Correspondence

A New Block-Exact Fast LMS/Newton Adaptive Filtering Algorithm

Y. Zhou, S. C. Chan, and K. L. Ho

***Abstract*—This correspondence proposes a new block-exact fast least–mean squares (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 less computational complexity than, but exact mathematical equivalence to, the fast 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.**

***Index Terms*—Adaptive filter, block exact, fast least-mean squares (LMS)/Newton 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. On the contrary, the RLS algorithm has a fast convergence, but it is more computationally 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] and [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 fast LMS/Newton algorithm [4], the input signal to the adaptive filter is modeled as a low M-order autoregressive (AR) process so that the Kalman gain vector in the Newton algorithm can be efficiently approximated. This leads to a reduced computational complexity of $2L+5M$ for the FNTF and $2L+6M$ for the fast LMS/Newton algorithms. On the other hand, the convergence rate is considerably improved over the LMS algorithm. According to [4], the fast LMS/Newton algorithm also possesses the attractive properties of regular hardware implementation and more stable adaptation than the FNTF algorithm 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 that the FNTF and the fast LMS/Newton algorithm will find applications in acoustic echo cancellation (AEC) as well as other related applications [5].

Manuscript received August 16, 2004; revised March 10, 2005. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Fulvio Gini.

The authors are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong SAR, China (e-mail: yizhou@eee.hku.hk; scchan@eee.hku.hk; klho@eee.hku.hk).

Digital Object Identifier 10.1109/TSP.2005.861099

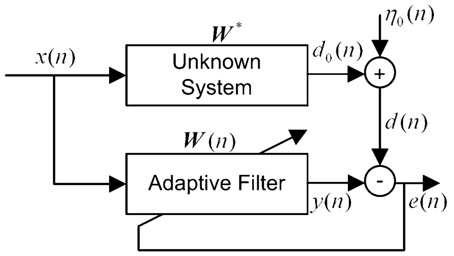

Fig. 1. System identification structure.

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 technique [6], [7], 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 significant computational saving. The main limitation of these 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 [8], a new block filtering approach called the fast block-exact 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 step-by-step nonblock counterparts with the same performance and a substantially reduced computational complexity. Moreover, because a smaller block size can be chosen, the delay introduced is relatively low. Because of these potential advantages, the block-exact concept has been applied to the FNTF [9], the fast RLS [10], and the affine projection algorithms (APA) [11]–[13].

In this correspondence, a new block-exact version of the fast LMS/Newton algorithm is proposed. The main difficulty in deriving such a block-exact algorithm is that the input is whitened by AR modeling before going through the LMS update. As a result, the structure of the LMS/Newton algorithm differs considerably from the LMS algorithm. Fortunately, it is found that the whitened input of the LMS/Newton algorithm also possesses the shifting property, which is a key property in deriving the FELMS algorithm. Using this property and the block-exact concept, a new block-exact update for the LMS part in the LMS/Newton algorithm is derived. The resulting algorithm is mathematically equivalent to the original fast LMS/Newton algorithm, but its computational complexity is significantly lower. On the other hand, due to the differences in the inputs of the fast LMS/Newton and the LMS algorithms, the proposed block-exact algorithm has a slightly higher complexity than, but a similar structure to, the FELMS algorithm.

This correspondence is organized as follows. The conventional fast LMS/Newton algorithm is described in Section II. The proposed block-exact fast LMS/Newton algorithm and its implementation issues are presented in Section III. Experimental results and comparisons are presented in Section IV. Finally, conclusions are drawn in Section V.

II. The Fast LMS/Newton Algorithm

Without loss of generality, consider the identification of an unknown system with impulse response $\boldsymbol{W}^*$ shown in Fig. 1. The unknown system and the adaptive filter with impulse response $\boldsymbol{W}(n)$ are simultaneously excited by an input signal $x(n)$. The adaptive filter continuously adjusts its weight coefficients to minimize certain performance criterion such as the mean-square error (MSE) of the instantaneous estimation error $e(n)$, which is the difference between the desired signal $d(n)$ and the filter output $y(n)$. $d_0(n)$ 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 equations are given by

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

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

where $\hat{\boldsymbol{R}}^{-1}(n)$ is the inverse of the estimated input covariance matrix, and μ is the step size that controls the convergence, tracking speed, and the steady-state error of the algorithm. In the FNTF and the fast LMS/Newton algorithms, the input $x(n)$ is modeled as an M-order AR process (usually $M \ll L$) so that $\hat{\boldsymbol{R}}^{-1}(n)$ can be efficiently approximated using linear prediction method. As a result, the computational complexity of the basic Newton method can be significantly reduced, similarly to that of the LMS algorithm, while offering significant performance improvement. More precisely, in the fast LMS/Newton algorithm, the input covariance matrix $\hat{\boldsymbol{R}}(n)$ is decomposed using the Cholesky factorization [1] so that its inverse can be written as

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

where $\boldsymbol{L}_M(n)$ is an $L \times L$ lower triangular matrix consisting of the coefficients of the backward predictors. However, due to the AR model assumption of the input, it can be simplified to (4), shown at the bottom of the page, where the element $a_{p,i}(n)$ is the ith coefficient of the pth-order backward predictor for $x(n)$, and $\boldsymbol{D}(n)$ is a diagonal matrix whose ith element is the estimated power of the ith backward prediction error.

