A NEW BLOCK EXACT FAST LMS/NEWTON ADAPTIVE FILTERING ALGORITHM

Y. Zhou, S. C. Chan and K. L. Ho
Department of Electrical and Electronic Engineering,
The University of Hong Kong, Pokfulam Road, Hong Kong
Email: {yizhou, scchan, klho}@eee.hku.hk

ABSTRACT

This paper proposes a new block exact fast LMS/Newton algorithm for adaptive filtering. It is obtained by exploiting the shifting property of the whitened input of the fast LMS/Newton algorithm so that a block exact update can be carried out in the LMS part of the algorithm. The proposed algorithm has significantly reduced arithmetic complexity than but exact arithmetic equivalence to the LMS/Newton algorithm. Since short block length is allowed, the processing delay introduced is not excessively large as in conventional block filtering generalization. Implementation issues and the experimental results are given to illustrate the principle and efficiency of the proposed algorithm.

I. INTRODUCTION

Adaptive filtering is frequently employed in communications, control, and many other applications in which the statistical characteristics of the signals to be filtered are either unknown a priori or, in some cases, slowly time varying. Many adaptive filtering algorithms have been proposed [1] and they can broadly be classified into two different classes: the least mean squares (LMS) algorithm and the recursive least squares (RLS) algorithm. The LMS algorithm has a low computational complexity of $O(L)$ (where L is the number of taps of the adaptive filter), but it usually converges slowly. In the contrary, the RLS algorithm has a fast convergence but it is computationally more expensive with a complexity of $O(L^2)$. Different approaches have been proposed to improve the convergence property of the LMS algorithm and to reduce the complexity of the RLS algorithm. Interested readers are referred to the textbooks [1], [2] for various aspects of these two algorithms.

One very efficient class of algorithms is the fast Newton algorithm[3],[4]. In the fast Newton transversal filters (FNTF) [3] and the LMS/Newton algorithm [4], the input signal to the adaptive filter is modeled as a low, Mth-order auto-regressive (AR) process so that the Kalman gain vector in the RLS algorithm can be efficiently approximated. This leads to a reduced arithmetic complexity of $2L+5M$ for the FNTF [3] and $2L+6M$ for the LMS/Newton [4] algorithms. On the other hand, the convergence rate is considerably improved over the LMS algorithm. The LMS/Newton algorithm [4] also possesses the attractive properties of regular hardware implementation and more stable adaptation than the FNTF because of the use of the LMS algorithm in updating the weight vector. Since AR signal modeling has been found to provide a sufficiently accurate representation for many different types of signals, such as speech processing, it is expected the FNTF and the LMS/Newton algorithms will find applications in acoustic echo cancellation (AEC) [4] and other related applications.

In AEC and many other acoustic problems, large filter length might be required, and even the $2L$ computational complexity is somewhat demanding in real-time implementation. Methods for further reducing this computational complexity are therefore highly desirable. One efficient scheme is the block adaptive filtering [5], [6] where the filter coefficients are updated once per block of input data of length N. By exploiting the filtering nature of the adaptive filters, fast filtering/convolution techniques such as fast Fourier transforms (FFTs) and aperiodic convolution can be applied to achieve computational saving. The main limitation of traditional block filtering algorithms is that the block length is usually equal to the filter length. Therefore, a significant processing delay will be introduced, especially for long filter length. In [7], a new block filtering approach called the fast exact block LMS (FELMS) algorithm was introduced. The block-exact-based algorithms calculate the filtering errors in blocks using fast convolution algorithms and update the filter taps every N iterations. They are mathematically equivalent to their sample-by-sample counterparts with the same performance and a substantially reduced arithmetic complexity. Moreover, because a smaller block size can be chosen, the delay introduced is relatively low. This approach has been extended recently to the FNTF [8] and the Fast Affine Projection Algorithm (FAPA) [9].

