PAPER

A Fast Projection Algorithm for Adaptive Filtering

Masashi TANAKA†, Yutaka KANEDA†, Shoji MAKINO†, *and* Junji KOJIMA†, *Members*

SUMMARY This paper proposes a new algorithm called the fast Projection algorithm, which reduces the computational complexity of the Projection algorithm from $(p+1)L+O(p^3)$ to $2L+20p$ (where L is the length of the estimation filter and p is the projection order.) This algorithm has properties that lie between those of NLMS and RLS, i.e. less computational complexity than RLS but much faster convergence than NLMS for input signals like speech. The reduction of computation consists of two parts. One concerns calculating the pre-filtering vector which originally took $O(p^3)$ operations. Our new algorithm computes the pre-filtering vector recursively with about $15p$ operations. The other reduction is accomplished by introducing an approximation vector of the estimation filter. Experimental results for speech input show that the convergence speed of the Projection algorithm approaches that of RLS as the projection order increases with only a slight extra calculation complexity beyond that of NLMS, which indicates the efficiency of the proposed fast Projection algorithm.

key words: *adaptive filtering, projection algorithm, complexity reduction*

1. Introduction

Adaptive filtering techniques are being used in many applications, such as echo canceling and vibration control [1]. Among the many adaptive filtering algorithms, the LMS (Least Mean Squares) and NLMS (Normalized LMS) algorithms [2] are widely used in practice because of their simplicity. Although the computational complexity of the LMS and NLMS algorithms is low, convergence is very slow and tracking is poor for a colored input signal like speech. The RLS (Recursive Least Squares) algorithm is known to overcome this problem for colored input signal. However, its computational complexity is four times that of NLMS, even for the fast version [3], [4].

In recent years, an algorithm called Projection (or Affine Projection) [5] has been drawing attention. This algorithm has properties that lie between those of NLMS and RLS, i.e. less computational complexity than RLS but much faster convergence than NLMS for input signals like speech, which can be modeled as a low-order AR process. The Projection algorithm, however, still requires more computation than NLMS, which remains a problem.

Recently, efforts have been made to reduce the amount of computation. These can be categorized into two approaches. Both calculate the adjustment quantity based on the same equations but they differ in adaptation interval. One applies block operations [6]–[9], in which adaptation is performed every block shift length. Although block-wise operation reduces computation, it is reported that it degrades the convergence speed [6]. The other is approach removes redundant computation from the original iterative Projection algorithm, which preserves the original performance [10]–[12]. Following the latter approach this paper proposes a new algorithm called the fast Projection algorithm, which significantly reduces the computational complexity of the Projection algorithm. The reduction consists of two parts, one of which was addressed by Maruyama [10] for the simplest case.

Section 2 explains the NLMS and Projection algorithms, Sect. 3 derives the proposed complexity reduction method, and Sect. 4 shows experimental results for speech input. Conclusions are presented in Sect. 5.

2. Normalized Least Mean Squares (NLMS) and Projection Algorithms

2.1 Framework of the Study

A block diagram of an adaptive filter is shown in Fig. 1. Assume that the unknown system can be modeled by an FIR filter whose coefficients are written as a vector $\boldsymbol{h}=[h_1, h_2, \cdots, h_L]^T$, where L denotes the number of taps and T denotes transposition. The unknown system is estimated by an FIR filter with coefficients $\hat{\boldsymbol{h}}(k)=[\hat{h}_1(k), \hat{h}_2(k), \cdots, \hat{h}_L(k)]^T$, where k is the discrete time index. The $x(k)$, $y(k)$, $\hat{y}(k)$, and $e(k)$ in Fig. 1 respectively denote the input to the unknown system, the output from the unknown system, the estimated output, and the estimation error. The signal $y(k)$ is also called the desired signal.

In this diagram, the estimation filter $\hat{\boldsymbol{h}}(k)$ is adjusted at every sample instance so as to make the estimation output $\hat{y}(k)$ close to the unknown system output $y(k)$ by adding an adjustment vector $\Delta\hat{\boldsymbol{h}}(k)$.

Manuscript received January 27, 1995.
Manuscript revised May 8, 1995.

† The authors are with NTT Human Interface Laboratories, Musashino-shi, 180 Japan.

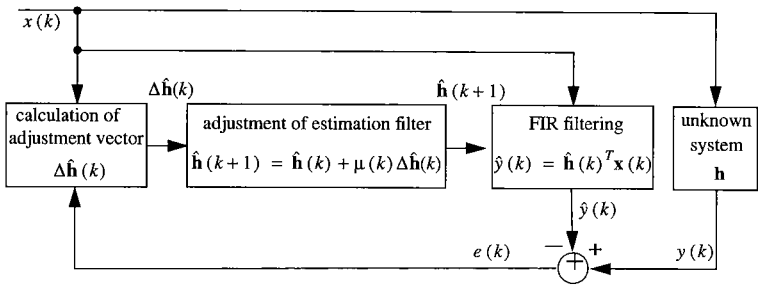


Fig. 1 Block diagram of an adaptive filter.

$$\hat{\boldsymbol{h}}(k+1)=\hat{\boldsymbol{h}}(k)+\mu(k)\Delta\hat{\boldsymbol{h}}(k) \tag{1}$$

Here, $\mu(k)$ is a scaling factor called the step size, which controls the convergence speed and the amount of residual error. We consider $\mu(k)$ time-variant in the general case, although it sometimes takes a time constant value.