In [4], two algorithms with different structural complexity and applicability are presented. The algorithm presented in this paper is based on Algorithm 2, which has a much simpler structure than Algorithm 1 and, hence, is more suitable for hardware implementation.

Note that the $(M+1)$th through the Lth rows of $\boldsymbol{L}_M(n)$ are shifted version of each other. To simplify notation, let us define the extended input and coefficient vectors of $\boldsymbol{X}(n)$ and $\boldsymbol{W}(n)$ as follows:

$$\boldsymbol{X}_E(n) = [x(n+M), \ldots, x(n), \ldots, x(n-L-M+1)]^T \tag{5}$$

$$\boldsymbol{W}_E(n) = [w_{-M}(n), \ldots, w_0(n), \cdots, w_{L+M-1}(n)]^T. \tag{6}$$

By freezing the first M and last M unnecessary elements of $\boldsymbol{W}_E(n)$ to zero during all iterations and denoting the resultant vector as $\boldsymbol{W}(n)$, the fast LMS/Newton algorithm can be written as

$$e(n) = d(n-M) - \boldsymbol{X}^T(n-M)\boldsymbol{W}(n) \tag{7}$$

$$\boldsymbol{W}(n+1) = \boldsymbol{W}(n) + 2\mu e(n)\boldsymbol{u}_a(n) \tag{8}$$

$$\boldsymbol{u}_a(n) = \boldsymbol{L}_2(n)\tilde{\boldsymbol{D}}^{-1}(n)\boldsymbol{L}_1(n)\boldsymbol{X}_E(n) \tag{9}$$

where $\tilde{\boldsymbol{D}}^{-1}(n)$ is a diagonal matrix with appropriate dimension and its ith diagonal element is the estimated power of the ith element of the vector $\boldsymbol{L}_1(n)\boldsymbol{X}_E(n)$. $\boldsymbol{L}_1(n)$ and $\boldsymbol{L}_2(n)$ are, respectively, $(L+M) \times (L+2M)$ and $L \times (L+M)$ matrices shown in (10) and (11) at the bottom of the page. By exploiting the shifting property of $\boldsymbol{u}_a(n)$ and $\boldsymbol{L}_1(n)\boldsymbol{X}_E(n)$, it is possible to reduce the computational complexity of the algorithm to $2L + 6M$ multiplications and additions for each iteration. The predictor parameters can be efficiently calculated using a lattice predictor and the Levinson–Durbin algorithm. The details can be found in [4].

$$\boldsymbol{L}_M(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ a_{1,1}(n) & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{M,M}(n) & a_{M,M-1}(n) & \cdots & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & a_{M,M}(n-1) & \cdots & a_{M,1}(n-1) & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & a_{M,M}(n-L+M+1) & a_{M,M-1}(n-L+M+1) & \cdots & 1 \end{bmatrix}. \tag{4}$$

$$\boldsymbol{L}_1(n) = \begin{bmatrix} a_{M,M}(n) & a_{M,M-1}(n) & \cdots & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & a_{M,M}(n-1) & \cdots & a_{M,1}(n-1) & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & a_{M,M}(n-L-M+2) & a_{M,M-1}(n-L-M+2) & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & a_{M,M}(n-L-M+1) & \cdots & a_{M,1}(n-L-M+1) & 1 \end{bmatrix} \tag{10}$$

$$\boldsymbol{L}_2(n) = \begin{bmatrix} 1 & a_{M,1}(n) & \cdots & a_{M,M}(n) & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & a_{M,M-1}(n-1) & a_{M,M}(n-1) & \cdots & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & a_{M,1}(n-L+2) & \cdots & a_{M,M}(n-L+2) & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1 & \cdots & a_{M,M-1}(n-L+1) & a_{M,M}(n-L+1) \end{bmatrix}. \tag{11}$$

III. Proposed Block-Exact Fast LMS/Newton Algorithm

For simplicity, let us assume that the block length N is a factor of the filter length, i.e., $L = NP$. The following equations can be obtained by iterating (7) and (8) for N consecutive time steps:

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

We can thus rewrite the error sequence of the fast LMS/Newton algorithm at time intervals $n-N+1, n-N+2, \ldots, n-1, n$ more compactly in matrix form as

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

where $\underline{e}(n) = [e(n-N+1), e(n-N+2), \ldots, e(n)]^T$ and $\underline{d}(n) = [d(n-N-M+1), d(n-N-M+2), \ldots, d(n-M)]^T$ are both $N \times 1$ vectors and

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

is an N x L matrix, and

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

$$s_i(n) = 2\mu\boldsymbol{X}^T(n-M)\boldsymbol{u}_a(n-i), \quad i = 1, 2, \ldots, N-1. \tag{13}$$

