A Block Exact Fast Affine Projection Algorithm

Masashi Tanaka, Shoji Makino, *Member, IEEE,* and Junji Kojima

***Abstract*— This paper describes a block (affine) projection algorithm that has exactly the same convergence rate as the original sample-by-sample algorithm and smaller computational complexity than the fast affine projection algorithm. This is achieved by 1) introducing a correction term that compensates for the filter output difference between the sample-by-sample projection algorithm and the straightforward block projection algorithm, and 2) applying a fast finite impulse response (FIR) filtering technique to compute filter outputs and to update the filter.**

We describe how to choose a pair of block lengths that gives the longest filter length under a constraint on the total computational complexity and processing delay. An example shows that the filter length can be doubled if a delay of a few hundred samples is permissible.

***Index Terms*—Adaptive filtering, affine projection, block exact.**

I. Introduction

ADAPTIVE filtering is a key technique in applications where the unknown system is time-varying. Among many adaptive filter algorithms, the least-mean-square (LMS) or the normalized LMS (NLMS) algorithms have been used because of their relatively small computational complexity of $2L$, where L is the filter length. The shortcoming of the LMS algorithm is that it has slow convergence rate for colored input signal such as speech. On the other hand, the affine projection algorithm [1], compared with the LMS algorithm, has faster convergence rate for colored input signal but has more computational complexity. The (computationally) fast affine projection algorithm [2]–[5] and the fast Newton transversal filters [6], which have a better convergence rate for colored input signals, have almost the same computational complexity as the LMS. Although their complexity of $2L$ is smaller than those of other adaptive filtering algorithms such as the fast recursive least squares (fast RLS) [7] ($7L$), $2L$ is still large in some applications like acoustic echo cancellation, where the filter length is several hundred and sometimes several thousand. Therefore it is desirable to further reduce computational complexity.

Block processing is an effective approach to reduce the computational complexity and is employed in the LMS and the affine projection algorithms [8]–[10]. However, these algorithms have slower convergence rate because of less frequent updating of the adaptive filter. In recent years, "fast exact" block algorithms have been proposed for the LMS [11], for the fast Newton transversal filters [12], and for the fast RLS [13]. The fast exact block algorithms achieve less computational complexity by using fast FIR filtering techniques [14], [15], and they have exactly the same convergence rate as corresponding sample-by-sample algorithms.

Manuscript received June 27, 1996; revised February 5, 1998. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Dennis R. Morgan.

The authors are with Nippon Telegraph and Telephone (NTT) Human Interface Laboratories, Tokyo, 180-8585 Japan (e-mail: tanaka@splab.hil.ntt.co.jp).

Publisher Item Identifier S 1063-6676(99)00177-7.

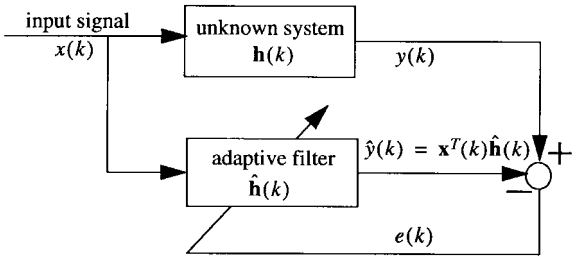


Fig. 1. Structure of adaptive filter.

This paper describes a fast exact version of the block projection algorithm. The proposed algorithm introduces two block lengths: one for computing the adaptive filter output and the other for updating the adaptive filter. Section II explains the projection algorithm and fast FIR filtering techniques, Section III derives the proposed fast exact block projection algorithm, and Section IV shows the relationship between the output delay and the adaptive filter length.

II. Preliminaries

This section gives a brief explanation of the projection algorithm and fast FIR filtering (FFF) techniques. In the following, nonboldfaced letters are used for scalars, boldfaced small letters are used for vectors, and boldfaced capital letters are used for matrices.

A. Projection Algorithm

First we define some notation using the block diagram of an adaptive filter shown in Fig. 1. The unknown system is assumed to be modeled as an FIR filter whose coefficients are written as a vector $\mathbf{h} = (h_1, h_2, \cdots, h_L)^T$, where L is the number of filter coefficients and T is the transpose. The unknown system is estimated by an adaptive FIR filter with coefficients $\hat{\mathbf{h}}(k) = [\hat{h}_1(k), \hat{h}_2(k), \cdots, \hat{h}_L(k)]^T$, where k is the discrete time index. The $x(k)$, $y(k)$, $\hat{y}(k)$, and $e(k)$ in Fig. 1 are the input signal to the unknown system, the output signal from the unknown system, the filter output, and the error signal.

In Fig. 1, the filter output $\hat{y}(k)$ is calculated by

$$\hat{y}(k) = \mathbf{x}^T(k)\hat{\mathbf{h}}(k) \tag{1}$$

where

$$\mathbf{x}(k) = [x(k), x(k-1), \cdots, x(k-L+1)]^T. \quad (2)$$

The adaptive filter $\hat{\mathbf{h}}(k)$ is adjusted to make the filter output $\hat{y}(k)$ close to the unknown system output $y(k)$ by adding an adjustment vector $\Delta\hat{\mathbf{h}}(k)$.

$$\hat{\mathbf{h}}(k+1) = \hat{\mathbf{h}}(k) + \mu(k)\Delta\hat{\mathbf{h}}(k) \quad (3)$$

where μ is a scaling factor called the step size, which controls the convergence rate and the amount of the residual error. We consider μ to be time-variant in the general case, although it sometimes takes a constant value.