In this paper, a new block-exact version of the fast LMS/Newton algorithm [4] is proposed. It is obtained by exploiting the shifting property of the whitened input of the LMS/Newton algorithm so that a block exact update can be carried out in the LMS part of the algorithm. The proposed algorithm has significantly reduced arithmetic complexity than but exact mathematical equivalence to the LMS/Newton algorithm [4]. The paper is organized as follows: the conventional fast LMS/Newton algorithm is described in section II. The proposed block exact Fast LMS/Newton algorithm is presented in Section III. Experimental results and comparison are given in section IV. Finally, conclusions are drawn in Section V.

II. THE FAST LMS/NEWTON ALGORITHM

Without loss of generality, consider the system identification problem as shown in Fig.1.

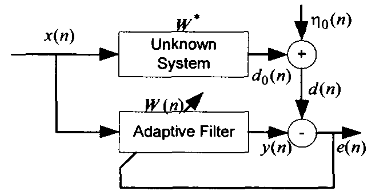

Fig. 1. System Identification Structure.

The unknown system with impulse response W^* and the adaptive filter $W(n)$ are simultaneously excited by an input signal $x(n)$. The adaptive filter continuously adjusts its weight to minimize some measures of the instantaneous error $e(n)$, which is the difference between the filter output $y(n)$ and the desired input $d(n)$. $d_0(n)$ is the output of the unknown system and $\eta_0(n)$ represents any possible modeling error and/or background noises. In the Newton algorithm, the weight update equation is given by

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

$$W(n+1) = W(n) + \mu \cdot e(n)\hat{R}^{-1}(n)X(n) \tag{2}$$

where $\hat{R}^{-1}(n)$ is the inverse of the estimated covariance matrix and μ is the stepsize. In the FNTF and the LMS/Newton algorithms, the input $x(n)$ is modeled as an M-th order AR process (usually $M \ll L$) so that $\hat{R}^{-1}(n)$ can be efficiently approximated using linear

0-7803-8346-X/04/$20.00 ©2004 IEEE

prediction method. As a result, the computational complexity of the basic Newton method can be significantly reduced, similar to the LMS algorithm, while offering significant performance improvement. More precisely, in the LMS/Newton algorithm, $\hat{\boldsymbol{R}}^{-1}(n)$ can be factored into the following form:

$$\hat{\boldsymbol{R}}^{-1}(n) = \boldsymbol{L}_M^{-T}(n)\boldsymbol{D}^{-1}(n)\boldsymbol{L}_M^{-1}(n) \qquad (3)$$

where

$$\boldsymbol{L}_M^{-1}(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ a_{1,1}(n) & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{M,M}(n) & a_{M,M-1}(n) & \cdots & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & a_{M,M}(n-1) & \cdots & a_{M,1}(n-1) & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & a_{M,M}(n-N+M+1) & a_{M,M-1}(n-N+M+1) & \cdots & 1 \end{bmatrix}$$

is an ($L \times L$) lower triangular matrix with the element $a_{p,i}(n)$ being the *i*-th coefficient of the *p*-th order backward predictor for $x(n)$, and $\boldsymbol{D}(n)$ is a diagonal matrix whose *i*-th element is the estimated power of the *i*-th backward prediction error. Note that the (*M*+*1*)-th through the *L*-th rows of $\boldsymbol{L}_M^{-1}(n)$ are shifted version of each other. Define the extended input and coefficient vector of $\boldsymbol{X}(n)$ and $\boldsymbol{W}(n)$ as follows:

$$\boldsymbol{X}_E(n) = [x(n+M),\ldots,x(n-L-M+1)]^T ; \qquad (4)$$

$$\boldsymbol{W}_E(n) = [w_{-M}(n),\ldots,w_{N+M-1}(n)]^T . \qquad (5)$$

By freezing the unneeded first and last *M* elements of $\boldsymbol{W}_E(n)$ at zero and denoting the resultant vector as $\boldsymbol{W}(n)$, the LMS/Newton algorithm can be written as follows:

