A New Family of Concurrent Algorithms for Adaptive Volterra and Linear Filters

Ajit Kumar Chaturvedi and Govind Sharma

***Abstract*—A novel idea for introducing concurrency in LS adaptive algorithms by sacrificing optimality has been proposed. The resultant class of algorithms provides schemes to fill the wide gap in the convergence rates of LS and SG algorithms. It will be particularly useful in the real-time implementations of large-order linear and Volterra filters for which both the LS and SG algorithms are unsuited.**

***Index Terms*— Adaptive filters, convergence rate, least mean square methods, least squares methods, parallel algorithms.**

I. Introduction

The emergence of numerically stable fast least squares (FLS) algorithms [1], [2] has been a major development in the design of adaptive algorithms. Thus, we now have algorithms that can provide least squares (LS) solution for a computational cost that is of the same order as stochastic gradient (SG) algorithms and will lie in the range of $7N$ to $10N$, where N is the filter length. Fast algorithms seek efficient ways of solving the normal equation [3] by exploiting various redundancies. However, it appears that this approach has almost reached a plateau, and we have to look in other directions for further developments. To begin with, let us list some of the reasons that necessitate further progress in the design of adaptive algorithms. When the order of the filter is few hundreds or thousands, such as for echo cancellation in audio conferencing, implementation of even a FLS algorithm with $7N$ complexity may be infeasible in real-time applications with today's technology. Otherwise, in multichannel applications for, e.g., Volterra filters, the complexity of FLS algorithms is higher compared with SG algorithms [4]. Furthermore, there is a wide gap in the rate of convergence of LS and SG algorithms. If algorithms requiring lower computational time per iteration than FLS algorithms can be designed even if with a lower rate of convergence, then they will be particularly suited for applications where the fast convergence of LS is either not required or not possible to achieve in real-time, and, at the same time, SG algorithms are not suited because of their slow rate of convergence.

There are two possible approaches to address the above issues. One is input signal modeling, and the other is suboptimal LS approaches. A useful method relying on the first approach has been proposed in [5] and [6]. However, it is useful only when the input signal satisfies the assumed model, which is autoregressive, and the filter order is much larger than the autoregressive order of the input signal. In this correspondence, we present an alternate method that relies on the later approach. We consider LS algorithms and introduce suboptimality such that it brings about concurrency in a natural way. As a result, the rate of convergence will be less compared with LS, but how much less will depend on the degree of suboptimality chosen by the user while there will be a corresponding saving in the computational time required per iteration by exploiting concurrency. Besides, if the desired signal power is low enough, there will be no loss in the rate of convergence compared with LS. The proposed method also has the potential of using the *a priori* information of the input signal to achieve LS or near LS performance without attempting an exact solution of the normal equation. As an interesting byproduct, it leads to a generic framework for LS and SG algorithms. The approach of the proposed algorithm is the same for linear and Volterra filters, and hence, we do not treat them as separate cases. The input data and weights of the adaptive filter are clubbed into $N \times 1$ vectors that are denoted $X(n)$ and $\hat{W}(n)$, respectively. The filter output and desired signal are represented as $y(n)$ and $d(n)$, respectively.

Manuscript received February 28, 1995; revised January 25, 1999. The associate editor coordinating the review of this paper and approving it for publication was Dr. Pierre Duhamel.

The authors are with the Department of Electrical Engineering, Indian Institute of Technology, Kanpur, India (e-mail: akc@iitk.ac.in).

Publisher Item Identifier S 1053-587X(99)06751-3.

II. The Parallel Recursive Least Squares Algorithm

Step 1: We refer to the filter under consideration as super filter and decompose it into q subfilters by partitioning $X(n)$ and $\hat{W}(n)$ into q vectors each as

$$X^T(n) = [X_1(n),\, X_2(n),\, \cdots,\, X_i(n),\, \cdots,\, X_q(n)]^T \quad (1)$$

$$\hat{W}^T(n) = [\hat{W}_1(n),\, \hat{W}_2(n),\, \cdots,\, \hat{W}_i(n),\, \cdots,\, \hat{W}_q(n)]^T \quad (2)$$

$X_i(n)$ and $\hat{W}_i(n)$ are the data vector and weight vector of the ith subfilter, respectively. The dimension of both $X_i(n)$ and $\hat{W}_i(n)$ is $N_i \times 1$ such that $\sum_{i=1}^{q} N_i = N$.