In the projection algorithm of order $p(\leq L)$, the adjustment vector $\Delta\hat{\mathbf{h}}(k)$ is the minimum-norm solution to the following simultaneous equations.

$$\begin{aligned} y(k) &= \mathbf{x}^T(k)[\hat{\mathbf{h}}(k) + \Delta\hat{\mathbf{h}}(k)] \\ y(k-1) &= \mathbf{x}^T(k-1)[\hat{\mathbf{h}}(k) + \Delta\hat{\mathbf{h}}(k)] \\ &\vdots \\ y(k-p+1) &= \mathbf{x}^T(k-p+1)[\hat{\mathbf{h}}(k) + \Delta\hat{\mathbf{h}}(k)]. \end{aligned} \quad (4)$$

The adjustment vector $\Delta\hat{\mathbf{h}}(k)$ is given by

$$\Delta\hat{\mathbf{h}}(k) = \mathbf{X}(k)\mathbf{g}(k) \quad (5)$$

where

$$\mathbf{X}(k) = [\mathbf{x}(k), \mathbf{x}(k-1), \cdots, \mathbf{x}(k-p+1)] \quad (6)$$

$$\mathbf{g}(k) = \mathbf{R}^{-1}(k)\mathbf{e}(k), \quad (7)$$

$$\mathbf{R}(k) = \mathbf{X}^T(k)\mathbf{X}(k), \quad (8)$$

$$\mathbf{e}(k) = \mathbf{y}(k) - \mathbf{X}^T(k)\hat{\mathbf{h}}(k), \quad (9)$$

$$\mathbf{y}(k) = [y(k), y(k-1), \cdots, y(k-p+1)]^T. \quad (10)$$

We call $\mathbf{g}(k)$ a prefilter vector because it filters the row vectors of $\mathbf{X}(k)$ to synthesize the adjustment vector.

The straightforward block projection algorithm is executed every N samples in the following order. ($N = 1$ corresponds to the sample-by-sample algorithm.)

$$\mathbf{e}_N(k) = \mathbf{y}_N(k) - \mathbf{X}_{L,N}^T(k)\hat{\mathbf{h}}(k') \quad (11)$$

$$\begin{aligned} \mathbf{g}(k-ip) &= \mathbf{R}^{-1}(k-ip)\mathbf{e}(k-ip), \\ &\qquad i = 0, 1, \cdots, \lfloor (N-1)/p \rfloor \end{aligned} \quad (12)$$

$$\begin{aligned} \hat{\mathbf{h}}(k+1) &= \hat{\mathbf{h}}(k') + \sum_{i=0}^{\lfloor (N-1)/p \rfloor} \mu(k-ip)\mathbf{X}(k-ip)\mathbf{g}(k-ip) \\ &= \hat{\mathbf{h}}(k') + \mathbf{X}_{L,p[\lfloor (N-1)/p \rfloor + 1]}(k)\mathbf{s}(k) \end{aligned} \quad (13)$$

where

$$k' = k - N + 1 \quad (14)$$

$$\mathbf{y}_N(k) = [y(k), y(k-1), \cdots, y(k-N+1)]^T, \quad (15)$$

$$\mathbf{X}_{L,N}(k) = [\mathbf{x}(k), \mathbf{x}(k-1), \cdots, \mathbf{x}(k-N+1)], \quad (16)$$

$$\begin{aligned} \mathbf{s}(k) = [&\mu(k)\mathbf{g}^T(k), \mu(k-1)\mathbf{g}^T(k-1), \cdots, \\ &\mu(k-p\lfloor (N-1)/p \rfloor)\mathbf{g}^T(k-p\lfloor (N-1)/p \rfloor)]^T. \end{aligned} \quad (17)$$

and $\lfloor x \rfloor$ is the floor function meaning the greatest integer not greater than x. We see two filterings with fixed coefficients: $\mathbf{X}_{L,p[\lfloor (N-1)/p \rfloor + 1]}(k)\mathbf{s}(k)$ in (13) and $\mathbf{X}_{L,N}^T(k)\hat{\mathbf{h}}(k')$ in (11). Although filtering with an FFF technique reduces the computational complexity, the straightforward block processing has less frequent updating of filter coefficients and results in slower convergence than the sample-by-sample algorithm.

B. Fast FIR Filtering (FFF) Techniques

This subsection gives an overview of fast FIR filtering or linear convolution algorithms based on a fast short convolution [14], [15] and cyclic convolution methods, and also evaluates their computational complexity. Here, we consider only FIR filtering, which can be expressed as a matrix-vector product (MVP) of a square Hankel (or Toeplitz) matrix and a vector because an MVP of a rectangular matrix can always be decomposed into MVP's of minor square matrices.

1) FFF Based on Linear Convolution: In the usual direct way, multiplying a square Hankel matrix consisting of four half-dimension Hankel matrices by a vector consisting of two half-dimension vectors needs four MVP's of half dimension and two vector-vector additions (VVA) as shown in the following:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{Au} + \mathbf{Bv} \\ \mathbf{Bu} + \mathbf{Cv} \end{pmatrix}. \quad (18)$$

On the other hand, according to a fast short convolution method, (18) can be rewritten as

$$\begin{aligned} \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = &\begin{pmatrix} \mathbf{0} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{I} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{C}-\mathbf{B} \\ \mathbf{0} & \mathbf{B} & \mathbf{0} \\ \mathbf{A}-\mathbf{B} & \mathbf{0} & \mathbf{0} \end{pmatrix} \\ &\cdot \begin{pmatrix} \mathbf{u} \\ \mathbf{u}+\mathbf{v} \\ \mathbf{v} \end{pmatrix} \end{aligned} \quad (19)$$

which needs three MVP's of half dimension, three VVA's, and two matrix subtractions. The three sub-MVP's are recursively computed by applying (19) to themselves. Among these operations, the matrix subtractions can be avoided because we consider a data sequence and the matrix subtractions are done when the submatrix appears for the first time. Also, when filtering various data with a fixed coefficient vector, a VVA, i.e., $\mathbf{u} + \mathbf{v}$, is necessary only once at the beginning of the filtering and can then be avoided.

