A98-37163 AIAA-98-4440

ACCELERATED CONVERGENCE OF NEURAL NETWORK SYSTEM IDENTIFICATION ALGORITHMS VIA PRINCIPAL COMPONENT ANALYSIS

David C. Hyland
Department of Aerospace Engineering
The University of Michigan, Ann Arbor, MI

Lawrence D. Davis
Planning Systems, Inc.
Melbourne Controls Group
Melbourne, Florida

Keith K. Denoyer
Air Force Research Laboratory
Space Vehicles Directorate
Kirtland AFB, NM

Abstract

While significant theoretical and experimental progress has been made in the development of neural network-based systems for the autonomous identification and control of space platforms, there remain important unresolved issues associated with the reliable prediction of convergence speed and the avoidance of inordinately slow convergence. Focusing here on autonomous identification of lightly damped space structures, we first show that even apparently benign and simple examples can exhibit unpredictably slow convergence when the standard Least Mean-Square (LMS)-style identification algorithms are applied. To speed convergence of neural identifiers, we introduce the preprocessing of identifier inputs using Principal Component Analysis (PCA) algorithms. PCA is a procedure (that is realizable via a neural network) for the automatic generation of a transformation of the neural identifier's external inputs that makes the correlation matrix identity. When inputs are pre-processed in this way, enormous improvements in the convergence speed of the neural identifier is obtained. From a study of several such algorithms, we developed a new PCA approach which exhibits excellent convergence properties, insensitivity to noise and reliable accuracy. Numerical examples show many orders of magnitude reduction in the time required for convergence of system identifiers.

1. Introduction

Modern engineering technology is leading to increasingly complex payloads with ever more demanding performance criteria. However, currently advocated control design approaches often require a high fidelity dynamic model containing identified system parameters. The methodology which calls for an iterative process of finite element analysis and system identification is time consuming and computationally expensive to validate. Also, time-critical control recovery due to catastrophic failures is often left unresolved. The ultimate pursuit of a higher degree of autonomous behavior that provides constant health monitoring and fault tolerance for space systems with minimum human intervention has high priority in order to achieve a successful flight mission [1]. To increase the ability to accommodate anticipated as well as unexpected dynamic variations in the complex interrelated structures of space systems, the authors have been engaged in the development of autonomous neural control systems, based solely on on-board instrumentation, that are capable of self-optimization, on-line adaptation, and autonomous fault detection and controller reconfiguration.

The foundation of our investigations is a new, highly modularized processing architecture for neural algorithms in identification (ID) and control [2] devised some years ago. The basic capabilities of this architecture have been thoroughly demonstrated experimentally. For example, in the Adaptive Neural Control Program for the USAF Phillips Lab, autonomous control algorithms for vibration suppression were demonstrated on the ASTREX testbed [3,4]. These experiments showed, for the first time, not only completely autonomous convergence to a high performance controller but also automatic controller reconfiguration/recovery following the disengagement (simulated failure) of randomly selected subsets of the actuator hardware units.

The above state-of-the-art advances and controls experience directly suggest that the following basic issues must be addressed in order to achieve our ultimate aims in space system autonomous control. First there are issues relating to system identification (ID) or "replicator" modules. These modules are used either directly, e.g. in scene interpretation, image representation and machine vision or, most importantly, are fundamental building blocks of every adaptive controller subsystem. While there are reasonably complete results on convergence, our ability to reliably predict learning speed or convergence time is incomplete and needs to be perfected. Secondly, in addition to this predictive capability, we need further work in devising

algorithmic refinements to considerably speed up convergence.

In this paper, we consider techniques for faster convergence. The methods considered here involve the pre-processing of the adaptive replicator inputs so as to improve the structure of the confluence matrix (as is explained further below). In particular, we consider several variants of the Principal Component Analysis (PCA) algorithm. The basic results yield a general, broadly applicable, advance in learning/adaptation speed since we are improving a basic module that underlies all perceptual and control subsystems. We begin in Section 2 by presenting the basic identification algorithm and illustrating the convergence speed issue by numerical results on an apparently benign, three mode example. PCA algorithms are introduced in Section 3 where we develop a variant of the standard algorithm which is less sensitive to process noise and faster to converge on higher order principal components. The efficacy of this algorithm is illustrated by numerical results on both the three-mode example and on a 100-mode model of the UltraLITE large space optics test bed of the Air Force Research Lab in Albuquerque, New Mexico.

2. Basic System Identification Algorithm and Convergence Speed Issues

In order to more clearly motivate the subject of this paper, we first discuss the nature of convergence speed issues encountered with backpropagation-based neural networks or similar LMS-style adaptive signal processing schemes. Moreover the issues with which we shall be mainly concerned do not depend upon whether or not the control system or identifier is linear or nonlinear. Therefore, for simplicity and to focus ideas, we consider the problem of identifying a linear plant. The structure of the neural identifier depends upon the model form with which the plant is represented. To address the most popular model form, suppose the output, y(k), of a linear SISO system with input x(k), can be represented by the ARMA model;

$$y(k+1) = Wp^T X(k) \quad (1.a)$$

$$X(k) = [y(k), y(k-1), \ldots y(k-N+1), x(k), x(k-1), \ldots x(k-N+1)]^T \quad (1.b)$$

where $(.)^T$ denotes the transpose, N is the order , X is the 2N dimensional "regressor vector", assumed to be measured and Wp is the vector of ARMA coefficients. Now, given X, the simplest neural net for representing the above system, using the definitions in Ref. [2], consists of an input layer, accepting X(k) at each time k, with signals feeding into a single output neuron with synaptic connectors having a 2N dimensional weight vector, W(k). The output, z(k+1), is thus given by:

$$z(k+1) = W(k)^T X(k) \quad (2)$$

This is then subtracted from y(k+1) to obtain the output error:

$$e(k+1) = y(k+1)-z(k+1) \quad (3)$$

Supposing that our neurons are two-way devices as in [2], we feed e(k+1) back into the backward path of the output neuron. Following the rules in [2] for updating the weight of each synapse, we find that the change in W is proportional to the dyadic formed from the forward path input to W and the backward path input to W:

$$W(k+1) = W(k)+ \mu(k)e(k+1)X(k) \quad (4.a)$$

where:

$$\mu(k) = \alpha / \|X(k)\|^2 \quad (4.b)$$

Here, $\mu(k)$ is a time varying adaptive speed and α is a constant. The definition of μ is one of the distinguishing features of our work. It allows us to choose the constant α once and for all and, under quite general conditions, guarantee convergence. These matters are considered in some detail in [2]. For a general identification system, even a nonlinear system, [2] obtained this result: If there exist weight values such that z can duplicate y exactly and if the square of the norm of the error , considered as a function of the weights, is a homogeneous function of degree M, then $\alpha<M$ implies that e(k) converges to zero. In the above linear problem, the square of the error is obviously a quadratic function of the weights (M = 2). Hence, if y can be represented exactly by a system of form (2) for some value(s) of W, then $\alpha<2$ implies convergence of |e(k)|. Because it results in one- step convergence in the one-dimensional case, we most often use $\alpha = 1$.