Step 2: The gains of the subfilters, which are denoted $k_i(n)$ for the ith subfilter, are computed independently and concurrently using their respective data vectors and correlation matrices $R_i(n)$'s and applying any LS algorithm (including the numerically stable fast versions).

Step 3: Without any loss of generality, the system from which we get $d(n)$ can also be thought of as made up of q subsystems, which can be modeled as filters of the same orders as the subfilters of the super filter. Thus

$$d(n) = \sum_{i=1}^{q} d_i(n) \quad (3)$$

where $d_i(n)$ represents the component of $d(n)$ that is contributed by the ith subsystem. If we know the $d_i(n)$'s, we can easily implement the super filter as q subfilters, each operating in parallel and independent of each other. In adaptive algorithms, the use of $d(n)$ lies in computing the error signal $e(n)$. Hence, if, instead of $d(n)$, we are given $e(n)$, it will serve our purpose equally well. In a practical situation, we do not know $d_i(n)$ or $e_i(n)$. Hence, we propose the following. Compute $y(n)$ by summing the outputs of the subfilters. Subtract $y(n)$ from $d(n)$ to get $e(n)$, and update all the subfilters using $e(n)$. In other words, assume $e_i(n) = e(n)$ for all $1 \le i \le q$.

We refer to this algorithm as the parallel recursive least squares (PRLS) algorithm because, except for the computation of $e(n)$, the subfilters can be run in parallel, and barring the approximation involving $e(n)$, each of them applies some LS algorithm.

Justification for Assuming $e_i(n) = e(n)$ for $1 \le i \le q$: Consider a scenario where the super filter has been decomposed into q subfilters, and instead of using the LS approach to compute the gains of the subfilters, the gains are computed as μ (a scalar constant) multiplied by the respective input vectors, i.e., the gain of the ith subfilter is $\mu X_i(n)$. Everything else remains the same as PRLS. Thus, we still assume $e_i(n) = e(n)$ for $1 \le i \le q$. It is not difficult to see that what we get is LMS [7]. Thus, the PRLS can be viewed as a generalized version of LMS, where the gains of the subfilters are computed in accordance with LS, and hence, it is expected to provide the benefits discussed earlier.

We can develop the PRLS algorithm around any LS or FLS algorithm. We give the PRLS algorithm based on the RLS algorithm as

given in [8] in Table III. $C_i(n)$ is used to denote the inverse of the correlation matrix of the ith subfilter, and λ denotes the forgetting factor.

III. Derivation of the Generalized Normal Equation

The weight vector update equation for the ith subfilter is

$$\hat{W}_i(n) = \hat{W}_i(n-1) + k_i(n)e(n) \tag{4}$$

where

$$k_i(n) = R_i^{-1}(n)X_i(n). \tag{5}$$

We combine (4) for all the q subfilters into the following single equation:

$$\hat{W}(n) = \hat{W}(n-1) + k(n)e(n) \tag{6}$$

where

$$k(n) = R'^{-1}(n)X(n) \tag{7}$$

$$R'(n) = \begin{bmatrix} R_1(n) & 0 & 0 & \cdots & 0 \\ 0 & R_2(n) & 0 & \cdots & 0 \\ 0 & 0 & R_3(n) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & R_q(n) \end{bmatrix} \tag{8}$$

and

$$e(n) = d(n) - \hat{W}^T(n-1)X(n). \tag{9}$$

Similar to $R(n)$, the matrix $R'(n)$ is also symmetric and non-negative definite. Substituting the value of $k(n)$ from (7) in (6), we get

$$\begin{aligned} \hat{W}(n) &= \hat{W}(n-1) + R'^{-1}(n)X(n)e(n) \\ &= \hat{W}(n-1) + R'^{-1}(n)X(n)\Big[d(n) - X^T(n)\hat{W}(n-1)\Big] \\ &= R'^{-1}(n)\Big[\Big(R'(n) - X(n)X^T(n)\Big) \\ &\quad \hat{W}(n-1) + X(n)d(n)\Big]. \end{aligned} \tag{10}$$