To evaluate the computational complexity of MVP's, we count the number of multiplications, additions, and multiplication-additions because they are the main part of the algorithm and they each consume one clock cycle on most DSP's. We use $\mathrm{MVP}_{\mathrm{FFF}}(N)$ to represent the number of operations to execute the MVP of dimension N by FFF, $\mathrm{MVP}_{\mathrm{direct}}(N)$ to represent those executed by MVP in the direct way, and $\mathrm{VVA}(N)$ to represent those executed by VVA, where we assume the dimension $N = 2^n$. From (19), we get a recursion for $\mathrm{MVP}_{\mathrm{FFF}}(N)$

$$\begin{aligned} \mathrm{MVP}_{\mathrm{FFF}}(N) &= 3\mathrm{MVP}_{\mathrm{FFF}}(N/2) + q\mathrm{VVA}(N/2), \\ &\qquad q = 2 \text{ or } 3. \end{aligned} \quad (20)$$

Here, we neglect the matrix subtractions and $q = 2$ corresponds to the case when the VVA $\mathbf{u}+\mathbf{v}$ can be avoided. From (20), we get

$$\mathrm{MVP}_{\mathrm{FFF}}(N) = 3^{n-n_0}\mathrm{MVP}_{\mathrm{direct}}(N_0) + q\sum_{i=1}^{n-n_0} 3^{i-1}\mathrm{VVA}(N/2^i) \quad (21)$$

where $N_0 = 2^{n_0}$ is the smallest dimension of the sub-matrices and the $\mathrm{MVP}_{\mathrm{direct}}(N_0)$ is done in the direct way. Considering the relationship $\mathrm{VVA}(N) = 2\mathrm{VVA}(N/2)$, (21) is rewritten as

$$\mathrm{MVP}_{\mathrm{FFF}}(N) = 3^{n-n_0}\mathrm{MVP}_{\mathrm{direct}}(N_0) + q(3^{n-n_0} - 2^{n-n_0})\mathrm{VVA}(N_0). \quad (22)$$

Furthermore, by substituting the relations $\mathrm{MVP}_{\mathrm{direct}}(N_0) = N_0^2$ and $\mathrm{VVA}(N_0) = N_0$ into (22), we get an evaluation formula for $\mathrm{MVP}_{\mathrm{FFF}}$ as follows:

$$\mathrm{MVP}_{\mathrm{FFF}}(N) = 3^n(4/3)^{n_0} + q[3^n(2/3)^{n_0} - 2^n]. \quad (23)$$

2) FFF Based on Cyclic Convolution: MVP's can also be computed using circular convolution, such as the fast Fourier transform (FFT). Let $\mathbf{A}$ be an N-by-N Hankel matrix defined as

$$\mathbf{A} = \begin{pmatrix} a_1 & a_2 & \cdots & a_N \\ a_2 & a_3 & \cdots & a_{N+1} \\ \vdots & \vdots & \ddots & \vdots \\ a_N & a_{N+1} & \cdots & a_{2N-1} \end{pmatrix} \quad (24)$$

and $\mathbf{u}$ be an N-element vector. Then the MVP $\mathbf{Au}$ is computed using the FFT in the following way:

$$\tilde{\mathbf{a}} = \mathrm{FFT}([0, a_1, a_2, \cdots, a_{2N-1}]^T), \quad (25)$$

$$\tilde{\mathbf{u}} = \mathrm{FFT}([\mathbf{0}_N, u_N, u_{N-1}, \cdots, u_1]^T), \quad (26)$$

$$\mathbf{Au} = \text{the first } N \text{ elements of } \mathrm{IFFT}(\tilde{\mathbf{a}} \otimes \tilde{\mathbf{u}}) \quad (27)$$

where IFFT() and $\otimes$ denote, respectively, inverse Fourier transform and element-by-element product. In adaptive filtering, the FFT in (25) is computed when the data block appears for the first time and either the FFT in (26) or the IFFT in (27) is computed at every block. Assuming a $2N$-point IFFT operation and an element-by-element product require the computational complexity of $16N\log_2(2N)$ and 4, respectively, $\mathrm{MVP}_{\mathrm{FFF}}$ is evaluated as

$$\mathrm{MVP}_{\mathrm{FFF}}(N) = 16N\log_2(2N) + 4N. \quad (28)$$

We have seen the two FFF techniques based on linear and circular convolution. Comparing their computational complexities using (23) ($q = 2.5$ and $n_0 = 2$) and (28), the technique based on circular convolution, i.e., FFT, is more efficient when the block length is longer than about 1024.

III. Proposed Block Projection Algorithm

This section describes a fast exact block projection algorithm that reduces the computational complexity by using the FFF, and has exactly the same convergence rate as the original sample-by-sample projection algorithm.

A. Correction of the Error from the Sample-By-Sample Algorithm

Straightforward block processing does not have the same convergence rate because it updates less frequently. We propose a method that corrects the error in the filter output from the sample-by-sample algorithm.

If the coefficient vector $\hat{\mathbf{h}}(k)$ is updated at every sample, then $\hat{\mathbf{h}}(k+i)$, $(i \geq 0)$ is written as

$$\begin{aligned}\hat{\mathbf{h}}(k+i) &= \hat{\mathbf{h}}(k) + \sum_{j=1}^{i}\mu(k+i-j)\mathbf{X}(k+i-j)\mathbf{g}(k+i-j) \\ &= \hat{\mathbf{h}}(k) + \sum_{j=1}^{i+p-1} s_j(k+i-1)\mathbf{x}(k+i-j) \\ &\quad - \sum_{j=1}^{p-1} s_j(k-1)\mathbf{x}(k-j) \end{aligned} \quad (29)$$

where $s_j(k)$ represents the step-size weighted sum of the prefilter coefficients for $\mathbf{x}(k-j+1)$ at time k defined as

$$s_j(k) = \sum_{l=1}^{\min(j,p)} \mu(k-j+l)g_l(k-j+l) \quad (30)$$

and the vector

$$\mathbf{s}_{i+p-1}(k+i-1) = [s_1(k+i-1), s_2(k+i-1), \cdots, s_{i+p-1}(k+i-1)]^T \quad (31)$$

is recursively obtained as

$$\mathbf{s}_{i+p-1}(k+i-1) = [0, \mathbf{s}_{i+p-2}^T(k+i-2)]^T + \mu(k+i-1)\cdot[\mathbf{g}^T(k+i-1), 0, \cdots, 0]^T. \quad (32)$$