Note, in the FELMS algorithm, the input vector $\boldsymbol{X}(n)$ will be used instead of the whitened input $\boldsymbol{u}_a(n)$ shown above. Fortunately, both of them satisfy the shifting property, and hence most of the techniques in deriving the FELMS are still applicable, after appropriate modifications. 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 (8) and combining properly the resulting equations for all time steps within the current block, as follows:

$$\boldsymbol{W}(n+1) = \boldsymbol{W}(n-N+1) + 2\mu\boldsymbol{U}_a(n)\underline{e}(n) \tag{14}$$

where $\boldsymbol{U}_a(n) = [\boldsymbol{u}_a(n-N+1), \boldsymbol{u}_a(n-N+2), \ldots, \boldsymbol{u}_a(n)]$ is a $L \times N$ matrix. Further, $\underline{e}(n)$ on both sides of (12) can be combined to give

$$\begin{aligned}
[\boldsymbol{S}(n) + \boldsymbol{I}]\underline{e}(n) &= \underline{d}(n) - \underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1) \\
\underline{e}(n) &= [\boldsymbol{S}(n) + \boldsymbol{I}]^{-1}[\underline{d}(n) - \underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1)] \\
&= \boldsymbol{G}(n)[\underline{d}(n) - \underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1)].
\end{aligned} \tag{15}$$

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

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

where $\boldsymbol{G}_i(n)$, as shown in (16), has nonzero elements only appearing at the $(i+1)$th row in the lower triangular part (see (16), shown at the bottom of the page). Apparently, the second term on the right-hand side of (15) represents a fixed coefficient filtering of the input within a block of length N. This characteristic can be utilized to reduce the computational complexity. Toward this end, we first rewrite it as

$$\underline{\boldsymbol{X}}(n-M)\boldsymbol{W}(n-N+1) = \boldsymbol{A}_{N\times N}(n)\boldsymbol{W}_N(n-N+1) \tag{17}$$

where

$$\boldsymbol{A}_{N\times N}(n) = \begin{bmatrix} \boldsymbol{A}_{N-1}(n) & \boldsymbol{A}_N(n) & \cdots & \boldsymbol{A}_{2N-3}(n) & \boldsymbol{A}_{2N-2}(n) \\ \boldsymbol{A}_{N-2}(n) & \boldsymbol{A}_{N-1}(n) & & & \boldsymbol{A}_{2N-3}(n) \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ \boldsymbol{A}_1(n) & \vdots & & & \boldsymbol{A}_N(n) \\ \boldsymbol{A}_0(n) & \boldsymbol{A}_1(n) & \cdots & \boldsymbol{A}_{N-2}(n) & \boldsymbol{A}_{N-1}(n) \end{bmatrix} \tag{18}$$

is a block-Toeplitz matrix whose block element

$$\begin{aligned}
\boldsymbol{A}_j(n) = [&x(n-M-j), x(n-M-N-j), \ldots, \\
&x(n-M-Ni-j), \ldots, x(n-M-L+N-j)] \\
&\text{for } j = 0, 1, \ldots, 2N-2; \quad i = 0, 1, \ldots, (L/N)-1
\end{aligned} \tag{19}$$

$$\boldsymbol{G}_i(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & & \vdots & \vdots \\ \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & & 0 & \vdots \\ -s_i(n-N+i+1) & -s_{i-1}(n-N+i+1) & \cdots\ -s_1(n-N+i+1) & 1 & \vdots \\ 0 & 0 & 0 & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & & & \vdots & 0 \\ 0 & 0 & \cdots\ 0 & 0\ \cdots\ 0 & 1 \end{bmatrix}. \tag{16}$$

is a $1 \times (L/N)$ vector, and

$$\begin{aligned}\boldsymbol{W}_N(n-N+1) &= \Big[\boldsymbol{w}_0^T(n-N+1), \boldsymbol{w}_1^T(n-N+1), \ldots, \\ &\qquad \boldsymbol{w}_{N-2}^T(n-N+1), \boldsymbol{w}_{N-1}^T(n-N+1)\Big]^T \\ \boldsymbol{w}_k(n-N+1) &= [w_k, w_{k+N}, \ldots, w_{k+Ni}, \ldots, w_{k+L-N}]^T\ (n-N+1), \\ &\quad \text{for } k=0,1,\ldots,N-1; \quad i=0,1,\ldots,(L/N)-1\end{aligned}$$

is an $(L/N) \times 1$ vector.

Likewise, $\boldsymbol{W}(n)$ and $\boldsymbol{U}_a(n)$ in (14) can be reorganized in a similar way. By defining

$$\begin{aligned}\boldsymbol{B}_j(n) = [u_a(n-j), u_a(n-N-j), \ldots, u_a(n-Ni-j), \ldots, \\ u_a(n-L+N-j)] \quad (20)\end{aligned}$$

for $j=0,1,\ldots,2N-2; i=0,1,\ldots,(L/N)-1; \boldsymbol{U}_a(n)$ can be partitioned to form

$$\boldsymbol{B}_{N\times N}(n) = \begin{bmatrix} \boldsymbol{B}_{N-1}(n) & \boldsymbol{B}_N(n) & \cdots & \boldsymbol{B}_{2N-3}(n) & \boldsymbol{B}_{2N-2}(n) \\ \boldsymbol{B}_{N-2}(n) & \boldsymbol{B}_{N-1}(n) & & & \boldsymbol{B}_{2N-3}(n) \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ \boldsymbol{B}_1(n) & \vdots & & & \boldsymbol{B}_N(n) \\ \boldsymbol{B}_0(n) & \boldsymbol{B}_1(n) & \cdots & \boldsymbol{B}_{N-2}(n) & \boldsymbol{B}_{N-1}(n) \end{bmatrix}. \quad (21)$$

