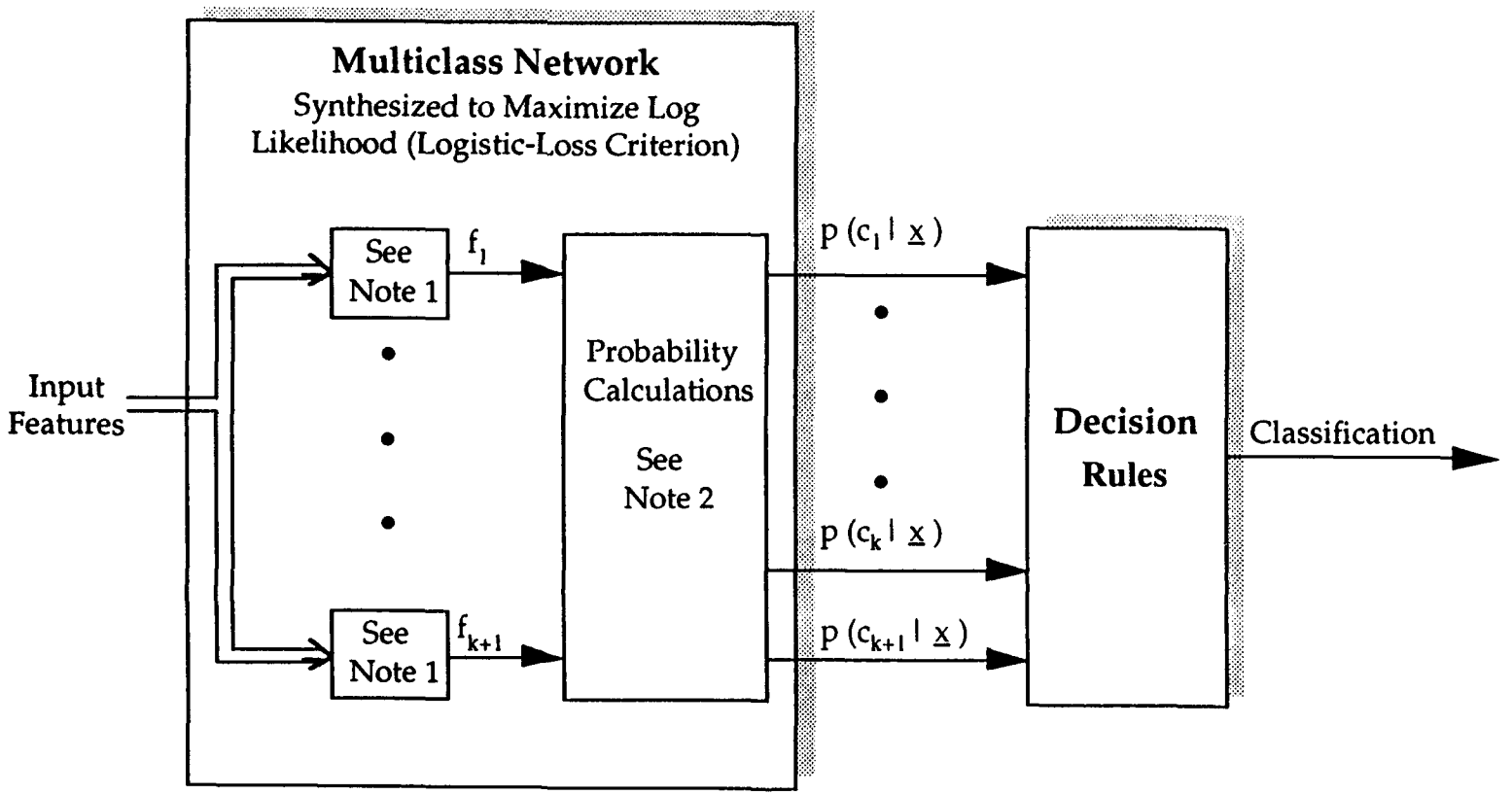


Figure 2.2: MIMO Network Controller

In some network applications, the desired network output is neither the best estimate of the database response values nor a best control signal, but is passed through an additional transformation. In many classification tasks, for instance, the training database response vector, $\underline{y}_i$, is assigned an integer scalar representing the class of the $\underline{x}_i$ vector. The desired network output, however, is a vector of estimated class probabilities (or log-odds) given that the input state is $\underline{x}_i$. This output vector may then be fed into appropriate decision logic to determine the signal classification (Fig. 2.3).



Note 1: Multi-Input, Single Output (MISO Network)

Note 2: The output probabilities are calculated using the formula:

$$p(j) = \frac{e^{f_j}}{1 + \sum_i e^{f_i}} \qquad \begin{array}{l} \text{where } i = 1, \ldots, k \\ \qquad\quad j = 1, \ldots, k+1 \end{array}$$

Figure 2.3: Neural Network Used for Data Classification

2.2.2 Element Definitions

Most artificial neural networks are comprised of fundamental building blocks called nodes, elements, or nodal elements; a generalized nodal element is shown in Fig. 2.4.

The element may be built upon an algebraic or other series expansion, sometimes called the core transformation. This expansion is often composed with a fixed post-transformation function, h(·), that may be linear or nonlinear. In addition, the inputs to a nodal element are often passed through shift registers or delay banks to allow the series expansion to have access to prior input values; this incorporation of memory via shift registers is essential in dynamic networks that deal with time-series data.

The **series expansion** of Fig. 2.4 is of the form

$$\theta_0 + \sum_{j=1}^{J} \theta_j \Phi(\underline{k}_j, \underline{x}) \qquad 2:4$$

where $\underline{\theta}$ is the vector of element coefficients, J is the total number of non-constant terms in the expansion, and $\underline{k}_j$ is a vector of integers. The series expansion within a neural network element has the same form as traditional series expansion techniques; however, with network function estimation, it is desirable that the total number of terms in any given element be kept as small as possible. This point will be elaborated on shortly.

The inclusion of $\underline{k}_j$, sometimes called the set of indices or multi-indices, in Eq. 2:4 allows the series expansion to handle both univariate and multivariate cases. For the multivariate case, each $\Phi(\underline{k}_j, \underline{x})$ is a product of functions of scalars. $\underline{k}_j$ is usually taken to be a vector of integers with each element of $\underline{k}_j$ corresponding to one of the variables in the $\underline{x}$ vector. Using this notation, the jth term in the series expansion may be written as:

$$\Phi(\underline{k}_j, \underline{x}) = \Phi(k_{j1}, x_1) \circ \Phi(k_{j2}, x_2) \circ \ldots \circ \Phi(k_{jD}, x_D) \qquad 2:5$$

where D is the total number of inputs to the series expansion.

The notation introduced above (and thus the nodal element) is sufficiently general to implement a variety of basis functions. Table 2.1 gives examples of how the function $\Phi(k_{jd}, x)$ may be chosen to implement some basis functions commonly used in function estimation (note that in Table 2.1 the subscripts have been dropped from k where the basis function does not depend on them).

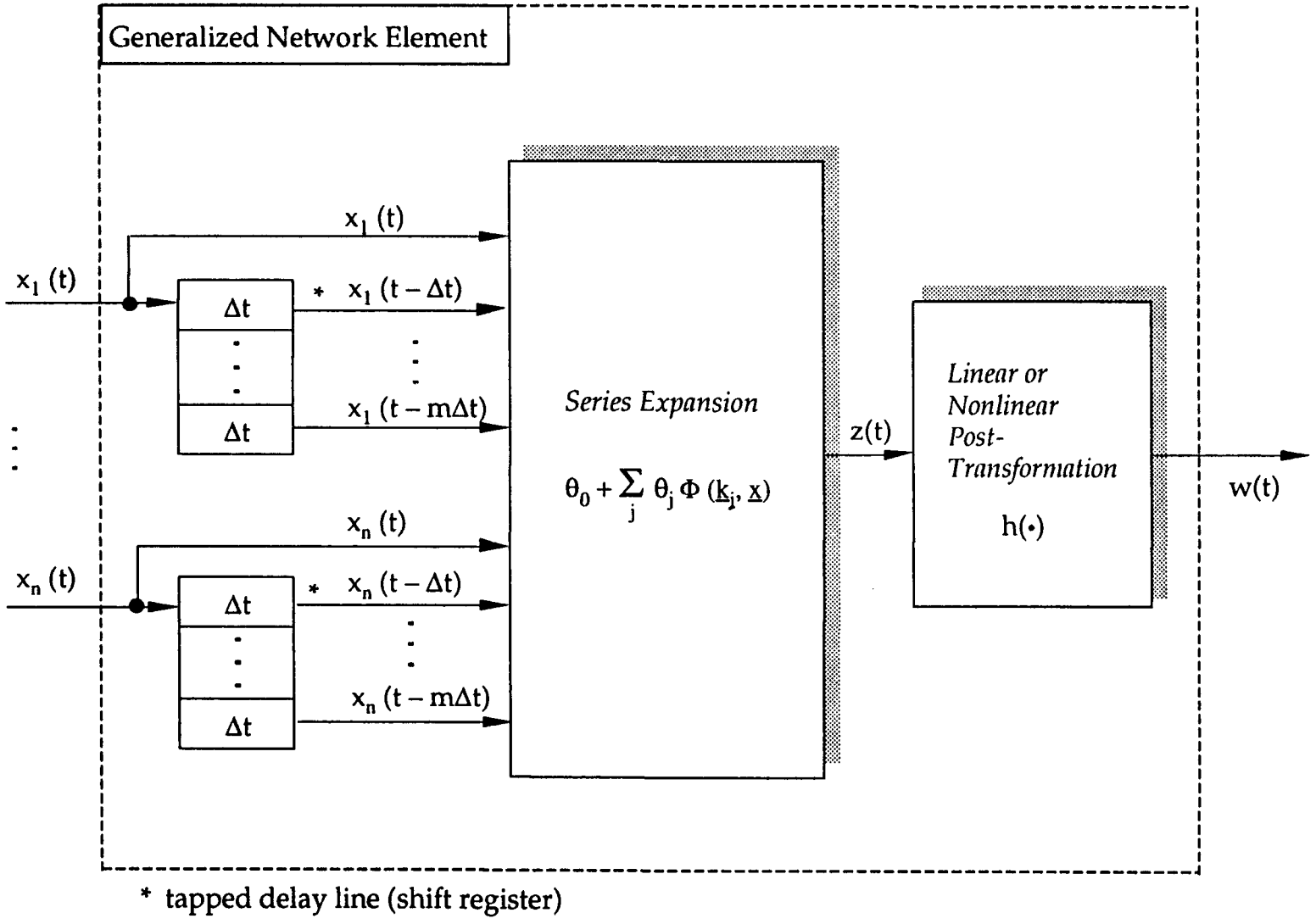


Figure 2.4: Generalized Network Element

Table 2.1: Some Basis Functions Commonly Used for Function Estimation

polynomial	$\Phi(k,x) = x^k$	2:6
spline	$\Phi(k_{jd},x) = \begin{cases} x^k & \text{if } k < r \\ (x-\alpha_{jd})_+^r & \text{if } k \geq r \end{cases}$	2:7
orthonormal wavelet	$\Phi(k_{jd},x) = \begin{cases} 2^{-k/2}\,\Psi(2^{-k}x - \alpha_{jd}) & \text{if } k > 0 \\ 1 & \text{if } k = 0 \end{cases}$	2:8
trigonometric	$\Phi(k,x) = \begin{cases} \sin\left(2\pi \frac{k+1}{2L} x\right) & \text{if } k \text{ is odd} \\ \cos\left(2\pi \frac{k}{2L} x\right) & \text{if } k \text{ is even} \end{cases}$	2:9

For the polynomial basis function (Eq. 2:6), the $\underline{k}_j$ vector is used to determine the powers to which the input variables are raised in the jth term of the expansion. The same is true for the spline basis function (Eq. 2:7); however, the degree of the function is never allowed to exceed r; thus r = 3 results in the commonly used cubic spline.

Note that in both the spline and the wavelet cases an additional set of multi-indices, α_{jd}, must be specified. The parameter α_{jd} in Eq. 2:7 is sometimes called the "knot" and is the value about which the approximation takes place. In some cases, such as uniformly spaced knots, the knot set can be obtained automatically, eliminating the need to pre-specify the additional set of multi-indices.

In the orthonormal wavelet basis function, $\Psi(\cdot)$ is termed the "mother wavelet" and must satisfy a number of specific conditions, including that it be continuous, integrate to zero, and be non-zero in a very specific limited range [Daubechies, 1992]. One such function is the Littlewood-Paley basis function:

$$\Psi(x) = \frac{\sin 2\pi x - \sin \pi x}{\pi x} \qquad 2:10$$

In the trigonometric basis function (Eq. 2:9), L represents the fundamental period of the expansion and depends on the sampling rate.

From Eqs 2:4 – 2:9 it can be seen that the core expansion may be fully specified via a univariate basis function (of the form in Table 2.1) and a $J \times D$ matrix $\underline{\underline{K}}$, where each row of $\underline{\underline{K}}$ is the vector of integers $\underline{k}_j$ as defined above. We will illustrate this with two examples:

Example 1: Consider the "Full Double" element of Fig. 2.5. Because this element has no input delays and no post-transformation $h(\cdot)$, it is completely specified by the series expansion of Eq. 2:11:

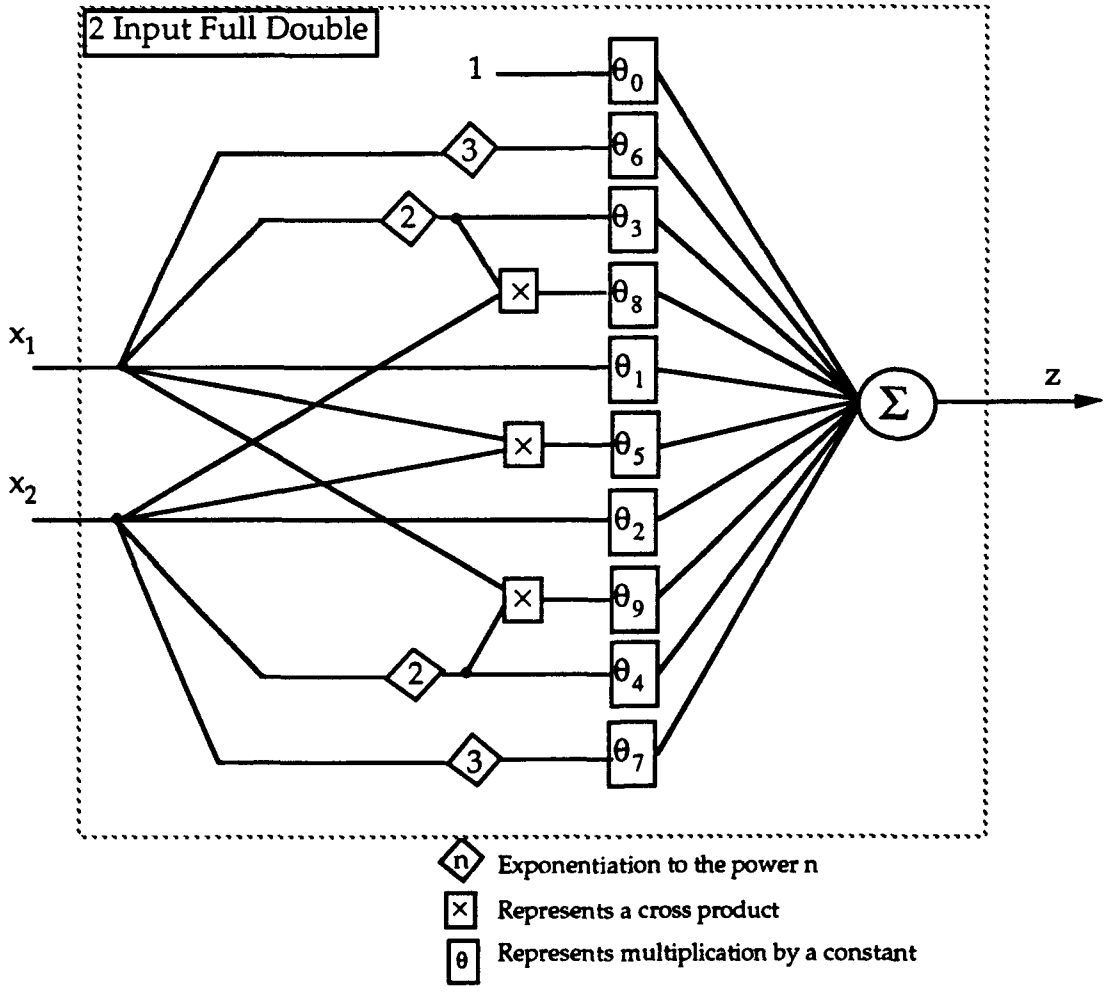


Figure 2.5: Example of a "Full Double" Network Element

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2$$
$$+ \theta_6 x_1^3 + \theta_7 x_2^3 + \theta_8 x_1^2 x_2 + \theta_9 x_1 x_2^2 \qquad 2:11$$

If we choose a polynomial basis function (Eq. 2:6), then the J × D matrix $\underline{\underline{K}}$ corresponding to the expansion in Eq. 2:11 is

$$\underline{\underline{K}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 0 \\ 0 & 2 \\ 1 & 1 \\ 3 & 0 \\ 0 & 3 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} \qquad 2:12$$

Note that because the basis functions of Table 2:1 were defined so that the value of any basis function at k = 0 is unity, the θ_0 coefficient may be removed from Eq. 2:4 if an additional row of zeroes is added to the $\underline{\underline{K}}$ matrix.

Example 2: Consider the trigonometric series expansion

$$\theta_0 + \theta_1 \sin\left(2\pi \frac{3x_1}{2} x\right) + \theta_2 \sin\left(2\pi \frac{5x_1}{2} x\right) \cos\left(2\pi \frac{2x_2}{2} x\right) \qquad 2:13$$

If we choose a trigonometric basis function (Eq. 2:9), then the J × D matrix $\underline{\underline{K}}$ that will yield the series in Eq. 2:13 is

$$\underline{\underline{K}} = \begin{bmatrix} 5 & 0 \\ 10 & 3 \end{bmatrix} \qquad 2:14$$

The first row of this $\underline{\underline{K}}$ matrix contains one non-zero index, 5. Since this index is odd, it corresponds to a sin (·) term with an internal coefficient of (k + 1)/2 (or 3). The second row contains two non-zero indices, 10 and 3, corresponding to cos (·) and sin (·) terms with coefficients of 5 and 2, respectively. Once again, the constant term in the expansion could have been represented implicitly by a row of zeroes in the $\underline{\underline{K}}$ matrix rather than explicitly as in Eq. 2:4.

While the generalized nodal element is capable of implementing many commonly used series-expansion basis functions, neural network function

estimation is fundamentally different from traditional series and nonparametric estimation techniques in the following ways:

- Each network element implements only a limited subset of the terms that would make up a complete series expansion; thus element complexity is kept low.

- A high level of connectivitity between network elements allows a set of relatively simple network elements to be combined so that they can implement complex transformations; thus the network connections do a great deal of the "work" involved in the estimation problem.

- As the number of inputs to the function increases, the bound on the mean squared error for network estimation can be shown to be more favorable than that of traditional function estimation techniques [A.R. Barron, 1991].

There are four factors, discussed below, that determine the number of coefficients (i.e., complexity) that will be needed in a given series expansion; by limiting one or more of these factors, the complexity of the nodal element may be kept low.

Maximum Interaction Order: In multivariate function estimation, the interaction order corresponds to the maximum number of different input variables that may appear at the same time in a given term. Thus, in Eq. 2:13 above the interaction order is two because both x_1 and x_2 appear in the last term of the series. High numbers of interactions result in a combinatorial explosion in the number of terms needed for the complete series expansion, so a limit on the total number of interactions is one of the most important restrictions that can be placed on the nodal element series expansion. A cap on the maximum interaction order can be thought of as limiting the total number of non-zero elements in each $\underline{k}_j$ vector.

Maximum Degree: For polynomial basis functions, the degree of a given term corresponds to the sum of the powers of the variables in the term. The degree of the basis function is the maximum of the degrees of its terms. Thus, Eq. 2:12 represents a third-degree series expansion. For any basis function, a limitation on the maximum degree can be thought of as limiting the sum of the elements in each $\underline{k}_j$ vector.

Number of Inputs: Limiting the number of inputs to a series expansion can also significantly reduce the number of terms. While the number of inputs to the network is largely determined by the application and not the analyst, it is possible to limit the number of inputs to individual elements internal to the network.

Expansion Density: Even after the interconnections, degree, and inputs for a given series expansion are limited, one may choose to remove some terms to obtain a sparse or low density expansion. Eq. 2:13 and the corresponding $\underline{\underline{K}}$ matrix in Eq. 2:14 exemplify a sparse expansion. While this series has an interaction order of two,

a maximum degree of 15, and two inputs, there exist other terms which meet the interaction order and degree constraints and yet are not included in the series expansion. Some specific examples of sparse polynomial expansions are given in Section 3.3.1.

Because it is desirable to keep the total number of network coefficients small, more emphasis is placed on determining an appropriate, efficient network structure, and less on problems associated with extremely high dimensional nonlinear optimization. In general, this approach proves to be more efficient in terms of computing resources and also leads to more robust models that do not have an excessive number of internal degrees of freedom.

The linear or nonlinear fixed **post-transformation**, h(·), of Fig. 2.3 allows the element specification to be general enough to encompass most neural network nodal elements currently in use. The transformation may be used to introduce helpful nonlinearities into the network, especially when there are few or no nonlinearities in the core transformation. Additionally, the transformation may be used to "clip" the output of the core transformation, which often improves the stability of the network (in the bounded-input, bounded-output sense). This may be especially important when a polynomial core transformation is evaluated near or outside the boundaries of its training region. Fig. 2.6 shows the role of the post-transformation in the popular sigmoidal element.

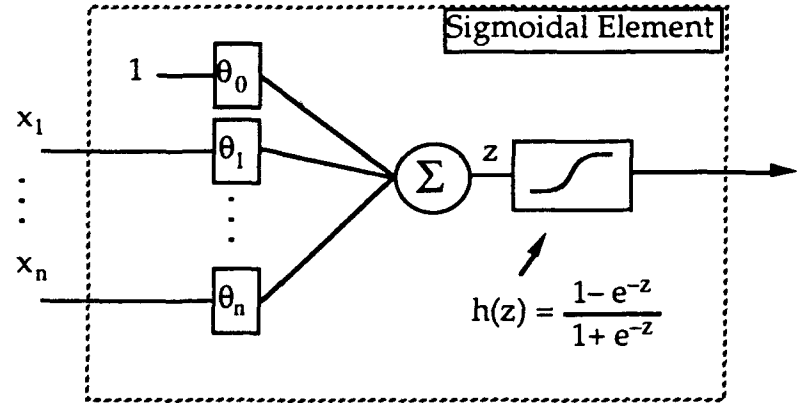


Figure 2.6: Example of a "Sigmoidal" Network Element

The element shown in Fig. 2.6 also has no time delays and implements a series expansion of the form

$$\theta_0 + \sum_{j=1}^{D} \theta_j x_j \qquad 2:15$$

Foliowing the same method outlined above, this series expansion can be represented by choosing the polynomial basis function of Eq. 2:6 and letting $\underline{\underline{K}}$ be a D

× D identity matrix. Because $\underline{\underline{K}}$ contains only first-order interactions and has a maximum power of one, the number of terms in the series expansion is kept low.

The post-transformation, h(·), of Fig. 2.6 is a sigmoidal transformation and has the formula shown in the figure. Due to the nonlinear post-transformation, the sigmoidal nodal element is nonlinear in its parameters.

Another use for the post-transformation, h(·), is to allow the generalized nodal element to implement other types of function approximations that are not simple series expansions. Suppose, for instance, one wants a trigonometric function of the form

$$z = \sin\left(\theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n\right) \qquad 2:16$$

In this case, the series expansion is a first-order polynomial expansion, while the post-transformation is the sin(·) function.

2.2.3 Layer Definition

A layer is a set of elements whose inputs are selected from the same set of candidates. It is important to define a layer as a distinct unit within the network; the reasons for this are:

First, when determining network structure, it is often convenient to build a unit of network structure and then freeze it while building other units of the structure. The network layer is this unit of structure. This is analogous to constructing a building one floor at a time; each subsequent floor is built upon the floors below it, and construction on a new floor cannot begin until a sufficient portion of the lower floors has been constructed.

Second, elements on a given layer are often trained to "work together" as a group to produce the desired network response (Section 2.3.3).

In addition to the internal layers, a network will often contain two special-purpose layers. The first receives inputs, normalizes them, and passes the normalized values to subsequent layers. Often if the inputs are normalized, the network is trained on normalized outputs as well. When this is the case, a second special-purpose layer is required to unitize (or de-normalize) the network outputs. By normalizing and unitizing, each network input is allowed to contribute equally to the solution of the problem, and the magnitudes of the network coefficients become a more accurate reflection of the relative importance of a given term.

2.2.4 Network Interconnections

Fig. 2.7 shows the various types of network interconnections.

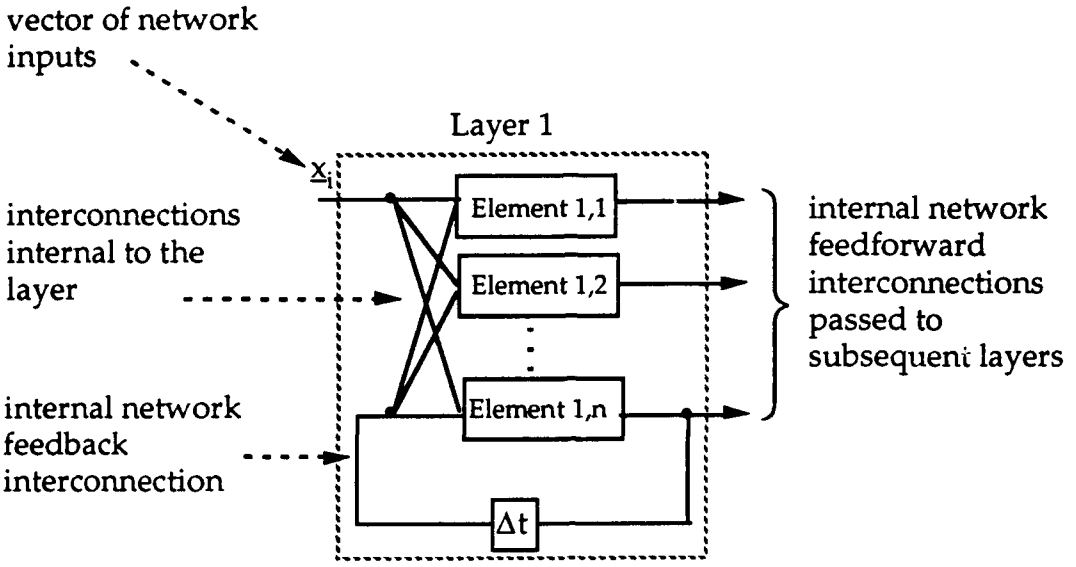


Figure 2.7: Network Interconnections

Intra-layer connections consist of making the inputs to each layer available to every element in the layer. The individual elements are then free to choose which subsets of the available inputs to use. Element outputs are then passed along as layer outputs; the layer outputs may be described by a vector containing scalar values corresponding to the element outputs. Network inputs are available as element inputs at successive layers.

It is important to note that in this paradigm, we do not allow feedback connections internal to the layer or elements. This restriction allows the same layer definition to serve for both feedforward and feedback networks. Connections between layers, however, may be passed forward as inputs to subsequent layers (feedforward networks) or may be passed back as inputs into the given layer and/or previous layers (feedback networks). Thus, for the generalized network structure outlined in Section 2.2, the only differences between dynamic and static networks are the types of inter-layer interconnections allowed.

2.3 Network Training

Often networks are trained using a gradient-based search technique to find the coefficients of a pre-structured network; the popular backpropagation algorithm [Rumelhart, Hinton, Williams, 1986] is an example of this type of training, where the specific optimization algorithm is a form of least mean squares (LMS). The recommended approach, however, is to allow for structural variations by including in the training algorithm(s) methods for determining a network structure suitable for the task at hand. Thus, building the network structure and optimizing

coefficients are intertwined processes used to create more robust networks with less training effort and time.

2.3.1 The Loss Function

For a given network structure the optimal coefficients are those which minimize the sum of a loss function evaluated at every observation in a training database:

$$\min \left(\sum_{i=0}^{N} d(\underline{y}_i, \underline{s}_i) \right) \qquad 2:17$$

where:

N is the number of observations in the training database

$\underline{y}_i$ is the *i*th output vector in the training database

$\underline{s}_i$ is the *i*th output vector of the network; $\underline{s}_i = f(\underline{x}_i, \underline{\theta})$ for feedforward networks (Eq 2:1)

$d(\cdot)$ is the loss or *distortion* function.

Because the goal of the network training algorithm is to minimize the output error as quantified by the loss function, it is helpful if the loss function is a convex, twice-differentiable function with respect to the coordinates of $\underline{s}_i$. By imposing these constraints on the loss function, one guarantees that, if the function being fitted is linear in its parameters, the fitting algorithm will be able to find the set of coefficients that globally minimizes the network error. Even if the network function is nonlinear in the parameters, a convex, twice-differentiable loss function will still result in the best performance possible for the optimization algorithm.

Depending on the nature of the application, a variety of loss functions may be used effectively. The **squared-error** loss function is by far the most commonly used and can be expressed as

$$d(\underline{y}_i, \underline{s}_i) = |\underline{y}_i - \underline{s}_i|^2 \qquad 2:18$$

In this case, the vector norm $|\cdot|^2$ is defined as the sum of the squares of the differences between the coordinates of $\underline{y}_i$ and $\underline{s}_i$. This loss function is most suited to create networks whose outputs estimate the data in the training database as closely as possible.

One problem with the squared-error loss function is that data outliers tend to have a greater than desirable impact on the coefficient optimization. A number

of **robust** loss functions have been suggested which reduce or nullify the effect of outlying data. One such function is Huber's loss function

$$d(\underline{y}_i,\underline{s}_i) = \begin{cases} |\underline{y}_i - \underline{s}_i|^2 & \text{if } |\underline{y}_i - \underline{s}_i|^2 \le A \\ 2A|\underline{y}_i - \underline{s}_i| - A^2 & \text{if } |\underline{y}_i - \underline{s}_i|^2 \ge A \end{cases} \qquad 2:19$$

where A is the distance at which outliers begin to have less effect. When $|\underline{y}_i - \underline{s}_i| > A$, $d(\cdot)$ becomes a 1-norm. Thus, this loss function has the advantages of a 1-norm; however, by using a 2-norm near the origin, the function is everywhere continuous in the first and second derivatives, which is not the case with a 1-norm loss function. Note that in Eq. 2:19 it may be desirable to shape each coordinate of the output norms differently by using an N-dimensional vector of values for A.

Optimization of Eq. 2:17 for the squared-error distortion function of Eq. 2:18 corresponds to the maximum likelihood rule in the case of a Gaussian probability model for the distribution of the errors. However, for multi-class classification problems with categorical output variables, a multinomial probability model in regular exponential form is more suitable than the Gaussian model. In this case, the network functions should be used to model the log-odds associated with the conditional probability of each class given the observed inputs. In this setting, the maximum likelihood rule corresponds to the choice of the **logistic** loss function,

$$d(\underline{y}_i,\underline{s}_i) = -\underline{y}_i \circ \underline{s}_i + \log\left(\sum_{k=1}^{C} e^{s_{i,k}}\right) \qquad 2:20$$

where C is the number of outputs (or classes); $s_{i,k}$ is the kth element of the $\underline{s}_i$ vector; and $\underline{y}_i$ is a vector with the coordinate of the observed class equal to one, and all other coordinates equal to zero (i.e., the observed conditional probabilities given $\underline{x}_i$). In this context, the likelihood associated with observation i is

$$p(\underline{y}_i|\underline{x}_i) = \frac{e^{\underline{y}_i \circ \underline{s}_i}}{\sum_{k=1}^{C} e^{s_{i,k}}} \qquad 2:21$$

and Eq. 2:20 expresses the minus log-likelihood $d(\cdot) = -\log p(\underline{y}_i|\underline{x}_i)$. In this way, it is possible to compute estimates of the probability that an observation is a member of class k', given that the input state is $\underline{x}_i$:

$$p(k'|\underline{x}_i) = \frac{e^{s_{i,k'}}}{\sum_{k=1}^{C} e^{s_{i,k}}} \qquad 2:22$$

Likelihood-based loss functions, such as the logistic loss function described above, can also be helpful for density estimation and clustering of input data. For instance, the loss function may take the form

$$d(s_i) = -\log s_i \qquad 2:23$$

where $s_i = f(\underline{x}_i, \underline{\theta})$ and $f(\cdot)$ is the estimated probability density function. In that case, the network output would need to satisfy

$$\int f(\underline{x}, \underline{\theta})dx = 1 \qquad 2:24$$

and

$$f(\underline{x}, \underline{\theta}) \geq 0 \qquad 2:25$$

If the network function output, $f(\underline{x}, \underline{\theta})$, does not satisfy the integrability requirement of Eq. 2:24, this condition can be reflected in the choice of the loss function by setting it equal to

$$-\log s_i + \log \int f(\underline{x}, \underline{\theta})dx \qquad 2:26$$

where the second term plays the role of normalizing the network output.

If the network does not satisfy the positivity requirement of 2:25, one can use the network function to model the log-density, and take the density function to be

$$p(\underline{x}_i, \underline{\theta}) = \frac{e^{f(\underline{x}_i, \underline{\theta})}}{\int e^{f(\underline{x}, \underline{\theta})}dx} \qquad 2:27$$

and the minus log-likelihood to be

$$-\log\left(p(\underline{x}_i, \underline{\theta})\right) = -s_i + \log\left(\int e^{f(\underline{x}, \underline{\theta})}d\underline{x}\right) \qquad 2:28$$

where $s_i = f(\underline{x}_i, \underline{\theta})$.

A **roughness penalty** may be added to any of the above loss functions to "smooth" the network output and impart an improved ability to interpolate between unseen data points. The addition of a roughness penalty can also improve network input-output stability, such that small variations in network input produce small variations in network output over the entire range of operating conditions. Any of the following, for example, may be used as a roughness penalty:

- Sum of squares of coefficient magnitudes

- Sum of squares of network gradients with respect to the inputs
- Minus the log of the prior density function of the network parameters

2.3.2 Model Selection Criterion

A.R. Barron [1991] has given general conditions such that the minimum mean integrated squared error for a sigmoidal neural network with one layer of sigmoidal nonlinearities will be bounded by

$$O\left(\frac{1}{n}\right) + O\left(\frac{nd}{N}\log N\right) \qquad 2:29$$

where O( ) represents "order of ( )," n is the number of elements, d is the dimensionality (number of coefficients per node), and N is the sample size (number of training exemplars). The first term in Eq. 2:29 bounds the approximation error, which decreases as the network size increases. The second term in Eq. 2:29 bounds the estimation error, which represents the error that will be encountered on unseen data due to overfitting of the training database; it is caused by the error in estimating the coefficients. Estimation error, unlike approximation error, increases as network size increases.

Pre-structured networks, because they often have excessive internal degrees of freedom, are prone to overfit training data, resulting in poor performance on unseen data. Additionally, because of a large number of network coefficients, optimization of pre-structured networks tends to be a slow and computationally intensive process. Without algorithms that learn the structure, the analyst often must resort to guesswork or trial and error if the network complexity is to be reduced.

Improvements in network performance on unseen data can be made if one incorporates into the optimization algorithm modeling criteria that allow the network structure to grow to a just-sufficient level of complexity. While this technique requires additional effort to search for an optimal structure, the overall network generation time is, in general, greatly reduced due to the reduction in the number of coefficients.

Two decades of research have gone into this topic. In Ukraine, Ivakhnenko [1968] introduced the *Group Method of Data Handling* (GMDH). With GMDH, the loss function is squared-error, and overfitting is kept under control by means of cross-validation testing that employs independent subsets (groups) of the data base for fitting and selection. GMDH is a satisfactory approach when ample data are available. Usually, however, the quantity and variety of the data are limited by operational considerations, and it is desirable to use all of the data in the fitting process. In Japan, Akaike [1972] introduced an information theoretic criterion that

uses all of the data and incorporates a penalty term for overfit control. Akaike's criterion is one of several that take the form

$$\frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i, \underline{s}_i) + C\frac{K}{N} \qquad 2:30$$

where K in this context is the number of non-zero coefficients in the model, N is the number of data vectors in the data base, and C is a constant. Since the second term does not depend on the network coefficient values, model selection criteria of the form shown in Eq. 2:30 are often optimized one term at a time.

Akaike's *information* criterion (AIC) and later criteria introduced by Schwarz [1977] and Rissanen [1983] require the loss functions to take the form of a minus log-likelihood. When the loss function takes this form, the AIC is given by Eq. 2:30 with C = 1, and the simplest forms of Schwarz's BIC and Rissanen's minimum description length (MDL) criteria are given by Eq. 2:30 with $C = \frac{1}{2}\log N$. Note that these criteria are applicable to both the squared-error loss for curve estimation with a Gaussian error model, and the logistic loss for probability estimation.

The AIC, BIC, and MDL criteria depend explicitly on an assumed probability model to yield the likelihood expressions. However, other criteria of the form of Eq. 2:30 can be justified by the principle of **predicted squared error** (PSE) [A.R. Barron 1984, Mallows, 1972] or the principle of complexity regularization [A.R. Barron, 1990].

To use the AIC or MDL criteria in the squared-error case, the loss function is recast in the form of a minus log-likelihood for a Gaussian model which may be written for the single-input case as

$$d(y_i,s_i) = \frac{|\, y_i - s_i \,|^2}{2\sigma^2} + \frac{1}{2}\log 2\pi\sigma^2 \qquad 2:31$$

where σ^2 is a constant that may be regarded as the variance of the error in the Gaussian model.[1] The constant σ^2 could be replaced with its maximum likelihood estimate

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} | y_i - s_i |^2 \qquad 2:32$$

which leads to a criterion of the form

$$\frac{1}{2} \log (\hat{\sigma}^2) + C \frac{K}{N} \qquad 2:33$$

with C = 1 or $C = \frac{1}{2} \log N$ for the AIC and MDL, respectively. A different $\hat{\sigma}^2$ is obtained for each candidate network model. However, choices for $\hat{\sigma}^2$ which depend on the candidate model have a serious drawback: "minimization" can occur when K is sufficiently large that the distortion function equals zero, which we know to be a specious result (as overfitting is then extremely likely). Furthermore, these criteria depend explicitly on the assumed family of the error distribution.

As an alternative, it is better to use a prior estimate, σ_p^2, of the model error variance that does not depend on the candidate models. A.R. Barron showed [1986] that even when a prior estimate σ_p^2 is not extremely accurate, the criterion can still prove useful.[2] By inserting Eq. 2:31 into the criterion of Eq. 2:30, multiplying by $2\sigma_p^2$

---

[1] Eq. 2:31 may be extended for multiple outputs as follows:

$$d(\underline{y}_i, \underline{s}_i) = \sum_{j=1}^{J} \frac{| \underline{y}_{ij} - \underline{s}_{ij} |^2}{2\sigma_j^2} + \frac{1}{2} \log 2\pi\sigma_j^2$$

where σ_j^2 is a constant that may be regarded as the variance of the error of output j.

[2] If no value of σ_p^2 is known *a priori*, one can use, for instance, the nearest-neighbor estimate of the function to obtain a good value for σ_p^2. The nearest-neighbor approximation consists of assuming that the output for a given data vector is to be estimated using the output value of the data vector which is closest to it in the data space; σ_p^2 may then be set to be the variance of these estimates. After modeling, the predicted error of the model can be checked to verify that it is less than or equal to σ_p.

$2\sigma_p^2$ and ignoring a constant, it can be seen that minimizing Eq. 2:30 is the same as minimizing

$$\frac{1}{N}\sum_{i=1}^{N} | y_i - s_i |^2 + 2\sigma_p^2 C\frac{K}{N} \qquad 2:34$$

For C = 1, this is the PSE criterion. A.R. Barron [1984] shows that this criterion, unlike the general AIC, is appropriate even when the error distributions are non-Gaussian. For $C = \frac{1}{2}\log N$, the terms in Eq. 2:32 become the leading terms of the complexity regularization criterion derived by A.R. Barron [1990].

For the classification or conditional-probability estimation problem, one may use the AIC or MDL criterion of Eq. 2:30 with the logistic-loss function. Since it already has been shown that the logistic-loss function takes the form of a minus log-likelihood, no modification to Eq. 2:30 is required, and the task becomes one of minimizing

$$\frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i,\underline{s}_i) + C\frac{K}{N} \qquad 2:35$$

where the distortion function, d(·), in Eq. 2:33, is the logistic-loss function of Eq. 2:20.

By minimizing Eqs. 2:32 and 2:33 instead of their predecessors Eqs. 2:18 and 2:20, the network complexity can be appropriately penalized so that the it does not overfit. One may follow the same steps to modify a variety of objective functions.

The CK/N term in the model selection criterion is called the **complexity penalty** and can be thought of as an additional term added to the loss function. The complexity penalty allows the loss function to account for both estimation error and approximation error. By adding the *roughness penalty* to the loss function

Loss = *distortion function* + *complexity penalty* + *roughness penalty* 2:36

one has all that is needed to create a robust objective function that not only takes into account estimation and approximation error, but also function smoothness and stability.

2.3.3 Optimization Strategy

Having defined the structural building blocks for a generic artificial neural network and an appropriate objective function, we next turn to consideration of an efficient search strategy that will find the network structure and optimize the coefficients of that structure.

The optimization strategy proposed here is distinctive in two ways. First, only small subsets of network coefficients are optimized at a given time, thus reducing the dimensionality of the search space and improving the performance of the search algorithm. In most cases, it is sufficient to optimize only the coefficients of a single element while holding all other elements fixed. Ivakhnenko [1968] was the first to propose this type of network construction. In his scheme, the coefficients of each element are optimized in such a way that *each element* attempts to solve the *entire* input-output mapping problem.

While Ivakhnenko's method is powerful, it can be improved upon. A second major distinction of the proposed optimization strategy consists of training the elements on a given layer *so that they work in linear combination with other elements on that layer* to minimize the objective function. This is accomplished using a technique inspired by the projection pursuit algorithm of Friedman et al. [1974, 1981, 1984]. In this strategy, an additional set of "dummy" coefficients, $\beta_1 \ldots \beta_k$, is used; these dummy coefficients multiply the outputs of the n elements on a given layer (Fig. 2.8):

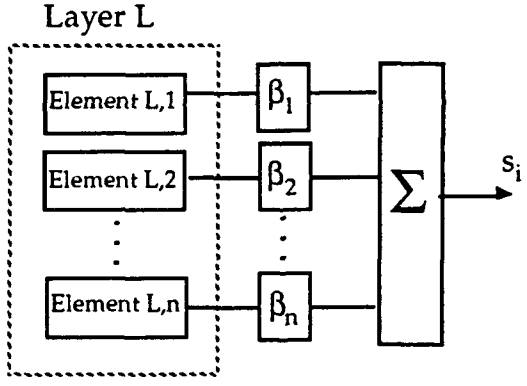


Figure 2.8: Optimization Strategy

The coefficients of the node under consideration, along with the additional dummy coefficients, may be optimized together so that the weighted sum of element outputs minimizes the objective function. This has the effect of training each new element to work well in combination with the existing elements of a given layer. Additional nodes are added to a layer only as long as their additional complexity is justified.

Additionally, coefficients within the new elements may be built up or carved away using an objective function that contains a complexity penalty; the complexity penalty allows only terms which contribute significantly to network performance to survive.

Entire layers may be optimized following a strategy inspired by the backfitting method of Breiman and Friedman [1985]. In this strategy, each coefficient subset is

improved by iterating the search algorithm a few steps while holding the rest of the coefficients fixed. This method is then repeated for another subset of network coefficients, etc. In our general paradigm, the nodal elements become the logical choice for the coefficient subsets to be optimized, and a layer may be optimized by successively recursing through each nodal element, iterating the optimization algorithm a few times for each element. Breiman and Friedman showed that under appropriate conditions this method will yield the same coefficient values as are obtained via a successful global optimization of the same structure. Practical implementation of the backfitting strategy has an advantage in that only a small set of linear equations needs to be solved at any given time.

One example of a way in which backfitting may be applied can be illustrated using a network as defined in Fig. 2:8. Once the structure of the layer has been determined, the coefficients of element L,1 and the dummy coefficients, $\underline{\beta}$, would be adjusted using one iteration of the search routine (see Section 2.3.4). Next the coefficients of element L,2 and the $\underline{\beta}$ are adjusted using one iteration of the search routine. This process continues n times until the coefficients of element L,n have been adjusted. At this point, the process begins again with element L,1. The optimization routine would continue until the optimization no longer improves performance significantly.

Another way backfitting can be used is during the actual search for network structure. Elements may be backfitted each time a new element is added, and the new element can be scored based on its performance in conjunction with the backfitted prior elements. In general, backfitting will increase training time, but it is a technique which can be used as often or seldom as desired. Even in small amounts, backfitting can be a highly efficient way of optimizing larger sets of coefficients so that they work well together.

Once the structure of a given layer is determined, subsequent layers have the choice of combining the layer outputs linearly using the $\underline{\beta}$ coefficients chosen above, or they may go on and recombine the outputs in more complex ways if the improved performance justifies the additional complexity. Layers are added one at a time in this fashion until overall network growth stops. The stopping rule is that the constrained fitting criterion has reached a minimum.

2.3.4 Optimization Method

An iterative least-squares (ILS) method for optimizing the types of nonlinear networks described in this section will now be derived. It will be shown that the ILS algorithm is closely related to other commonly used optimization methods such as the least-squares, Gauss-Newton, and Levenberg-Marquardt searches.

The ILS algorithm consists of finding the local least-squares solution to a linearized version of the network function at each consecutive operating point ($\underline{\theta}$).

Since our network optimization strategy consists of optimizing subsets of the coefficients, in particular those contained in a single network element, the entire network optimization task can be reduced to a series of single-element optimization tasks.

Let $\nabla \underline{f}(\underline{x}_i, \underline{\theta}_0)$ be the gradient of the network output with respect to the element coefficients, $\underline{\theta}$, evaluated at $\underline{\theta}_0$ and abbreviated $\nabla \underline{f}_{\underline{\theta}_0}$. This gradient can then be used to make a local linear approximation of the network function about $\underline{\theta}_0$:

$$\underline{f}(\underline{x}_i, \underline{\theta}) \equiv \underline{f}(\underline{x}_i, \underline{\theta}_0) + (\nabla \underline{f}_{\underline{\theta}_0})^T (\underline{\theta} - \underline{\theta}_0) \qquad 2:37$$

Since the general form of the method is iterative, we wish to find a $\Delta\underline{\theta}$ such that the iteration

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \alpha \Delta\underline{\theta} \qquad 2:38$$

produces a minimum of the loss linearized about $\underline{\theta}_{old}$. Here the parameter α controls the step size and may, for now, be taken to be unity. Taking $\underline{\theta}_0$ as $\underline{\theta}_{old}$, $\underline{\theta}$ as $\underline{\theta}_{new}$, and α to be unity, substitution of Eq. 2:38 into Eq. 2:37 yields

$$\underline{f}(\underline{x}_i, \theta) \equiv \underline{f}(\underline{x}_i, \underline{\theta}_0) + (\nabla \underline{f}_{\underline{\theta}_0})^T (\Delta\underline{\theta}) \qquad 2:39$$

Now, let $\nabla d(\underline{y}_i, \underline{s}_{i,0})$ and $\nabla^2 d(\underline{y}_i, \underline{s}_{i,0})$ be the gradient and Hessian, respectively, of the distortion function evaluated at $\underline{s}_i = \underline{s}_{i,0}$. These are abbreviated $\nabla d_{\underline{s}_0}$ and $\nabla^2 d_{\underline{s}_0}$, respectively. Because restrictions are put on the objective function such that it is convex and everywhere twice-differentiable, the gradient and Hessian are known everywhere and can be used to make a local quadratic approximation of the loss function in the vicinity of the current network output, $\underline{s}_0$:

$$d(\underline{y}_i, \underline{s}_i) \equiv d(\underline{y}_i, \underline{s}_0) + (\nabla d_{\underline{s}_0})^T (\underline{s}_i - \underline{s}_0) + \frac{1}{2}(\underline{s}_i - \underline{s}_0)^T (\nabla^2 d_{\underline{s}_0})(\underline{s}_i - \underline{s}_0) \qquad 2:40$$

Since $\underline{s}_i = \underline{f}(\underline{x}_i, \underline{\theta})$ by definition, Eq. 2:39 may be substituted into Eq 2:40 to yield an approximation to the *i*th component of the objective function in terms of $\Delta\underline{\theta}$:

$$d(\underline{y}_i, \underline{s}_i) \equiv$$

$$d(\underline{y}_i, \underline{s}_0) + (\nabla d_{\underline{s}_0})^T (\nabla_{f_{\underline{\theta}}})^T (\Delta\underline{\theta}) + \frac{1}{2}(\Delta\underline{\theta})^T \left((\nabla_{f_{\underline{\theta}}})(\nabla^2 d_{\underline{s}_0})(\nabla_{f_{\underline{\theta}}})^T \right) (\Delta\underline{\theta}) \qquad 2:41$$

The total empirical loss, J, may then be calculated by summing the approximation of the distortion function over all observations:

$$J(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} d(\underline{y}_i,\underline{s}_0) + \frac{1}{N}\sum_{i=1}^{N} (\nabla d_{\underline{s}_0})^T(\nabla_{f_{\underline{\theta}}})^T(\Delta\underline{\theta})$$

$$+ \frac{1}{N}\sum_{i=1}^{N} (\Delta\underline{\theta})^T\left((\nabla_{f_{\underline{\theta}}})(\nabla^2 d_{\underline{s}_0})(\nabla_{f_{\underline{\theta}}})^T\right)(\Delta\underline{\theta}) \qquad 2:42$$

or

$$J(\underline{\theta}) = J(\underline{\theta}_0) + \underline{b}^T(\Delta\underline{\theta}) + (\Delta\underline{\theta})^T\underline{\underline{A}}\,(\Delta\underline{\theta}) \qquad 2:43$$

where

$$\underline{\underline{A}} = \frac{1}{N}\sum_{i=1}^{N} (\nabla_{f_{\underline{\theta}}})^T(\nabla^2 d_{\underline{s}_0})(\nabla_{f_{\underline{\theta}}}) \qquad 2:44$$

and

$$\underline{b} = \frac{1}{N}\sum_{i=1}^{N} (\nabla d_{\underline{s}_0})^T\nabla_{f_{\underline{\theta}}} \qquad 2:45$$

Eq. 2:43 is minimized over $\Delta\underline{\theta}$ by the choice

$$(\Delta\underline{\theta}) = \underline{\underline{A}}^{-1}\underline{b} \qquad 2:46$$

Thus

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \alpha\,\underline{\underline{A}}^{-1}\underline{b} \qquad 2:47$$

is the desired iteration.

It can be shown that if the α is unity, the nodal element is linear in its coefficients, and the loss function is squared-error, then Eqs. 2:44 – 2:38 will generate the least-squares solution in a single iteration. If α is unity, the nodal element is nonlinear in its coefficients, and the distortion function is squared-error, these equations correspond to the Gauss-Newton optimization method. If desirable, the

step factor, α, may be used to implement a variety of common modifications to the Gauss-Newton technique.

The Levenberg-Marquardt (LM) method of Appendix B is a combination of Gauss-Newton and gradient descent methods. The ILS optimization technique may be modified in a way which corresponds to the LM method by modifying the $\underline{\underline{A}}$ matrix of Eq 2:44 so that

$$\underline{\underline{A}}' = \underline{\underline{A}} + \lambda \, \mathrm{diag}(\underline{\underline{A}}) \qquad 2:48$$

where α and λ are adjusted as described in Appendix B. (Here, the matrix diag ($\underline{\underline{A}}$) denotes the matrix $\underline{\underline{A}}$ with all but its diagonal elements set to zero.)

To make use of the ILS search technique, the analyst must provide the analytic form of the first and second partials of the objective function with respect to the network outputs. Additionally, one needs an analytic form of the partial of the element output with respect to the nodal coefficients $\underline{\theta}$. Recall that each element is the composition of a transformation h(z), with a linearly parameterized expansion (Fig. 2:4). The derivative of the element output with respect to a given coefficient can be given as follows:

$$\frac{\partial h}{\partial \theta_j} = \frac{dh}{dz} \, \Phi_j(\underline{k}_j, \underline{x}) \qquad 2:49$$

The only information necessary to compute the gradient is the first derivative of the post-transformation h(z).

If the network contains feedback, calculating the derivatives becomes slightly more complicated, because the inputs to a given nodal element may in fact depend on prior values of the outputs of the same element. Hence, the inputs are functions of the parameters of the element, and the core transformation is no longer linear in the parameters. In this case we must add an additional chain-rule term to the derivative calculation of 2:51. Thus, for a single input element, Eq. 2:51 becomes

$$\frac{\partial h}{\partial \theta_j} = \frac{dh}{dz} \, \Phi_j(\underline{k}_j, \underline{x}) + \sum_{j=1}^{J} \theta_j \Phi_j(\underline{k}_j, \underline{x}) \sum_{d=1}^{D} \frac{\Phi_j'(\underline{k}_{j,d}, x_d)}{\Phi_j(\underline{k}_{j,d}, x_d)} \frac{\partial x_d}{\partial \theta_j} \qquad 2:50$$

where $\Phi_j'(\underline{k}_{j,d}, x_d)$ is the partial of the j,d term of the series expansion with respect to the input x_d. The only additional information needed to compute the derivative given by Eq. 2:50 is (1) an analytic form of the derivative of the series expansion basis function with respect to the inputs, and (2) the gradient of the inputs with respect to the current parameter subset. The derivatives of the series expansion must be provided by the analyst; the information may already be known since these

derivatives prove useful in computing some forms of the roughness penalty as described in Section 2.3.2.

The derivatives of the inputs are readily calculated because the inputs to the current nodal element are outputs from another element, and we have provided an algorithm for calculating the derivatives for the output of an element.

In some cases, analytic forms of the gradients of the network or the objective function cannot be provided (Fig. 2:2). In these cases, the ILS method cannot be used and a direct search method such as simulated annealing, Powell search, or *GR/GARS* must be used. A description of *GR/GARS* is provided in Appendix B.

2.4 Summary

This section has provided a way of viewing generalized function estimation in a neural network context. The intent is to provide a paradigm that is general to cover many estimation techniques currently in practice, including sigmoidal networks, static and dynamic polynomial neural networks, and many of the estimation techniques popular in the statistics community. What follows is a discussion of dynamic network algorithms that the authors have implemented, with an explanation of how they fit into the overall function estimation paradigm described above. The hope is that the general and comprehensive paradigm in this section will allow the reader to understand the relationships between various techniques and the nature of suggested improvements to network generation algorithms.

3. DYNAMIC POLYNOMIAL NEURAL NETWORKS

A major focus of the work under this contract has been to investigate two specific aspects of the synthesis and use of neural networks for estimation. The first concerns the use of tapped delay lines (i.e., memory) within the network elements, while the second pertains to the use of feedback connections internal to the network. By including time delays and feedback connections, dynamic networks can:

- compute time-varying transformations given static inputs
- perform infinite impulse response (IIR) filtering operations along with finite impulse response (FIR) filtering operations
- provide a phase-shift operator between time-varying inputs
- provide better modeling accuracy with fewer internal degrees of freedom
- handle time-series data more effectively than feedforward networks.

After defining some key terms, this section outlines two basic algorithms for constructing and training dynamic networks. The first, *Dyn3*, builds and fits a multi-input single-output (MISO) single-layer network with feedback of the output. The second, *DynNet*, constructs a multi-input multi-output (MIMO) multi-layer network with a variety of possible internal feedback arrangements. By placing these algorithms into the function-estimation paradigm outlined in the previous section, we will see how they are similar to, and different from, other neural network implementations. Under the scope of this project, two software packages have been created to implement *Dyn3* and *DynNet*. These software packages will be described in this section to the extent needed to elucidate the methodologies behind the synthesis and use of the dynamic networks.

The core transformation of the networks described here uses the polynomial basis function given in Eq. 2:6. This is the same basis function used in sigmoidal networks; however, unlike the elements used in sigmoidal networks, which employ only a first-degree polynomial expansion (i.e., a linear combination of inputs), the nodal elements used in this work may contain a polynomial expansion of any order. Since this higher-degree polynomial expansion introduces nonlinearities into these nodal elements, networks comprised of these elements are alled *polynomial neural networks* (PNNs). (Sigmoidal networks derive their name from the sigmoidal post-transformation, shown in Fig. 2.6, which is used to introduce nonlinearity into the network.)

Because the networks described in this section usually incorporate internal time delays and feedback paths, they will sometimes be referred to as *dynamic polynomial neural networks* (DPNNs).

3.1 Equation-Error and Output-Error Modeling

Recently, two approaches to adaptive IIR filtering have evolved that may also be applied to dynamic neural networks; these are known as the *equation-error* and *output-error* methods. In the equation-error method, the output estimate is found using previous output measurements obtained from the system itself or from the training database. Fig. 3.1 shows the application of an equation-error network to a system identification problem.

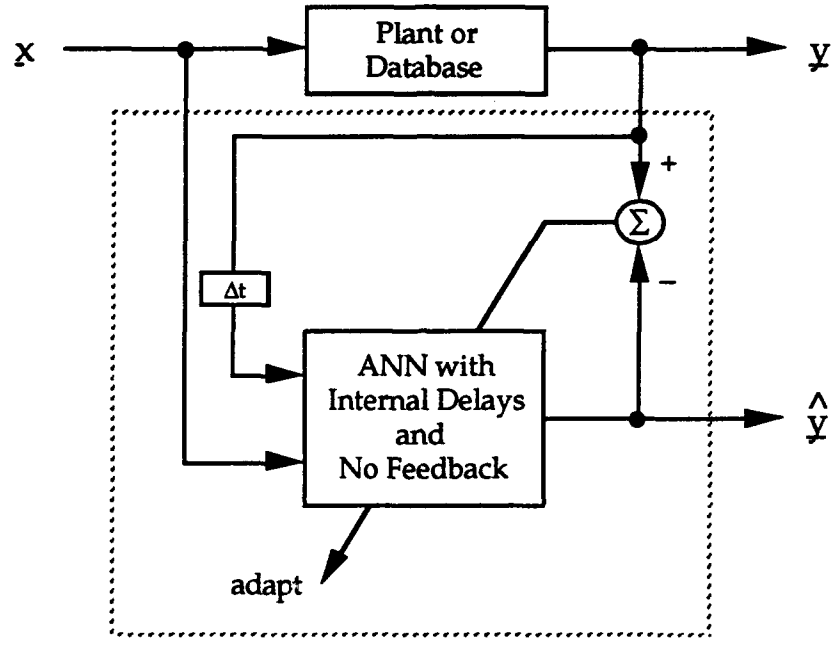


Figure 3.1: Equation-Error System Identification

Because an equation-error network need not contain internal feedbacks, one can treat the equation-error formulation as a feedforward network. As such, the rapid batch least-squares techniques used for the synthesis of static feedforward polynomial neural networks (Appendix A) can be applied to the synthesis of equation-error dynamic networks.

It is not, however, always possible (nor necessarily desirable) to use the equation-error formulation. In an on-line situation, prior measurements of the system output(s) may not always be available for use as inputs to the network. Even if prior measurements are available, it has been shown [Shynk, 1989] that the equation-error formulation can lead to biased coefficient estimates, especially in cases where the measured responses contain additive noise.

The output-error formulation feeds back *previously estimated* output values. Fig. 3.2 shows the application of an output-error network to the same system identification problem that was shown in Fig. 3.1.

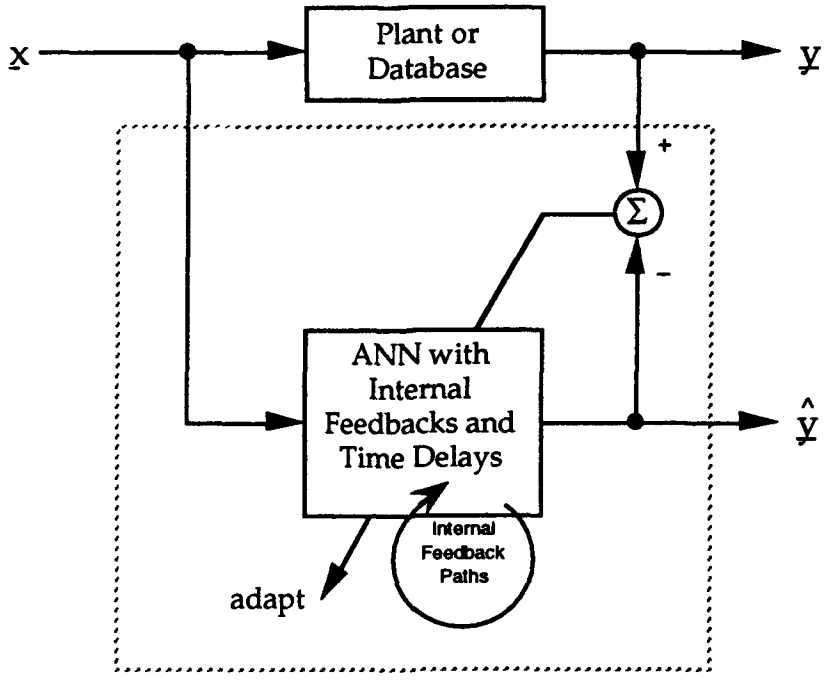


Figure 3.2: Output-Error System Identification

The output-error network has both internal time delays and internal feedback paths, and as such has all of the advantages of dynamic networks that were listed above.

Dyn3 and *DynNet* can synthesize both equation-error and output-error dynamic neural networks. Since internal time delays are the only aspect of equation-error networks that distinguishes them from static feedforward networks, this report will instead concentrate on the creation and optimization of output-error networks. However, it will be seen that even in the synthesis of output-error networks, a number of equation-error networks may be created as intermediate steps.

3.2 Optimization and Initialization

The potentially improved performance of output-error dynamic networks comes at a price. Because of internal feedbacks, the nodal elements of these networks can become nonlinear in their coefficients. Fig. 3.3 shows a simple example of a network containing feedback and time delays. Note that Δt represents a delay of one time step.

In Fig. 3.3, the output, z_A, of nodal element A depends on the input vector $\underline{x}_i$ and the coefficients, $\underline{\theta}_A$, of element A:

$$z_A = f(\underline{x}_i, \underline{\theta}_A) \qquad 3:1$$

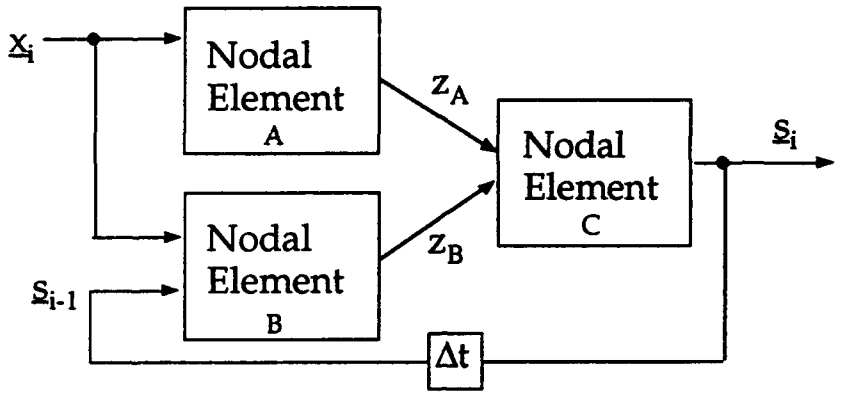


Figure 3.3: Sample Dynamic Network

Thus, if $f(\cdot)$ is linear in its coefficients, the coefficients of A may be optimized using a batch least-squares technique. For nodal element B, however, the story is different:

$$z_B = f(\underline{x}_i, \underline{s}_{i-1}, \underline{\theta}_B) \qquad 3:2$$

But $\underline{s}_{i-1}$ is also some function, $g(\underline{\theta}_B)$, involving $\underline{\theta}_B$; thus

$$z_B = f(\underline{x}_i, g(\underline{\theta}_B), \underline{\theta}_B) \qquad 3:3$$

which is usually not linear in the coefficients $\underline{\theta}_B$. Any nodal element that is involved in a feedback loop is nonlinear in its coefficients, and the batch least-squares techniques that are useful in synthesizing static feedforward networks (Appendix A) cannot be used to fit these nodal elements. Instead, the optimization of dynamic networks and nodal elements generally involves one of the more computationally intensive iterative techniques described in Section 2.3.4 and Appendix B.

Feedback connections introduce another difficulty concerning network optimization. Parameter nonlinearities tend to distort the surface of the objective function and often introduce multiple minima, making it impossible to guarantee global optimization in all cases. Because of these multiple minima, great care must be taken when initializing the parameters for an iterative search; additionally, it may be desirable to use a multi-modal search method that is better suited to find the global minimum of a surface having numerous local minima.