Assuming $\hat{W}(0)$ to be a null vector we can write:

$$\hat{W}(1) = R'^{-1}(1)X(1)d(1). \tag{11}$$

We can obtain $\hat{W}(2)$ by substituting for $\hat{W}(1)$ in (10). Thus

$$\begin{aligned} \hat{W}(2) = R'^{-1}(2)\Big[&\Big(R'(2) - X(2)X^T(2)\Big) \\ &\cdot R'^{-1}(1)X(1)d(1) + X(2)d(2)\Big]. \end{aligned} \tag{12}$$

This process can be continued. Thus, it can be verified that

$$\begin{aligned} \hat{W}(3) = R'^{-1}(3)\Big\{&\Big[R'(3) - X(3)X^T(3)\Big]R'^{-1}(2) \\ &\cdot \Big[R'(2) - X(2)X^T(2)\Big]R'^{-1}(1)X(1)d(1) \\ &+ \Big[R'(3) - X(3)X^T(3)\Big]R'^{-1}(2)X(2)d(2) \\ &+X(3)d(3)\Big\}. \end{aligned} \tag{13}$$

We see that as n increases, the number of terms in the expression for $\hat{W}(n)$ also increase. A close examination reveals that the relation can be expressed in the following compact form:

$$\hat{W}(n) = R'^{-1}(n)\sum_{j=1}^{n}\beta_n(j)X(j)d(j) \tag{14}$$

$$\Rightarrow R'(n)\hat{W}(n) = \sum_{j=1}^{n}\beta_n(j)X(j)d(j). \tag{15}$$

Here, $\beta_n(j)$ is an $N \times N$ matrix and is a function of only the input signal and the forgetting factor. This equation is in the same form as the widely known normal equation [3]

$$R(n)\hat{W}(n) = P(n) = \sum_{j=1}^{n}\lambda^{n-j}X(j)d(j) \tag{16}$$

with $R(n)$ replaced by $R'(n)$ and λ^{n-j} replaced by $\beta_n(j)$. We refer to (15) as the generalized normal equation (GNE) because the normal equation can be obtained from it as a special case and, because by appropriately choosing $R'(n)$, depending on the configuration used for implementing PRLS, it can be used to represent any SG or LS algorithm or combinations thereof. Thus, the GNE can be viewed as a generic framework providing a unified approach to adaptive algorithms. Furthermore, the intermediate configurations in between the two extremes when the super filter is not decomposed ($q = 1$ and it corresponds to RLS) and when it is decomposed into unity length subfilters ($q = N$ and it corresponds to a SG algorithm) provide a sort of *bridge* connecting the LS and SG family of algorithms.

Remark: The RLS algorithm has been derived as a method for solving the normal equation by the application of the matrix inversion lemma (MIL) [8]. Therefore, instead of PRLS, why not use the MIL to provide the solution of the GNE? Even if it is possible to do this, it is not desirable because it will destroy the parallel nature of the algorithm; hence, the savings that accrue because of the parallelism will be lost.

A. Expression for $\beta_n(j)$

It can be verified that $\beta_n(n) = I$, which is the identity matrix, whereas $\beta_n(j)$ can be expressed as

$$\begin{aligned} \beta_n(j) = \prod_{k=n}^{j+1}\Big[R'(k) - X(k)X^T(k)\Big]R'^{-1}(k-1) \\ \text{for} \quad 1 \le j \le (n-1). \end{aligned} \tag{17}$$

The computation of $\beta_n(j)$ is facilitated by the following two recursive relations:

$$\begin{aligned} \beta_n(j) = \beta_n(j+1)\Big[R'(j+1) - X(j+1)X^T(j+1)\Big] \\ \cdot R'^{-1}(j) \quad \text{for} \quad 1 \le j \le (n-1) \end{aligned} \tag{18}$$

$$\begin{aligned} \beta_n(j) = \Big[R'(n) - X(n)X^T(n)\Big]R'^{-1}(n-1)\beta_{n-1}(j) \\ \text{for} \quad 1 \le j \le (n-1). \end{aligned} \tag{19}$$