2.2 NLMS Algorithm

In the NLMS algorithm, the adjustment vector $\Delta\hat{\boldsymbol{h}}(k)$ is determined as the minimum-norm solution of the following equation.

$$y(k)=\boldsymbol{x}(k)^T[\hat{\boldsymbol{h}}(k)+\Delta\hat{\boldsymbol{h}}(k)], \tag{2}$$

where

$$\boldsymbol{x}(k)=[x(k),\ x(k-1),\ \cdots,\ x(k-L+1)]^T. \tag{3}$$

Equation (2) means $\hat{\boldsymbol{h}}(k)$ is adjusted so that it outputs $y(k)$ for input $\boldsymbol{x}(k)$ when $\mu(k)=1$. The solution of $\Delta\hat{\boldsymbol{h}}(k)$ is written as

$$\Delta\hat{\boldsymbol{h}}(k)=\frac{\boldsymbol{x}(k)}{\|\boldsymbol{x}(k)\|^2}e(k), \tag{4}$$

where

$$e(k)=y(k)-\hat{y}(k) \tag{5}$$

$$\hat{y}(k)=\boldsymbol{x}(k)^T\hat{\boldsymbol{h}}(k). \tag{6}$$

2.3 Projection Algorithm

The Projection algorithm (or Affine Projection algorithm) was proposed as a generalization of the NLMS algorithm [5]. In the Projection algorithm of order p ($\leqq L$), the adjustment vector $\Delta\hat{\boldsymbol{h}}(k)$ is determined so as to satisfy the following p equations.

$$\begin{aligned}
&y(k)=\boldsymbol{x}(k)^T[\hat{\boldsymbol{h}}(k)+\Delta\hat{h}(k)]\\
&y(k-1)=\boldsymbol{x}(k-1)^T[\hat{\boldsymbol{h}}(k)+\Delta\hat{\boldsymbol{h}}(k)]\\
&\qquad\vdots\\
&y(k-p+1)=\boldsymbol{x}(k-p+1)^T[\hat{\boldsymbol{h}}(k)+\Delta\hat{\boldsymbol{h}}(k)].
\end{aligned} \tag{7}$$

In matrix form,

$$\boldsymbol{y}(k)=\boldsymbol{X}(k)^T[\hat{\boldsymbol{h}}(k)+\Delta\hat{\boldsymbol{h}}(k)] \tag{8}$$

or equivalently

$$\boldsymbol{X}(k)^T\Delta\hat{\boldsymbol{h}}(k)=\boldsymbol{y}(k)-\boldsymbol{X}(k)^T\hat{\boldsymbol{h}}(k)=\boldsymbol{e}(k), \tag{9}$$

where

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

$$\boldsymbol{X}(k)=[\boldsymbol{x}(k),\ \boldsymbol{x}(k-1),\ \cdots,\ \boldsymbol{x}(k-p+1)] \tag{11}$$

$$\boldsymbol{e}(k)=\boldsymbol{y}(k)-\boldsymbol{X}(k)^T\hat{\boldsymbol{h}}(k). \tag{12}$$

Here, $\boldsymbol{e}(k)$ is called the error vector.

Since the number of unknown variables L is greater than the number of equations p, Eq. (7) has an infinite number of solutions. In the sense of a minimum-norm solution, however, $\Delta\hat{\boldsymbol{h}}(k)$ is uniquely determined.

$$\Delta\hat{\boldsymbol{h}}(k)=\boldsymbol{X}(k)[\boldsymbol{X}(k)^T\boldsymbol{X}(k)]^{-1}\boldsymbol{e}(k) \tag{13}$$

This is rewritten as

$$\Delta\hat{\boldsymbol{h}}(k)=\boldsymbol{X}(k)\boldsymbol{g}(k), \tag{14}$$

where

$$\boldsymbol{g}(k)=\boldsymbol{R}(k)^{-1}\boldsymbol{e}(k) \tag{15}$$

$$\boldsymbol{R}(k)=\boldsymbol{X}(k)^T\boldsymbol{X}(k). \tag{16}$$

We call $\boldsymbol{g}(k)$ a pre-filtering vector in this paper, because it filters the row vectors of $\boldsymbol{X}(k)$ to synthesize the adjustment vector $\Delta\hat{\boldsymbol{h}}(k)$.