From (29), the filter output $\hat{y}(k+i)(i \geq 0)$ is written as

$$\begin{aligned}\hat{y}(k+i) &= \mathbf{x}^T(k+i)\hat{\mathbf{h}}(k+i) \\ &= \mathbf{x}^T(k+i)\hat{\mathbf{h}}(k) + \sum_{j=1}^{i+p-1} s_j(k+i-1)\mathbf{x}^T(k+i)\cdot\mathbf{x}(k+i-j) - \sum_{j=1}^{p-1} s_j(k-1)\mathbf{x}^T(k+i)\mathbf{x}(k-j) \\ &= \mathbf{x}^T(k+i)\hat{\mathbf{h}}(k) + \sum_{j=1}^{i+p-1} s_j(k+i-1)r_j(k+i) - \sum_{j=1}^{p-1} s_j(k-1)r_{i+j}(k+i). \end{aligned} \quad (33)$$

Here, the correlation $r_j(k+i)$ defined by

$$r_j(k+i) = \mathbf{x}^T(k+i)\mathbf{x}(k+i-j) \quad (34)$$

can be obtained recursively [11] as

$$\begin{aligned} r_j(k+i) &= r_j(k+i-1) + x(k+i)x(k+i-j) - x(k-L+i)x(k-L+i-j) \\ &\qquad j = 1, 2, \cdots, i+p-2 \end{aligned} \quad (35)$$

$$\begin{aligned} r_{i+p-1}(k+i) &= r_{i+p-1}(k-1) + \rho_{i+p-1}(k+i) - \rho_{i+p-1}(k-L+i) \end{aligned} \quad (36)$$

where

$$\rho_{i+p-1}(k+i)=\sum_{l=0}^{i} x(k+l)x(k+l-i-p+1). \quad (37)$$

The first term on the right side of (33) corresponds to the filter output for straightforward block processing and the second and third terms correct the error from the sample-by-sample algorithm.

B. Introducing a Modified Filter

The third term in (33) can be eliminated by introducing a modified filter vector

$$\mathbf{z}(k)=\hat{\mathbf{h}}(k)-\sum_{j=1}^{p-1} s_j(k-1)\mathbf{x}(k-j) \quad (38)$$

instead of the filter vector $\hat{\mathbf{h}}(k)$ [2], [4]. From (33) and (38), the filter output $\hat{y}(k+i)$ is rewritten as

$$\hat{y}(k+i)=\mathbf{x}^T(k+i)\mathbf{z}(k)+\mathbf{s}_{i+p-1}^T(k+i-1)\mathbf{r}_{i+p-1}(k+i) \quad (39)$$

where

$$\mathbf{r}_j(k)=[r_1(k),\, r_2(k),\, \cdots,\, r_j(k)]^T. \quad (40)$$

For the block length N, (39) ($i=0,\,1,\,\cdots,\,N-1$) is written in matrix form.

$$\hat{\mathbf{y}}_N(k+N-1)=\mathbf{X}_{L,N}^T(k+N-1)\mathbf{z}(k)+\begin{pmatrix}\mathbf{s}_{p+N-2}^T(k+N-2)\mathbf{r}_{p+N-2}(k+N-1)\\ \vdots\\ \mathbf{s}_p^T(k)\mathbf{r}_p(k+1)\\ \mathbf{s}_{p-1}^T(k-1)\mathbf{r}_{p-1}(k)\end{pmatrix}. \quad (41)$$

The updating formula for the vector $\mathbf{z}(k)$ can be derived in the following way. First, in (38), by replacing k by $k+N$, we get

$$\mathbf{z}(k+N)=\hat{\mathbf{h}}(k+N)-\sum_{j=1}^{p-1} s_j(k+N-1)\mathbf{x}(k+N-j). \quad (42)$$

Then, by substituting (38) and (42) into (29) ($i=N$), the updating formula for the vector $\mathbf{z}(k)$ is derived:

$$\mathbf{z}(k+N)=\mathbf{z}(k)+\sum_{j=p}^{p+N-1} s_j(k+N-1)\mathbf{x}(k+N-j). \quad (43)$$

In matrix form:

$$\mathbf{z}(k+N)=\mathbf{z}(k)+\mathbf{X}_{L,N}(k+N-p)\mathbf{s}_{p+N-1}|_{p+N-1}^{p}\cdot(k+N-1) \quad (44)$$

where $\mathbf{s}|_j^i$ denotes a sub-vector consisting of the i- through jth elements of $\mathbf{s}$.

C. Application of FFF to the Projection Algorithm

The following two filterings in the projection algorithm can be computed by the FFF, as follows.

- $\mathbf{X}_{L,N_1}^T(k+N_1-1)\mathbf{z}(k)$ in (41)
- $\mathbf{X}_{L,N_2}(k+N_2-p)\mathbf{s}_{p+N_2-1}|_{p+N_2-1}^{p}(k+N_2-1)$ in (44).

Here, N_1 is the block length for computing the filter output and N_2 is the block length for updating the filter. We assume here that each is a factor of the filter length L, i.e., $L=M_iN_i$, $(i=1,\,2)$ (M_i are positive integers) and that one of the block lengths is a multiple of the other.