Next, in order to say more about the rate of convergence, we need to recast (4.a) into a more suitable form. Define:

$$w(k) = Wp - W(k) \quad (5)$$

Then:

$$e(k+1) = w(k)^T X(k) \quad (6)$$

Using this expression in (4.a) and replacing W in favor of w, we get:

$$w(k+1) = [I - \mu(k)X(k)X(k)^T]w(k) \quad (7)$$

where I denotes the 2N dimensional identity. By the way, equation (7) is in no way peculiar to our particular neural net architecture - it is the basic "prototype" equation for studying the convergence behavior of virtually any adaptive system: controllers, identifiers, linear or nonlinear neural networks or LMS. Correspondingly, the convergence speed issues are common to all these systems as well.

Equation (7) describes the evolution of the deviation, w, of the weight vector from the values that permit z to exactly match y. The behavior of the mean of w or the norm of w gives some notion of the rapidity of convergence of (4). There are very few exact results,

particularly when α is of order unity (α not small). However, for small α, results such as those reported in [5] show that the expected value of w(k) asymptotically approaches wa(k), where:

$$wa(k+1) = [\, I - \alpha R]wa(k) \qquad (8)$$

where:

$$R = E[X(k)X((k)^T / \|X\|^2\,] \qquad (9)$$

We see that R (often called the "confluence matrix") is the second moment matrix of the normalized net input, X/ ||X||. The sum of the eigenvalues of R, trace(R), is unity. Thus the rates of geometric convergence have the form $(1 - \alpha\lambda)$, where λ is any one of the (non-negative) eigenvalues of R. We cannot make this factor small by increasing α indefinitely because, in general, $\alpha<2$ is necessary to ensure that W(k) is bounded. Thus, if R is ill-conditioned, i.e., has a set of small eigenvalues, then $|1-\alpha\lambda|$ will be nearly unity, and convergence can be very slow. Equations (8-9) pertain to the case $\alpha \ll 1$, but it is still arguable that (8-9) gives the leading term in a sequence of asymptotic approximations and adequately portrays the dominant factors in convergence speed even when α is of order unity.

Slowness of convergence is a problem that depends on a very complicated way on the system dynamic parameters, sample rate, the number of delays in the ARMA model, etc. The sensitivity of convergence speed to system parameters and the fact that slow convergence can even occur for relatively low order, apparently innocuous systems are well illustrated by the following "three mode example". This is a single-input, single-output system with just three lightly damped resonances - apparently a trivial example upon which to apply autonomous system identification. The discrete-time model is given by:

$$y(k+1)=[1, 0]x1(k+1)+[1, 0]x2(k+1)+ [1, 0]x3(k+1) \qquad (10)$$

where:

$$x1(k+1) = \rho_1 \begin{bmatrix} c1 & -s1 \\ s1 & c1 \end{bmatrix} x1(k) + \begin{bmatrix} 0 \\ 2.0 \end{bmatrix} \chi(k) \qquad \text{a. (11)}$$

$$x2(k+1) = \rho_2 \begin{bmatrix} c2 & -s2 \\ s2 & c2 \end{bmatrix} x2(k) + \begin{bmatrix} 0 \\ 2.0 \end{bmatrix} \chi(k) \qquad \text{b.}$$

$$x3(k+1) = \rho_3 \begin{bmatrix} c3 & -s3 \\ s3 & c3 \end{bmatrix} x3(k) + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \chi(k) \qquad \text{c.}$$

Here, ck and sk (k = 1,2,3) denote $\cos(\theta_k)$ and $\sin(\theta_k)$, respectively. θ_k and ρ_k are parameters of a balanced modal representation. The values of these parameters are given by:

$$\theta_1 = 2\pi/50\,,\ \theta_2 = 2\pi/30,\ \theta_3 = 2\pi/10 \qquad (12.a,b,c)$$

$$\rho_1 = 0.995\,,\ \rho_2 = 0.995\,,\ \rho_3 = 0.990 \qquad (13.a,b,c)$$

Finally, in (10) $\chi(k)$ is a discrete time white noise process that is normally distributed with unit variance.

Given the above system, we construct a simulation T steps long in which the regressor vector, X, is assembled and the system identification algorithm, (2) - (4) is implemented. Although rigorous requirements for convergence are satisfied, a typical run with N=6, α =1 and T=1000 results in very little decline in the output error. Figure 1 shows the system output and the output error, e(k), versus time steps for these values of parameters and for an initially zero weight vector. Apparently, e(k) persists throughout the entire 1000 steps without any evidence of further improvement. For the same case, Figure 2 shows the time variation of the elements of the weight vector. Again, there is no appreciable tendency toward convergence. The standard algorithm, (2)-(4) *does* converge but it would require a simulation over a much larger time period to reveal this numerically.

For the above simulation of length T = 1000, the eigenvalues of R are found to be:

$\lambda(R)$ = [2.910e-09 5.813e-06 3.373e-04 1.880e-03
1.812e-03 1.607e-03 2.539e-03 3.827e-03
4.189e-03 1.308e-02 2.311e-01 7.375e-01] (14)

In view of relations (8) and (9) and the eight orders of magnitude spread in the above eigenvalues, it is not surprising that convergence is glacially slow. Considering that this sort of phenomenon can occur for such a simple example it is vital to address the root cause of the problem in order to assure reliable operation of neural adaptive systems for applications of practical importance.

3. Pre-Processing of Identifier Inputs : Basic PCA Algorithm and Its Limitations

The above discussion hopefully provides convincing motivation for the present study. Furthermore, equations (8) and (9) suggest some ways out of our difficulty. The first possibility considered is to make the confluence matrix well conditioned by transforming the regressor vector before it is used by the neural identifier. In other words, replace X(k) by Y(k) where:

$$Y(k) = QX(k) \qquad (15)$$

Furthermore, we form the estimated system output as:

$$z(k+1) = W(k)^T Y(k) \qquad (16)$$

so that the output error is:

$$e(k+1) = y(k+1) - z(k+1) \qquad (17)$$

and W is updated according to:

$$W(k+1)=W(k)+\mu(k)e(k+1)Y(k) \qquad (18.a)$$

where:

$$\mu(k) = \alpha / \|Y(k)\|^2 \qquad (18.b)$$

In this case, in place of equation (7), we have:

$$w(k+1) = [I - \mu(k)Y(k)Y(k)^T]w(k) \qquad (19)$$

In view of the analysis underlying (8) and (9), the idea is to choose Q so that the correlation matrix of Y is well-conditioned. In fact, the ideal requirement is that:

$$E[Y(k))Y(k)^T] = I \qquad (20)$$

Moreover, we seek algorithms for determining Q to satisfy (20) that can operate in real time on current data, are adaptive and can be implemented in parallel computations. It turns out that these basic requirements are satisfied by the family of Principal Component Algorithms (PCA's). What is more, PCA's can be directly implemented as neural network algorithms. In the following, we discuss the background of PCAs, then briefly describe the innovative developments of the present study.