Another cost of dynamic networks is the initialization required, which results from the use of time delays. System input and/or output values must be used to fill all delay banks before the element core transformation can generate an output value. This does not pose a problem if one can afford to wait M observations before generating the network output, where M is the maximum delay value. In other

applications, however, such as tracking and control, output estimates must be generated beginning at the first observation. In these cases, one may need to create a static network to provide the network output or to initialize the values of the delay banks contained within the dynamic network.

3.3 Single Layer MISO Dynamic Network (*Dyn3*)

The *Dyn3* algorithm is next described; note that the *DynNet* algorithm is based upon and subsumes the *Dyn3* algorithm.

3.3.1 Network Structure

In previous work on dynamic networks [Barron, Abbott, 1989; Parker, Cellucci, Abbott, 1991] a distinction was made between a *non-memory feedforward* nodal element and a *memory feedforward* nodal element; the distinction being that the former did not include the optional tapped delay line described as part of the nodal element definition (see Section 2.2). Although in some cases this is a helpful distinction, the *non-memory feedforward* element can be seen as a special case of the *memory feedforward* element where there are no time delays. Additionally, with the exception of some initialization issues which are addressed below, the inclusion of input time delays does not fundamentally change either the nature of the network nor the method for creating the optimal network. Outputs of the tapped delay lines are treated simply as an additional set of candidate inputs to the series expansion.

Removal of the distinction between the two types of feedforward elements does not mean that the decision to build elements with or without tapped delay lines is arbitrary. The delays are often appropriate and helpful when dealing with time-series data. Conversely, the delays are inappropriate when the network will be interrogated using a sequential set of data vectors that have nothing to do with each other.[3] The dynamic network algorithms and their corresponding software implementations described herein can incorporate input delay banks; however, decisions whether to use them and what delay values to use are left to the analyst.

The series expansion, or core transformation, chosen for the dynamic nodal element is the Kolmogorov-Gabor (KG) multinomial [R.L. Barron, 1984], which is an algebraic sum of terms:

---

[3] As an example of a network that should not use time delays, consider a network that estimates the cholesterol level of an individual given data concerning his age, weight, eating habits, etc. It would be inappropriate to retain information concerning a prior individual (i.e., internal time delays) when estimating the cholesterol of the current individual, especially if there were no connection between the two individuals (genetic, same diet, etc.).

$$z = \theta_0 + \sum_i \theta_i x_i + \sum_{ij} \theta_{ij} x_i x_j + \sum_{ijk} \theta_{ijk} x_i x_j x_k + \ldots \qquad 3:4$$

Since it has been shown that the KG multinomial can model any analytic single-valued transformation, it is a good choice. But, in principal, many kinds of building-block elements could be used in modeling by induction (see Section 2.2.2). The attention to algebraic elements derives from the pioneering work in the 1940s of Kolmogorov and, working independently, Gabor. They demonstrated the near universality of multinomials in representing physical processes, including dynamic systems. In fact, recent developments in statistics, information theory, computational methods, and approximation theory suggest that a multinomial description of the network learning process highlights some of the similarities (as well as important differences) among network syntheses and modern statistical inference methods [A. R. Barron, R. L. Barron, 1988].

The number of terms (coefficients) in the complete KG multinomial of Eq. 3:4 is

$$\frac{(R + N_{kg})!}{R!N_{kg}!} \qquad 3:5$$

where R is the rank of the multinomial, which is the maximum interaction-order (i.e., how many cross-couplings are allowed) and the maximum degree (i.e., the sum of the powers of any given term). Eq. 3:4 explicitly shows the first four KG multinomial terms, corresponding to all rank 0, 1, 2, and 3 terms.

N_{kg} in 3:5 is the number of inputs to the multinomial and is determined by

$$N_{kg} = \sum_{i=1}^{N} d_i \qquad 3:6$$

where N is the number of inputs to the network and d_i is the number of time delays associated with the *i*th input value. The number of network inputs is determined largely by the physical nature of the application, and the analyst has only marginal control over this quantity. The analyst has complete control, however, over the number of delays and the order, R, of the KG multinomial; both quantities should be specified with care to avoid a combinatorial explosion in the number of coefficients associated with a given element.

As can be seen from Eqs. 3:5 and 3:6, the number of terms in the complete KG multinomial can become large very quickly for even small values of N, d_i, and R. To assist the analyst in keeping the number of coefficients down, some very specific subsets of the full KG multinomial can be used with great effectiveness. Experience has shown that in many applications the cross terms, $x_i x_j$, are more important than the power terms, x_i^2 or even $x_i^2 x_j$. By eliminating all terms where any given input

is raised to a power greater than one, one can significantly reduce the complexity of the element, yet still retain its power to introduce nonlinearities when such are needed to produce a desired output. This form of reduced KG multinomial is often called a *multilinear multinomial;* a three-input rank-two multilinear expansion is as follows:

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_1 x_2 + \theta_5 x_1 x_3 + \theta_6 x_2 x_3 \qquad 3:7$$

If the analyst wishes a further reduction in the number of terms in the core transformation, another subset of the KG multinomial may be used which eliminates from Eq. 3:4 all terms which involve more than one variable. This series expansion is often called an *additive* polynomial, and a three-input rank-two additive expansion is:

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_1^2 + \theta_5 x_2^2 + \theta_6 x_3^2 \qquad 3:8$$

Any of the above KG multinomial expansions may be reduced to a simple linear combination of terms simply by specifying a rank of one.

The dynamic nodal elements do not make use of a post transformation, h(·), as described in Section 2; however, the addition of a clipping transformation might prove useful in future work. A potentially useful transformation might be:

$$h(z) = \begin{cases} A & z \geq A \\ z & B \leq z \leq A \\ B & z \leq B \end{cases} \qquad 3:9$$

where A and B are chosen such that h(z) is not allowed to exceed the values taken on during network training. This transformation would have no effect on the training algorithm because h(z) is linear in the training region; however, it may improve network stability on unseen data.

Our previous work on dynamic nodal elements made the distinction between *memory-feedback* (MFB) and *memory-feedforward* (MFF) nodal elements [Barron, Abbott, 1989; Parker, Cellucci, Abbott, 1991], as shown in Figs. 3.4 and 3.5. However, allowing the internal feedbacks of the MFB nodal element prevents us from using the general network estimation paradigm that was outlined in the previous section. For this reason, rather than handling the feedback internal to the element structure, it is better always to use a *memory-feedforward* nodal element, which fits into the paradigm of Section 2, and to handle the feedback via *layer* interconnections.

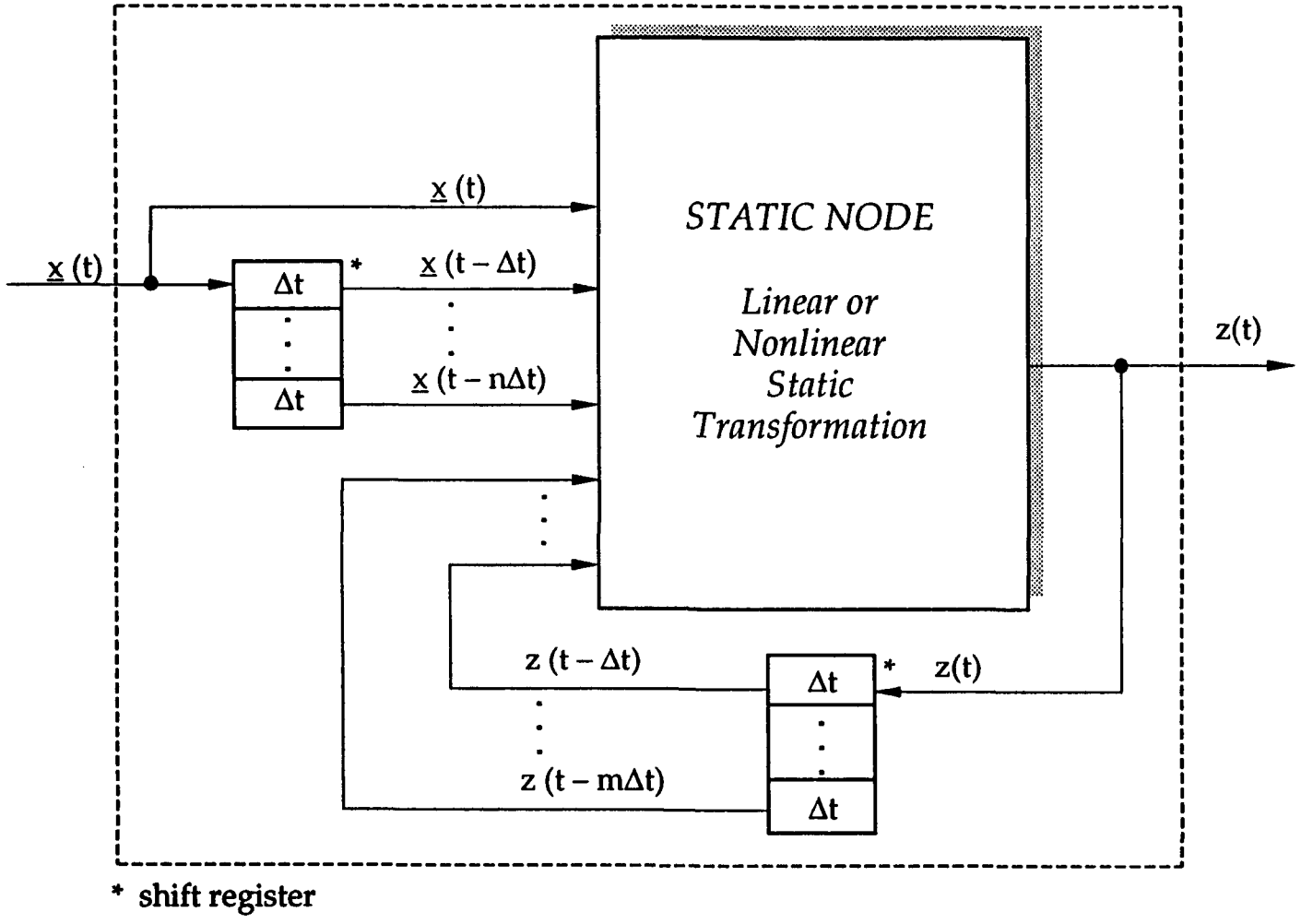


Figure 3.4: Memory-Feedback (MFB) Nodal Element

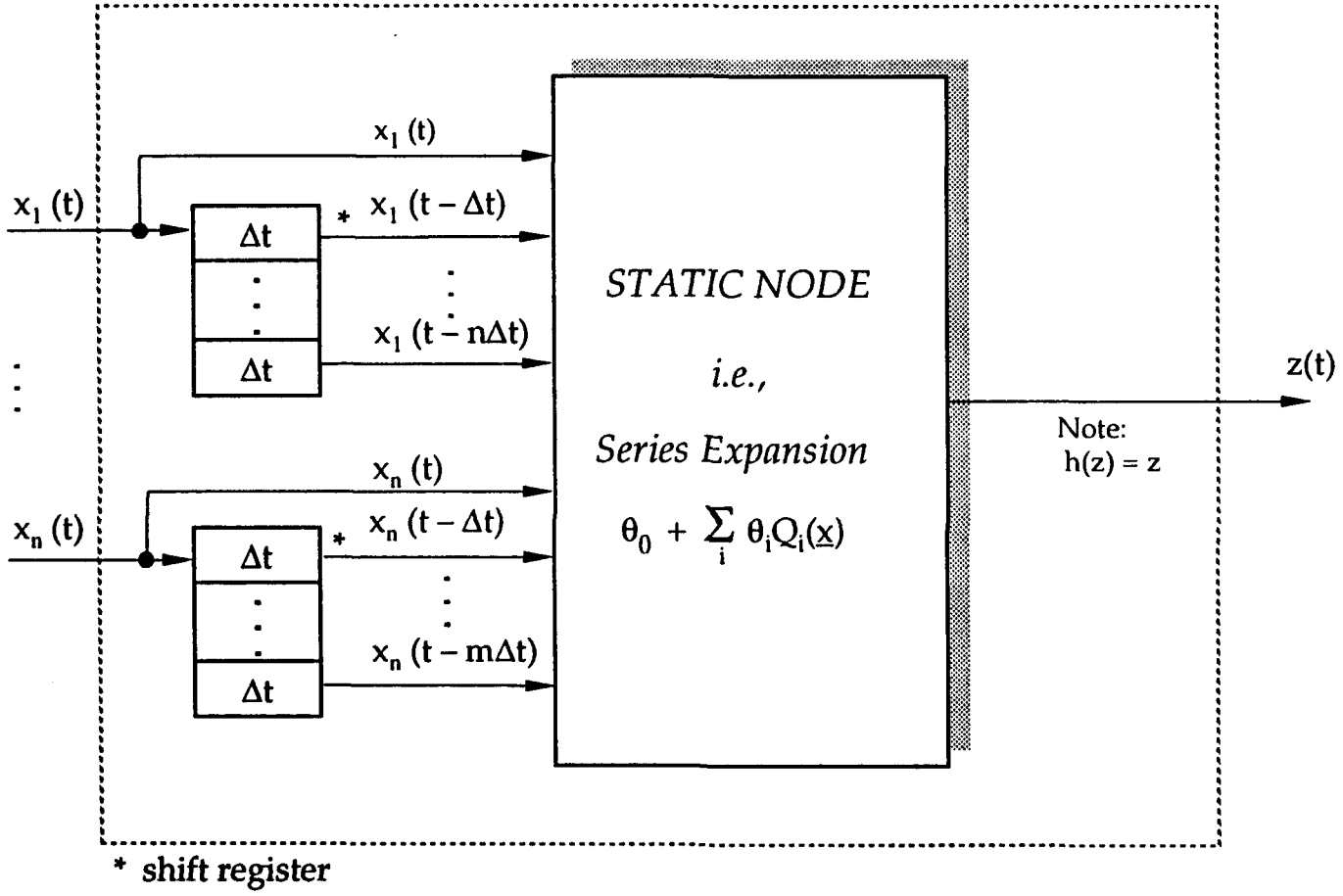


Figure 3.5: Memory-Feedforward (MFF) Nodal Element

Since *Dyn3* is a MISO network, and since the core transformation internal to the nodal element can take on an arbitrary degree of complexity, each *Dyn3* layer is limited to a single element. The output from the *Dyn3* layer may be fed back through a single time delay and made available as an input to the network (Fig. 3.6). This construction results in a network that fits the general function estimation paradigm described in Section 2 and is still functionally equivalent to the MFB nodal element described in previous work.

3.3.2 Network Training

Because the complexity of the *Dyn3* element is not limited (its maximum complexity is set by the analyst), determining the structure of a *Dyn3* network entails using the PSE criterion (Eq. 2:31) to build or carve terms from the arbitrarily complex single element. The "building" algorithm is as follows:

1. Fit all single terms (i.e., parameters) one at a time, fixing all other parameters to zero. Keep the one that yields the best score as measured by the objective function.

2. Now fit all combinations of two terms in which one of the terms is the element found in Step 1. Keep the one with the "best" score.

3. Since the objective function includes a complexity penalty, if the score in Step 2 is not less than the score in Step 1, the additional complexity did not yield a corresponding improvement in the network performance; therefore stop the building. If, on the other hand, the score in Step 2 is better than that of Step 1, the performance is improving and terms should continue to be added, one at a time, until the score ceases to improve.

This method of building is called a *greedy* build, because once a term is found that improves the score, it is kept. Another method of building consists of trying all possible pairs of terms in Step 2; however, its increased computing time, in general, does not pay off, so it is not included as an option with *Dyn3* or *DynNet*.

The carving algorithm is similar to the building algorithm:

1. Fit the entire element, consisting of D non-zero terms.

2. Now, by removing one term at a time, fit all combinations of D–1 terms and keep the combination with the "best" score.

3. Since the objective function includes a complexity penalty, if the score in Step 2 is not less than the score in Step 1, the complexity of the element in Step 1 is just sufficient, and no further terms should be "carved" away. If, on the other hand, the score in Step 2 is better than that of Step 1, the

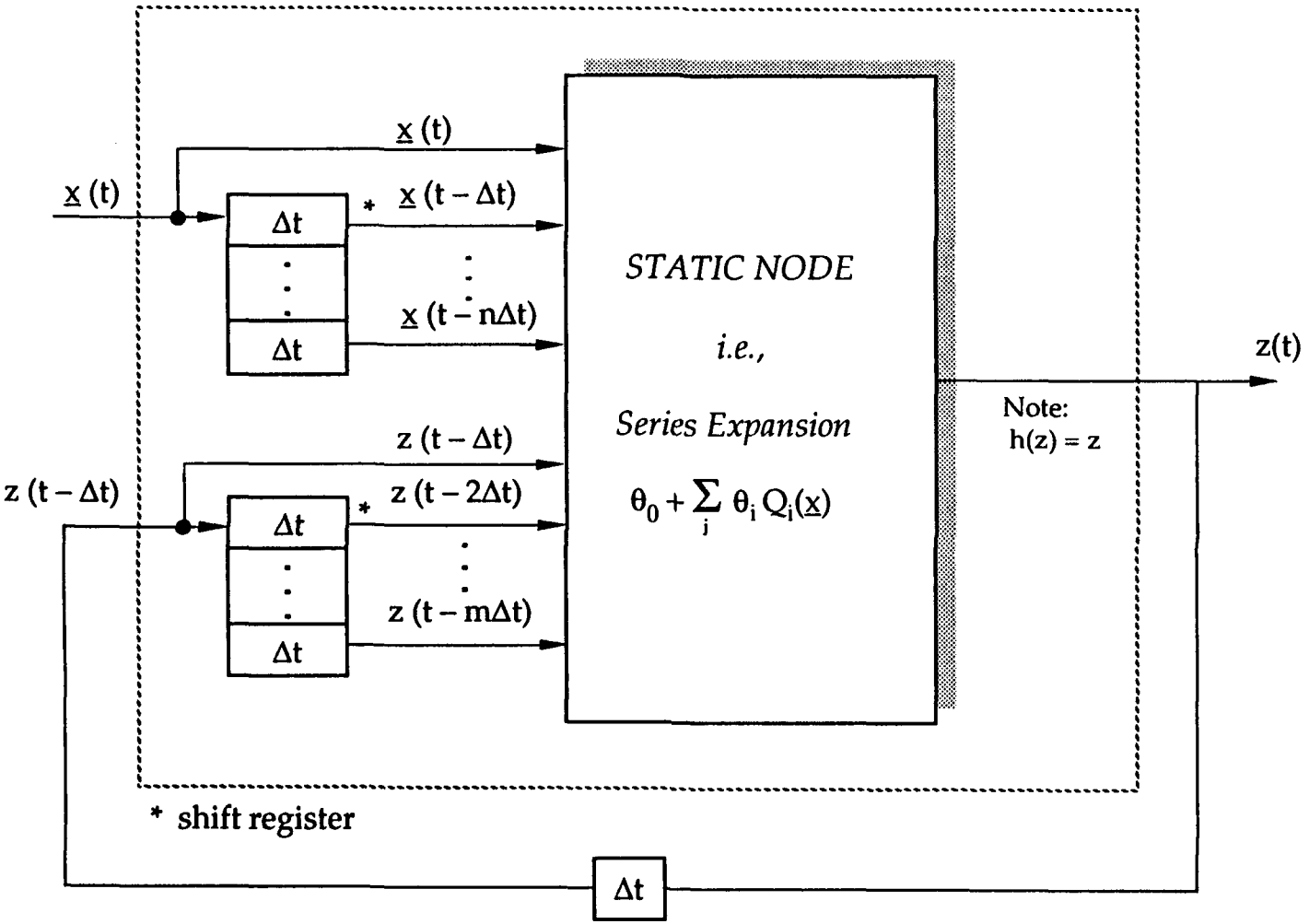


Figure 3.6: *Dyn3* **Layer and Interconnections (MFB)**

performance is improving and terms should continue to be carved, one at a time, until the score ceases to improve.

This method of carving is called a *greedy* carve since once a term is carved away it cannot be used again. Another method of carving consists of carving all possible combinations of terms in Step 2; however, the additional computing time required by a non-greedy method of carving is not warranted. *It should be noted that neither building nor carving would be possible without an objective function that appropriately penalizes network complexity.*

Building or carving of terms also corresponds to choosing which delay values are important, because the coefficients that correspond to unimportant delay values will not survive the process. Thus, in the same manner, both the element weights and the appropriate time delays are chosen automatically. This method of choosing the time delays is the most reliable, but it is also the most computationally intensive, because there are several terms associated with each delay value.

One alternative means of structure selection would be to build or carve in "chunks." In this method, all the coefficients that correspond to the output of a particular time delay would be added or removed simultaneously. Once the best structure is determined, building or carving could continue in the normal fashion if desired. This type of building and carving was not implemented in *Dyn3*.

Since the *Dyn3* layer structure is fixed and only one feedback is possible, analytic forms of the network gradient and Hessian with respect to the element coefficients can be specified. Additionally, since the loss function used is squared-error, it is possible to specify the first and second partial derivatives of the objective function with respect to network output. Since the gradient information is available, the gradient-based Levenberg-Marquardt iterative search can be used to fit the coefficients of a *Dyn3* network. (See Appendix B for details of the Levenberg-Marquardt algorithm.)

As mentioned above, with a multi-modal search surface, great care must be taken in initializing the network coefficients, especially if a unimodal gradient-based search technique is used to fit the coefficients of the network. *Dyn3* allows for two options concerning the initialization of the coefficients: zero initialization and least-squares initialization. In general, it has been found that initializing all coefficients to zero provides the best results. Since a network of all zero coefficients is stable (albeit trivial), any further optimization done from that point will tend also to result in a stable network. Terms that ensure stable solutions may also be added to the objective (i.e., loss) function as appropriate.

Occasionally it is helpful to initialize the search using the coefficients found when the network is set up in *equation-error* configuration, so that the batch least-squares method can be used to optimize the coefficients. Once the feedback is connected, however, stability is no longer guaranteed. If the least-squares solution

with feedback is unstable, it is possible but difficult for the optimization technique to bring the network back into a region of stability.

3.4 Multi-Layer MIMO Dynamic Networks (*DynNet*)

3.4.1 Network Structure

DynNet uses the same basic nodal elements that are used by *Dyn3*. As with *Dyn3*, the complexity of the core transformation (the maximum number of cross-couplings allowed) is specified in advance by the analyst. However, *DynNet* differs from *Dyn3* in its use of nodal elements.

While *DynNet* can create an arbitrarily complex single-element *Dyn3* network, in *DynNet* the overriding philosophy is to use larger numbers of simpler elements. Element complexity (i.e., the number of coefficients) is limited in the following ways:

1. Element core transformations (KG multinomials) take on a maximum allowable rank as specified by the analyst (*same as Dyn3*).

2. Terms in the complete KG multinomials may be eliminated by using either the *multilinear* or *additive* reduced form of the multinomial (*same as Dyn3*).

3. If an element has a core transformation of rank greater than one, the maximum number of inputs to that element is limited to three. Or, to restate the constraint another way, an element may make use of more than three inputs only if its core transformation is a simple linear combination of the inputs (unique to *DynNet*).

It is the third constraint in the above list that allows the elements to be simplified and forces the network interconnections to perform a large share of the estimation task. A nodal element with a KG multinomial of rank one and unlimited numbers of inputs is called a linear (affine) element and is shown in Fig. 3.7. If the analyst so desires, *DynNet* will make linear elements available for use by the network construction routines in addition to elements of the specific type and complexity that were specified by the analyst.

DynNet layers can be made up of any number of elements. Following the method described by Ivakhnenko [1969], each *DynNet* element is trained to solve the entire input-output mapping problem as best it can; therefore, each *DynNet* layer always begins with at least one element responsible for fitting one of the network outputs. The layer size may later be reduced, however, if subsequent layers choose not to make use of some of the outputs from a prior layer.

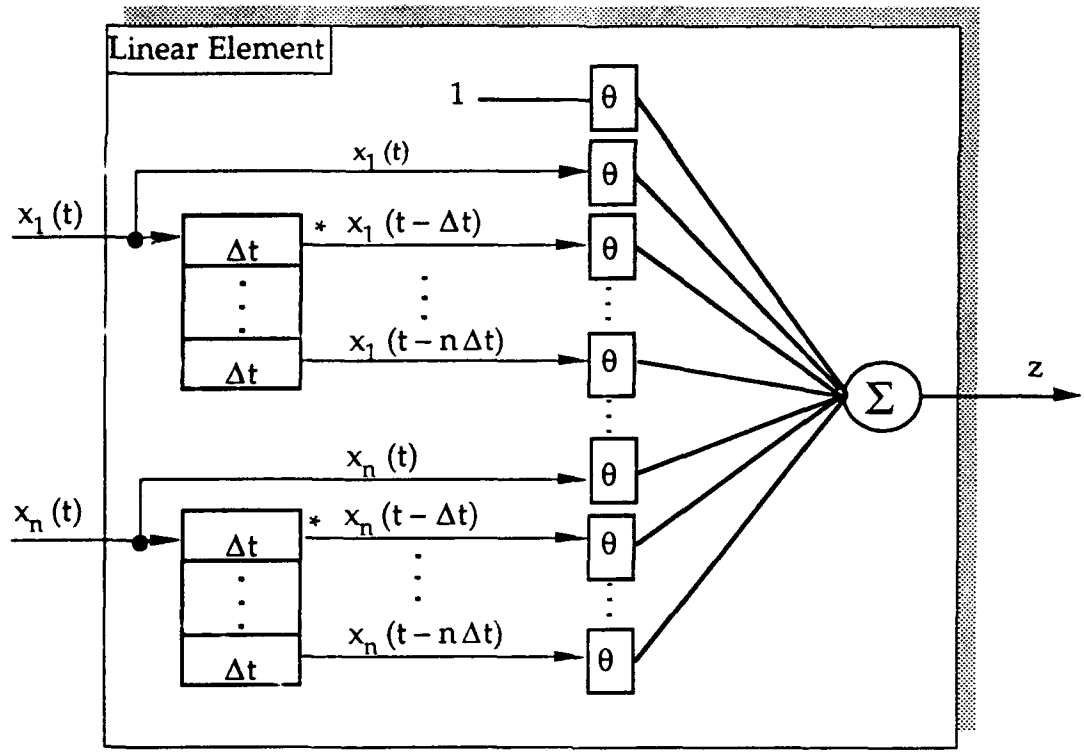


Figure 3.7: Linear (Affine) Element

Because of the large number of potential feedbacks and the computational difficulty involved in fitting a network with feedbacks, *DynNet* limits the feedback connections to either *intra-layer* feedback or *output* feedback, as shown in Figs. 3.8 and 3.9.

In a network with *intra-layer* feedback, layer L is independent of all the subsequent layers; thus layer L can be determined before the rest of the network structure has been finished. In the *output* feedback case, however, layer L is dependent on all previous and subsequent layers, and the coefficients of layer L may need to be refitted as new layers are added.

3.4.2 Network Training

DynNet incorporates a variety of heuristics to search for a network structure of just-sufficient complexity. Key to the *DynNet* algorithm is a method whereby a large variety of *equation-error* network structures are evaluated very rapidly. The most promising structures are then saved, feedbacks are connected, and the algorithm searches this subset for the best *output-error* network.

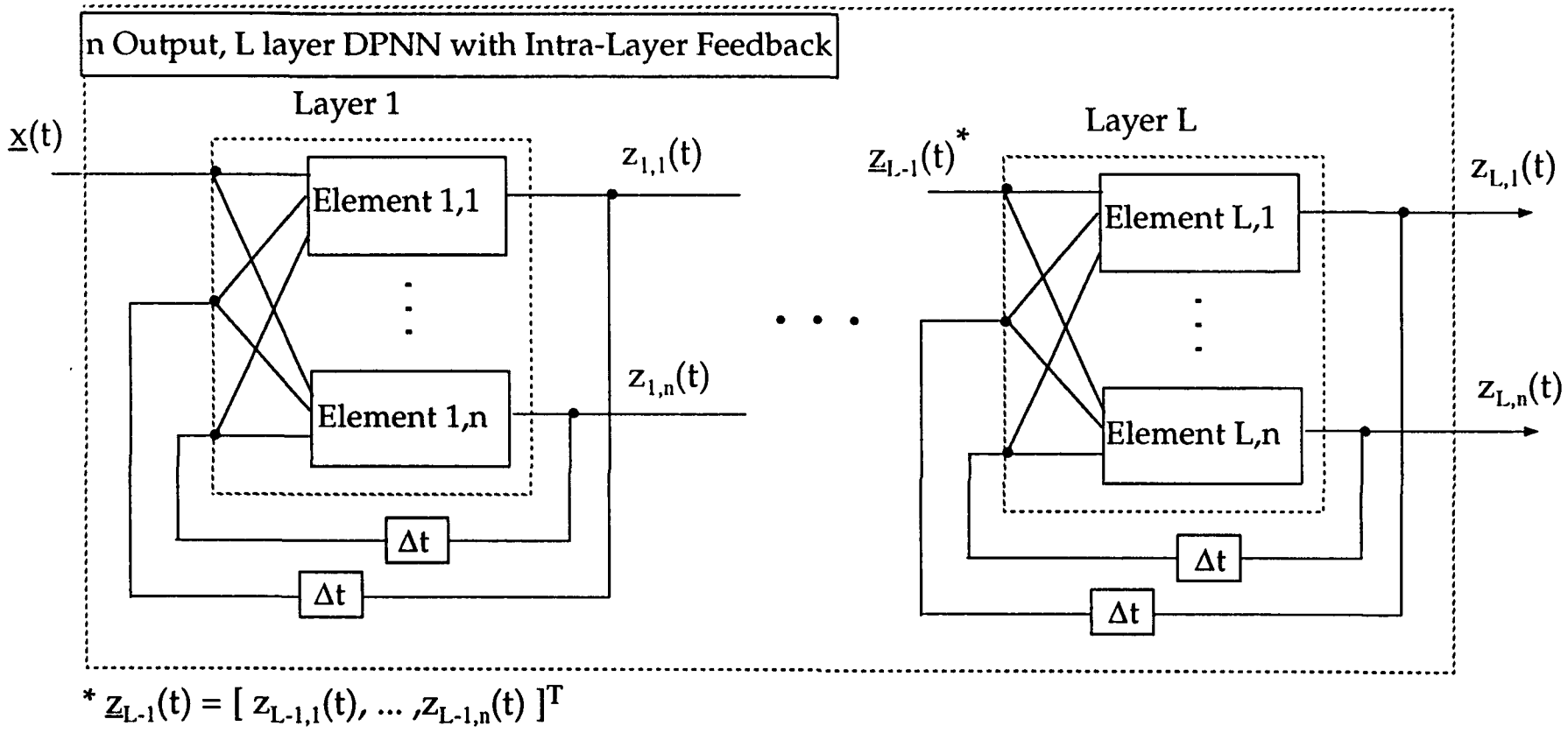


Figure 3.8: MIMO DPNN with Intra-Layer Feedback

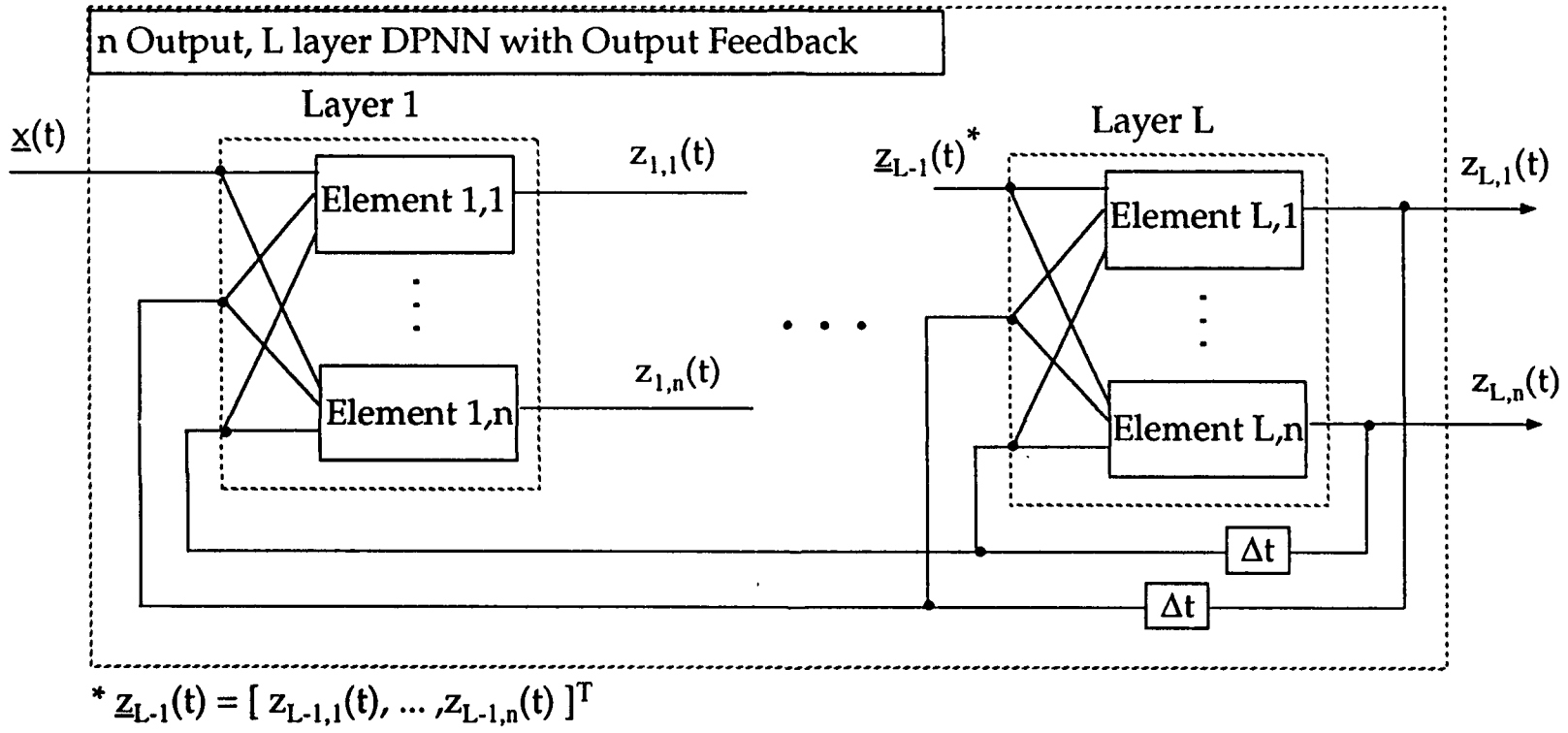


Figure 3.9: MIMO DPNN with Output Feedback

Fig. 3.10 shows the general algorithm for constructing either an output-feedback or intra-layer-feedback MISO DPNN. This algorithm may be easily extended to the MIMO case as in *DynNet;* however, the MISO case is shown here for readability.

The *DynNet* algorithm shown in Fig. 3.10 may be summarized as follows:

1. Find the M best MISO equation-error *Dyn3* networks for each output, with the element complexities limited as described above. If a database output is used as an input variable, make sure it passes through a one-step time delay.

2. To account for additional error that might be introduced when the true equation-error (database) values of the network output are replaced by feedbacks of the output estimates, add an additional PSE penalty term proportional to the prior estimate of the error variance of the variable that will be fed back.

3. Recombine the elements from Step 1 so that there are M potential MIMO layers, with each layer containing one element corresponding to a network output.

4. If intra-layer feedback is desired, replace the *equation-error* network inputs which are prior values of the output columns of the training database, with feedbacks of the output estimates, thus creating M potential *output-error* first layers.

5. Optimize the coefficients of each layer using a *GR/GARS* search technique.

6. Continue adding layers until the overall network score ceases to improve. Subsequent layers are built in the same way as the first layer, except that in addition to system inputs and outputs, every element on subsequent layers must have at least one input which comes from the immediately preceding layer.

The above algorithm may be tuned and modified in a number of ways depending on the application.

3.4.3 Command-Line Options

A useful way by which to summarize the use of (and potential modifications of) to the algorithm is to describe the command-line options available in *DynNet.* To execute *DynNet* in UNIX, simply type "DynNet", followed by one or more of the following command-line options:

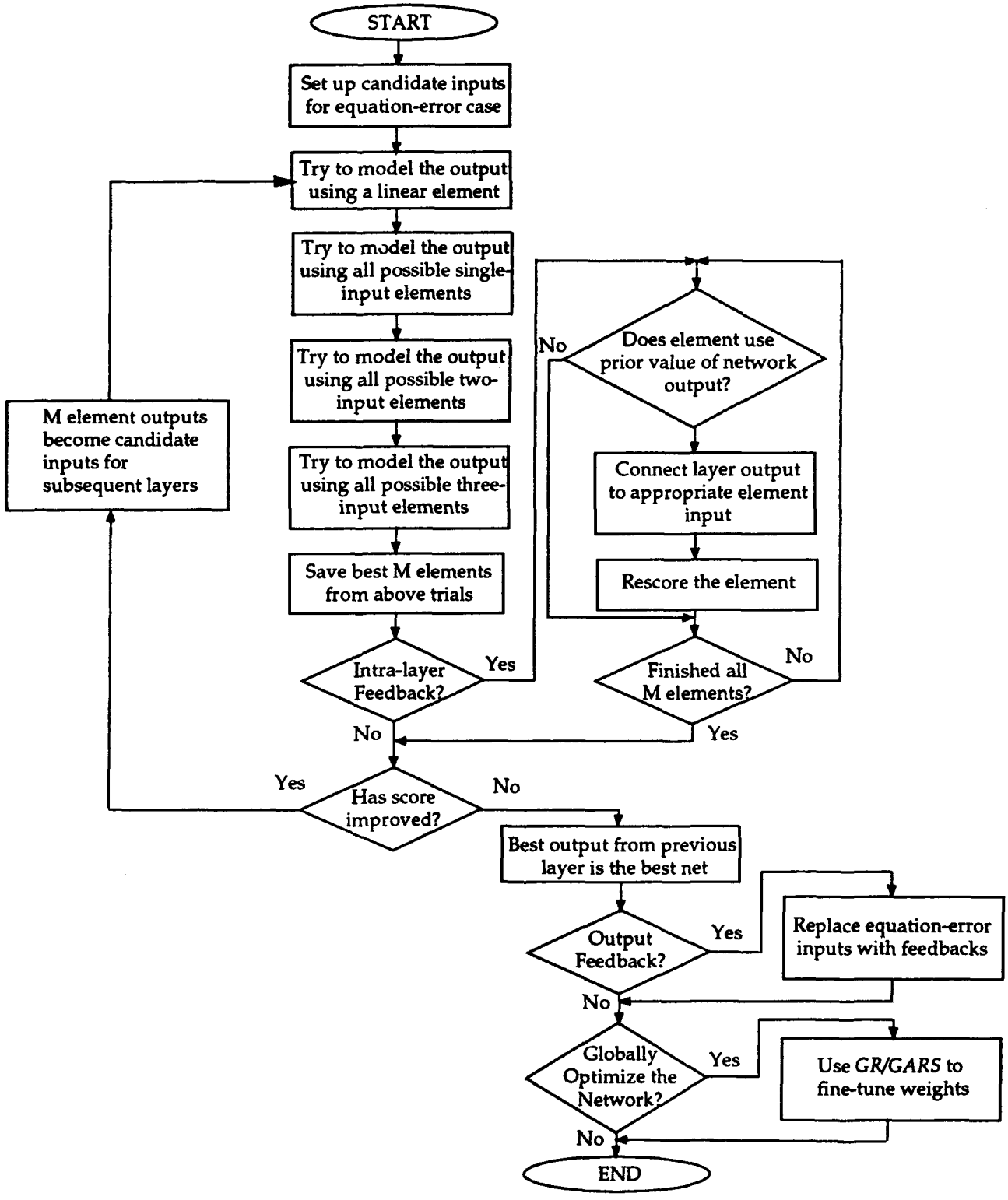


Figure 3.10: ***DynNet*** **Algorithm for Constructing a MISO DPNN (Extendable to MIMO DPNN)**

- **-train** *training_data*

  To train a network, the training data must be in a file called the training database. Data files are composed of blocks, observations, and variables. A block (typically a time series) is made up of observations. Each block may contain a different number of observations. A data file is broken into a header section and a data section. For TEXT files, the header section is composed of an optional "size" line, and a line of variable names. The "size" line is composed of a "#", followed by the number of variables, the number of blocks, and the number of observations in each block. In BINARY files, the variable names are stored along with the number of blocks and the number of observations in each block.

  In TEXT files, each block is terminated with four hyphens on a line by itself. Each observation has the same number of variables and is terminated by a new line. Observations may be split onto multiple lines by ending a line with a backslash. The number of variables, observations, and blocks is not limited.

  After a *DynNet* network is created it is written out to a file with the name "DN.mod". In addition, the performance of the network on the training database is recorded in the "DN.eval" file.

- **-eval** *mod_file evaluation_data*

  The **-eval** option allows a network that is stored in the *mod_file* to be evaluated on unseen data that are stored in the *evaluation_data* file. The evaluation data file has the same format as the training data file described above; although the order of the columns containing the variables is not important, variable names are required to be the same as those used to train the network. Either **-train** or **-eval** must be specified each time *DynNet* is executed.

- **-o** *output_variable(s)*

  The **-o** option allows the user to specify which column(s) in the training data base is (are) to be considered network output(s). A case-sensitive list of variable names follows the -o option. For each name, there must be a corresponding column in the training database, otherwise *DynNet* will report an error.

- **-i** *input_variable(s)*

  The **-i** option allows the user to specify which column(s) in the training data base is (are) to be considered network input(s). If the **-i** option is not

specified, all variables besides the output variables are assumed to be candidate network inputs.

- **-ignore** *ignore_variable(s)*

  The **-ignore** option allows the user to specify columns in the database that should be ignored by the network generation algorithm. Typically either the **-i** or the **-ignore** option is specified, but not both.

- **-od** *output_delay(s)* and **-id** *input_delay(s)*

  DynNet allows the user to specify the values of delays to be used in the tapped delay lines internal to the *DynNet* nodal element. The algorithm allows the user to distinguish between the delay values that will be used on element inputs that have been fed forward from network inputs or previous layers, **-id**, and delay values that will be used on element inputs that have been fed back from this or subsequent layer outputs, **-od**.

  If **-id** is not specified, it is assumed to be zero. If **-od** is not specified, it is assumed to be one; an **-od** value of zero is not valid.

- **-olog** *maxdelay [start]* and **-ilog** *maxdelay [start]*

  -olog and **-ilog** are shortcuts for specifying feedback and feedforward delay values respectively. They allow the analyst to specify logarithmically spaced (base two) delay values between *maxdelay* and *start*. If *start* is not specified, it is assumed to be 1. For example, the command **-olog** 32 would result in feedback delays of 2, 4, 8, 16, and 32.

- **-olin** *maxdelay [start spacing]* and **-ilin** *maxdelay [start spacing]*

  -olin and **-ilin** are shortcuts for specifying feedback and feedforward delay values, respectively. They allow the analyst to specify linearly spaced delay values between *maxdelay* and *start*; the delays are *spacing* apart. For example, the command **-ilin** 0 20 5 would result in feedforward delays of 0, 5, 10, 15, and 20.

- **-netfb** and **-layerfb**

  The **-netfb** and **-layerfb** options tell *DynNet* whether to use output-feedback or intralayer-feedback mode respectively. Only one of these options may be specified.

- **-nodefb**

  Typically, nodal elements are initially fitted using the equation-error method described above. If the **-nodefb** flag is specified, *DynNet* will feed back the nodal element output as input to the node, instead of using the

corresponding equation-error database input. For example, suppose one of the possible nodes to be tried had as inputs x_1, y_1, and y_2. Further, suppose that this particular node is responsible for producing $\hat{y}_2$. With the **-nodefb** flag set, *DynNet* would set up this node using only two database inputs, x_1 and y_1; however, *DynNet* would also set up a feedback connection between the node output and its input.

The **-nodefb** flag can significantly slow down training time because a number of candidate nodes will now have feedback connections and cannot be fitted using batch least squares. **-nodefb** can be especially helpful for single output networks when the equation-error network is not bootstrapping the output-error network sufficiently.

- **-nosharing**

  When building a multiple output network, *DynNet* allows nodes responsible for modeling one of the outputs to use, as candidate inputs, outputs from nodes responsible for modeling other outputs. When the **-nosharing** flag is set, *DynNet* treats the M-output network as M separate single-output networks.

- **-ml** *degree*, **-a** *degree*, and **-c** *degree*

  The **-ml**, **-a**, and **-c** options allow the user to specify what type of KG multinomial expansion should be used internal to the nodal element. The user has a choice of the *multilinear*, *additive*, or *complete* polynomials as defined in Section 3.4.1. The *degree* specifies the expansion rank.

- **-sigmap** *sigma_prior(s)*

  Since *DynNet* uses the PSE criterion to score a network, the analyst must specify a *sigma prior* (σ_p) for each output. The σ_p value is an *a priori* estimate of the model errors in the units of the data. Sigma-prior values directly affect the model complexity penalty; the larger the value chosen, the more each added degree of freedom (each new coefficient) will be penalized in a candidate model, so that the final model will be less complex. Therefore, the value of σ_p is adjusted by the analyst to obtain models with appropriate levels of complexity.

  The default value of σ_p is one-tenth the measured standard deviation of the output variable; however, it is recommended that the analyst override this value with their own estimate. If, after network generation, it turns out that the σ_p chosen was either significantly larger or smaller than the

true model error standard deviation, σ_p may be adjusted and a new model created.

- **-globalo**

  The **-globalo** option instructs *DynNet* to optimize *globally* all the network coefficients once the generation of the network structure is complete. Globalo uses the *GR/GARS* algorithm (outlined in Appendix B) for the optimization.

- **-ls**

  The **-ls** option instructs *DynNet* to begin the output-error optimization with the coefficients found via equation-error optimization. These coefficients are called the *least-squares* coefficients.

- **-carve** and **-build**

  Either **-carve** or **-build** may be specified, but not both. These options allow the number of coefficients internal to each element to be reduced using the *carving* or *building* algorithms described in Section 3.3.2.

- **-white**

  This option instructs *DynNet* to try *linear* elements (Fig. 3.7) in addition to the type of element specified via the **-ml**, **-a**, or **-c** options described above.

- **-layerlim**

  -layerlim specifies the maximum number of layers that *DynNet* will try while generating a specific network model.

- **-layersize** *layersize(s)*

  -layersize allows the analyst to specify the number of "best" elements that will be kept after all the equation-error element possibilities have been tried (Fig. 3.10). The analyst may specify a unique M for each layer.

3.5 Future Research

The *DynNet* and *Dyn3* algorithms have been successfully applied to a number of modeling, prediction, and control tasks. Some of these applications are described in Section 4 of this report. During the course of applying the algorithms, some potential improvements and recommended areas for future research have been identified. These are mentioned below.

While in many circumstances finding good MFF nodes is a good way to "bootstrap" the dynamic network generation process, in some situations, especially with noisy data, the MFF results may be misleading. At the moment, fitting time limits the number of MFB structural possibilities that may be tried.

There are two improvements that could increase training time sufficiently to allow more exploration of MFB structural variations. First, by implementing the projection-pursuit type of layer training paradigm that is described in Section 2.3.3, it may be possible to reduce the number of "best" layers that are kept as candidate inputs to subsequent layers. Second, algorithms could be developed to allow a gradient-based search on any subset of coefficients in a DPNN of arbitrary structure. While this search would potentially be subject to local minima problems, it would converge, in general, much more rapidly than the *GR/GARS* direct search method currently employed. Eq. 2.52 provides the mathematical basis for such an extension.

Another potential improvement to *DynNet* would be to investigate the effectiveness of a recursive-least-squares (RLS) type of network optimization. For a given nodal element, this would involve the assumption that the first term in Eq. 2:49 is the dominant term and the effect of the gradients of the element inputs have a small effect on the overall gradient of the nodal element. If this is the case, the batch gradient routines outlined in the last chapter may be converted into recursive techniques of the least-squares or Levenberg-Marquardt type depending on the choice of the $\underline{\underline{A}}$ matrix.

Currently, for estimation networks built using the Ivakhnenko strategy (Section 2.3.3), initialization of the delay banks is not a problem. For other types of loss functions or other optimization strategies, however, it may not be obvious how to initialize the shift registers internal to the network. One solution to this problem would be to modify the optimization algorithms of Section 2.3.4 so that the initial values of the outputs of each element are also treated as parameters which need to be optimized. This could potentially eliminate the need for a heuristic to find the best set of initializing parameters.

Both *Dyn3* and *DynNet* use the squared-error loss function to build networks suited for estimation purposes. By replacing this loss function with the logistic-oss function (Eq. 2.20), and by adding a post processor that calculates conditional class probabilities (Eq. 2.22), it would be possible to build networks that can classify time-series data. Currently, much of the work involved in classification of time-varying signals consists of determining and extracting appropriate feature "snapshots" of the data to send as inputs to a static feedforward network. A dynamic neural network classifier could potentially find and generate its own time-varying features as a natural outcome of the optimization process. Such a classifier might be helpful in signal processing and active control applications.

4. APPLICATION EXAMPLES

4.1 Active Control of Complex Combustion Processes

In Section 2, a general paradigm was presented for understanding both static and dynamic neural network function estimation. In Section 3, two specific algorithms, *Dyn3* and *DynNet* were introduced. These algorithms are specific implementations of a general paradigm for networks based on polynomial expansions and containing internal feedbacks and time delays. The current section illustrates the utility of these dynamic network synthesis algorithms for active prediction and control by demonstrating their successful use in several applications and comparing these results with those obtained using static feedforward networks.

The need to control complex systems under conditions of uncertainty has made apparent the need for new methods that overcome the shortcomings of conventional control techniques. For example, active control of complex processes requires that the controller learn the dynamics of (generally) incompletely specified (and possibly nonlinear and time-varying) plants from noisy data, rather than having these dynamics specified analytically. Neural-network based controllers, or neurocontrollers, offer promise in this area because of their ability to learn inductively (which thereby provides an on-line adaptation capability and helps avoid model development costs), their ability to approximate functions, and their potential for parallel hardware implementation (and therefore rapid computation) [Antsaklis, 1992].

Despite the large number of designs in the literature, neurocontrollers may generally be placed into one or more of five generic categories: (1) supervised controllers, (2) direct inverse controllers, (3) neural adaptive controllers, (4) controllers based on the backpropagation of utility, and (5) controllers based on adaptive critic methods [Werbos, 1991]. These controllers tend to grow in complexity in the order that they are listed. These five basic approaches are briefly outlined next.

Supervised control requires that the desired controller output be known prior to network training. *Direct inverse control* is based on using the inverse of the plant dynamics to control the system. In *neural adaptive control,* neural networks replace the conventional linear mappings used, for example, in model-reference adaptive controllers and in self-tuning regulators. *Backpropagation of utility* (through time) adapts an optimal controller by adjusting network weights so as to extremize an objective function. (Note that adaptive optimizing control, where the controller continually seeks on-line to extremize an objective function, without *a priori* attempt to capture in a controller model the dynamics of the plant, is included here as a special case.) Finally, the underlying idea in *adaptive critic control* is to approximate the Bellman equation of dynamic programming using adaptive networks.

Each of these generic approaches may be further subdivided into indirect and direct adaptive control methods. *Indirect* methods employ system identification techniques to obtain an explicit model of the plant; model parameters are then inserted directly into the controller. *Direct* methods determine the control rule without forming such a system model; controller parameters are adjusted to reduce some norm of the output error [Sutton, 1992; Narendra and Parthasarathy, 1990]. The latter approach is taken in the application examples illustrated below.

4.2 Inverse Control of a Ducted Flame

The first application example comes from work that was performed in collaboration with personnel of the Naval Air Warfare Center, Weapons Division (NAWC/WD), China Lake, California. This work was focused on extending the flammability limits in a ducted flame apparatus, and on suppression of combustion pressure oscillations in a one-megawatt laboratory combustor. Specifically envisioned is the use of active control techniques to acoustically manipulate the initial shear-layer instability at the flameholder, thereby disrupting the development of large-scale coherent vortical structures; the latter have been shown to affect the flameholding process and to drive combustion instabilities. Prior experiments have demonstrated the capability of active control of acoustic forcing to affect the reacting shear layers, thereby extending the operating envelopes over those achievable with passive techniques. This work is leading to the use of adaptive nonlinear neural network feedback control of combustion systems, which are open-loop unstable over part of their operating range [Hansen and Hendricks, 1992].

The data provided by NAWC/WD included six data sets (herein referred to as ACTD1.1, ACTD2.1, ACTD3.1, ACTD3.2, ACTD3.3, and ACTD3.4), each composed of 16,384 samples representing the input acoustic modulation (AM) signal and the flame response measured by a photodiode (PD), as shown in Fig. 4.1. These data were collected with the system in an open-loop configuration and presumably over a linear region of operation. Files ACTD1.1 and ACTD2.1 represent data collected on the plant using a different (and poorer quality) AM excitation signal than was used for data sets ACTD3.1, ACTD3.2, ACTD3.3, and ACTD3.4. Therefore, in the results below, only the latter four data sets are discussed. The task is to synthesize a controller for the plant, whose function is to process the measured PD signal and to provide the subsequent AM excitation of the system.

Because no mapping of desired controller actions for different sensor measurements was available, supervised control was not relevant; direct inverse control then represents the next-simplest technique. As shown in Fig. 4.2, direct inverse control uses the system identification of the plant to synthesize an effective controller; the method is basically an open-loop technique, but the loop is actually

closed through adaptation of the plant model.[4] In actual implementation, the configuration for implementing direct inverse control illustrated in Fig. 4.3 would be preferred for computational reasons.

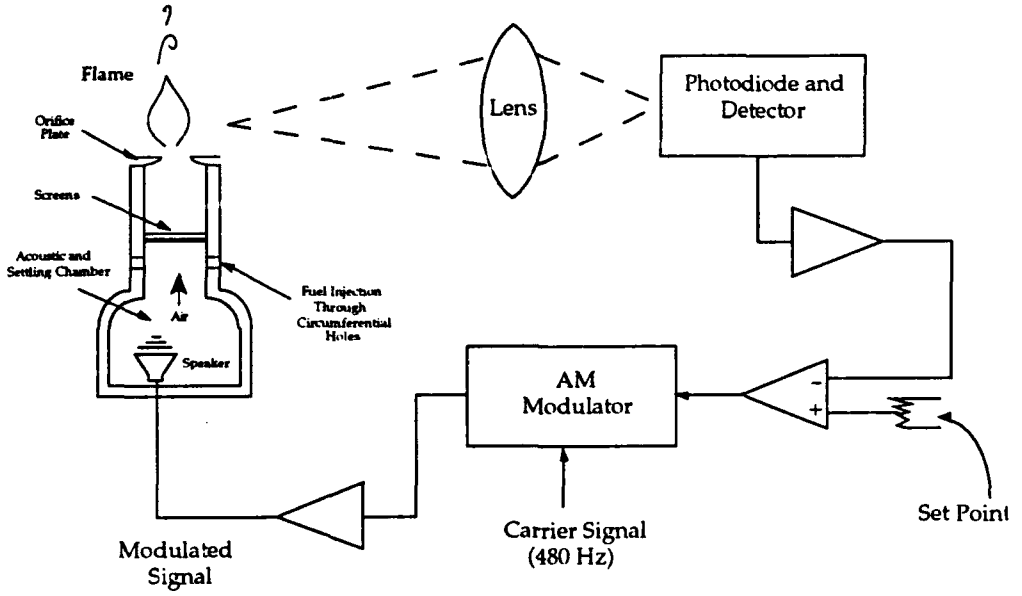


Figure 4.1: Schematic Diagram of Ducted Flame Apparatus

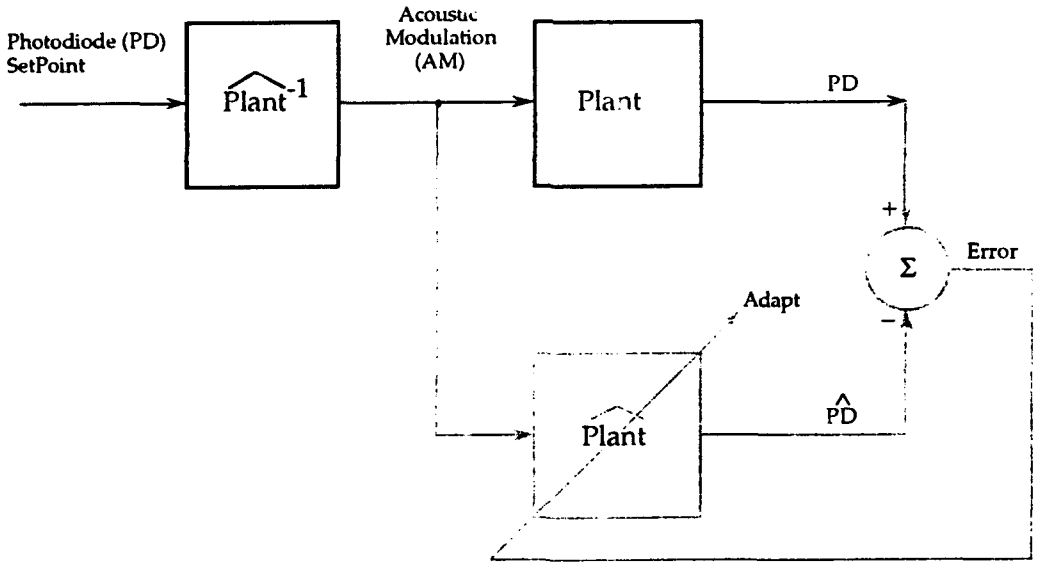


Figure 4.2: Control System for Regulating Flame Height or Quality

[4] In all figures, the light-gray lines represent processes relating only to adaptation of the model, not the main processing path, which is reflected by the black lines.

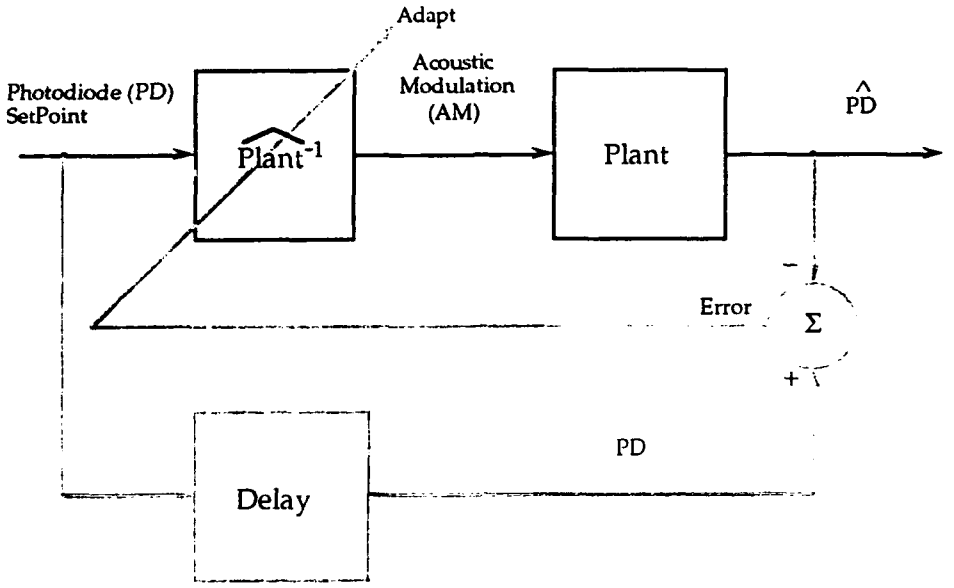


Figure 4.3: Preferred Implementation of Control System for Regulating Flame Height or Quality

4.3 Static Polynomial Neural Network System Identification Results

Fig. 4.4 illustrates the test setup used to evaluate how the polynomial neural network (PNN) models performed in predicting the flame response (measured by a photodiode) to acoustic modulation (using AM excitation signals). The static PNN configuration used is shown in Fig. 4.5. Performance results are tabulated in Table 4.1 and are shown in Figs. 4.6 and 4.7, where each model is used to predict the photodiode response to the AM excitation data. Fig. 4.6 gives prediction results for the same AM excitation as that used to identify the model; Fig. 4.7 gives an example of the fitting performance when the AM excitation was different from that used to identify the plant (*viz.*, file ACTD3.1). Fig. 4.6 thus corresponds to the diagonal terms in Table 4.1, and Fig. 4.7 corresponds to the off-diagonals in Table 4.1 for the row labeled ACTD3.1.

As mentioned above, the data on the diagonal in Table 4.1 represent the performance on the fitting data, whereas off-diagonals represent the test (i.e., evaluation) data. In the operating region around which the systems were characterized, the models performed nearly as well and sometimes better on the evaluation data as on the fitting data. It is therefore concluded that the models have extracted the relevant information from the available data.