The update of $\boldsymbol{G}(n)$ in (15) can be efficiently implemented by utilizing the relationship between successive calculations. More precisely, at the current update, the elements in the first column can be expressed respectively in terms of those in the last row at the previous update, as shown in (22) at the bottom of the page. The remaining elements along each subdiagonal can be updated one after another using the results obtained in (22), as follows:

$$\begin{aligned}s_i(n+1) = s_i(n) + 2\mu[x(n-M+1)u_a(n-i+1) \\ -x(n-M-L+1)u_a(n-i-L+1)] \quad (23)\end{aligned}$$

There are altogether $N(N-1)/2$ such elements to be updated. Since in (22) and (23) the second summation items in the brackets have been obtained at time $n-L$, only a total of N^2-N-1 multiplications and $(3/2)N(N-1)$ additions are required. Summarizing, we can obtain the following equivalent block-exact version of the fast LMS/Newton algorithm:

$$\underline{e}(n) = \boldsymbol{G}(n)[\underline{d}(n) - \boldsymbol{A}_{\boldsymbol{N}\times\boldsymbol{N}}(n)\boldsymbol{W}_N(n-N+1)] \quad (24a)$$

$$\boldsymbol{W}_N(n+1) = \boldsymbol{W}_N(n-N+1) + 2\mu\boldsymbol{B}_{N\times N}^T(n)\underline{e}(n). \quad (24b)$$

Similar to the FELMS algorithm in [8], the matrix multiplications of $\boldsymbol{A}_i(n)$ and $\boldsymbol{B}_i(n)$ in (24) can be implemented using the fast convolution algorithms in [14] and [15] or FFT. Here, we give an example with $N=2$ for the purpose of illustration.

Rewrite the second items on the right-hand side of (24a) and (24b) into (25a) and (25b) as follows:

$$\begin{aligned}\begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2 \\ \boldsymbol{A}_0 & \boldsymbol{A}_1\end{bmatrix}\begin{bmatrix}\boldsymbol{w}_0 \\ \boldsymbol{w}_1\end{bmatrix}(n-1) &= \begin{bmatrix}\boldsymbol{A}_1(\boldsymbol{w}_0+\boldsymbol{w}_1) + (\boldsymbol{A}_2-\boldsymbol{A}_1)\boldsymbol{w}_1 \\ \boldsymbol{A}_1(\boldsymbol{w}_0+\boldsymbol{w}_1) - (\boldsymbol{A}_1-\boldsymbol{A}_0)\boldsymbol{w}_0\end{bmatrix}(n-1) \quad (25a) \\ \begin{bmatrix}\boldsymbol{B}_1^T & \boldsymbol{B}_0^T \\ \boldsymbol{B}_2^T & \boldsymbol{B}_1^T\end{bmatrix}\begin{bmatrix}e(n-1) \\ e(n)\end{bmatrix} &= \begin{bmatrix}\boldsymbol{B}_1^T(e(n-1)+e(n)) - (\boldsymbol{B}_1-\boldsymbol{B}_0)^Te(n) \\ \boldsymbol{B}_1^T(e(n-1)+e(n)) + (\boldsymbol{B}_2-\boldsymbol{B}_1)^Te(n-1)\end{bmatrix}. \quad (25b)\end{aligned}$$

According to the definitions of (19) and (20), we have

$$\begin{aligned}(\boldsymbol{A}_1-\boldsymbol{A}_0) &= [x_\Delta(n-M-1), x_\Delta(n-M-3), \ldots, \\ &\qquad x_\Delta(n-M-L+1)] \quad (26) \\ (\boldsymbol{B}_1-\boldsymbol{B}_0) &= [u_{a\Delta}(n-1), u_{a\Delta}(n-3), \ldots, \\ &\qquad u_{a\Delta}(n-L+1)] \quad (27)\end{aligned}$$

where $x_\Delta(n) = x(n) - x(n+1)$ and $u_{a\Delta}(n) = u_a(n) - u_a(n+1)$.

It can be easily verified that only the first items $x_\Delta(n-M-1)$ in (26) and $u_{a\Delta}(n-1)$ in (27) 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 update step. Similar computational savings also apply to $(\boldsymbol{A}_2-\boldsymbol{A}_1)$ and $(\boldsymbol{B}_2-\boldsymbol{B}_1)$. This property can be generalized to other cases where different block lengths N are selected. In these cases, only the first items of $(\boldsymbol{A}_k-\boldsymbol{A}_{k-1})$ and $(\boldsymbol{B}_k-\boldsymbol{B}_{k-1}), k=2N-2, 2N-3, \ldots, 2, 1$ are required to be updated for each iteration.