1) Special Case When $q = 1$: When the PRLS is run with $q = 1$, $R'(n)$ becomes equal to $R(n)$. Then

$$\beta_n(j) = \prod_{k=n}^{j+1}\Big[R(k) - X(k)X^T(k)\Big]R^{-1}(k-1) \tag{20}$$

$$= \lambda^{n-j}I \quad \text{for} \quad 1 \le j \le (n-1). \tag{21}$$

Substituting this value of $\beta_n(j)$ and $R'(n)$ in (14) we get, as expected, the normal equation

$$\hat{W}(n) = R^{-1}(n)\sum_{j=1}^{n}\lambda^{n-j}X(j)d(j). \tag{22}$$

2) Least Mean Square: Even though we derived the GNE assuming that each subfilter computes its gain according to least squares, in the resulting framework, we can treat even those cases where this is not true. Thus, when $q = N$ and the gains of the subfilters are computed to satisfy $R'^{-1}(n) = \mu I$, it corresponds to the LMS algorithm with μ as the step size. Then, it can be verified that

$$\beta_n(j) = \prod_{k=n}^{j+1}\Big[I - \mu X(k)X^T(k)\Big] \quad \text{for} \quad 1 \le j \le (n-1). \tag{23}$$

TABLE I
LMS ALGORITHM FOR ONE PROCESSOR

Available at time n	$X(n)$,$W(n-1)$ & $d(n)$
Step 1.	$y(n) = X^T(n)W(n-1)$
Step 2.	$e(n) = d(n) - y(n)$
Step 3.	$W(n) = W(n-1) + \mu X(n)e(n)$

TABLE II
CONCURRENT IMPLEMENTATION OF THE LMS ALGORITHM

Available at time n	$X_i(n)$,$W_i(n-1)$ for $1 \le i \le N$ & $d(n)$
Step 1. Do for $i = 1, 2, \ldots N$	$y_i(n) = X_i^T(n)W_i(n-1)$
Step 2.	$e(n) = d(n) - \sum_{i=1}^{N} y_i(n)$
Step 3. Do for $i = 1, 2, \ldots N$	$W_i(n) = W_i(n-1) + \mu X_i(n)e(n)$

Now, substituting this value of $\beta_n(j)$ and $R'(n)$ in (14) we get

$$\hat{W}(n) = \mu\left[\sum_{j=1}^{n-1}\left(\prod_{k=n}^{j+1}\left(I - \mu X(k)X^T(k)\right)\right)\cdot X(j)d(j) + X(n)d(n)\right]. \quad (24)$$

In the above equation, the weight vector computed by the LMS algorithm has for the first time been expressed only in terms of the input and desired signals and the step size. Equation (24) has the same relationship to the LMS algorithm as the normal equation has to all the LS algorithms.

We can do a similar straightforward analysis for the NLMS algorithm [9] by substituting $R'^{-1}(n)$ by $\mu(X^T(n)X(n))^{-1}$ or for any other algorithm (stochastic gradient or not) by choosing an appropriate $R'(n)$, depending on the expression for gain in that algorithm. The expressions for $\beta_n(j)$ will indicate the manner in which that algorithm is controlling its memory.

IV. SAVINGS IN COMPUTATIONAL TIME

Before analyzing the computational time required per iteration by PRLS, it is worthwhile to examine the same for LMS if its inherent concurrency is exploited. We define α as the time required for one multiplication and neglect the operations requiring addition. From Table I, we can see that if we have just one processor, LMS requires $2N\alpha$ time. Assuming that $X_i(n)$ and $W_i(n)$ are 1×1 vectors, i.e., scalars, and represent the ith elements of the vectors $X(n)$ and $W(n)$, respectively, we can rewrite LMS as in Table II, where $y_i(n)$ is a scalar denoting the output of the ith tap of the filter. If we have $N+1$ processors and allocate one processor for each coefficient and one (say, EP) for computing $e(n)$, we can see that each of steps 1 and 3 now requires α time instead of $N\alpha$. Hence, the total time required is $2\alpha+$ time required by EP for accessing $y_i(n)$'s + time required for global broadcast of $e(n)$ to N processors. Thus, the time required is very small and independent of N. For small values of N, this strategy may not be advantageous, but for large values of N, despite the overhead, it will offer significant savings.