Based on Eqs. (1) and (12), and using the fact that $\Delta\hat{\boldsymbol{h}}(k-1)$ is determined at $k-1$ such that

$$y(k-j)-\boldsymbol{x}(k-j)^T[\hat{\boldsymbol{h}}(k-1)+\Delta\hat{\boldsymbol{h}}(k-1)]=0, \tag{17}$$

the $(j+1)$-th$(j=1,\ \cdots,\ p-1)$ element of the error vector $\boldsymbol{e}(k)$ can be updated as follows.

$$\begin{aligned}
e_{j+1}(k)&=y(k-j)-\boldsymbol{x}(k-j)^T\hat{\boldsymbol{h}}(k)\\
&=y(k-j)-\boldsymbol{x}(k-j)^T[\hat{\boldsymbol{h}}(k-1)\\
&\quad+\mu(k-1)\Delta\hat{\boldsymbol{h}}(k-1)]\\
&=\mu(k-1)[y(k-j)-\boldsymbol{x}(k-j)^T(\hat{\boldsymbol{h}}(k-1)\\
&\quad+\Delta\hat{\boldsymbol{h}}(k-1))]
\end{aligned}$$

$$+(1-\mu(k-1))[y(k-j) - \boldsymbol{x}(k-j)^T\hat{\boldsymbol{h}}(k-1)] = (1-\mu(k-1))\,e_j(k-1). \quad (18)$$

This update is rewritten in vector form as

$$\boldsymbol{e}(k) = \begin{bmatrix} e(k) \\ (1-\mu(k-1))\,\boldsymbol{e}_{p-1}(k-1) \end{bmatrix}, \quad (19)$$

where $\boldsymbol{e}_{p-1}(k-1)$ consists of the first $p-1$ elements of $\boldsymbol{e}(k-1)$, namely

$$\boldsymbol{e}_{p-1}(k-1) = [e_1(k-1),\ e_2(k-1), \cdots,\ e_{p-1}(k-1)]^T. \quad (20)$$

Equations (5) and (19) indicate that the error vector is updated with p multiplication-addition operations.

To obtain $\Delta\hat{\boldsymbol{h}}(k)$, the original Projection algorithm must first calculate $\boldsymbol{R}(k)^{-1}$, then compute $\boldsymbol{g}(k)$ by Eq. (15), and finally multiply $\boldsymbol{X}(k)$ by $\boldsymbol{g}(k)$ as shown in Eq. (14). Calculation of $\boldsymbol{R}(k)^{-1}$ increases the computational complexity, — specifically, multiplication-addition operations—by $O(p^3)$. Multiplying $\boldsymbol{X}(k)$ by $\boldsymbol{g}(k)$, together with updating $\hat{\boldsymbol{h}}(k)$ by Eq. (1), requires pL multiplication-addition operations. Including calculation of $\hat{y}(k)$ by Eq. (6), which takes L multiplication-addition operations, the total computational complexity of the original Projection algorithm is about $(p+1)L+O(p^3)$. Because of this computational burden, the Projection algorithm has been considered to be impractical.

3. Fast Projection Algorithm

This section introduces a new algorithm called fast Projection. This reduces the computational requirement to $2L+20p$. The reduction consists of two parts, one is the recursive updating of the prefiltering vector $\boldsymbol{g}(k)$, and the other involves pre-filtering $\boldsymbol{X}(k)$ by $\boldsymbol{g}(k)$.

3.1 A recursive Update Formula for Pre-Filtering Vector $\boldsymbol{g}(k)$

Computing $\boldsymbol{g}$(k) requires $O(p^3)$ multiplication-addition operations. For small value of p, this calculation cost is much smaller than L, and thus negligible. As p gets larger, however, the cost becomes comparable to or much larger than L. We focus on the simultaneous equations of Eq. (15) and show that the prefiltering vector $\boldsymbol{g}(k)$ can be recursively obtained with $O(p)$ operations.

According to Eqs. (11) and (16), $\boldsymbol{R}(k)$ is a $p\times p$ matrix that can be rewritten as

$$\boldsymbol{R}(k) = \sum_{i=1}^{L} \boldsymbol{x}_p(k+1-i)\,\boldsymbol{x}_p(k+1-i)^T, \quad (21)$$

where

$$\boldsymbol{x}_p(k) = [x(k),\ x(k-1), \cdots,\ x(k-p+1)]^T. \quad (22)$$

Similarly, we define $(p-1)$-th order covariance matrix $\boldsymbol{R}_{p-1}(k)$,

$$\boldsymbol{R}_{p-1}(k) = \sum_{i=1}^{L} \boldsymbol{x}_{p-1}(k+1-i)\,\boldsymbol{x}_{p-1}(k+1-i)^T, \quad (23)$$

where

$$\boldsymbol{x}_{p-1}(k) = [x(k),\ x(k-1), \cdots,\ x(k-p+2)]^T. \quad (24)$$

The inverse matrices of these covariance matrices have the following two relationships [14].

$$\boldsymbol{R}(k)^{-1} = \left[\begin{array}{c|c} 0 & \cdots\ 0 \\ \hline \vdots & \boldsymbol{R}_{p-1}(k-1)^{-1} \\ 0 & \end{array}\right] + \frac{\boldsymbol{a}(k)\,\boldsymbol{a}(k)^T}{F(k)} \quad (25)$$

$$\boldsymbol{R}(k)^{-1} = \left[\begin{array}{c|c} & 0 \\ \boldsymbol{R}_{p-1}(k)^{-1} & \vdots \\ & 0 \\ \hline 0\ \cdots & 0 \end{array}\right] + \frac{\boldsymbol{b}(k)\,\boldsymbol{b}(k)^T}{B(k)} \quad (26)$$