Principal Component Analysis is discussed by Preisendorfer [6] and Jolliffe [7]. PCA was first introduced by Pearson [8] and later developed independently by Hotelling [9] and Karhunen [10]. It was further generalized by Loeve [11]. The modern version, in the form of a feedforward neural network that extracts the desired number of eigenvalues ('principal components") is described by Sanger [12], who also gives a proof of convergence. This basic neural network version is termed the Generalized Hebbian Algorithm (GHA).

The most elementary component of GHA that extracts the first "principal component" of an N-dimensional vector time series, {X(k); k=1,2,...}, consists of an input layer of N neurons (which receive the elements of X at each time step) with synaptic connections that converge into a single hidden neuron, the output of which fans out to N output neurons. All neurons are linear and the weight vectors of both the "fan-in" and the "fan-out" sets of synapses are the same weight vector, call it w(k). Thus the net output is $(w^TX)X$. We form the error signal by subtracting this output from the original input, X. The next step is to use some algorithm for recursively adjusting w so that the norm of $X-(w^TX)X$ is minimized. In other words, we are trying to make the network optimally approximate (in a least mean square sense) the time varying vector, X(k), by a constant vector (the vector to which w(k) converges) multiplied by a time varying scalar. It turns out that when we succeed in doing this, w is the eigenvector of the input second moment matrix, $E[XX^T]$, corresponding to the largest eigenvalue. Moreover, the time averaged square of the hidden node output, $w^T X$, is the largest eigenvalue of $E[XX^T]$. The type of learning rule that is typically used is called "Hebbian" because the increment in w is formed solely by multiplying the net inputs and outputs and does not involve any explicitly backpropagated signals.

Next, it is obvious how to generalize the above network to construct as many principal components as we wish. Suppose we want the second largest eigenvalue and associated eigenvector of $E[XX^T]$. To the first network we add a second network of identical form but with weight vector w2 (and rename the w of the first network by w1). We form the output error of the second network by subtracting its output from $X-(w1^T X)w1$ (not from X). In other words, we require the second network to approximate what remains of X when its "principal component" has been subtracted off. With the correct learning rule, once the first net converges for w1, the second net converges so that the mean square value of $(w2^T X)$ approaches the second largest eigenvector of $E[XX^T]$ and w2 approaches the corresponding normalized eigenvector (and w2 becomes orthogonal to w1). Continuing in this way, we can build a cascaded sequence of networks that automatically extract as many eigenvalues and eigenvectors of $E[XX^T]$ as desired. In particular, the complete network can generate all the information needed to form the transformation Q in (15) such that the correlation matrix of Y is identity (as in (20)). Here we proceed to derive the GHA in the context of the neural architecture defined in Ref [2], Section 1. Using the two-way neurons and synaptic connectors defined in that paper, we find that the neural algorithm for determining the M largest principal components is as follows. Let:

$$W = [w1, w2, \ldots, wM]^T \qquad (21)$$

i.e., wk^T forms the kth row of W where wk is the kth "principal vector". Also, let:

$$V = WX \qquad (22)$$

Each element of V is seen to be the component of the input data, X, along one of the eigenvectors. Finally, let LT[.] denote the lower triangular matrix (including the diagonal) formed from the matrix argument [.]. Then the Generalized Hebbian Algorithm (generalized because it constructs an arbitrary number of principal components) is:

$$W(k+1) = W(k) + \mu(k)\{V(k)X(k)^T - LT[V(k)V^T(k)]W(k)\} \qquad \text{a. (23)}$$

$$\mu(k) = \beta / X(k)^T X(k) \qquad \text{b.}$$

Again, following ref.[2], Section 1, in order to assure reliably bounded behavior, we choose the adaptive speed, μ, so that $\mu(k) = \beta/(w^T X)^2$ where we use β (instead of α) to denote the learning rate constant to keep this distinct from the learning rate used in the system identification algorithm. In the present study, we typically take $\beta \leq 1$. (23) differs from usually considered PCA algorithms in that μ is time-

varying according to (23.b) rather than being a constant or some explicit , and decreasing function of time, such as 1/k. However, if X is normalized μ is very nearly constant anyway and there little difference produced in the final result. The mean square values of the elements of V are readily seen to be the eigenvalues of $E[X(k)X(k)^T]$. As part of our on-line algorithm we form estimates of the mean square values of the elements of V by means of time averaging. Denoting these estimates by the vector λ, we compute λ_n from:

$$\lambda_n(k+1) = \lambda_n(k) - (1/Tav)(\lambda_n(k) - V_n^2) \qquad (24)$$

where Tav denotes the averaging time used to compute the mean square value. Once the algorithm converges, the desired transformation, Q, that allows (20) to be satisfied is:

$$Q = [\mathrm{diag}(\lambda^{1/2})]^{-1} W \qquad (25)$$

It is easily seen that after convergence, the correlation matrix of Y = QX is:

$$E[YY^T] = [\mathrm{diag}(\lambda^{1/2})]^{-1} W R W^T [\mathrm{diag}(\lambda^{1/2})]^{-1}$$

But since $WR = [\mathrm{diag}(\lambda^{1/2})]^{-1} W$ and $WW^T = I$, the above reduces to the MxM identity.

To illustrate how GHA works, we first simulate the three-mode example for T=1000 to generate X(k), k+1,...,T. With this data and the parameter values: N = 6, T = 1000, β = 0.01, Tav = 100, M=6; we run the algorithm (23)-(25). One can monitor convergence behavior by plotting the separate rows of W versus time. Figure 3 shows the time variation of the elements of the first three rows of W, i.e. the first three Principal Component vectors. Clearly, the role of β is a reciprocal averaging time. Larger β means quicker initial convergence but larger steady state fluctuating or noisy error. Smaller β slows convergence but reduces the steady state noise coming through to W. The various rows of W are seen to converge sequentially: the first row converges first, the second row second, etc. The first few rowsconverge rather quickly and the mutual orthonormality conditions take hold after the firstseveral hundred time steps. The higher order eigenvectors take much longer to converge and seem to be impeded by residual errors still remaining from the lower order eigenvectors.