$$e(n) = d(n-M) - \boldsymbol{X}^T(n-M)\boldsymbol{W}(n) \qquad (6)$$

$$\boldsymbol{W}(n+1) = \boldsymbol{W}(n) + 2\mu e(n)\boldsymbol{u}_a(n) \qquad (7)$$

$$\boldsymbol{u}_a(n) = \boldsymbol{L}_2(n)\widetilde{\boldsymbol{D}}^{-1}(n)\boldsymbol{L}_1(n)\boldsymbol{X}_E(n) \qquad (8)$$

where $\boldsymbol{L}_1(n)$ and $\boldsymbol{L}_2(n)$ are respectively (*L*+*M*)-by-(*L*+*2M*) and *L*-by-(*L*+*M*) matrices whose rows are consecutively shifted and delayed coefficients of the *M*-th order forward and backward predictors $[a_{M,M}(n), a_{M,M-1}(n),\ldots,1]$ and $[1, a_{M,1}(n),\ldots,a_{M,M}(n)]$. By exploiting the shifting property of $\boldsymbol{u}_a(n)$ and $\boldsymbol{b}_E(n) = \boldsymbol{L}_1(n)\boldsymbol{X}_E(n)$, it is possible to reduce the computational complexity of the algorithm to *2N*+*6M* multiplications and additions. The predictor parameters can be efficiently calculated using a lattice predictor and the Levinson-Durbin algorithm.

III. THE PROPOSED BLOCK EXACT FAST LMS/NEWTON ALGORITHM

For simplicity, let us assume that the block length N is a factor of the filter length: $L = NP$. The following equations can be obtained by iterating (6) and (7) for *N* consecutive time steps:

$$\begin{aligned} e(n-N+1) &= d(n-N-M+1) - \boldsymbol{X}^T(n-N-M+1)\boldsymbol{W}(n-N+1); \\ e(n-N+2) &= d(n-N-M+2) - \boldsymbol{X}^T(n-N-M+2)\boldsymbol{W}(n-N+2) \\ &= d(n-N-M+2) - \boldsymbol{X}^T(n-N-M+2)\boldsymbol{W}(n-N+1) \\ &\quad -2\mu e(n-N+1)\boldsymbol{X}^T(n-N-M+2)\boldsymbol{u}_a(n-N+1) \\ \vdots \qquad & \qquad \vdots \qquad \qquad \vdots \end{aligned}$$

We can thus rewrite the error sequence of the fast LMS/Newton algorithm at time interval: $n-N+1, n-N+2,\ldots,n-1,n$ as

$$\begin{bmatrix} e(n-N+1) \\ e(n-N+2) \\ \vdots \\ e(n) \end{bmatrix} = \begin{bmatrix} d(n-N-M+1) \\ d(n-N-M+2) \\ \vdots \\ d(n-M) \end{bmatrix} - \begin{bmatrix} \boldsymbol{X}^T(n-N-M+1) \\ \boldsymbol{X}^T(n-N-M+2) \\ \vdots \\ \boldsymbol{X}^T(n-M) \end{bmatrix} \times$$

$$\boldsymbol{W}(n-N+1) - \boldsymbol{S}(n)\begin{bmatrix} e(n-N+1) \\ e(n-N+2) \\ \vdots \\ e(n) \end{bmatrix} \qquad (9)$$

where $\boldsymbol{S}(n) = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ s_1(n-N+2) & 0 & 0 & \cdots & 0 \\ s_2(n-N+3) & s_1(n-N+3) & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{N-1}(n) & s_{N-2}(n) & \cdots & s_1(n) & 0 \end{bmatrix}$,

$$s_i(n) = 2\mu \boldsymbol{X}^T(n-M)\boldsymbol{u}_a(n-i),\ i = 1,2,\ldots,N-1. \qquad (10)$$