Here, $\boldsymbol{a}(k)$, $\boldsymbol{b}(k)$, $F(k)$, and $B(k)$ respectively denote the forward linear prediction coefficient vector, the backward linear prediction coefficient vector, the minimum value of the sum of forward a posteriori prediction-error squares, and the minimum value of the sum of backward a posteriori prediction-error squares. It is known that $\boldsymbol{a}(k)$, $\boldsymbol{b}(k)$, $F(k)$, and $B(k)$ satisfy the following augmented normal equations [14].

$$\boldsymbol{R}(k)\,\boldsymbol{a}(k) = [F(k),\ 0, \cdots,\ 0]^T \quad (27)$$

$$\boldsymbol{R}(k)\,\boldsymbol{b}(k) = [0, \cdots,\ 0,\ B(k)]^T \quad (28)$$

By substituting Eqs. (19) and (25) into (15), we get

$$\boldsymbol{g}(k) = (1-\mu(k-1))\begin{bmatrix} 0 \\ \boldsymbol{f}(k-1) \end{bmatrix} + \frac{\boldsymbol{a}(k)^T\boldsymbol{e}(k)}{F(k)}\,\boldsymbol{a}(k), \quad (29)$$

where

$$\boldsymbol{f}(k-1) = \boldsymbol{R}_{p-1}(k-1)^{-1}\boldsymbol{e}_{p-1}(k-1). \quad (30)$$

Similarly, from Eqs. (15), (19), and (26), we get

$$\begin{bmatrix} \boldsymbol{f}(k) \\ 0 \end{bmatrix} = \boldsymbol{g}(k) - \frac{\boldsymbol{b}(k)^T\boldsymbol{e}(k)}{B(k)}\,\boldsymbol{b}(k). \quad (31)$$

To avoid small denominators in the second term of the right side of Eqs. (29) and (31), we should add a positive number δ to denominators $F(k)$ and $B(k)$ as in the following equations.

$$\boldsymbol{g}(k)=(1-\mu(k-1))\begin{bmatrix}0\\ \boldsymbol{f}(k-1)\end{bmatrix}+\frac{\boldsymbol{a}(k)^T\boldsymbol{e}(k)}{F(k)+\delta}\boldsymbol{a}(k) \tag{32}$$

$$\begin{bmatrix}\boldsymbol{f}(k)\\ 0\end{bmatrix}=\boldsymbol{g}(k)-\frac{\boldsymbol{b}(k)^T\boldsymbol{e}(k)}{B(k)+\delta}\boldsymbol{b}(k) \tag{33}$$

Equations (32) and (33) give the recursion formula for updating $\boldsymbol{g}(k)$. Equation (32) indicates that $\boldsymbol{g}(k)$ is derived from $\boldsymbol{f}(k-1)$, which is derived from $\boldsymbol{g}(k-1)$ as can be seed in Eq. (33). The number of multiplication-addition operations for Eqs. (32) and (33) in $5p$. The $\boldsymbol{a}(k)$, $\boldsymbol{b}(k)$, $F(k)$, and $B(k)$ values are calculated using the sliding window version of the FTF algorithm with about $10p$ multiplication-addition operations [13] (FTF is known to have numerical instability. A remedy for this problem is to feed back numerical errors [4], which is also applicable to the sliding window version of the FTF.) In total, about $15p$ operations are required to derive $\boldsymbol{g}(k)$ with our proposed algorithm.

3.2 Pre-Filtering $\boldsymbol{X}(k)$ by $\boldsymbol{g}(k)$

The reduction of the computational complexity of the pre-filtering operation uses the approximation vector $\boldsymbol{z}(k)$, which was first introduced by Maruyama for $p=2$ [10]. In this section, we extend Maruyama's method to a more general formula.

Based on Eq. (1), $\hat{\boldsymbol{h}}(k+1)$ is written as

$$\begin{aligned}\hat{\boldsymbol{h}}(k+1)&=\mu(k)\Delta\hat{\boldsymbol{h}}(k)+\mu(k-1)\Delta\hat{\boldsymbol{h}}(k-1)\\&\quad+\cdots+\hat{\boldsymbol{h}}(0)\\&=\sum_{i=1}^{k}\mu(k+1-i)\Delta\hat{\boldsymbol{h}}(k+1-i)+\hat{\boldsymbol{h}}(0).\end{aligned} \tag{34}$$

By substituting Eq. (11) into Eq. (14), we get $\Delta\hat{\boldsymbol{h}}(k)$ as

$$\Delta\hat{\boldsymbol{h}}(k)=\sum_{j=1}^{p}g_j(k)\boldsymbol{x}(k-j+1), \tag{35}$$

where $g_j(k)$, $(j=1, \cdots, p)$ are elements of the vector $\boldsymbol{g}(k)$. From Eqs. (34) and (35), $\hat{\boldsymbol{h}}(k+1)$ is written as a linear combination of the past input signal vectors.