As one explores the behavior of PCA trying various values of parameters, one observes several unsatisfactory aspects. First, the convergence of the higher order principal component vectors (the lower rows of W) appears to be inordinately slow. Secondly, a large fraction of the plant disturbance noise is transmitted through the algorithm to appear as a steady state fluctuation error in W. Small values of beta partially remedy this but at the cost of slow convergence. A third difficulty is that although the algorithm eventually makes the principal vectors mutually orthogonal, the higher order principal vectors still retain small components along the lower order vectors and this seems to impede the convergence of the higher order vectors. The next Section addresses the remedy of these difficulties.

4. PCA Refinements: Variable Speeds, Time Averaging and Weight Orthogonalization

Here we consider several modifications to the GHA presented earlier designed to address the difficulties noticed in the last Section.

The first difficulty noted in Section 3 was that the convergence of the higher order principal vectors (the lower rows of W) appears to be too slow. An obvious source of this difficulty is that while GHA is really an ordered sequence or cascade of single-component algorithms, the adaptive speed, $\mu(k)$, is the same for all stages of the cascade. In other words, we have not taken advantage of the cascaded structure of the algorithm to define larger adaptive speeds for the higher order principal vectors. This inefficiency accounts for at least a part of the slowness of convergence of GHA.

First, let us explore the cascaded structure of GHA in more detail. Using the notation in the last Section, we may write out the algorithm in an expanded form as:

$$w1(k+1)=w1(k)+\mu(k)[X(k)-(w1^T X)w1](w1^T X) \quad \text{a. (26)}$$

$$w2(k+1)=w2(k)+\mu(k)[X(k)-(w1^T X)w1 -(w2^T X)w2](w2^T X) \quad \text{b.}$$

:

etc.

Clearly the algorithm is sequential in that the results for w1,...,wk-1 form the basic inputs to the computation of wk. We can modify the above relations slightly to make this aspect even more explicit. Recall that the GHA, during convergence, tends to enforce the mutual orthogonality of the wk's. Thus in (26.b) very little is changed if, in the inner product $(w2^T X)$, we replace X by $X1=X-(w1^T X)w1$. In other words, in the second principal vector computation, the input is not X but rather the residual error found by subtracting the first principal component from X. Likewise, we could modify the third principal vector computation by replacing X by the residual signal obtained after subtracting off the first two principal components. Proceeding in this way, we get the following sequence of relations:

$$w1(k+1) = w1(k)+\mu(k)[X(k)-(w1^T X)w1](w1^T X) \quad \text{a.}$$

$$X1 = X - (w1^T X)w1 \quad \text{b.}$$

$$w2(k+1)=w2(k)+\mu(k)[X1(k)-(w2^T X1)w2](w2^T X1) \quad c.$$
$$X2 = X1 - (w2^T X1)w2 \quad d.$$
$$w3(k+1)=w3(k)+\mu(k)[X2(k)-(w3^T X2)w3](w3^T X2) \quad e.$$
$$X3 = X2 - (w3^T X2)w3 \quad f.$$
(27)

:

etc.

In this form, the algorithm for each principal vector is basically an independent computation. Further, we can consider the convergence of each component based only on its local output error and its immediate input. Note that in the above μ is the same for each component, i.e.: $\mu = \beta / \|X\|^2$. However, with the algorithm in the form of (27), the boundedness and convergence results of Ref.[2], Section 1 would require that, for the kth principal vector, μ is bounded by two divided by the square of the norm of the local input, Xk-1. Obviously, if there is any convergence at all for components 1 through k-1, then $Xk\text{-}1^T$ Xk-1 is smaller than $X^T X$. Thus, for the kth principal vector computation, we can actually use a μ which is much larger than what was previously defined. With this reasoning, we replace (27) above by the relations:

$$\mu_0(k) = \beta / X^T X \quad a.$$
$$w1(k+1) = w1(k)+\mu_0(k)[X(k)-(w1^T X)w1](w1^T X) \quad b.$$
$$X1 = X - (w1^T X)w1 \quad c.$$
$$\mu_1(k) = \beta / X1^T X1 \quad d.$$
$$w2(k+1)=w2(k)+\mu_1(k)[X1(k)-(w2^T X1)w2]((w2^T X1) \quad e.$$
$$X2 = X1 - (w2^T X1)w2 \quad f.$$
$$\mu_2(k) = \beta / X2^T X2 \quad g.$$
$$w3(k+1)=w3(k)+\mu_2(k)[X2(k)-(w3^T X2)w3](w3^T X2) \quad h.$$
$$X3 = X2 - (w3^T X2)w3 \quad i.$$
(28)

:

etc.

A second difficulty noted in connection with the basic GHA in the last section was that a large fraction of the plant disturbance noise is transmitted through the algorithm to appear as a steady state fluctuation error in W. A straightforward way to address this is to pass the wk's as defined by (27) through a low-pass filter. In other words, if Fn denotes the expression given in (27) for wn(k+1)-wn(k), then low-pass filter Fk then use the output as wn(k). This device clearly reduces the presence of noise in the principal vectors. Also, it has the effect of time averaging second-order products of the input, thereby strengthening the importance of the regressor correlation matrix, $R = E[XX^T]$. With this modification, the first few stages of the algorithm are as follows:

$$\mu_0(k) = \beta / X^T X \quad a.$$
$$F1 = \mu_0(k)[X(k) - (w1^T X)w1] (w1^T X) \quad b.$$
$$w1av(k+1) = w1av(k) + (F1-w1av(k))/Tav \quad c.$$
$$w1(k+1) = w1(k) + w1av(k) \quad e.$$
$$X1 = X-(w1^T X) w1 \quad f.$$
$$\mu_1(k) = \beta / X1^T X1 \quad g.$$
$$F2 = \mu_1(k)[X1(k) - (w2^T X1)w2](w2^T X1) \quad h.$$
$$w2av(k+1) = w2av(k) + (F2-w2av(k))/Tav \quad i.$$
$$w2(k+1) = w2(k) + w2av(k) \quad j.$$
$$X2 = X1 - (w2^T X1)w2 \quad k.$$

:

etc.