When $N=2$, $\boldsymbol{S}(n)$ only comprises of one element, its update is

$$\begin{aligned}s_1(n) = s_1(n-2) + 2\mu[&x(n-M)u_a(n-1) \\ &+ x(n-M-1)u_a(n-2) \\ &- x(n-M-L)u_a(n-L-1) \\ &- x(n-M-L-1)u_a(n-L-2)].\end{aligned}$$

The block-exact fast LMS/Newton algorithm with a block length of $N=2$ described by (24) and (25) requires $3L+3$ multiplications and $4L+10$ additions plus another $12M$ operations for updating $\boldsymbol{u}_a$. Compared with $4L$ multiplications and $4L$ additions plus the $12M$ operations cost for updating $\boldsymbol{u}_a$ needed by the conventional algorithm, the block-exact scheme reduces 25% of the total number of multiplications with only a slight increase in additions. For the special case $N=2^n$, the FFT can be employed, and the computational complexity of the proposed algorithm is

$$\begin{aligned}&2(3/2)^nL + 2^n(3\cdot 2^n - 3)/2 + 2^n\cdot 6M \quad \text{multiplications} \\ &2(2(3/2)^n - 1)L + 2^{n+2}(2^{n-1}-1) \\ &\qquad +4\cdot 3^n - 2 + 2^n\cdot 6M \quad \text{additions.}\end{aligned}$$

$$\begin{aligned}s_i(n-N+i+1) = s_i(n-N) + 2\mu\Bigg[&\sum_{j=0}^{i} x(n-M-N+i-j+1)u_a(n-N-j+1) \\ &- \sum_{j=0}^{i} x(n-M-L-N+i-j+1)u_a(n-L-N-j+1)\Bigg], \\ &\text{for } i=1,\ldots,N-1. \quad (22)\end{aligned}$$

TABLE I
COMPUTATIONAL COMPLEXITY COMPARISON OF THE PROPOSED ALGORITHM AND ITS NONBLOCK COUNTERPART

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	394	280
128	16	5	4576	4576	3586	2136
512	32	5	33728	33728	18378	10224
1024	64	5	132992	132992	57378	31296