$$\begin{aligned}\hat{\boldsymbol{h}}(k+1)&=\sum_{i=1}^{k}\mu(k+1-i)\sum_{j=1}^{p}g_j(k+1-i)\\&\quad\cdot\boldsymbol{x}(k-i-j+2)+\hat{\boldsymbol{h}}(0)\\&=\mu(k)[g_1(k)\boldsymbol{x}(k)+g_2(k)\boldsymbol{x}(k-1)+\cdots\\&\quad+g_p(k)\boldsymbol{x}(k-p+1)]\\&\quad+\mu(k-1)[g_1(k-1)\boldsymbol{x}(k-1)\\&\quad+g_2(k-1)\boldsymbol{x}(k-2)+\\&\quad\cdots+g_p(k-1)\boldsymbol{x}(k-p)]+\cdots\\&\quad+\mu(1)[g_1(1)\boldsymbol{x}(1)+g_2(1)\boldsymbol{x}(0)+\cdots\\&\quad+g_p(1)\boldsymbol{x}(2-p)]+\hat{\boldsymbol{h}}(0)\\&=\sum_{i=1}^{k+p-1}s_i(k)\boldsymbol{x}(k+1-i)+\hat{\boldsymbol{h}}(0),\end{aligned} \tag{36}$$

where

$$\begin{aligned}s_1(k)&=\mu(k)g_1(k)\\s_2(k)&=\mu(k)g_2(k)+\mu(k-1)g_1(k-1)\\s_3(k)&=\mu(k)g_3(k)+\mu(k-1)g_2(k-1)\\&\quad+\mu(k-2)g_1(k-2)\\&\vdots\\s_p(k)&=\mu(k)g_p(k)+\mu(k-1)g_{p-1}(k-1)+\cdots\\&\quad+\mu(k-p+1)g_1(k-p+1)\\s_{p+1}(k)&=\mu(k-1)g_p(k-1)+\mu(k-2)g_{p-1}\\&\quad\cdot(k-2)+\cdots+\mu(k-p)g_1(k-p)\\s_{p+2}(k)&=\mu(k-2)g_p(k-2)+\mu(k-3)\\&\quad\cdot g_{p-1}(k-3)+\cdots+\mu(k-p-1)\\&\quad\cdot g_1(k-p-1)\\&\vdots\\s_{k+p-2}(k)&=\mu(2)g_p(2)+\mu(1)g_{p-1}(1)\\s_{k+p-1}(k)&=\mu(1)g_p(1).\end{aligned} \tag{37}$$

We call $s_i(k)$ smoothing coefficients. As can be seen from Eq. (37), $s_i(k)$ are updated as

$$\begin{aligned}&\text{for } i=1: && s_1(k)=\mu(k)g_1(k)\\&\text{for } i=2, \cdots, p: && s_i(k)=s_{i-1}(k-1)\\&&&\quad+\mu(k)g_i(k)\\&\text{for } i=p+1, \cdots, k+p-1: && s_i(k)=s_{i-1}(k-1).\end{aligned} \tag{38}$$

Note that all that require computation in the update are $s_i(k)$ for $i=1, \cdots, p$, and that the update formula is written in vector form as

$$\boldsymbol{s}(k)=\begin{bmatrix}0\\ \boldsymbol{s}_{p-1}(k-1)\end{bmatrix}+\mu(k)\boldsymbol{g}(k), \tag{39}$$

where

$$\boldsymbol{s}(k)=[s_1(k), s_2(k), \cdots, s_p(k)]^T \tag{40}$$

$$\boldsymbol{s}_{p-1}(k-1)=[s_1(k-1), s_2(k-1), \cdots, s_{p-1}(k-1)]^T. \tag{41}$$

Now, taking the summation from $i=p$ to $k+p-1$ instead of from $i=1$ to $k+p-1$, as shown in Eq. (36), we define the approximation vector $\boldsymbol{z}(k+1)$ of $\hat{\boldsymbol{h}}(k$

$+1)$ as

$$\boldsymbol{z}(k+1)=\sum_{i=p}^{k+p-1} s_i(k)\,\boldsymbol{x}(k+1-i)+\hat{\boldsymbol{h}}(0). \tag{42}$$

This linear combination becomes constant after time $k+1$, because the coefficients $s_i(k)$, $(i=p, \cdots, k+p-1)$ are not updated after time $k+1$. From Eqs. (15) and (37), this approximation vector $\boldsymbol{z}(k+1)$ is expected to approach $\hat{\boldsymbol{h}}(k+1)$ as it converges because $g_i(k)$, and thus $s_i(k)$ $(i=1, \cdots, p)$ approach zero as the error approaches zero.