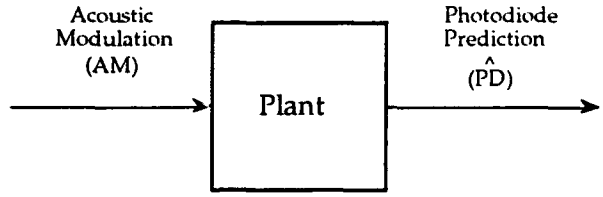


Figure 4.4: Use of Identified Plant Model in Estimation Experiment

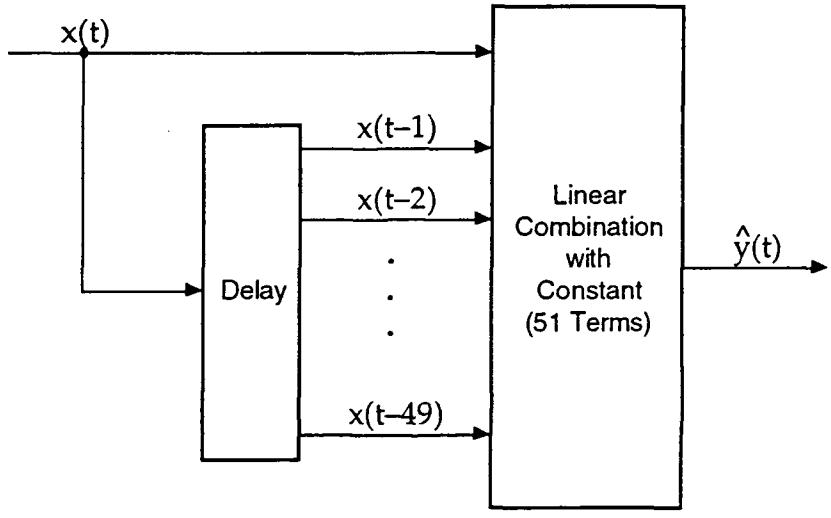
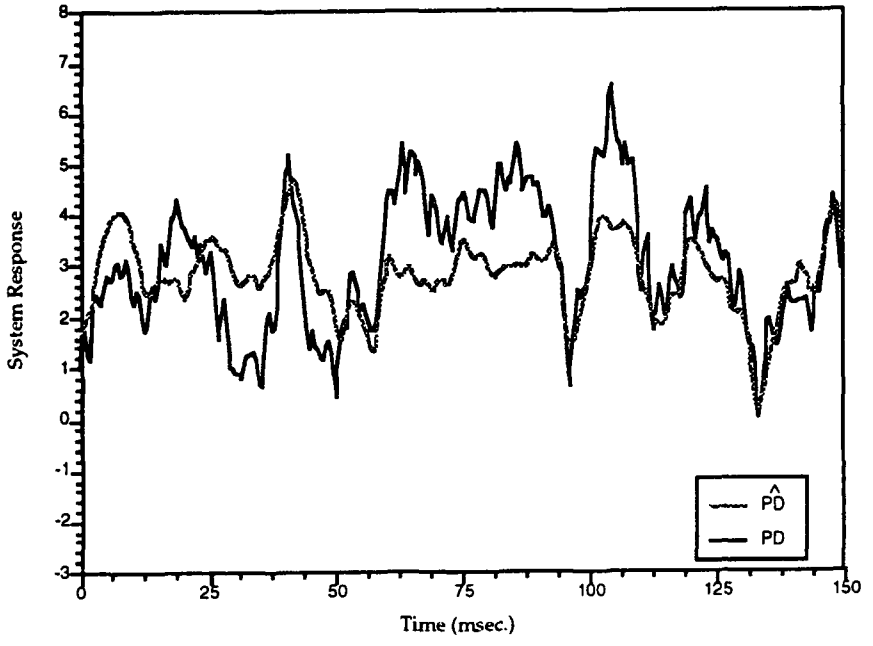


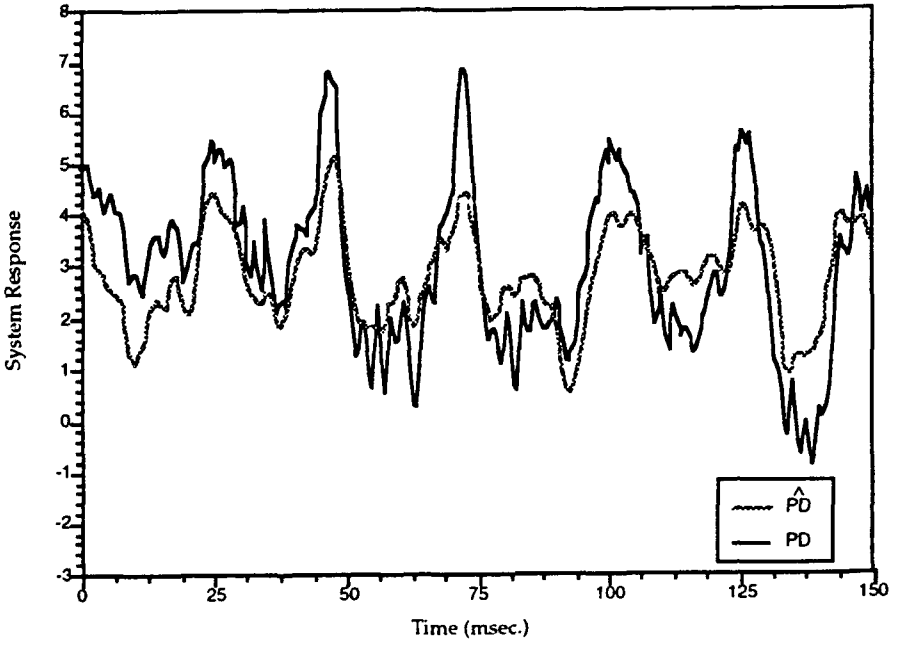
Figure 4.5: Static PNN Used in the System Identification Experiments

Table 4.1: Mean Square Error of Plant Output Prediction Using Static PNN

Test→ Fit ↓	ACTD3.1	ACTD3.2	ACTD3.3	ACTD3.4
ACTD3.1	1.43	1.61	1.57	1.47
ACTD3.2	1.45	1.59	1.58	1.45
ACTD3.3	1.44	1.61	1.56	1.48
ACTD3.4	1.46	1.60	1.59	1.44

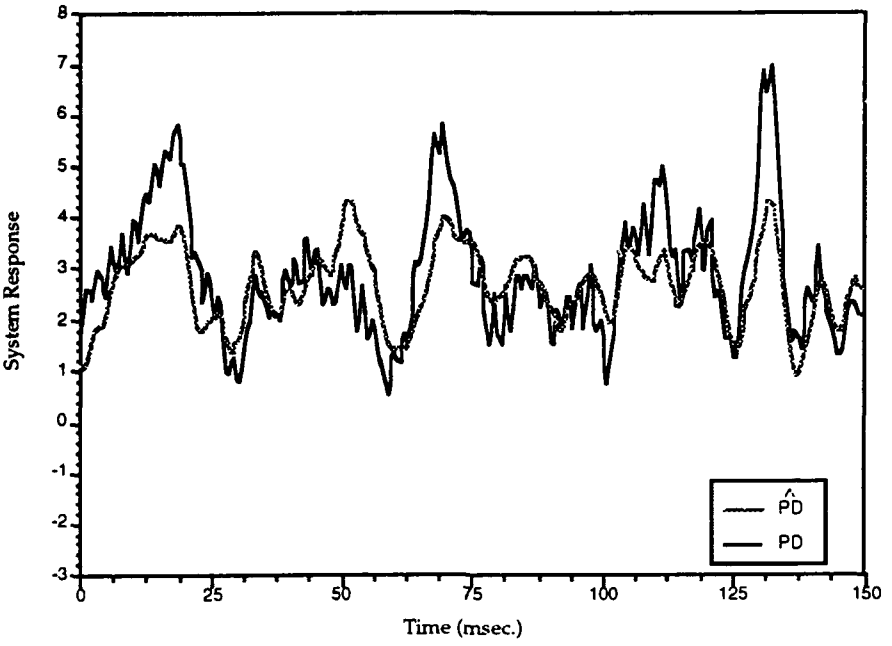


(a) ACTD3.1

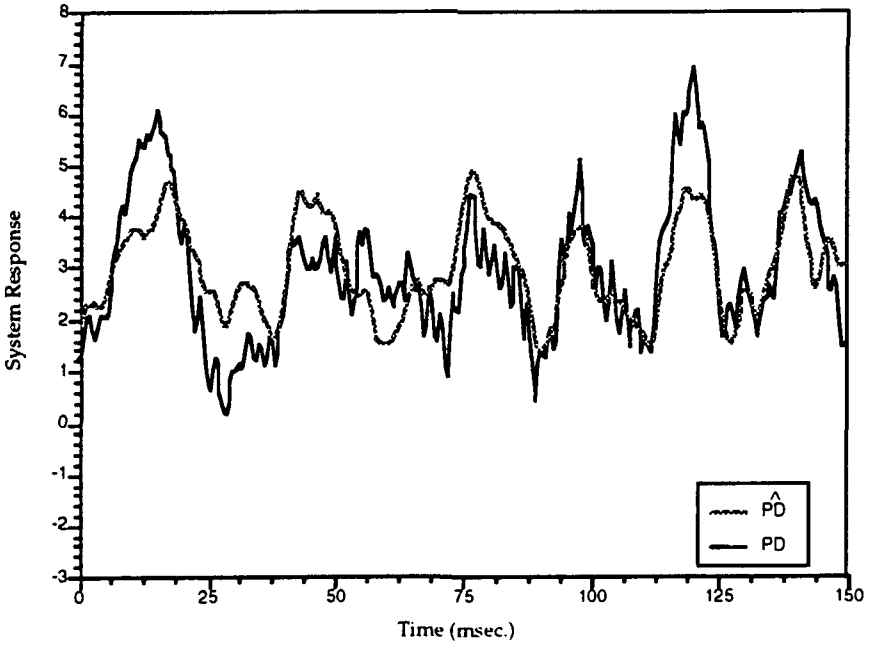


(b) ACTD3.2

Figure 4.6: Static PNN Prediction of Photodiode Output Using the Same AM Excitation Signal as that Used to Fit the Model

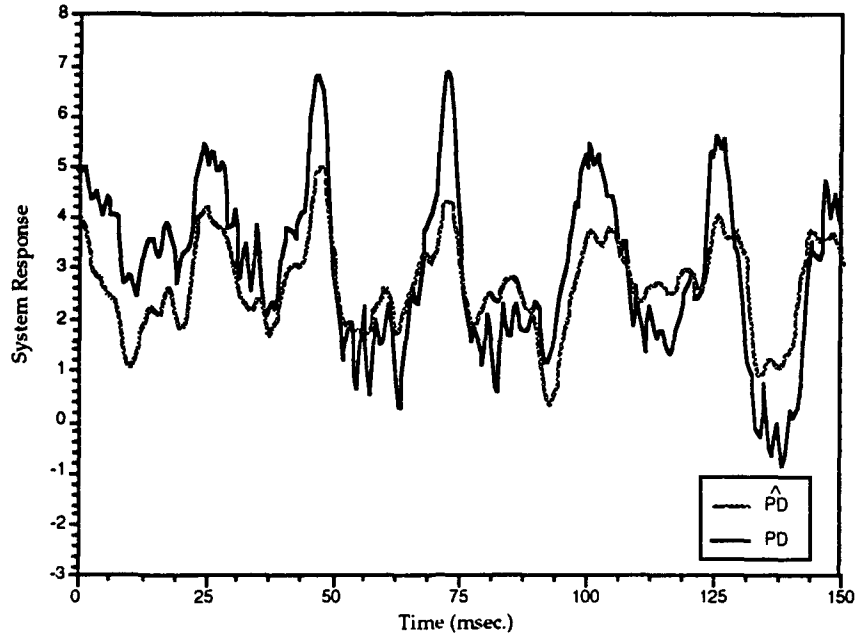


(c) ACTD3.3

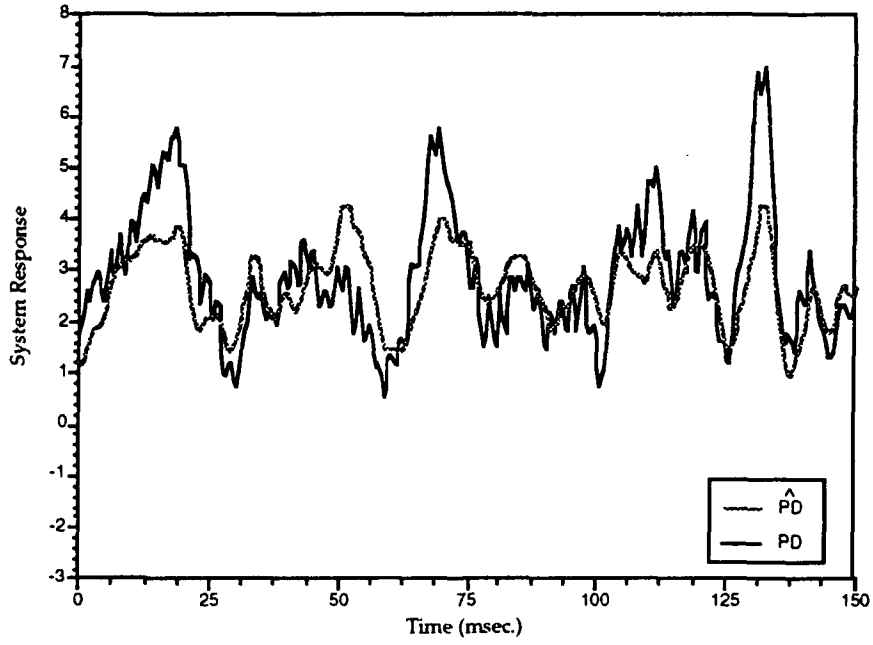


(d) ACTD3.4

Figure 4.6 (continued): Static PNN Prediction of Photodiode Output Using the Same AM Excitation Signal as that Used to Fit the Model

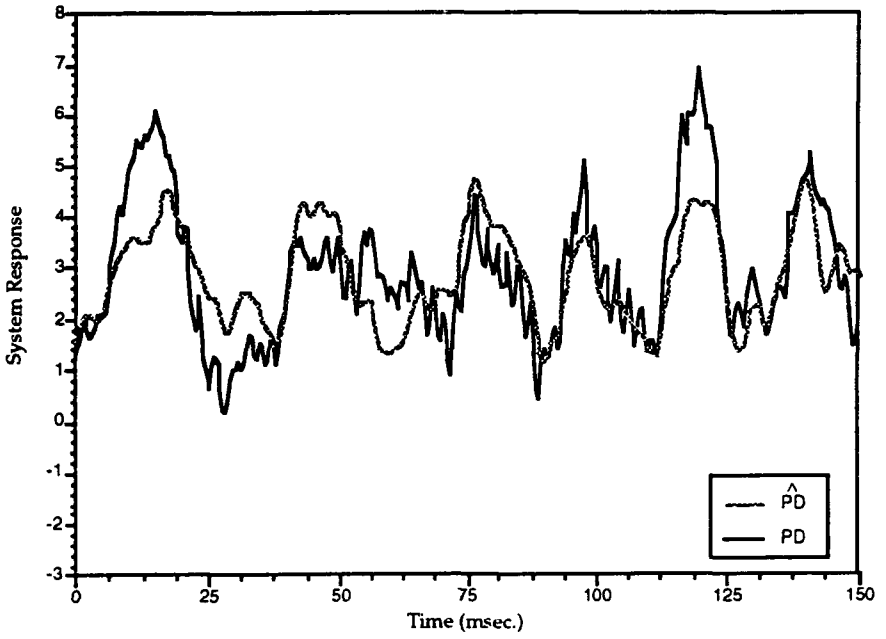


(a) ACTD3.2



(b) ACTD3.3

Figure 4.7: **Static PNN Prediction of Photodiode Output in Response to AM Excitation Different than was Used to Fit the Model (*viz.*, file ACTD3.1)**



(c) ACTD3.4

Figure 4.7 (continued): Static PNN Prediction of Photodiode Output in Response to AM Excitation Different Than Was Used to Fit the Model (*viz.*, file ACTD3.1)

4.4 Dynamic Polynomial Neural Network System Identification Results

The architecture of the dynamic polynomial neural network (DPNN) used to model the plant is given in Fig. 4.8. Table 4.2 and Figs. 4.9 – 4.10 provide results for the DPNN for the same experiments as conducted above for the static PNNs. *Note in Table 4.2 that performance is improved over the static case even though the DPNN model had less than one-fifth as many internal degrees of freedom.*

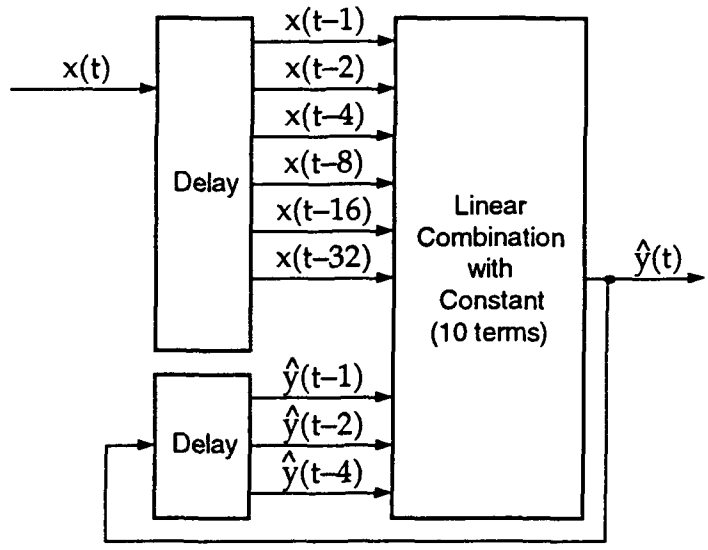
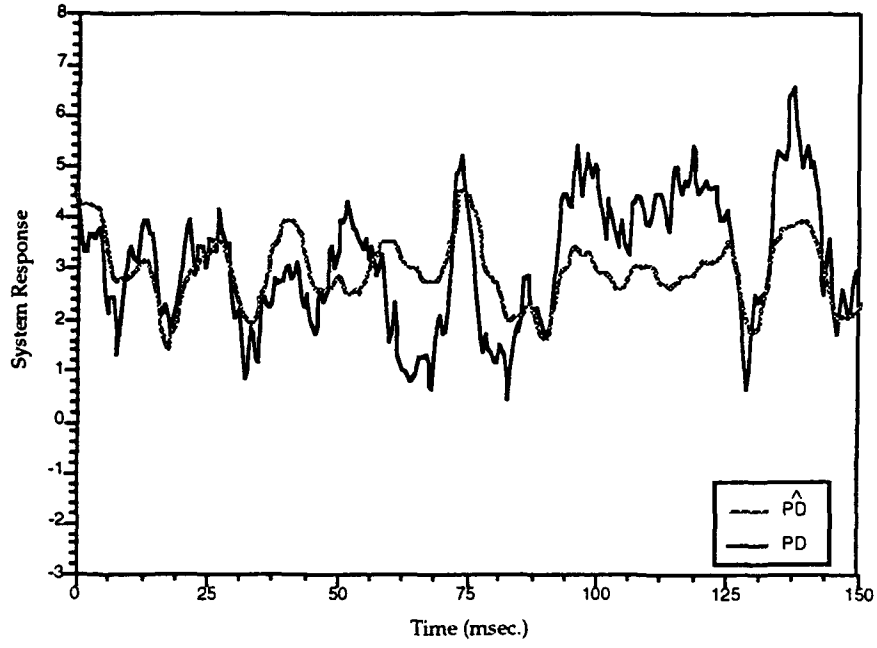


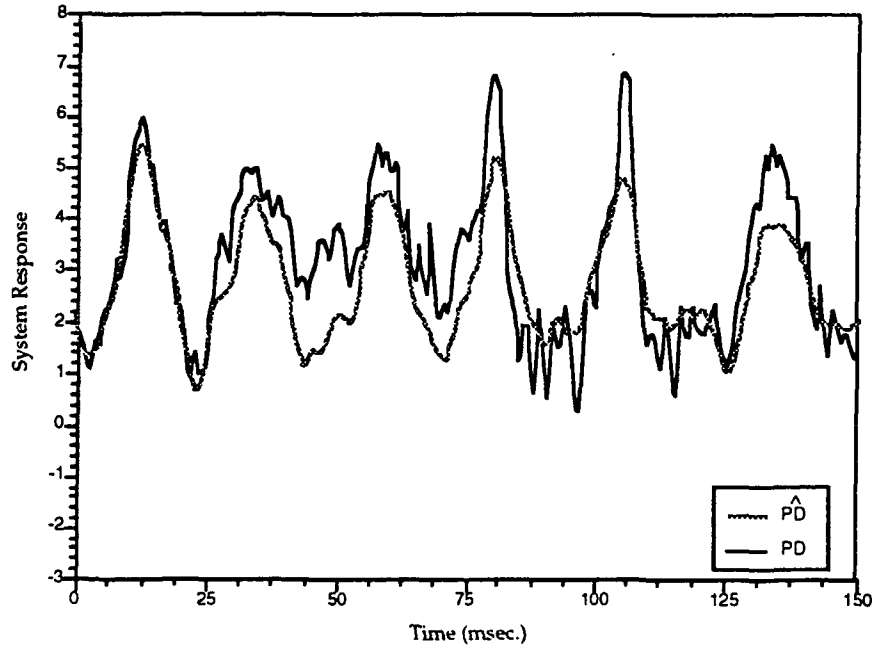
Figure 4.8: **Dynamic PNN Used in the System Identification Experiments**

Table 4.2: **Mean Square Error of Plant Output Prediction Using Dynamic PNN**

Test→ Fit ↓	ACTD3.1	ACTD3.2	ACTD3.3	ACTD3.4
ACTD3.1	1.40	1.56	1.53	1.43
ACTD3.2	1.42	1.54	1.54	1.42
ACTD3.3	1.40	1.56	1.52	1.44
ACTD3.4	1.43	1.55	1.56	1.40

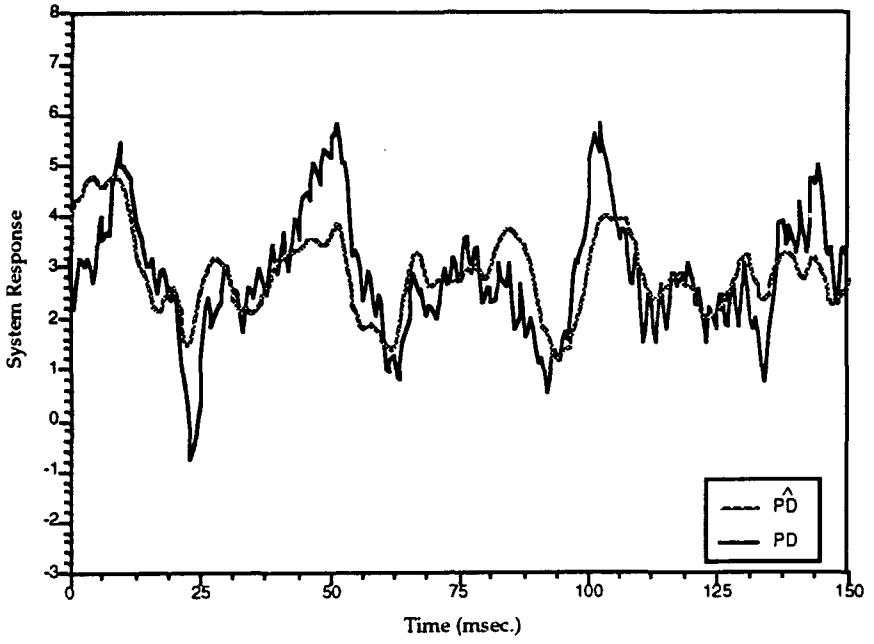


(a) ACTD3.1

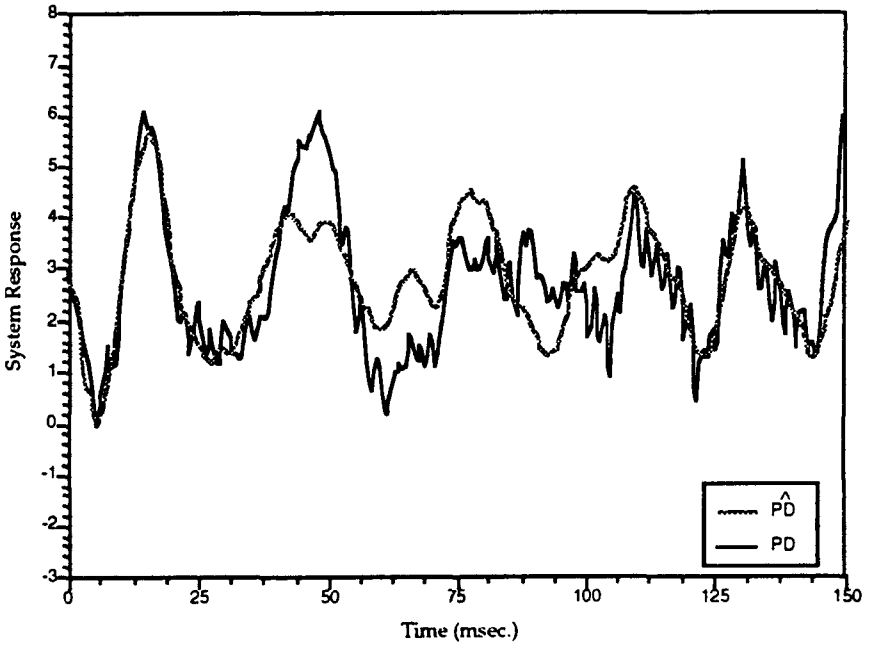


(b) ACTD3.2

Figure 4.9: **Dynamic PNN Prediction of Photodiode Output Using the Same AM Excitation Signal as that Used to Fit the Model**

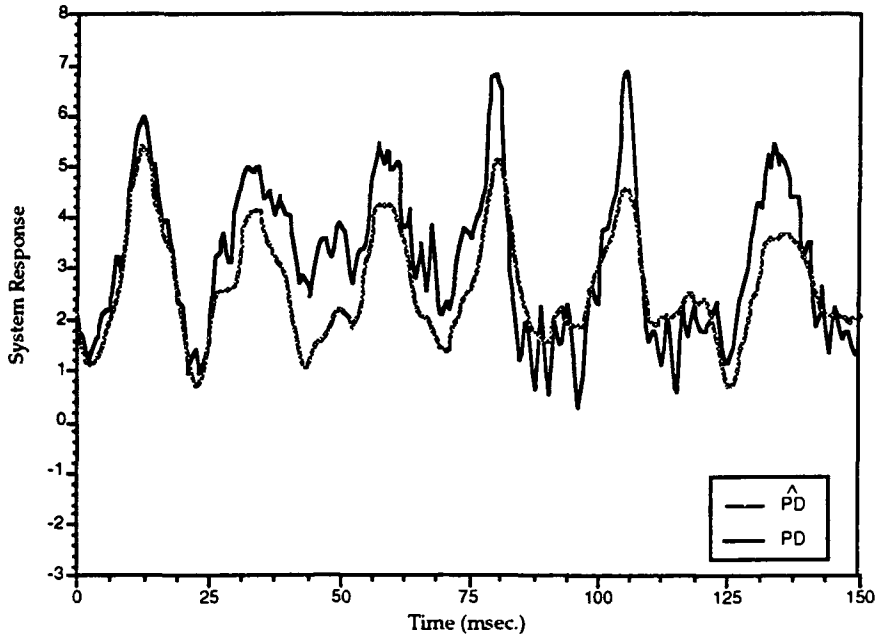


(c) ACTD3.3

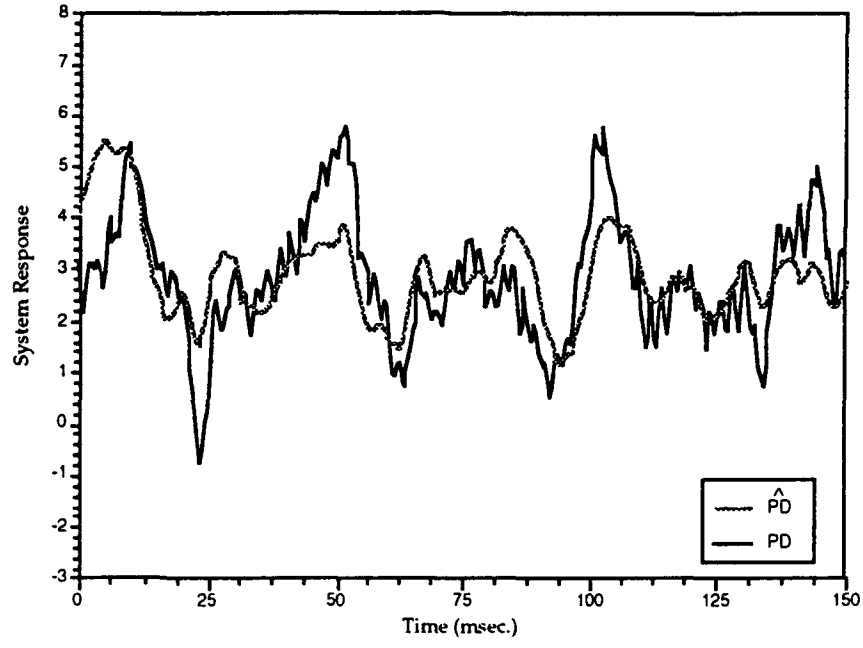


(d) ACTD3.4

Figure 4.9 (continued): Dynamic PNN Prediction of Photodiode Output Using the Same AM Excitation Signal as that Used to Fit the Model

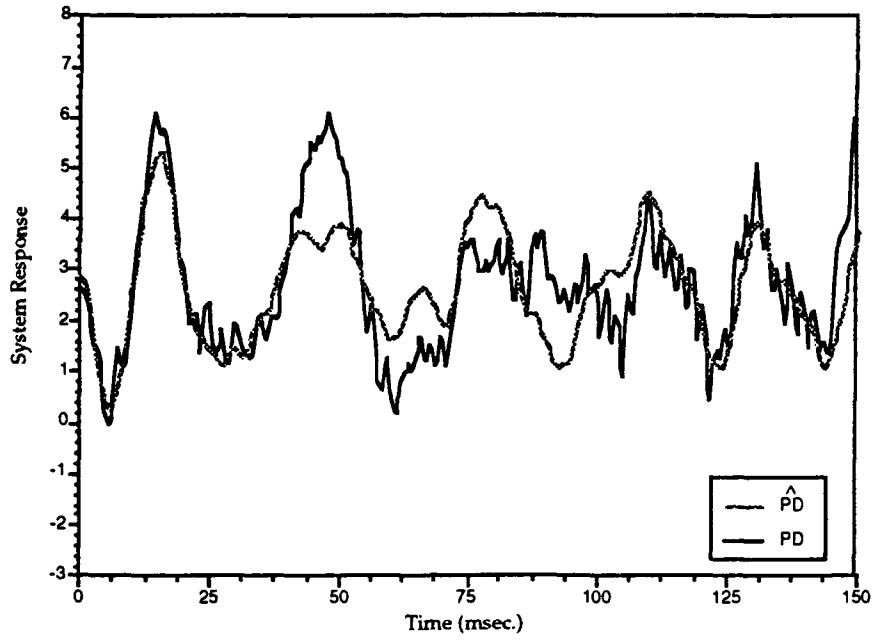


(a) ACTD3.2



(b) ACTD3.3

Figure 4.10: Dynamic PNN Prediction of Photodiode Output Using Different AM Excitation Signal than was Used to Fit the Model (*viz.*, ACTD3.1)



(c) ACTD3.4

Figure 4.10 (continued): Dynamic PNN Prediction of Photodiode Output for AM Excitation Signals Different than were Used to Fit the Model (*viz.*, file ACTD3.1)

4.5 Plant Control

The models identified above were next used in the feedforward control system illustrated in Fig. 4.11. In illustrating the technique below, the plant was taken to be the static PNN model identified using the file ACTD3.1. The approach then was to use the inverse of the identified model (i.e., $PLANT^{-1}$), computed using deconvolution, for each of the data sets in open-loop plant control. A unit step input signal was used to evaluate the performance of the controller; these results are given in Fig. 4.12, where the step input and the system responses are plotted together. Note that in Fig. 4.12(a), as expected, the desired and actual responses are the same; this is because the convolution of the PLANT with its inverse, $PLANT*PLANT^{-1}$, by definition, is unity in this case. Due to differences in the estimated $PLANT^{-1}$ for the other data files, however, $PLANT*PLANT^{-1}$ is not unity in the other cases. In actual application, steady-state (and transient) errors would be reduced through on-line adaptation of the plant inverse model coefficients.

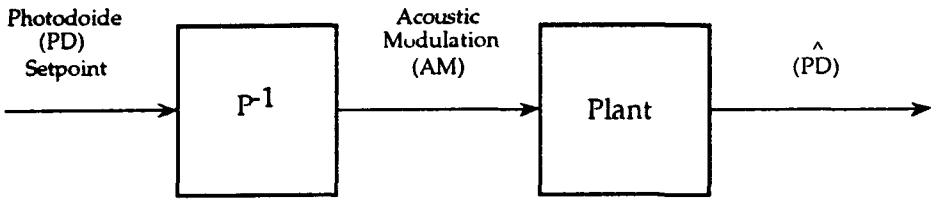
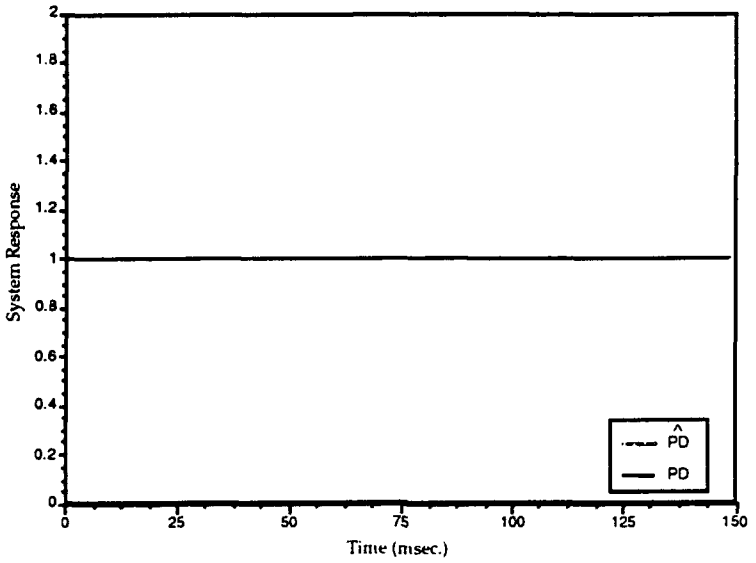
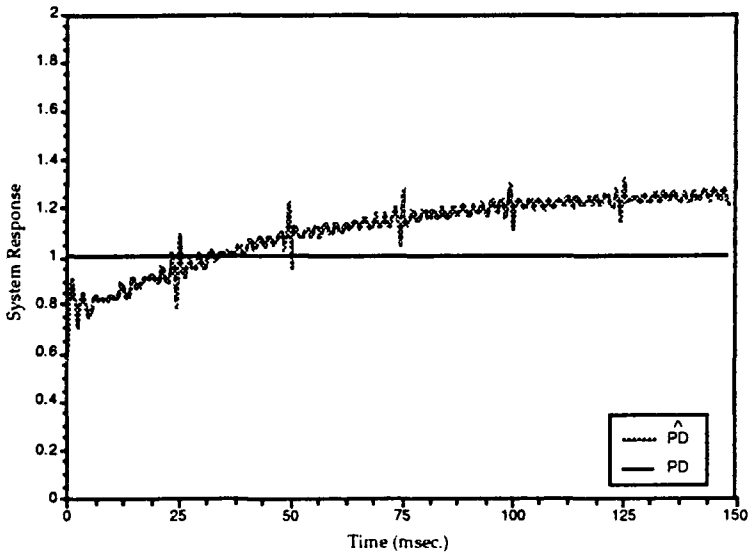


Figure 4.11: Use of Direct Inverse Model in Feedforward Control Experiment

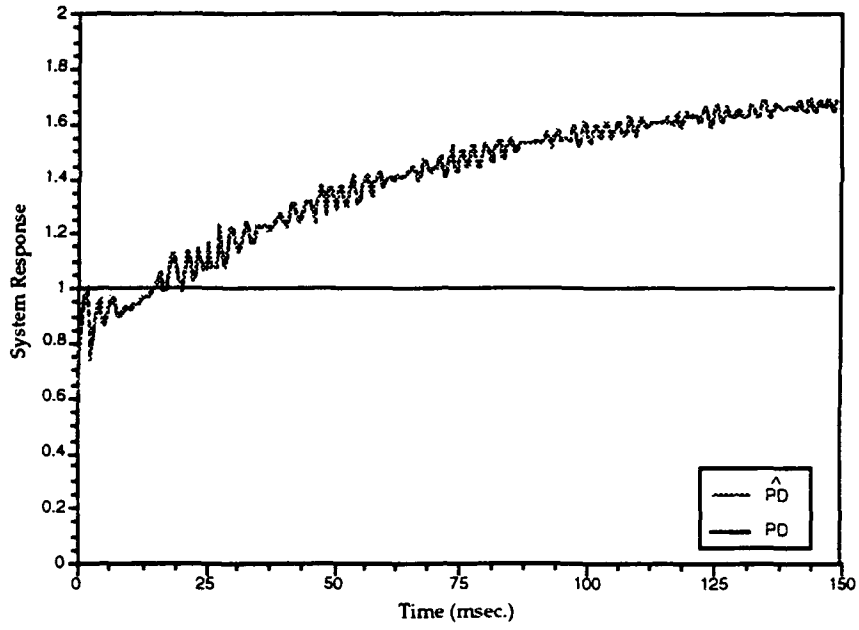


(a) ACTD3.1

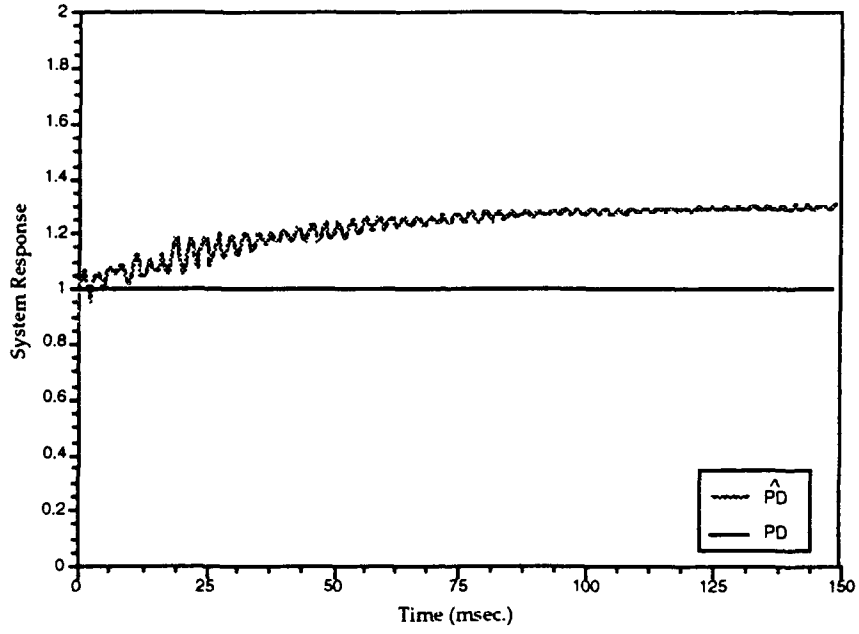


(b) ACTD3.2

Figure 4.12: Unit Step Input and Response of Direct Inverse Controller



(c) ACTD3.3



(d) ACTD3.4

Figure 4.12 (continued): Unit Step Input and Response of Direct Inverse Controller

Appendix E provides further discussion of this application study.

These results demonstrate the utility of PNNs for prediction and active control of complex systems. Both static and dynamic networks successfully modeled the dynamics of the process relating AM excitation to photodiode response. This was accomplished using a relatively small amount of training data. Dynamic networks were demonstrated to be superior to static networks both in terms of accuracy and in parsimoniously modeling the process. This parsimony is a consequence, in part, of infinite impulse response (IIR) modeling, as contrasted with finite impulse response (FIR) modeling. Parsimonious models are important because fewer internal degrees of freedom provide for model robustness; moreover, in on-line learning, performance surface complexity and adaptation time are both related to the number of parameters that must be identified. In addition, dynamic PNNs are applicable in modeling a larger class of systems than static models, as the latter are limited to application in finite-memory systems. As a result, dynamic PNNs may be used, for example, to model oscillators as naturally as they model finite-memory systems.

4.6 Modeling Synchronous Machines Using Static and Dynamic PNNs

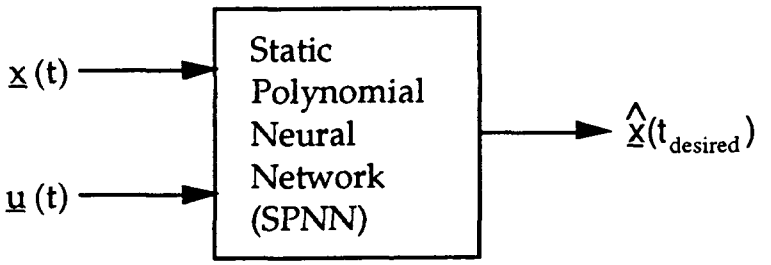
In this work, which was performed under Defense Advanced Research Projects Agency (DARPA) funding, static and dynamic polynomial neural network (PNNs) were used to model an electrical power system element as a preliminary step leading to the development of methods for optimizing control of a power grid. The eventual role for PNNs is to harmonize the differing and sometimes conflicting performance objectives of multiple machine controllers by tuning the controller parameters pursuant to a system-level criterion. PNN techniques will also be useful in power grid fault detection, isolation, estimation, and reconfiguration (FDIEC).

In this initial investigation, a synchronous machine was selected as a representative element, since this type of machine is challenging to model and is used as both a source of power (synchronous generator) and as a power system load (induction machine). Static and dynamic PNNs were used to establish computationally efficient state variable representations of a synchronous generator. The differential equations that describe synchronous machines are inherently coupled and nonlinear. Small time steps are required to integrate numerically these equations using conventional models, complicating fast predictions. If PNNs could learn to perform this prediction as a form of pattern recognition, future states of the machine could be forecast nearly instantaneously. The use of a neural network to predict future states of a device is based on the notion that knowledge of the state, $\underline{x}(t_0)$, of the device at some time t_0, along with knowledge of the system inputs $\underline{u}(t)$ at times later than t_0, allows the determination of the states at all future times [Brogan, 1974]. (The inverse is also true, i.e., knowledge of $\underline{x}(t_0)$ and $\underline{x}(t_f)$, for $t_f > t_0$, allows determination of $\underline{u}(t)$ for $t_0 \le t \le t_f$.)

The PNN results will also be compared to the reduced-order analytic models of P. C. Krause, which neglect stator electrical transients. Although reduced-order models can be used to speed the simulation of synchronous machines, they have been shown to be conservative (and therefore inaccurate) estimators of system instability [Krause, 1986]. Although reduced-order models are sometimes adequate for transient stability studies of large-scale power systems, they may not be sufficiently accurate for low-power, small-system applications [Krause, 1986]. In addition, there are also questions concerning the relevance of reduced-order models in *distributed* power system applications which, for survivability reasons, are the intended configuration for future ship power distribution networks.

There are several approaches that may be taken in using static PNNs (SPNNs) for state prediction. (DPNN emulator models of the synchronous machine are also demonstrated below.) One could consider training an SPNN to predict future states of the machine, as shown in Fig. 4.13. In theory, under appropriate restrictive conditions, rather than forecasting a time increment τ into the future, higher accuracy may be achieved by forecasting only one step ahead, and then feeding the output estimate(s) back as input(s) for an appropriate number of iterations. Note, however, that this approach requires that the SPNN be trained initially with prior output *measurements* as inputs (equation error). The iterative use of output *estimates* (rather than output *measurements*) as inputs creates a strong potential for instability in the output estimates. The proper approach for state prediction using static networks is diagrammed in Fig. 4.14. Here multiple outputs of the SPNN are employed, each corresponding to a specific prediction horizon.

Note that feedback of output estimates is the essential property that distinguishes dynamic (recurrent) neural network synthesis from static synthesis; DPNNs are trained with the feedback connection already in place (output error), thereby helping to avoid the problem of having unstable output predictions.



$$\underline{x} = (i_q, i_d, i_{kq}, i_{fd}, i_{kd}, \omega_r, \theta_r)$$

$$\underline{u} = (T_i, e_{xfd}, v_{q\_d}, v_0)$$

Figure 4.13: Use of a Static PNN to Predict State Variables of a Synchronous Generator at a Future Time

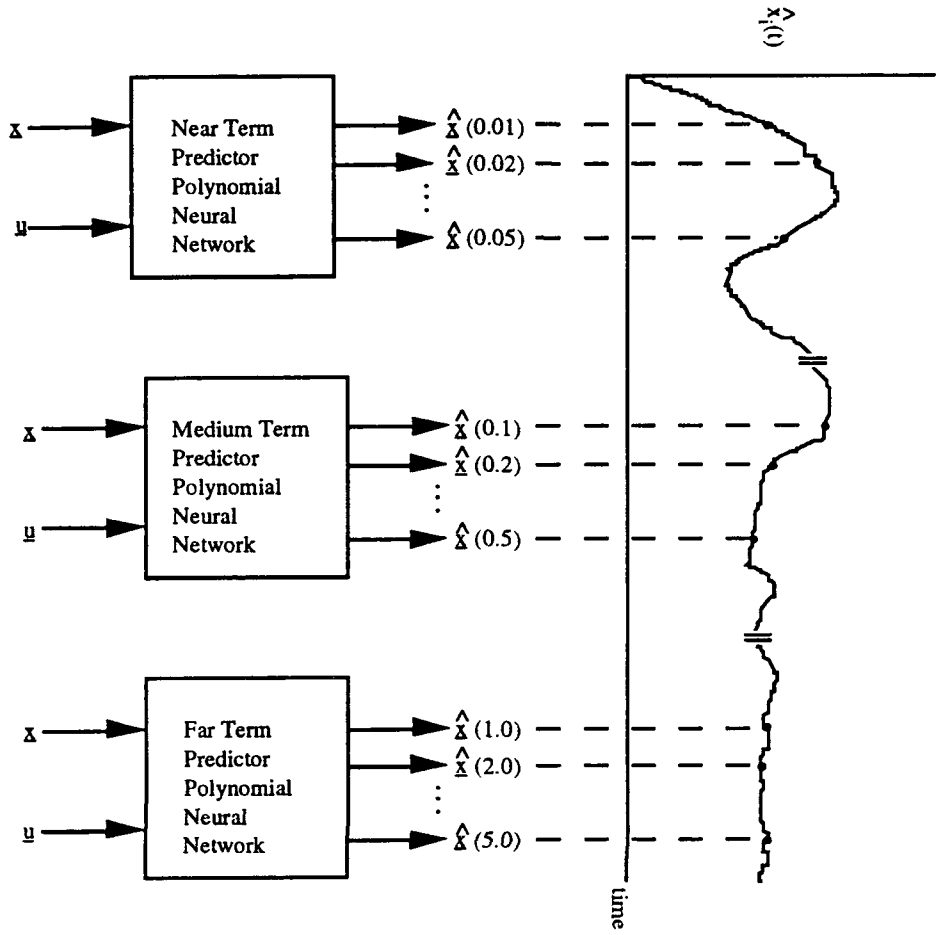


Figure 4.14: **Use of Multiple SPNNs to Predict the State Variables at Future Times**

4.6.1 SPNN Prediction of Synchronous Machine State Variable Outputs

To establish databases for synthesizing PNNs to emulate a synchronous machine, the coupled nonlinear differential equations that describe the machine were first established; these are given in Tables 4.3 – 4.5. Equation parameters for the hydro-turbine generator specified in Table 4.6 were used [Krause, 1986].

These equations were solved on a computer using a fourth-order Runge-Kutta numerical integration algorithm; a step size of ten microseconds was determined to be necessary to ensure numerical stability in all of the dynamic simulations that were conducted. Databases were generated representing the responses of the machine, acting as a synchronous generator, to the following inputs: (1) a step increase in input torque, with the machine connected to an infinite bus; (2) a three-phase short-circuit fault, with the machine connected to an infinite bus; and (3) a step change in load from an infinite bus to an RL load. The database included a single time series of each of these event scenarios. An SPNN was then used to model this database, as shown in Fig. 4.13. The state variables included the currents, $i_q(t)$, $i_d(t)$, $i_{kq}(t)$, $i_{fd}(t)$, $i_{kd}(t)$, the rotor electrical angular velocity, $\omega_r(t)$, and the electrical rotor position, $\theta(t)$, all of which were transformed to the rotor reference frame using Park's transformation. The inputs included the input torque, $T_i(t)$, the external field voltage, e_{xfd} (which was held constant for a given scenario), and the Park-transformed voltages, $v_q(t)$, $v_d(t)$, $v_0(t)$, which define the load on the synchronous generator.

Table 4.3: Nonlinear Coupled Flux Linkage Equations for Synchronous Machine [Krause, 1986]

$$\psi_q = \frac{\omega_b}{p}\left[v_q - \frac{\omega_r}{\omega_b}\psi_d + \frac{r_s}{X_{ls}}(\psi_{mq} - \psi_q)\right]$$

$$\psi_d = \frac{\omega_b}{p}\left[v_d + \frac{\omega_r}{\omega_b}\psi_q + \frac{r_s}{X_{ls}}(\psi_{md} - \psi_d)\right]$$

$$\psi_0 = \frac{\omega_b}{p}\left[v_0 - \frac{r_s}{X_{ls}}\psi_0\right]$$

$$\psi_{kq} = \frac{\omega_b}{p}\left[v_{kq} + \frac{r_{kq}}{X_{lkq}}(\psi_{mq} - \psi_{kq})\right]$$

$$\psi_{fd} = \frac{\omega_b}{p}\left[\frac{r_{fd}}{X_{md}}e_{xfd} + \frac{r_{fd}}{X_{lfd}}(\psi_{md} - \psi_{fd})\right]$$

$$\psi_{kd} = \frac{\omega_b}{p}\left[v_{kd} + \frac{r_{kd}}{X_{lkd}}(\psi_{md} - \psi_{kd})\right]$$

where

$$\psi_{mq} = X_{aq}\left(\frac{\psi_q}{X_{ls}} + \frac{\psi_{kq}}{X_{lk!}}\right)$$

$$\psi_{md} = X_{ad}\left(\frac{\psi_d}{X_{ls}} + \frac{\psi_{fd}}{X_{lfd}} + \frac{\psi_{kd}}{X_{lkd}}\right)$$

$$X_{aq} = \left(\frac{1}{X_{mq}} + \frac{1}{X_{ls}} + \frac{1}{X_{lkq}}\right)^{-1}$$

$$X_{ad} = \left(\frac{1}{X_{md}} + \frac{1}{X_{ls}} + \frac{1}{X_{lfd}} + \frac{1}{X_{lkd}}\right)^{-1}$$

Table 4.4: Current Equations for Synchronous Machine [Krause, 1986]

$$i_q = \frac{1}{X_{ls}} (\psi_q - \psi_{mq})$$

$$i_d = \frac{1}{X_{ls}} (\psi_d - \psi_{md})$$

$$i_0 = \frac{1}{X_{ls}} \psi_0$$

$$i_{kq} = \frac{1}{X_{lkq}} (\psi_{kq} - \psi_{mq})$$

$$i_{fd} = \frac{1}{X_{lfd}} (\psi_{fd} - \psi_{md})$$

$$i_{kd} = \frac{1}{X_{lkd}} (\psi_{kd} - \psi_{md})$$

Table 4.5: Definitions and Other Relationships

$p \equiv \frac{d}{dt}$

$\omega_r \equiv$ the electrical angular velocity

$\omega_b \equiv$ the base electrical angular velocity

$X \equiv$ reactances

$v \equiv$ voltages

$i \equiv$ currents

$\psi \equiv$ flux linkages per second

$r \equiv$ resistances

Torque Equation

$T_e = \psi_d i_q - \psi_q i_d$

$T_e \equiv$ the output electromagnetic torque

Rotor Speed Equation

$T_i - T_e = J\left(\frac{2}{P}\right) p\, \omega_r$

$J \equiv$ moment of inertia,

$P \equiv$ number of poles

$T_i \equiv$ the input torque

Table 4.6: Synchronous Machine (Hydro-Turbine Generator) Parameters Used in Simulations [Krause, 1986]

Rating: 325 MVA

Line-to-line voltage: 20 kV Power factor: 0.85

Poles: 64 ω_m = 112.5 r/min

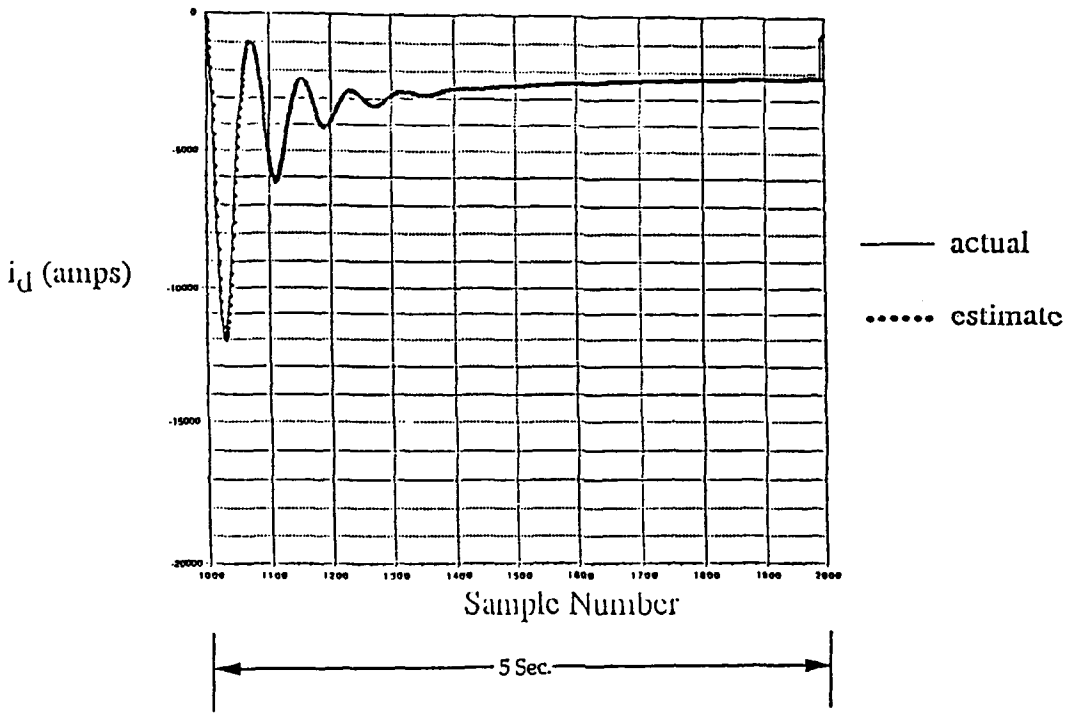
Combined inertia of generator and turbine:

$J = 35.1 \times 10^6$ J · sec^2 or $WR^2 = 833.1 \times 10^6$ lbm · ft^2 H = 7.5 sec

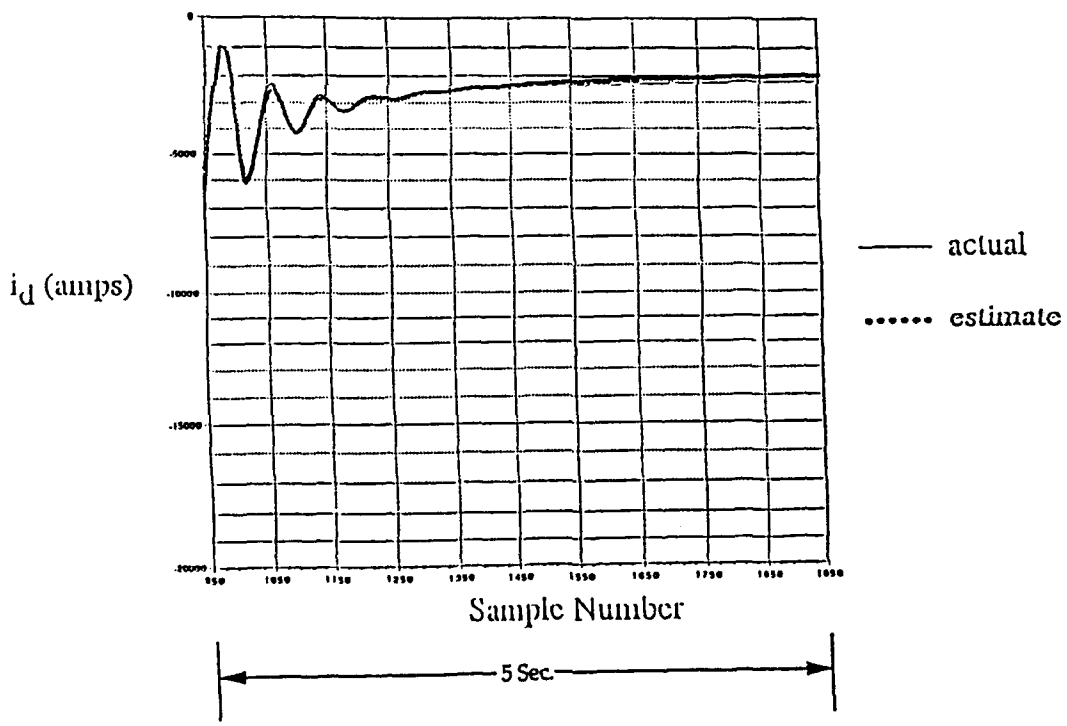
Parameters in ohms and per unit:

r_s	= 0.00234 Ω, 0.0019 pu	X_{ls}	= 0.1478 Ω, 0.120 pu
X_q	= 0.5911 Ω, 0.480 pu	X_d	= 1.0467 Ω, 0.850 pu
r_{fd}	= 0.00050 Ω, 0.00041 pu	X_{lfd}	= 0.2523 Ω, 0.2049 pu
r_{kq}	= 0.01675 Ω, 0.0136 pu	X_{kd}	= 0.01736 Ω, 0.0141 pu
X_{lkq}	= 0.1267 Ω, 0.1029 pu	X_{lkd}	= 0.1970 Ω, 0.160 pu

An SPNN was trained to predict the output of various state variables and other system variables in both the near- and far-term, using only the system state variables and inputs, as shown in Fig. 4.14. Typical results are given in Figs. 4.15 – 4.16. These figures demonstrate that even with an extremely limited database, static PNNs can be trained to model a synchronous machine both accurately and rapidly; accuracy depends on the extent and variety of the training database. The output of the PNN is essentially instantaneous; it may be used as a fast, accurate substitute for numerical propagation of the detailed nonlinear differential equations.

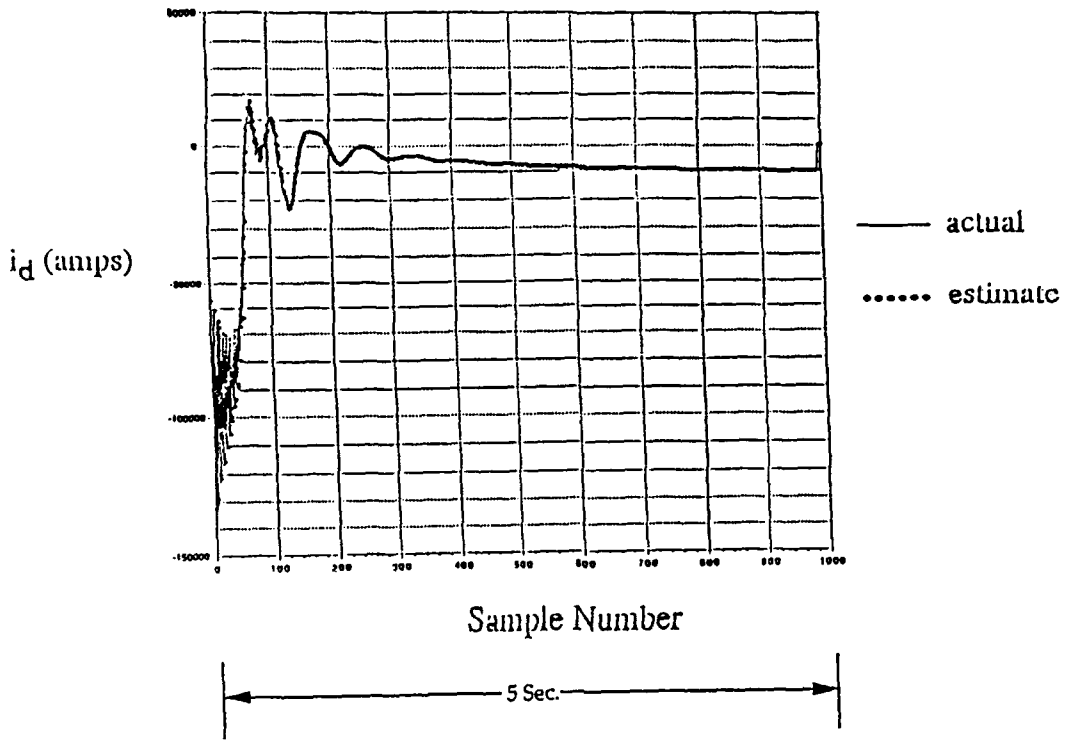


(a) PNN Prediction with 0.05 Sec. Forecast Horizon

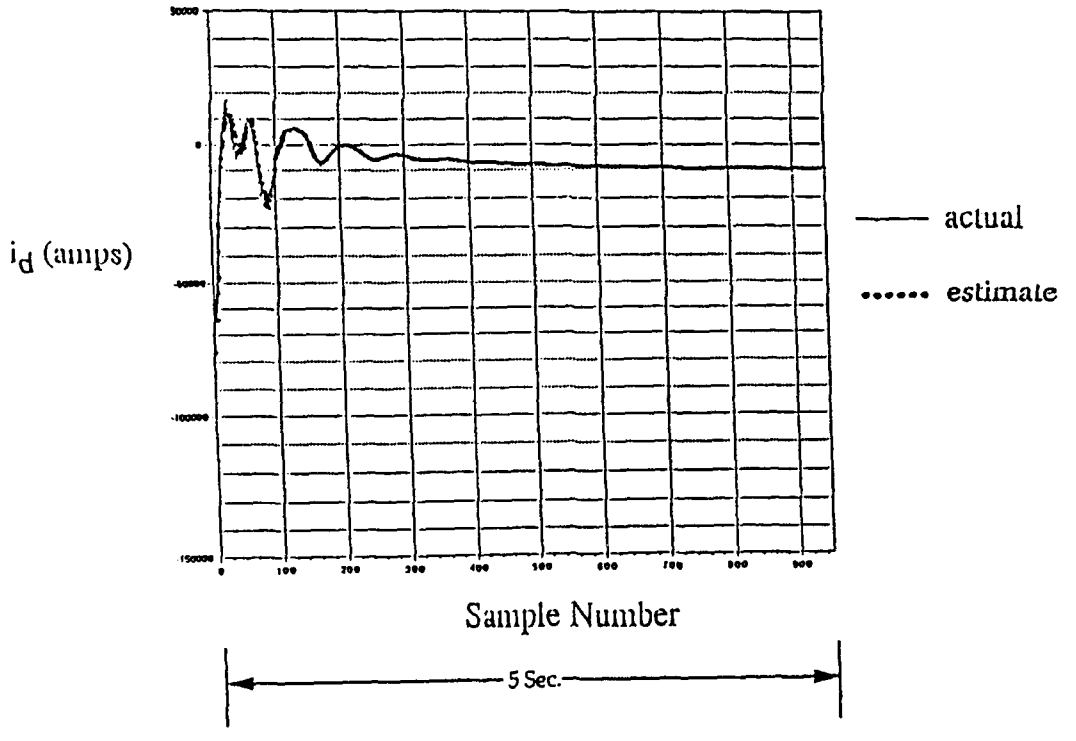


(b) PNN Prediction with 0.5 Sec. Forecast Horizon

Figure 4.15: Polynomial Neural Network Model of Dynamic Response of Hydro-Turbine Generator Connected to Infinite Bus During Step Increase of Input Torque



(a) PNN Prediction with 0.05 Sec. Forecast Horizon

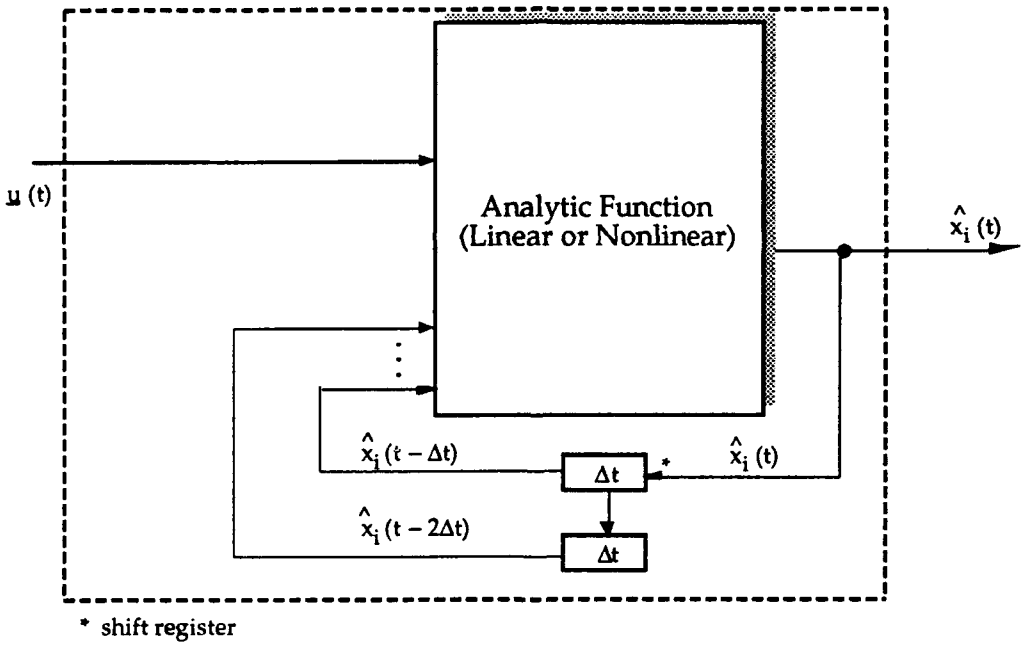


(b) PNN Prediction with 0.5 Sec. Forecast Horizon

Figure 4.16: **Polynomial Neural Network Model of Dynamic Response of Hydro-Turbine Generator Connected to Infinite Bus During Short Circuit Fault at Terminals**

4.6.2 DPNN Prediction of Synchronous Machine State Variable Outputs

Dynamic PNNs (DPNNs) were next investigated to test their ability to emulate a synchronous machine. As shown in Fig. 4.17, separate *linear* DPNNs were trained to model each of the state variables of the hydro-turbine generator. The generator was assumed to be connected to an infinite bus. The state variables included the currents, $i_q(t)$, $i_d(t)$, $i_{kq}(t)$, $i_{fd}(t)$, $i_{kd}(t)$, the rotor electrical angular velocity, $\omega_r(t)$, and the electrical rotor position $\delta(t)$. In each DPNN model, the state variable *estimate* being modeled was fed back as an input through time delays (also shown in Fig. 4.17). (Note that when *measurements* rather than *estimates* are fed back, essentially perfect modeling accuracy can be achieved; this accuracy, however, comes at the expense of: (1) the requirement for sensors to provide such measurements and (2) a loss of the capability to forecast future states of the machine in response to arbitrary inputs, because intermediate measurements are not available.) Inputs to the machine included the stator voltages, $v_q(t)$, $v_d(t)$, and the input torque, $T_i(t)$, of



$$\underline{x}^T = [i_q, i_d, i_{kq}, i_{fd}, i_{kd}, \omega_r, \delta]$$

$$\underline{u}^T = [T_i, v_q, v_d]$$

Figure 4.17: Linear Dynamic Polynomial Neural Network Used to Model Each State Variable

which only $T_i(t)$ was varied arbitrarily. (Balanced conditions were assumed so that results using DPNN models could be compared to the reduced-order models of P.C. Krause where unbalanced conditions are ignored, as well as to the full nonlinear differential equation model of the synchronous machine.) To train DPNNs to model the *general* characteristics of a synchronous machine, broadband uniform

random white noise was used to simulate $T_i(t)$. Specifically, $T_i(t)$ was randomly varied over the rated operating range of the machine, zero to 27.6×10^6 N*m, using frequency components from dc to approximately 100 Hz,[5] while recording the state variable outputs of the full differential equation model.

Figs. 4.18 – 4.20 illustrate the state variable responses of the hydro-turbine generator to a step in input torque of 23.5×10^6 N*m (0.85 rated). Fig. 4.18 is the "truth" or full differential equation model, which captures accurately all of the dynamics of the machine. This is the response that one would expect to obtain in actual use of the machine. Fig. 4.19 depicts the response of the reduced-order model of Krause, in which the stator electrical transients are neglected. This model provides for more rapid computation of the state variables than the full differential equation model, at the expense, however, of accuracy; specifically, the offset transients and 60 Hz oscillations seen in Fig. 4.18 are not present in Fig. 4.19. One important consequence of this is imprecision in determining the critical clearing time of faults with use of the reduced-order model. In the case of three-phase short-circuit faults, the critical clearing time is the maximum time duration that the fault may remain in effect such that, when the fault is cleared, the machine will still return to synchronous operation (i.e., remain stable). The reduced-order models tend to underestimate critical clearing times [Krause, 1986].