In the block exact algorithm, the filter weight is adjusted once per data block instead of being updated at every data sample. Its block update recursion can be readily derived by beginning from the step-by-step update of (7) and by combining properly the resulting equations for all time steps within the current block as follows:

$$\boldsymbol{W}(n+1) = \boldsymbol{W}(n-N+1) + 2\mu[\boldsymbol{u}_a(n-N+1) \quad \boldsymbol{u}_a(n-N+2) \quad \cdots$$

$$\cdots \quad \boldsymbol{u}_a(n)]\begin{bmatrix} e(n-N+1) \\ e(n-N+2) \\ \vdots \\ e(n) \end{bmatrix}. \qquad (11)$$

Equations (9) and (11) can be rewritten more compactly as:

$$\underline{e}(n) = \underline{d}(n) - \underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1) - \boldsymbol{S}(n)\underline{e}(n) \qquad (12a)$$

$$\boldsymbol{W}(n+1) = \boldsymbol{W}(n-N+1) + 2\mu\boldsymbol{U}_a(n)\underline{e}(n) \qquad (12b)$$

where $\underline{e}(n)$ and $\underline{d}(n)$ are both ($N \times 1$) vectors; $\underline{\boldsymbol{X}}(n-M)$ and $\boldsymbol{U}_a(n)$ are ($N \times L$) and ($L \times N$) matrices respectively. Further, $\underline{e}(n)$ on both sides of (12a) can be combined to give:

$$\begin{aligned} [\boldsymbol{S}(n) + \boldsymbol{I}]\underline{e}(n) &= \underline{d}(n) - \underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1) \\ \underline{e}(n) &= [\boldsymbol{S}(n) + \boldsymbol{I}]^{-1}[\underline{d}(n) - \underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1)] \\ &= \boldsymbol{G}(n)[\underline{d}(n) - \underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1)]. \end{aligned} \qquad (13)$$

$\boldsymbol{G}(n)$ is a lower triangular matrix which can be factored further as the product of a series of component matrices as follows:

$$\boldsymbol{G}(n) = \boldsymbol{G}_{N-1}(n)\boldsymbol{G}_{N-2}(n)\cdots\boldsymbol{G}_1(n)$$

$$\boldsymbol{G}_i(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 & & & 0 \\ 0 & 1 & & \vdots & & & \vdots \\ \vdots & \vdots & & \vdots & & & \vdots \\ \vdots & \vdots & & 0 & & & \vdots \\ -s_i(n-N+i+1) & -s_{i-1}(n-N+i+1) & \cdots & -s_1(n-N+i+1) & 1 & & \vdots \\ 0 & & & 0 & & & \vdots \\ \vdots & & & \vdots & & & \vdots \\ \vdots & & & \vdots & & & 0 \\ 0 & & & 0 & \cdots & 0 & 1 \end{bmatrix} \qquad (14)$$

with the nonzero elements only appearing at the (*i*+*1*)-th row in the lower triangular part. Apparently the second term on the right hand side of (13) represents a fixed coefficient filtering of the input within the block of length *N*, which can be utilized to reduce the arithmetic complexity. Towards this end, we first rewrite it as:

$$\underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1) = \begin{bmatrix} \boldsymbol{A}_{N-1} & \boldsymbol{A}_N & \cdots & \boldsymbol{A}_{2N-3} & \boldsymbol{A}_{2N-2} \\ \boldsymbol{A}_{N-2} & \boldsymbol{A}_{N-1} & & & \boldsymbol{A}_{2N-3} \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ \boldsymbol{A}_1 & \vdots & & & \boldsymbol{A}_N \\ \boldsymbol{A}_0 & \boldsymbol{A}_1 & \cdots & \boldsymbol{A}_{N-2} & \boldsymbol{A}_{N-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \\ \vdots \\ \vdots \\ \boldsymbol{w}_{N-2} \\ \boldsymbol{w}_{N-1} \end{bmatrix} (n-N+1).$$

where

$$\boldsymbol{A}_j = [x(n-M-j), x(n-M-N-j), \cdots x(n-M-Ni-j), \cdots$$

$$\cdots, x(n-M-L+N-j)] \ ,$$
$$\text{for } j = 0,1,\ldots,2N-2; \ i = 0,1,\ldots,(L/N)-1 \qquad (15)$$