Similarly, we can exploit the natural concurrency in PRLS by allocating one processor for each subfilter. When PRLS is developed around RLS, as in Table III, we see that steps 1 to 4 and step 6 can be done concurrently for each subfilter, and the time required for these steps is simply the time required by the longest subfilter. Hence, the total time required is the time required by the longest subfilter for steps 1 to 4 and step 6 + *overhead* (= time required by EP for accessing the scalar $y_i(n)$'s + time required for global broadcast of $e(n)$, which is a scalar, to q processors). When PRLS is developed around any FLS algorithm, the expression for time remains unchanged except for the first term, which will be replaced by time required by the longest subfilter for the computation of its gain $k_i(n)$, output $y_i(n)$, and update $W_i(n)$. We conclude that smaller the value of N_h (length of the longest subfilter) the greater will be the savings in computational time compared with the time required by even FLS algorithms (which use only one processor), particularly if N is large. Thus, PRLS has the potential of being used in real-time applications where computational time required per iteration is critical, and FLS may not be feasible. General expressions for the computational complexity and time required by PRLS cannot be written because they depend on the chosen configuration and the LS algorithm (conventional or fast) chosen for the gain computation of the subfilters.

V. COMPARATIVE PERFORMANCE OF THE ALTERNATIVES

For each value of N, q can take any value from 1 to N, and for each value of q, N_i's can take any value such that $\sum_{i=1}^{q} N_i = N$. Thus, depending on the value of N, a filter can be decomposed into subfilters (i.e., configured) in a large number of ways. In a given situation, an appropriate choice will depend on the number of processors available, the required convergence rate, and whether a real-time application is involved. We shall see in the simulation results that greater the value of q, the rate of convergence is less because each further decomposition implies that some terms in the input correlation matrix are being ignored. In other words, the larger the value of q, the farther the solution from LS and, hence, the rate of convergence is slower, i.e., there is a trade-off between the two. The rate of convergence is also dependent on which elements of the correlation matrix have been combined and which have been put in separate blocks. In fact, although we do not explore it here, it will be possible to find optimal decompositions for implementing PRLS, keeping in mind the correlation structure of the input signal. Thus, depending on the filter structure and input signal, it may be possible to obtain LS or, approximately, LS performance at a computational time that is less than that required by FLS algorithms.

In a practical situation, we can compute the time required by all the possible configurations and then arrange them in an order such that from one end to the other, there is a monotonic increase in the computational time. Then, we can start with the configuration with the lowest time and examine its performance. If it is not satisfactory for the application under consideration, go on to the next configuration in the hierarchy. Continue this way until a configuration that gives a satisfactory rate of convergence is obtained. Thus, there is no need to jump directly to LS because PRLS offers intermediate alternatives at a lower computational time. *This feature is specially attractive in the case of real-time implementations of large-order linear and Volterra filters or other multichannel applications, where FLS may be infeasible due to higher computational complexity, and SG algorithms may be inappropriate due to slower rate of convergence.*

If the power of the desired signal is low (how low depends on the structure of the filter) and the SNR is high enough, all PRLS configurations give the LS solution. Actually, the solutions are different for each configuration, but the difference between them is so small that it is not discernible on a plot of reasonable size. This happens because for low signal power, the error due to the approximation involved in PRLS also becomes low, and hence, the convergence of the algorithm improves. This is an important result as it implies that whenever we can have low power desired signals, there is no need to use any LS algorithm because the same performance can be obtained by using any PRLS configuration.

A. Comparison with FNTF

PRLS and FNTF in [5] and [6] are very similar in their objectives, and in a given situation, it may be required to make a choice between

TABLE III
CONCURRENT IMPLEMENTATION OF THE PRLS ALGORITHM (DEVELOPED AROUND RLS)

Available at time n		$X_i(n), W_i(n-1), C_i(n-1)$ for $1 \le i \le q$ & $d(n)$
Do Steps 1 to 4 for $i = 1, 2, \ldots q$	Step 1.	$y_i(n) = X_i^T(n) W_i(n-1)$
	Step 2.	$Y_i(n) = \lambda^{-1} C_i(n-1) X_i(n)$
	Step 3.	$k_i(n) = \frac{Y_i(n)}{1 + X_i^T(n) Y_i(n)}$
	Step 4.	$C_i(n) = \lambda^{-1} C_i(n-1) - k_i(n) Y_i^T(n)$
	Step 5.	$e(n) = d(n) - \sum_{i=1}^{q} y_i(n)$
Do Step 6 for $i = 1, 2, \ldots q$	Step 6.	$W_i(n) = W_i(n-1) + k_i(n) e(n)$