Fig. 4.20 illustrates the step response of the DPNN synchronous machine emulator which, as noted above, was trained on uniform random white noise. Note that all state variables are modeled reasonably accurately. No attempt was made to improve these results, either through the inclusion of nonlinearities in the estimation models or additional time-delayed input or feedback variables, or by increasing the duration of the training sequence. In addition, whereas generation of the machine state-variable step response using the full differential equation model is a relatively slow process due to the small step sizes necessitated by numerical integration, the DPNN output is provided much more rapidly (by several orders of magnitude). For one-step-ahead prediction with the DPNN, each iteration requires the computation of only a few dot products, which occurs virtually instantaneously. Prediction of state variables at arbitrary time steps into the future is accomplished by iterative feedback of the output estimates and by setting the input variables to their respective values. This is also an extremely rapid process.

The power of the DPNN approach, however, lies not only in its superior speed relative to the full differential equation model, and its higher accuracy and speed relative to the reduced-order model; but also in the ability to synthesize linear

---

[5] A second-order Butterworth low-pass filter was used to color the white noise samples, such that the passband was from dc to 100 Hz, the filter output was down 3 dB at 50 Hz, and down 20 dB beyond 100 Hz.

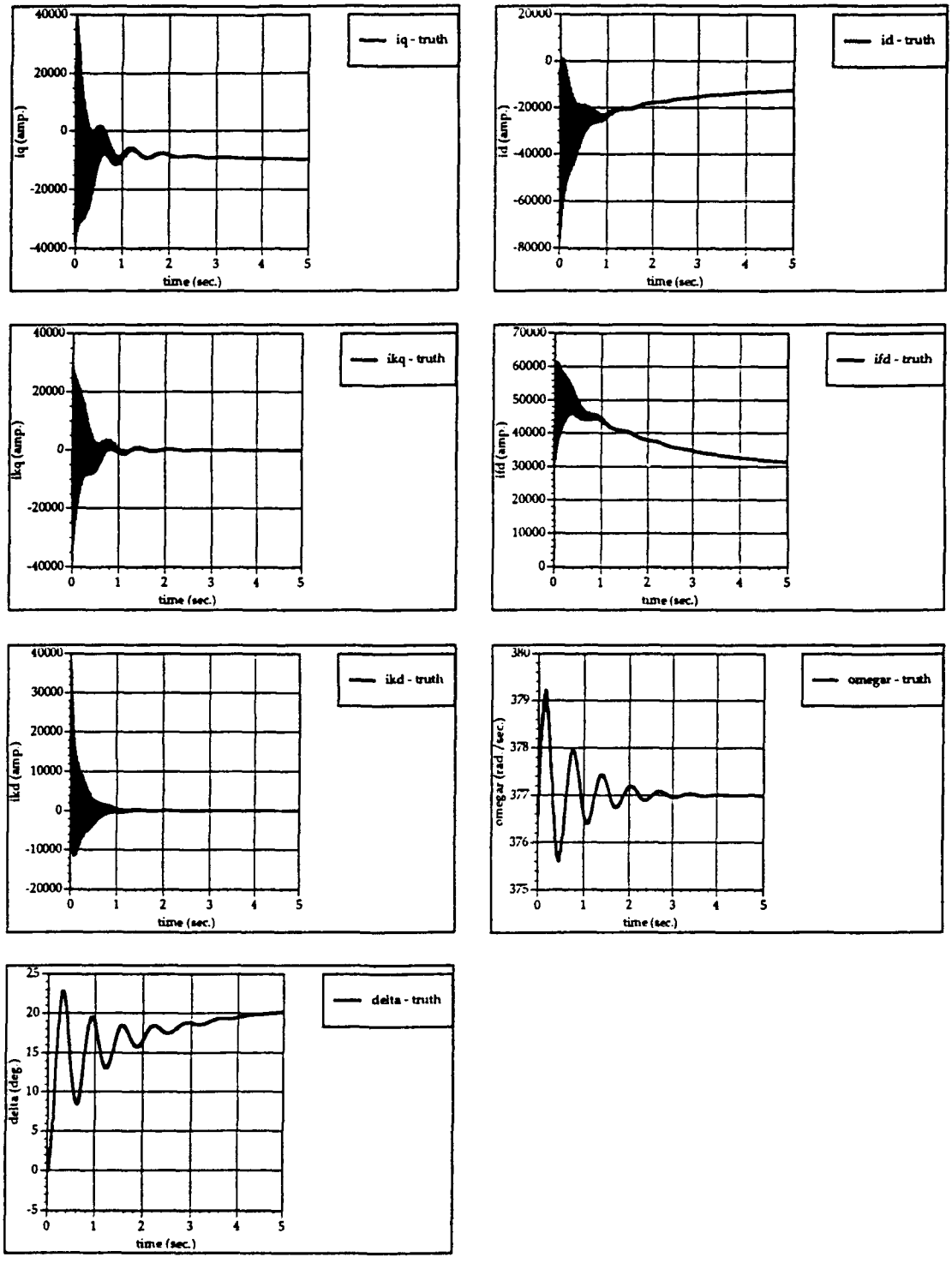


Figure 4.18: "Truth" or Full Differential Equation Model Response to a Step in Input Torque of 23.5×10^6 N*m (0.85 Rated)

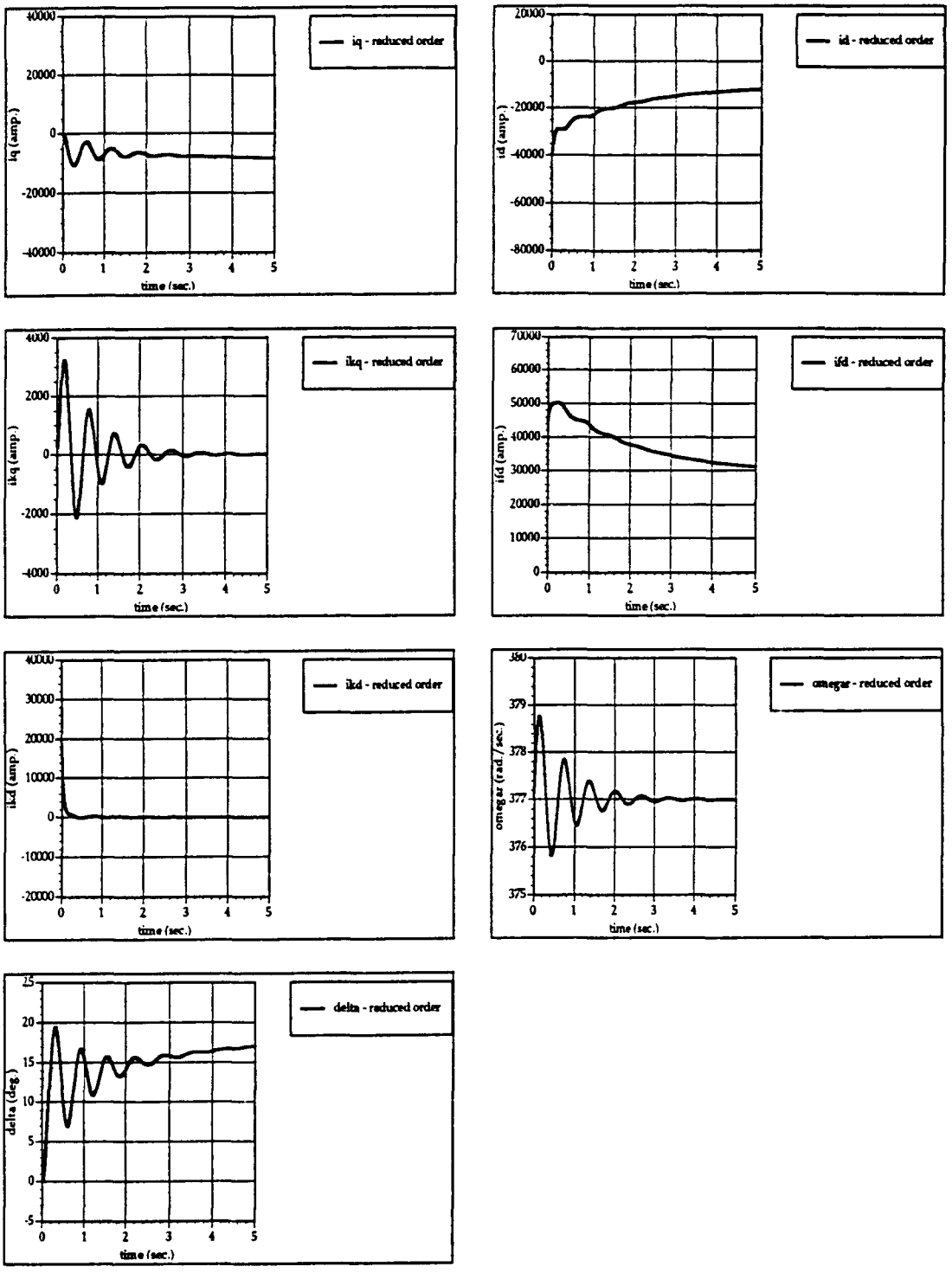


Figure 4.19: Reduced-Order (Stator Electric Transients are Neglected) Differential Equation Model Response to a Step in Input Torque of 23.5×10^6 N*m (0.85 Rated)

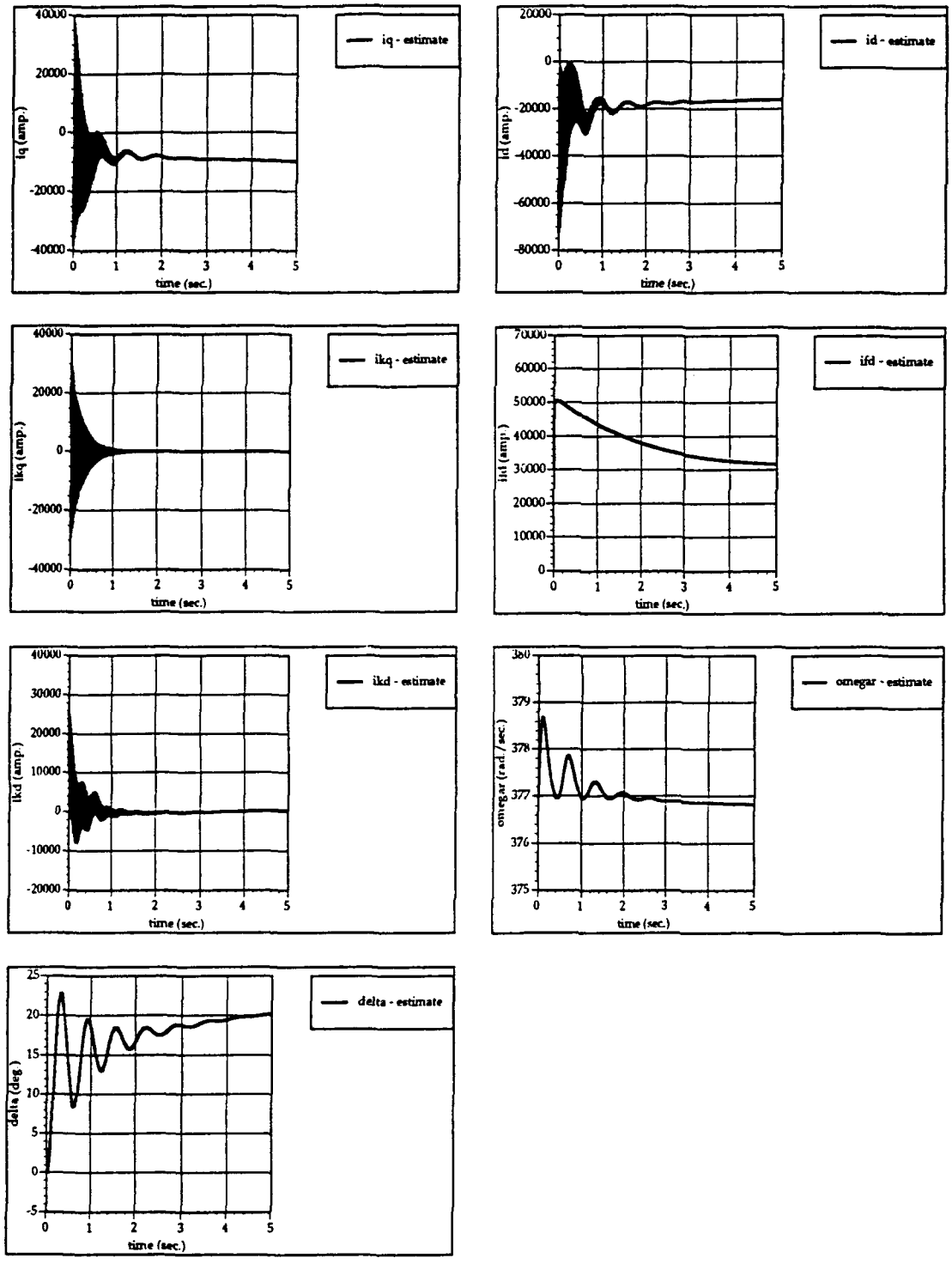


Figure 4.20: "Estimate" or DPNN Synchronous Machine Emulator Response to a Step in Input Torque of 23.5×10^6 N*m (0.85 Rated)

and nonlinear prediction models inductively, based on process data collected off- or on-line, either in laboratory experiments or in application settings. This allows power distribution system nodal elements, such as synchronous machines, to be characterized without substantial *a priori* analytic modeling effort; then, using the same process, these models can be adapted on-line to account for changes in, or subtle differences between, individual machines. Such models may also be used directly in fault detection, isolation, and estimation (FDIE) applications by monitoring parameter values, or equivalently, model prediction-error residuals. Reconfiguration can be performed, as necessary, subsequent to FDIE.

As mentioned earlier, the DPNN models used above to emulate the synchronous machine were all linear; the inputs and delayed feedback values were simply weighted by their corresponding model parameters and then summed. Arbitrarily high accuracy may be achievable with such models by including nonlinear terms, and possibly additional time-delayed input variables. Further accuracy may also be achieved by using longer random noise training sequences. In their present configuration, the DPNN models emulate accurately only the stable operation of the synchronous machine, which is the operational regime on which they were trained.

Because the DPNN models above were linear, they do not exhibit a critical holding time for three-phase short-circuit faults. Simultaneously handling stable *and* unstable responses requires that nonlinearity be included in the models, as only wh nonlinearity is present can substantially different responses occur for small changes in the input conditions. The capability of DPNNs to model instability can be demonstrated using, for example, the instability induced by a small increase in the holding time of a three-phase short-circuit fault, increasing it to a duration greater than the critical holding time.

Below, results are given illustrating the capability of DPNNs to model this nonlinear response. Specifically, in these experiments, a DPNN model having linear and nonlinear polynomial terms up to degree two was used. With two time-delayed output feedback terms being used, this resulted in a model with 21 coefficients. The network, whose output represents the electrical rotor position state variable, $\delta(t)$, was trained assuming an infinite bus and using three eight-second training-data sequences; these data included: (1) random input torque (T_i) excitation, having an average value of 0.85 rated; (2) a five-second step response (0.85 rated), followed by a three-phase short-circuit fault that lasted 0.465 second before being cleared; and (3) a five-second step response (0.85 rated), followed by a three-phase short-circuit fault lasting 0.467 second before being cleared. (For the synchronous machine under study, the critical holding time at 0.85 rated input torque is 0.466 second; three-phase short-circuit faults lasting longer than 0.466 second will result in unstable responses. Unstable responses are defined by electrical rotor positions, $\delta(t)$, exceeding 360 degrees.)

Fig. 4.21 illustrates the "true," $\delta(t)$, and "estimated," $\hat{\delta}(t)$, responses to a 0.85 rated step-input torque, which is interrupted by a short-circuit fault at t = 5 second; the short-circuit fault is held 0.465 second before being cleared. This figure represents a *stable* short-circuit response, and the DPNN models accurately both the qualitative and quantitative aspects of the response. Fig. 4.22 shows the results of a similar experiment, where the short-circuit fault is instead held for 0.467 second before being cleared; this evokes an unstable machine response, where the "true" $\delta(t)$ is seen to increase dramatically, eventually exceeding 360 degrees. Here the DPNN model is seen initially to track the $\delta(t)$ response until the DPNN output begins to oscillate and eventually saturate (this can better be seen in Fig. 4.23); the DPNN model thus provides only qualitative evidence of the *unstable* response. For unstable responses, however, it is probably not important to have high quantitative accuracy, because one is interested in forecasting future states of the machine for the purpose of taking immediate action. If predicted future states are unstable, one would usually be interested in avoiding such behavior, rather than in quantifying it to a high degree of accuracy. We see in these examples that the DPNN is able to define very accurately the critical holding time, which is not true for reduced-order models, which consistently underestimate the critical holding time, providing an estimate of 0.424 second [Krause, 1986].

We note that these results are preliminary and are based on the use of a very limited amount of training data. Additional quantitative accuracy may be possible through the inclusion of other nonlinear terms, additional time-delayed input and output variables, and longer training-data sequences. Although the DPNN model was evaluated using a portion of the training data, these results are mainly intended to demonstrate the capability of a nonlinear DPNN to capture simultaneously the stable quantitative and unstable qualitative behavior of the synchronous machine.

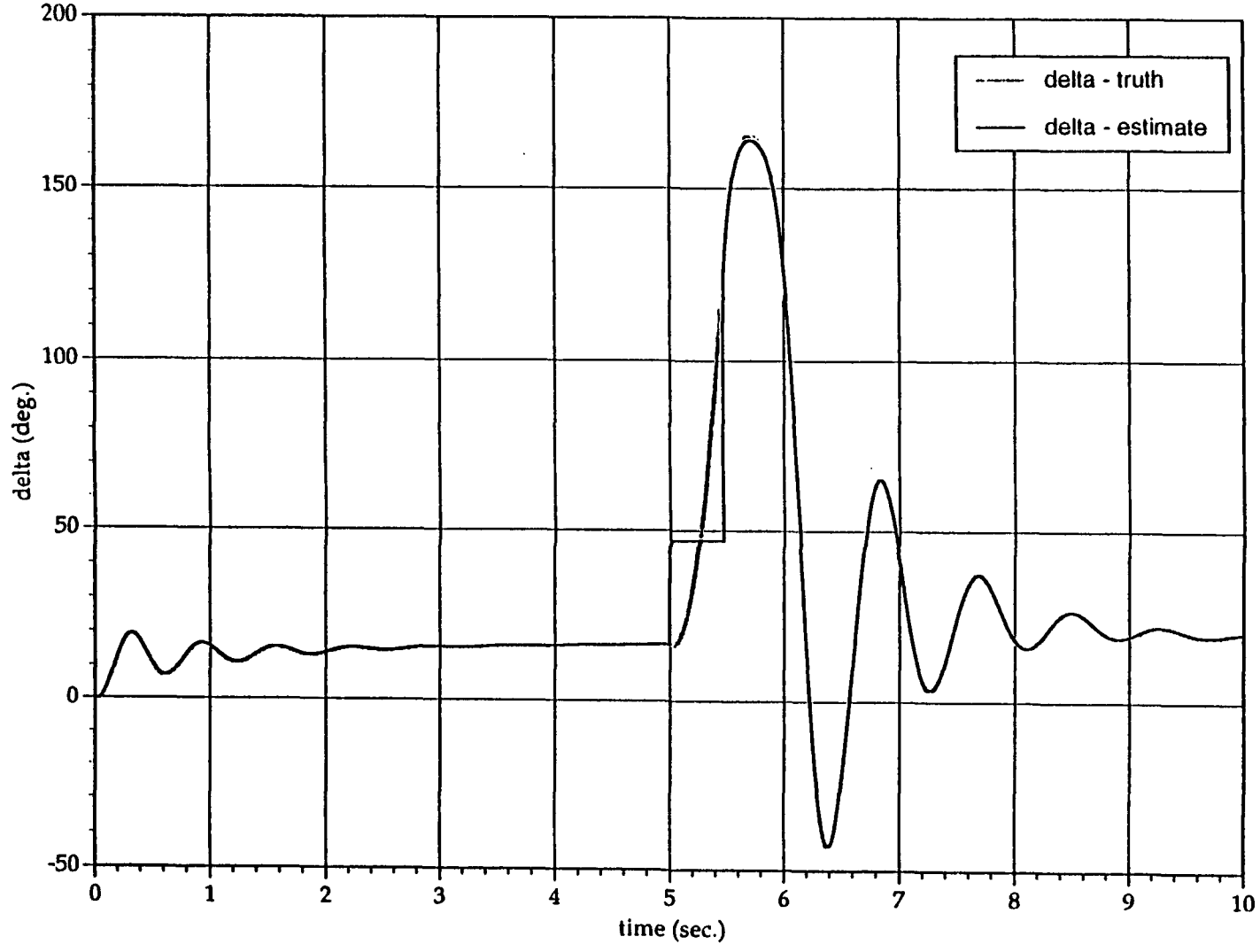


Figure 4.21: **Actual and Estimated Stable Responses of δ(t) to a Step in Input Torque, Followed by Introduction of a Three-Phase Short-Circuit Fault at 5 Sec., which Lasts 0.465 Sec.**

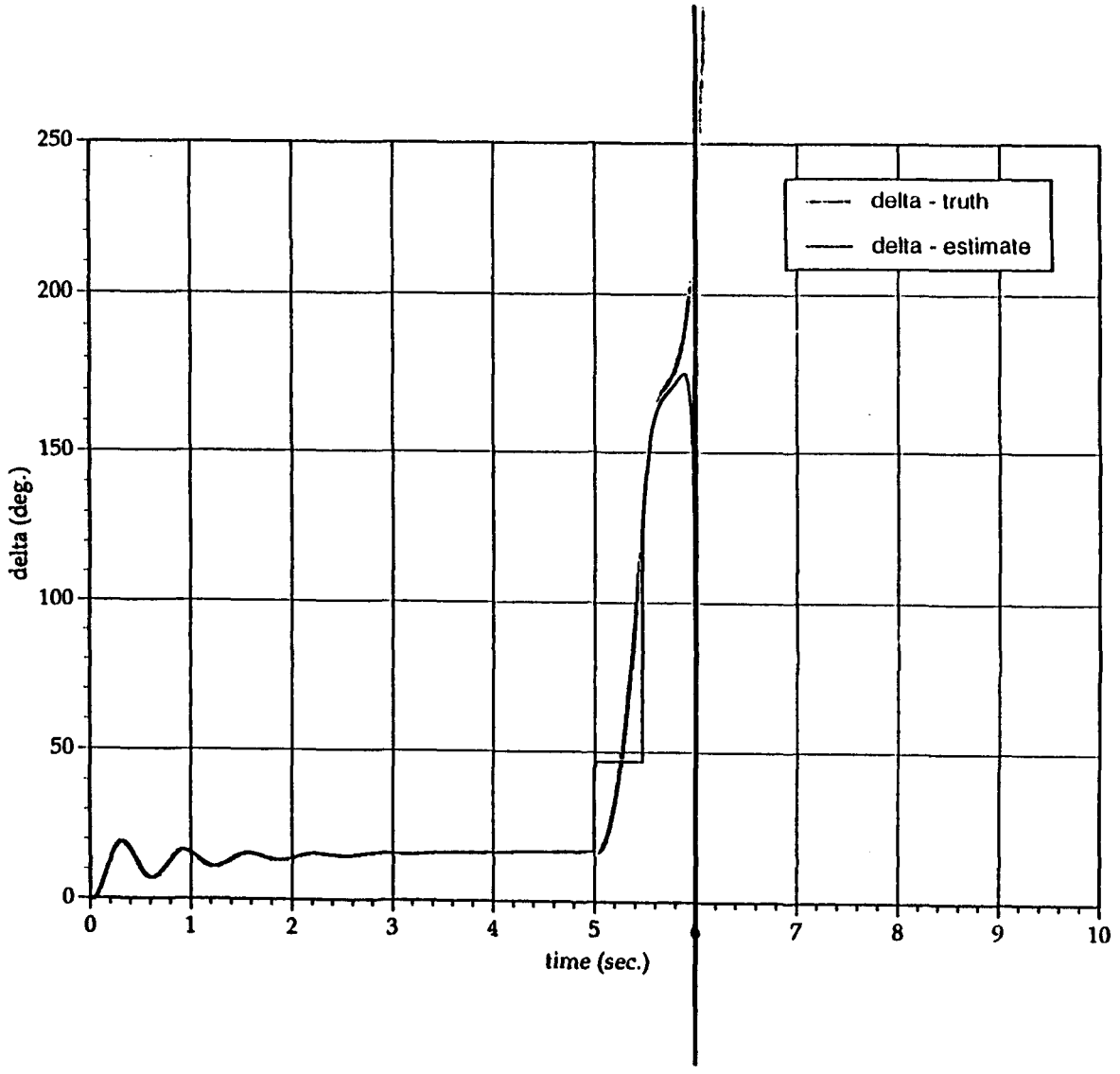


Figure 4.22: **Actual and Estimated Unstable Responses of $\delta(t)$ to a Step in Input Torque, Followed by Introduction of a 3-Phase Short-Circuit Fault at 5 Sec., which Lasts 0.467 Sec.**

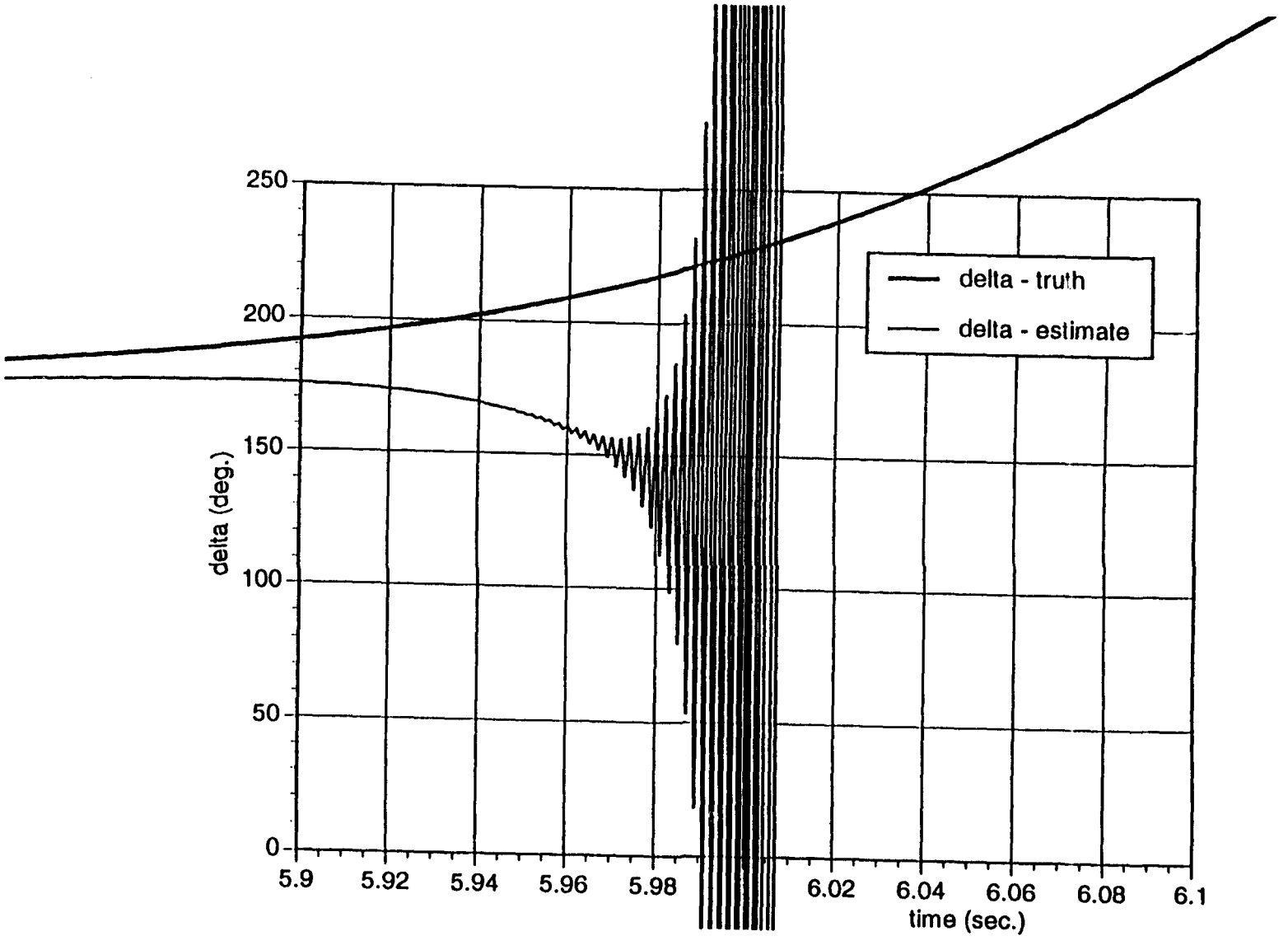


Figure 4.23: **Actual and Estimated Unstable Responses of δ(t) to a Step in Input Torque, Followed by Introduction of a 3-Phase Short-Circuit Fault at 5 Sec., which Lasts 0.467 Sec. (Same as Fig. 4.22, but Scale has been Expanded)**

5. CONCLUSIONS AND RECOMMENDATIONS

For both estimation and classification problems, the benefits of using artificial neural networks include inductive learning, rapid computation, and the ability to handle high-order and/or nonlinear processing. Neural networks reduce the need for simplifying assumptions that use *a priori* statistical models (such as "additive Gaussian noise") or that neglect nonlinear terms, cross-coupling effects, high-order dynamics, and so forth. Neural networks can provide accurate long-term predictions and can also give inverse solutions for complex dynamics, such as the values of initializing Lagrange multipliers for on-line, optimum, real-time, two-point boundary-value guidance of flight vehicles. Moreover, through on-line learning, neural networks can improve upon the performance achievable with prior knowledge.

This report presents the foundations for a statistical-modeling perspective on the inductive synthesis of static and dynamic (recurrent) artificial neural networks. (Dynamic ANNs incorporate internal feedbacks and time delays.) Network nodal element basis functions and output transformations are discussed. Information-theoretic synthesis criteria for estimation and classification networks are introduced. Algorithms are described for evolving the networks from their simplest possible forms to just-sufficient levels of complexity, with or without internal feedbacks and time delays, including algorithms for optimization at the node, layer, and network levels. For static sigmoidal networks, the statistical bounds on approximation and estimation errors are presented.

Applications of static and dynamic polynomial neural networks are detailed and compared in this report for several estimation, prediction, and control problems:

(1) prediction of behavior, both well-behaved and chaotic, of May's deterministic nonlinear difference equation for population sequences (Appendix C);

(2) prediction of photodiode outputs (indicative of combustion quality) in response to changes in amplitude modulation levels in an acoustically-driven turbopropulsion combustion process (Section 4 and Appendix E). This study includes a comparison with classical Wiener-Volterra techniques;

(3) two-point boundary-value guidance of a tactical air-to-air missile (Appendix D). Accuracy, simplicity, robustness, and noise sensitivity of static and dynamic neural networks are compared; and

(4) high-speed predictions of hydro-turbine generator responses to input and load changes, including estimation of clearing time (before instability) for an output short-circuit fault.

Other concurrent work, not reported herein, has also successfully applied dynamic polynomial neural networks in structural acoustics [Parker, Poor, et al.; 1991]; has made and is making extensive applications of neural networks to detection and classification of acoustic transients in undersea warfare (DARPA *DANTES* program), with fast ANN updating in the at-sea environment [Barron, Parker, Ward, Abbott, and Jordan; to be reported in 1993]; and is in-progress on several complex multivariable control and diagnostic problems defined by the U.S. Navy.

Increasingly, the focus in neural network applications is on meeting requirements that are best satisfied using dynamic ANNs. In summary, dynamic networks offer important advantages in that they:

- require, in general, fewer internal degrees of freedom, resulting in more robust networks;
- readily process time-varying inputs (prior measures of the input vectors are retained automatically where needed);
- may reduce the need for intensive feature-extraction preprocessing or may provide the means for feature computation;
- as with static networks, are well-suited to on-line adaptation using recursive optimization algorithms;
- readily constitute infinite impulse response (IIR) filters in addition to providing finite impulse response (FIR) filters; and
- as with static networks, can use information-theoretic modeling criteria to build a network structure from zero connections to just-sufficient complexity.

A prototype synthesis algorithm, *DynNet*, has been created for *dynamic estimation* polynomial neural networks. It complements existing, mature algorithms (*ASPN* and *CLASS*) intended for syntheses of static estimation and classification networks.

Further tool development is needed to:

(1) obtain *dynamic classification* ANN synthesis capabilities. This will be useful for automatically capturing spatio-temporal information in classifiers (including detection, discrimination, and diagnostic systems), thereby possibly avoiding the need for time-frequency or time-space front-end transformations. Either time-variables or time-varying features could be used as inputs;

(2) refine the static classification ANN synthesis algorithm to incorporate structure learning (as in *ASPN*);

(3) refine all of the static and dynamic synthesis algorithms to take advantage of the projection-pursuit protocol [Friedman and Tukey, 1974; Friedman and Stuetzle, 1981] as discussed in Section 2;

(4) produce an integrated synthesis tool for off-line static and dynamic ANNs for estimation and classification;

(5) provide on-line mechanizations of the above synthesis algorithms (including structure learning) and provide off-line and on-line mechanizations of counterpart adaptation algorithms (without structure learning).

6. REFERENCES

Akaike, H., "Information theory and an extension of the maximum likelihood principle," *Proc. Second Int'l. Symp. on Information Theory*, B.N. Petrov and F. Csaki (Eds.), Akadémiai Kiadó Budapest, pp. 267 -281, 1972.

Antsaklis, P.J., "Neural networks in control systems," *IEEE Control Systems Magazine*, Vol. 12, No. 2, April 1992.

Barron, A.R., *Properties of the Predicted Squared Error: A Criterion for Selecting Variables, Ranking Models, and Determining Order*, Adaptronics, Inc., McLean, Virginia, 1981.

Barron, A.R., "Predicted Squared Error: A Criterion for Automatic Model Selection," *Self-Organizing Methods in Modeling: GMDH Type Algorithms* (S.J. Farlow, Ed.), Marcel Dekker, Inc., New York, Chap. 4, pp. 87-103, 1984.

Barron, A.R., *Logically Smooth Density Estimation*, Ph.D. Dissertation, Dept. of E.E., Stanford University, Palo Alto, California, 1985.

Barron, A.R., "Statistical properties of artificial neural networks," *Proc. IEEE 1989 Conf. on Decision and Control*, Tampa, Florida, December 13-15, 1989.

Barron, A.R., "Complexity regularization with applications to artificial neural networks," *Proc. NATO ASI on Nonparametric Functional Estimation*, Spetses, Greece, G. Roussas, Ed., Kluwer Academic Publishers, Dordrecht, Netherlands, August 1-10, 1990.

Barron, A.R., "Approximation and estimation bounds for artificial neural networks," *Computational Learning Theory: Proc. of 4th Ann. Workshop*, Morgan Kaufman, 1991.

Barron, A.R., "Universal approximation bounds for superpositions of a sigmoidal function," Submitted to *IEEE Trans. on Information Theory*, 1991.

Barron, A.R. and R.L. Barron, "Statistical learning networks: A unifying view," *Proc. 20th Symposium on the Interface: Computing Science and Statistics*, Reston, Virginia, April 1988.

Barron, A.R. and X. Xiao, "Discussion on multivariate adaptive regression," *Annals of Statistics* , Vol 19, No. 1, pp. 67-82, 1991.

Barron, R.L., "Adaptive transformation networks for modeling, prediction, and control," *Proc. Joint Nat'l. Conf. on Major Systems*, IEEE/ORSA, October 25-26, 1971.

Barron, R.L., "Guided accelerated random search as applied to adaptive array AMTI radar," *Proc. Adaptive Antenna Systems Workshop*, Naval Research Laboratory, Washington DC, 11-13 March 11-13 1974.

Barron, R.L., "Theory and application of cybernetic systems: An overview," *Proc. 1974 NAECON*, pp. 107-118, May 1974.

Barron, R.L., "Learning networks improve computer-aided prediction and control," *Computer Design*, August 1975.

Barron, R.L., A.N. Mucciardi, F.J. Cook, J.N. Craig, and A.R. Barron, "Adaptive Learning Networks: Development and Application in the United States of Algorithms Related to GMDH," *Self-Organizing Methods in Modeling: GMDH Type Algorithms* (S.J. Farlow, Ed.), Marcel Dekker, Inc., New York, Chap. 2, pp. 25-65, 1984.

Barron, R.L. and D.W. Abbott, "Use of polynomial networks in optimum real-time, two-point boundary-value guidance of tactical weapons," *Proc. 1988 Military Computing Conference*, May 3-5, 1988.

Barron, R.L., D.W. Abbott, et al., *Trajectory Optimization and Optimum Path-to-Go Guidance of Tactical Weapons: Vol. I – Theory and AIWS Application*, August, 1988; *Vol. II – Closed-Loop OPTG Guidance of Mk 82 Glide Weapon*, September 1987; *Vol. III – Open-Loop Trajectory Optimization of Skipper Boost-Glide Weapon*, June 1988; *Vol. IV – Calculation of Lagrange Multipliers of Vertical-Plane Maximum-Range Trajectories of 11:1 L/D Boost-Glide AIWS*, January 1989, Barron Associates, Inc. Final Technical Report for HR Textron Inc. under U.S. Naval Weapons Center contract N60530-88-C-0036.

Barron, R.L., R.L. Cellucci, P.R. Jordan, III, N.E. Beam, P. Hess, and A.R. Barron, "Applications of polynomial neural networks to FDIE and reconfigurable flight control," *Proc. 1990 NAECON*, May 1990.

Barron, R.L. and R.L. Cellucci, *Optimum Management of KEW Divert Energy*, Barron Associates, Inc. Final Rept. for SDIO and Air Force Armament Laboratory, Contract F08635-90-C-0383, January 24, 1991.

Barron, R.L., D.W. Abbott, R.L. Cellucci, and D.E. Blackman, *Neural Network Flight Control System Design for High-Agility Air Combat*, Barron Associates, Inc., Final Report for Air Force Wright Laboratories, Contract F33615-90-C-3614, April 15, 1991.

Barron, R.L., P. Hess, P.R. Jordan, III, and C.M. Hawes, "*Diagnostic Abductive and Inductive Reasoning for Flight Control, Effector FDIE and Reconfiguration, Part I: Synthesis Algorithms and Results for Pre-Trained FDIE, Part II: Recursive Estimation of Aircraft Parameters Using Neural Network Calculation of Impairment Probability,*" Barron Associates, Inc. Final Report for Flight Dynamics Directorate, Wright Laboratory (AFSC), under Contract F33615-88-C-3615, WL-TR-91-3108, March 1992.

Breiman, L. and J.H. Friedman, "Estimating optimal transformations for multiple regression and correlation," *J. Amer. Statist. Assoc.*, Vol. 80, pp. 580-619, 1985.

Brogan, W.L., *Modern Control Theory*, Quantum Publishers: New York, 1974.

Daubechies, I., *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics, Capital City Press, Montpelier, Vermont, 1992.

Devroye, L., *A Bibliography on Random Search*, Technical Report SOCS-79.9, McGill University, May 1979.

Efron, B., *The Perceptron Correction Procedure in Nonseparable Situations*, Rome Air Dev. Center Tech. Documentary Report, RADC-TDR-63-533, February 1964.

Elder, J.F. IV and R.L. Barron, "Automated design of continuously-adaptive control: The 'super-controller' strategy for reconfigurable systems," *Proc. 1988 American Control Conference*, June 15-17, 1988.

Farley, B.G. and W.A. Clark, "Simulation of self-organizing systems by digital computers," *IRE Trans. on Inform. Theory*, Vol. PGIT-4, pp. 76-84, 1954.

Fisher, R.A., "The use of multiple measurements in axonomic problems," *Annals of Eugenics*, Vol. 7, pp. 179-188, 1936.

Friedman, J.H., "Fitting functions to noisy scattered data in high dimensions," *Proc. 20th Symposium on the Interface: Computing Science and Statistics*, Reston, Virginia, April 1988.

Friedman, J.H. and J.W. Tukey, "A projection pursuit algorithm for exploratory data analysis," *IEEE Trans. on Computers*, Vol. 23, pp. 881-889, 1974.

Friedman, J. H. and W. Stuetzle, "Projection pursuit regression," *J. American Statistical Assoc.*, Vol. 76, pp. 817-823, 1981.

Friedman, J.H., "Multivariate adaptive regression splines," *Annals of Statistics* , Vol 19, No. 1, pp. 1-66, 1991.

Gabor, D., "Communication theory and cybernetics," *Trans. of IRE*, Vol. CT-1, No. 4, p. 19, 1954.

Gabor, D., P.L. Wilby, and R. Woodcock, "A universal non-linear filter, predictor and simulator which optimizes itself by a learning process," *J. IEE*, paper received October 17, 1959.

Giles, C.L. and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics*, Vol. 26, No. 23, pp. 4972-4978, December 1, 1988.

Gilstrap, L.O. Jr., "An adaptive approach to smoothing, filtering and prediction," *Proc. 1969 NAECON*, pp. 275-280, 1969.

Gilstrap, L.O. Jr., "Keys to developing machines with high-level artificial intelligence," ASME Design Engineering Conf., ASME Paper No. 71-DE-21, April 19-22, 1971.

Gilstrap, L.O. Jr., S.R. Goldschmidt, and J.F. Elder IV, *Estimating Signals with Polynomial Filters*, Barron Associates, Inc. Interim Technical Report for Century Computing, Inc. under Purchase Order 5035 of F33615-84-C-3609, March 6, 1986.

Gleick, J., *Chaos--Making a New Science*, Penguin Books, New York, 1988.

Goldschmidt, S.R., R.L. Barron, and J.F. Elder IV, *Super-Controllers: Adaptive Control with APN's*, Barron Associates, Inc. Task Final Report, Part II for Universal Energy Systems, Inc. under Task Order 85-10 of F33614-83-C-3000, February 28, 1986.

Hansen, R.J. and E.W. Hendricks, "Active control of complex physical systems: An overview," to be presented at the June 1992 *International Gas Turbine Symposium.*

Hastie, T., R. Tibshirani, "Generalized additive models (with discussion)." *Statist. Sci.*, pp. 297-318, 1986.

Hebb, D. O., *The Organization of Behavior*, John Wiley & Sons, Inc., New York, 1949.

Hecht-Nielsen, R., *Neurocomputing*, Addison-Weslay Publ. Co., pp. 183-191, 1989.

Hess, P. and D.W. Abbott, *Multidimensional Search and Optimization with the OMNIsearch Algorithm*, Barron Associates, Inc. Informal Documentation, Century Computing, Inc. Purchase Order 5035 under F33615-84-C-3609, August 1988.

Ivakhnenko, A.G., "The group method of data handling — A rival of stochastic approximation," *Soviet Automatic Control*, Vol. 1, pp. 43 - 55, 1968.

Ivakhnenko, A.G., "Polynomial theory of complex systems," *IEEE Trans. on Systems, Man, & Cybernetics*, Vol. SMC-1, No. 4, pp. 364-378, October 1971.

Krause, P.C., *Analysis of Electric Machinery*, McGraw-Hill: New York, 1986.

Kuan, C.-M. and K. Hornik, "Implementing recurrent networks", *Proc. of the Seventh Yale Workshop on Adaptive and Learning Systems*, Yale University, pp. 64-68, May 1992.

Kuan, C.-M., K. Hornik, and H. White, "Some convergence results for learning in recurrent neural networks," *Proc. of the Sixth Yale Workshop on Adaptive and Learning Systems*, Yale University, pp. 103-109, 1990.

Lee, R. J., "Internal Circuitry of a Reron," privately published, Lib. of Congress Card TK 7882.C5L4, June 13, 1955.

Lee, R. J., "Generalization of learning in a machine," *Preprints of Papers Presented at Fourteenth Natl. Meeting ACM*, pp. 21-1-21-4, September 1-3, 1959.

Lorentz, G. G., The 13th Problem of Hilbert, in *Mathematical Developments Arising from Hilbert Problems*, F. E. Browder (Ed.), American Mathematical Society, Providence, Rhode Island, 1976.

Mallows, C.L., "Some comments on Cp," *Technometrics*, Vol. 15, pp. 661-675, 1973.

McCulloch, W.S. and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, Vol. 5, pp. 115-133, 1943.

Minsky, M.L. and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, M.I.T. Press, Cambridge, Massachusetts, 1969.

Moddes, R.E.J., R.J. Brown, L.O. Gilstrap, Jr., R.L. Barron, et al., *Study of Neurotron Networks in Learning Automata*, Adaptronics, Inc., AFAL-TR-65-9, February 6, 1965.

Mucciardi, A.N., "Neuromime Nets as the Basis for the Predictive Component of Robot Brains," *Cybernetics, Artificial Intelligence, and Ecology* (Robinson, Ed.), Spartan Books, 1972.

Narendra, K.S and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, March 1990.

Parker, Jr., B.E., R.L. Cellucci, D.W. Abbott, *Active Control of Complex Systems via Dynamic Neural Networks: Combustion Processes in Propulsion Systems*, Contract N00014-89-C-0137, TPR 3, April 1991.

Parker, Jr., B.E., H. V. Poor, et al., *Adaptive Nonlinear Polynomial Neural Networks for Control of Boundary Layer/Structural Interaction*, Barron Associates, Inc. Final Technical Report for NASA Langley Research Center under Contract NAS1-19271, August 15, 1991.

Pearlmutter, B.A. *Dynamic Recurrent Neural Networks*, Technical Report CMU-CS-90-196, School of Computer Science, Carnegie Mellon University, 1990.

Pineda, F.J., "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, 59, pp. 2229-2232, 1990.

Rissanen, J., "A universal prior for integers and estimation by minimum description length," *Ann. Stat.*, Vol. 11, No. 2, pp. 416-431, 1983.

Rosenblatt, F., "The Perceptron," *Cornell Aeronaut. Lab. Rept.* VG-1196-G-1, January 1958.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, M.I.T. Press, Cambridge, Massachusetts, pp. 354-361, 1986.

Schwarz, G., "Estimating the dimension of a model," *Ann. Stat.*, Vol. 6, No. 2, pp. 461-464, 1977.

Specht, D.F., "Generation of polynomial discriminant functions for pattern recognition," *IEEE Trans. on Electronic Computers*, Vol. EC-16, No. 3, pp. 308-319, June 1967.

Shrier, S., R.L. Barron, and L.O. Gilstrap, "Polynomial and Neural Networks: Analogies and Engineering Applications," *Proc. IEEE First Int'l. Conf. on Neural Networks*, Vol. II, pp. 431-439, June 21-24, 1987.

Shynk, J.J., "Adaptive IIR filtering," *IEEE ASSP Magazine*, Vol. 6, No. 2, pp. 4-21, April 1989.

Sutton, R.S., A.G. Barto, and R.J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems Magazine*, Vol. 12, No. 2, April 1992, pp. 19-22.

Werbos, P.J., "An overview of neural networks for control," *IEEE Control Systems Magazine*, Vol. 11, No. 1, pp. 40-41, January 1991.

Williams, R.J., and D. Zipser, *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks*, ICS Report 8805, Institute of Cognitive Science, University of California at San Diego, 1988.

APPENDIX A: Static Polynomial Neural Network Synthesis Algorithms

A.1. *ASPN Facility*

The *ASPN Facility* synthesizes static *estimation* neural networks having no internal feedback paths, no internal time delays, no post-transformations and, in general, a polynomial basis function. The main purpose of *ASPN* is to create a neural network that estimates the value(s) of a dependent variable or variables when interrogated with an input observation vector. In synthesizing a network, *ASPN* selects the most relevant inputs from a list of candidates, determines the most appropriate structure (connectivity) of the network, determines the best algebraic function to use at each node, and optimizes the weights in the network. With *ASPN*, the network structure evolves from the simplest form (a single input connected to the output) to a feedforward network having just-sufficient complexity for the database under consideration. Because *ASPN* synthesizes a static network, the network output vector is a single-point transformation of the input data.

Each *ASPN* network is a combination of nodal elements, where each nodal element may contain a series expansion made up of terms that are a subset of the complete Kolmogorov-Gabor (KG) multinomial (either pre-defined or analyst defined). A number of subsets of the complete KG multinomial are pre-defined in *ASPN;* these subsets, the "single," "double," "triple," and "linear" (affine) are used for one, two, three, or n inputs, respectively:

$$y = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \theta_3 x_i^3 \qquad \text{("single")} \quad A:1$$

$$y = \theta_0 + \theta_1 x_i + \theta_2 x_j + \theta_3 x_i x_j + \theta_4 x_i^2 + \theta_5 x_j^2 + \theta_6 x_i^3 + \theta_7 x_j^3 \qquad \text{("double")} \quad A:2$$

$$y = \theta_0 + \theta_1 x_i + \theta_2 x_j + \theta_3 x_k + \theta_4 x_i x_j + \theta_5 x_i x_k + \theta_6 x_j x_k + \theta_7 x_i^2 + \theta_8 x_j^2 + \theta_9 x_k^2 + \theta_{10} x_i^3 + \theta_{11} x_j^3 + \theta_{12} x_k^3 + \theta_{13} x_i x_j x_k \qquad \text{("triple")} \quad A:3$$

$$y = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \qquad \text{("linear")} \quad A:4$$

In *ASPN*, the analyst may change the terms contained in a nodal element simply by specifying a different set of indices corresponding to different terms in the complete KG multinomial. Exponential (Eq. A:5) and cube-root (Eq. A:6) elements are also available in *ASPN*. These transcendental elements, as well as sigmoidal

functions, offer other forms of nonlinearity that are helpful in some applications. However, experience has shown that Eqs. A:5, A:6, and sigmoidal functions are rarely selected by *ASPN* during model syntheses.

$$y = \theta_0 + \theta_1 e^{(\theta_2 x_i)} \qquad \text{(exponential) A:5}$$

$$y = \theta_0 + \theta_1 (x_i - \theta_2)^{1/3} \qquad \text{(cube-root) A:6}$$

One method used by *ASPN* to keep the number of coefficients ($\underline{\theta}$) from exploding exponentially is to employ subsets of the element terms. Eq. A:3 has the largest number of coefficients of any nodal-element series expansion defined in *ASPN* (except, Eq. A:4 when $n \geq 12$). Even with an interaction order of three, a limit of three inputs, and a maximum degree of three, the function in Eq. A:3 is still sparse (has a low density expansion), because it is lacking terms of the form $\theta_{14} x_i^2 x_j$. The removal of such terms from the function definition allows *ASPN* to be more efficient in use of computing resources and leads to more robust models, as the nodal elements do not have excessive internal degrees of freedom.

Using *ASPN*, the model for each dependent variable usually consists of a network of polynomial elements arranged in layers (see the representative network in Fig. A.1). y_1 and y_2, the two outputs of the example network, are simultaneous estimates for two dependent variables. The sub-network for y_1 is three layers deep and employs six elements and six original inputs. The y_2 sub-network consists of only two elements, one on each of its two layers, and employs four original inputs. Note that there is cross-coupling, or sharing of results, between the sub-networks, and only six of the original twelve or more candidate inputs are used. As with all *ASPN*-synthesized networks, the inputs pass through a normalization stage (N) before introduction into the network, and the dependent variable estimates are re-scaled, i.e., unitized (U) at their outputs.

For each layer of the *ASPN*-synthesized network, a succession of the various nodal functions, with different combinations of inputs, is fitted and scored. Fitting consists of computing the optimum values for the candidate element coefficients using a batch least-squares technique. The candidate element is fitted in such a way that it attempts to solve the entire input-output mapping problem by itself [Ivakhnenko, 1968 and 1971]. A candidate element is scored using a model selection criterion that considers a loss function and a complexity penalty (see Section 2). The model selection criterion used by *ASPN* is either the predicted squared error, PSE; minimum description length, MDL; or predicted classification error, PCE.

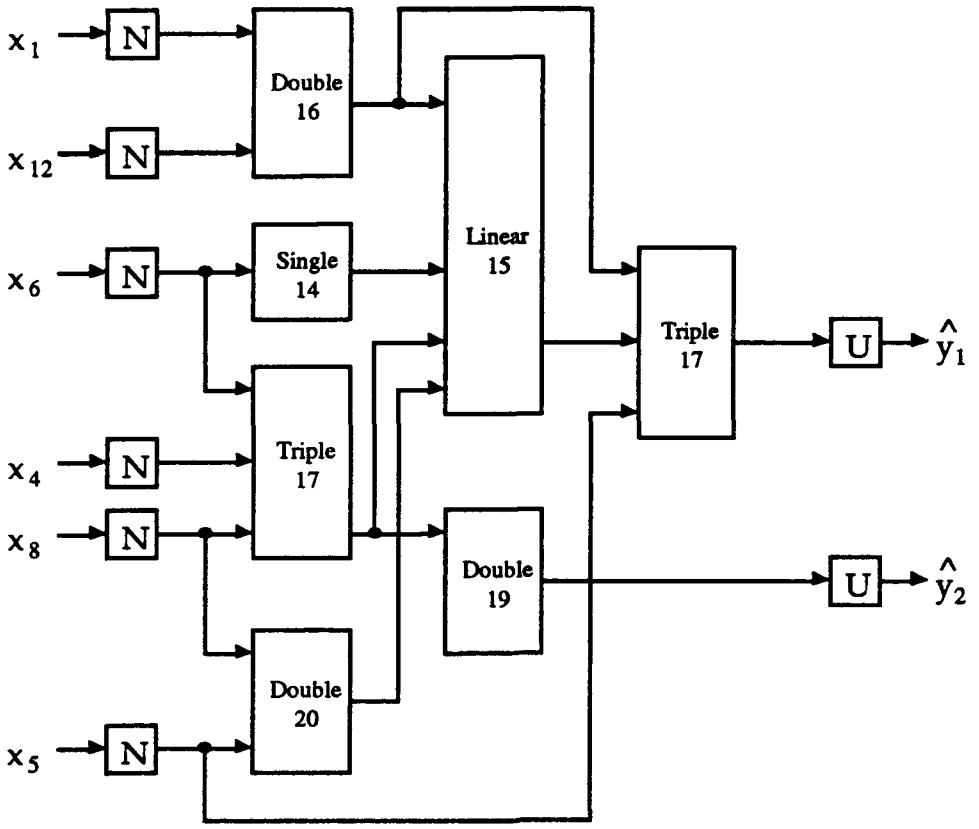


Figure A.1: Sample Polynomial Network

In its simplest form, PSE is written

$$PSE = FSE + \frac{2K}{N}\sigma_p^2 \qquad A:7$$

in which σ_p^2 is a prior estimate of the true error variance that does not depend on the model being considered, K is the number of parameters within the model, N is the number of exemplars in the training database, and FSE is the fitting squared error (i.e., the mean squared error):

$$FSE = \sum_{i=1}^{n} \left(y_i - f(\underline{x}_i, \hat{\theta})\right)^2 \qquad A:8$$

where $f(\underline{x}_i, \hat{\theta})$ is the network output.

PSE is a very conservative criterion provided the fixed σ_p^2 exceeds the true error variance, a condition readily verified upon completion of model synthesis. This conservatism is important in engineering and predictive estimation applications. PSE is not dependent on the shape of the error distribution.

The MDL criterion is based on Shannon coding and follows the equation

$$MDL = \text{Ceiling}\left[BN + \frac{N}{2}\log\left(2\pi e\ FSE\ N_c\right)\right] \text{ bits,} \qquad A:9$$

where B is the number of bits that would be used to encode each observation error (say, 16), N_c is N raised to the (K+1)/N power, and the Ceiling function refers to use of the next-highest integer. Use of the MDL criterion is most appropriate if the shortest explanation of known data is believed to provide a good description of statistically similar data in the future.

The final model selection criterion available in *ASPN* is PCE:

$$PCE = \left[1.0 - \frac{1}{N}\sum_{i=1}^{N} H\left(|y_i - \hat{y}_i|, C_{Tol}\right)\right] + C_{Mult}\frac{K}{N} \qquad A:10$$

where:

$$H \equiv \begin{cases} 1 \text{ if } |y_i - \hat{y}_i| < C_{Tol} \\ 0 \text{ if } |y_i - \hat{y}_i| \geq C_{Tol} \end{cases} \qquad A:11$$

y_i is the true output given the *i*th input vector i. (For PCE, true outputs must be positive.)

$\hat{y}_i$ is the candidate model output given input vector i

C_{Tol} is the classification tolerance (default value 0.5)

C_{Mult} is the complexity penalty multiplier (nominally 1.0).

When all possible candidate elements have been fitted and scored, a given layer is constructed by selecting the "best" elements, as gauged by the model selection criterion. The maximum number of "best" elements allowed in each layer and the maximum number of layers are specified by the analyst. To improve performance (minimize the modeling criteria), elements may be joined in series and in parallel, creating a feedforward network. The inputs for the next layer include the previous layer outputs and the network inputs. Since only the best elements or nodes are retained in each layer, successive layers can only improve or maintain the performance of the previous layer. The growth of the network is halted when the model ceases to improve with the addition of new elements, or the maximum number of layers has been reached.

At the completion of network construction, the network may include many elements that do not contribute to the final output (are not used in the

transformation of the input into the network output). *ASPN* purges the unused elements from the network, and the final model is output in forms appropriate for both the user and other *ASPN Facility* utility programs.

The *ASPN Facility* utilities, as described below, can be used to improve the model coefficients, interrogate the model on unseen data, and provide variations in the form in which the model is displayed or executed.

The *GLOBALO* utility optimizes the parameters of a multinomial model, leaving the model structure unchanged, to minimize fitting error over the optimization data set (which may not be the same data with which the model was synthesized). *GLOBALO* performs a global, multivariate search. The fitting error criterion, minimized to determine the optimum values of the coefficients, is based on a user-defined function of errors, such as minimum squared error or minimum absolute error. Using a parameter space search algorithm, *GLOBALO* tries various parameter values over the data observations that are read until an iteration limit is reached. Lastly, the best coefficients are inserted into the network structure and a new, optimized, model is output.

The *IMP* utility implements (encodes and evaluates) the neural network model generated by the *ASPN* program. *IMP* provides two main options to the user, known as *Create* and *Evaluate*. With the *Create* option, *IMP* creates an output file in the form of a FORTRAN or C routine which implements the model; that is, *IMP* provides a subroutine or function which provides network model outputs given a vector of input values. With the *Evaluate* option, *IMP* interrogates the *ASPN* model using an evaluation data base and outputs both detailed and summary results and statistics.

Because the output vector of an *ASPN*-synthesized network is a single-point transformation of the input data, models created by *ASPN* can be alternatively represented in an algebraic, rather than network, form. The *XPRESS* utility expresses an *ASPN* model entirely in terms of the original input variables; that is, it expands the network to compute algebraically the "function of functions" it represents. *XPRESS* can also graph the network, derive the derivatives of the output polynomial with respect to any input, and interactively display the expanded equations in a variety of formats.

A.2 *CLASS*

CLASS synthesizes static polynomial *classification* networks. The *CLASS* algorithm uses the minimum-logistic-loss criterion derived for a C-class problem, where C is the number of outputs (classes), to classify input observation vectors.

The structure of a *CLASS*-synthesized classifier network is a static feedforward network of polynomial elements and a fixed, static logistic transformation (probability computation) of the network outputs (see Fig. A.2). In practice, the

network structure is fixed in advance, and each node usually contains the same polynomial equation (core transformation). Since the probability computation outputs always sum to unity, only C–1 nodes are synthesized to solve a C-class problem. Each of the C–1 nodes (sub-networks) returns essentially unbounded values (∞ to $-\infty$) that indicate the probabilities of membership in their respective classes or in an arbitrary baseline class. The C^{th} node is defined to be identically zero.

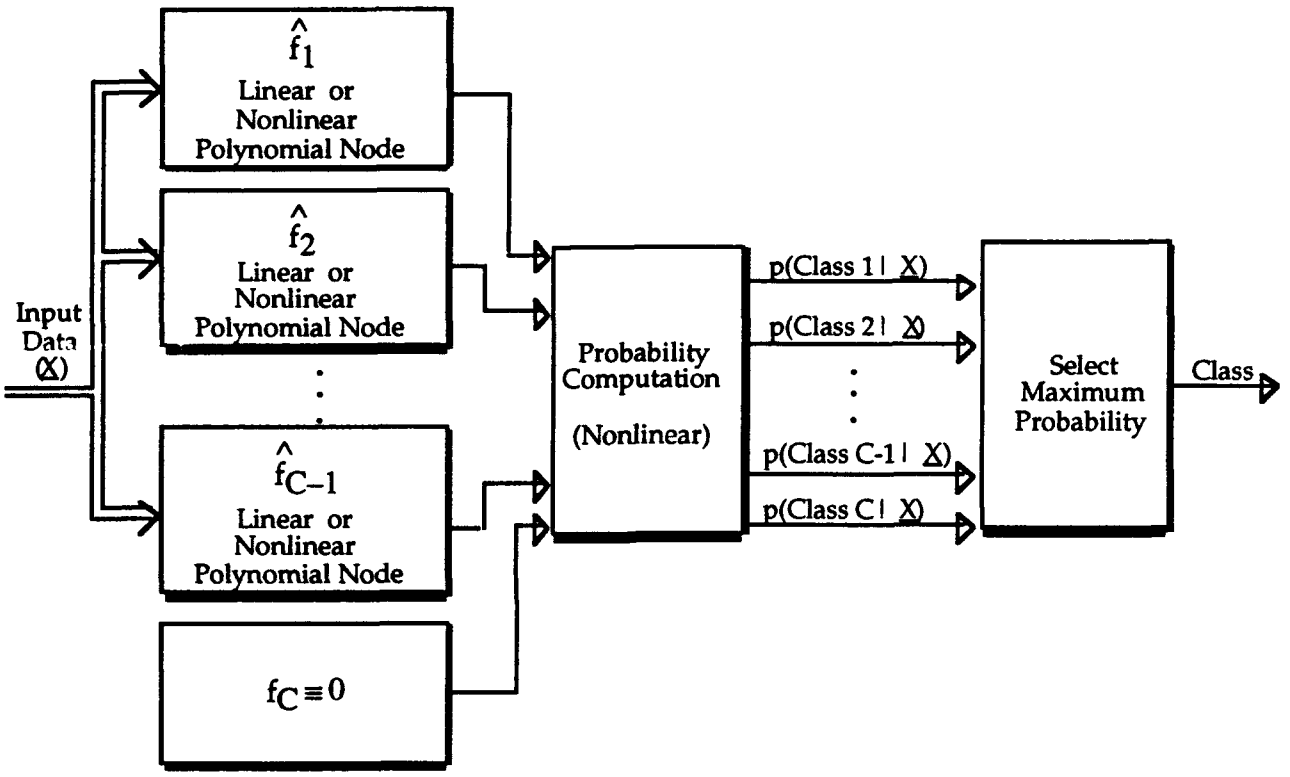


Figure A.2: The General Structure of the *CLASS* Network

The series expansion of a *CLASS*-synthesized network is a polynomial having an initial structure (before carving) that is specified by the analyst. Possible nodal element structures are subsets of the KG multinomial and are defined as:

$$y = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \qquad \text{("linear")} \quad \text{A:12}$$

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_2 + \theta_2 x_2^2 \qquad \text{("additive")} \quad \text{A:13}$$

$$+ \theta_3 x_2^3 + \ldots$$

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_2 + \theta_5 x_2 x_1 \quad \text{("complete")} \quad \text{A:14}$$
$$+ \theta_6 x_2 x_1^2 + \theta_7 x_2^2 + \theta_8 x_2^2 x_1 + \theta_9 x_2^3 + \ldots$$

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_1 x_2 + \theta_5 x_1 x_3 \quad \text{("multilinear")} \quad \text{A:15}$$
$$+ \theta_6 x_2 x_3 + \theta_7 x_1 x_2 x_3 + \ldots$$

CLASS allows the analyst to specify the degree of Eqs. A:13 and A:14. *CLASS* also allows arbitrary multinomial (multivariate polynomial; i.e., any combination of Eqs. A:14 and A:15) node structures to be specified and named in a node file. All input data components ($\underline{x}$) are submitted to each node. Therefore, to prevent the number of coefficients from exploding exponentially, it is important for the analyst to keep the degree of the specified polynomial reasonably low.

CLASS can fit the entire network simultaneously, can build the network parameters one at a time, or can build the parameters in groups. Fitting all the parameters simultaneously is straightforward. (The search algorithm is applied to all of the parameters.) However, to build parameters individually or in groups, a criterion that trades off complexity and fitting error must be employed. The model selection criterion for synthesis of a *CLASS* network is given by

$$J = \text{(Fitting Error)} + \text{(Complexity Penalty)}$$
$$= \left(\frac{1}{n} \sum_{i=1}^{n} d_i \right) + \left(\frac{C_{Mult} \tilde{Q}}{n} \right) \quad \text{A:16}$$

where $\tilde{Q}$ is the number of non-zero parameters in the network.

The complexity constraining term

$$\frac{C_{Mult} \tilde{Q}}{n} \quad \text{A:17}$$

penalizes networks with large numbers of coefficients to discourage overfitting. The complexity multiplier, C_{Mult}, provides a way to increase or decrease the complexity penalty *a priori*. To implement the Akaike Information Criterion, C_{Mult} should be set to unity.