$$\boldsymbol{w}_k(n-N+1) = [w_k, w_{k+N}, \ldots, w_{k+ni}, \ldots, w_{k+l-N}]^T (n-N+1)$$
$$\text{for } k = 0,1,\ldots, N-1, \qquad (16)$$

are respectively $(1 \times (L/N))$ row vector and $((L/N) \times 1)$ column vector, and $\underline{\boldsymbol{X}}(n-M)$ is found to be a block-Toeplitz matrix with polyphase components. Likewise, $\boldsymbol{U}_a(n)$ in (11) can be reorganized by defining

$$\boldsymbol{B}_j = [u_a(n-j), u_a(n-N-j), \cdots u_a(n-Ni-j), \cdots$$
$$\cdots, x(n-L+N-j)] \ ,$$
$$\text{for } j = 0,1,\ldots,2N-2; \ i = 0,1,\ldots,(L/N)-1 . \qquad (17)$$

so that (11) can be rewritten as (20b) shown below.

The multiplication of $\boldsymbol{G}(n)$ in (14) can be efficiently implemented by utilizing the relationship between successive calculations. More precisely, the elements in the first column can be expressed in terms of those in the last row of the previous update as

$$s_i(n-N+i+1) = s_i(n-N) + 2\mu[\sum_{j=0}^{i} x(n-M-N+i-j+1) \times$$
$$u_a(n-N-j+1) - \sum_{j=0}^{i} x(n-M-L-N+i-j+1) \cdot u_a(n-L-N-j+1)],$$
$$\text{for } i = 1,\ldots, N-1 , \qquad (18)$$

and the remaining elements on each sub-diagonal can be updated one after another using the results obtained in (17):

$$s_i(n+1) = s_i(n) + 2\mu[x(n-M+1) \cdot u_a(n-i+1) -$$
$$x(n-M-L+1) \cdot u_a(n-i-L+1)]. \qquad (19)$$

Summarizing, we have obtained the following equivalent block version of the fast LMS/Newton algorithm:

$$\underline{e}(n) = \boldsymbol{G}(n)[\underline{d}(n) -$$
$$\begin{bmatrix} \boldsymbol{A}_{N-1} & \boldsymbol{A}_N & \cdots & \boldsymbol{A}_{2N-3} & \boldsymbol{A}_{2N-2} \\ \boldsymbol{A}_{N-2} & \boldsymbol{A}_{N-1} & & & \boldsymbol{A}_{2N-3} \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ \boldsymbol{A}_1 & \vdots & & & \boldsymbol{A}_N \\ \boldsymbol{A}_0 & \boldsymbol{A}_1 & \cdots & \boldsymbol{A}_{N-2} & \boldsymbol{A}_{N-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \\ \vdots \\ \vdots \\ \boldsymbol{w}_{N-2} \\ \boldsymbol{w}_{N-1} \end{bmatrix} (n-N+1)] \qquad (20a)$$

$$\begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \\ \vdots \\ \vdots \\ \boldsymbol{w}_{N-2} \\ \boldsymbol{w}_{N-1} \end{bmatrix} (n+1) = \begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \\ \vdots \\ \vdots \\ \boldsymbol{w}_{N-2} \\ \boldsymbol{w}_{N-1} \end{bmatrix} (n-N+1) + 2\mu \times$$
$$\begin{bmatrix} \boldsymbol{B}_{N-1}^T & \boldsymbol{B}_{N-2}^T & \cdots & \boldsymbol{B}_1^T & \boldsymbol{B}_0^T \\ \boldsymbol{B}_N^T & \boldsymbol{B}_{N-1}^T & & & \boldsymbol{B}_1^T \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ \boldsymbol{B}_{2N-3}^T & \vdots & & & \boldsymbol{B}_{N-2}^T \\ \boldsymbol{B}_{2N-2}^T & \boldsymbol{B}_{2N-3}^T & \cdots & \boldsymbol{B}_N^T & \boldsymbol{B}_{N-1}^T \end{bmatrix} \underline{e}(n) \qquad (20b)$$