The former filtering is executed every N_1 samples and consists of M_1MVP's of dimension N_1 as follows. (For practical implementation of the FFF, see [11] or [14].)

$$\mathbf{X}_{L,N_1}^T(k+N_1-1)\mathbf{z}(k)=\sum_{i=1}^{M_1}\mathbf{X}_{N_1,N_1}^T(k-(i-2)N_1-1)\cdot\mathbf{z}_{N_1}^{(i)}(k) \quad (45)$$

where $\mathbf{X}_{N_1,N_1}$ represents an N_1-by-N_1 matrix and $\mathbf{z}_{N_1}^{(i)}$ represents an N_1-element vector defined, respectively, as

$$\mathbf{X}_{N_1,N_1}(k)=\begin{pmatrix}x(k) & \cdots & x(k-N_1+1)\\ \vdots & \ddots & \vdots\\ x(k-N_1+1) & \cdots & x(k-2N_1+2)\end{pmatrix} \quad (46)$$

$$\mathbf{z}_{N_1}^{(i)}(k)=[z_{(i-1)N_1+1}(k),\, z_{(i-1)N_1+2}(k),\, \cdots,\, z_{(i-1)N_1+N_1}(k)]^T. \quad (47)$$

Computational complexity for the above filtering is evaluated as M_1MVP(N_1). Similarly, the latter filtering executed every $N_2(=L/M_2)$ samples can be rewritten as

$$\mathbf{z}(k+N_2)=\begin{pmatrix}\mathbf{z}_{N_2}^{(1)}(k)\\ \mathbf{z}_{N_2}^{(2)}(k)\\ \vdots\\ \mathbf{z}_{N_2}^{(M_2)}(k)\end{pmatrix}+\begin{pmatrix}\mathbf{X}_{N_2,N_2}(k+N_2-p)\\ \mathbf{X}_{N_2,N_2}(k-p)\\ \vdots\\ \mathbf{X}_{N_2,N_2}(k-L+2N_2-p)\end{pmatrix}\cdot\mathbf{s}_{p+N_2-1}|_{p+N_2-1}^{p}(k+N_2-1). \quad (48)$$

Here, the second term on the right side consists of M_2MVP(N_2).

In addition to the MVP's, the matrix subtractions shown in Section II-B-1 need $N_1\log_2 N_1$ and $N_2\log_2 N_2$ operations. Summing these numbers gives the total computational complexity for the two filterings as $M_1\text{MVP}(N_1)/N_1+\log_2 N_1+M_2\text{MVP}(N_2)/N_2+\log_2 N_2$ per sample.

D. Complexity Comparison with Other Block Exact Algorithms

All of the procedures for the fast exact projection algorithm are listed in Fig. 5. The algorithm consists of three parts: 1) the filtering part that generates the filter output by using the FFF (1 in Fig. 5); 2) the correction part that recursively computes the correction term and adds it to the filter output obtained in the filtering part to make the filter output exactly the same as that of the sample-by-sample projection algorithm (2–7 in Fig. 5); and 3) the filter updating part that updates the filter by

TABLE I
COMPARISON OF THE COMPUTATIONAL COMPLEXITIES PER SAMPLE AMONG VARIOUS ALGORITHMS

Filter length L	Block length N	FAP	BEFAP (proposed)	FELMS	BEFNTF	FSU-RLS
128	16	416	234	226	372	864
512	64	1184	627	615	827	2298
1024	128	2208	1020	1006	1298	3703
2048	256	4256	1672	1657	2102	5987

using the FFF (8 in Fig. 5). Computational complexities per sample for the three parts are evaluated as follows.

- *Filtering Part (1 in Fig. 5)*
  This part generates the filter output by using the FFF. Computational complexity per sample for this part is evaluated as

$$C_{\mathrm{f}} = (L/N_1)\mathrm{MVP}_{\mathrm{FFF}}(N_1)/N_1 + \log_2 N_1. \tag{49}$$

- *Correction Part (2–7 in Fig. 5)*
  This part recursively computes the correction term and adds it to the filter output obtained in the filtering part to make the filter output exactly the same as that of the sample-by-sample projection algorithm. If the prefilter $\mathbf{g}(k)$ is updated using the fast transversal filters (FTF) algorithm [3], [4] with computational complexity $15p$, then computational complexity per sample for the correction part is evaluated as

$$\begin{aligned} C_{\mathrm{c}} &= \sum_{i=0}^{\max(N_1, N_2)-1} (4i + 20p - 2)/\max(N_1, N_2) \\ &\approx 2\max(N_1, N_2) + 20p. \end{aligned} \tag{50}$$

- *Filter Updating Part (8 in Fig. 5)*
  This part updates the filter by using the FFF. Computational complexity per sample for this part is evaluated as

$$C_{\mathrm{u}} = (L/N_2)\mathrm{MVP}_{\mathrm{FFF}}(N_2)/N_2 + \log_2 N_2. \tag{51}$$

Total computational complexity for the proposed algorithm is

$$\begin{aligned} C_{\mathrm{T}} &= C_{\mathrm{f}} + C_{\mathrm{c}} + C_{\mathrm{u}} \\ &= L\mathrm{MVP}_{\mathrm{FFF}}(N_1)/N_1^2 + \log_2 N_1 + 2\max(N_1, N_2) \\ &\quad + 20p + L\mathrm{MVP}_{\mathrm{FFF}}(N_2)/N_2^2 + \log_2 N_2. \end{aligned} \tag{52}$$

Table I compares the computational complexity of the proposed block exact fast affine projection algorithm C_{BEFAP} with that of sample-by-sample affine projection algorithm C_{FAP}, the fast exact LMS algorithm C_{FELMS}, block exact fast Newton transversal filtering algorithm C_{BEFNTF}, and the fast subsampled-updating RLS C_{FSURLS}. We use the following equations to approximately evaluate the computational complexities:

$$C_{\mathrm{FAP}} = 2L + 20p \tag{53}$$

$$C_{\mathrm{FELMS}} = 2(L/N^2)\mathrm{MVP}_{\mathrm{FFF}}(N) + 2N \tag{54}$$

$$\begin{aligned} C_{\mathrm{BEFNTF}} &= [2(L/N) + 1]\mathrm{MVP}_{\mathrm{FFF}}(N)/N + 3N + 12p \\ &\quad + 3\mathrm{MVP}_{\mathrm{FFF}}(p)/p \end{aligned} \tag{55}$$

$$C_{\mathrm{FSURLS}} = 8(L/N^2)\mathrm{MVP}_{\mathrm{FFF}}(N) + 5.5N. \tag{56}$$

Here, $p = 8$ is used in (52), (53), and (55) and $N_1 = N_2 = N$ is used in (52). In computing $\mathrm{MVP}_{\mathrm{FFF}}(N)$ by (23), $n_0 = 2$ and $q = 2.5$ are used. Table I evaluates (52)–(56) for various filter lengths L and block lengths N. We can see that the proposed algorithm saves more than 50% of computation compared with the sample-by-sample FAP for $L > 1024$ and that it has computational complexity comparable to the FELMS algorithm, which is smaller than the other block exact algorithms BEFNTF and FSU-RLS.