The multiclass logistic-loss function, $d_i(\,)$, evaluated at the *i*th observation, is

$$d_i(\,) = d\left(y_i, \hat{f}(\underline{x}_i; \underline{\theta})\right) = -\underline{y}_i \cdot \hat{f}(\underline{x}_i; \underline{\theta}) + \log_e \left(\sum_{k=1}^{C} \exp\left(\hat{f}_k(\underline{x}_i; \underline{\theta}_k)\right) \right) \quad \text{A:18}$$

The fitting error is normalized by $\log_e(C)$. When the parameters are all identically zero (indicating a no-knowledge model), the fitting error is 1.0.

Table A.1 defines the symbols used in the multi-class logistic-loss function.

Table A.1: Mathematical Symbol Descriptions

Symbol	Description
J	The objective function. The normalized average of the loss function over n observations and the complexity penalty.
$d(\underline{y}_i, \hat{\underline{f}}_i)$	The C-class logistic loss function evaluated on observation i of the data.
$\underline{x}_i$	The vector of input data for observation i.
$\underline{y}_i$	The modeling output vector for observation i. This vector has only one component that is unity (the rest are zero). The non-zero component indicates the actual class.
$\hat{\underline{f}}(\underline{x}_i, \underline{\theta})$	The vector of sub-network outputs. The range of these outputs is essentially unbounded.
$\underline{\theta}$	The list of all parameters in the classifier network (all of the $\underline{\theta}_k$'s combined).
$\underline{\theta}_k$	The parameters in node k.

To interrogate the *CLASS*-synthesized network, the input data are fed into each of the discriminating nodes, $\hat{f}_1$ through $\hat{f}_{C-1}$. The outputs of these nodes are unbounded. The C probabilities of class membership are then computed using

$$\hat{p}(\text{class } k \mid \underline{x}) = \frac{\exp\left(\hat{f}_l(\underline{x}; \underline{\theta}_l)\right)}{\sum_{k=1}^{C} \exp\left(\hat{f}_k(\underline{x}; \underline{\theta}_k)\right)} \quad ; \quad 1 \leq l \leq C \qquad \text{A:19}$$

Note that $\hat{f}_C$ is identically zero. The distribution of the C probabilities is determined, and the corresponding class is then chosen.

Because minimization of logistic loss involves a nonlinear transformation from the classifier weights to the estimated probabilities, least-squares adjustment of the weights is inappropriate. A Levenberg-Marquardt (LM) search may be employed (see Appendix B). LM is a nonlinear regression technique that exploits the derivative information that is known analytically.

The constrained minimum-logistic-loss criterion, which is explicitly designed for classification problems, provides performance superior to classifiers fitted using estimation criteria. Estimation networks place emphasis on estimation accuracy; optimum minimum-logistic-loss networks instead place emphasis on maximizing the likelihood of correct class discrimination.

APPENDIX B: Optimization Techniques†

B.1 Description of the *Guided Random Search Algorithm*

The *Guided Random (GR) Search Algorithm* is a random optimization search intended primarily for the initial and final stages of numerical searches. It has multimodal search capabilities, but these are not as powerful as in *GARS*, which is described in Section B.2. The basic *GR* algorithm is quite simple, yet it embodies many qualities of other, more sophisticated, search algorithms.

GR uses an "amoeba," which is a set of search points, typically five, comprising the current information to be used by the search algorithm. The amoeba moves through the search space by adding and removing points from its search set in the following manner:

1. Remove worst point from the amoeba.
2. Add most recent point to the amoeba.
3. Recompute mean and standard deviation of the amoeba for each search dimension.
4. Determine the next search point to test using the following vector equation:

$$\underline{\mu}_g = \underline{X}_n^* + speed \cdot (\underline{X}_n^* - \underline{\mu}_n) \qquad \text{B:1}$$

$$\underline{X}_{n+1} = \underline{\mu}_g + dispersion \cdot N[\underline{\mu}_g, \underline{\sigma}_n] \qquad \text{B:2}$$

where n denotes the iteration number and:

$\underline{\mu}_g$ is the mean of the goal region,

$\underline{X}_n^*$ is the location of the best point in the amoeba,

$\underline{\mu}_n$ is the vector mean of the five amoeba points,

$\underline{X}_{n+1}$ is the next search point to be tested,

---

† Reprinted from Barron, Abbott, Cellucci, and Blackman [1991].

$\underline{\sigma}_n$ is the vector standard deviation of the five amoeba points,

$N[\underline{\mu}_g, \underline{\sigma}_n]$ is a Gaussian random vector of mean $\underline{\mu}_g$ and standard deviation $\underline{\sigma}_n$, and

speed and *dispersion* are search parameters (constants) set by the analyst, each typically to unity.

Thus, the strategy of the amoeba search is very simple: move from the centroid of the amoeba beyond the point with the best score. The distance beyond the current best point to use, the "goal" region, is proportional to the distance between the best point and the amoeba centroid. With *speed* set to unity, the mean of the goal region for the next search point will be collinear with $\underline{X}_n^*$ and $\underline{\mu}_n$, at a distance $(\underline{X}_n^* - \underline{\mu}_n)$ beyond $\underline{X}_n^*$. The shape of the goal region is determined by the standard deviations of the current amoeba.

If an amoeba repeatedly receives good points in a particular direction it will stretch out along that direction and thus begin to accelerate. (See Figure B.1) If an amoeba comes across an area representing a minima, it will surround that region and shrink, causing it to focus on that region or, when necessary, "thread the needle" and re-expand after getting through that portion of the search space.

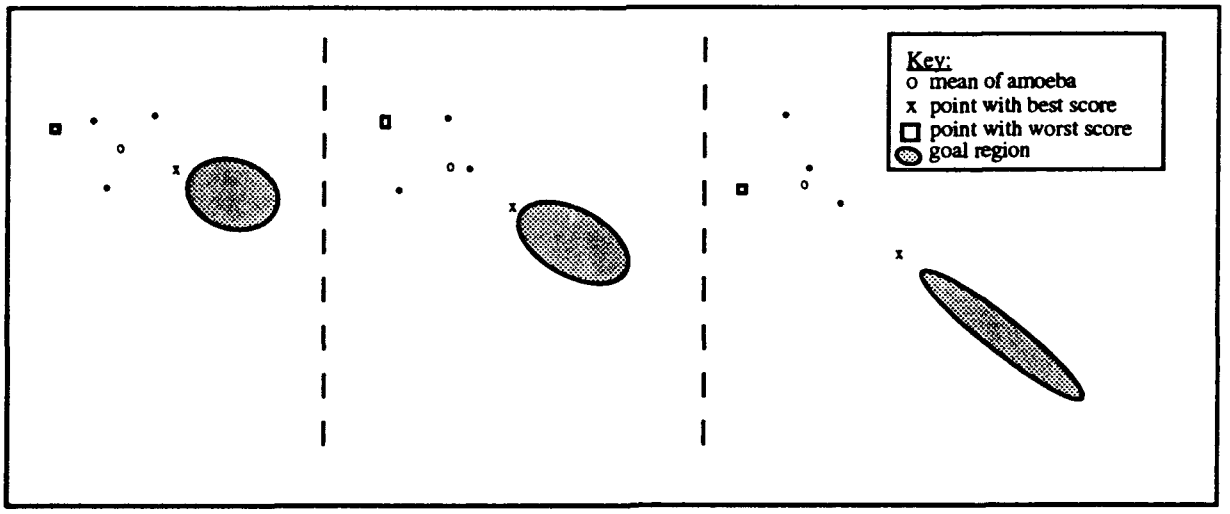


Figure B.1: Sample *GR* Amoeba Acceleration

To initialize the *GR* search, initial values and standard deviations for each search dimension are specified.

B.2 Description of the *Guided Accelerated Random Search* Algorithm

The *Guided Accelerated Random Search (GARS)* algorithm is a random optimization search intended primarily for the mid-game stage of search. *GARS* is a powerful multimodal search that avoids trapping in local minima of the performance surface. It is also effective when there is a shallow score slope to be traversed, in which case the *GARS* acceleration/deceleration capabilities quickly find the local minimum.

To govern random experiments, the *GARS* algorithm uses the following vector equation:

$$\underline{X}_{n+1} = \underline{X}_n^* + \textit{dispersion} \cdot N[0, \underline{\sigma}_n] \qquad \text{B:3}$$

where $\underline{X}_n^*$ is the location of the best-to-date trial, $N[0, \underline{\sigma}_n]$ is a Gaussian random vector of mean zero and standard deviation $\underline{\sigma}_n$, and *dispersion* is a search parameter (constant) set by the analyst, typically to unity. Note the similarity of Eq. B:3 for *GARS* to Eq. B:2 used in the *GR* search. Whereas *GR* conducts its random explorations in a region removed from $\underline{X}_n^*$, *GARS* performs its random trials in a region centered at $\underline{X}_n^*$.

Whereas *GR* establishes $\underline{\sigma}_n$ from the standard deviation of the five amoeba points, *GARS* determines $\underline{\sigma}_n$ from the score of the best-to-date trial. Thus, for *GARS*

$$\underline{\sigma}_n = \frac{J_n^*}{J_0^*} \cdot \underline{\sigma}_0 \qquad \text{B:4}$$

where:

$\underline{\sigma}_n$ is the standard deviation vector used in computing $\underline{X}_{n+1}$

$\underline{\sigma}_0$ is the initial standard deviation vector

J_n^* is the best-to-date score as of iteration n

J_0^* is the initial best-to-date score.

J_0^* can be the value determined from an opening *GR* search. $\underline{\sigma}_0$ is usually specified by the analyst using *GARS*, but may be determined from an opening search with *GR*.

Using Eq. B:4, the standard deviations of the random trial components shrink to zero as the score J_n^* approaches zero (perfection). However, even when the standard deviations become small, the search can still move quickly to remote

regions of the search space, propelled by a deterministic acceleration heuristic that will now be described.

Once $\underline{X}_{n+1}$ is selected, the new point and the corresponding score are computed. If a new best score is not found at this new point, another random trial is selected and another new point is tested. If a new best score is found, the step $\Delta\underline{X} = \underline{X}_{n+1} - \underline{X}_n$ is multiplied by two to establish the next trial. As long as consecutive new best scores are found, $\Delta\underline{X}$ is repeatedly doubled, causing exponential acceleration of the search. Once a new best is not found, the search decelerates (it has gone too far) and tries again. If deceleration fails to improve the score, a final point is attempted on the opposite side of the best point found so far from the decelerated attempt, and the search resets, finding a new random $\Delta\underline{X}$ with acceleration factor of one.

Figure B.2 presents details of the *GARS* algorithm.

B.3 Combined *Guided Random/Guided Accelerated Random Search* Search

Care must be taken in initializing *GARS*. If *GARS* is initialized with standard deviations that are too large, the random phase may not readily find a new best score from which to begin the acceleration process, thus increasing search time. However, initial standard deviations that are too small may prolong the search time by requiring very large accelerations to provide meaningful improvement of the best score. Therefore, *GARS* is most effective when preceded by a search algorithm such as *GR*. The flow chart for a combination of *GR* and *GARS* (called *GR/GARS*) is shown in Fig. B.3. The *GR* startup finds a good region for *GARS* and, because *GR* sets its own standard deviations based upon the four best scores found and the most recent point tested, an estimate of standard deviations is made available for use by *GARS*.

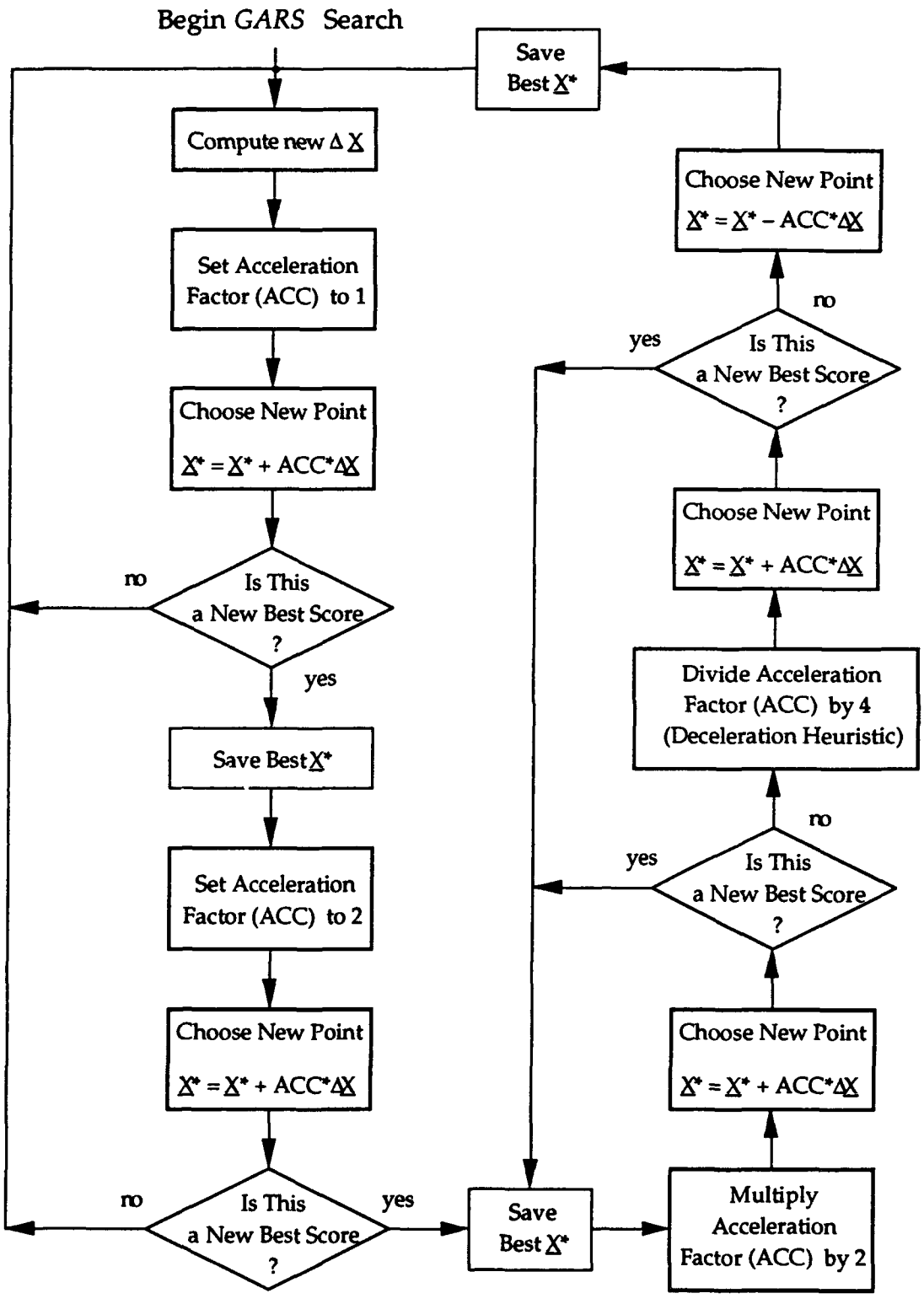


Figure B.2: *GARS* Algorithm Block Diagram

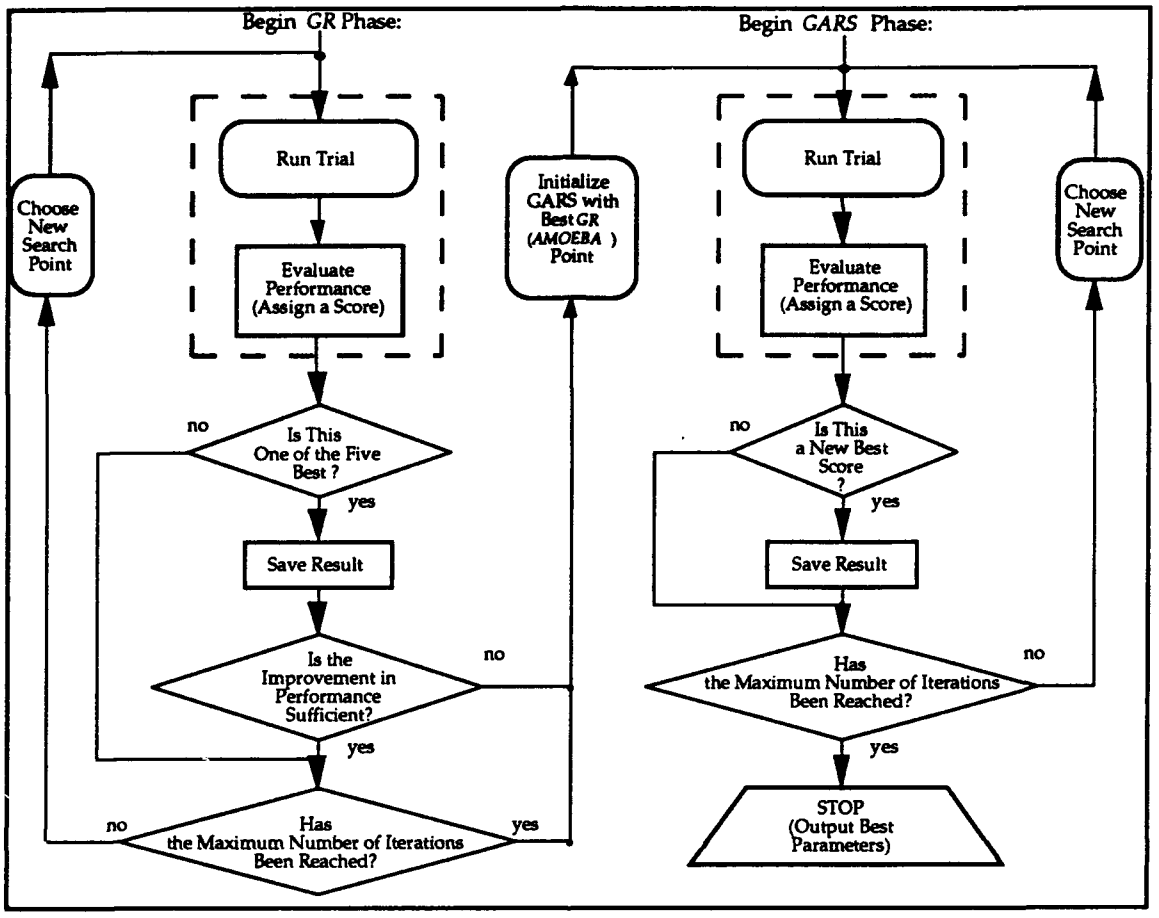


Figure B.3: Combined *GR/GARS* Search

The combined *GR/GARS* numerical search is outlined in Fig. B.3. Fig. B.4 shows a representative search convergence (learning) curve for this search. The abscissa of Fig. B.4 of this chart is the iteration number, and the ordinate is the relative penalty assigned by the utility function. Note the rapid acceleration of search convergence several times during the search.

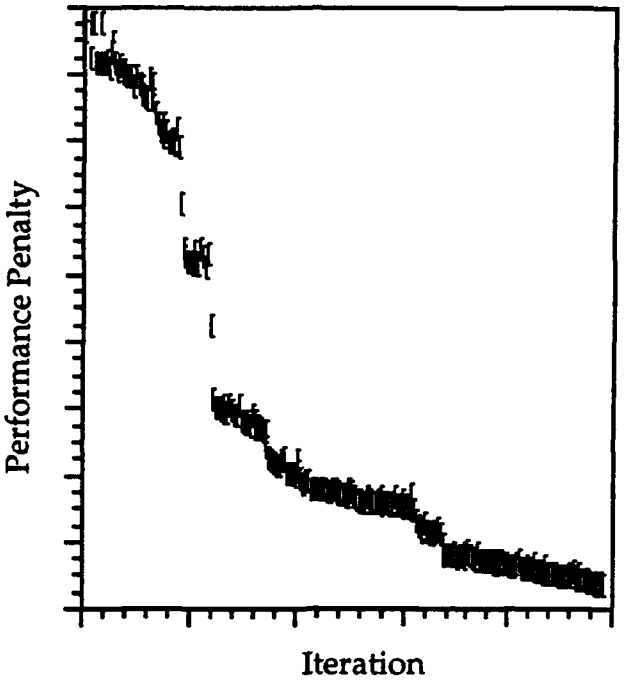


Figure B.4: Example Learning Curve for *GR/GARS* Search

B.4 The Levenberg-Marquardt Search

The Levenberg-Marquardt (LM) search is useful in batch nonlinear regression to:

(1) Synthesize nonlinear classification neural networks.

(2) Adapt nonlinear estimation neural networks, if a batch algorithm is used instead of a recursive algorithm to identify system parameters.

Given n nonlinear equations in m variables,

$$\underline{y} = \underline{f}(\underline{x}) \qquad \text{B:5}$$

the goal of the LM search is to minimize the objective function

$$J(\underline{x}) = \| \underline{y} - \underline{f}(\underline{x}) \|_2^2 = [\underline{y} - \underline{f}(\underline{x})]^T [\underline{y} - \underline{f}(\underline{x})] = \sum_{i=1}^{n} [y_i - f_i(\underline{x})]^2 \qquad \text{B:6}$$

The gradient and truncated Hessian of the objective function are given, respectively, by

$$\nabla J(\underline{x}) = \frac{\partial J}{\partial \underline{x}} = -\sum_{i=1}^{n} 2\left[y_i - f_i(\underline{x})\right]\left(\frac{\partial f_i(\underline{x})}{\partial \underline{x}}\right) \qquad \text{B:7}$$

$$\underline{\underline{H}}(\underline{x}) = \frac{\partial^2 J}{\partial \underline{x}^2} \approx \sum_{i=1}^{n} 2\left(\frac{\partial f_i(\underline{x})}{\partial \underline{x}}\right)\left(\frac{\partial f_i(\underline{x})}{\partial \underline{x}}\right)^T \qquad \text{B:8}$$

where, following Gauss, the Hessian terms corresponding to the second derivatives of the function, $f(\cdot)$, are ignored. The gradient of the objective function is set to zero to minimize $J(\underline{x})$. With the gradient and Hessian of $J(\underline{x})$, iterative search algorithms of the form

$$\underline{x}_\alpha^{(k+1)} = \underline{x}^{(k)} - \alpha\,\underline{\delta} \qquad \text{B:9}$$

can be implemented, where α is a parameter that affects the rate of the search and $\underline{\delta}$ is the search direction. The LM algorithm is given by the particular choice

$$\underline{\delta} = \left[\underline{\underline{H}}(\underline{x}^{(k)}) + \Lambda\,\mathrm{diag}\,\underline{\underline{H}}(\underline{x}^{(k)})\right]^{-1} \nabla J(\underline{x}^{(k)}) \qquad \text{B:10}$$

where diag $\underline{\underline{H}}$ $(\underline{x}^{(k)})$ represents the matrix $\underline{\underline{H}}$ $(\underline{x}^{(k)})$ with all but the diagonal elements set to zero and Λ is a parameter of the search. The parameters found through this algorithm should be saved and used as a starting point for later iterations. This algorithm has super-linear to quadratic convergence capabilities; it provides a solution very rapidly in comparison with alternative search techniques. However, it is a unimodal search, subject to trapping at local minima. Thus, the *GR/GARS* search, which is multimodal, should also be considered.

APPENDIX C: Prediction of Behavior of a Deterministic Chaotic Process

C.1 Background

Over the last two decades, there has developed a deepening awareness that many forms of apparently random behavior are, in fact, manifestations of *deterministic* nonlinear dynamical relationships present in the underlying fluid dynamics, biology, chemistry, economics, etc. of observed processes. Robert May and others have demonstrated that process governing equations of deceptive simplicity can be capable of amazingly complex and chaotic behavior. [See Gleik, 1988.] One such relationship, known as May's population equation, i.e.

$$x(k+1) = r\,x(k)\,[1 - x(k)] \qquad \text{C:1}$$

has been widely studied. In this equation, x(k) denotes the population of a single species at sample time k, while x(k+1) is the population of this species at the next succeeding sample. For given x(0), population fluctuations are governed by the nonlinear parameter r, and the resulting population sequence is simple, complex, or chaotic depending entirely on the value of r.

Below a certain small value of r, the sequence of population levels decays to extinction. Above this critical value, but below a second threshold, the population smoothly approaches a constant steady-state level determined by r, with larger r producing a larger equilibrium population. Between the second threshold and a third threshold, the approach to the constant steady-state level exhibits a damped oscillation. Between the third and a fourth threshold, the population shows a steady-state limit-cycle oscillation between two levels, where these levels are determined by r. Between the fourth and a fifth threshold, the limit-cycle fluctuations involve four discrete levels. Above this there are thresholds related to eight, 16, 32, ... levels. Then, at a certain critical value of r, known as the "point of accumulation" ($r \approx 3.7$), the periodic steady-state oscillations break into chaotic behavior. (Even in the chaotic regime, fascinating subtleties arise as r is increased further. Detailed discussion of this subject can be found in the literature.)

Engineering systems can also exhibit complex, even chaotic, forms of behavior, particularly when "driven hard," i.e., to the limits of their performance capabilities. For example, the fluctuations of population level characterized by May's equation bear resemblance to variations of gas flow parameters in a turbopropulsion system compressor in the presence of rotating stall, during the advent of compressor surge, and in surge recovery. Studies by other investigators in the OCNR Active Control Initiative have tended to dispel the notion that the complexities of compressor behavior are predominantly due to random causes.

Rather, deterministic models are beginning to offer very promising explanations and, even, many of the needed predictive capabilities.

Underlying attempts to improve the control of systems that have fundamentally chaotic, yet deterministic modes of response is the question, "Can we model chaotic systems from observations of their behavior in well-behaved regimes, or must we drive them into chaotic behavior (possibly at great cost) to obtain the needed information for modeling?" In this context, inspection of Eq. C:1 indicates that it should be possible to "learn" May's equation from observational data obtained in *any* mode of population behavior. That is, the structure of Eq. C:1 is invariant and presumably can be identified, along with a particular value of r, from observations of the population sequence. If this identification is made, it should then be possible to change the identified value of r to other values and predict what will happen in regimes not previously observed.

Notationally, a learned difference equation model for May's equation might have the form

$$\hat{x}(k+1) = \hat{f}\,[x(k),\ x(k-1),\ \ldots,\ x(k-n)] \qquad \text{C:2}$$

in which the back-reference factor, n, would be learned along with the form and coefficient values within $\hat{f}[\cdot]$. Then, taking

$$\hat{x}^*(k+1) = G\,\hat{f}\,[\cdot] \qquad \text{C:3}$$

and experimenting with different values of the gain, G, one could readily simulate the expected system behavior for arbitrarily many cases.

More generally, the learned model structure for an engineering system will usually involve a set of linear and nonlinear parameters, $\underline{\theta}$, not just the single (nonlinear) parameter, G, appearing in the above example. Artificial neural network synthesis methods (Sections 1 – 3 of this report) can be used to identify the appropriate structure from observational (and, sometimes, simulation) data. *This identification may be made at an arbitrary point in a well-behaved regime,* for which the corresponding parameter vector, i.e., $\hat{\underline{\theta}}$, is readily found. Then, by exploring with different $\hat{\underline{\theta}}$ values inserted in the learned model structure, one may perform *gedanken* experiments to predict the system behavior in complex and chaotic regimes that may not have been directly observed in gathering the database for modeling.

Looking beyond the above conceptualization, consider that, from Eq. C:1

$$r = \frac{x\,(k+1)}{x(k)\,[1 - x(k)]} \qquad \text{C:4}$$

so that, if the structure is given, r may be calculated from two or more successive observations. Extending this idea, one may employ least squares or recursive least squares to obtain the estimated $\hat{\underline{\theta}}$ from a data sequence, if a model exists (has been specified or learned) that is linear in the parameter-vector components. One may also identify $\hat{\underline{\theta}}$ in a model that is nonlinear in its components; this may be accomplished by a Gauss-Newton form of numerical search (such as an iterative version of the Levenberg-Marquardt algorithm, Appendix B).

If the process to be controlled has a structure that has been characterized in a well-behaved regime, but the process is found to be operating in a complex or chaotic regime, the knowledge needed for recovery of control (i.e., to return the process to a well-behaved condition) can be acquired by estimating $\hat{\underline{\theta}}$ in the complex/chaotic regime. Such estimation would generally be more rapid than complete re-learning of the process structure and coefficients. For example, if the structure of Eq. C:1 has been previously learned, $\hat{r}$ may be estimated from successive measurements of x.

Figures C.1 – C.6 illustrate the population sequences resulting from May's equation using x(0) = 0.4 and the indicated values of r. These figures reveal the behavior regimes discussed in Section C.1 above.

C.2 Learning the Population Equation from Observations

C.2.1 Static PNNs

Static polynomial neural network nodal elements were trained (fitted) with data generated by May's population equation using the discrete values of r shown in Figures C.1 – C.6. (The network synthesis algorithm, *ASPN*, was never explicitly "told" what these values were.) The trained elements were then evaluated in the chaotic regime, and the following median absolute error (MAE) values were obtained for training on r = 2.5, 2.75, 3.0, 3.25, 3.5, and 3.75, with evaluation in the chaotic regime at r = 3.75:

1–A. For *multilinear triple*, three candidate inputs x(k), x(k–1), x(k–2), and eight candidate terms carved to four terms by *ASPN*.....0.146

1–B. For *multilinear double*, two candidate inputs x(k), x(k–1), and four terms (not carved by *ASPN*).....0.134

1–C. For *triple* with squares and cubes, three candidate inputs x(k), x(k–1), x(k–2), and 14 terms (not carved by *ASPN*).....0.009

1–D. For *double* with squares and cubes, two candidate inputs x(k), x(k–1), and eight terms (not carved by *ASPN*).....0.011

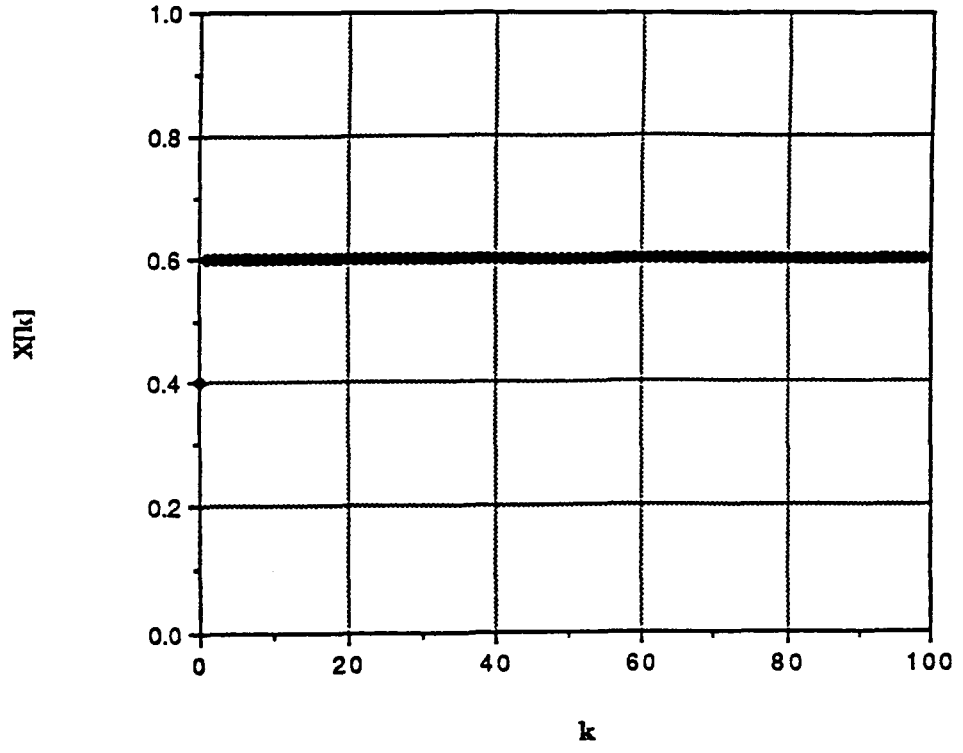


Figure C.1: Population Equation Response for r = 2.5; x[0] = 0.4

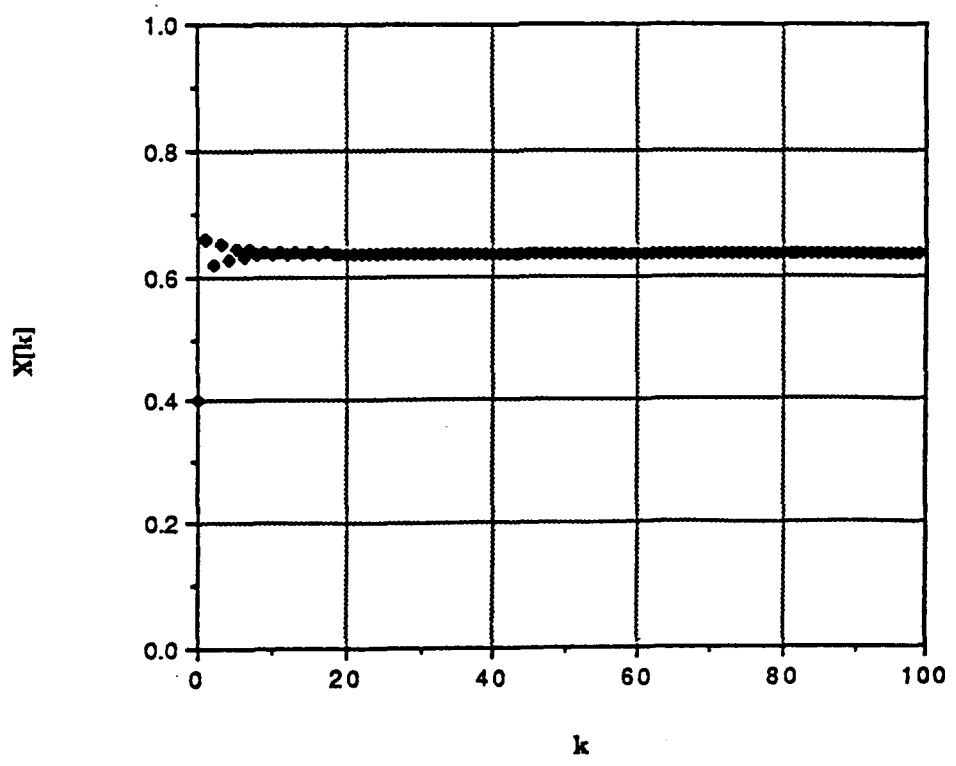


Figure C.2: Population Equation Response for r = 2.75; x[0] = 0.4

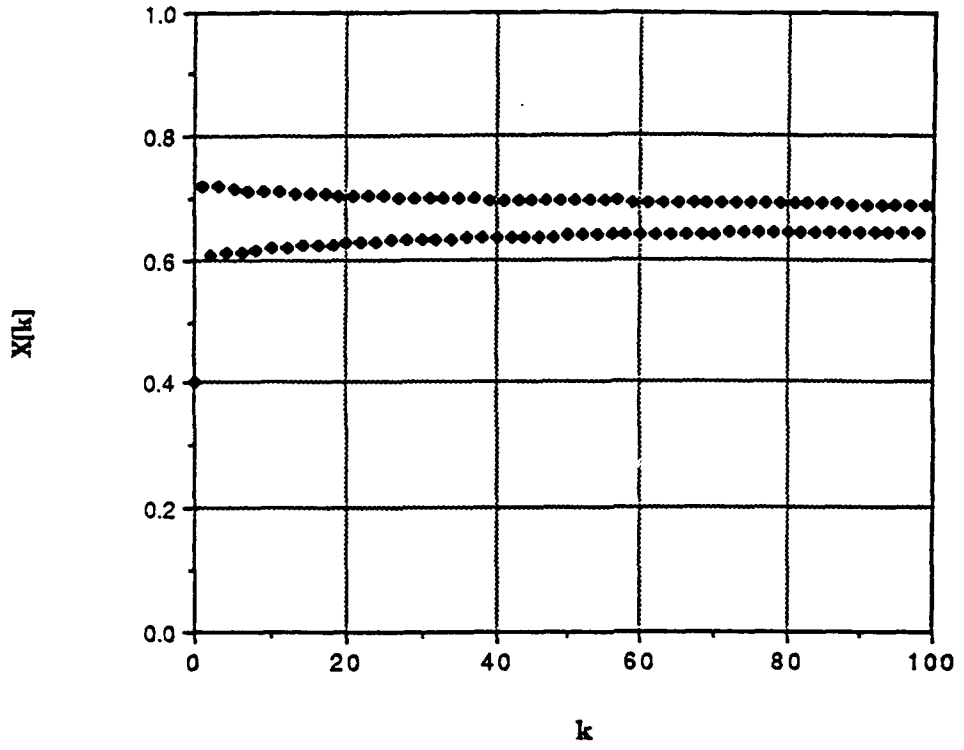


Figure C.3: Population Equation Response for r = 3.0; x[0] = 0.4

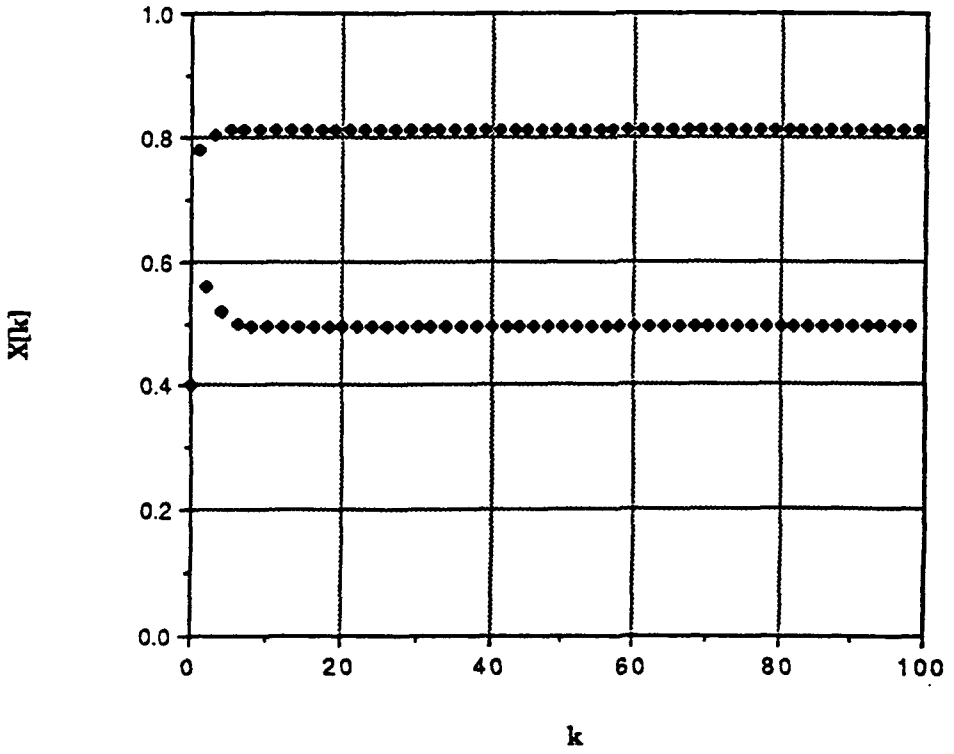


Figure C.4: Population Equation Response for r = 3.25; x[0] = 0.4

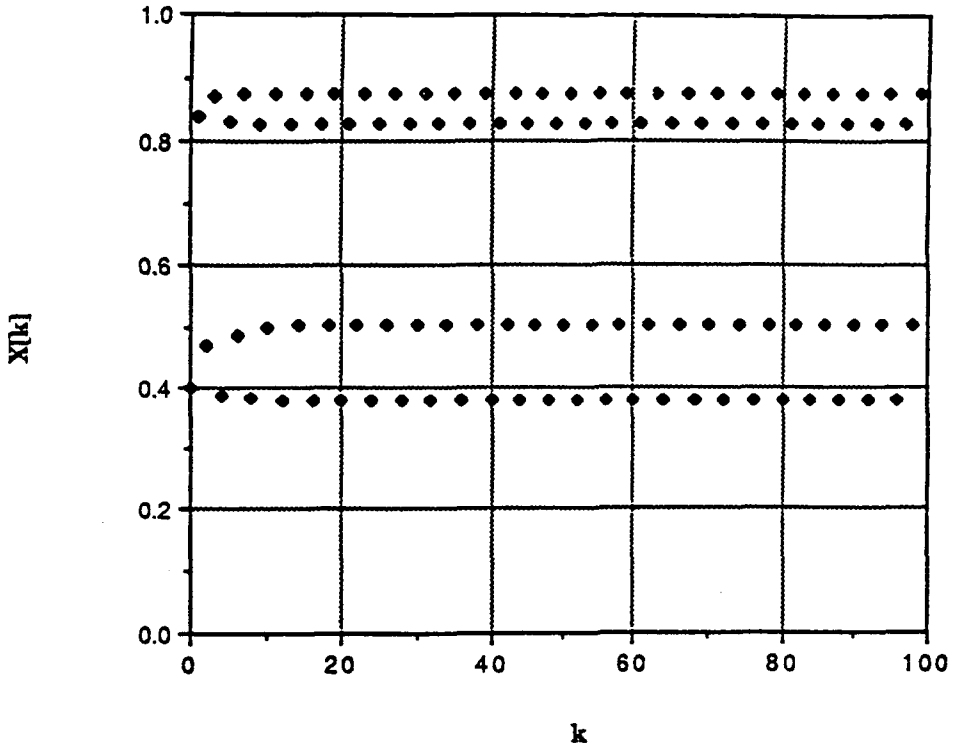


Figure C.5: Population Equation Response for r = 3.5; x[0] = 0.4

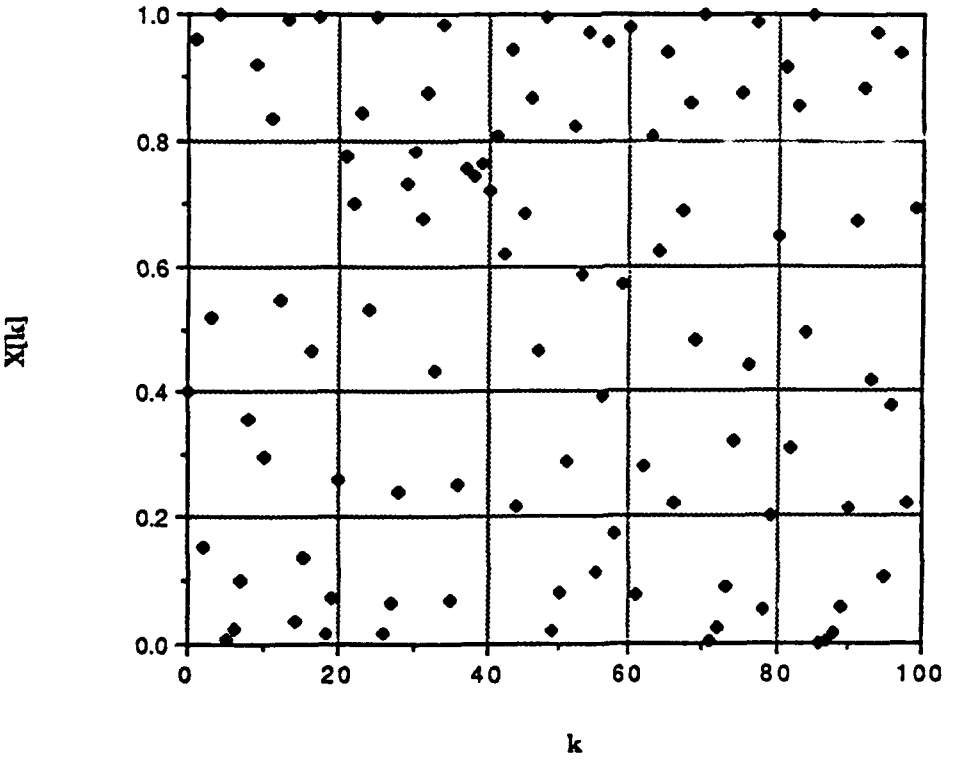


Figure C.6: Population Equation Response for r = 3.75; x[0] = 0.4

The above work was then repeated with training data *omitted* for r = 3.5 and 3.75, that is, with data for the four-level limit-cycle and chaotic regimes omitted. The following MAE values were obtained for training on r = 2.5, 2.75, 3.0, and 3.25, with evaluation in the chaotic regime at r = 3.75:

2–A. For *multilinear triple*, three candidate inputs x(k), x(k–1), x(k–2), and eight candidate terms carved to four terms by *ASPN*.....0.195

2–B. For *multilinear double*, two candidate inputs x(k), x(k–1), and four candidate terms carved to three terms by *ASPN*.....0.133

2–C. For *triple* with squares and cubes, three candidate inputs x(k), x(k–1), x(k–2), and 14 candidate terms carved to four terms by *ASPN*).....0.195

2–D. For *double* with squares and cubes, two candidate inputs x(k), x(k–1), and eight terms (not carved by *ASPN*).....0.016

Cases 1–D and 2–D are graphed in Figures C.7 and C.8, respectively, where the solid squares represent evaluations at the training-data values of r, while the open squares represent evaluations at r values not used for training.

C.2.2 Dynamic PNN Elements

The work with dynamic PNN nodal elements in this application was very preliminary as it pre-dated creation of the *Dyn3* and *DynNet* algorithms described in this report. Four dynamic nodal elements were synthesized. Using training data for r = 2.5, 2.75, 2.0, 3.25, 3.5, and 3.75, evaluating in the chaotic regime for r = 3.75, the MAE value was:

3. *Modified multilinear double with one self-feedback of* $\hat{x}(k)$, a unit delay in the feedback, one input x(k), and six terms in the core transformation.....0.103

Omitting r = 3.5 and 3.75 data during training, but evaluating at r = 3.75:

4. (Same as 3.).....0.162

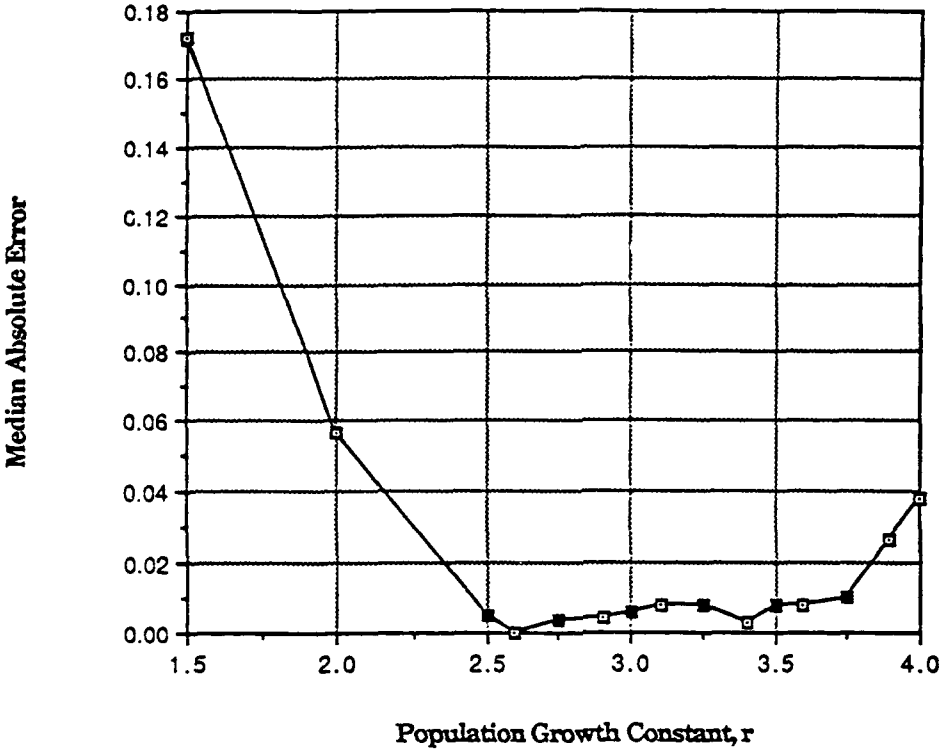


Figure C.7(a): Population Growth Model Results: Median Absolute Error (model synthesized from observations for r = 2.5, 2.75, 3.0, 3.25, 3.5, and 3.75 using a "Double" with squares and cubes — 8 coefficients)

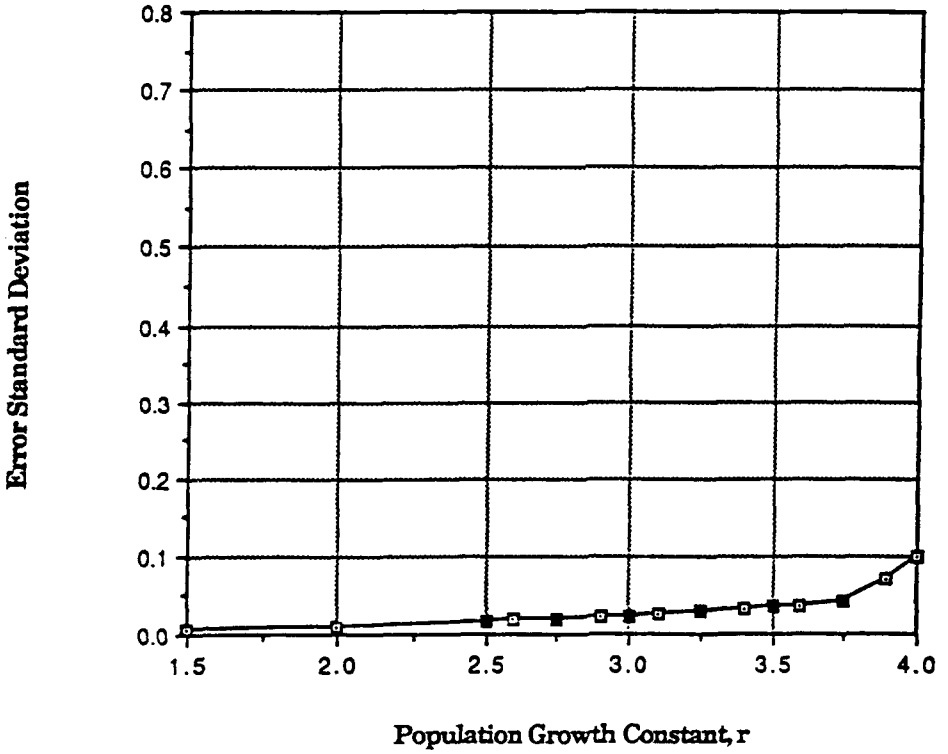


Figure C.7(b): Population Growth Model Results: Error Standard Deviation (model synthesized from observations for r = 2.5, 2.75, 3.0, 3.25, 3.5, and 3.75 using a "Double" with squares and cubes — 8 coefficients)

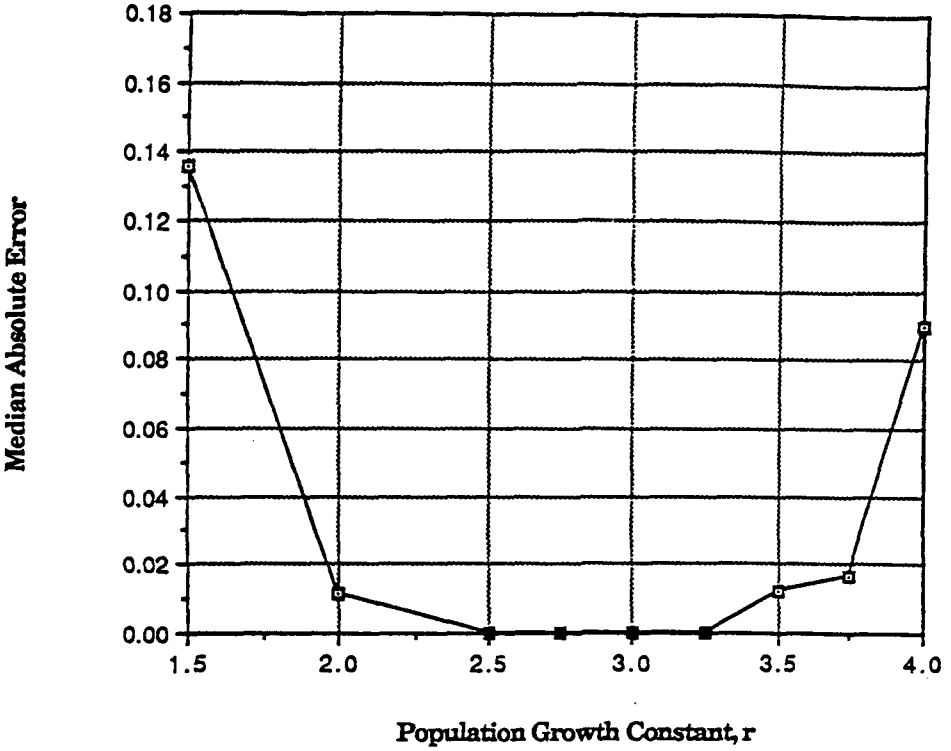


Figure C.8(a): Population Growth Model Results: Median Absolute Error (model synthesized from observations for r = 2.5, 2.75, 3.0, and 3.25 using a "Multilinear Double" — 4 coefficients)

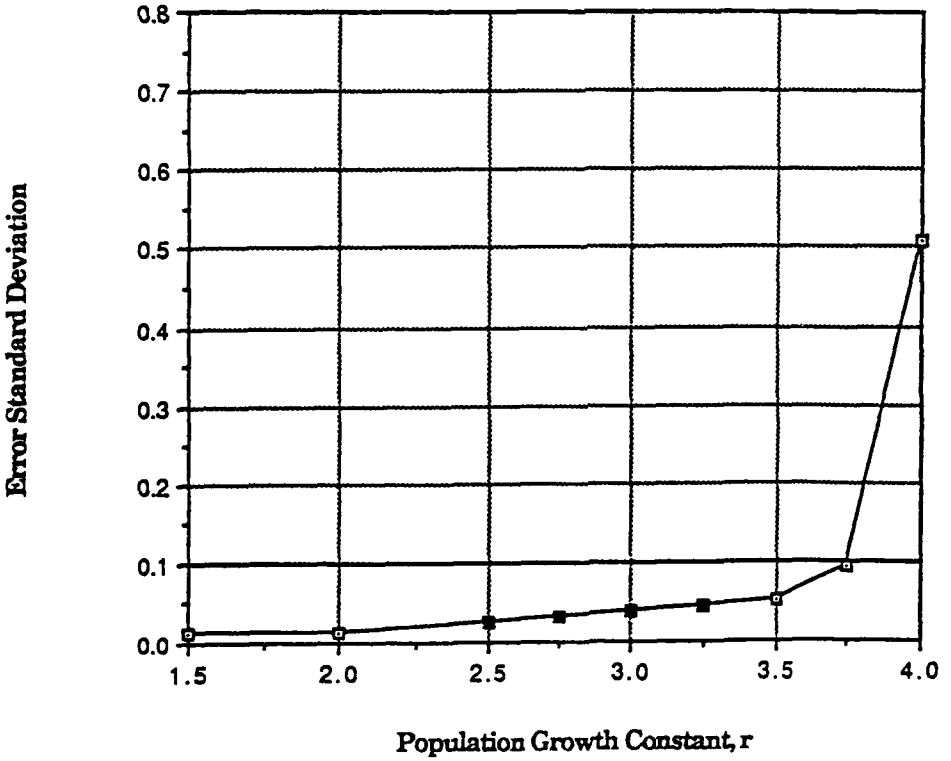


Figure C.8(b): Population Growth Model Results: Error Standard Deviation (model synthesized from observations for r = 2.5, 2.75, 3.0, and 3.25 using a "Multilinear Double" — 4 coefficients)

Next, training on r = 2.5, 2.75, 3.0, 3.25, 3.5, and 3.75, evaluating for r = 3.75:

5. *Modified multilinear triple with two self-feedbacks of $\hat{x}(k)$, one with a unit delay and one with a two-unit delay,* one input x(k), and eight terms in the core transformation.....0.167

Omitting r = 3.5 and 3.75 data during training, but evaluating at r = 3.75:

6. (Same as 5.).....0.145

Note that this last case actually showed *improvement* when training in the chaotic regime was withheld.

It would be desirable to re-visit the dynamic nodal element experiments using *Dyn3* and *DynNet,* so as to obtain a more meaningful comparison between static and dynamic nodal elements used to model the behavior of May's equation.

APPENDIX D

Contract N00014-89-C-0137
Technical Progress Report 2

COMPARATIVE STUDY: STATIC AND DYNAMIC POLYNOMIAL NEURAL NETWORKS FOR REAL-TIME OPTIMUM TPBV GUIDANCE OF A TACTICAL AIR-INTERCEPT MISSILE

Todd M. Nigro
Dean W. Abbott
Roger L. Barron

April 20, 1990

Prepared for:

DEPARTMENT OF THE NAVY
Office of the Chief of Naval Research
Applied Research and Technology Directorate
800 North Quincy Street
Arlington, Virginia 22217-5000

Attention: Dr. Robert J. Hansen, Director

Prepared by:

BARRON ASSOCIATES, INC.
Route 1, Box 159
Stanardsville, Virginia 22973-9511
(804) 985-4400

FOREWORD

This technical progress report has been prepared by Barron Associates, Inc., Stanardsville, Virginia, to document part of the work performed from August 15, 1989, to March 31, 1990, under Contract N00014-89-C-0137 with the Office of the Chief of Naval Research, Applied Research and Technology Directorate, under Dr. Robert J. Hansen, Director. The research is part of the ONR initiative in active control of complex systems.

The authors express their gratitude to Dr. Hansen, ONR Scientific Officer for this project, and to Mr. James G. Smith of ONR for their strong encouragement and support.

Opinions expressed in this report are those of the authors and Barron Associates, Inc., who are solely responsible for its content.

TABLE OF CONTENTS

FIGURES

TABLES

1. INTRODUCTION

Under the subject contract, Barron Associates, Inc. is developing an algorithm for synthesis of dynamic (i.e., reverberant or recurrent) neural networks, i.e., networks having internal feedback paths and time delays, and applying these networks in two principal areas: (a) active control of propulsion systems that are to be operated close to optimum combustion conditions and (b) detection and discrimination of sonar acoustic waveforms. Additionally, to facilitate algorithm development, other applications have been studied from time to time. Thus, Ref. 1 describes an investigation made by Barron Associates into modeling the behavior of the population growth equation, which can exhibit forms of deterministic disorder that include chaotic behavior. It was shown that polynomial neural networks can model the behavior of the population growth equation using data from a regime of simple behavior, then accurately predict population changes in the extinction and chaos regimes.

The present report describes a comparative study of static and dynamic polynomial neural networks; the application chosen for this study is control of a tactical air-intercept missile to achieve real-time, optimum, two-point boundary-value (TPBV) guidance within a vertical plane of motion. It is demonstrated that both static and dynamic networks are feasible for this application and that dynamic networks have performance and robustness superior to that of static networks.

The TPBV missile guidance problem is a good proving ground for comparative analyses of neural networks for active control. Solution of the TBPV problem requires real-time generation of a time-varying steering function for the missile such that:

(1) the desired control path is followed closely,

(2) boundary conditions on the trajectory are accurately fulfilled,

(3) the guidance system accommodates large variations in intercept conditions,

(4) induced drag and kinetic energy losses due to maneuvering of the missile are minimized, and

(5) the system is robust in the presence of uncertainties in measurements of the missile states.

This report outlines a nonlinear two-degree-of-freedom guidance problem, lists the derived governing equations for optimality using the calculus of variations, describes the procedure for establishing a representative data base (field) of optimum trajectories, summarizes the steering-function models of these trajectories created as static and dynamic neural networks, and assesses and compares the performance and robustness of both types of networks.

2. MISSION PROFILE, ASSUMED MISSILE DYNAMICS, AND THE VARIATIONAL OPTIMUM SOLUTION

2.1 Mission Profile

Consider an air-intercept missile that is guided in a vertical plane to a predicted intercept point (PIP) for a moving target. The PIP comprises predicted intercept time, predicted intercept downrange position, and predicted intercept height. Let the initial conditions be identical for all trajectories of the missile and assume a constant speed for the target. Also, assume a horizontal movement of the target at an altitude of 80,000 ft., as shown in Figure 1, below:

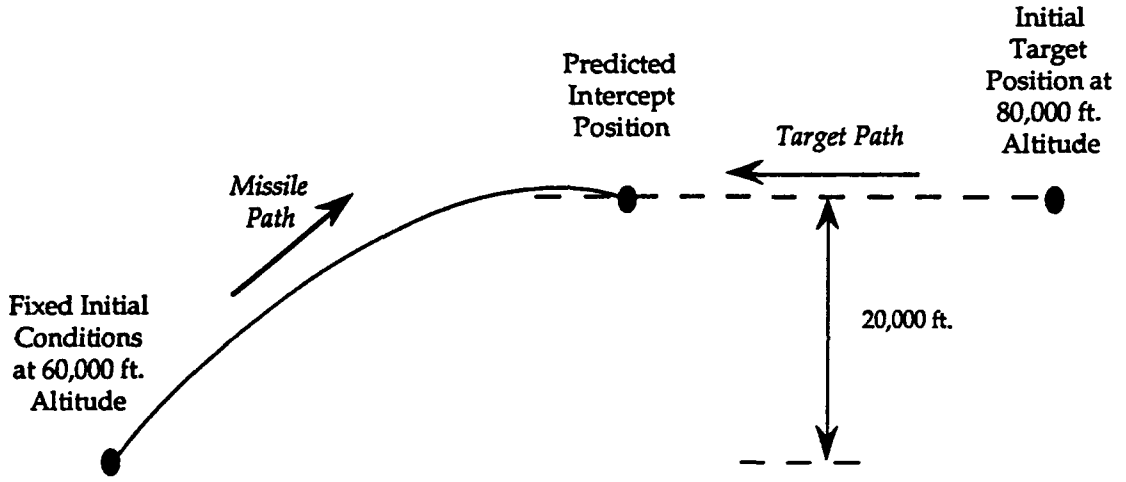


Figure 1: Mission Profile

2.2 Assumed Missile Dynamics

Let the missile and PIP positions be defined in terms of a horizontal axis, $\vec{N}$ (i.e., North), and an orthogonal vertical axis, $\vec{H}$. These axes are assumed to be fixed in inertial space, i.e., for present purposes it is assumed that the Earth is flat and non-rotating. Geometric altitude, H, is measured upward from mean sea level along $\vec{H}$. Assume that the missile is unthrusted, has constant mass, and that the time constant of the missile rotational responses to pitch-attitude commands is negligibly small. Assume that, over the altitude band of the guided flight of the missile, the atmosphere density decreases exponentially with altitude and the acceleration of gravity decreases gradually with geopotential height. Assume, moreover, that the aerodynamic characteristics of the missile are independent of Mach number.

Pursuant to these simplifying assumptions, the dynamic and kinematic equations of motion of the missile become:

$$f_1 \equiv m\dot{V} + mg\sin\gamma + qS\,C_D = 0 \qquad 2:1$$

$$f_2 \equiv 0 \qquad 2:2$$

$$f_3 \equiv mV\dot{\gamma} + mg\cos\gamma - qS\,C_L = 0 \qquad 2:3$$

$$f_4 \equiv 0 \qquad 2:4$$

$$f_5 \equiv \dot{N} - V\cos\gamma = 0 \qquad 2:5$$

$$f_6 \equiv \dot{H} - V\sin\gamma = 0 \qquad 2:6$$

where:

m = missile mass

g = acceleration of gravity = $g_0 R_e/(R_e + H)$

g_0 = acceleration of gravity at mean sea level

R_e = Earth radius at reference latitude

V = missile speed = $(\dot{N}^2 + \dot{H}^2)^{1/2}$

N = missile downrange (North) position

H = missile altitude (Height)

γ = missile flight path angle = arctan $(\dot{H}/\dot{N})$

q = missile dynamic pressure = $\frac{1}{2}\rho V^2$

ρ = atmosphere density = $\rho_b \exp\left(\frac{H_b - H_p}{H_s}\right)$

ρ_b = atmosphere density at base altitude

H_b = base altitude

H_p = geopotential height = $\left(\frac{R_e}{R_e + H}\right)H$

H_s = atmosphere scale height

S = missile reference area

and, for a small aerodynamic angle of attack, α, which is defined as the missile pitch attitude minus the missile flight path angle:

C_D = missile drag coefficient = $C_{D0} + K C_L^2 = C_A + C_{N\alpha} \cdot \alpha^2$

C_{D0} = coefficient of missile drag at zero lift

K = coefficient of missile drag induced by lift = $C_{N\alpha}/(C_{N\alpha} - C_A)^2$

C_L = missile lift coefficient = $(-C_A + C_{N\alpha})\,\alpha$

C_A = missile axial force coefficient (= C_{D0} for small α)

$C_{N\alpha}$ = partial derivative of missile normal force coefficient (C_N) with respect to α = $K C_{L\alpha}^2$

It may be readily shown (Ref. 2) that the maximum lift-to-drag ratio of the missile is

$$(L/D)_{max} = \frac{1}{2}\left(\sqrt{\frac{C_{N\alpha}}{C_A}} - \sqrt{\frac{C_A}{C_{N\alpha}}}\right)$$

and that:

$$\alpha_{(L/D)max} = \sqrt{\frac{C_A}{C_{N\alpha}}}$$

$$C_{L(L/D)max} = \alpha_{(L/D)max}\,(C_{N\alpha} - C_A)$$

Assuming:

C_A = 0.4

$C_{N\alpha}$ = 7.2/rad.

it follows that $(L/D)_{max} = 2.0$, $K = 0.1557$, $\alpha_{(L/D)max} = 13.5$ deg., and $C_{L(L/D)max} = 1.603$.