The value of $\boldsymbol{z}(k+1)$ is recursively updated with L multiplication-addition operations for every sample instance as follows.

$$\begin{aligned}\boldsymbol{z}(k+1)&=s_p(k)\,\boldsymbol{x}(k+1-p)\\&\quad+\sum_{i=p+1}^{k+p-1} s_i(k)\,\boldsymbol{x}(k+1-i)+\hat{\boldsymbol{h}}(0)\\&=s_p(k)\,\boldsymbol{x}(k+1-p)+\boldsymbol{z}(k)\end{aligned} \tag{43}$$

From Eq. (42), Eq. (36) is rewritten as

$$\hat{\boldsymbol{h}}(k+1)=\boldsymbol{z}(k+1)+\sum_{i=1}^{p-1} s_i(k)\,\boldsymbol{x}(k+1-i). \tag{44}$$

Therefore calculating $\hat{\boldsymbol{h}}(k+1)$ via $\boldsymbol{z}(k+1)$ requires $p \times L$ computations, which is the same number as the conventional method using Eq. (14). This means that introducing the approximation vector $\boldsymbol{z}(k+1)$ is not an efficient way to obtain $\hat{\boldsymbol{h}}(k)$. In many adaptive filtering applications, however, it is sufficient to obtain the estimated output of the unknown system $\hat{y}(k)$, even if $\hat{\boldsymbol{h}}(k+1)$ is not available at every sample instance.

In the following, we show that the estimated output of the unknown system $\hat{y}(k)$ can be calculated using $\boldsymbol{z}(k)$ rather than using $\hat{\boldsymbol{h}}(k)$. By antedating Eq. (44) from $k+1$ to k, we obtain

$$\begin{aligned}\hat{\boldsymbol{h}}(k)&=\boldsymbol{z}(k)+\sum_{i=1}^{p-1} s_i(k-1)\,\boldsymbol{x}(k-i)\\&=\boldsymbol{z}(k)+[\boldsymbol{x}(k-1),\ \boldsymbol{x}(k-2),\\&\quad \cdots,\ \boldsymbol{x}(k-p+1)]\boldsymbol{s}_{p-1}(k-1).\end{aligned} \tag{45}$$

Equation (6) for $\hat{y}(k)$ is rewritten using Eq. (45).

$$\hat{y}(k)=\boldsymbol{x}(k)^T\boldsymbol{z}(k)+\boldsymbol{r}_{p-1}(k)^T\boldsymbol{s}_{p-1}(k-1) \tag{46}$$

Here, $\boldsymbol{r}_{p-1}(k)$ is a $(p-1)$-th order vector that consists of covariances of $x(k)$.

$$\begin{aligned}\boldsymbol{r}_{p-1}(k)=[&\boldsymbol{x}(k)^T\boldsymbol{x}(k-1),\ \boldsymbol{x}(k)^T\boldsymbol{x}(k-2),\\&\cdots,\ \boldsymbol{x}(k)^T\boldsymbol{x}(k-p+1)]^T\end{aligned} \tag{47}$$

By Eq. (46), $\hat{y}(k)$ is obtained with $L+p-1$ multiplication-addition operations using approximation vector $\boldsymbol{z}(k)$, covariance vector $\boldsymbol{r}_{p-1}(k)$ and smoothing vector $\boldsymbol{s}_{p-1}(k)$.

Since the i-th $(i=1, \cdots, p-1)$ element of the covariance vector $\boldsymbol{r}_{p-1}(k)$ is written as

$$\begin{aligned}\boldsymbol{x}(k)^T\boldsymbol{x}(k-i)&=\sum_{n=0}^{L-1} x(k-n)\,x(k-i-n)\\&=\sum_{n=0}^{L-1} x(k-1-n)\,x(k-1-i-n)\\&\quad+x(k)\,x(k-i)\\&\quad-x(k-L)\,x(k-i-L)\\&=\boldsymbol{x}(k-1)^T\boldsymbol{x}(k-i-1)\\&\quad+x(k)\,x(k-i)\\&\quad-x(k-L)\,x(k-i-L).\end{aligned} \tag{48}$$

$\boldsymbol{r}_{p-1}(k)$ is updated as

$$\begin{aligned}\boldsymbol{r}_{p-1}(k)&=\boldsymbol{r}_{p-1}(k-1)+x(k)\,\boldsymbol{x}_{p-1}(k-1)\\&\quad-x(k-L)\,\boldsymbol{x}_{p-1}(k-L-1).\end{aligned} \tag{49}$$

Thus the covariance vector $\boldsymbol{r}_{p-1}(k)$ is recursively updated with $2(p-1)$ multiplication-addition operations.

Here we introduce the approximation vector $\boldsymbol{z}(k)$. The pre-filtering process by Eq. (14), together with the update of the estimation filter $\hat{\boldsymbol{h}}(k)$ by Eq. (1), which takes pL operations in the original algorithm, is replaced by the update of the approximation vector $\boldsymbol{z}(k)$ by Eq. (43) which takes only L operations. Using the approximation vector $\boldsymbol{z}(k)$, the covariance vector $\boldsymbol{r}_{p-1}(k)$, and the smoothing vector $\boldsymbol{s}_{p-1}(k)$, the estimated output of the unknown system

Table 1 Fast Projection algorithm.