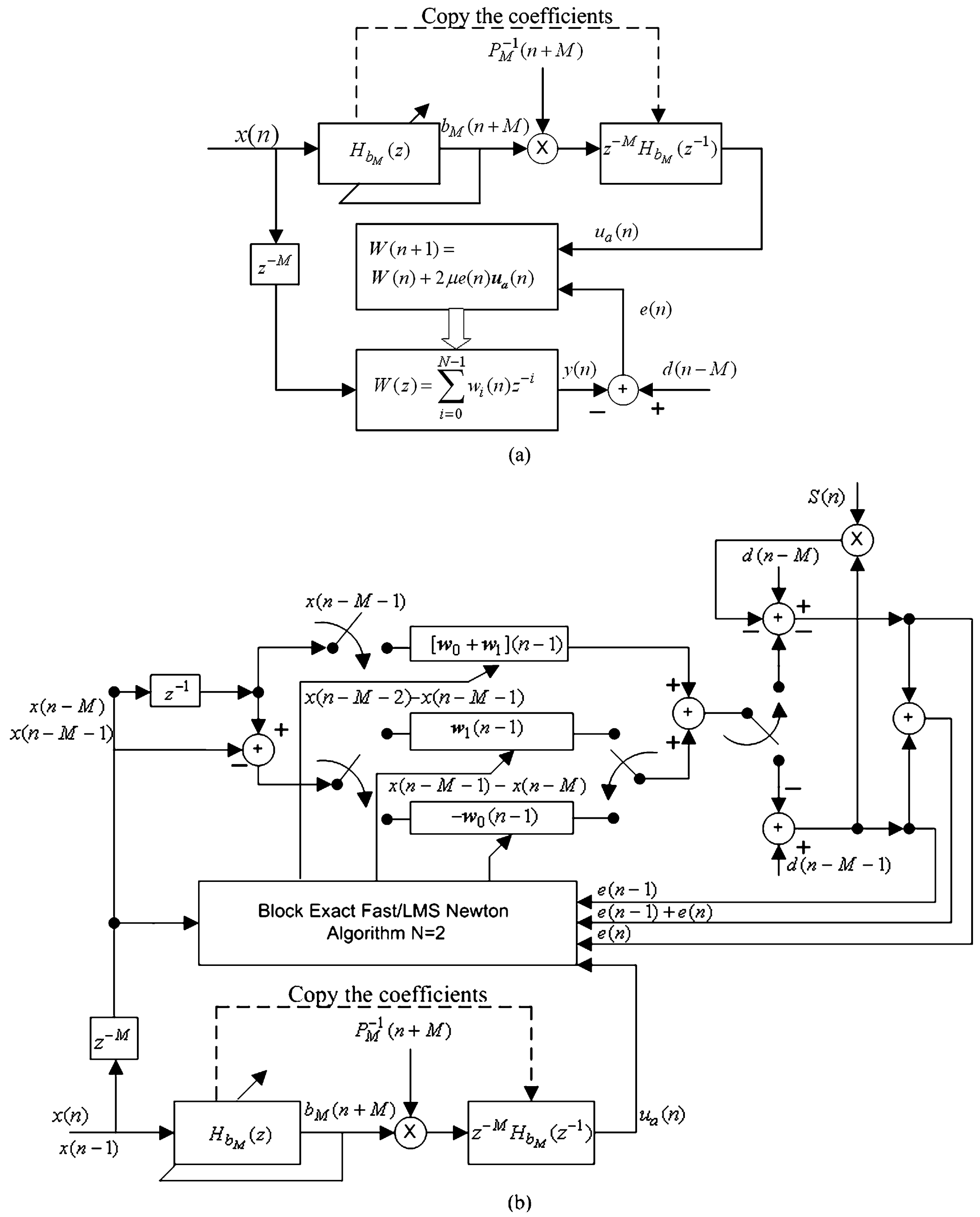

Fig. 2. Implementation block diagrams for (a) fast LMS/Newton algorithm and (b) block-exact fast LMS/Newton algorithm.

The computational complexities of the proposed algorithm and its nonblock counterpart for different block length N and filter length L are summarized in Table I. It can be seen that significant savings in arithmetic operation is achieved by the proposed block-exact algorithm. A closer look at (24) also reveals that except for the additional cost for updating $\boldsymbol{u}_a$ (i.e., whitening the input using a low-order AR process, which is quite small compared with the rest), the computational complexity of the proposed algorithm is rather close to that of the FELMS algorithm in [8]. Two main differences are: first, in addition to calculating $(\boldsymbol{A}_k - \boldsymbol{A}_{k-1})$, the vector subtraction $(\boldsymbol{B}_k - \boldsymbol{B}_{k-1})$ in

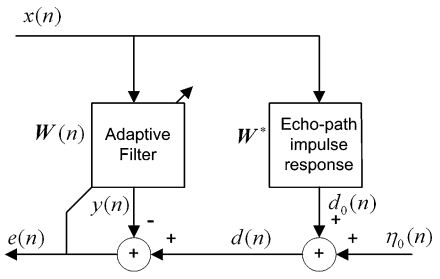

Fig. 3. Adaptive echo cancellation.

(24) has to be computed, whereas this is unnecessary in the FELMS because it solely deals with $x(n)$. Fortunately, due to the computational saving in updating $(\boldsymbol{B}_k - \boldsymbol{B}_{k-1})$, only $2N - 2$ extra additions are needed for each iteration. The second difference lies in the updating of $\boldsymbol{S}(n)$ in (22) and (23). Due to the presence of $\boldsymbol{u}_a(n)$, the "one multiplication" saved in FELMS cannot be exempted from the proposed algorithm. This results in $N - 1$ more multiplications in each iteration for the proposed algorithm. As mentioned previously, our work is based on the fast LMS/Newton algorithm 2 in [4], which is simpler in structure and thus more suitable for hardware implementation. A block diagram depicting this algorithm is reprinted from [4] in Fig. 2(a). Similarly, it is found in [8] that the block-exact technique we employ also has an efficient implementation structure. This structure is very regular, which is attractive for hardware or very large scale integration (VLSI) implementation. Based on these structures, the block diagram for implementing the proposed algorithm with block length $N = 2$ is derived and plotted in Fig. 2(b). It is worth noticing that, because of the similarity between the FELMS algorithm and our proposed algorithm, many properties of the former is also applicable to the latter. For example, like FELMS, the proposed algorithm requires fewer operations than its nonblock counterpart as long as $P \geq 2 \cdot (P = L/N)$. However, if N is of the same order of magnitude as L, the proposed algorithm will require more operations. By compromising the update of $\boldsymbol{S}(n)$ and the utilization of the fast FIR technique, an "optimum" value n_{opt} for the proposed algorithm can also be approximated. The block length N can thus be chosen with the same order as n_{opt}. Since the generalization of such relevant conclusions in [8] to the proposed algorithm is rather straightforward, and also due to the space limitation, these issues are not elaborated further in this paper.

IV. Simulation Results

We now verify the efficiency and the mathematical equivalence of the proposed block-exact fast LMS/Newton algorithm with its nonblock counterpart using computer simulations of an acoustic echo cancellation problem. The system model is depicted in Fig. 3. The input signal $x(n)$ is modeled as a speech signal using an AR process with coefficients $[1\ -0.65\ 0.693\ -0.22\ 0.309\ -0.177]$ as given in [4]. The echo path impulse response, which is a realistic one and has a length of 128, is given as $m_1(k)$ by the ITU-T recommendation G.168 [17]. The background noise $\eta_0(n)$ is a white Gaussian random sequence with variance $\delta^2_{\eta_0}(n) = 0.0001$. For simplicity, no double-talk is assumed to present and the block length is $N = 4$. As in [4], we compare the proposed algorithm and its nonblock counterpart with the normalized LMS (NLMS) algorithm [1]. The step sizes of the various algorithms were appropriately chosen so that they have identical steady-state MSE. The results are averaged over 100 independent runs. From Fig. 4, it can be seen that the MSE curves of the proposed algorithm and its original counterpart are identical, which substantiates their mathematical equivalence. Fig. 5 shows the residual echoes obtained by the various algorithms. These simulation results comply well with those obtained in [4]. Regarding the performance analysis of the FNTF and related Newton algorithms, readers are referred to [18], [19], and [4] for more information.