Similar to the FELMS algorithm in [7] , the matrix multiplications of $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ in (20a) and (20b) can be implemented using the fast convolution algorithms in [10],[11] or the FFT. Here we give an example with N=2 for the purpose of illustration. Consider

$$\begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ \boldsymbol{A}_0 & \boldsymbol{A}_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \end{bmatrix} (n-1) = \begin{bmatrix} \boldsymbol{A}_1(\boldsymbol{w}_0 + \boldsymbol{w}_1) + (\boldsymbol{A}_2 - \boldsymbol{A}_1)\boldsymbol{w}_1 \\ \boldsymbol{A}_1(\boldsymbol{w}_0 + \boldsymbol{w}_1) - (\boldsymbol{A}_1 - \boldsymbol{A}_0)\boldsymbol{w}_0 \end{bmatrix} (n-1) \qquad (21a)$$

$$\begin{bmatrix} \boldsymbol{B}_1^T & \boldsymbol{B}_0^T \\ \boldsymbol{B}_2^T & \boldsymbol{B}_1^T \end{bmatrix} \begin{bmatrix} e(n-1) \\ e(n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_1^T(e(n-1) + e(n)) - (\boldsymbol{B}_1 - \boldsymbol{B}_0)^T e(n) \\ \boldsymbol{B}_1^T(e(n-1) + e(n)) + (\boldsymbol{B}_2 - \boldsymbol{B}_1)^T e(n-1) \end{bmatrix} . \qquad (21b)$$

According to the definition of (15) and (17), we have:

$$(\boldsymbol{A}_1 - \boldsymbol{A}_0) = [x(n-M-1) - x(n-M), x(n-M-3) - x(n-M-2), \ldots$$
$$\ldots, x(n-M-L+1) - x(n-M-L+2)]. \qquad (22)$$

$$(\boldsymbol{B}_1 - \boldsymbol{B}_0) = [u_a(n-1) - u_a(n), u_a(n-3) - u_a(n-2), \ldots$$
$$\ldots, u_a(n-L+1) - u_a(n-L+2)]. \qquad (23)$$

It can be easily verified that only the first items $x(n-M-1) - x(n-M)$ in (22) and $u_a(n-1) - u_a(n)$ in (23) need to be calculated for each update, and the remaining elements can be acquired from the calculation of $(\boldsymbol{A}_1 - \boldsymbol{A}_0)$ and $(\boldsymbol{B}_1 - \boldsymbol{B}_0)$ at the previous updating step. Similar savings also apply to $(\boldsymbol{A}_2 - \boldsymbol{A}_1)$ and $(\boldsymbol{B}_2 - \boldsymbol{B}_1)$.

The block exact fast LMS/Newton algorithm with a block length of N=2 described by (20)~(21) requires $3L+2$ multiplications and $4L+10$ additions plus another $12M$ multiplications and $12M$ additions for updating $\boldsymbol{u}_a$. Compared with $4L$ multiplications and $4L$ additions plus the same cost for updating $\boldsymbol{u}_a$ needed by the original algorithm, the block exact scheme reduces 25% of the total number of multiplications with only very few extra additions. Following [4] and [7], the implementation block diagrams of the original and proposed algorithms can be naturally obtained in Figure 2. For the special case where $N = 2^n$ the fast Fourier transform (FFT) can be employed, the computational complexity is $2(3/2)^n L + 2^n(3 \cdot 2^n - 5)/2 + 1$ multiplications and $2(2(3/2)^n - 1)L + 2^{n+2}(2^{n-1} - 1) + 4 \cdot 3^n - 2$ additions. The computational complexities of the proposed algorithm and its non-block counterpart for different block length N and filter length L are summarized in Table 1. It can be seen that significant saving in arithmetic operation is achieved by the proposed block exact algorithm. A closer look at (21) also reveals that other than the cost for updating $\boldsymbol{u}_a$ (i.e. whitening the input as a low order AR process, which is quite small compared with the rest), the computational complexity of the proposed algorithm is rather close to the FELMS. The main difference is (21b), where $\boldsymbol{B}_k - \boldsymbol{B}_{k-1}$ has to be computed in addition to $\boldsymbol{A}_k - \boldsymbol{A}_{k-1}$ as in [7]. Fortunately, thanks for the computational saving in updating $\boldsymbol{B}_k - \boldsymbol{B}_{k-1}$, only $2N-2$ extra additions are needed. Together with the good numerical stability and simple hardware implementation, the proposed algorithm is a good alternative to the block exact FNTF algorithm in [8].