(29)

where Tav denotes the averaging time selected by the user. A third difficulty associated with the basic algorithm is that in the course of convergence, the higher order principal vectors become corrupted with small components along the lower order principal vectors. This happens because of progressively accumulating errors and despite the fact that the algorithm eventually tends to make the vectors mutually orthogonal. Moreover, such errors tend to become more and more serious for the higher order vectors. These errors appear to significantly impede the convergence of the high order principal vectors. A simple remedy for this is to subtract off from each iterate for wn the components along all the lower order vectors. Specifically, in addition to equations of the form (28) for each vector, wn, and just before wn(k) is updated to wn(k+1), we have the sequence of replacement relations:

$$\begin{aligned} wn(k) &\leftarrow wn(k) - (w1(k)^T wn(k)) w1(k) \\ wn(k) &\leftarrow wn(k) - (w2(k)^T wn(k)) w2(k) \\ &\vdots \\ wn(k) &\leftarrow wn(k) - (wn\text{-}1(k)^T wn(k))wn\text{-}1(k) \end{aligned} \quad (30)$$

The modified GHA incorporates all of the refinements described above, specifically including relations (29) and (30). To provide a numerical example, it is convenient to run the three-mode example to generate X. β=0.01 is a reasonable value for the learning rate constant and it is advisable to select an averaging time that is consistent with beta, say Tav=1/β or less. If one examines the behavior of the algorithm for values of parameters comparable to the foregoing, it becomes evident that with the new values of the adaptive speeds, convergence is more rapid for the higher order vectors. Also, the principal vector results settle down more quickly and are far less noisy. Finally, the forcible orthogonalization of the vectors via relations of the form (30) noticeably helps convergence, particularly of the mid-order principal vectors.

Despite the above noted positive results, sufficient experimentation with the modified GHA can reveal some disadvantages to both (30) and to the strict use of modified inputs such as in (28.c, f,and i). What can happen in the case of (30) is that due to accumulated

errors, the repeated subtraction from a high order vector of components along lower order vectors results in a nearly zero vector, at which point the convergence process for the vector in question as well as for all higher vectors collapses. In addition, and in analogous fashion, the precise subtraction from the input at stage n of components along all the lower order principal vectors can result in some Xn(k) being nearly zero. This makes the convergence process break down. It is especially necessary to "back off" on requirements (30).

Before concluding this section, we address the above problem and describe how the appropriately modified algorithm can be expressed in a completely general form, capable of extracting an arbitrary number of principal components. First, we do wish to subtract off components of the input to a vector of given order along vectors of lower order and we still wish to enforce the orthogonality of successive vectors. The basic motives for doing these things remain valid. However, we do not need to carry out these steps precisely; it suffices to impose these conditions to first order accuracy only. Thus in place of the sequence (28.c), (28.f), (28.i),etc.,we would specify the regressor inputs to the several principal component stages as :

$$X1 = X - (w1^T X)w1 \quad \text{a (31)}$$

$$X2 = X - (w1^T X)w1 - (w2^T X)w2 \quad \text{b.}$$

$$X3 = X - (w1^T X)w1 - (w2^T X)w2 - (w3^T X)w3 \quad \text{c.}$$

:

etc.

Likewise, in place of the restrictive relations (30), we impose only the first order versions:

$$w2(k) \leftarrow w2(k) - (w1^T w2)w1(k) \quad \text{a.(32)}$$

$$w3(k) \leftarrow w3(k) - (w1^T w3)w1(k) - (w2^T w3)w2(k) \quad \text{b.}$$

:

etc.

With the above changes but retaining the time averaging operation and the basic update equations for mun and Fn , n=1...M, the overall algorithm can be expressed in the general form:

$$W \leftarrow W - C_{ww} W \quad , (W \text{ is MxN}) \quad (33)$$

$$C_{ww} = [\ (w1^T w2) \quad 0 \quad 0 \quad \dots\ ;$$
$$(w1^T w3)\ (w2^T w3) \quad 0 \quad \dots\ ;$$
$$(w1^T w4)\ (w2^T w4)\ (w2^T w4) \quad \dots\ ;$$
$$\vdots \quad \vdots \quad \vdots \quad] \quad (34)$$

$$XI^T \leftarrow [X\ X\ \dots\ X] - W^T C_{wx}^T \quad (35)$$

$$C_{wx} = [\ (w1^T X) \quad 0 \quad 0 \quad \dots\ ;$$
$$(w1^T X)\ (w2^T X) \quad 0 \quad \dots\ ;$$
$$(w1^T X)\ (w2^T X)\ (w3^T X) \quad \dots\ ;$$
$$\vdots \quad \vdots \quad \vdots \quad] \quad (36)$$

$$Z = \text{diag}\{\ w1^T X, w2^T X, \dots, wM^T X\} \quad (37)$$

$$\aleph \leftarrow \beta\ (\text{diag}(XIXI^T) + \varepsilon)^{-1} \quad (38)$$

$$F \leftarrow \aleph(Z(XI) - (Z^2)W) \quad (39)$$

$$Fav(k+1) = Fav(k) + (1/Tav)*[F(k) - Fav(k)] \quad (40)$$

$$W(k+1) = W(k) + Fav(k) \quad (41)$$

All of the above are embodied in the new refined version of GHA. As part of this algorithm, we also calculate the output variable "Cor". This is the correlation matrix of Y over the whole record length. Clearly, the resemblance of Cor to the identity indicates the closeness of the algorithm to convergence. Some experimentation with the refined algorithm shows that it works quite reliably and performs up to the limitations inherent in PCA algorithms.

In the next section, we will examine the convergence behavior of PCA as embodied in equations (33) - (41), using numerical results from the three mode example discussed above. We conclude by summarizing both the useful features and the limitations of the PCA algorithms studied herein.

5. Performance of the Refined PCA Algorithm: Numerical Results

An issue that should be explored further is whether or not the neural ID and PCA algorithms should be run sequentially or concurrently. It is easy to combine the various algorithms given earlier to investigate the behavior of a combined algorithm; i.e. the Q transformation matrix is being updated every time step while the neural identification is proceeding. Results from such an experiment should quickly deter one from the possibility of concurrent operation. We have found that changing the basis of the ID weights, which is what one is doing when Q is updated each time step, greatly disrupts their convergence. On the three mode example we have been unable to obtain convergence under concurrent operation. For now the only tenable conclusion seems to be that one should keep Q constant in the ID algorithm, except for step-wise changes, and that one should run PCA for substantial blocks of time (long enough to get reasonably accurate statistics) before updating (in step-wise manner) the transformation Q.

Adopting the above policy it appears that the final algorithm of Section 4 (equations (33)-(41)) can work quite well and the difficult three mode example, for which R exhibits an eight order-of-magnitude spread in its eigenvalues, can be treated successfully. This can be observed by applying the refined GHA to the same data sequence {X(k): k = 1,...,1000} generated for the three mode example in the case of Figure 3. In other words, we suppose that for the first 1000 time steps, the basic identification algorithm is run with Q = I. The data generated for X during the first 1000 steps is used by the refined GHA to obtain a better estimate for the transformation Q. At some time subsequent to

this, we reset Q equal to the value generated by the refined GHA and continue running the identifier with the new (constant) value of Q.