	Number of multiplication-additions (divisions)
Initialization $\mathbf{r}_{p-1}(0) = [\mathbf{x}(0)^T\mathbf{x}(-1), \mathbf{x}(0)^T\mathbf{x}(-2), \ldots, \mathbf{x}(0)^T\mathbf{x}(-p+1)]^T$, $\mathbf{f}(0) = 0$, $\mathbf{e}(0) = 0$, $\mathbf{s}(0) = 0$	
1. $\mathbf{r}_{p-1}(k) = \mathbf{r}_{p-1}(k-1) + x(k)\mathbf{x}_{p-1}(k-1) - x(k-L)\mathbf{x}_{p-1}(k-L-1)$	$2p$
2. $\hat{y}(k) = \mathbf{x}(k)^T\mathbf{z}(k) + \mathbf{r}_{p-1}(k)^T\mathbf{s}_{p-1}(k-1)$	$L+p-1$
3. $e(k) = y(k) - \hat{y}(k)$	1
4. $\mathbf{e}(k) = \begin{bmatrix} e(k) \\ (1-\mu(k-1))\mathbf{e}_{p-1}(k-1) \end{bmatrix}$	p
5. Computing $\mathbf{a}(k)$, $\mathbf{b}(k)$, $F(k)$, and $B(k)$ by the sliding window version of FTF. [13]	$10p$ (6)
6. $\mathbf{g}(k) = (1-\mu(k-1))\begin{bmatrix} 0 \\ \mathbf{f}(k-1) \end{bmatrix} + \frac{\mathbf{a}(k)^T\mathbf{e}(k)}{F(k)+\delta}\mathbf{a}(k)$	$3p$ (1)
7. $\begin{bmatrix} \mathbf{f}(k) \\ 0 \end{bmatrix} = \mathbf{g}(k) - \frac{\mathbf{b}(k)^T\mathbf{e}(k)}{B(k)+\delta}\mathbf{b}(k)$	$2p$ (1)
8. $\mathbf{s}(k) = \begin{bmatrix} 0 \\ \mathbf{s}_{p-1}(k-1) \end{bmatrix} + \mu(k)\mathbf{g}(k)$	p
9. $\mathbf{z}(k+1) = \mathbf{z}(k) + s_p(k)\mathbf{x}(k-p+1)$	L
Total (approximately)	$2L+20p$ (8)

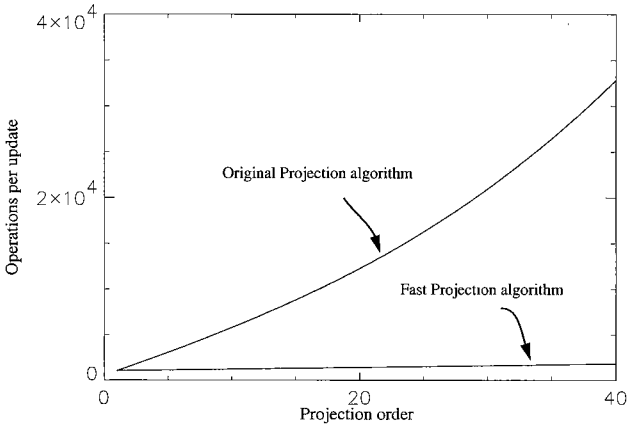


Fig. 2 Comparison of the number of operations between the fast Projection algorithm and the original Projection algorithm.

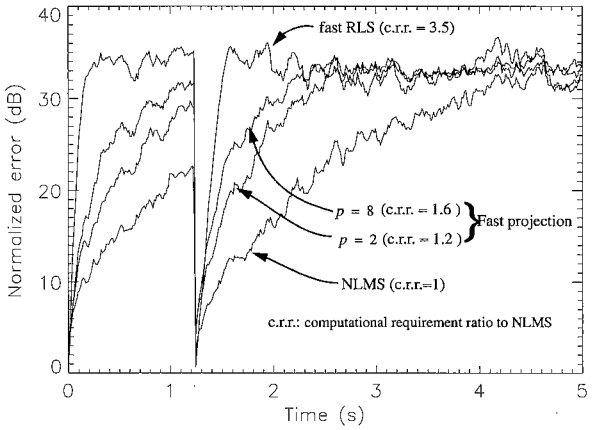


Fig. 3 ERLE curves for various orders of fast projection, NLMS, and fast RLS algorithms Computational requirement ratios (c.r.r.) to NLMS are given in parentheses. The impulse response is changed at time 1.25 (s).

$\hat{y}(k)$ is calculated by Eq. (46).

The complete procedures of the proposed algorithm are summarized in Table 1, where the approximate numbers of operations are also listed. The total computational requirement is about $2L+20p$, which is much smaller than the $(p+1)L+O(p^3)$ of the conventional method. Figure 2 compares the number of operations per update of the fast Projection algorithm $(2L+20p)$ and the original Projection algorithm $((p+1)L+p^3/6+p^2)$ for $L=500$.