them. It is difficult to clearly demarcate the domains in which either of them would be the obvious choice. However, the following simple procedure can serve as a useful guideline. We will illustrate it for [5] (cost $2N + 5M$). It can be similarly extended to [6]. For the application under consideration, find the smallest value of M in [5] that gives the required convergence rate and the smallest value of q (and the associated N_i's) in PRLS which gives a similar performance. Now, if $(2N+5M)\alpha > 8N_h\alpha +$ overhead, choose PRLS; otherwise, choose FNTF. Here, we assume that the cost of the FLS chosen for implementing PRLS is $8N$ [1], [2]. Overhead time is architecture dependent, but it is almost guaranteed that for large values of N, it will be insignificant compared with the total time requirement. Due to its simplicity, better numerical properties, etc., RLS may sometimes be preferred to FLS algorithms in the implementation of PRLS. Then, we should compare $(2N + 5M)\alpha$ with $3N_h^2 + 4N_h +$ overhead (in the case of RLS, because of the square term, the total computational cost of PRLS becomes configuration dependent). Therefore, in this case, the smaller the value of N_h, the total cost will be less, i.e., PRLS provides low-cost alternatives.

VI. SIMULATION RESULTS

A detailed simulation study involving a wide range of lengths of linear and Volterra filters, different possible decompositions, optimal decompositions, different input signals, SNR's, comparison with SG algorithms, etc., is required to accurately fix the coordinates of PRLS in the pantheon of adaptive algorithms. The present study is directed at demonstrating its feasibility and useful properties and its potential as a suitable candidate in situations pointed out earlier. With this in mind, we present the learning curves for three different situations. They have been generated by ensemble averaging over 200 independent runs. The value of the forgetting factor λ has been chosen as 0.99. Plots of the *a priori* squared error versus the number of iterations for the case of zero mean white Gaussian input signals are presented.

Example 1—Fig. 1: The Volterra filter considered is defined as ($a_i(\cdot)$'s denote the filter taps):

$y(n) = \sum_{i=0}^{1} a_1(i)x(n-i) + \sum_{i=0}^{1} a_2(i, i+1)x(n-i)x(n-i-1) + \sum_{i=0}^{1} a_3(i, i+1, i+2)x(n-i)x(n-i-1)x(n-i-2) + \sum_{i=0}^{1} a_4(i, i+1, i+2, i+3)x(n-i)x(n-i-1)x(n-i-2)x(n-i-3)$.

Four configurations, which are denoted as Filters 1 to 4, have been chosen for applying PRLS. They include the following.

Filter 1: $q = 4$ and the four subfilters are defined as $y_1(n) = \sum_{i=0}^{1} a_1(i)x(n-i)$, $y_2(n) = \sum_{i=0}^{1} a_2(i, i+1)x(n-i)x(n-i-1)$, $y_3(n) = \sum_{i=0}^{1} a_3(i, i+1, i+2)x(n-i)x(n-i-1)x(n-i-2)$,and $y_4(n) = \sum_{i=0}^{1} a_4(i, i+1, i+2, i+3)x(n-i)x(n-i-1)x(n-i-2)x(n-i-3)$.

Filter 2: $q = 3$; $y_1(n) = \sum_{i=0}^{1} a_1(i)x(n-i)$, $y_2(n) = \sum_{i=0}^{1} a_2(i, i+1)x(n-i)x(n-i-1)$, and $y_3(n) = \sum_{i=0}^{1} a_3(i, i+1, i+2)x(n-i)x(n-i-1)x(n-i-2) + \sum_{i=0}^{1} a_4(i, i+1, i+2, i+3)x(n-i)x(n-i-1)x(n-i-2)x(n-i-3)$.