The following hypothetical properties of the missile are also assumed:

Weight = 1,000 lb., whence m = 31.08 slugs (453.6 kg.)

$S = 1.0\ ft.^2$ $(0.0929\ m.^2)$

The release (initial) conditions for all flights of the missile are assumed to be:

$N(t_0) = N_0 = 0$

$H(t_0) = H_0 = 60{,}000$ ft. (18,288 m.)

$V(t_0) = V_0 = 2{,}239$ ft./s. (682.4 m./s.)

$\gamma(t_0) = \gamma_0 = 30.0$ deg.

The geophysical parameter values assumed are:

$g_0 = 32.199\ ft./s.^2$ $(9.814\ m./s.^2)$

$R_e = 20{,}844{,}531.5$ ft. (6,356,785.2 m.)

$\rho_b = 0.002{,}376{,}92\ slug/ft.^3$ $(1.225\ kg./m.^3)$

$H_b = 0$

$H_s = 32{,}111$ ft. (7,044 m.)

2.3 Point of Intercept Calculation

Computation of the PIP is external to the guidance system. However, the PIP is a function of the expected time of flight of the missile, and therefore the PIP is designated for a specific final (intercept) time t_f. At intercept, $N(t_f) = N_T(t_f)$ and $H(t_f) = H_T(t_f)$, where the subscript "T" denotes the target. For simplicity, it is assumed that the target moves with a constant negative $\dot{N}_T$ at a constant altitude of 80,000 ft., whence

$$N_T(t_f) = N_T(t_0) + (t_f - t_0)\dot{N}_T$$

and

$$H_T(t_f) = H_T(t_0) = 80{,}000 \text{ ft.} \quad (24{,}384 \text{ m.})$$

An example PIP calculation is shown below:

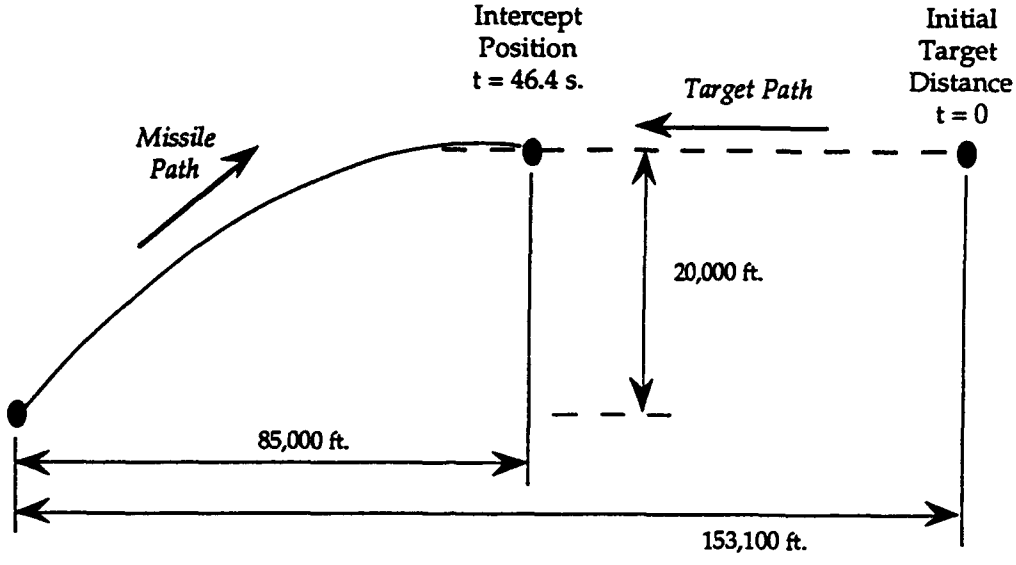


Figure 2: Example PIP Calculation

The above assumptions regarding target motion are not limiting, because, in principle, the target path to the PIP may be any trajectory for which a valid prediction of the intercept point can be made. In general, the intercept prediction involves iteration, although it should be possible to employ a neural network to compute the PIP (or a reasonable first iterate) from knowledge of target and missile states at t_0. A few examples of PIP and time of flight calculations are presented in Table 1, below:

Table 1 : Example PIP Calculations, Assuming Constant Target Velocity

Initial Target Position, ft. $N_T(t_0)$	Target Velocity, ft./s. V_T	Intercept Position, ft $N_T(t_f)$	Time of Flight, s. t_f
134,300	1,467	75,000	40.4
143,600	1,467	80,000	43.4
153,100	1,467	85,000	46.4
162,800	1,467	90,000	49.6
173,000	1,467	95,000	53.1

2.4 Variational Optimum Solution

Let us take as the criterion of guidance system performance the following functional that is to be minimized

$$I = -C_0\, mV^2(t_f)/2 + \int_{t_0}^{t_f} (C_1\,\alpha^2 + \underline{\lambda}^T \underline{f})\, dt \qquad 2:7$$

in which C_0 and C_1 are constants (Lagrange multipliers), $\underline{\lambda}^T = [\lambda_1, ..., \lambda_6]$, and $\underline{f}$ has the components $f_1, ..., f_6$ from Eqs. 2:1 – 2:6. The C_0 term (outside the integrand) introduces the final kinetic energy of the missile, which is to be maximized by minimizing its negative. The C_1 term (within the integrand) provides a constraint on "control effort", defined as the time integral of α^2. (At a constant speed and altitude, control effort would be proportional to induced drag impulse.)

It is convenient to transform Eq. 2:7 into its equivalent Lagrange representation

$$I = -C_0\, mV^2(t_0)/2 + \int_{t_0}^{t_f} F\, dt$$

where F is the augmented integrand (Hamiltonian)

$$F \equiv - C_0 m V\dot{V} + C_1 \alpha^2 + \underline{\lambda}^T \underline{f} \qquad 2:8$$

Because $V(t_0)$ is prescribed at the initial boundary, it is ignored in the optimization, and we seek minimization of the functional

$$I' = I + C_0 mV^2(t_0)/2 = \int_{t_0}^{t_f} F\,dt \qquad 2:9$$

between the fixed limits (prescribed end times) t_0 and t_f.

The variational freedoms for this optimization are:

$\alpha(t)$, i.e., the steering function

$V(t), \gamma(t), N(t), H(t),$ i.e., the missile states

The boundary conditions are:

prescribed boundary values

$t_0, N_0, H_0, V_0, \gamma_0$ (whence $\delta t_0 = \delta N_0 = \delta H_0 = \delta V_0 = \delta\gamma_0 = 0$)

t_f, N_f, H_f (whence $\delta t_f = \delta N_f = \delta H_f = 0$)

free boundary values

α_0 (whence $\delta\alpha_0$ is variable)

$\alpha_f, V_f, \alpha_f, \gamma_f$ (whence $\delta\alpha_f, \delta V_f, \delta\gamma_f$ are variable)

From the theory of the calculus of variations (Ref. 2), the following conditions apply:

first-variation conditions

$F_\alpha = 0$ (since $F_{\dot{\alpha}} = 0$) (Euler-Lagrange)

$F_V = F_{\dot{V}}, F_\gamma = F_{\dot{\gamma}}, F_N = F_{\dot{N}}, F_H = F_{\dot{H}}$ (Euler-Lagrange)

$$(F_{\dot{V}})_f \delta V_f + (F_{\dot{\gamma}})_f \delta\gamma_f + (F_{\dot{N}})_f \delta N_f + (F_{\dot{H}})_f \delta H_f$$
$$+ [F - (F_{\dot{V}})\dot{V} - (F_{\dot{\gamma}})\dot{\gamma} - (F_{\dot{N}})\dot{N} - (F_{\dot{H}})\dot{H}]_f \delta t_f = 0 \qquad \text{(transversality)}$$

i.e., $(F_{\dot{V}})_f = 0$; $(F_{\dot{\gamma}})_f = 0$

second-variation condition

$$F_{\alpha\alpha} > 0 \qquad \text{(Legendre)}$$

From Eq. 2:8 (the Hamiltonian) and the above conditions, one obtains the *optimum steering function (command equation)*

$$\alpha = \frac{\lambda_3 qS (C_{N\alpha} - C_A)}{2 (C_1 + \lambda_1 qS C_{N\alpha})} \qquad 2:10$$

in which (Legendre condition) it is required that

$$C_1 + \lambda_1 q S C_{N\alpha} > 0 \qquad 2:11$$

and the *co-state differential equations*:

$$\dot{\lambda}_1 m = q_V S (\lambda_1 C_D - \lambda_3 C_L) + \lambda_3 m\dot{\gamma} - \lambda_5 \cos\gamma - \lambda_6 \sin\gamma \qquad 2:12$$

$$\dot{\lambda}_3 mV = -\lambda_3 m (g \sin\gamma + \dot{V}) + \lambda_1 mg \cos\gamma + \lambda_5 V \sin\gamma - \lambda_6 V \cos\gamma \qquad 2:13$$

$$\dot{\lambda}_5 = 0 \qquad 2:14$$

$$\dot{\lambda}_6 = q_H S (\lambda_1 C_D - \lambda_3 C_L) \qquad 2:15$$

In Eq. 2:12

$$q_V = \rho V \qquad 2:16$$

and in Eq. 2:15

$$q_H = -\frac{q}{H_s}(H_p)_H = -\frac{q}{H_s}\left(\frac{H_p}{H}\right)^2 \qquad 2:17$$

The necessary *final* values (from the transversality conditions) are:

$$\lambda_1 (t_f) = C_0 V_f \qquad 2:18$$

$$\lambda_3 (t_f) = 0 \qquad 2:19$$

There are no transversality conditions to be satisfied at t_0.

3. DATA BASE GENERATION

To generate an optimum trajectory for the polynomial neural network synthesis data base, the missile equations of motion (2:1 - 2:6) and the co-state differential equations (2:12 - 2:15) are integrated *backward* in time under control of the steering function (Eq. 2:10) from the PIP using $\lambda_1(t_f)$ and $\lambda_3(t_f)$ from Eqs. 2:18 and 2:19, the prescribed values of N_f, H_f, and the desired values of V_f, γ_f. A numerical search (called the first-stage search) is employed, adjusting $\lambda_5(t_f)$ and $\lambda_6(t_f)$, to obtain the requisite H_0 and γ_0. Any trajectory for which $|\alpha|$ and/or $|\gamma|$ exceed maximum values should be eliminated from the data base, as should any trajectory that violates the Legendre test, 2:11. When this first-stage search has converged, $\lambda_5(t_f)$ and $\lambda_6(t_f)$ are adjusted until a maximum value of V_0 is obtained while satisfying the prescribed values of H_0 and γ_0. This is tantamount to minimizing terminal kinetic energy once V_0 is fixed. Once V_0 has been maximized, the time of flight, the end states, and the parameters of the extremal arc are recorded:

$$(t_f - t_0), N_0, H_0, V_0, \gamma_0, N_f, H_f, V_f, \gamma_f; C_0, C_1, \lambda_1(t_0), \lambda_3(t_0), \lambda_5(t_0), \lambda_6(t_0).$$

Note that one may keep C_0 fixed at unity without loss of generality. One may choose instead to keep $\lambda_5(t_f)$ or $\lambda_6(t_f)$ fixed, and vary all other Lagrange multipliers with respect to the one fixed multiplier. Note also that downrange distance is not directly constrained, but can only achieve values that are consistent with extremization of the performance criterion.

A somewhat simpler search is obtained by freezing γ_f at a prescribed value. In this case $\lambda_3(t_f)$ is not constrained to be zero by the transversality conditions (Eq. 2:19), but rather participates in the first-stage search along with $\lambda_5(t_f)$ and $\lambda_6(t_f)$, and convergence to N_0, H_0, γ_0 can be obtained in the first stage. It is then easier to adjust $\lambda_3(t_f)$, $\lambda_5(t_f)$, and $\lambda_6(t_f)$ while seeking a maximum V_0. Prescribed γ_f values arise operationally because of anti-jam or weapons-effects considerations.

A data base is created for given initial states by extending the solution space to a field of PIP conditions, either by following the same procedure as for the first entry, or by changing the $\lambda(t_0)$ component values. In the present work, $\lambda_5(t_0)$ was systematically changed to produce a family of intercept points, i.e., $N_T(t_f)$ values, at 80,000 ft. altitude. A data base of 388 optimum trajectories was generated, each trajectory corresponding to a different $N_T(t_f)$ within a large intercept window at $H_T(t_f)$ = 80,000 ft. The smallest $N_T(t_f)$ value was 72,062 ft. (21,964 m.), for which t_f was 38.8 s. The largest $N_T(t_f)$ value was 97,518 ft. (29,723 m.), for which t_f was 55.1 s. Note that these trajectories are extremals in that they satisfy the Euler-Lagrange necessary conditions, but do not necessarily satisfy the transversality conditions at t_f. In general, the transversality conditions are weaker than the Euler-Lagrange conditions, and thus trajectory optimization was not substantially degraded. The

benefit is that data base generation is considerably simplified when the transversality conditions are not stringently applied.

The data base entries can contain information for each trajectory about its initial co-state values, $\lambda_1(t_0)$, $\lambda_3(t_0)$, $\lambda_5(t_0)$, $\lambda_6(t_0)$, and equivalent information about its steering function history, $\alpha(t_0)$, $\alpha(t_1)$, ..., $\alpha(t_f)$. These equivalent steering function histories are used for syntheses of both static and dynamic networks. The two synthesis procedures are discussed in Section 5.

In summary, the method used to generate a data base in the present example was:

1. Set desired H_f = 80,000 ft., V_f = 1,800 ft./sec., and γ_f = –10 deg.

2. Select $C_0 = 1$, $C_1 = 1$ (thus C_1 did not contribute significantly to the cost functional).

3. Compute $\lambda_1(t_f)$ based on V_f from the transversality condition.

4. Select arbitrary $\lambda_3(t_f)$, $\lambda_5(t_f)$, and $\lambda_6(t_f)$.

5. Integrate differential equations backward until H = 60,000 ft., denoting the time at this altitude as t_0.

6a. Note γ_0. If γ_0 is within specified tolerance (30 deg., ± 3 deg.), note V_0, and search on $\lambda_3(t_f)$, $\lambda_5(t_f)$, and $\lambda_6(t_f)$ for maximum V_0. Then go to Step 7.

6b. If γ_0 is not within specified tolerance, go to Step 4 and repeat until the requirement on γ_0 is satisfied.

7. Create a family of reverse-integration trajectories, each for the same initial states but a different N_f. (Each individual trajectory is obtained by following steps 1 - 6.)

   or

   Create a family of extremal trajectories by varying one or more of the initializing Lagrange multipliers and noting the intercept position.

This method may be extended to the generation of a field of optimal trajectories for a variety of initial states and a variety of final states.

4. OVERVIEW OF DYNAMIC NETWORKS

Dynamic polynomial neural networks (PNNs) can compute a time-varying prediction or control signal given a static input and provide a phase-shift operator between time-varying inputs and time-varying prediction or control signals. In addition, they can employ all of the available training data, so there is no need to interpolate between discrete-time outputs. Thus, smoother on-line operation results with dynamic neural networks. As we will show, for a given application, dynamic PNNs are simpler, resulting in less probability of overfit and an increase in accuracy and robustness. However, there is a price to pay: (1) there is an initialization requirement and (2) the synthesis of dynamic networks requires iteration. Moreover, there is presently no synthesis algorithm that efficiently learns the *structure* of dynamic networks.

The following definitions are used:

Static Node: a node with no *internal* self-feedbacks or time delays.

Dynamic Node: a node with self-feedbacks and/or input time delays *internal* to the node. Note that self-feedback connections are implemented *with* time delays in the self-feedback paths. Otherwise, the node is tantamount to a node that uses division, i.e., a rational function.

Non-Memory Feedforward (NMFF) Node: a static node.

Memory Feedforward (MFF) Node: a dynamic node with internal time delays of the inputs, but no self-feedback connections.

Memory Feedback (MFB) Node: a dynamic node with internal self-feedback connections, but not necessarily with internal time delays of the inputs.

Shift Register: a device in the dynamic node that stores previous input or output estimates.

4.1 Initialization

Prior to the time a dynamic network is first interrogated, time t_1, initialization with previous values of inputs and/or the output to estimate the output value is required. This does not pose a problem if no estimate of the output is needed prior to time t_1, i.e., before the shift registers of the dynamic node are filled. For example, in signal processing applications, it may be sufficient that the first output estimate be available only after enough of the signal has been seen (i.e.,

after the dynamic network shift registers have been filled), and therefore, prior to time t_1, no output estimates are needed. In other applications, however, such as control problems, output estimates must be generated beginning at time t_0, and therefore an additional step must be taken to produce output estimates prior to time t_1.

When output estimates are required before time t_1, at least one additional static or dynamic startup (initialization) node must be synthesized to provide the needed output estimates. These startup nodes use short-term memory, i.e., they contain only one or two self-feedback paths. With this short-term memory dynamic node, the network contains at least three nodes: a static node to initialize the system at t_0, a startup dynamic node with one self-feedback path to provide output estimates from time t_0 to t_1, and a primary dynamic node to operate from time t_1 to the end of the control interval.

4.2 Synthesis Methods for Dynamic Networks

There are two types of algorithms with which to train dynamic PNNs: *equation-error methods* and *output-error methods*.

Equation-Error Methods: the output estimate is found as shown in Figure 3, using previous output values obtained from the data base, not from previous output estimates.

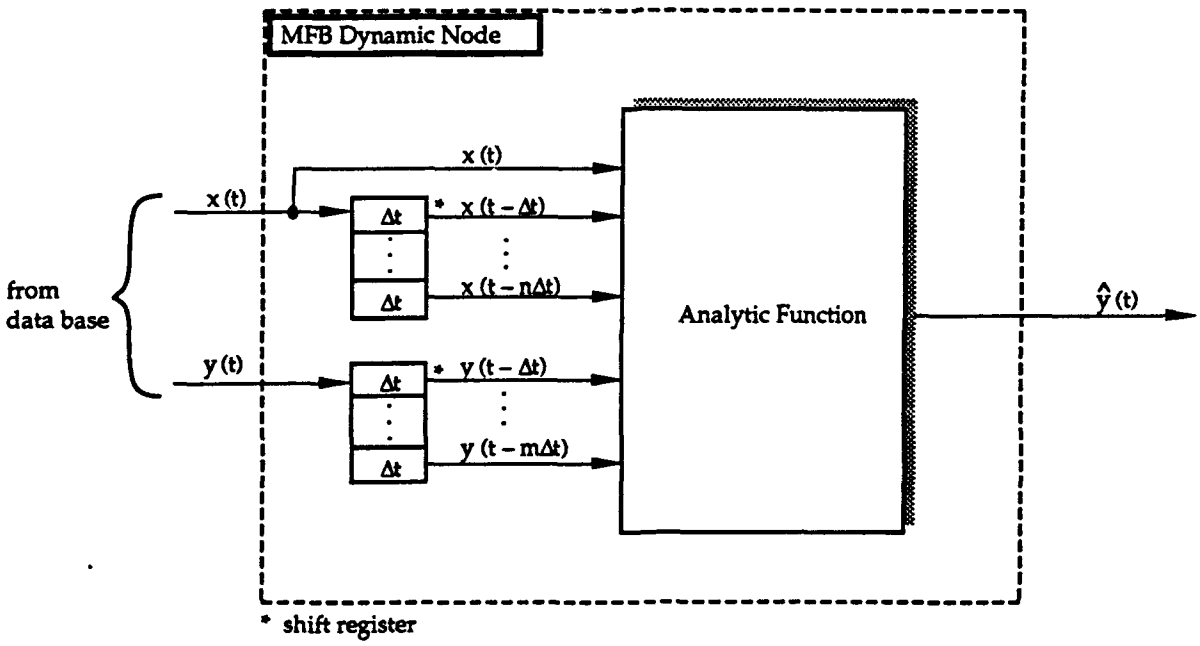


Figure 3: Equation-Error Method for Synthesizing Dynamic Nodes

The equation-error method does not require iteration for synthesis, thus, in general, it is computationally simpler than output-error methods. A fully-developed algorithm, ASPN-II (Algorithm for Synthesis of Polynomial Networks, see Appendix), implements the equation-error method using a general *linear* least-squares fitting algorithm, i.e., the model equation is linear with respect to the model coefficients. However, a large amount of pre-processing may be required to manipulate the data base into a usable form and equation-error training is only appropriate for certain kinds of problems. Equation-error training is analogous to a finite impulse response (FIR) technique.

Output-Error Method: the output estimate is found as shown in Figure 4, using previous output estimates obtained by implementing the MFB node.

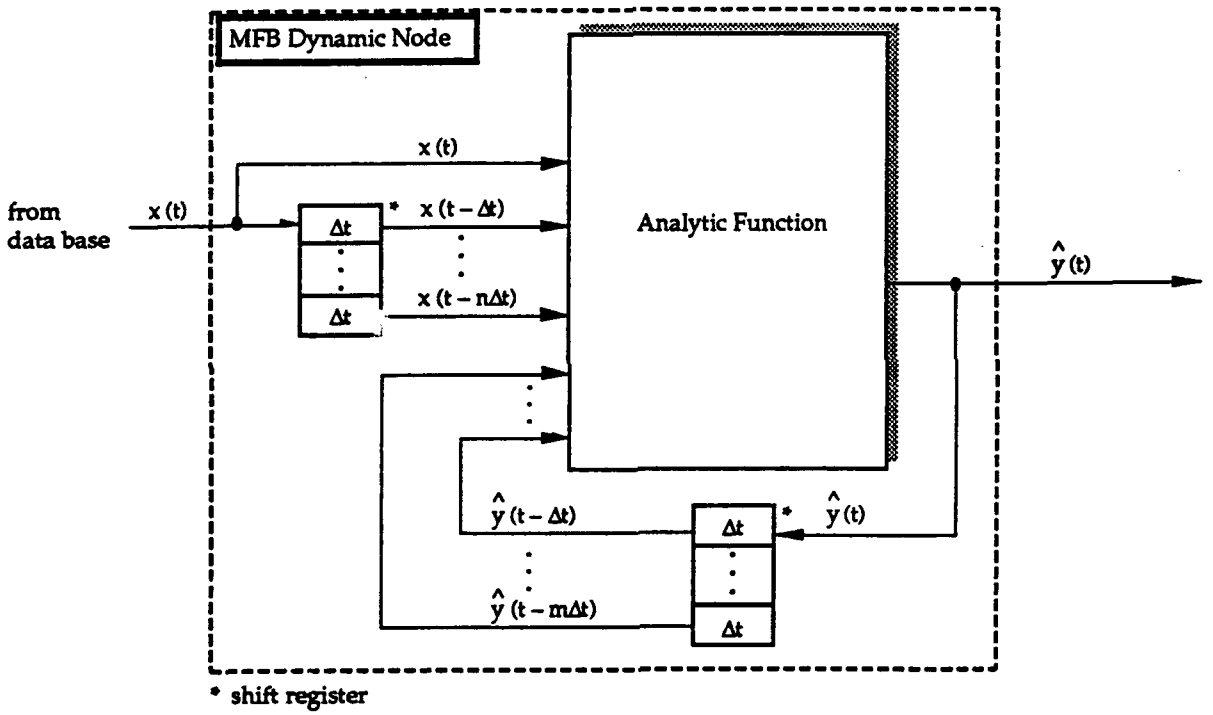


Figure 4: Output-Error Method for Synthesizing Dynamic Nodes

To find the model coefficients, a general *nonlinear* least squares fitting algorithm is used, i.e., the model equation is nonlinear with respect to the model coefficients. The algorithm conducts an iterative search, and therefore requires longer for a solution to be obtained than equation-error methods. Output-error training is analogous to an infinite impulse response (IIR) approach.

The preferred method of training depends on the application in which the network is to be used. Equation-error syntheses are best suited for MFB predictors (Figure 5) because output estimates do not directly influence future network inputs and outputs. Thus, output estimates should not affect future output estimates in synthesis.

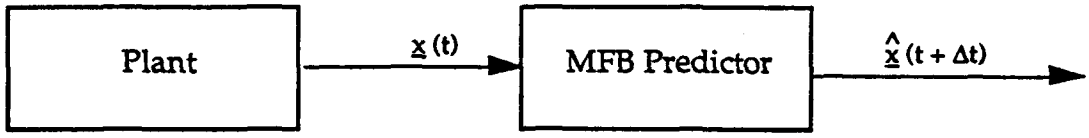


Figure 5: MFB Network Synthesized as a Predictor

On the other hand, when a controller is desired (Figure 6), output estimates directly influence future MFB network inputs and outputs; therefore, in synthesis, current output estimates should be reflected in future output estimates.

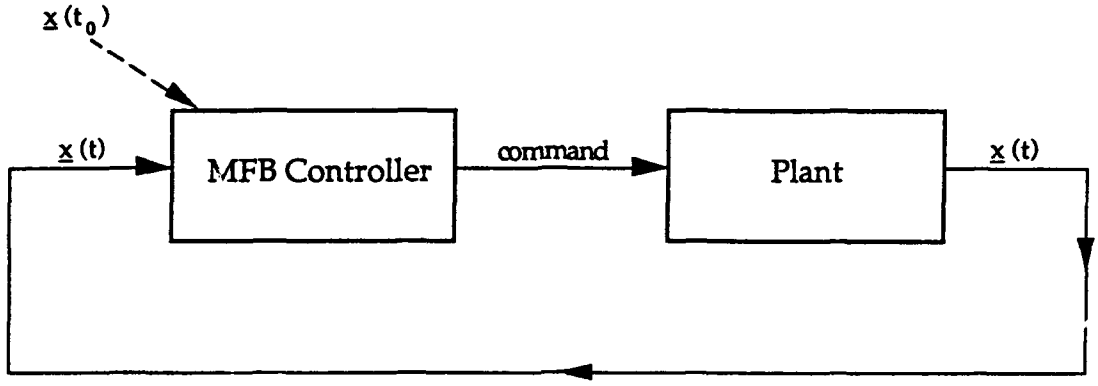


Figure 6: MFB Network Synthesized as a Controller

Table 2 summarizes the off-line synthesis methods and their corresponding synthesis algorithms and applications. The algorithms used for equation-error and output-error syntheses are, respectively, generalized linear least-squares (Ref. 3) and a modified Levenberg-Marquardt (Ref. 4) search.

Table 2: Off-Line Synthesis Methods for Prediction and Control

Node Type	Error Type	Iteration	Preferred Synthesis Algorithm	Application for Which Best Suited
NMFF	Equation Error	No	Least Squares	Prediction/Control
MFF	Equation Error	No	Least Squares	Prediction/Control
MFB	Equation Error	No	Least Squares	Prediction
	Output Error	Yes	Levenberg-Marquardt	Control

5. STATIC AND DYNAMIC NETWORK SYSTEM SYNTHESIS PROCEDURES FOR THE TPBV GUIDANCE SOLUTION

As discussed earlier, a data base of 388 optimum trajectories was generated, each corresponding to a different intercept point. The data for each trajectory included the intercept position and time-histories of the missile steering function, α, position, speed, and flight path angle. The time of flight (t_f) was different for each trajectory, 38.8 s. $\leq t_f \leq$ 55.1 s. To provide a simplification and allow a region in which to test time extrapolation capabilities of the network designs, each trajectory in the data base was truncated at 38.5 s.

The initial conditions were identical for all trajectories created; therefore, an initialization node at t_0 was not required. Since the dynamic network required a startup node, an appropriate data base was needed over the startup interval. Thus, the original data base was divided into two parts with different sampling rates covering different portions of the trajectories. The *startup* data base contained the first 10 s. with a sampling rate of 0.01 s. With a sampling rate of 0.5 s., the *primary* data base consisted of the entire trajectory up to 38.5 s. Due to the comparative nature of this investigation, the static and dynamic network designs each used both the startup and primary data bases in training. These data bases included a set of candidate inputs for the synthesis process. Although a static startup node is not, in general, a requirement for static designs, performance was improved with its inclusion.

The output of each of the models was $\alpha(t)$, the aerodynamic angle of attack (also called the "steering function"). Figure 7 below, shows $\alpha(t)$ for three optimum trajectories. It should be remembered that only the first 38.5 of each trajectory was used in the training of the primary network.

5.1 Static Network Synthesis Procedures

Using ASPN-II (the Algorithm for Synthesis of Polynomial Networks), startup and primary networks were synthesized. ASPN-II is based on an information theoretic criterion called predicted squared error (PSE). The model fitting score is penalized using a complexity penalty to avoid overfitting the data. PSE, in its simplest form, is written (Ref. 5)

$$PSE = FSE + \frac{2K}{N}\sigma_p^2$$

where FSE is the fitting squared error, K is the number of coefficients in the model, N is the number of data vectors used for the model, and σ_p^2 is an *a priori* estimate of the error variance of the model, independent of the model structure being considered. The second term in the above equation is commonly called the

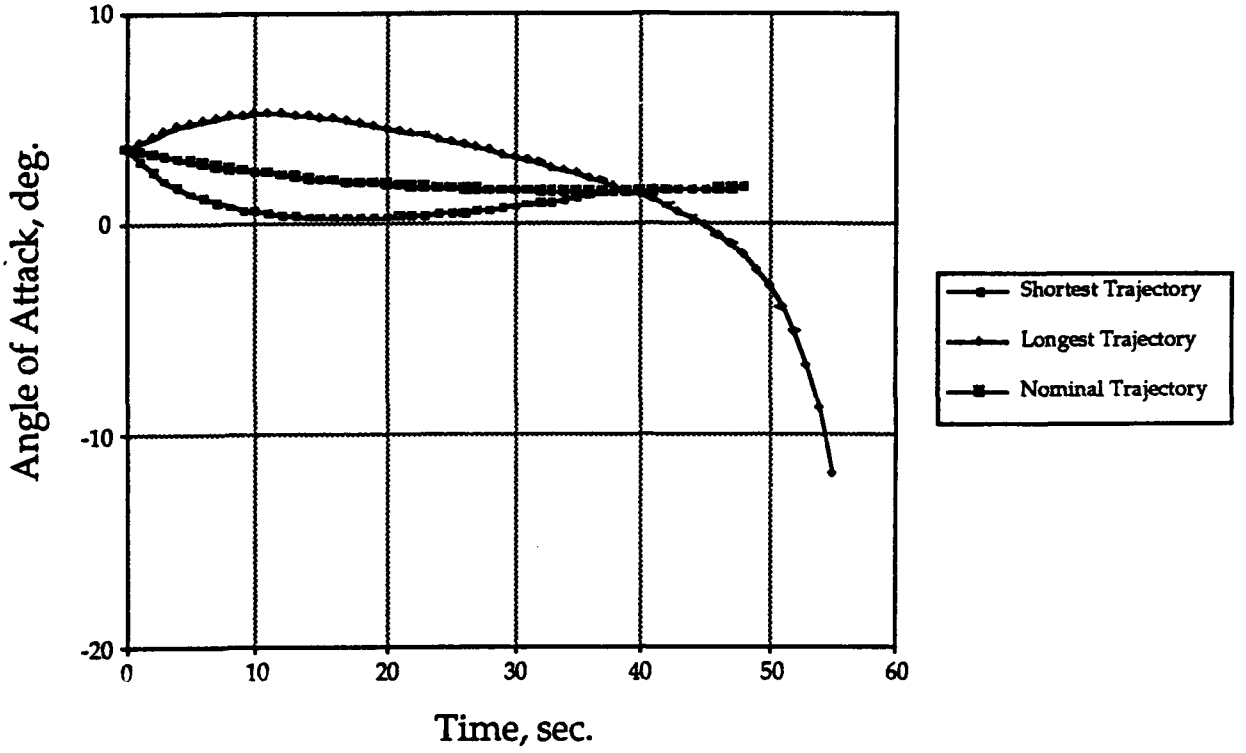


Figure 7: Angle of Attack, α, vs. Time for Three Optimum Trajectories

complexity penalty. The PSE criterion governs selection of nodes in the model, the carving of unproductive terms in the model, and the cessation of network growth at an optimum level of complexity.

The static networks used only "time-now" values of the observables: missile speed, missile downrange distance travelled, predicted intercept downrange distance, and the cosine of flight path angle. Training data for the static startup node consisted of 25 trajectories with downrange intercept position distributed uniformly. The startup network, shown in Figure 8 below, consisted of two layers, with a total of 17 terms, and was used as the controller for the first 8 s. of flight, after which the primary network was implemented.

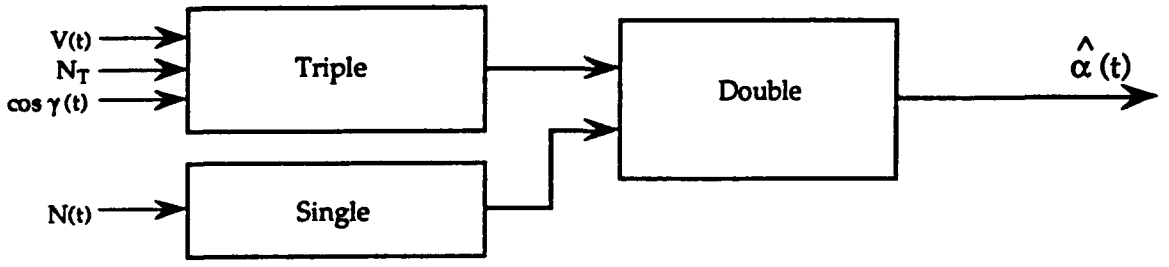


Figure 8: Startup Static Network

The primary network was trained from 35 exemplars with downrange intercept position distributed uniformly. The two-layer network contained 26 terms as shown in Figure 9 below.

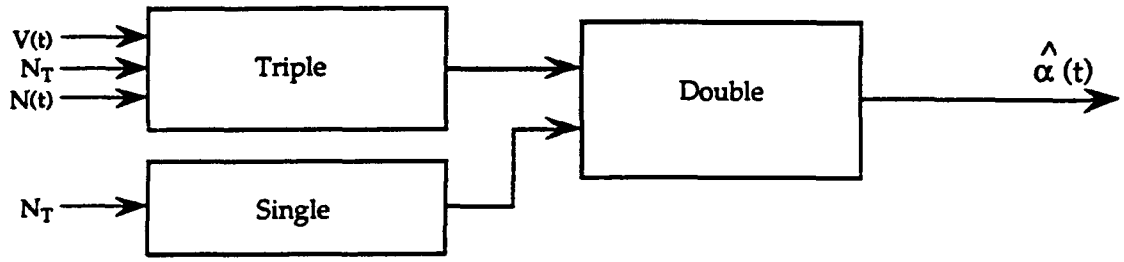


Figure 9: Primary Static Network

5.2 Dynamic Network Synthesis

An algorithm has not yet been developed that is capable of finding the best structure for dynamic controller networks. The network structure has several characteristics that must be determined: the number of internal self-feedback paths, the maximum delay in time, and the relevant inputs. Additionally, a trade-off between network complexity, fitting accuracy, and robustness must be considered.

Figure 10, below, shows the improvement of the score (fitting squared error) as the number of terms in the model is increased for a representative network synthesis run.

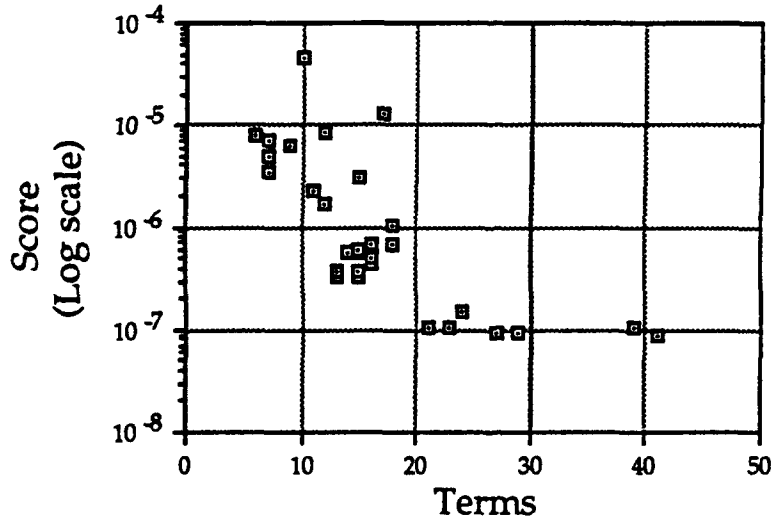


Figure 10: Score vs. Degrees of Freedom for Different Structures

The search for the structure that yielded the best trade-off between accuracy of fit and complexity was aided by ASPN-II. An equation-error network was synthesized using ASPN-II, with the resulting inputs and time delays noted. Using the same inputs and variations on them, the dynamic node was synthesized using *dyn2.*, an algorithm that synthesizes nodes using output-error. The equation-error solution by ASPN-II was found to be an efficient initial starting point for the Levenberg-Marquardt search employed in *dyn2.*

The resulting startup network consisted of ten terms, with missile downrange distance, altitude, speed, and the intercept downrange distance as inputs. There was only one self-feedback path associated with this network, thus, the maximum delay was 0.01 s. The primary network consisted of 14 terms, with inputs of missile speed, downrange distance, height, and the intercept point. Four feedback paths were used: one, four, 12, and 16 time-steps back. The maximum delay, therefore, was 8 s. The figure below shows the inner structure of the primary dynamic network.

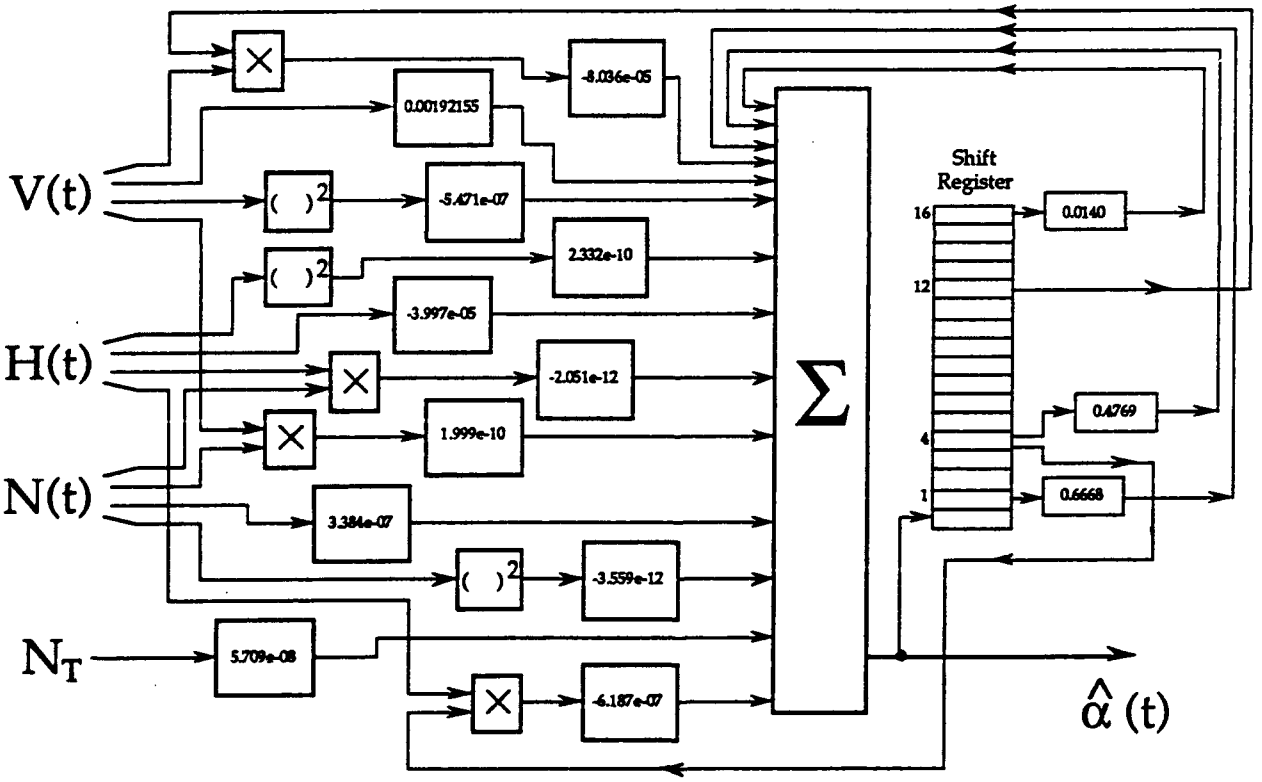


Figure 11: Primary Dynamic Network, Microscopic View

6. STATIC AND DYNAMIC NETWORK PERFORMANCE IN THE TPBV GUIDANCE SOLUTION

In this section, selected performance and robustness tests are described and results compared for the static and dynamic designs. It should be noted that all of these tests were done open-loop, i.e., after initialization, so there was no information available for use in the guidance law reflecting its performance (such as predicted miss). The statistics presented compare optimum trajectories stored in the training data bases with results from implementation of the guidance law. With the exception of the "extrapolation-in-time" test, all results were calculated for the first 38.5 s. of flight. The design region for all tests is defined to be that for which the intercept position is between 75,000 and 95,000 ft. downrange.

All results presented are based on evaluation data (target positions not included in the training data base). For each test, more than 100 trajectories were flown.

6.1 Accuracies within the Design Region without Measurement Uncertainties

The median alpha RMS error in degrees was 0.054 and 0.014 for the static and dynamic neural network designs, respectively: an improvement ratio of approximately four to one (dynamic vs. static), as shown in Table 3 and Figure 12. Also, the median terminal position error (miss distance) was 5.9 and 3.8 ft. for the static and dynamic networks, respectively.

Table 3: Steering Function and Trajectory Accuracies: Operating in Design Region, without Measurement Uncertainties

	Alpha RMS Error, deg. *		
	Static	Dynamic	S/D Ratio
Mean	0.060	0.015	4.0
Std. Dev.	0.053	0.011	4.8
Max	0.105	0.026	4.0
Median	**0.054**	**0.014**	**3.9**

	Miss Distance, ft.		
	Static	Dynamic	S/D Ratio
Mean	5.9	4.0	1.5
Std. Dev.	4.1	2.5	1.6
Max	13.8	9.9	1.4
Median	**5.9**	**3.8**	**1.55**

* alpha = aerodynamic angle of attack (steering function)

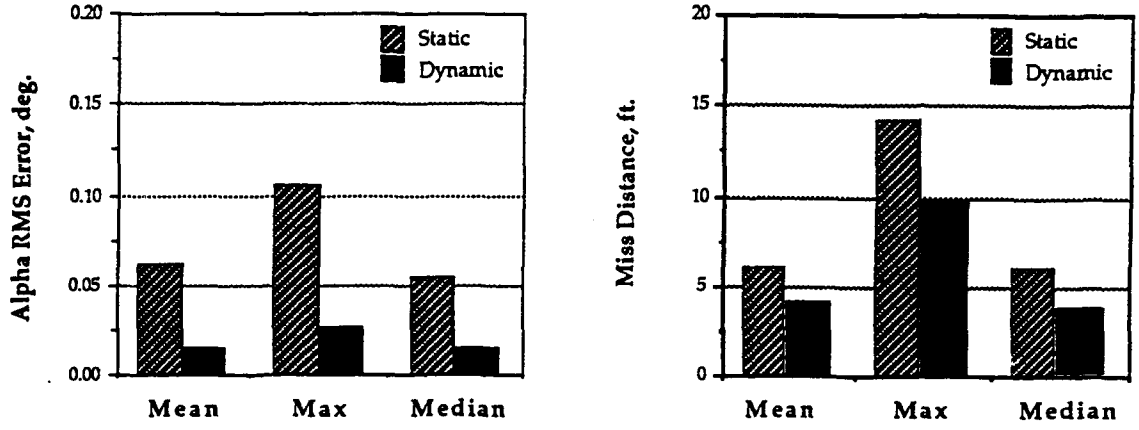


Figure 12: Steering Function and Trajectory Accuracies: Operating in Design Region, without Measurement Uncertainties

6.2 Robustness: Accuracies on Boundaries of the Design Region without Measurement Uncertainties

Robustness is of great importance when considering neural network solutions. These networks generally perform better on the interior regions of the design data base because the boundaries of the performance envelope are not usually represented as fully as the interior points during training. Trajectories that operate near the upper boundary have to generate as much lift as possible to reach the far intercept point, thus causing α to come close to the maximum allowed ($|\alpha| \leq$ 25 deg.). Both network designs handle the upper boundary accurately, with median alpha RMS errors of 0.140 and 0.034 deg., respectively, for static and dynamic networks, again an improvement ratio of approximately four to one (dynamic vs. static), as shown in Table 4. On the lower boundary, although errors in RMS steering function error are less for the dynamic network, miss distance results are more comparable, as shown in Table 5. This is because the miss distance is not completely a function of the steering function errors, but miss distance is affected also by when the errors occur.

Table 4: Steering Function and Trajectory Accuracies: Operating on Upper Boundary, without Measurement Uncertainties

(95,000 ft. < intercept point < 97,518 ft.)

	Alpha RMS Error, deg. *		
	Static	Dynamic	S/D Ratio
Mean	0.168	0.052	3.2
Std. Dev.	0.129	0.051	2.5
Max	0.258	0.100	2.6
Median	**0.140**	**0.034**	**4.1**

	Miss Distance, ft.		
	Static	Dynamic	S/D Ratio
Mean	39.6	14.2	2.8
Std. Dev.	29.1	13.9	2.1
Max	99.0	47.4	2.1
Median	**25.8**	**8.3**	**3.1**

* alpha = aerodynamic angle of attack (steering function)

Table 5: Steering Function and Trajectory Accuracies: Operating on Lower Boundary, without Measurement Uncertainties

(72,121 ft. < intercept point < 75,000 ft.)

	Alpha RMS Error, deg. *		
	Static	Dynamic	S/D Ratio
Mean	0.123	0.038	3.2
Std. Dev.	0.059	0.028	2.1
Max	0.151	0.061	2.5
Median	**0.120**	**0.034**	**3.5**

	Miss Distance, ft.		
	Static	Dynamic	S/D Ratio
Mean	23.2	22.3	1.04
Std. Dev.	11.7	7.9	1.48
Max	49.2	38.0	1.29
Median	**22.0**	**21.6**	**1.02**

* alpha = aerodynamic angle of attack (steering function)

6.3 Robustness: Accuracies when Extrapolating in Time without Measurement Uncertainties

As discussed earlier, the networks were synthesized using data for only the first 38.5 s. of flight. A test was performed to compare the capacities of the static and dynamic networks to guide the missile trajectories beyond the flight time used for their syntheses. The results obtained are shown in Table 6 and Figure 13 below. A comparison with Table 3 reveals that median RMS errors in alpha are approximately six times the median RMS errors in the design region, yet the median miss distances increased by only forty percent. Once again, the dynamic network exhibits approximately a four to one advantage over the static network in median alpha RMS error.

Table 6: Steering Function and Trajectory Accuracies: Extrapolation in Time, without Measurement Uncertainties

	Alpha RMS Error, deg. *		
	Static	Dynamic	S/D Ratio
Mean	0.933	0.336	2.8
Std. Dev.	1.164	0.467	2.5
Max	2.362	1.007	2.3
Median	**0.335**	**0.089**	**3.8**

	Miss Distance, ft.		
	Static	Dynamic	S/D Ratio
Mean	19.5	7.6	2.6
Std. Dev.	21.3	8.5	1.9
Max	80.2	37.4	2.5
Median	**9.5**	**5.0**	**1.90**

* alpha = aerodynamic angle of attack (steering function)

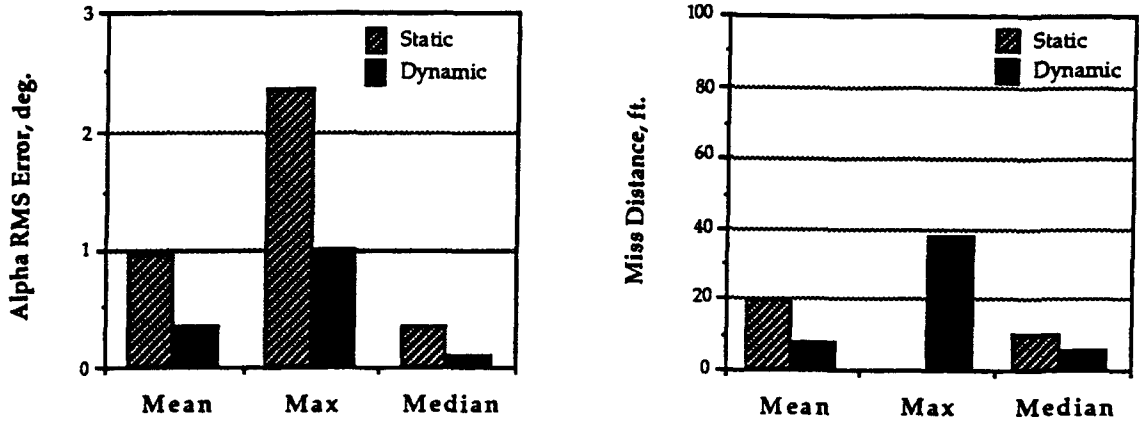


Figure 13: Steering Function and Trajectory Accuracies: Extrapolation in Time, without Measurement Uncertainties

6.4 Robustness: Accuracies within the Design Region with Measurement Uncertainties

Measurement uncertainties were introduced to determine their influence on accuracies of the static and dynamic networks. Altitude (H) and missile downrange distance (N) were given RMS uncertainties of 2 ft., while $\dot{H}$ and $\dot{N}$ were given RMS uncertainties of 1 ft./s. Comparing the results in Table 7 and Figure 14 with those in Table 3, it is seen that median alpha RMS error increased by a factor of 2.7 for the static network and 2.5 for the dynamic network.

Table 7: Steering Function and Trajectory Accuracies: Operating in Design Region, with Measurement Uncertainties

	Alpha RMS Error, deg. *		
	Static	Dynamic	S/D Ratio
Mean	0.151	0.036	4.2
Std. Dev.	0.062	0.014	4.4
Max	0.187	0.042	4.5
Median	**0.147**	**0.035**	**4.2**

* alpha = aerodynamic angle of attack (steering function)

	Miss Distance, ft.		
	Static	Dynamic	S/D Ratio
Mean	6.5	4.2	1.5
Std. Dev.	3.9	2.5	1.6
Max	16.9	11.3	1.5
Median	**6.0**	**4.0**	**1.50**

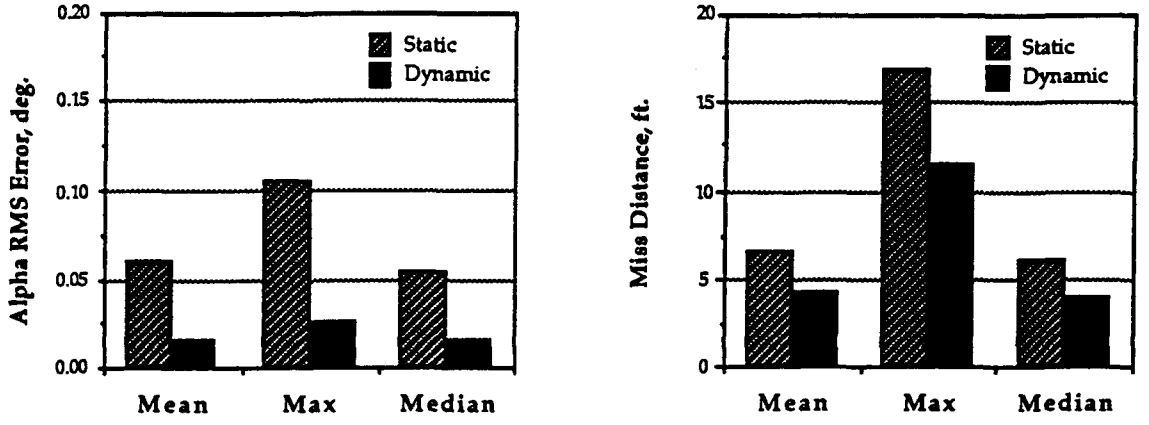


Figure 14: Steering Function and Trajectory Accuracies: Operating in Design Region, with Measurement Uncertainties

6.5 Robustness: Simplicity of the Designs

A key indicator of the robustness of network designs is the number of degrees of freedom used in the network. The simpler the model, the less prone it is to overfit. In combination, the two dynamic networks (startup and primary networks) had a total of 24 terms, whereas the static networks had 43 terms total. It is expected therefore that the dynamic networks are likely to generalize better than are the static networks.

6.6 Maximization of Missile Terminal Speed

Both network designs maximized missile terminal speed: the mean terminal speed was 1785 ft./s. for both. There was little deviation between the two because both network systems modeled the calculus of variations solution accurately.

6.7 Summary of Performance Results

Table 8 below summarizes the results shown in Sections 6.1 through 6.6.

Table 8: Summary of Results

		Static	Dynamic
1. Static and dynamic PNN designs provide good steering function accuracy along the control path and good terminal trajectory accuracy. Dynamic PNN is superior.	Median Alpha RMS Error, deg. CEP*: Median Miss Distance, ft.	0.054 5.9	0.014 3.8
2. Static and dynamic PNNs are robust: a) Dynamic PNN is better at the boundaries of design data base.	<u>Upper Boundary</u>: Median Alpha RMS Error, deg. CEP: Median Miss Distance, ft. <u>Lower Boundary</u>: Median Alpha RMS Error, deg. CEP: Median Miss Distance, ft.	 0.140 25.8 0.120 22.0	 0.034 8.3 0.034 21.6
b) Dynamic PNN is superior at extrapolating over time.	Median Alpha RMS Error, deg. CEP: Median Miss Distance, ft.	0.335 9.5	0.089 5.0
c) Dynamic PNN is more forgiving when state observations are noisy.	Median Alpha RMS Error, deg. CEP: Median Miss Distance, ft.	0.147 6.0	0.035 4.0
d) Dynamic PNN is simpler.	Number of coefficients	43	24
3. Static and dynamic PNN designs maximize missile terminal speed.	Mean Terminal Speed, ft./s.	1785	1785

* CEP = Circular Error Probable

7. CONCLUSIONS

The TPBV missile guidance solution has provided an excellent framework in which to compare the static and dynamic neural networks. The particular application considered involved the guidance of a tactical air-intercept missile, guided from a specified initial condition to a horizontally moving target.

The proper synthesis method depends on the application. In general, an equation-error synthesis method is appropriate for prediction, while output-error synthesis methods should be used in control problems. In addition, initial *structures* for dynamic controller nodes can be found using equation-error synthesis which requires less computational time. Once the general structure is found, an output-error synthesis algorithm can search for the optimum model coefficients.

The results show that both the static and dynamic network designs provide good steering function accuracy along the missile trajectory, and good terminal position accuracy relative to a specified intercept point. However, the dynamic network models the steering function approximately four times more accurately than does the static network. Performance results on several robustness tests, including operation on the boundaries of the design region, extrapolation in time, and operation in the presence of measurement uncertainties, all show that the dynamic network is more robust than is the static network used in this application. The dynamic network required approximately half the degrees of freedom of its static counterpart, diminishing the possibility of overfit. Both networks performed well in maximizing terminal speed.

APPENDIX D
REFERENCES

1. Abbott, D. W., and R. L. Barron, *Active Control of Complex Systems Via Neural Networks Having Internal Feedback Paths and Time Delays,* Barron Associates, Inc. Technical Progress Report 1 under Contract N00014-89-C-0137, August 15, 1989.

2. Barron, R. L., D. W. Abbott, et al., *Trajectory Optimization and Optimum-Path-to-Go Guidance of Tactical Weapons* (four volumes), Final Report by Barron Associates, Inc. for HR Textron, Inc. under Naval Weapons Center Contract N60530-88-C-0036, January 1989.

3. Golub, G. H., and C. F. van Loan, *Matrix Computations,* Johns Hopkins University Press, Baltimore, 1985.

4. Press, W. H., B. P. Flannery, et al., *Numerical Recipes,* Cambridge University Press, New York, 1986.

5. Barron, A. R., "Predicted Squared Error: A Criterion for Automatic Model Selection", *Self-Organizing Methods in Modeling: GMDH Type Algorithms* (S.J. Farlow, Ed.), Marcel Dekker, Inc., New York, Chap. 4, pp. 87-105, 1984.

APPENDIX E

Contract N00014-89-C-0137
Technical Progress Report 3

ACTIVE CONTROL OF COMPLEX SYSTEMS VIA DYNAMIC NEURAL NETWORKS: COMBUSTION PROCESSES IN PROPULSION SYSTEMS

B. Eugene Parker, Jr., Ph.D.
Richard L. Cellucci
Dean W. Abbott

April 8, 1991

Prepared for:

DEPARTMENT OF THE NAVY
Office of the Chief of Naval Research
Defense Sciences Division
800 North Quincy Street
Arlington, Virginia 22217-5000

Attention: Dr. Robert J. Hansen, Director

Prepared by:

BARRON ASSOCIATES, INC.
Route 1, Box 159
Stanardsville, Virginia 22973-9511
(804) 985-4400

TABLE OF CONTENTS

Dynamic Polynomial Neural Networks Applied to Active Control of Combustion Processes in Propulsion Systems

Abstract

Dynamic polynomial neural networks (PNNs) are used to predict the output of a combustion process and to synthesize a neural network controller. In addition, the PNN approach is reconciled with the classical Volterra-Wiener system identification technique often used to identify plant models. PNN techniques are shown to offer advantages over this traditional system identification methodology, both in terms of modeling accuracy and in reducing the number of parameters necessary to achieve a given accuracy. In particular, memory-feedback PNN performance was shown to exceed that of memory-feedforward PNNs while using fewer coefficients to fit the data.

1. Introduction

Techniques for performing system identification for the purpose of process control are well described in the literature. In one sense, the various approaches that have been developed may be distinguished based upon the amount of *a priori* information that is available for use in the modeling process. For example, if a structural model based on the physical process under consideration exists, the task of system identification is reduced to one of parameter estimation. If no such structural model is available, the task of system identification is inherently more complex.

Polynomial neural network (PNN) techniques are applicable both with and without *a priori* structural models. In this investigation, we are concerned with the identification of a combustion process (plant) for which no structural model is available. A classical approach that is also applicable in this case is based on Volterra-Wiener theory. In general, both the PNN and Volterra-Wiener techniques apply to analytic multi-input, multi-output, nonlinear, nonstationary systems. Because PNNs allow feedback of their outputs, they are not limited to application in systems having finite memory,[1] as is true for Volterra-Wiener techniques.

Fig.1.1 illustrates the combustion control process; amplitude modulation (AM) is used to excite a speaker, which provides an acoustic disturbance that affects the flame. Flame height or quality is measured using a photodiode (PD). The objective of the controller is to provide an AM signal to regulate the process, i.e., to maintain the flame at a value selected by the operator. Since the combustion process under consideration here has finite memory,[2] both approaches may be used in designing a controller.

---

[1] A system's memory is the time it takes for the system to complete its response to an instantaneous input. Excluded therefore are sytems such as oscillators, which have infinite memory.

[2] Note that even though the input to the speaker in Fig. 2.1 is an amplitude modulated sinusoid, the unmodulated carrier signal is considered the zero-input condition and is not modeled here.

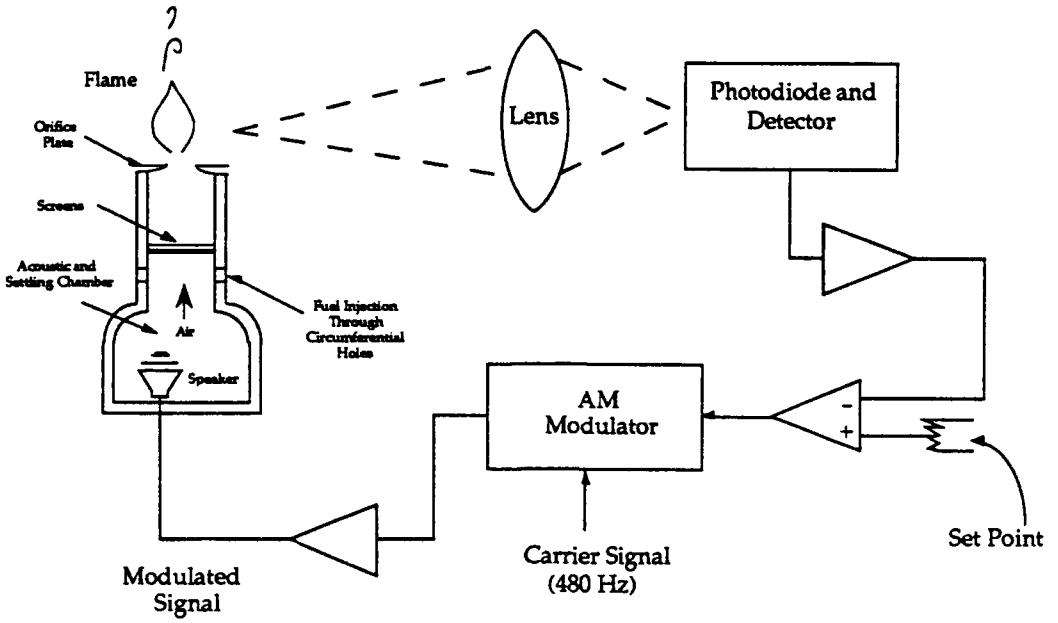


Figure 1.1: Typical Combustion Experimental Controller Configuration (NWC)

In the discussion below, the PNN approach is reconciled with the classical Volterra-Wiener system identification technique. PNNs are shown to be more general, since the classical approach represents a memory-feedforward PNN, meaning that there is no feedback of the output estimates. Moreover, the PNN formulation is shown to offer advantages over the classical approach both in terms of modeling accuracy and in terms of the number of degrees of freedom necessary to achieve a given level of accuracy.

In addition, PNNs allow easy incorporation of any available *a priori* structural information in the modeling process, allowing very flexible modeling. *A priori* information is much more difficult to take advantage of using other modeling techniques, such as in Volterra-Wiener system identification, when complete structural information is not available.

For a glimpse ahead at the excellent modeling results achieved on the limited set of PNN training data available, the reader is referred to Figures 5.9 and 5.10. These figures illustrate the predictive accuracy of memory-feedback PNNs as compared to the actual process data.

2. Polynomial Neural Networks

2.1 Background

Polynomial neural networks (PNNs) are compositions of Kolmogorov-Gabor (KG) multinomials, which are algebraic sums of terms. The KG multinomial is

$$y = a_0 + \sum_i a_i x_i + \sum_i \sum_j a_{ij} x_i x_j + \sum_i \sum_j \sum_k a_{ijk} x_i x_j x_k + \ldots \qquad (2:1)$$

The KG multinomial can model any analytic single-valued transformation. All neural networks synthesized using analytic nodal elements (including the now popular linear-sum-with-sigmoid element) can be closely approximated by a finite KG multinomial. The classical Farley and Clark elements [1954], Rosenblatt's "Perceptron" [1958], and Widrow's "Adaline" [1960] used zero-crossing detectors rather than sigmoids; those also can be approximated by a finite KG multinomial.

Barron Associate's *Algorithm for Synthesis of Polynomial Networks (ASPN)*, a composition-of-functions synthesis method, builds a KG transformation from linear elements and nonlinear second-degree and third-degree KG elements; *ASPN* quickly and efficiently creates specific KG multinomials required for given applications. *ASPN* also automatically incorporates transcendental functions, including the sigmoid, if and where such functions are relevant in the network (it is *not* usual for very many transcendental functions to be required). Thus, by using a small family of KG element and transcendental element types, *ASPN* synthesizes essentially any of the common neural network transformations, but does this without overfitting, as is explained below.

As shown in Table 2.1, polynomial networks are significantly different from conventional "sigmoidal" neural networks, the latter of which are used mainly for classification tasks. Sigmoidal networks are structured in advance by the designer, whereas *ASPN*-type algorithms build the network from the simplest possible form to a just-sufficient level of complexity. Pre-structured networks tend to have excessive internal degrees of freedom and therefore often overfit the synthesis data. In addition, they are often designed by the analyst using a trial and error approach; with *ASPN*, synthesis time is dramatically reduced since it selects the relevant network architecture. Moreover, one usually cannot guarantee that a pre-structured network has the specific interactions between variables and the nonlinearities that are needed for a given application. Neural network synthesis using a composition-of-functions algorithm based upon the KG multinomial (such as *ASPN*) provides the needed transformation and a just-right level of complexity.

There are many kinds of polynomial network nodal elements, but there is typically only one type of node used in sigmoidal networks, namely a weighted linear sum feeding into the sigmoidal transformation. In addition, there are often thousands of nodes in sigmoidal neural networks, whereas *ASPN*-type algorithms use as few nodes as possible. Experience has shown that the simpler the network, the less sensitive it is to noise and the better able it is to generalize, because it has fewer internal degrees of freedom.

Currently, the practice used by some to avoid overfitting in neural network syntheses is to add noise to the data during fitting. It can be shown that the Fisher information matrix, which characterizes the asymptotic efficiency of best estimators, is degraded by the corruption-by-noise method; BAI believes that it is preferable to employ *constrained* fitting criteria, as discussed below.