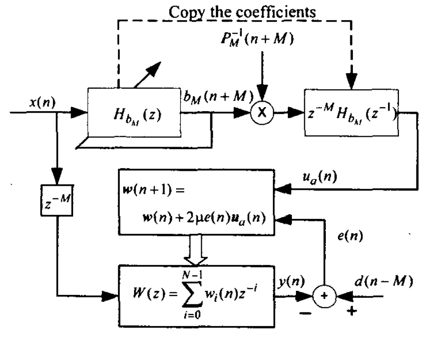

Fig. 2 (a)

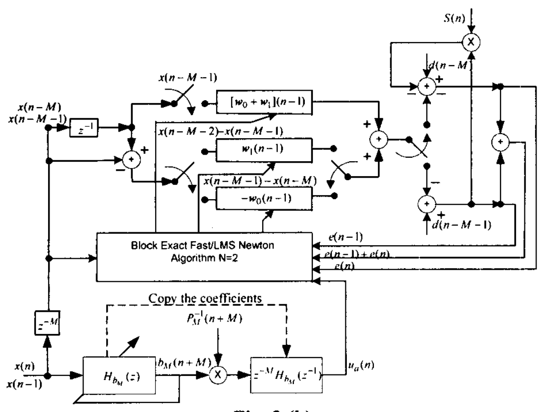

Fig. 2 (b)

Fig. 2. The implementation block diagrams for (a) fast LMS/Newton algorithm and (b) block exact Fast LMS/Newton algorithm.

Filter Length L	Block Length N	AR Process Order M	Fast LMS/Newton Algorithm		Block Exact Fast LMS/Newton Algorithm	
			Number of Additions	Number of Multiplications	Number of Additions	Number of Multiplications
32	4	5	376	376	386	279
128	16	5	4576	4576	3586	2121
512	32	5	33728	33728	18378	10193
1024	64	5	132992	132992	57378	31233

Table 1. Arithmetic complexity comparison of the proposed algorithm and its non-block counterpart.

IV. SIMULATION RESULTS

We now evaluate the performance of the proposed algorithm using computer simulation. The block length is chosen to be N=4 and the experimental environment is identical to that in [4], except that the order of the unknown system is increased to 32 so that it is an integer multiple of N. The system coefficients are randomly generated and the AR process is fixed with coefficients [1 –0.65 0.693 –0.22 0.309 –0.177] and is normalized to have unit power. The power of the additive white Gaussian noise is set to be $\delta^2_{\eta_0} = 0.0001$. Four different algorithms, the RLS, normalized LMS (NLMS), the proposed block exact fast LMS/Newton and the original fast LMS/Newton are tested. The forgetting factor of the RLS algorithm, the step size of NLMS algorithm, and the step size of the fast LMS/Newton algorithms have been tuned so that the steady state MSE of all algorithms are appropriately identical at –40dB. The results are averaged for 100 independent runs. From Fig. 3, it can be seen that the two fast LMS/Newton algorithms exhibit superior convergence performance in comparison with the NLMS algorithm. Besides, the MSE curves of the block exact fast LMS/Newton algorithm and the original fast LMS/Newton algorithm are identical to each other, which substantiates their arithmetic equivalence. For clarity of presentation, these two curves are separately plotted in Figure 4.