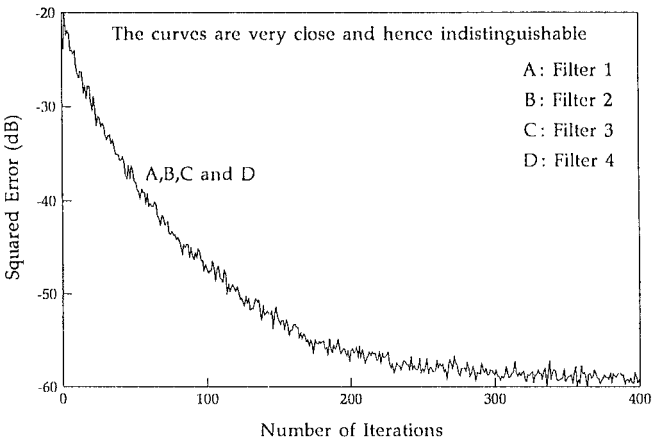

Fig. 1. PRLS algorithm (for low power desired signal).

Filter 3: $q = 2$; $y_1(n) = \sum_{i=0}^{1} a_1(i)x(n-i)$ and $y_2(n) = \sum_{i=0}^{1} a_2(i, i+1)x(n-i)x(n-i-1) + \sum_{i=0}^{1} a_3(i, i+1, i+2)x(n-i)x(n-i-1)x(n-i-2) + \sum_{i=0}^{1} a_4(i, i+1, i+2, i+3)x(n-i)x(n-i-1)x(n-i-2)x(n-i-3)$.

Filter 4: $q = 1$, i.e., it corresponds to the case when PRLS degenerates into RLS.

The power of the desired signal is -30 dB, and the noise power is -60 dB, to give an SNR of 30 dB. Although the four filters produce *different* curves, they are so close that they cannot be distinguished from each other. All of them require different computational times but give the same solution as given by Filter 4, which corresponds to LS. This example demonstrates the useful feature of PRLS, wherein it gives LS performance at the computational time of an SG algorithm, even for Volterra filters *if the power of the desired signal is sufficiently low*.

Example 2—Fig. 2: This is the same as Example 1, except that now, the power of the desired signal is increased to 6 dB, and that of the noise is increased to -34 dB, to give an SNR of 40 dB. It can be seen that the learning curves of the alternatives are widely separated. The computational time of Filter 1 that corresponds to the SG class is lowest, and hence, its convergence is also the slowest of the lot. As the computational time increases from Filters 1 to 4, the convergence rates also improve. *Thus, the trade-off between the number of subfilters and rate of convergence is demonstrated.* This example also demonstrates that given any filter, different configurations can be arranged in an order such that time and convergence rates increase from one end to the other so that the user has the flexibility to choose the right combination of time and convergence rate. Note that due to higher SNR, the convergence of Filter 4 in this example is better than that in Example 1.

Example 3—Fig. 3: A linear filter of order 10 has been chosen. Seven alternative configurations have been considered for implement-

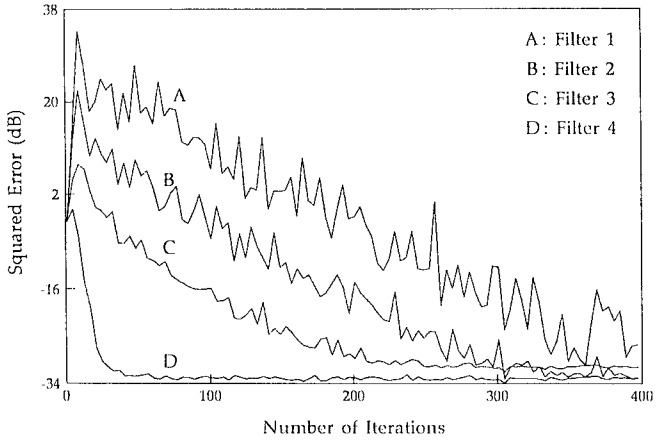

Fig. 2. PRLS algorithm for different configurations.

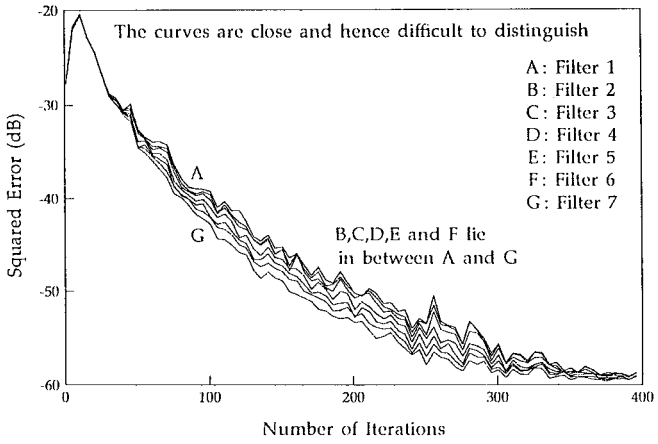

Fig. 3. PRLS algorithm (for low-power desired signal).

ing PRLS. The different configurations, designated as Filters 1–7 are described as follows.