A main advantage of sigmoidal neural networks is their ability to process a large number of inputs. Their main disadvantage is inability to model flexibly. Polynomial networks have more complex and, therefore, more "intelligent" nodes than just the sigmoidal transformation. A key ability of *ASPN* is to combine these "intelligent" nodes rapidly in the best way possible. Each trial *ASPN* network can be synthesized and evaluated on the design data base in a fraction of a second. Sigmoidal networks are traditionally pre-structured by the analyst and trained using a form of gradient search, which can be a slow process, prone to converge to false minima.

Table 2.2 outlines the key principles that have been learned about the syntheses and applications of neural networks. *These principles are violated by most other neural network syntheses algorithms in present-day use.*

Applications involving PNNs are often separated into two classes, *estimation* (which includes prediction and filtering) and *classification*. Moreover, estimation comprises *forward* (predictive) and

Table 2.1: Approaches to Neural Network Synthesis

CONVENTIONAL APPROACH	BARRON ASSOCIATE'S APPROACH
• Use of large, pre-structured networks with adaptation of coefficients	• Simple structures learned rapidly via composition-of-functions algorithm
• Linear-sum-with-sigmoid nodal elements	• Nonlinear elements based upon Kolmogorov-Gabor multinomial
• Unconstrained modeling criteria	• Criteria constrained by information-theory measure of model degrees of freedom
• Artificial exemplars created by adding noise to true exemplars	• Emphasis on robust designs achieved with information-theoretic methods
• Recursive learning using least mean square gradient descent techniques	• Combined batch-supervised and recursive learning using predicted squared error, recursive least squares, recursive prediction error, and/or Levenberg-Marquardt methods
• Static networks (internal feedforward connections only, no internal time delays or feedbacks)	• Static networks and dynamic networks (internal feedbacks, internal time delays) to minimize network degrees of freedom and minimize noise sensitivity

Table 2.2: Key Principles in Neural Network Syntheses [Barron *et al.* 1990]

- Nodal elements should be analytic and incorporate cross-products as well as sums of their inputs.
- Network structures should evolve from simplest forms to levels of just-sufficient complexity as syntheses proceed.
- Excessive complexity of neural networks must be avoided to prevent overfitting and consequent unreliability when processing new data.
- Global searches are indispensable in assuring network optimality.
- Information theoretic (statistical inference) criteria, including overfit penalties, should be used to govern network syntheses. Different performance criteria apply to estimation-network and discrimination (classification)-network syntheses. Estimation techniques require the use of predicted squared error, whereas classification techniques should use logistic-loss techniques.

inverse (control) applications. Since combustion process modeling falls into the this category, estimation networks are discussed in more detail below.

2.2 Estimation Using Polynomial Neural Networks

Reviewing the status of work by BAI on fitting criteria, a generic statistical criterion has been developed to govern composition and selection of functions during network syntheses. This criterion takes the form

$$J = \frac{1}{n} \sum_{i=1}^{n} d\left(y_i, f(\underline{x}_i, \hat{\underline{\theta}})\right) + \frac{k}{n} \qquad (2:2)$$

where J is the criterion value (to be minimized) for a given output variable y, n is the number of input-output pairs (data vectors) in the synthesis data base, y_i is the ith value of the output variable in the data base, $\underline{x}_i$ is the ith value of the input vector in the data base, $\hat{\underline{\theta}}$ is the estimate of the optimum parameter vector for the network, and k is the dimensionality of $\hat{\underline{\theta}}$. The loss or *distortion function*, $d(y_i, \hat{f}_i)$, can take various forms. Of particular interest for *estimation* networks is the *squared-error loss*

$$d(y_i, \hat{f}_i) = \frac{1}{2\sigma^2} (y_i - \hat{f}_i)^2 \qquad (2:3)$$

in which σ^2 is the network error variance (constant). Using Eq. 2:3 in Eq. 2:2 [Barron 1981; Barron 1984; Barron and Barron 1988]

$$PSE = 2\sigma^2 J = FSE + 2\sigma^2 k/n \qquad (2:4)$$

where PSE is the *predicted squared error* of the network, FSE is the average *fitting squared error*, $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}_i)^2$, and $2\sigma^2 k/n$ is the complexity penalty used to constrain the fit. (Nearest-neighbor tests may be used prior to network syntheses to determine conservative values of σ^2). PSE is also sometimes used for preliminary syntheses of *classification* networks.

The least-squares algorithm is suitable for fitting static polynomial estimation networks off-line and is used in *ASPN*. *ASPN* and recursive least squares (RLS) permit on-line adaptation of these networks. Recursive prediction error (RPE) and the discrete Levenberg-Marquardt (DLM) algorithm provide off-line fitting capabilities for dynamic estimation networks. On-line adjustment of dynamic network parameters may be achieved with a variation of RPE or the recursive Levenberg-Marquardt (RLM) procedure.

2.2.1 Off-Line Estimation

Estimation networks are constructed by *ASPN* using the predicted squared error (PSE) criterion, which not only computes the fitting error of the model, but also takes into account the model complexity. PSE penalizes more complex network structures more heavily to emphasize that a simpler

model is usually better, thereby avoiding the *overfit* problem. Thus, *ASPN* is able to construct a network of optimum complexity without overfitting. Further details may be found in Section 3.4.

2.2.2 On-Line Estimation

Recursive (on-line) updating of static polynomial networks is accomplished through the use of recursive least squares (RLS) [Ljung and Söderström 1983; Ljung 1987; Söderström and Stoica 1989]. By applying the RLS algorithm successively to each node in the polynomial network, the whole network is effectively updated. Initially, RLS is given the values of the parameters, $\underline{\theta}$, and the regression matrix, **R**. Both of these are generated off-line. **R** is then inverted to produce an updating matrix, **P**.

The RLS algorithm updates $\underline{\theta}$ and **P** each time it is applied. A weighting factor λ (which must be positive), can be chosen for each new observation (λ is set to unity during off-line synthesis). This weighting factor determines the rate at which the polynomial learns. The following equations are solved to update $\underline{\theta}$ and **P**; in the equations, $\underline{\phi}$ represents the independent variables in the model:

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \mathbf{L}\left[y - \underline{\phi}^T \underline{\theta}_{old}\right] \tag{2:5}$$

$$\mathbf{L} = \frac{\mathbf{P}_{old}\,\underline{\phi}}{\lambda + \underline{\phi}^T \mathbf{P}_{old}\,\underline{\phi}} \tag{2:6}$$

$$\mathbf{P}_{new} = \mathbf{P}_{old} - \frac{\mathbf{P}_{old}\,\underline{\phi}\,\underline{\phi}^T \mathbf{P}_{old}}{\lambda + \underline{\phi}^T \mathbf{P}_{old}\,\underline{\phi}} \tag{2:7}$$

These formulas apply only to polynomial nodal elements that have been fitted with the least-squares criterion. Similar equations exist for recursive maximum likelihood algorithms, which can update polynomial networks that have been fitted using the logistic-loss criterion. See Section 3.5 for further details.

Recursive learning, while advantageous for on-line learning, is suitable only for networks having defined structure. Batch-supervised learning, using a pre-stored data base (batch), allows composition-of-functions learning, and usually is the method of choice for pre-training of neural networks. An algorithm that combines the two types of learning in a unified program that provides composition-of-functions learning *on-line* as well as during pre-training is under development.

2.3 Dynamic Polynomial Neural Networks

Dynamic (reverberatory or recurrent) polynomial neural networks (PNNs) contain time-delays and/or self-feedbacks within one or more of the nodes or sub-networks comprising the network. By including self-feedback connections, dynamic PNNs can:

- compute time-varying transformations given static inputs
- perform infinite impulse response (IIR) filtering operations in addition to finite impulse response filtering (FIR) operations
- provide a phase-shift operator between time-varying inputs.

Dynamic PNNs require fewer internal degrees of freedom (i.e., coefficients) than do static networks created for the same application. The greater simplicity of dynamic networks results in improved model accuracy and robustness. Software has been developed for the synthesis of nodal

elements having internal feedbacks and time delays. An algorithm that will provide feedback connections between elements in the same or different layers is under development. It has been demonstrated [Nigro, Abbott, and Barron 1990] that dynamic neural networks can be significantly (up to four times) more accurate than their static counterparts in steering accuracy and trajectory control of a tactical air-intercept missile. The message here again is that the simpler the network, the less sensitive it is to noise and the better able it is to generalize. However, there is a price to pay: off line, the synthesis of dynamic PNNs requires iteration and typically more computation than static PNNs; on line, dynamic PNNs require initialization before they can provide output estimates.

The following definitions are used in the discussion below:

- *Static Node:* a node with no *internal* self-feedbacks or time delays (see Fig. 2.2)
- *Dynamic Node:* a node with self-feedbacks and/or input time delays *internal* to the node. Note that self-feedback connections are implemented *with* time delays in the self-feedback paths.
- *Non-Memory Feed-forward (NMFF) Node:* a static node.
- *Memory Feed-forward (MFF) Node:* a dynamic node with internal time delays of the inputs, but no self-feedback connections.
- *Memory Feedback (MFB) Node:* a dynamic node with internal self-feedback connections, but not necessarily with internal time delays of the inputs (see Fig. 2.3.) Note that an MFB node contains the node connections of NMFF and MFF nodes, along with self-feedback connections.
- *Shift Register:* a device in the dynamic node that stores previous input or output estimates.

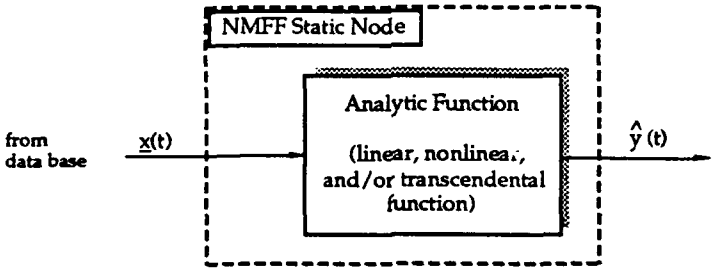


Figure 2.2: Static Nodal Element

2.3.1 Initialization of Dynamic Networks

Dynamic networks require initialization, using input and/or output values to fill the shift registers of the feedforward and self-feedback connections. This does not pose a problem if no estimate of the output is needed at t_0, i.e., before the shift registers of the dynamic node are filled. For example, in prediction applications the first output of the network occurs after enough of the process has been seen (i.e., after the dynamic network shift registers have been filled). In other applications, however, such as in tracking and control, output estimates must be generated beginning at time t_0, and therefore another controller, such as a static network, must be used until the dynamic network is initialized.

2.3.2 Synthesis Methods for Dynamic Nodes

There are two ways in which to synthesize MFB dynamic nodes: *equation-error* and *output-error*.

Equation-Error Method: The output estimate is found using previous output values obtained from the data base, not from output estimates of the dynamic node. The equation-error method does not require iteration for synthesis; in general, it is computationally simple.

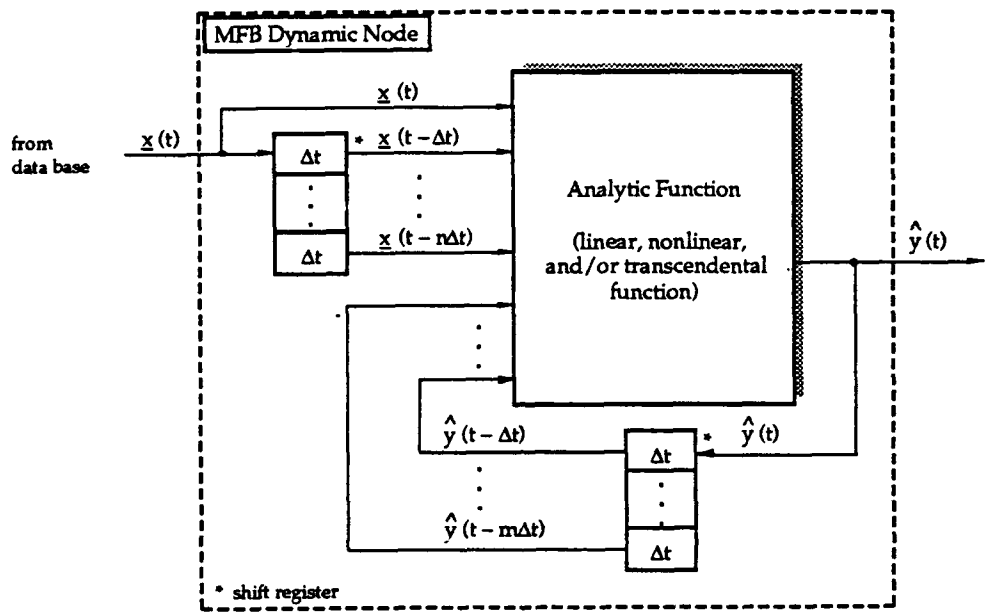


Figure 2.3: Dynamic Nodal Element

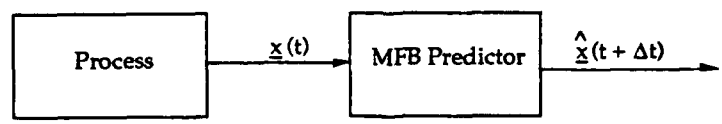


Figure 2.4: MFB Network Synthesized as a Predictor

Output-Error Method: The output estimate is found using previously estimated output values. Because the node must first produce an output before it can be used as a new input, iteration is required during model synthesis; output-error modeling thus requires more computation than equation-error syntheses.

The preferred method of training depends on the application in which the network is to be used. Equation-error syntheses are best suited for MFF predictors, as shown in Fig. 2.4, because output estimates are not used as future network inputs; thus, output estimates should not be used during synthesis.

On the other hand, when a controller or tracker is to be synthesized, the use of output-error in MFB model synthesis is appropriate because output estimates directly influence future network outputs, as is shown in Fig. 2.5. An output-error synthesis trains the network in the same way in which it will be used; thus, current output estimates should affect future output estimates during synthesis.

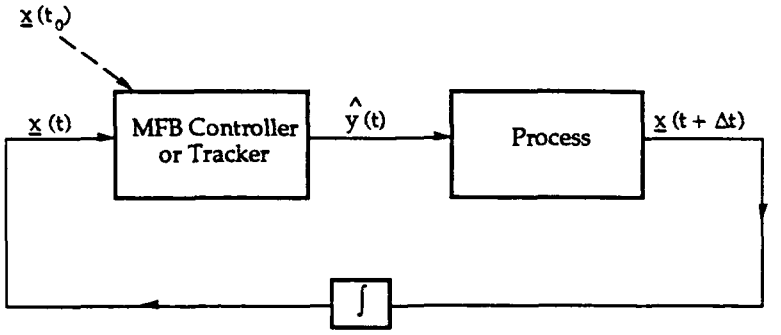


Figure 2.5: MFB Network Synthesized as a Controller or Tracker

Table 2.3 summarizes the off-line synthesis algorithms that are appropriate for NMFF, MFF, and MFB nodes. The algorithms used for equation-error and output-error syntheses are least-squares and a modified Levenberg-Marquardt search, respectively. Least-squares, a linear regression algorithm, requires linearity with respect to the node coefficients, which is true only for NMFF, MFF, and MFB (equation error) nodes. The Levenberg-Marquardt algorithm [Cadzow 1990] is a nonlinear regression algorithm and is appropriate for nodes that are nonlinear with respect to the coefficients (MFB output error nodes).

Table 2.3: Off-Line Synthesis Methods for Prediction, Control, and Tracking

Application	Node Type	Synthesis Algorithm Method	Preferred Synthesis Algorithm	Iteration Needed?
Prediction or Control	NMFF	N/A	Least Squares	No
Prediction or Control	MFF	N/A	Least Squares	No
Prediction		Equation Error	Least Squares	No
Control or Tracking	MFB	Output Error	Levenberg-Marquardt	Yes

3. Plant Identification

3.1 Background

The basic idea behind plant or system identification is the use of input-output measurements to determine the *impulse response* or *kernels* of the system. For a single-input, single-output, linear, stationary system having impulse response $h(\tau)$, the response of the system to any input excitation $x(t)$ may be determined by convolving the system impulse response with the input,

$$y(t) = \int_{-\infty}^{\infty} k(\tau)\ x(t-\tau)\, d\tau \qquad (3:1)$$

For causal systems, the lower limit of the integral may be set equal to zero since $k(\tau) = 0$ for $\tau < 0$. For nonstationary systems, the impulse response is a function of both time t as well as lag τ, *viz.* $k(t, \tau)$. Nonstationarity changes the computational details somewhat, but the basic solution approach is not otherwise affected. Unless specifically mentioned in the discussion below, stationarity in both the input signal and in the plant itself is assumed.

Vito Volterra showed that the basic concept of the linear impulse response could be extended for the case of nonlinear systems, such that the system output is represented using higher-order convolutions of the nonlinear kernels with multiple independently delayed input signals. In general,

$$y(t) = k_0 + \int_{-\infty}^{\infty} k_1(\tau)\ x(t-\tau)\, d\tau + \int_{-\infty}^{\infty} k_2(\tau, \tau_2)\ x(t-\tau_1)\, x(t-\tau_2)\, d\tau_1\, d\tau_2 + \text{H.O.T.}^3 \qquad (3:2)$$

or

$$y(t) = \sum_{m=0}^{\infty} G_m[k_m(t_1, \ldots, t_m); x(t'), t' \le 0] \qquad (3:3)$$

where $G_m[k_m(t_1, \ldots, t_m; x(t))]$ represents the mth-order Volterra functional[4] . The first three Volterra functionals are given in Table 3.1.

In the functional series, k_0 represents the zero-order kernel (i.e., the average value), $k_1(\tau)$ the linear kernel, $k_2(\tau_1, \tau_2)$ the second-order kernel, and so on for higher-order nonlinearities. The zeroth-order kernel may, in general, be taken to be zero, since if the input $x(t) = 0$, with a change of variables

---

[3] H.O.T. stands for higher-order terms.

[4] A functional is a function whose argument is also a function, but whose value is a number.

Table 3.1: First Three Volterra Functionals

$$G_0[k_0; x(t)] = k_0$$

$$G_1[k_1; x(t)] = \int_0^\infty k_1(\tau)\ x(t-\tau)\, d\tau$$

$$G_2[k_2; x(t)] = \int_0^\infty k_2(\tau_1, \tau_2)\ x(t-\tau_1)\, x(t-\tau_2)\, d\tau_1\, d\tau_2$$

we may make y(t) = 0. For a linear system, the impulse response is $k_1(\tau)$; $k_2(\tau_1, \tau_2)$ and higher-order kernels are zero. For nonlinear systems, these higher-order terms become important. For example, the impulse response of a nonlinear system is

$$y(t) = k_0 + k_1(t) + k_2(t, t) + \text{H.O.T.}; \qquad (3:4)$$

note that the impulse response of a nonlinear system includes the diagonal terms of *all* kernels, not just the first-order kernel.

Various techniques may be used to obtain the kernels from input and output data records. Norbert Wiener showed that if the input is a Gaussian white-noise (GWN) signal, then each term of the functional series can be made orthogonal *for such a stimulus*, and each kernel can be measured independently. Orthogonality means that $E\{G_i[k_i; x(t)]\ G_j[k_j; x(t)]\} = 0$, for $i \neq j$. In general, the Wiener kernels are different from the Volterra (system) kernels; for systems with nonlinearities no greater than second-order, the Volterra and Wiener kernels are the same. The first three Wiener functionals are given in Table 3.2.

In general, the Wiener kernels are polynomials in P, the input signal power, with coefficients that depend on higher-order Volterra kernels. This attests to the efficiency of the orthogonal basis of the Wiener representation since, for example, the first-order Wiener kernel of a nonlinear system will represent some system nonlinearities. Thus, the first-order Wiener functional of a nonlinear system is, in general, different from the linear part of the system (i.e., first-order Volterra functional). In addition, orthogonalization of the Wiener functionals means that truncating the series after the nth-order term provides the best nth-order model in the mean-square-error sense. It should be noted that the Volterra functional series can be orthogonalized for any signal possessing suitable auto-correlation properties.

3.2 Estimation of the Kernels

There exist several approaches to estimating the Wiener kernels, such as that of Wiener-Bose, which is based on expanding the kernels in terms of Laguerre and Hermite functions. Such methods have, to a large extent, been replaced by less cumbersome cross-correlation techniques, which may be

Table 3.2: First Three Wiener Functionals

$$G_0[h_0; x(t)] = h_0$$

$$G_1[h_1; x(t)] = \int_0^\infty h_1(\tau)\ x(t-\tau)\, d\tau$$

$$G_2[h_2; x(t)] = \int_0^\infty h_2(\tau_1, \tau_2)\ x(t-\tau_1)\, x(t-\tau_2)\, d\tau_1\, d\tau_2 \quad - P \int_0^\infty h_2(\tau_1, \tau_2)\, d\tau_1 ,$$

where P is the power level of the Gaussian white-noise (GWN) input.

implemented in either the time or frequency domains. The cross-correlation technique is derived by multiplying both sides of Eq. 3:3 by $x(t-\sigma)$ and taking expectations on both sides of the equation. For example, for the first-order Wiener functional,

$$y(t) = \int_0^\infty h_1(\tau)\ x(t-\tau)\, d\tau \qquad (3:5)$$

or

$$E[\, y(t)\, x(t-\sigma)\,] = E[\, x(t-\sigma) \int_0^\infty h_1(\tau)\ x(t-\tau)\, d\tau\,] \qquad (3:6)$$

or

$$\phi_{yx}(\sigma) = \int_0^\infty h_1(\tau)\ E[\, x(t-\sigma)\, x(t-\tau)\,]\, d\tau \qquad (3:7)$$

Because x(t) is a GWN signal,

$$E[\, x(t-\sigma)\ x(t-\tau)\,] = \phi_{xx}(\tau-\sigma) = P\,\delta(\tau-\sigma) \qquad (3:8)$$

where δ is the Dirac delta function defined as $\delta(\tau-\sigma) = 1$ when $\tau = \sigma$, and $\delta(\tau-\sigma) = 0$ when $\tau \neq \sigma$. Thus,

$$\phi_{yx}(\sigma) = \int_0^\infty h_1(\sigma)\ \phi_{xx}(\tau - \sigma)\, d\tau \tag{3:9}$$

$$= \int_0^\infty h_1(\sigma)\ P\, \delta(\tau - \sigma)\, d\tau \tag{3:10}$$

$$= P\, h_1(\sigma) \tag{3:11}$$

or

$$h_1(\sigma) = \frac{\phi_{yx}(\sigma)}{P} \tag{3:12}$$

where $P = \phi_{xx}(0)$.

Carrying out the same process for the second-order kernel yields

$$h_2(\sigma_1, \sigma_2) = \frac{\phi_{yxx}(\sigma_1, \sigma_2)}{2P^2}. \tag{3:13}$$

The overall approach to computing the kernels is outlined in Table 3.3.

Table 3.3: Steps in Computation of the Wiener Kernels

Step	Description
1	Compute the response of the system to a GWN stimulus.
2	Average the response to obtain h_0, the mean value.
3	Subtract h_0 from y(t) to obtain $y_0(t)$, the zero-mean response.
4	Cross-correlate $y_0(t)$ with $x(t - \tau)$ to obtain $h_1(\tau)$.
5	Convolve the input sequence x(t) with $h_1(\tau)$ to obtain the modeled response $y_1(t)$.
6	Subtract $y_1(t)$ from $y_0(t)$ to obtain the nonlinear response residual.
7	If the nonlinear response residual in Step 6 is insignificant, go to Step 4, substituting higher-order correlations and convolutions in place of the second-order computations; else, go to Step 8.
8	Done.

3.3 Reconciliation of Volterra-Wiener Approach with Polynomial Neural Networks

As shown in Eqs. 3:2 and 3:3, the kernels represent the coefficients of the model used to estimate the output of the system for new inputs based on a functional series. Considering for the moment the single-input, single-output system discussed above, we see that estimating the system's output for any input involves convolving the input data with the kernels,

$$y(t) = \sum_{n=0}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} k_n(\tau_1, \ldots, \tau_n)\, x(t-\tau_1) \cdots x(t-\tau_n)\, d\tau_1 \cdots d\tau_n \tag{3:14}$$

For the first-order kernel, at each point in time at which an estimate is needed, this requires summing the product of k_1 (in reverse order) with the input data:

$$\sum_i k_1(i)\ x(m-i)\ \Delta t \tag{3:15}$$

For the second-order kernel, this involves summing the product of k_2 (in reverse order) with a product of the input data:

$$\sum_i \sum_j k_2(i, j)\ x(m-i)\, x(m-j)\ \Delta t^2 \tag{3:16}$$

This is equivalent to the approach taken in modeling with polynomial neural networks, where compositions of KG multinomials (algebraic sums of terms) are used. Recall that the KG multinomial is

$$y(t) = a_0 + \sum_i a_i x_i + \sum_i \sum_j a_{ij} x_i x_j + \ldots \tag{3:17}$$

Thus, the KG coefficients and the Volterra-Wiener kernels are related as follows:

$$\begin{aligned} a_0 &= k_0 \\ a_i &= k_1(i) \\ a_{ij} &= k_2(i, j) \\ &\vdots \\ &\text{etc.} \end{aligned} \tag{3:18}$$

Note that although we have been concerned here only with univariate time series, both the Volterra-Wiener and PNN techniques are perfectly compatible with multivariate data. This is illustrated in Fig. 3.1 for the two-input, one-output, second-order system. The corresponding PNN would have the form

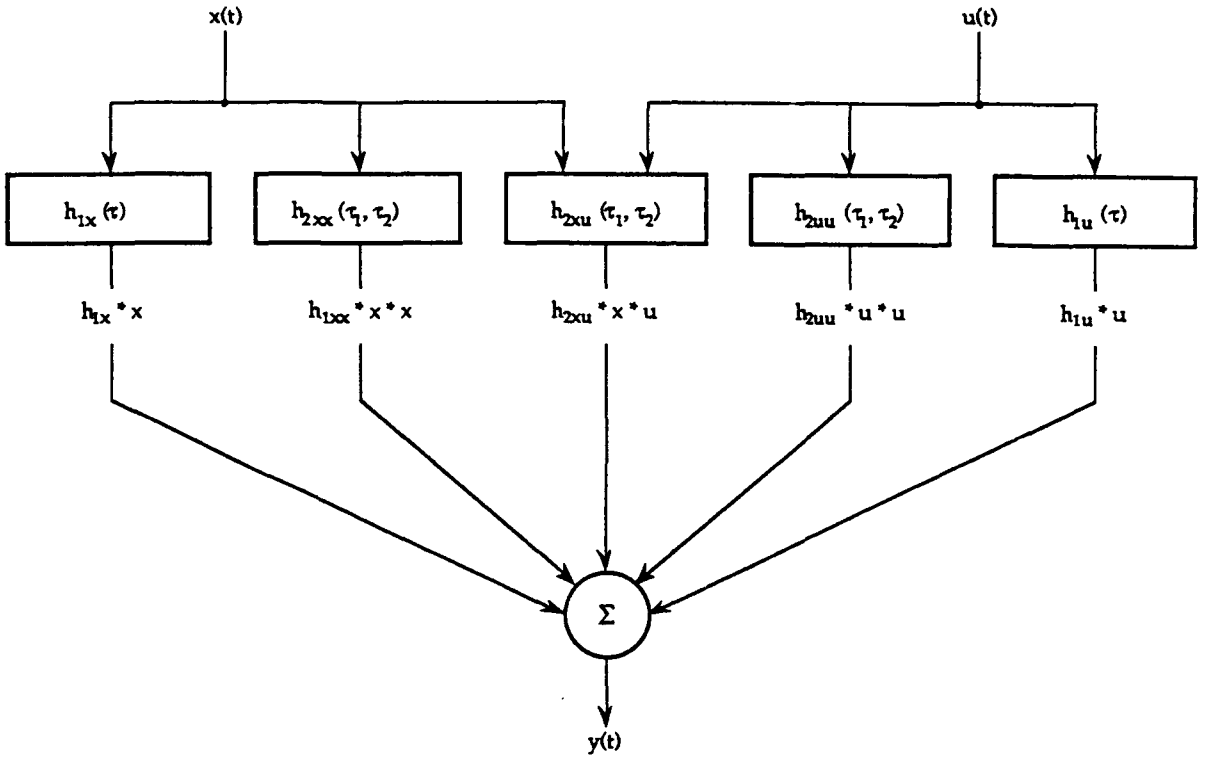


Figure 3.1: Kernel Diagram for Two-Input, One-Output, Second-Order System [Marmarelis and Marmarelis 1978]

$$y(t) = a_0 + \sum_i a_i x_i + \sum_i b_i u_i + \sum_i \sum_j a_{ij} x_i x_j + \sum_i \sum_j b_{ij} u_i u_j + \sum_i \sum_j c_{ij} x_i u_j + \ldots \quad (3:19)$$

Fig. 3.2 illustrates the corresponding kernel diagram for a system with two inputs and two outputs. The corresponding PNN would have the form

$$y(t) = a_0 + \sum_i a_i x_i + \sum_i b_i u_i + \sum_i \sum_j a_{ij} x_i x_j + \sum_i \sum_j b_{ij} u_i u_j + \sum_i \sum_j c_{ij} x_i u_j + \ldots \quad (3:20)$$

$$w(t) = c_0 + \sum_i c_i x_i + \sum_i d_i u_i + \sum_i \sum_j d_{ij} x_i x_j + \sum_i \sum_j e_{ij} u_i u_j + \sum_i \sum_j f_{ij} x_i u_j + \ldots \quad (3:21)$$

An alternative formulation related to the white noise system identification method described above, which fully reconciles the PNN and Volterra-Wiener techniques, was suggested by Marmarelis and Marmarelis [1978]. It involves creating a data base of input-output vectors $\underline{x}$ and $\underline{y}$, such that the output at any time t, $\underline{y}_i(t)$, corresponds to $\underline{x}_i(t)$, where $\underline{x}_i(t)$ is comprised of the current input and a number of previous inputs corresponding to the memory of the system. While a white-noise input signal could be used to generate new input and output vectors, this is not a requirement. Instead, a computer

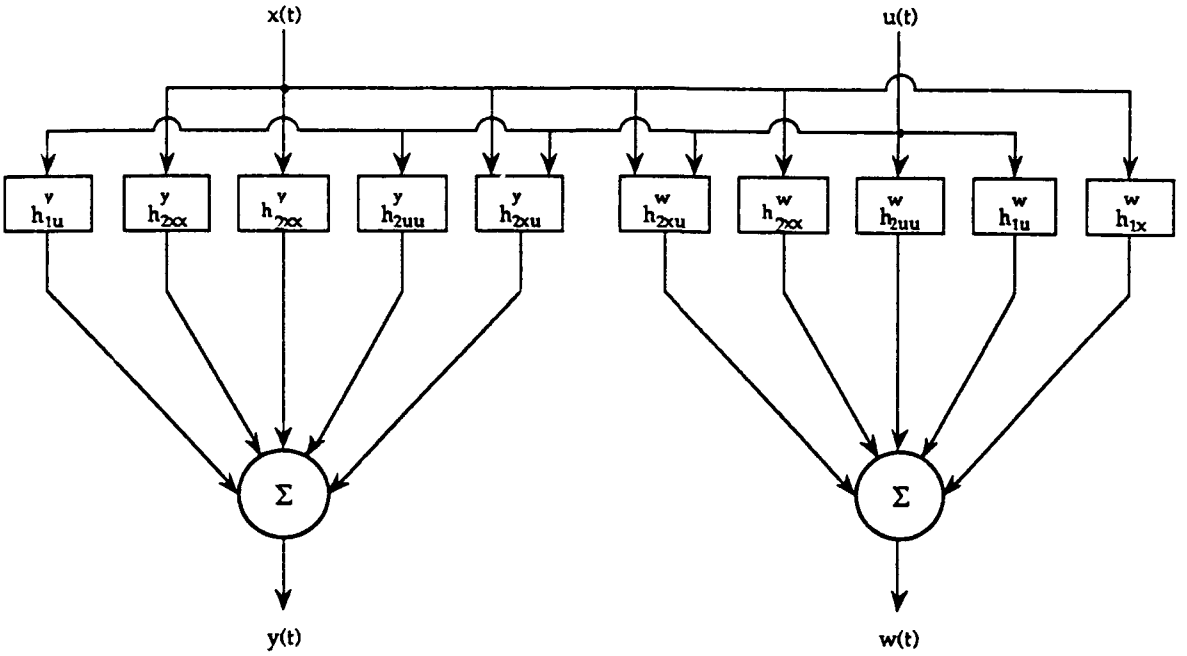


Figure 3.2: Kernel Diagram for Two-Input, Two-Output, Second-Order System [Marmarelis and Marmarelis 1978]

might be used to generate all possible input stimulation patterns. Since, in practice, all data is sampled using a finite resolution (i.e., number of bits), this is theoretically possible. This would reduce the time required to identify a system, since a white-noise signal is potentially repetitious and therefore redundant. For a single-input system, for example, using n-bit sampling would require 2^{2n^2} input patterns, which quickly results in an excessive number of input patterns over which the system must be excited. It is possible that the proportion of data collected in different regions of the stimulus space could be varied to reduce this number and to improve results where the identification process requires more temporal and/or spatial resolution. Once such a data base is generated, a PNN can be used to fit the data over the whole data base; this PNN would then represent the general (i.e., linear and nonlinear) transfer function of the system.

3.4 Reconciliation of Least Squares with Cross-Correlation Computation of Kernels

In practice, the cross-correlation identification technique outlined above works well for ideal GWN, where $\phi_{xx}(\tau) = P$ for $\tau = 0$, and $\phi_{xx}(\tau) = 0$ for $\tau \neq 0$. In practice, the stimulus signal is often not ideal GWN and the kernels require scaling to achieve accurate modeling. We may write Eq. 3:9 in discrete form as

$$\phi_{yx}(k) = \sum_{i=0}^{m} h_1(i)\ \phi_{xx}(i-k)\ \Delta t, \qquad \text{where } k = 0, 1, \ldots, m \tag{3:22}$$

Then, if we assume that $\phi_{xx}(j)$ is effectively zero for $j \geq n$,

$$\phi_{yx}(k) = \sum_{i=k-n}^{k+n} h_1(i)\ \phi_{xx}(i-k)\ \Delta t, \qquad \text{where } k = 0, 1, \ldots, \tag{3:23}$$

which can be written in matrix form as

$$\begin{bmatrix} \phi_{xx}(0) & \phi_{xx}(1) & \cdots & \phi_{xx}(n) & 0 & \cdots & 0 \\ \phi_{xx}(1) & \phi_{xx}(0) & \phi_{xx}(1) & \cdots & \phi_{xx}(n) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \phi_{xx}(n) & \phi_{xx}(n-1) & \cdots & \phi_{xx}(0) \end{bmatrix} \begin{bmatrix} h(0) \\ h(1) \\ \vdots \\ h(m) \end{bmatrix} = \begin{bmatrix} \phi_{yx}(0) \\ \phi_{yx}(1) \\ \vdots \\ \phi_{yx}(m) \end{bmatrix} \tag{3:24}$$

noting that $\phi_{xx}(i) = \phi_{xx}(-i)$ and denoting $\Delta t\ \phi_{xx}(i)$ by $\phi_{xx}(i)$. This system of equations is easily solved using batch or recursive least squares techniques. They represent a model of the form

$$\underline{y}(t) = \Phi\, \underline{\theta} \tag{3:25}$$

where $\underline{y}$ is the output vector, Φ is the input data matrix, and $\underline{\theta}$ is the parameter vector. We want to minimize the Fitting Squared Error (FSE) , given by

$$\Psi \equiv \frac{1}{n} \| \underline{y} - \Phi\underline{\theta} \|_2^2 \tag{3:26}$$

$\hat{\underline{y}} = \Phi\underline{\theta}$ is then the best approximation to $\underline{y}$ using the L_2 norm (least-squares). The data matrix, Φ, has variables in m columns and observations in n rows. To construct Φ for the combustion process data, the original $n \times 1$ time series is used to form the first row of Φ, and subsequent rows are formed by lagging the time series by one more than the previous row for each new row. Thus, if we are interested in identifying the model out to m lag values, Φ will be an $n \times m$ data matrix,

$$\begin{bmatrix} \phi_1 & \phi_2 & \phi_3 & \cdots & \phi_m \\ \phi_2 & \phi_3 & \phi_4 & \cdots & \phi_{m+1} \\ & & \vdots & & \\ \phi_n & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{3:27}$$

Multiplying $\Phi\underline{\theta} = \underline{y}$ by Φ^T produces $\Phi^T\Phi\underline{\theta} = \Phi^T\underline{y}$ or

$$\begin{bmatrix} \sum\phi_i^2 & \sum\phi_i\phi_{i+1} & \sum\phi_i\phi_{i+2} & \cdots & \sum\phi_i\phi_{i+m} \\ & & \vdots & & \\ . & . & . & \cdots & \sum\phi_{i+m}^2 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix} = \begin{bmatrix} \sum\phi_i y_i \\ \vdots \\ \sum\phi_{i+m} y_i \end{bmatrix} \qquad (3:28)$$

where $i = 1 \ldots n$. However, $\sum\phi_i^2 = \varnothing_{xx}(0)$, $\sum\phi_i\phi_{i+1} = \varnothing_{xx}(1)$, and

$\sum\phi_i\phi_{i+m} = \varnothing_{xx}(m)$; thus $\Phi^T\Phi\underline{\theta} = \sum_{i=k-n}^{k+n} h_1(i)\ \varnothing_{xx}(i-k)\,\Delta t$. Similarly,

$\sum\phi_i y_i = \varnothing_{yx}(0)$, $\sum\phi_{i+1} y_i = \varnothing_{yx}(1)$, and $\sum\phi_{i+m} y_i = \varnothing_{yx}(m)$,

so that $\Phi^T\underline{y} = \varnothing_{yx}(k)$, which illustrates that the cross-correlation and batch least squares equations are the same.

To solve this system of equations, define $S \equiv \Phi^T\Phi$, a symmetric (m x m) matrix that, hopefully, is positive definite. Define $\underline{b} \equiv \Phi^T\underline{y}$, the right side from above; so now, we solve $S\underline{\theta} = \underline{b}$. Note that s_{ij} is given by $\langle \underline{\phi}_i, \underline{\phi}_j \rangle$, where $\langle , \rangle$ denotes the inner (dot) product, and $\underline{\phi}_i$ is the ith column (ith variable) of the Φ matrix. Find the LU decomposition of S. $S = LU = LL^T$ since S is symmetric (Cholesky).

$$\begin{bmatrix} s_{11} & s_{12} & s_{13} & \cdots & s_{1m} \\ s_{21} & s_{22} & s_{23} & \cdots & s_{2m} \\ & & \vdots & & \\ s_{m1} & s_{m2} & s_{m3} & \cdots & s_{mm} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ & & \vdots & & \\ l_{m1} & l_{m2} & l_{m3} & \cdots & l_{mm} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1m} \\ 0 & u_{22} & u_{23} & \cdots & u_{2m} \\ & & \vdots & & \\ 0 & 0 & 0 & \cdots & u_{mm} \end{bmatrix} \qquad (3:29)$$

where $u_{ij} = l_{ji}$, and $s_{ii} > \sum_{k=1}^{i-1} l_{ik}^2$. Next, solve for the L matrix

$$i = 1 \ldots n \qquad \begin{cases} l_{ij} = \dfrac{1}{l_{jj}}\left(s_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) & i = 1, \ldots, m; \quad j = 1, \ldots, i-1 \\ l_{ii} = \sqrt{s_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \end{cases} \qquad (3:30)$$

To get $\underline{\theta}$, solve $L\underline{v} = \underline{y}$ and $U\underline{\theta} = \underline{v}$ (or $L^T\underline{\theta} = \underline{v}$). Solve for $\underline{v}$ first

$$v_i = \frac{1}{l_{ii}}\left(y_i - \sum_{j=1}^{i-1} l_{ij} v_j \right) \qquad i = 1, \ldots, m \qquad (3:31)$$

And then for $\underline{\theta}$

$$\theta_i = \frac{1}{l_{ii}}\left(v_i - \sum_{j=i+1}^{m} l_{ji} \theta_j \right) \qquad i = m, \ldots, 1 \qquad (3:32)$$

$s_{ii} \leq \sum_{k=1}^{i-1} l_{ik}^2$ implies that ϕ_i should be "chopped" from the matrix, since the matrix will not be positive-definite. θ_i should be set to zero (indicating that the ith input contains no new information). The S matrix should be reformed by removing the ith row and column, and b_i should be removed from $\underline{b}$ as well. Resolving the system should produce a solution, unless more chopping is needed. This process will have the same effect as a singular value decomposition, without going through the difficulties of using SVD.

The FSE, Ψ, can be found without direct recalculation as follows:

$$\Psi \equiv \frac{1}{n} \| \underline{y} - \Phi \underline{\theta} \|_2^2 \qquad (3:33)$$

$$\Psi = \frac{1}{n} \left[\| \underline{y} \|_2^2 + \| \Phi \underline{\theta} \|_2^2 - 2 \underline{y}^T \Phi \underline{\theta} \right] \qquad (3:34)$$

$$\Psi = \frac{1}{n} \left[\underline{y}^T y + \underline{\theta}^T \Phi^T \Phi \underline{\theta} - 2 \underline{y}^T \Phi \underline{\theta} \right] \qquad (3:35)$$

$$\Psi = \frac{1}{n} \left[\underline{y}^T y + \underline{\theta}^T S \underline{\theta} - 2 \underline{b}^T \underline{\theta} \right] \qquad (3:36)$$

Thus, if it is desirable to carve the coefficients, it is not necessary to solve for the estimated output $\hat{\underline{y}}$ to compute the FSE. The above equation gives Ψ in terms of $\underline{\theta}$ (assuming S and $\underline{b}$ are known).

Estimation of second-order and higher kernels follows a similar course. For example, for the second-order kernel, the system of equations to be solved is

$$\phi_{yxx}(k_1, k_2) = 2 \sum_{j=k_2-n}^{k_2+n} \sum_{i=k_1-n}^{k_1+n} h_2(i, j)\ \phi_{xx}(i - k_1)\ \phi_{xx}(j - k_2)\ \Delta t^2 \qquad (3:37)$$

where $k_1 = 0, 1, \ldots, m$ and $k_2 = 0, 1, \ldots, m$, which can be cast into a least squares formulation just as was demonstrated for the first-order kernel.

3.5 Reconciliation of Recursive Least Squares (RLS) with Cross-Correlation Computation of Kernels

Note that Eq. 3:25 may also be solved recursively with the arrival of each new observation, rather than in batch mode. Given a model of the form

$$\hat{y}(t) = \Phi(t)\,\underline{\theta}(t-1) \tag{3:38}$$

define

$\underline{\phi}(t)$, the input vector at time t.

$\underline{\theta}(t)$, the parameter vector at time t.

$\underline{K}(t)$, the gain vector at time t.

$P(t)$, the inverse Hessian at time t.

$\varepsilon(t)$, the residual at time t.

$\lambda(t)$, the forgetting factor at time t.

$y(t)$, the output at time t.

$s(t)$, the approximate number of samples in the learning window at time t.

Initially, $P(0)$, $\underline{\theta}(0)$, and $\lambda(t)$ must be set. $\lambda(t) \geq 1 - \frac{1}{s(t)}$, where $s(t)$ is the number of sample coefficients being estimated. Use of $\lambda < 1$ allows the parameters to adapt, which is useful in tracking nonstationary systems.

The following are calculated for each new $\underline{\phi}(t)$:

$$\varepsilon(t) = y(t) - \underline{\phi}^T(t)\underline{\theta}(t-1) \qquad \text{Prediction Error} \tag{3:39}$$

$$\underline{K}(t) = P(t)\underline{\phi}(t) = \frac{P(t-1)\underline{\phi}(t)}{\lambda(t) + \underline{\phi}^T(t)P(t-1)\underline{\phi}(t)} \qquad \text{Gain Vector} \tag{3:40}$$

$$P(t) = \frac{1}{\lambda(t)}\left[P(t-1) - \frac{P(t-1)\underline{\phi}(t)\underline{\phi}^T(t)P(t-1)}{\lambda(t) + \underline{\phi}^T(t)P(t-1)\underline{\phi}(t)}\right] \qquad \text{New Covariance} \tag{3:41}$$

$$\underline{\theta}(t) = \underline{\theta}(t-1) + \underline{K}(t)\varepsilon(t) \qquad \text{New Parameter Estimates} \tag{3:42}$$

To simplify, define

$$\tilde{\underline{K}}(t) \equiv P(t-1)\underline{\phi}(t) \tag{3:43}$$

$$D(t) \equiv \lambda(t) + \underline{\phi}^T(t)\tilde{\underline{K}}(t). \tag{3:44}$$

Now the equations become

$$\underline{K}(t) = \frac{\tilde{\underline{K}}(t)}{D(t)} \qquad \text{Gain Vector} \qquad (3:46)$$

$$P(t) = \frac{1}{\lambda(t)}\left[P(t-1) - \frac{\tilde{\underline{K}}\tilde{\underline{K}}^T}{D(t)}\right] \qquad \text{New Covariance} \qquad (3:47)$$

$$\underline{\theta}(t) = \underline{\theta}(t-1) + \underline{K}(t)\varepsilon(t) \qquad \text{New Parameter Estimates} \qquad (3:48)$$

4. Control

As shown above, the plant model may be identified using batch (off-line) or recursive (on-line) techniques. In practice, recursive least squares (RLS) techniques are useful for identifying the plant model when no *a priori data* is available. The system may also be "trained" off-line using existing data, which will improve control performance early in the identification process. If the plant changes with time, the RLS identification process can be used to track the plant by setting the RLS parameter $\lambda < 1$.

After estimating the plant, the next task is to construct a control system that allows the desired photodiode response (i.e., flame height or quality) to be set by the operator. Fig. 4.1 illustrates how the model identified using the techniques outlined in Section 3 will be used in the control system. The over-all strategy is based on inverse modeling.

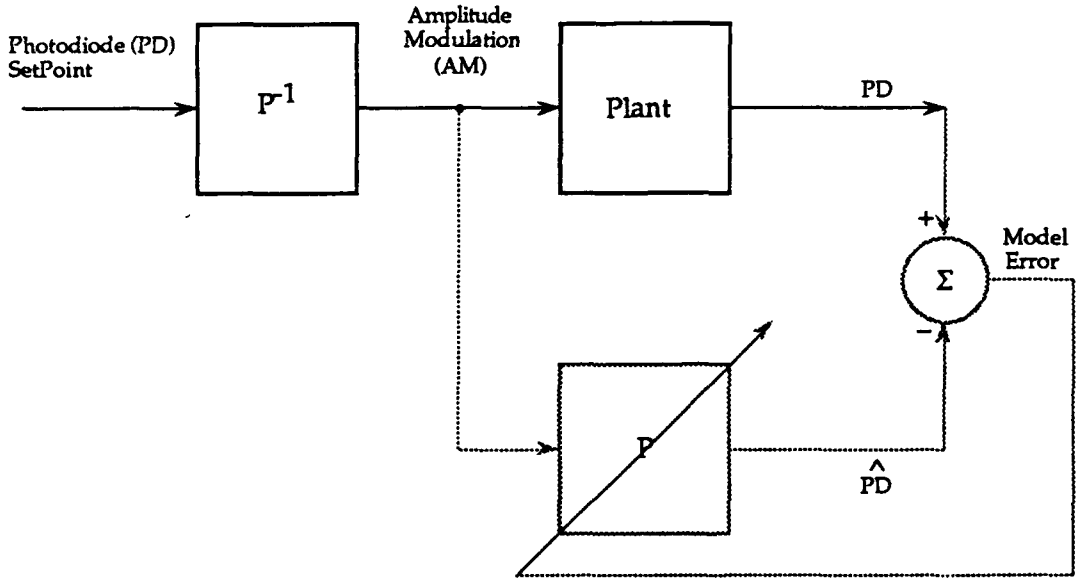


Figure 4.1: Control System Used to Regulate Flame Height or Quality

5. Results

5.1 Experimental Data

Combustion data were provided by NWC, which presumably were collected over a linear region of operation with respect to the amplitude modulation (AM) signal used to excite the system. The data consisted of six data files, each composed of two time series 16,384 samples in length, representing the input (AM) signal and system (photodiode) response. These data are summarized in Table 5.1. All data were collected with the system in an open-loop configuration; that is, the photodiode (PD) response was measured, but was not used to determine the subsequent excitation provided to the system.

Table 5.1: Summary of Experimental Data

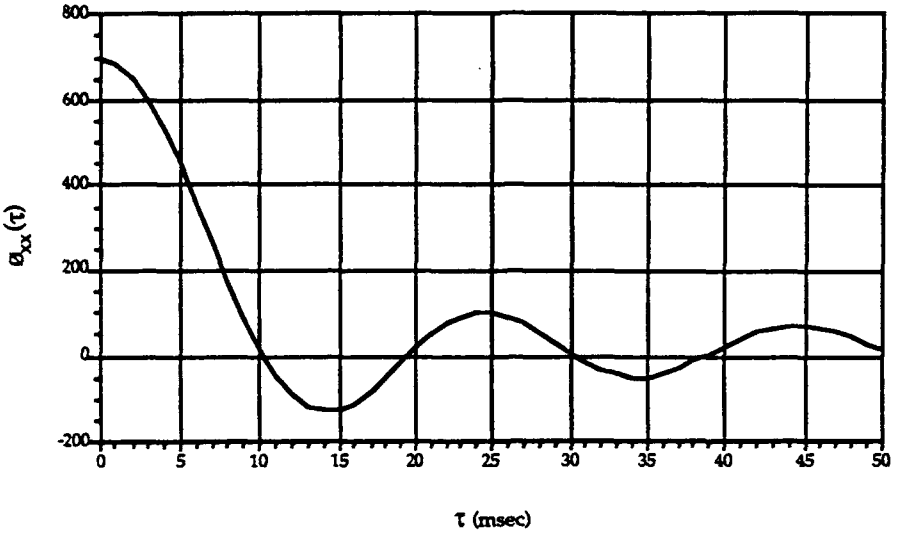
Data File	Data Length (samples)	Data Length (sec)
ACTD1.1	16,384	16.384
ACTD2.1	16,384	16.384
ACTD3.1	16,384	8.192
ACTD3.2	16,384	8.192
ACTD3.3	16,384	8.192
ACTD3.4	16,384	8.192

5.2 Plant Identification

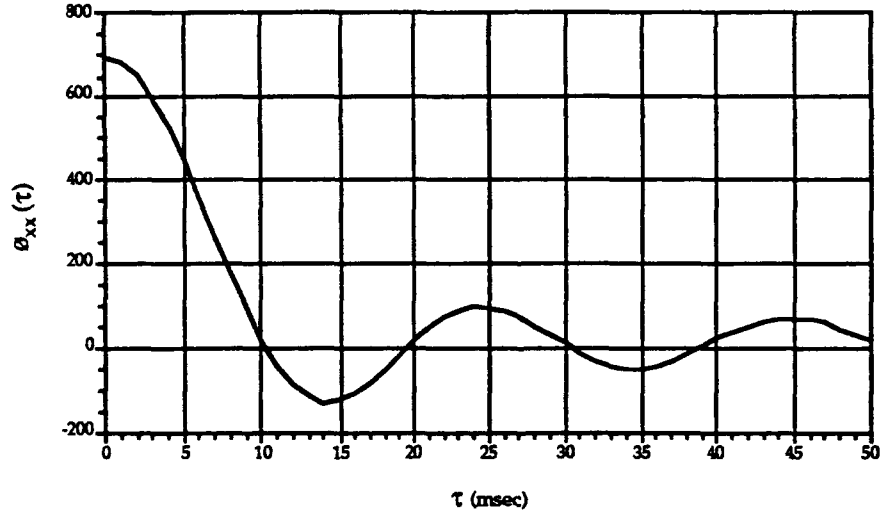
The auto-correlation functions of the AM excitation signal for the six data files are given in Fig. 5.1. For a GWN signal, the *second* zero crossing occurs at $\tau = \frac{1}{B}$, where B is the bandwidth of the signal, seen here to be equal to approximately 50 Hz for files ACTD 1.1 and 2.1, and approximately 400 Hz for files ACTD 3.1-3.4. The shape of the auto-correlation function also confirms that the input stimulus is Gaussian. Note also that $\phi_{xx}(0) = 2PB$, where P is the power or mean square value of the signal, which is seen to be about eight times larger for data sets ACTD1.1 and 2.1 than it is for data sets ACTD 3.1-3.4. Thus, the plant will be identified at different operating regions for the two groups of data.

The auto-correlation functions in Fig. 5.1 do not represent "ideal" GWN, which by definition has a significant value only at lag zero; achieving a more white-noise-like auto-correlation function would require a larger signal bandwidth. As a result, using the cross-correlation technique to directly (i.e., using Eq. 3:12) identify the plant model will result in scaling inaccuracies; this was confirmed using the experimental data. Kernels may be more accurately identified through the cross-correlation technique by solving Eqs. 3:24, which was shown above to be equivalent to using batch and recursive least squares. *Thus, system identification using Volterra-Wiener analysis is equivalent to modeling with memory-feedforward PNNs.*

Before proceeding to the plant identification and control results, we first mention a few interesting observations. It was found that direct (Eq. 3:12) application of the cross-correlation technique provided inaccurately scaled, but smooth, estimates of the first-order Wiener kernels. When this approach was compared to that obtained using Eqs. 3:24 or 3:28, depending on the number of auto-

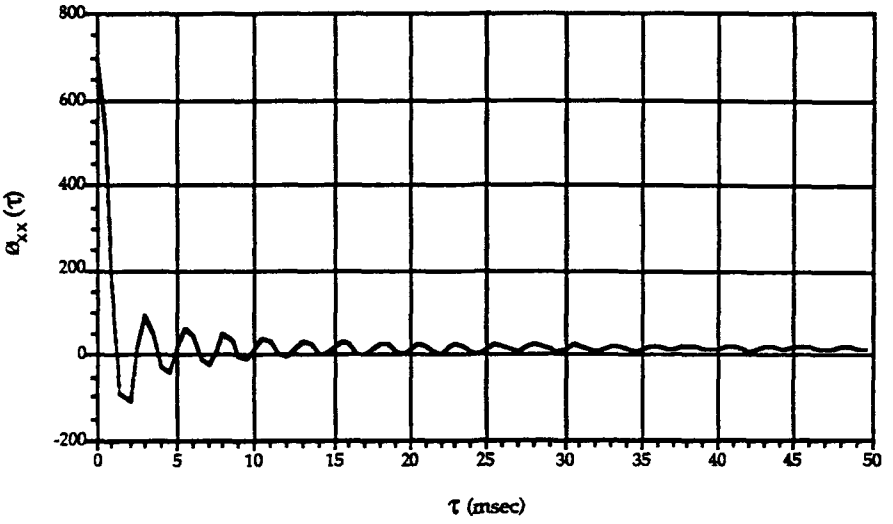


(a) ACTD 1.1

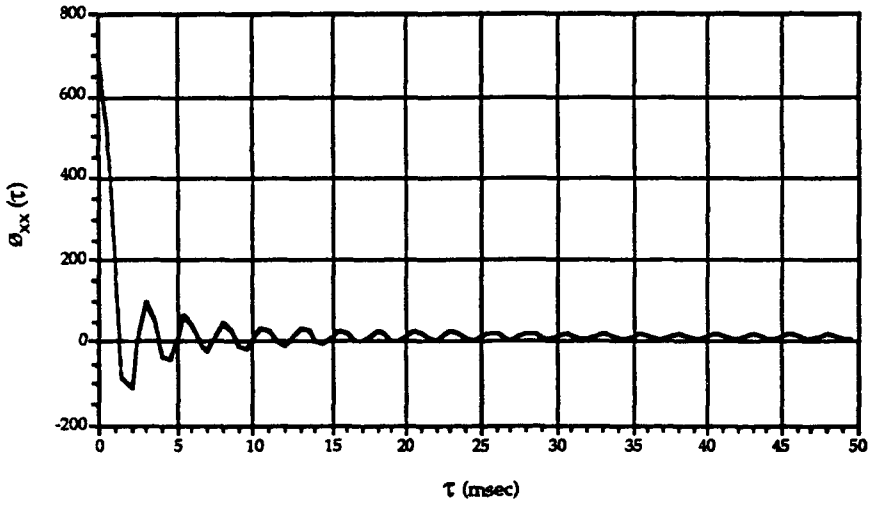


(b) ACTD 2.1

Figure 5.1: Auto-correlation Functions of Various Data Files

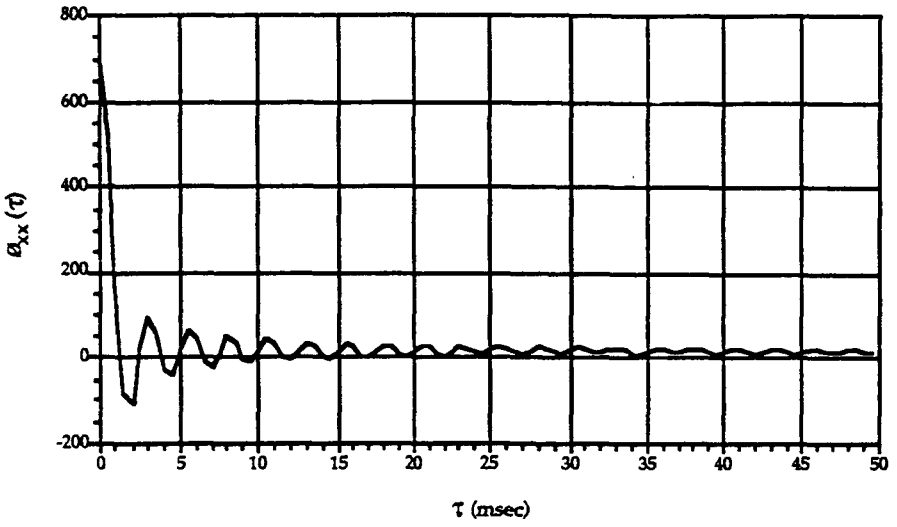


(c) ACTD 3.1

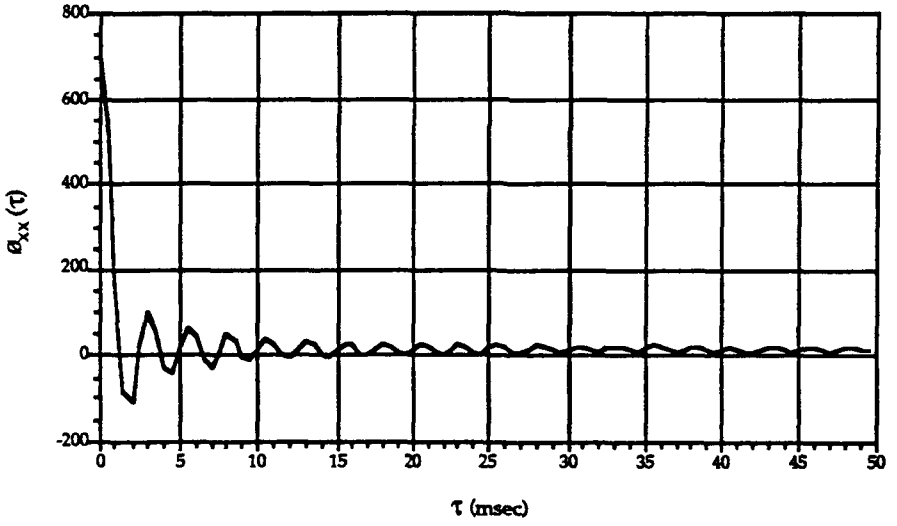


(d) ACTD 3.2

Figure 5.1 (continued): Auto-correlation Functions of Various Data Files



(e) ACTD 3.3



(f) ACTD 3.4

Figure 5.1 (continued): Auto-correlation Functions of Various Data Files

correlation lag values included in the data matrix, different results were obtained. Use of the auto-correlation value at lag zero only in the data matrix of Eqs. 3:24 and 3:28 is identical to using Eq. 3:12. Use of additional terms in the data matrix increasingly improves the kernel scaling; however, this is at the expense of kernel smoothness. By using additional terms in the data matrix, we move increasingly away from system identification towards curve fitting.

In an attempt to improve the smoothness of the kernel estimates, while still achieving good scaling, we used the Levenburg-Marquardt (LM) algorithm to fit the data. Heuristically, the approach is similar to fitting the data based on the least squares criterion; however, LM allows additional terms to be included in the objective function that must simultaneously be satisfied in the coefficient estimation process; the terms in the objective function may be linear or nonlinear. The additional term employed in the objective function was $[h_1(i) - h_1(i-1)]^2$, which penalizes large changes in the kernel estimates at contiguous lags. This criterion worked reasonably well, as can

be seen in Fig. 5.2, where the weighting (i.e., significance) of the smoothing penalty was varied from zero to 1000.

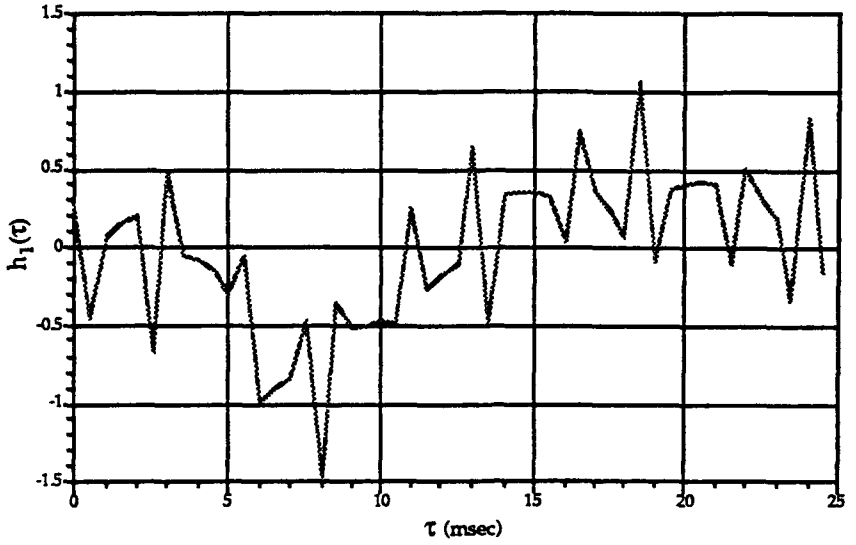
In the work below, due to the limited amount of experimental data available, no attempt was made to use nonlinearities in the prediction equations.

5.3 Memory-Feedforward Polynomial Neural Networks

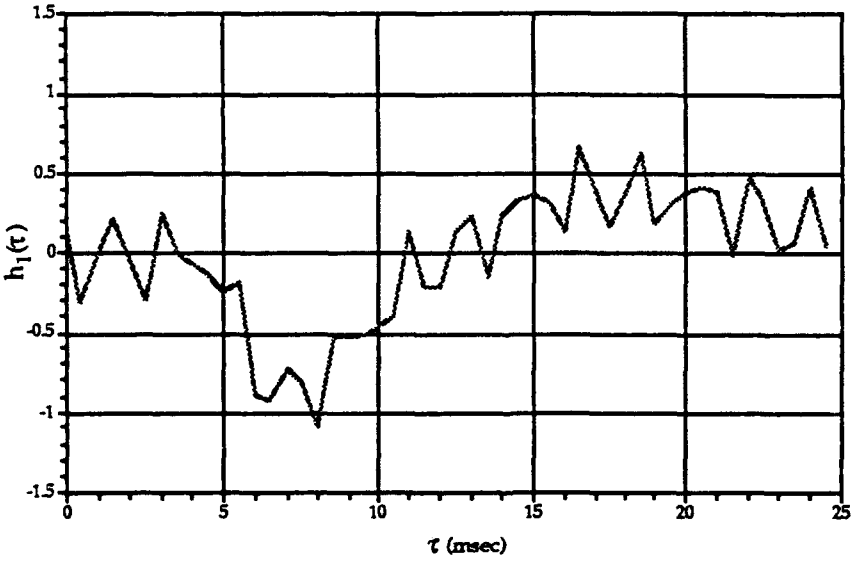
The architecture of the memory-feedforward polynomial neural network used to model the plant is given in Fig. 5.3. The first-order kernels computed using the LM technique with k=1000 are given in Fig. 5.4. The kernels were computed for 100 lag values, which if all are significant, represents the memory of the plant in each case. Increasing the number of lag values did not improve the estimation results. Note that for the ACTD 3.1-3.4 data, there is a delay of about 4 msec., which presumably represents the time it takes the AM excitation to affect the flame. This delay was *not* present in the ACTD 1.1 and 2.1 data sets. This may be due to estimation error, which is greater for these data files due to their AM excitation characteristics; it may also reflect differences in plant behavior at the operating region around which these systems were characterized due to power-level differences in the AM modulation. Note also that all data sets display a major negative peak; for files ACTD 3.1-3.4 the peak occurs at approximately 7 msec., whereas for files ACTD 1.1 and 2.1, this peak occurs at approximately 30 msec.