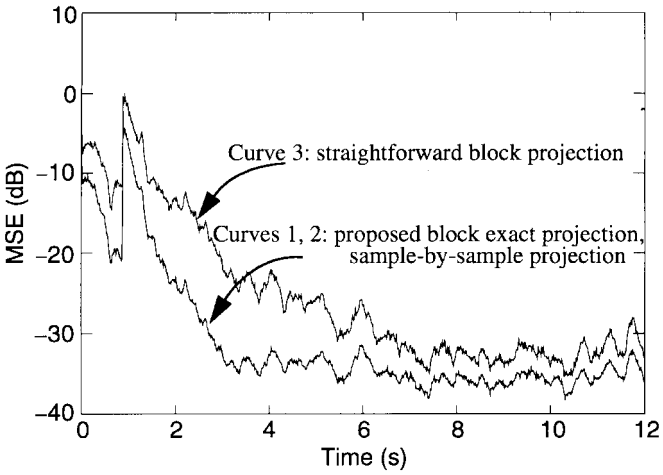


Fig. 2. MSE curves for the straightforward block, sample-by-sample, and proposed exact block projection algorithms (average of 50 trials). Experimental conditions: filter length $L = 1024$, projection order $p = 8$, block lengths $N_1 = N_2 = 8$, step sizes $\mu = 0.5$ for the sample-by-sample and proposed projection algorithms, $\mu = 1$ for the straightforward block projection, noise level -40 dB, input signal is speech.

E. Comparison of Convergence Curves with the Conventional Projection Algorithms

This section compares convergence curves for the original sample-by-sample projection algorithm, the straightforward

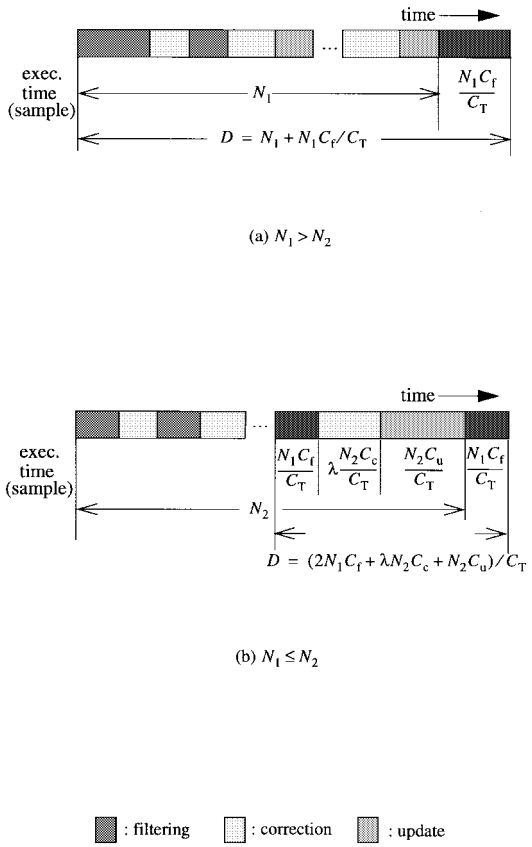


Fig. 3. Order of execution and filter output delay D.

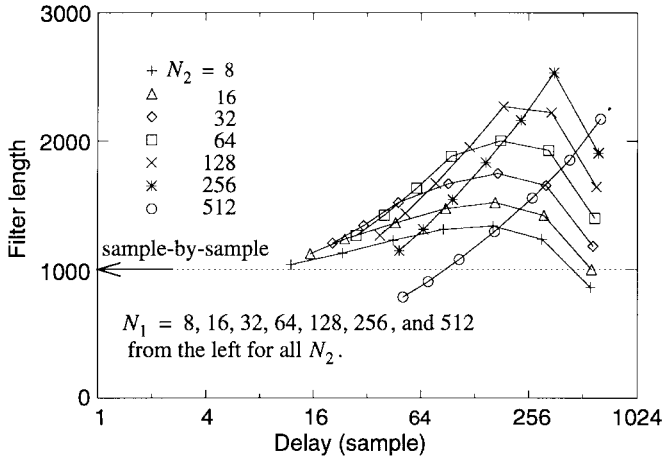


Fig. 4. Filter length versus delay for various pairs of block lengths N_1 and N_2. Total computational complexity $C_T = 2160$, projection order $p = 8$.

block projection algorithm, and the proposed fast exact block projection algorithm. Experiments were carried out by computer simulation for acoustic echo cancellation. The unknown system was the acoustic path from the loudspeaker to the microphone. The experimental conditions were as follows: length of the true acoustic impulse response $\mathbf{h}$ and that of the adaptive filter were both $L = 1024$, projection order $p = 8$, block lengths were $N_1 = N_2 = 8$, and the input signal was speech sampled at 8 kHz with bandpass filter 0.3–3.4 kHz. White noise was added at the microphone at a level of -40 dB relative to the average speech level. The performance of the adaptive filter (echo canceler) is indexed by the mean squared error (MSE) defined as

$$\text{MSE} = 10 \log_{10} |\mathbf{x}^T(k)\mathbf{h}(k) - \hat{y}(k)|^2 \,\text{dB}. \tag{57}$$

Fig. 2 shows the results of the computer simulation in which MSE curves for the three projection algorithms were drawn. The impulse response is changed at 1 s. Curves 1, 2, and 3 each shows the average of 50 trials. They show the behavior of MSE for the original sample-by-sample projection algorithm, the proposed fast exact block projection algorithm, and the straightforward block projection algorithm. Curves 1 and 2 are identical. This verifies that the proposed fast exact block projection algorithm has exactly the same convergence rate as the original sample-by-sample projection algorithm. We also see that the fast exact block projection algorithm has faster convergence than the straightforward block projection algorithm.