V. CONCLUSION

A new block exact fast LMS/Newton algorithm for adaptive filtering is presented. The proposed algorithm is mathematically equivalent to its original counterpart but has a substantially reduced arithmetic complexity. Since short block length is allowed, the processing delay introduced is not excessively large as in conventional block algorithm generalization. Implementation issues and the experimental results are also presented to reveal the principle and efficiency of the proposed algorithm.

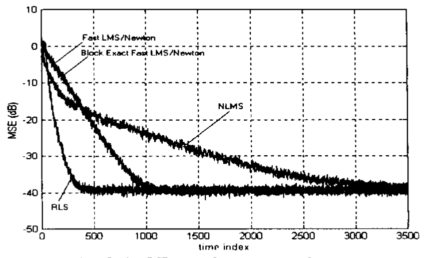

Fig. 3. MSE results versus time *n*.

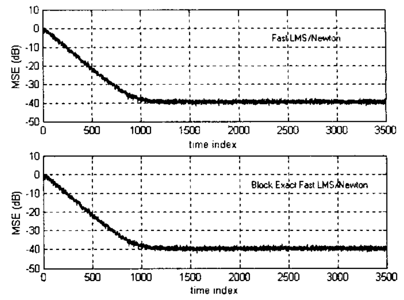

Fig. 4. MSE results versus time *n*: (lower) the proposed algorithm and (upper) its non-block counterpart.

REFERENCES

[1] S. Haykin, *Adaptive Filter Theory*. 4th edition, Prentice Hall Press, 2001.

[2] J. G. Proakis, C. M. Rader, F. Ling, C. L. Nikias, M. Moonen, and I. K. Proudler. *Algorithms for Statistical signal Processing*. Prentice Hall Press, New Jersey, 2002.

[3] G. V. Moustakides and S. Theodoridis, "Fast Newton Transversal Filters-A New Class of Adaptive Estimation Algorithms," *IEEE Trans. on Signal Processing*, vol. 39, No. 10, pp. 2184-2193, Oct. 1991.

[4] B. F. Boroujeny, "Fast LMS/Newton Algorithms Based on Autoregressive Modeling and Their Application to Acoustic Echo Cancellation." *IEEE Trans. on Signal Processing*. vol. 45, No. 8, pp. 1987-2000, Aug. 1997.

[5] G. A. Clark, S. K. Mitra, and S. R. Parker, "Block Implementation of Adaptive Digital Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, No. 3, pp. 744-752, June 1981.

[6] S. Narayan, A. M. Peterson and M. J. Narasimha, "Transform domain LMS algorithm," *IEEE Trans. Acoust. Speech, Signal Proc.*, vol. ASSP-31, pp. 609-615, June 1983.

[7] J. Benesty and P. Duhamel, "A Fast Exact Least Mean Square Adaptive Algorithm," *IEEE Trans. Signal Processing*, vol. 40, No. 12, pp. 2904-2920, Dec. 1992.

[8] K. Berberidis and S. Theodoridis, "A New Fast Block Adaptive Algorithm," *IEEE Trans. Signal Processing*, vol. 47, No. 1, pp. 75-87, Jan. 1999.

[9] M. Tanaka, S. Makino and J. Kojima, "A Block Exact Fast Affine Projection Algorithm," *IEEE Trans. Speech and Audio Processing*, vol.7, No. 1, pp. 79-86, Jan. 1999.

[10] Z. J. Mou and P. Duhamel, "Fast FIR filtering: algorithms and implementation," *Signal Processing*, vol. 377-384, Dec. 1987.

[11] Z. J. Mou and P. Duhamel, "Short-Length FIR Filters and Their Use in Fast Nonrecursive Filtering," *IEEE Trans. Signal Processing*, vol. 39, No. 6, pp. 1322-1332, June 1991.