Fig. 5.5 illustrates the test setup used to evaluate how the first-order models performed in predicting the photodiode responses to AM excitation signals. Performance results are tabulated in Table 5.2 and in Figures 5.6 and 5.7, where each model is used to predict the photodiode response to the AM excitation data. Fig. 5.6 gives prediction results for the same AM excitation used to identify the model; Fig. 5.7 gives an example of the fitting performance when the AM excitation was different from that used (*viz.* file ACTD 3.1) to identify the plant. Fig. 5.6 thus corresponds to the diagonal terms in Table 5.2, and Fig. 5.7 corresponds to the off-diagonals in this table for the row labeled ACTD 3.1.

Note in Table 5.2, the data on the diagonal represents the performance on the fitting data, whereas off-diagonals represent the evaluation data. In each of the two operating regions around which the systems was characterized, the models performed nearly as well on the fitting data as they did on the evaluation data. The models have thus extracted the relevant information from the available data. For comparison purposes, similar data are provided in Table 5.3 where the smoothing penalty was set equal to zero; *this is equivalent to using the least squares fitting criterion.* Interestingly, *prediction accuracy was independent of the smoothness of the kernel even* when the plant in Fig. 5.5 was identified on a data file different from the one used for evaluation. Because of this

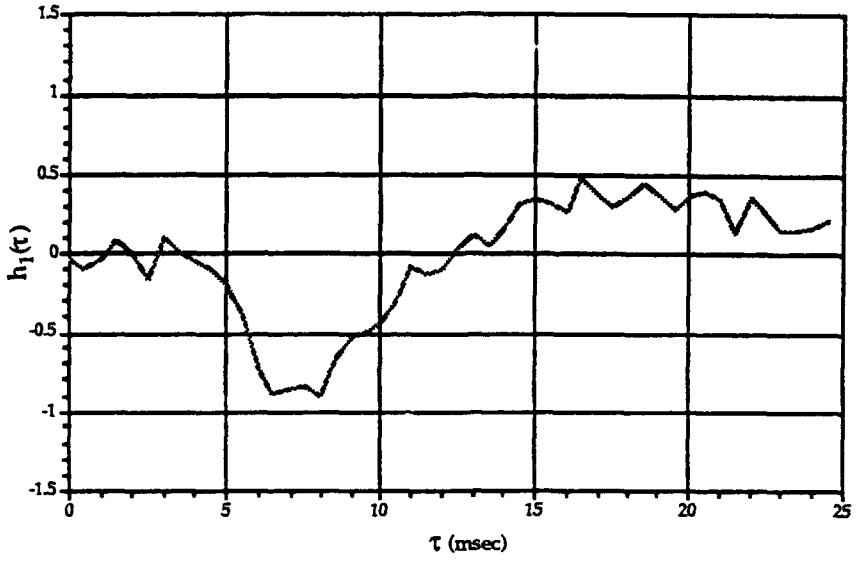


(a) k = 0

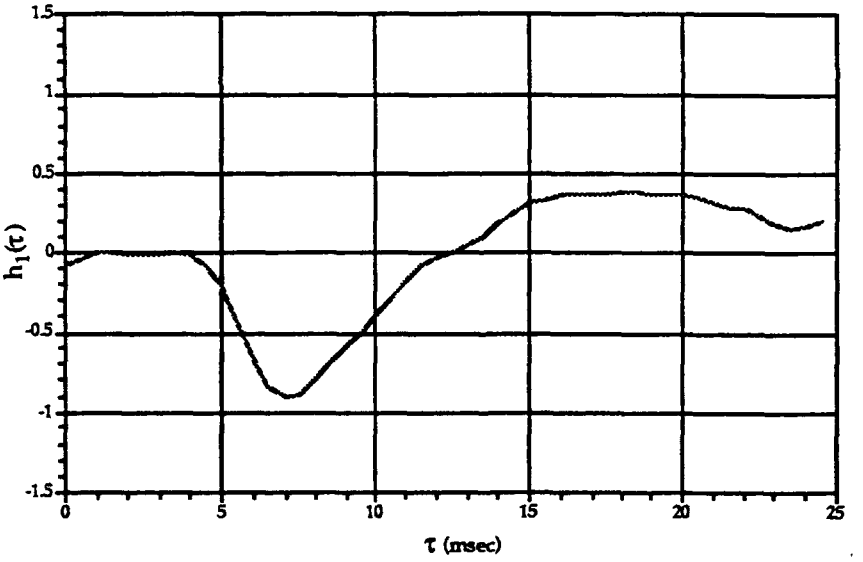


(b) k = 0.125

Figure 5.2: First-Order Kernel Computed by Levenberg-Marquardt Algorithm using Different Smoothing Penalty (k)



(c) k = 1



(d) k = 1000

Figure 5.2 (continued): First-Order Kernel Computed by Levenberg-Marquardt Algorithm using Different Smoothing Penalty (k)

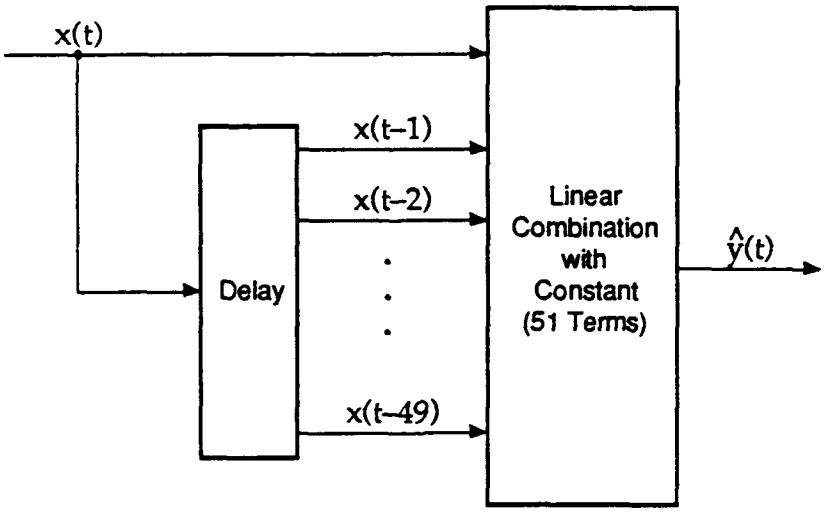


Figure 5.3: Architecture of Memory-Feedforward Polynomial Neural Network Used in the Experiments

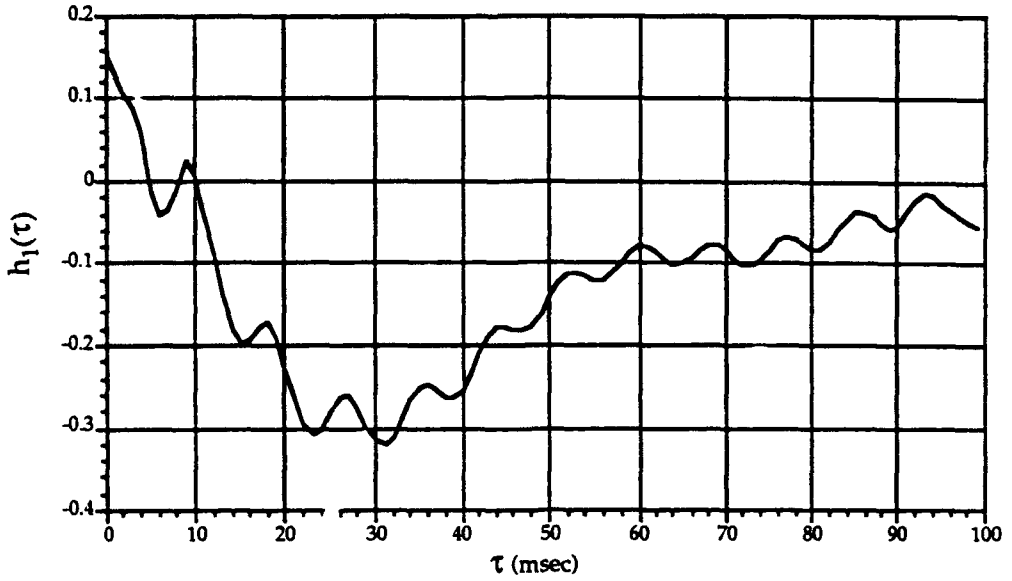
finding, either LM or least squares fitting could have been used for plant estimation. It should be mentioned that LM, like batch least squares, can also be formulated into a recursive implementation.

5.4 Memory-Feedback Polynomial Neural Networks

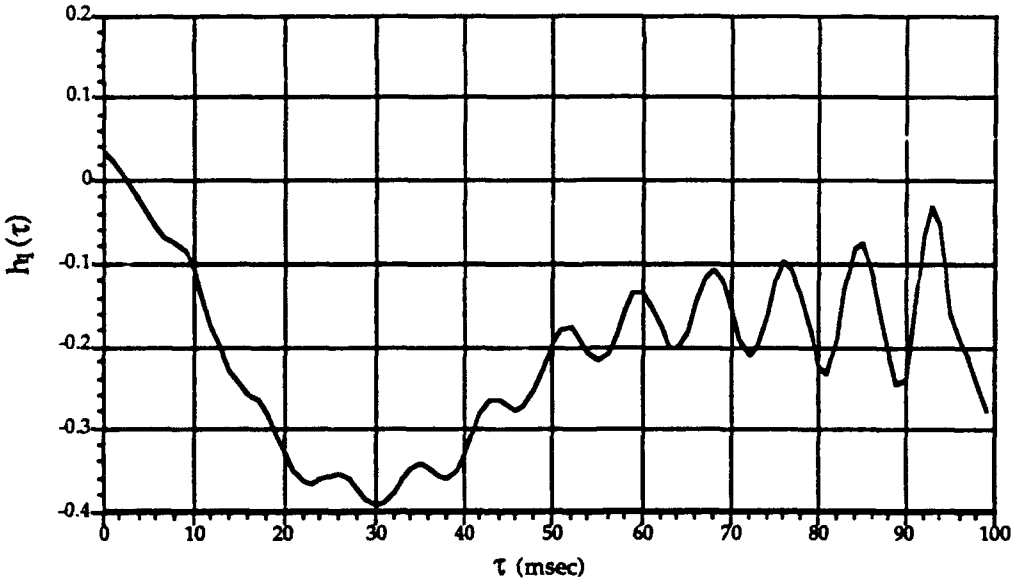
The architecture of the memory-feedback polynomial neural network used to model the plant is given in Fig. 5.8. Table 5.4 and Figures 5.9 and 5.10 provide the results for the memory-feedback PNNs used in the same experiment as conducted in Section 5.2. Note that performance is improved over the memory-feedforward case *even though the memory-feedback model had more than five times fewer internal degrees of freedom.*

5.5 Plant Control

The models identified in Section 5.2 were next used in the feedforward control system illustrated in Fig. 5.11. The plant in the figure was taken to be the PNN model for the file ACTD 3.1. The approach is to use the inverse of the identified model (i.e., P^{-1}) for each of the data sets in open-loop plant control. A unit step input signal was used to evaluate the performance of the controller; these results are given in Figures 5.12, where the step input and the system responses are plotted together. Note that in Fig. 5.12a, as expected, the desired and actual responses are the same; this is because PP^{-1}, by definition, is unity in this case. Due to differences in the estimated P^{-1} for the other data files, however, PP^{-1} is not unity. Because the plant is different for files ACTD 1.1 and 2.1, these responses are not shown.

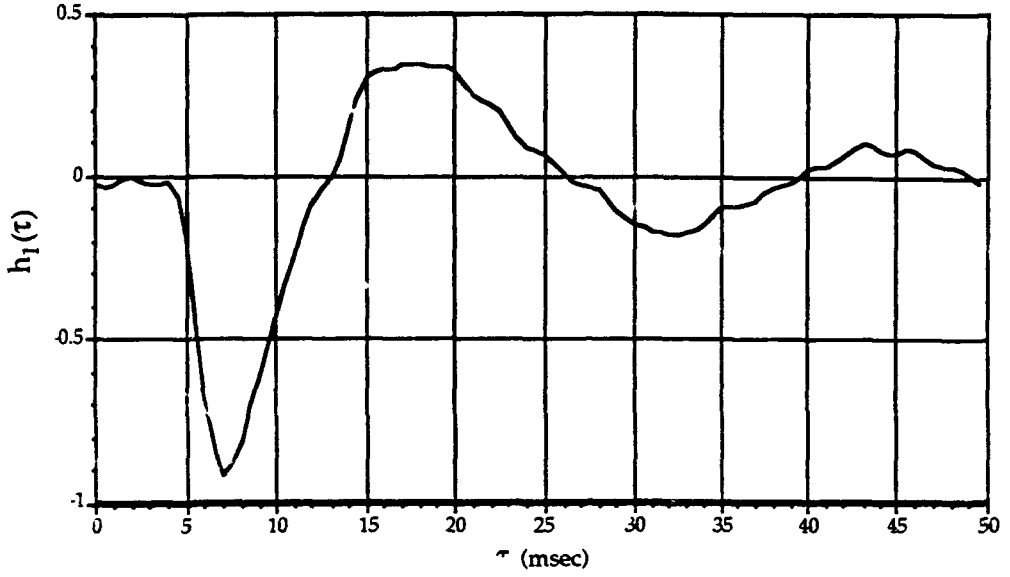


(a) ACTD 1.1

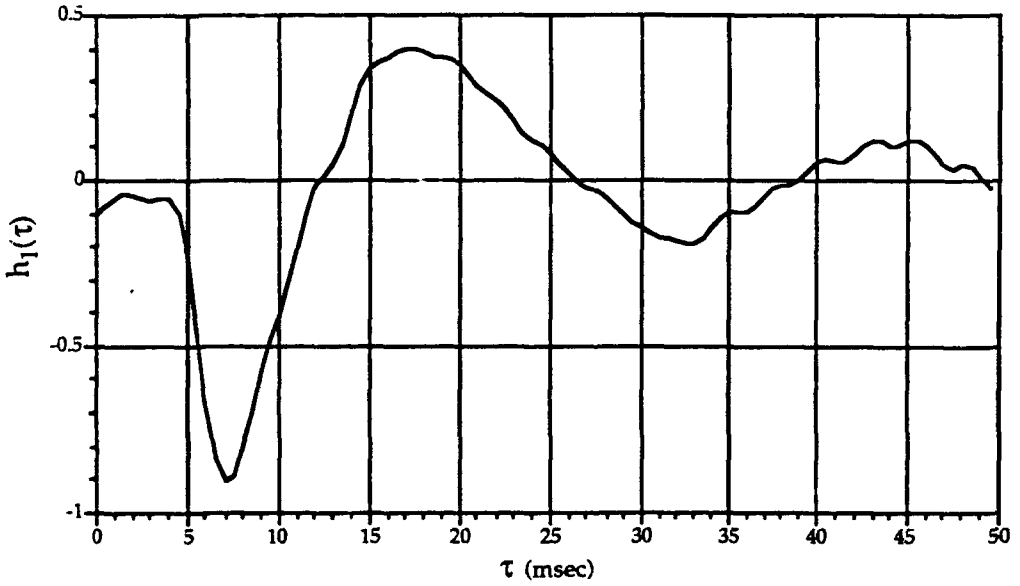


(b) ACTD 2.1

Figure 5.4: First-Order Kernels of Identified Plants Computed by Levenberg-Marquardt Algorithm Using Smoothing Penalty of k=1000

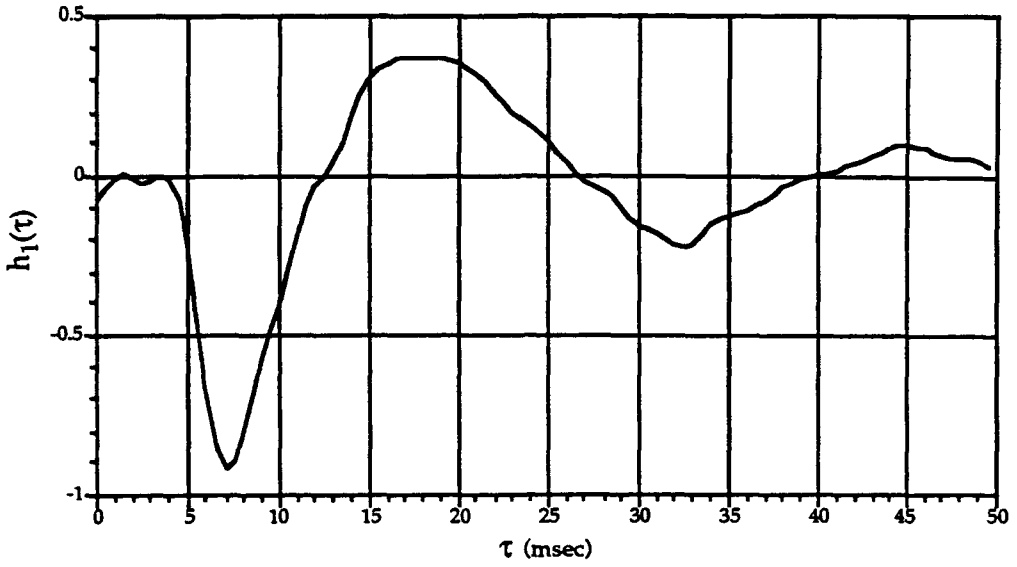


(c) ACTD 3.1

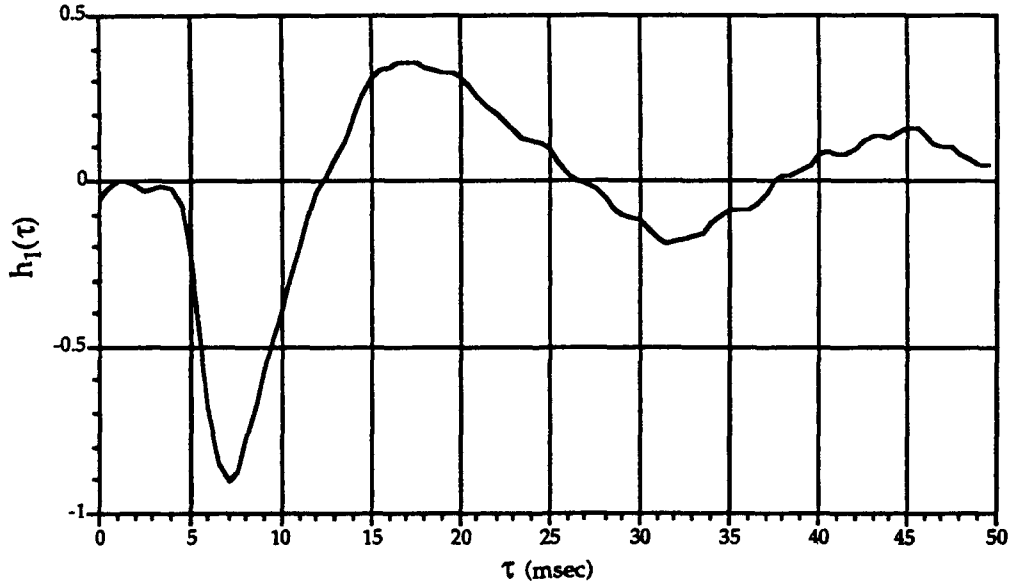


(d) ACTD 3.2

Figure 5.4 (continued): First-Order Kernels of Identified Plants Computed by Levenberg-Marquardt Algorithm Using Smoothing Penalty of k=1000



(e) ACTD 3.3



(f) ACTD 3.4

Figure 5.4 (continued): First-Order Kernels of Identified Plants Computed by Levenberg-Marquardt Algorithm Using Smoothing Penalty of k=1000

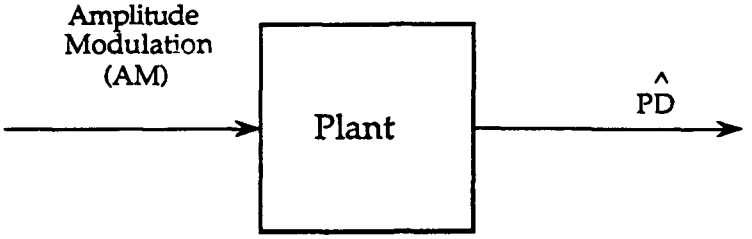


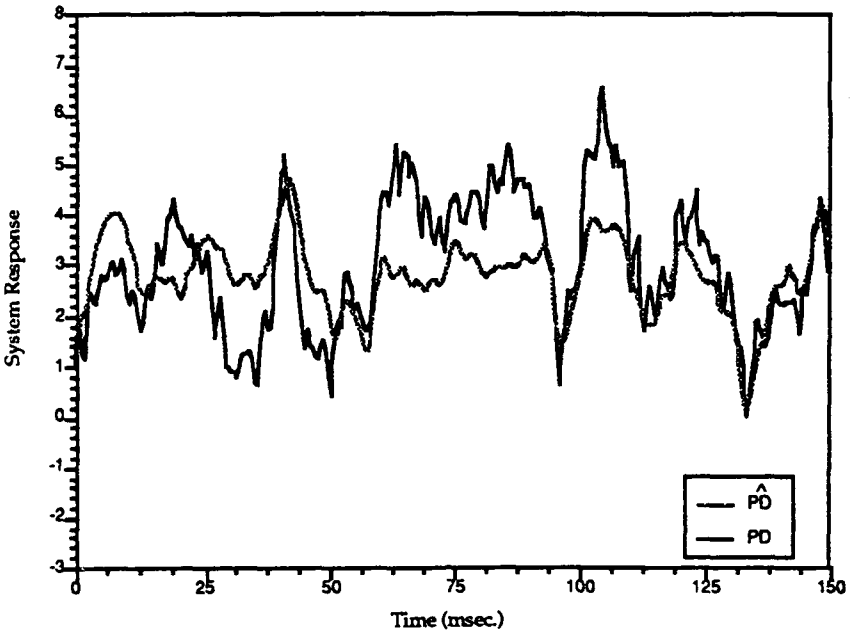
Figure 5.5: Use of Plant Model in Estimation Experiment

Table 5.2: Results of Plant Output Prediction Experiment Using Memory-Feedforward PNN With Smoothing Penalty k=1000

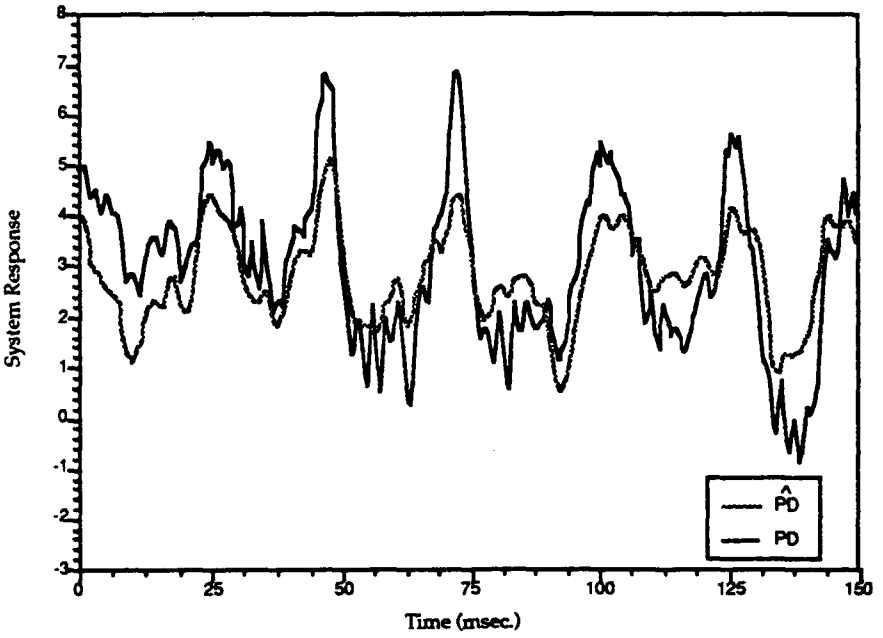
Eval → / Fit ↓	ACTD 1.1	ACTD 2.1	ACTD 3.1	ACTD 3.2	ACTD 3.3	ACTD 3.4
ACTD 1.1	**8.68**	10.38	15.87	16.89	15.94	17.14
ACTD 2.1	8.78	**10.27**	15.69	16.75	15.80	16.98
ACTD 3.1	25.72	27.37	**1.43**	1.61	1.57	1.47
ACTD 3.2	26.80	28.51	1.45	**1.59**	1.58	1.45
ACTD 3.3	25.83	27.55	1.44	1.61	**1.56**	1.48
ACTD 3.4	26.87	28.54	1.46	1.60	1.59	**1.44**

Table 5.3: Results of Plant Output Prediction Experiment Using Memory-Feedforward PNN With Smoothing Penalty k=0

Eval → / Fit ↓	ACTD 1.1	ACTD 2.1	ACTD 3.1	ACTD 3.2	ACTD 3.3	ACTD 3.4
ACTD 1.1	**8.66**	10.34	18.38	19.40	18.49	19.69
ACTD 2.1	8.78	**10.23**	33.01	34.07	33.19	34.34
ACTD 3.1	25.74	27.39	**1.43**	1.61	1.57	1.47
ACTD 3.2	26.81	28.52	1.45	**1.59**	1.58	1.45
ACTD 3.3	25.85	27.58	1.44	1.61	**1.56**	1.48
ACTD 3.4	26.88	28.56	1.46	1.61	1.60	**1.44**

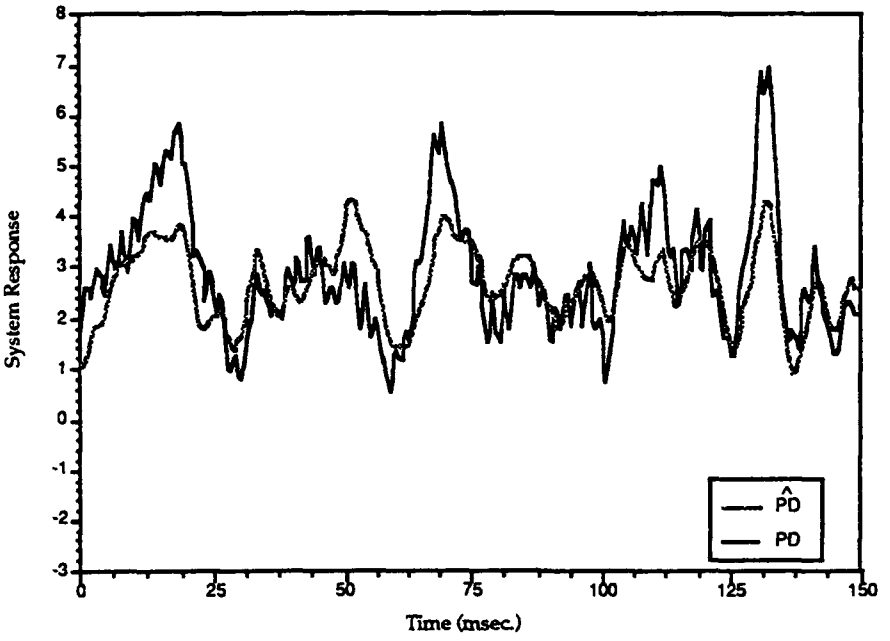


(c) ACTD 3.1

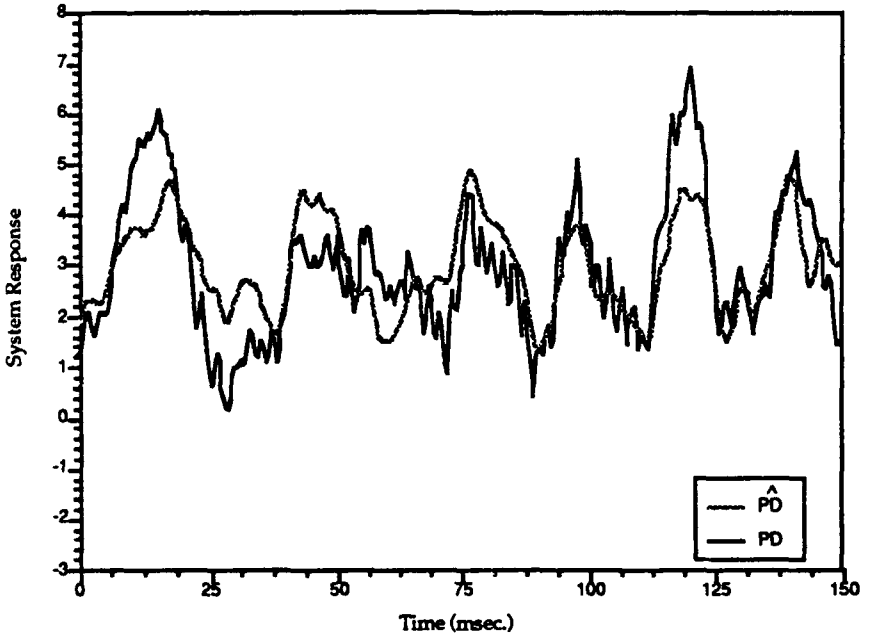


(d) ACTD 3.2

Figure 5.6: Memory-Feedforward PNN Prediction of Photodiode Output Using the Same AM Excitation Signal Used to Fit the Model

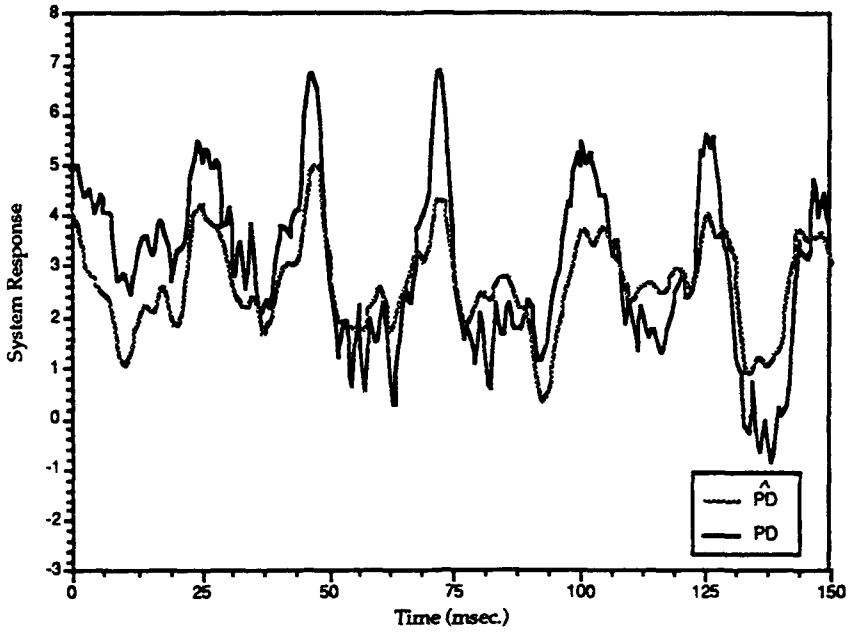


(e) ACTD 3.3

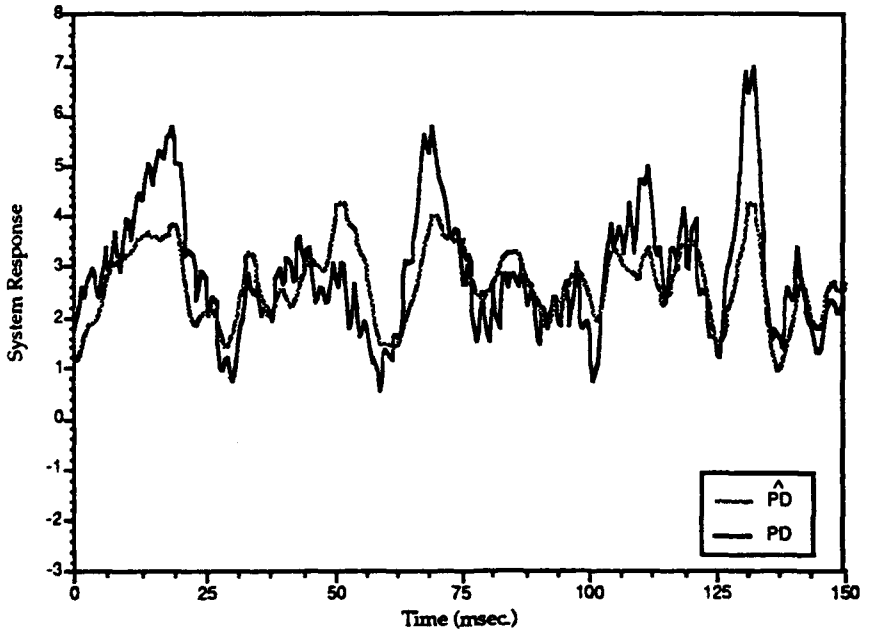


(d) ACTD 3.4

Figure 5.6 (continued): Memory-Feedforward PNN Prediction of Photodiode Output Using the Same AM Excitation Signal Used to Fit the Model

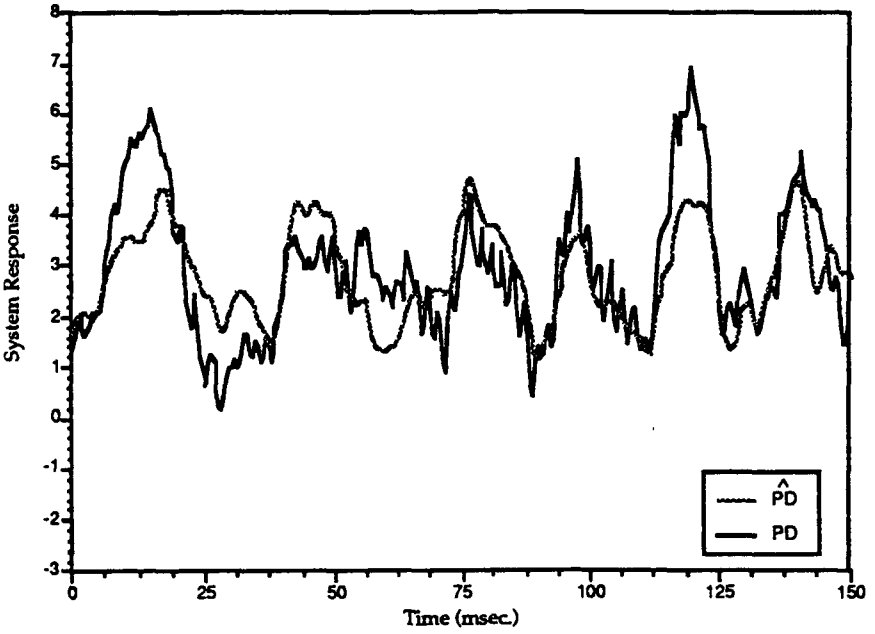


(c) ACTD 3.2



(d) ACTD 3.3

Figure 5.7: Memory-Feedforward PNN Prediction of Photodiode Output for AM Excitation Signals Different Than Was Used to Fit the Model (*viz.* file ACTD 3.1)



(e) ACTD 3.4

Figure 5.7 (continued): Memory-Feedforward PNN Prediction of Photodiode Output for AM Excitation Signals Different Than Was Used to Fit the Model (*viz.* file ACTD 3.1)

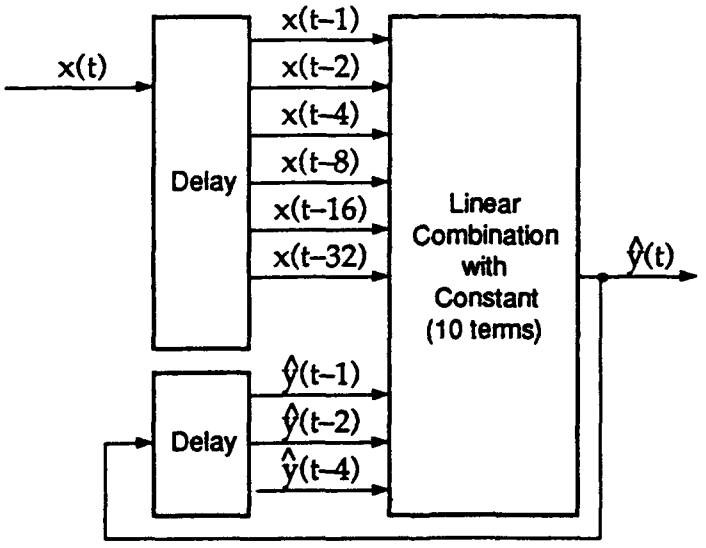
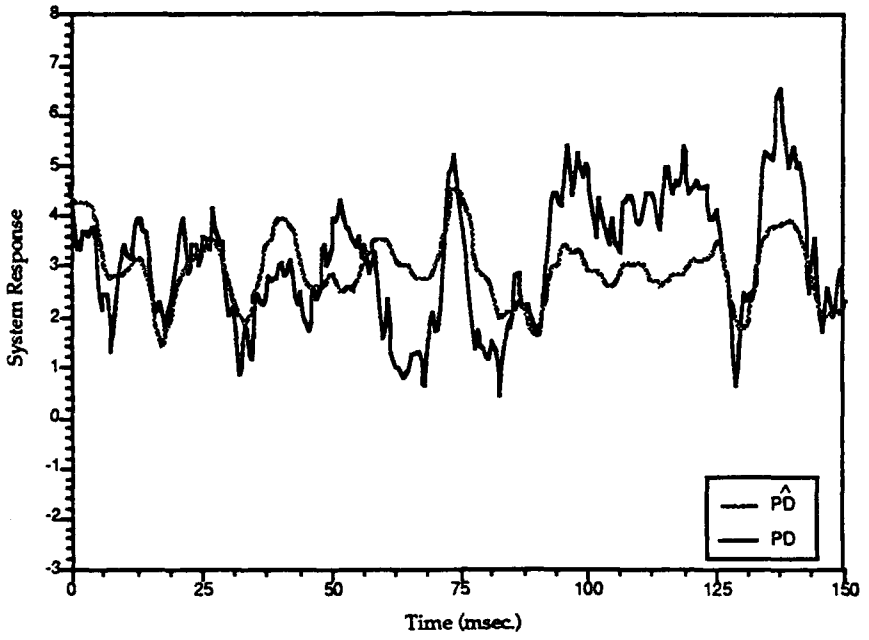
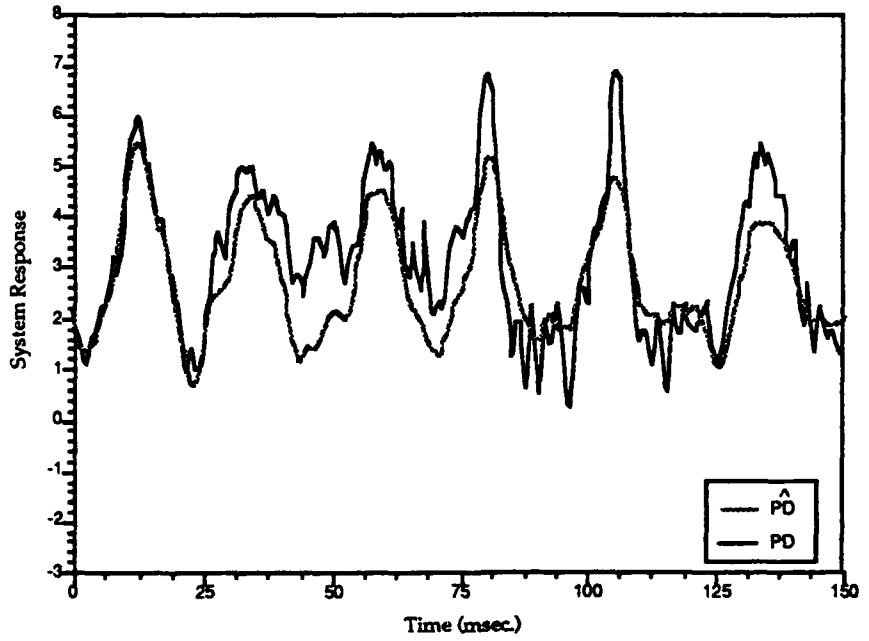


Figure 5.8: Architecture of Memory-Feedback PNN Used in the Experiments

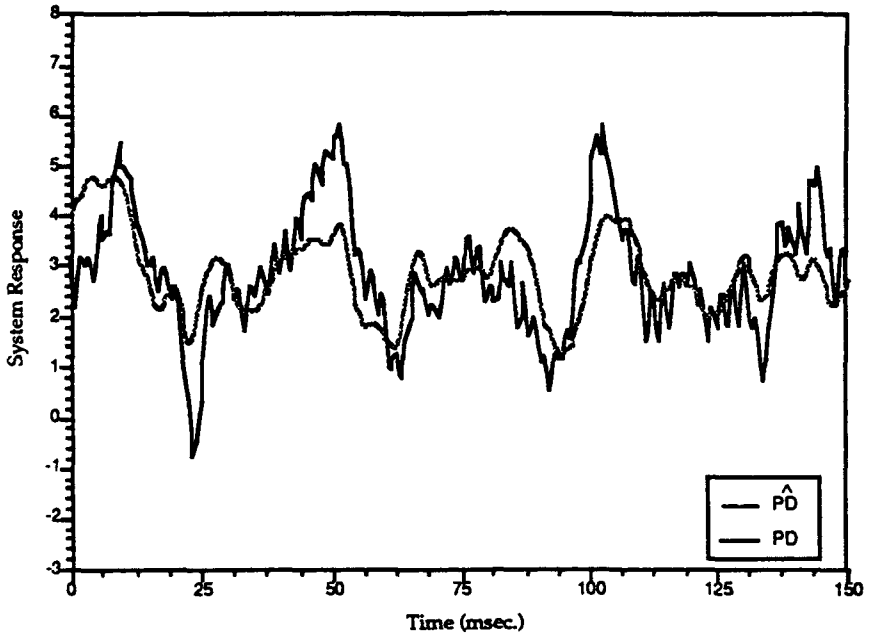


(c) ACTD 3.1

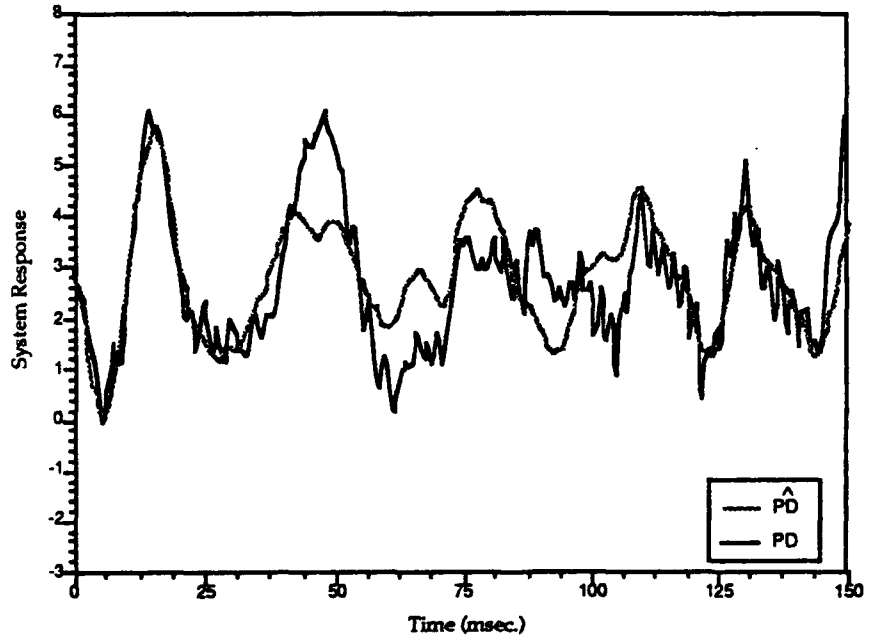


(d) ACTD 3.2

Figure 5.9: Memory-Feedback PNN Prediction of Photodiode Output Using the Same AM Excitation Signal Used to Fit the Model

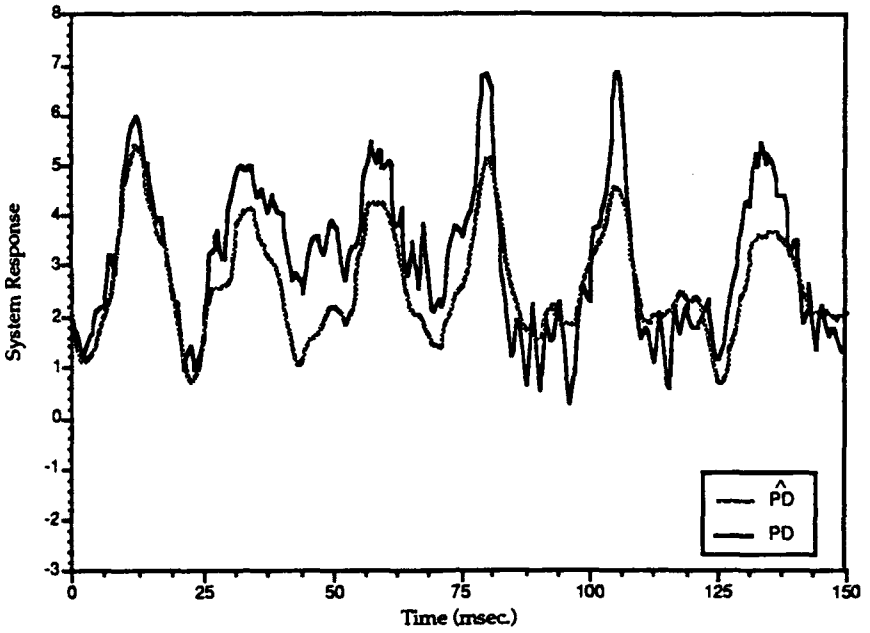


(e) ACTD 3.3

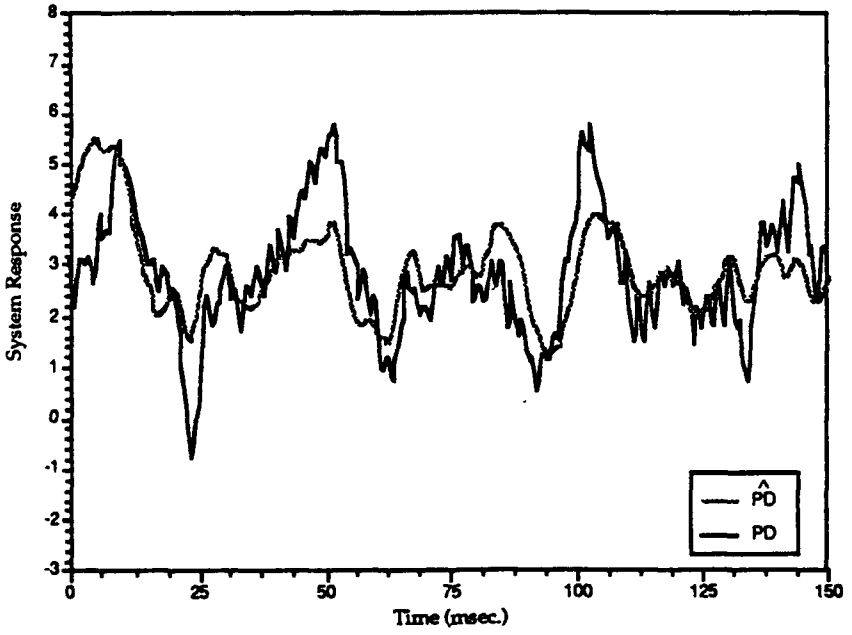


(f) ACTD 3.4

Figure 5.9 (continued): Memory-Feedback PNN Prediction of Photodiode Output Using the Same AM Excitation Signal Used to Fit the Model

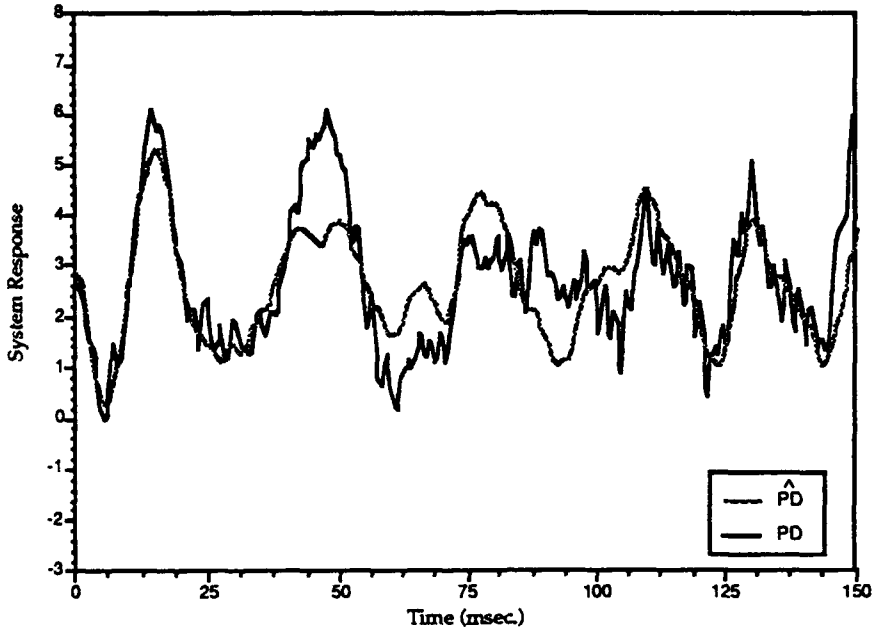


(c) ACTD 3.2



(d) ACTD 3.3

Figure 5.10: Memory-Feedback PNN Prediction of Photodiode Output Using Different AM Excitation Signal Than Was Used to Fit the Model (*viz.* ACTD 3.1)



(e) ACTD 3.4

Figure 5.10 (continued): Memory-Feedback PNN Prediction of Photodiode Output for AM Excitation Signals Different Than Was Used to Fit the Model (*viz.* file ACTD 3.1)

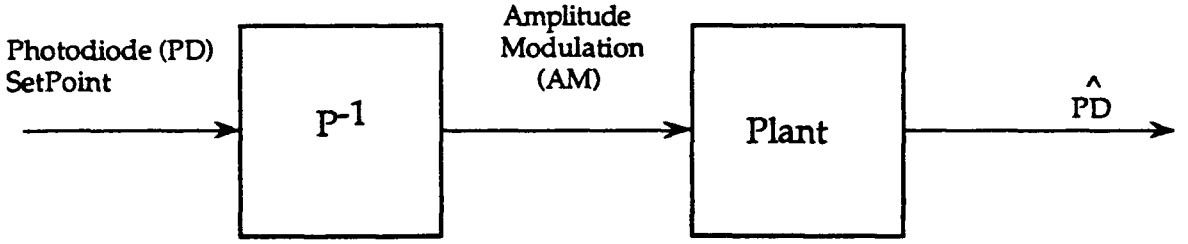
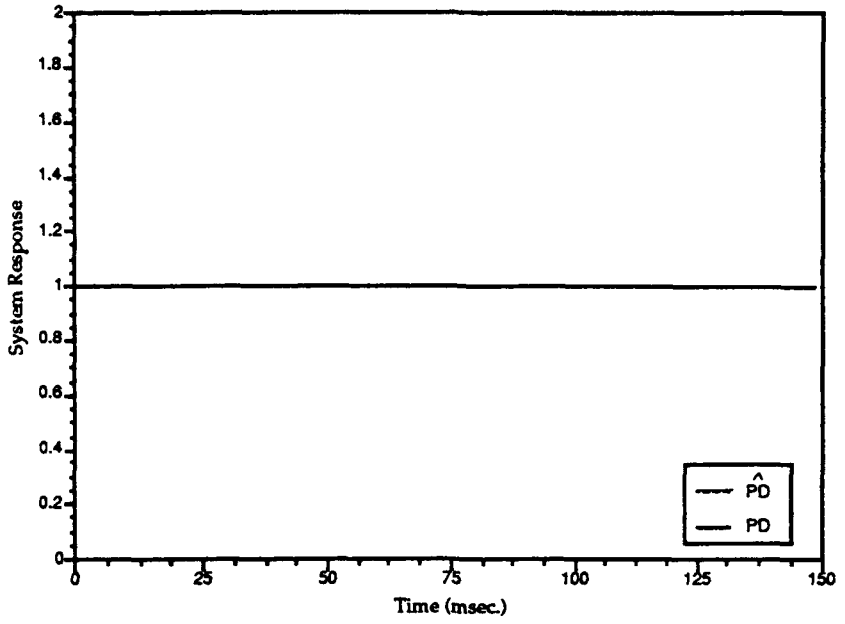
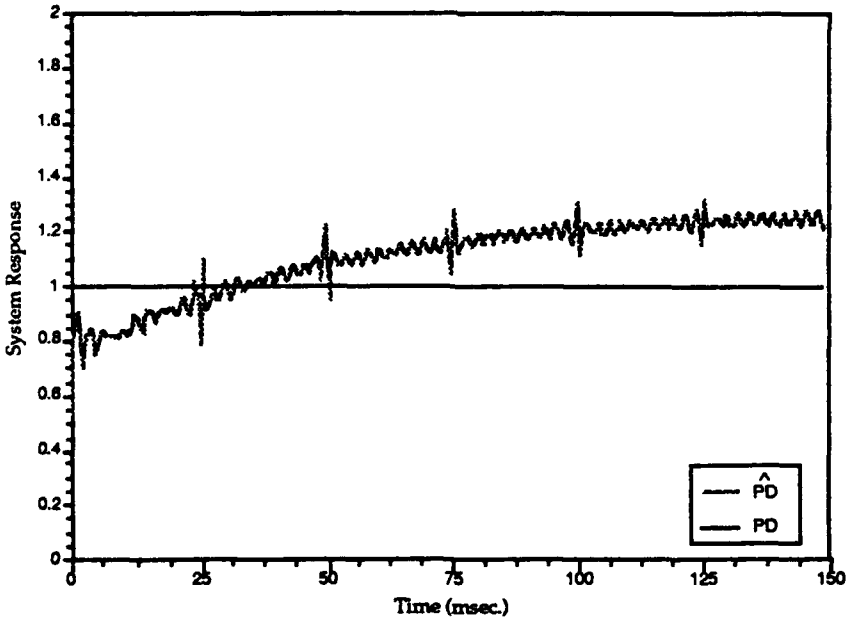


Figure 5.11 Use of Model Inverse in Feedforward Control Experiment

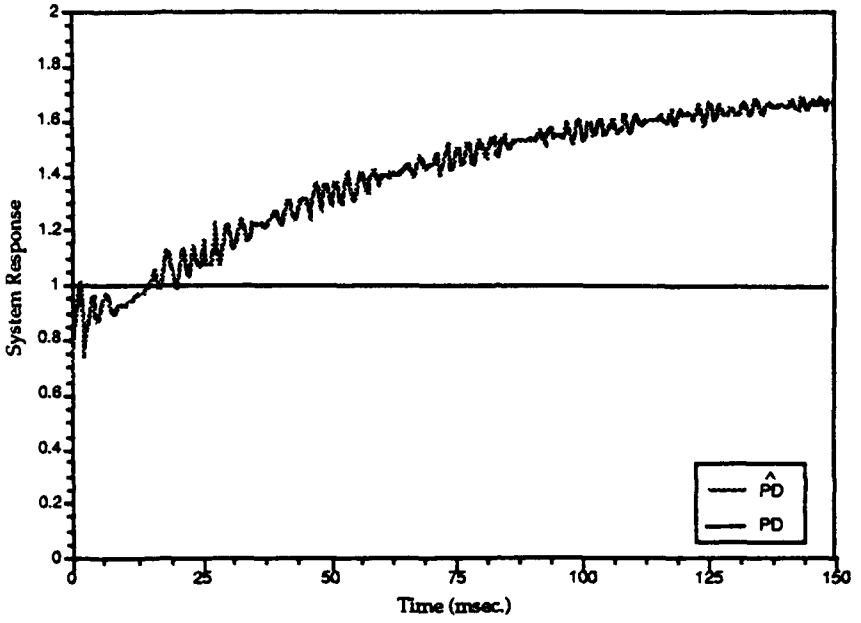


(a) ACTD 3.1

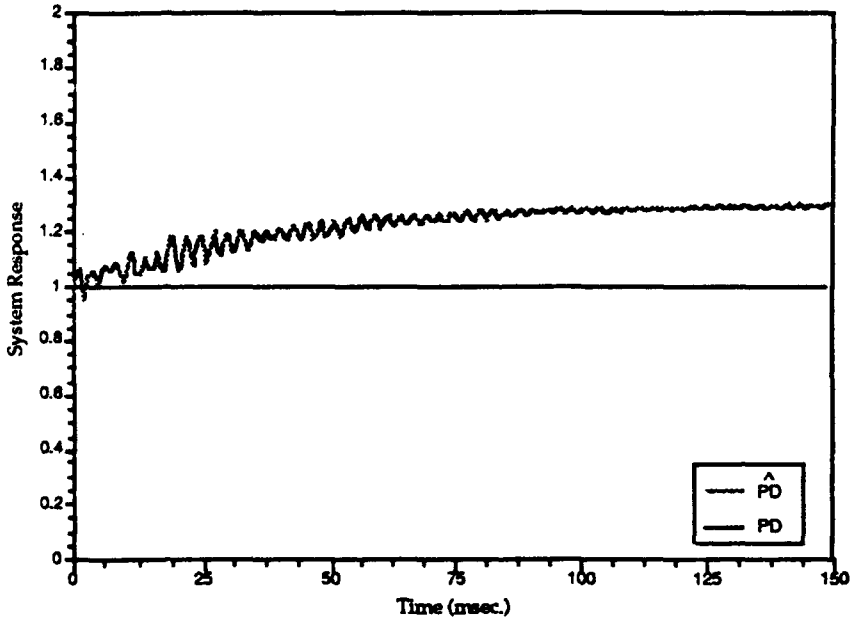


(b) ACTD 3.2

Figure 5.12: Unit Step Input and Response of Feed-Forward Controller



(c) ACTD 3.3



(d) ACTD 3.4

Figure 5.12 (continued): Unit Step Input and Response of Feedforward Controller

Table 5.4: Results of Plant Output Prediction Experiment Using Memory-Feedback PNN

Eval → / Fit ↓	ACTD 1.1	ACTD 2.1	ACTD 3.1	ACTD 3.2	ACTD 3.3	ACTD 3.4
ACTD 1.1	8.52	9.91	15.74	16.68	15.90	19.54
ACTD 2.1	8.99	9.50	15.85	17.01	16.05	17.71
ACTD 3.1	25.86	27.42	1.40	1.56	1.53	1.43
ACTD 3.2	26.87	28.42	1.42	1.54	1.54	1.42
ACTD 3.3	25.99	27.61	1.40	1.56	1.52	1.44
ACTD 3.4	26.98	28.55	1.43	1.55	1.56	1.40

6. Discussion

Memory-feedback PNN performance was shown to exceed that of memory-feedforward PNNs in modeling stochastic combustion process data *while using fewer coefficients to fit the data.* This is analogous to infinite impulse response modeling (IIR) as compared to finite impulse response (FIR) modeling. It is also similar to curve fitting using polynomial functions where, with enough coefficients, curves can be made to pass *exactly* through all data points. Using a large numbers of coefficients, however, limits the performance of such models on previously unseen evaluation data. In curve-fitting, this is reflected by large oscillations in the fitting functions in between data points and outside of the region used to fit the data (e.g., extrapolation). *Fewer internal degrees of freedom provide more robust models.*

Memory-feedforward PNNs were shown to be a generalization of the classical Volterra-Wiener modeling approach. Note that if previous measurements of estimated quantities (e.g., PD measurements in the combustion process) are available, these may also be used to estimate future outputs; such a model is said to include autoregressive terms. In modeling a process, say $y = f(x)$, this information is not inherently useful since y is completely determined by some (in general nonlinear) function of x. However, for a model $y = f(x,y)$, autoregressive information is particularly useful in that y is necessarily a nonlinear function of y, otherwise $y = f(x,y)$ could be rewritten as $y = f(x)$. Another way to think about the usefulness of autoregressive terms is that they provide information exactly (not approximately as for lagged values of $\hat{y}$) concerning the region of operation of the system to be taken into consideration in the modeling process. This allows different behavior at different operating points in general to be modeled.

Dynamic PNNs go beyond Volterra-Wiener modeling because the latter are limited to application in finite-memory systems. Thus, dynamic PNNs may be used, for example, to model oscillators as naturally as they model finite-memory systems. For example, the second-order constant-coefficient ordinary differential equation (ODE)

$$\ddot{y}(t) + a_1 \dot{y}(t) + a_2 y(t) = b_0 x(t)$$
$$x(t) = \cos(2t)$$

with initial conditions

$$\dot{y}(0) = \dot{y}_0$$

and

$$\dot{y}(0) = \dot{y}_0$$

has the solution

$$y(t) = c_1 \exp(\lambda t)\cos(\mu t) + c_2 \exp(\lambda t)\sin(\mu t) + A \sin(2t) + B \cos(2t)$$

where

$$\lambda = -\frac{a_1}{2} \qquad \mu = \frac{\sqrt{4 a_2 - a_1}}{2}$$

$$A = \frac{2a_1 b_0}{4 a_1^2 + a_2^2 - 8 a_2 + 16} \qquad B = \frac{(a_2 - 4) b_0}{4 a_1^2 + a_2^2 - 8 a_2 + 16}$$

$$c_1 = y_0 - B \qquad c_2 = \frac{\dot{y}_0 - 2A - \lambda c_1}{\mu}$$

Using a finite-difference approximation to the ODE, a dynamic PNN of the form

$$\hat{y}_n = \theta_0 \hat{y}_{n-1} + \theta_1 \hat{y}_{n-2} + \theta_2 x_{n-1}$$

where

$$\theta_0 = \frac{2b_0}{2 + a_1 \Delta t}$$

$$\theta_1 = \frac{4 - 2a^2 \Delta t^2}{2 + a_1 \Delta t}$$

$$\theta_2 = 1 - \frac{4}{2 + a_1 \Delta t}$$

can be used to *exactly* predict the output [Cellucci 1991]. To exactly model such as system using an FIR model would require an infinite number of coefficients, i.e., infinite memory. The only way around this is to accept a less accurate model using a finite number of coefficients or to use a change of variable of the input and output signals to remove the oscillatory behavior.

In addition, PNNs allow easy incorporation of any available *a priori* structural information in the modeling process, allowing very flexible modeling. *A priori* information is much more difficult to take advantage of using other modeling techniques, such as in Volterra-Wiener system identification, when complete structural information is not available.

7. Future Directions

The next step in the investigation of dynamic PNNs will be to include autoregressive terms (i.e., previous values of the output process being predicted) in the plant control model. Further accuracy is expected since this represents access to actual photodiode measurements rather than just their estimates, which may further reduce the number of parameters that must be estimated. This may allow second-order models to be computed even with the limited data that are currently available.

Note, however, that using autoregressive terms in an emulator is not possible, since there is no measurement of the actual output process available. The final design will then be used in a closed-loop controller of the actual combustion process.

The possibility of exciting the system using selected stimulus patterns, rather than GWN, should be taken into consideration in future experiments on the actual combustion process. This may allow better modeling accuracy to be achieved, as discussed in Section 3.3 of this report.

Additional data will help improve the modeling accuracy that has been achieved to date. Further investigation will be required to determine if the estimate of the plant model can be improved upon with the inclusion of nonlinear terms.

8. References (Appendix E)

Barron, A.R. Properties of the Predicted Squared Error: A Criterion for Selecting Variables, Ranking Models, and Determining Order, Adaptronics, Inc., McLean, VA, 1981.

Barron, A.R. "Predicted Squared Error: A criterion for automatic model selection," *Self-Organizing Methods in Modeling: GMDH Type Algorithms* (S.J. Farlow, Ed.), Marcel Dekker, Inc., NY, Chap. 4, pp. 87-103, 1984.

Barron, A.R. and R.L. Barron. "Statistical learning networks: A unifying view," *Proc. 20th Symposium on the Interface: Computing Science and Statistics*, Reston, VA, April 1988.

Barron, R. L., R. L. Celluci, P. R. Jordan, III, *et al.*, Applications of polynomial neural networks to FDIE and reconfigurable flight control," *Proc. 42nd Ann. NAECON*, May 1990.

Cadzow, J,A., Signal Processing via Least Squares Error Modeling, *IEEE Signal Processing Magazine*, Oct. 1990.

Cellucci, R.L., Barron Associates, Inc. Internal Memorandum, 1991.

Farley, B.G. and W.A. Clark, "Simulation of self-organizing systems by digital computers," *IRE Trans. on Inform. Theory*, Vol. PGIT-4, pp. 76-84, 1954.

Ljung, L. and T. Söderström, *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA, 1983.

Ljung, L., "Adaptation using recursive parameter estimation," Chapter 8 in *Advances in Statistical Signal Processing - Vol. 1: Estimation.*, H. V. Poor, Ed. JAI Press; Greenwich, CT, 1987.

Marmarelis, P.Z. and V.Z. Marmarelis, *Analysis of Physiological Systems: The White-Noise Approach*, Plenum Press, New York, 1978.

Nigro, T. M., D. W. Abbott, and R. L. Barron, *Comparative Study: Static and Dynamic Polynomial Neural Networks for Real-Time Optimum TPBV Guidance of a Tactical Air Intercept Missile*, Barron Associates, Inc. Technical Progress Rept. 2 for Office of Naval Research, Contract N00014-89-C-0137, April 1990.

Rosenblatt, F., "The Perceptron," *Cornell Aeronaut. Lab. Rept.* VG-1196-G-1, Jan. 1958.

Söderström, T. and P. Stoica, *System Identification*, Prentice Hall Int'l. (UK) LTD, 1989.

Widrow, B. and M.E. Hoff, "Adaptive switching circuits," *IRE Wescon Convention Record*, pp. 96-104, 1960.