IV. Choice of Block Length

In designing an adaptive filter using the proposed algorithm, we need to find a pair of block lengths N_1 and N_2 that satisfies certain requirements for both the delay and the computational complexity and also give the longest adaptive filter length. This section shows the relationship between the output delay and the computational complexity of the fast exact block projection algorithm in terms of the two block lengths.

- *Filter Length*: If the computational complexity per sample is constant, then by rearranging (52), we get the relationship between the filter length L and the block lengths N_1 and N_2 as

$$L = \frac{C_T - 2\max(N_1, N_2) - 20p - \log_2 N_1 - \log_2 N_2}{\text{MVP}(N_1)/N_1^2 + \text{MVP}(N_2)/N_2^2}. \tag{58}$$

- *Delay*: The input signal at the beginning of a filtering part has the longest wait to be filtered. Then the output corresponding to the input is computed immediately as the correction part right after the next filtering part begins. Thus, the maximum output delay can be evaluated as the longest period from one filtering part to the end of the next filtering part. Fig. 3 illustrates the maximum filter output delay for two cases, i.e., $N_1 > N_2$ and $N_1 \le N_2$. We see that the filter output is delayed by

$$D = \begin{cases} N_1 + N_1 C_f / C_T & (N_1 > N_2) \\ (2N_1 C_f + \lambda N_2 C_c + N_2 C_u)/C_T & (N_1 \le N_2) \end{cases}. \tag{59}$$

The coefficient λ represents the ratio of time consumed by the correction section just before the update section to the total correction sections within the block and is evaluated as

$$\begin{aligned} \lambda &= \sum_{i=N_2-N_1}^{N_2-1} (4i + 20p - 2) \Bigg/ \sum_{i=0}^{N_2-1} (4i + 20p - 2) \\ &= N_1(-2N_1 + 4N_2 + 20p - 4)/[N_2(2N_2 + 20p - 4)]. \end{aligned} \tag{60}$$

0. Initialization

$k_{\text{current}} = N_2 - \nu$, $k = k_{\text{current}} - \nu$. ν : delay to avoid data shortage

$k' = k - N_1 + 1$,

$r_j(k'-1) = \mathbf{x}^T(k'-1)\mathbf{x}(k'-1-i)$, $j = 0, 1, \ldots, N_2 + p - 2$

$$\rho_{i+p-1}(k'-jN_2+i) = \sum_{l=0}^{i} x(k'-jN_2+l)x(k'-jN_2-i-p+1+l),$$

$i = 0, 1, \ldots, N_2 - 1$, $j = 1, 2, \ldots, L/N_2$

$\mathbf{e}(k'-1) = [y(k'-1), y(k'-2), \ldots, y(k'-p)]^T$,

$\mathbf{s} = \mathbf{0}$,

$\mu(k'-1) = 0$

# computation	computational complexity
For $i = 0, 1, \ldots, N_2 - 1$ do 1. to 7.	
1. If $i \bmod N_1 = 0$ then $\hat{\mathbf{y}}'(k) = \mathbf{X}^T_{L,N_1}(k)\mathbf{z}(k')$.	$(L/N_1)\text{MVP}(N_1) + N_1 \log_2 N_1$
2. $r_j(k'+i) = r_j(k'+i-1) + x(k'+i)x(k'+i-j)$, $-x(k'-L+i)x(k'-L+i-j)$, $j = 0, 1, \ldots, i+p-2$	$2(i+p-1)$
$\rho_{i+p-1}(k'+i) = \sum_{l=0}^{i} x(k'+l)x(k'-i-p+1+l)$	$i+1$
$r_{i+p-1}(k'+i) = r_{i+p-1}(k'-1) + \rho_{i+p-1}(k'+i) - \rho_{i+p-1}(k'-L+i)$	2
3. $e(k'+i) = y(k'+i) - \hat{y}'(k'+i) - \mathbf{s}^T_{i+p-1}(k'+i-1)\mathbf{r}_{i+p-1}(k'+i)$	$i+p$
4. $\begin{bmatrix}\mathbf{e}(k'+i)\\ *\end{bmatrix} = \begin{bmatrix} e(k'+i)\\ (1-\mu(k'+i-1))\mathbf{e}(k'+i-1)\end{bmatrix}$	$p-1$
5. Constitute $\mathbf{R}(k'+i)$ from $r_j(k'+i-l)$, $(j, l = 0, 1, \ldots, p-1)$, and solve $\mathbf{R}(k'+i)\mathbf{g}(k'+i) = \mathbf{e}(k'+i)$.	$15p$, if FTF is used.
6. $\mathbf{s}_{i+p}(k'+i) = \begin{bmatrix}0\\ \mathbf{s}_{i+p-1}(k'+i-1)\end{bmatrix} + \mu(k'+i)\begin{bmatrix}\mathbf{g}(k'+i)\\ 0\end{bmatrix}$	p
7. $k = k+1$	
8. $\mathbf{z}(k'+N_2) = \mathbf{z}(k') + \mathbf{X}_{L,N_2}(k'-p+N_2)\mathbf{s}\big|^{p}_{p+N_2-1}(k'+N_2-1)$	$(L/N_2)\text{MVP}(N_2) + N_2 \log_2 N_2$
9. $k' = k'+N_2$	

Fig. 5 Block exact fast affine projection algorithm ($N_1 \leq N_2$).