Figures 4, 5 and 6 show the time variation of the first three Principal Components to illustrate the operation of the refined GHA. We used the same parameter values as in Fig. 3 but extracted all 12 Principal Components. Initially, W was set to the identity matrix. It can be seen from these results that high frequency fluctuations due to the training signal stimulus are absent. Moreover the higher component vectors converge considerably more quickly.

As the second step of the process, we now use the Q generated by the refined GHA as described above and run the revised identification algorithm, (15)-(18), holding Q constant. Figure 7 shows the results in terms of the system response and the identifier error. This may be compared directly with Figure 1. It can be seen that the modified ID algorithm substantially converged within 200 time steps. This is a vast improvement over the initial results. Also, it should be noted good ID convergence behavior was obtained even though the PCA convergence was far from complete at the end of 1000 steps. In fact the correlation matrix of the transformed regressor still had an almost order-of-magnitude spread in the eigenvalues. This emphasizes the fact that only approximate convergence in PCA is needed to produce profound improvement in the identification algorithm performance.

So far in the above, we have investigated the effectiveness of transforming the regressor via PCA on the apparently simple but actually challenging "three mode model". Now it is time to see how well the more effective of the algorithms so far evaluated work on a problem of more representative complexity; namely one based on the hundred mode UltraLITE model.

The original data on UltraLITE provided to this study was in the form of the standard [A,B,C,D] matrices for a continuous-time model. Specifically, the 200x200 A matrix was in 2x2 block diagonal modal form, the C matrix pertains to the position sensors and the B matrix corresponds to the co-located piezo actuators. This continuous-time model was then transformed into a discrete time model assuming a 200Hz sample rate. To set up a suitable identification problem, we assume that the system represented by the hundred-mode model is stimulated with discrete-time white noise injected through actuator number 1. It is also assumed that we monitor response through the (co-located) position sensor number 1.

Now if, under the above assumptions, we look at system responses to the white noise disturbance, we see that response is largely made up of the oscillation of the first mode having frequency near 11 Hz. Beyond the first mode, there is a sizable frequency gap before the next modes start at approximately 35 Hz. There is also a large group of "substructure" or "component" modes from approximately 90 Hz to approximately 165 Hz, with some ninety modes in this band. On the whole, however, the main "global" modes, i.e. modes in which the motion embraces all parts of the structure, not just localized components, appear to be the first ten modes. These extend in frequency up to approximately 90 Hz. Consequently, we consider a model consisting of the first ten modes in the following.

When we apply the standard algorithm (without transformation of the regressor) to the identification of the above system and inspect the output signal and identification output error, we see that there is reasonable convergence for the first mode response (which has a level of 1.5 to 2.0) over the first 200 time steps but that there is also a persistent, higher frequency error (with magnitude of a few tenths). Clearly, there are slow modes of convergence in this case. Further, when we compute the correlation matrix of the regressor, R, using sample time averages, we see that the eigenvalues of R range from 4.6e-19 to 0.13. This means that R is even more poorly conditioned in this case than in the three-mode example. Little wonder that the standard algorithm apparently fails to converge for this UltraLITE example.

Next, we run the refined GHA with the parameter values $\beta = 0.01$, $T = 500$, $Tav = 100$ and $N = 40$. We monitor convergence by looking at the off-diagonal elements of Cor, the correlation matrix of the transformed regressor, Y. If all such elements are of magnitude less than 0.1, we deem the algorithm to have converged to an acceptable extent. Under these conditions, the refined GHA extracted all 40 principal components in approximately 4.5 minutes. This is not an inconveniently long time considering that this initial component extraction only has to be undertaken once initially and subsequent system changes can be tracked relatively quickly.

For purposes of comparison, Figure 3.1.4-8 shows a thousand time step simulation of the ten mode UltraLITE model with no transformation of the regressor. Although the identifier error settles down to a magnitude approximately an order-of-magnitude less than the system output, there is no evidence of further significant improvement. In contrast, Figure 3.1.4-9 shows the result of the use of the regressor transformation, Q, computed as described above. In this case, judging from the identification error, the identification algorithm converged in approximately 600 to 700 steps. In summary, then, the performance of PCA on the ten mode UltraLITE example is quite

analogous to the results obtained on the three mode model.

6. Concluding Remarks

Our work has concentrated on the development of effective learning algorithms for PCA networks and the use of such networks for accelerated convergence of neural network or LMS series-parallel system identifiers via the appropriate transformation of the identifier inputs (symbolized in the text by the transformation matrix Q). After studying several major variations and extensions of previously considered algorithms, we have devised a new algorithm that converges rapidly, is insensitive to disturbance or training input noise and can preserve accuracy tolerances for the higher order principal components. With mild constraints on the learning rate parameter, the weight vectors converge to the normalized (and mutually orthogonal) eigenvectors of the regressor vector corelation matrix. When this algorithm is applied to generating the Q transformation for the "three-mode example" described in the text, the system identifier converges to high accuracy within 200 time steps. Similar results are obtained on the much larger model of the dynamics of the UltraLITE test bed. Besides the many orders-of-magnitude improvement in convergence speed, the convergence time required is much more predictable. Specifically, with the use of transformation Q generated by a PCA network, one can make a reasonably accurate estimate of the convergence time given a bound on the system order and the neural network parameters. Finally, the approach introduced here is, within the neural architecture of [2], immediately applicable to nonlinear networks for system ID.

In closing, it must be noted that PCA, notwithstanding the above advantages, still needs to operate over a relatively longer term time scale than the ID algorithms. Basically, we recommend that one should typically use a PCA module which operates long-term in the background, constantly preparing the ID module for fast reaction. In general, the context within which one uses PCA assumes that input/output statistics are roughly persistant; i.e. significant order-of-magnitude changes are long term. This can be true, at least in an approximate sense, even for some step changes in system parameters. However, this is not true for sudden changes in certain types of parameters - e.g. the dimension of the input space changes abruptly (an actuator fails). To handle such cases, we must examine alternative algorithms.

Finally, we must mention that, beyond improving the convergence speed of identification, PCA has other, very important uses. PCA is a natural way to assess the needed order of the model. Using PCA, one can always truncate components associated with eigenvalues that are below some threshold of statistical significance. This is an essential capability if one is to build autonomous intelligent systems that are resistant to noise. Secondly and just as importantly, PCA gives a tool for automatically controlling network complexity. Analogous to Hankel norm approximation and similar "balancing" concepts the Principal Components give a criterion for the truncation of input spaces and the pruning of network weights. These features are sufficient reason for desiring to include some form of PCA algorithm as a standard module within any automatic system ID or intelligent control system.