Filter 1: $q = 10$ and each $N_i = 1$.
Filter 2: $q = 5$ and each $N_i = 2$.
Filter 3: $q = 4$ and $N_1 = N_2 = N_3 = 2$ while $N_4 = 4$.
Filter 4: $q = 4$ and $N_1 = N_2 = N_3 = 1$ while $N_4 = 7$.
Filter 5: $q = 2$, $N_1 = 2$, and $N_2 = 8$.
Filter 6: $q = 2$, $N_1 = 1$, and $N_2 = 9$.
Filter 7: $q = 1$.

The power of the desired signal is $-$ 20 dB, and the noise power is $-$ 60 dB, resulting in an SNR of 40 dB. All the alternatives are very close to the learning curve of LS, i.e., Filter 7. However, within the narrow band, the curve of an algorithm requiring lower computational time is above (implying lower convergence rate) all those algorithms requiring comparatively more time. Thus, the curves are in the following descending order (in terms of their position in Fig. 3): $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$. Filters 5 and 6 have both been decomposed into two subfilters. However, the convergence of Filter 6 ($N_h = 9$) is better because it uses more information about the input signal by ignoring lesser number of terms in the input correlation matrix than Filter 5 ($N_h = 8$), and hence, its computational time is also higher. If the power of the desired signal is further reduced and the SNR is kept the same, the band in the graph will be replaced by a line that will correspond to LS. We have seen the same phenomenon in Fig. 1. We deliberately chose the power of the desired signal to be slightly higher than the case when all the alternatives give LS solution to bring out the feature that as the power is increased, the learning curves of the alternatives begin to separate.

VII. Conclusions

We have demonstrated the feasibility and utility of a new class of adaptive algorithms that are, in general, suboptimal in the LS sense but require lesser computational time compared with existing schemes. The proposed class provides the user with the flexibility to choose the degree of suboptimality that can be traded off with the computational time requirement. However, it has inspired several questions, such as optimal decompositions for known input signals, proof of convergence, relation of signal power to rate of convergence, etc., which need to be explored. The equation the solution of which is sought by this class, has been derived and has been shown to lead to a generic framework for adaptive algorithms. This suggests that a unifying structure should also be possible from FNTF [5] and may be worth investigating.

References

[1] J. L. Botto and G. V. Moustakides, "Stabilizing the fast Kalman algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1332–1348, Sept. 1989.
[2] D. T. M. Slock and T. Kailath, "Numerically stable fast transversal filters for recursive least squares adaptive filtering," *IEEE Trans. Signal Processing*, vol. 39, pp. 92–114, Jan. 1991.
[3] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1985.
[4] J. Lee and V. J. Mathews, "A fast recursive least-squares adaptive second-order Volterra filter and its performance analysis," *IEEE Trans. Signal Processing*, vol. 41, pp. 1087–1102, Mar. 1993.
[5] G. V. Moustakides and S. Theodoridis, "Fast Newton transversal filters—A new class of adaptive estimation algorithms," *IEEE Trans. Signal Processing*, vol. 39, pp. 2184–2193, Oct. 1991.
[6] S. Theodoridis, G. V. Moustakides, and K. Berbiridis, "A fast Newton multichannel algorithm for decision feedback equalization," *IEEE Trans. Signal Processing*, vol. 43, pp. 327–331, Jan. 1995.
[7] B. Widrow, "Adaptive filters," in *Aspects of Network and System Theory*, R. E. Kalman and N. De Claris, Eds. New York: Holt, Rinehart, and Winston, 1970, pp. 563–587.
[8] S. Haykin, *Adaptive Filter Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1986.
[9] J. I. Nagumo and A. Noda, "A learning method for system identification," *IEEE Trans. Automat. Contr.*, vol. AC-12, pp. 282–287, June 1967.