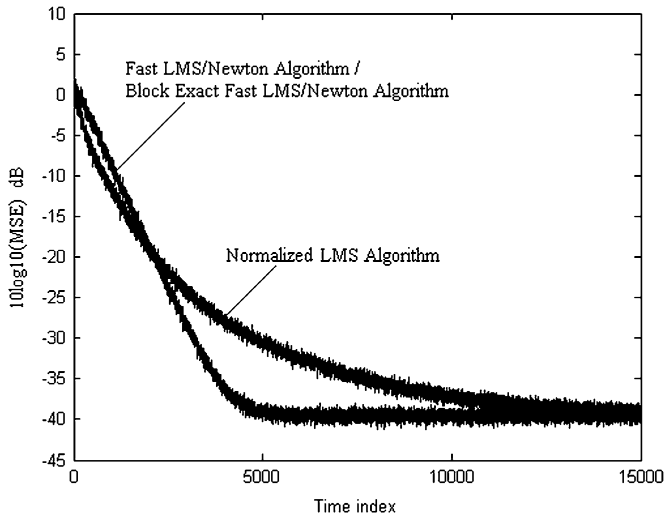

Fig. 4. MSE results versus time n.

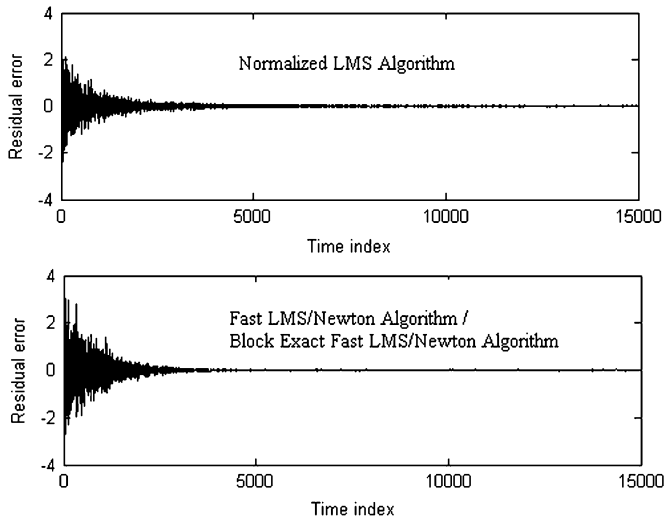

Fig. 5. Residual errors versus time n.

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 computational 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. Together with the good numerical stability, the proposed algorithm may be a good alternative to the block-exact FNTF algorithm [9] in applications where long adaptive filters are required.

References

[1] S. Haykin, *Adaptive Filter Theory*, 4th ed. Englewood Cliffs, NJ: Prentice-Hall, 2001.

[2] J. G. Proakis, C. M. Rader, F. Ling, C. L. Nikias, M. Moonen, and I. K. Proudler, *Algorithms for Statistical Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[3] G. V. Moustakides and S. Theodoridis, "Fast newton transversal filters-a new class of adaptive estimation algorithms," *IEEE Trans. Signal Process.*, 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. Signal Process.*, vol. 45, no. 8, pp. 1987–2000, Aug. 1997.

[5] J. Benesty, T. Gansler, D. R. Morgan, M. M. Sondhi, and S. L. Gay, *Advances in Network and Acoustic Echo Cancellation*. New York: Springer-Verlag, 2001.

[6] G. A. Clark, S. K. Mitra, and S. R. Parker, "Block implementation of adaptive digital filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-29, no. 3, pp. 744–752, Jun. 1981.

[7] S. Narayan, A. M. Peterson, and M. J. Narasimha, "Transform domain LMS algorithm," *IEEE Trans. Acoust. Speech, Signal Process.*, vol. ASSP-31, no. 3, pp. 609–615, Jun. 1983.

[8] J. Benesty and P. Duhamel, "A fast exact least mean square adaptive algorithm," *IEEE Trans. Signal Process.*, vol. 40, no. 12, pp. 2904–2920, Dec. 1992.

[9] K. Berberidis and S. Theodoridis, "A new fast block adaptive algorithm," *IEEE Trans. Signal Process.*, vol. 47, no. 1, pp. 75–87, Jan. 1999.

[10] D. T. M. Slock and K. Maouche, "The fast subsampled-updating recursive least-square (FSU-RLS) algorithm for adaptive filtering based on displacement structure and FFT," *Signal Process.*, vol. 40, pp. 5–20, Oct. 1994.

[11] M. Tanaka, S. Makino, and J. Kojima, "A block exact fast affine projection algorithm," *IEEE Trans. Speech Audio Process.*, vol. 7, no. 1, pp. 79–86, Jan. 1999.

[12] G. Rombouts and M. Moonen, "A sparse block exact affine projection algorithm," *IEEE Trans. Speech Audio Process.*, vol. 10, no. 2, pp. 100–108, Feb. 2002.