References

1. G.G. Yen, "Autonomous neural control in flexible space structures," *Control Engineering Practice,* 3(4), April 1995, pp. 471-483.

2. D. C. Hyland, "Connectionist Algorithms for Identification and Control: System Structure and Convergence Analysis", Paper No. AIAA 97-0686, 35th Aerospace Sciences Meeting, Reno, NV, January, 1997.

3. Hyland, D.C., Davis, L.D., Das, A., and Yen, G. "Autonomous Neural Control for Structure Vibration Suppression", AIAA -96-3923, AIAA Guidance, Navigation and Control Conference, San Diego, CA, July 1996.

4. Hyland, D.C. and Davis,L.D., "Adaptive Neural Control Program - Final Report", A F Phillips Lab. contractor report, February, 1997.

5. R.R. Bitmead, "Persistence of Excitation Conditions and the Convergence of Adaptive Schemes", *IEEE Trans. On Infor. Theory,* vol. IT-30, No. 2, March 1984.

6. Preisendorfer, R. W., Principal Component Analysis in Meteorology and Oceanography. New York: Elsevier. 1988.

7. Jolliffe, I. T., Principal Component Analysis. New York: Springer-Verlag. 1986.

8. Pearson, K., "on lines and planes of closest fit to systems of points in space." Philosophical Magazine 2, pp.559-572. 1901.

9. Hotelling, H., "Analysis of a complex of statistical variables into principal components." Journal of Educational Psychology 24, pp.417-441, 498-520. 1933.

10. Karhunen, K., "Uber lineare methoden in der Wahrscheinlichkeitsrechnung." Annales Academiae Scientiarum Fennicae, Series A1: Mathematica-Physica 37, pp. 3-79 (Transl.: RAND Corp., Santa Monica, CA, Rep. T-131, 1960).

11. Loeve. M., Probability Theory, 3rd ed. New York: Van Nostrand. 1963.

12. Sanger, T. D. "Optimal unsupervised learning in a single-layer linear feedforward neural network." Neural Networks 12, pp. 459-473. 1989.

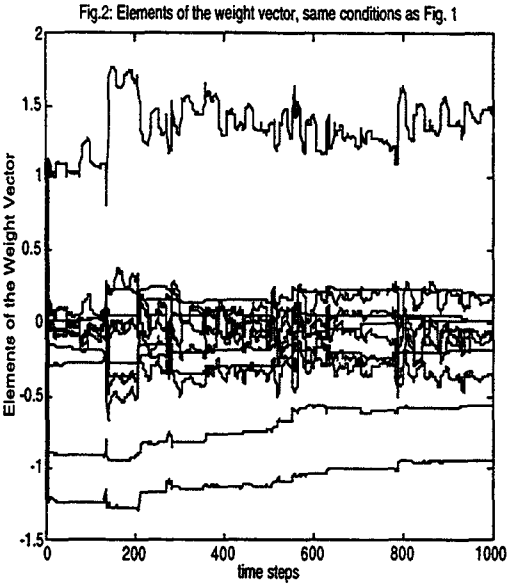

Figure 2: Elements of the weight vector of the identifier versus time, same conditions as Fig.1.

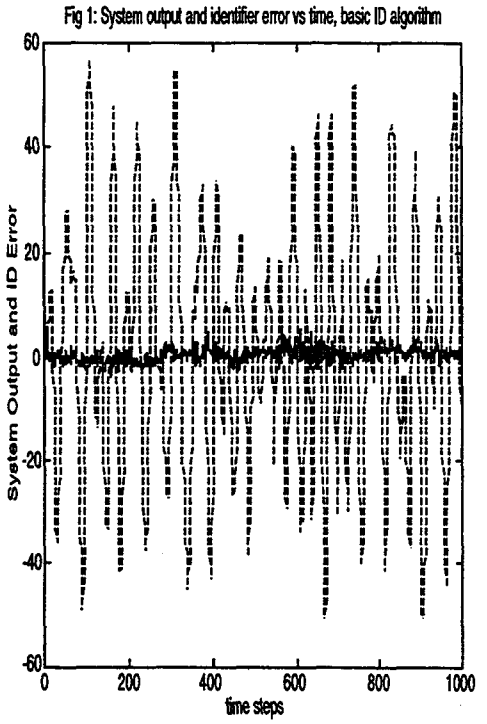

Figure 1: System output, y(k), and identifier output error, e(k) versus time for the three mode example, basic identification algorithm with N = 6, and alpha = 1. The initial weight vector is zero.

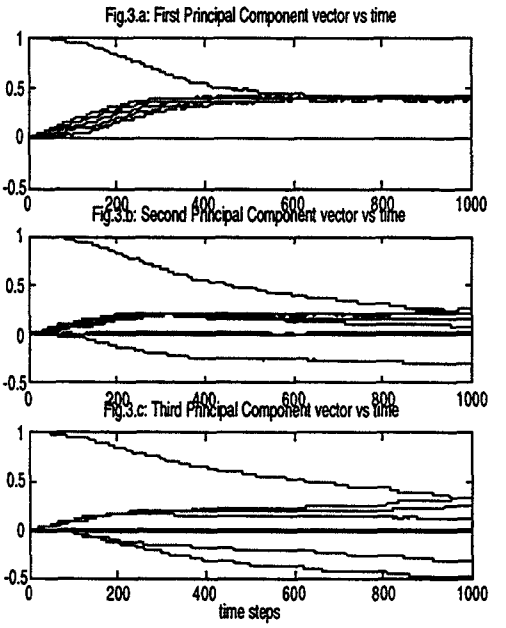

Figure 3: The first three Principal component vectors versus time, using the GHA with N=6, beta-0.01, Tav-100, M=6. Part a: First Principal Component; Part b: Second Principal Component; Part c: Third Principal Component.

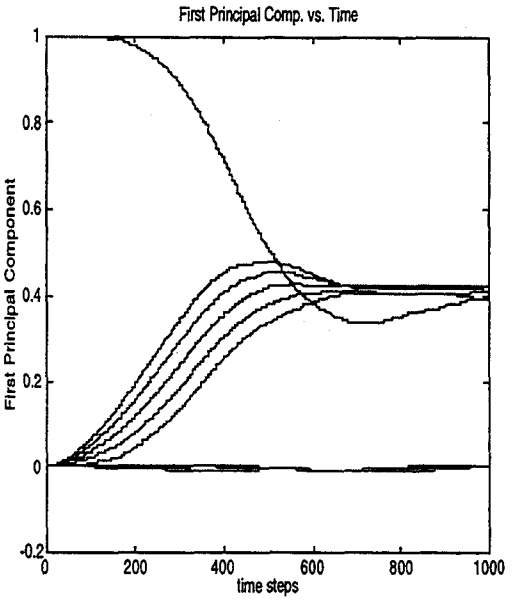

Figure 4: Results of the refined GHA with the same X(k) as in Fig. 3 and beta = 0.01, Tav = 100. This shows the time variation of the elements of the first Principal Component vector.