Using (58) and (59), for various pairs of block lengths N_1 and N_2, we can plot the relationship between the delay and the filter length and then determine the pair of block lengths that gives the longest filter length for a given delay. Fig. 4 gives an example of the relationship between the delay and the filter length, where total complexity $C_{\text{T}} = 2160$ and the projection order $p = 8$ is assumed. Here, we see that the filter length can be 20% longer than the conventional sample-by-sample projection algorithm with a delay of about 16 samples when pair of the block lengths is $(N_1, N_2) = (8, 32)$ and that, if a delay of a few hundred samples is permissible, the pair $(N_1, N_2) = (128, 128)$ can double the filter length compared with the sample-by-sample fast projection algorithm.

V. Conclusions

We have developed a fast exact block projection algorithm that overcomes the degradation in convergence rate seen in the straightforward block projection algorithm, and has exactly the same convergence rate as the original sample-by-sample projection algorithm. This is achieved by 1) introducing a correction term that compensates for the filter output difference between the sample-by-sample projection algorithm and the straightforward block projection algorithm, and 2) applying a fast FIR filtering technique for computing filter outputs and updating the filter. We also described how to choose a pair of block lengths that gives the longest filter length where the total computational complexity is constrained. An example showed

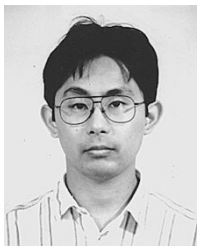
that the filter length can be doubled if a delay of a few hundred samples is permissible.

ACKNOWLEDGMENT

The authors would like to thank Dr. N. Kitawaki for his continuous encouragement.

REFERENCES

[1] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," *Electron. Commun. Jpn.*, vol. 67-A, pp. 19–27, 1984.
[2] Y. Maruyama, "A fast method of projection algorithm," in *Proc. 1990 IEICE Spring Conf.*, 1990, p. B-774, in Japanese.
[3] M. Tanaka, S. Makino, Y. Kaneda, and J. Kojima, "A fast projection algorithm for adaptive filtering," *IEICE Trans. EA*, vol. E78-A, pp. 1355–1361, Oct. 1995.
[4] S. L. Gay, "The fast affine projection algorithm," in *Proc. ICASSP-95*, pp. 3023–3026.
[5] D. R. Morgan and S. G. Kratzer, "On a class of computationally-efficient, rapidly-converging, generalized NLMS algorithms," *IEEE Signal Processing Lett.*, vol. 3, pp. 245–247, Aug. 1996.
[6] 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.
[7] J. M. Cioffi and T. Kailath, "Fast recursive-least-squares transversal filters for adaptive filtering," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 2, pp. 304–337, Jan. 1984.
[8] K. Kurosawa and T. Furusawa, "A geometric interpretation of adaptive algorithms," *Trans. IEICE (A)*, vol. J71-A, pp. 343–347, Feb. 1988, in Japanese.
[9] T. Furukawa, H. Kubota, and S. Tsujii, "A fast block adaptive algorithm and its performance," *Trans. IEICE (A)*, vol. J72-A, pp. 1069–1076, July 1989, in Japanese.
[10] M. Fukumoto, H. Kubota, and S. Tsujii, "New block algorithm using conjugate-gradient method in noisy environment and its performance," *Trans. IEICE (A)*, vol. J77-A, pp. 16–23, Jan. 1994, in Japanese.
[11] J. Benesty and P. Duhamel, "A fast exact least mean square adaptive algorithm," *IEEE Trans. Signal Processing*, vol. 40, pp. 2904–2920, Dec. 1992.
[12] K. Berberidis and S. Theodoridis, "An efficient block Newton-type algorithm," in *Proc. ICASSP-95*, 1995, pp. 1133–1136.
[13] 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 Processing*, vol. 40, pp. 5–20, Oct. 1994.
[14] Z. J. Mou and P. Duhamel, "Fast FIR filtering: Algorithms and implementation," *Signal Process.*, vol. 13, pp. 377–384, Dec. 1987.
[15] R. E. Blahut, *Fast Algorithms for Digital Signal Processing.* Reading, MA: Addison-Wesley, 1987.



Masashi Tanaka was born in Sapporo, Japan, on January 5, 1996. He received the B.E. and M.E. degrees from Hokkaido University, Hokkaido, Japan, in 1988 and 1990, respectively.

He joined Nippon Telegraph and Telephone Corporation (NTT), Tokyo, Japan, in 1990. Since then, he has been engaged in signal processing in microphone array and acoustic echo cancellation.

Mr. Tanaka is a member of the Acoustical Society of Japan and the Institute of Electronics, Information, and Communication Engineers of Japan.

Shoji Makino (A'89–M'90) was born in Nikko, Japan, on June 4, 1956. He received the B.E., M.E., and Ph.D. degrees from Tohoku University, Sendai, Japan, in 1979, 1981, and 1993, respectively.

He joined the Electrical Communication Laboratory of Nippon Telegraph and Telephone Corporation (NTT), Tokyo, Japan, in 1981. Since then, he has been engaged in research on electroacoustic transducers and acoustic echo cancelers. He is now a Senior Research Engineer and a Supervisor at the Speech and Acoustics Laboratory of the NTT Human Interface Laboratories. His research interests include acoustic signal processing, and adaptive filtering and its applications.

Dr. Makino is a Member of the Acoustical Society of Japan and the Institute of Electronics, Information, and Communication Engineers of Japan.

Junji Kojima was born in Tokyo, Japan, on May 25, 1949. He received the B.E. degree from Waseda University in 1973.

He joined Nippon Telegraph and Telephone Corporation (NTT), Tokyo, in 1973. He is now General Manager of Multimedia Service Promotion Headquarters. His current research interests include acoustical signal processing.

Mr. Kojima is a member of the Acoustical Society of Japan and the Information Processing Society of Japan.