[13] F. Albu and H. K. Kwan, "Fast block exact Gauss–Seidel pseudo affine projection algorithm," *IEE Electron. Lett.*, vol. 40, no. 22, pp. 1451–1453, Oct. 2004.

[14] Z. J. Mou and P. Duhamel, "Fast FIR filtering: Algorithms and implementation," *Signal Process.*, vol. 377–384, Dec. 1987.

[15] Z. J. Mou and P. Duhamel, "Short-length FIR filters and their use in fast nonrecursive filtering," *IEEE Trans. Signal Process.*, vol. 39, no. 6, pp. 1322–1332, Jul. 1991.

[16] Y. Zhou, S. C. Chan, and K. L. Ho, "A new block exact fast LMS/newton adaptive filtering algorithm," in *Proc. IEEE 2004 47th Midwest Symp. Circuits Systems*, Hiroshima, Japan, Jul. 25–28, 2004, pp. II-29–II-32.

[17] *Digital Network Echo Cancellers*, ITU-T Recommendation G.168, 2000.

[18] K. Ikeda, S. Tanaka, and Y. Wang, "Convergence rate analysis of fast predictor-based least squares algorithm," *IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process.*, vol. 49, no. 1, pp. 11–15, Jan. 2002.

[19] Y. Wang, K. Ikeda, and K. Nakayama, "A numerically stable fast Newton-type adaptive filter based on order recursive least squares algorithm," *IEEE Trans. Signal Process.*, vol. 51, no. 9, pp. 2357–2368, Sep. 2003.

On the Spectral Factor Ambiguity of FIR Energy Compaction Filter Banks

Andre Tkacenko and P. P. Vaidyanathan

***Abstract*—This paper focuses on the design of signal-adapted finite-impulse response (FIR) paraunitary (PU) filter banks optimized for energy compaction (EC). The design of such filter banks has been shown in the literature to consist of the design of an optimal FIR compaction filter followed by an appropriate Karhunen–Loève transform (KLT). Despite this elegant construction, EC optimal filter banks have been shown to perform worse than common nonadapted filter banks for coding gain, contrary to intuition. Here, it is shown that this phenomenon is most likely due to the nonuniqueness of the compaction filter in terms of its spectral factors. This nonuniqueness results in a finite set of EC optimal filter banks. By choosing the spectral factor yielding the largest coding gain, it is shown that the resulting filter bank behaves more and more like the infinite-order principal components filter bank (PCFB) in terms of numerous objectives such as coding gain, multiresolution, noise reduction with zeroth-order Wiener filters in the subbands, and power minimization for discrete multitone (DMT)-type nonredundant transmultiplexers.**

***Index Terms*—Compaction filter, energy compaction, multirate filter bank, principal components filter bank.**

I. Introduction

A signal-adapted filter bank is any multirate filter bank whose filters depend on the nature or statistics of its input. The problem of the design of optimal signal-adapted multirate filter banks has been of interest to the signal processing community on account of its applications in data compression, signal denoising, and digital communications [11], [15], [16]. A typical model for the filter bank used is the M-channel maximally decimated filter bank [14] shown in Fig. 1(a). Here, the subband processors $\{\mathcal{P}_k\}$ need not be linear and are typically scalar quantizers, constant multipliers, or threshold devices.

An equivalent polyphase representation [14] of the filter bank of Fig. 1(a) is shown in Fig. 1(b). The analysis filters $\{H_k(z)\}$ and synthesis filters $\{F_k(z)\}$ are, respectively, related to the analysis polyphase matrix $\mathbf{H}(z)$ and synthesis polyphase matrix $\mathbf{F}(z)$ as follows [14]:

$$\begin{aligned}[H_0(z) \quad H_1(z) \quad \cdots \quad H_{M-1}(z)]^T &= \mathbf{H}(z^M)\mathbf{a}(z)\\ [F_0(z) \quad F_1(z) \quad \cdots \quad F_{M-1}(z)] &= \widetilde{\mathbf{a}}(z)\mathbf{F}(z^M).\end{aligned} \tag{1}$$

Here, $\mathbf{a}(z)$ denotes the $M \times 1$ *advance chain* vector given by

$$\mathbf{a}(z) = [1 \quad z \quad \cdots \quad z^{M-1}]^T$$

and the tilde notation denotes the *paraconjugate* [14] of any system (i.e., $\widetilde{\mathbf{A}}(z) \triangleq \mathbf{A}^\dagger(1/z^*)$ for any $\mathbf{A}(z)$).

From here on, we will assume that the M-fold blocked signal vector $\mathbf{x}(n)$ from Fig. 1(b) is wide sense stationary (WSS) with a known

Manuscript received August 6, 2004; revised January 19, 2005. This work was supported in part by the NSF grant CCF-0428326, ONR grant N00014-06-1-0011, and the California Institute of Technology. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Trac D. Tran.

A. Tkacenko was with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA. He is now with the Digital Signal Processing Research Group, Jet Propulsion Laboratory, Pasadena, CA 91109 USA (e-mail: andre@systems.caltech.edu).

P. P. Vaidyanathan is with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: ppvnath@systems.caltech.edu).

Digital Object Identifier 10.1109/TSP.2005.861060