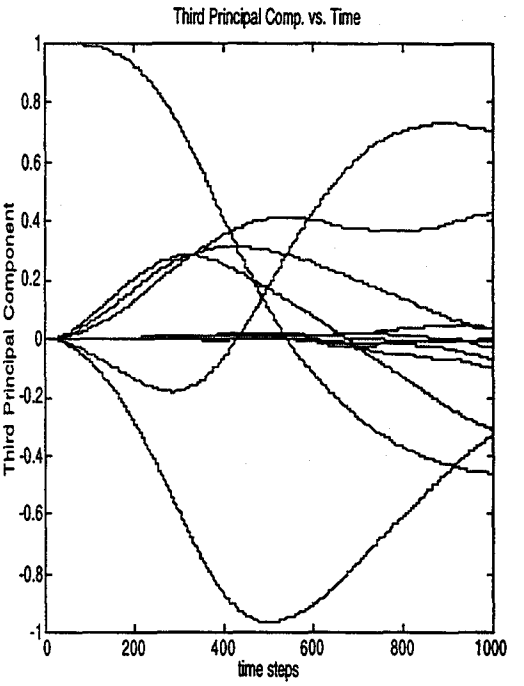

Figure 6: Time variation of the elements of the third Principal Component vector obtained via the refined GHA, same conditions as in Fig 4.

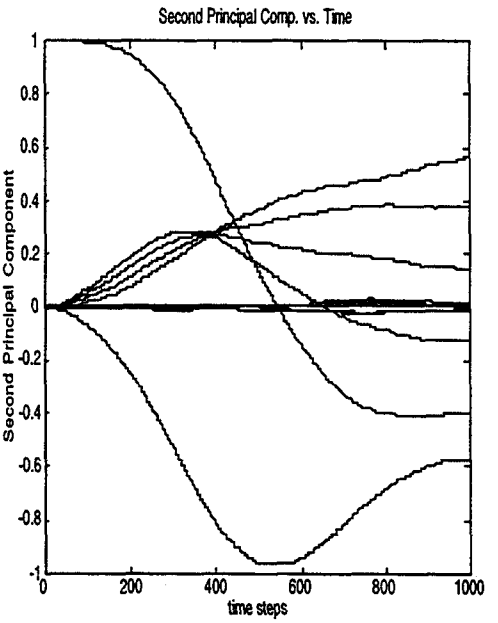

Figure 5: Time variation of the elements of the second Principal Component vector obtained via the rfined GHA, same conditions as in Fig. 4.

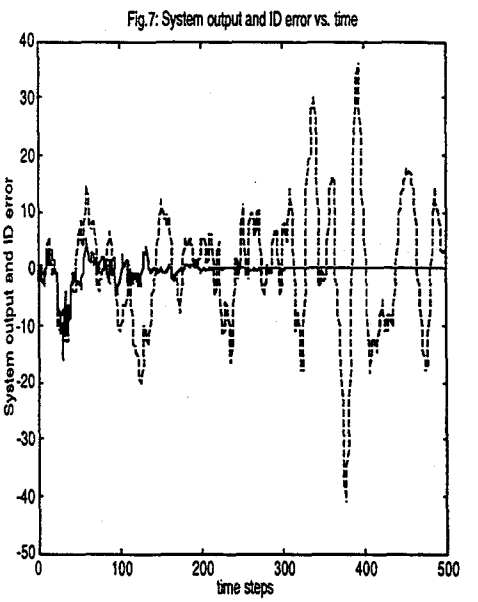

Figure 7: System output and identification error - under the same conditions as Fig.1 except that the modified ID algorithm is used with Q determined by the refined GHA over 1000 time steps.

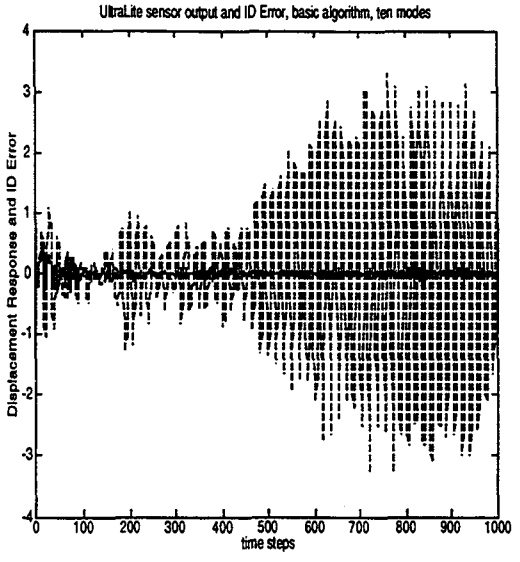

Figure 8: System output (dashed line), y(k), and identifier output error (solid line), e(k) versus time for the UltraLITE model with 10 modes, T = 1000, basic identification algorithm.

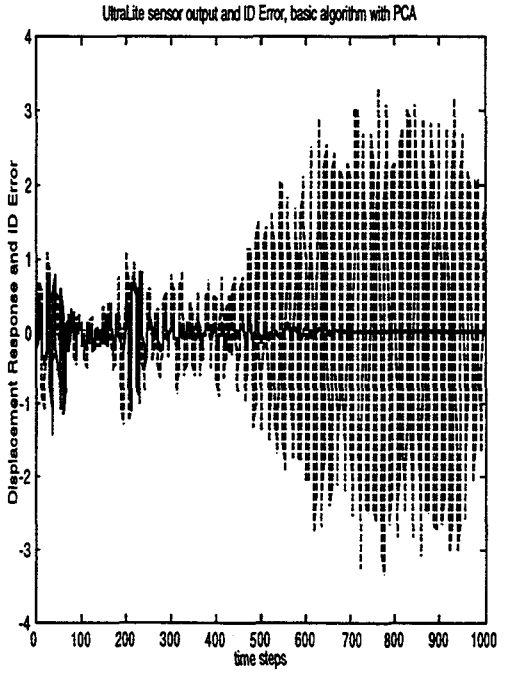

Figure 9: System output (dashed line), y(k), and identifier output error (solid line), e(k) versus time for the UltraLITE model with 10 modes, T = 1000. Here, the PCA-generated transformation was used on the regressor. The training stimulus time series is exactly the same as that used in Figure 8.