4. Experimental Results

Convergence curves for various orders of the Projection algorithm and fast RLS [4] are shown in Fig. 3, in which the normalized error is defined as

$$\text{normalized error} = \mathrm{E}\left[20\log_{10}\left|\frac{y(k)}{y(k)-\hat{y}(k)}\right|\right]. \tag{50}$$

Here, E [.] means the average of 100 trials and the temporal average of 100 successive times. The computational requirement ratios (c.r.r.) to NLMS are given in parentheses in Fig. 3, where computational requirements are estimated as $2L$ for NLMS, $2L+20p$ for fast Projection, and $7L$ for fast RLS (L is the filter length and p is the projection order.) The conditions of this computer simulation are: filter length L is 128, impulse response length of the unknown system is also 128, and the input signal is speech. Noise is added to $y(k)$ with 40 dB SNR. The time-invariant step sizes μ and forgetting factor λ are selected through trial and error so that the steady-state normalized errors are the same for all methods; namely, $\mu=1$ for NLMS, $\mu=0.7$ for $p=2$, $\mu=0.5$ for $p=8$, $\mu=0.2$ for $p=32$, and $\lambda=0.9973$. The computer experiments are done with double precision, because calculating with single precision sometimes causes an accumulation of round-off errors.

From the figure, we can see that the convergence speed of the Projection algorithm approaches that of fast RLS as projection order p increases while the calculation complexity remains as much as that of NLMS. This result indicates that the proposed fast Projection algorithm is efficient because it improves convergence speed with little extra calculation cost.

5. Conclusion

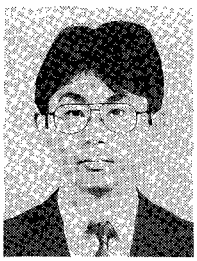
The fast version of the Projection algorithm reduces the total computation complexity from $(p+1)L+O(p^3)$ to $2L+20p$ (where L is the length of the estimation filter and p is the projection order.) The reduction of computation consists of two parts. One concerns calculating the pre-filtering vector which originally took $O(p^3)$ operations. Our new algorithm computes the pre-filtering vector recursively with about $15p$ operations. The other reduction is accomplished by introducing an approximation vector of the estimation filter. Update of the estimation filter, which takes pL operations in the original algorithm, is replaced by the update of the approximation vector, which takes only $L+3p$ operations. The estimated output of the unknown system is calculated using the approximation vector, the covariance vector, and the smoothing vector with $L+p$ operations. Another p operations are required to calculate the error vector. Experimental results for speech input show that the Projection algorithm approaches RLS as the projection order increases while the calculation complexity remains as much as that of NLMS. This result indicates the efficiency of the proposed fast Projection algorithm because it improves convergence speed with little extra calculation cost.

Acknowledgment

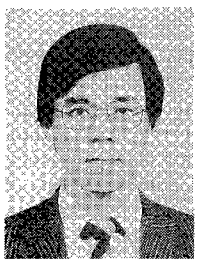
The authors would like to thank Dr. N. Kitawaki for his continuous encouragement.

References

[1] B. Widrow and S. D. Sterns, "Adaptive signal processing," Prentice-Hall, 1985.
[2] J. Nagumo, and A. Noda, "A learning method for system identification," IEEE Trans. Autom. Control, vol. AC-12, no. 3, pp. 282–287, June 1967.
[3] J. M. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," IEEE Trans. Acoust., Speech & Signal Processing, vol. ASSP-32, no. 2, pp. 304–337, April 1984.
[4] D. T. Slock and T. Kailath, "Numerically stable fast transversal filters for recursive least squares adaptive filtering," IEEE Trans. Acoust., Speech & Signal Processing, vol. 39, no. 1, pp. 92–114, Jan. 1991.
[5] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," Trans. ICEIE, vol. 67-A, no. 2, pp. 126–132, Feb. 1984.
[6] T. Furukawa, H. Kubota, and S. Tsujii, "Orthogonal projection algorithm for block adaptive signal processing and its properties," Trans. IEICE, vol. J71-A, no. 5, pp. 2138–2146, Dec. 1988.
[7] T. Furukawa, H. Kubota, and S. Tsujii, "A fast block adaptive algorithm and its performance," Trans. IEICE, vol. J72-A, no. 7, pp. 1069–1076, July 1989.
[8] M. Fukumoto, H. Kubota, and S. Tsujii, "New block algorithm using conjugate-gradient method in noisy environment and its performance," Trans. IEICE, vol. J77-A, no. 1, pp. 16–23, Jan. 1994.
[9] K. Kurosawa and T. Furusawa, "A geometric interpretation of adaptive algorithms," Trans. IEICE, vol. J71-A, no. 2, pp. 343–347, Feb. 1988.
[10] Y. Maruyama, "A fast method of projection algorithm," 1990 IEICE Spring Natl. Conv. Rec., B-744.
[11] M. Tanaka, Y. Kaneda, and S. Makino, "Reduction of computation for high-order projection algorithm," Proc. IEICE Fall Conf. '93, A-103.
[12] S. L. Gay, "A fast converging, low complexity adaptive filtering algorithm," proceedings of 3rd International Workshop on Acoustic Echo Control, pp. 223–226, Sept. 1993.
[13] J. M. Cioffi and T. Kailath, "Windowed fast transversal filters adaptive algorithms with normalization," IEEE Trans. Acoust., Speech & Signal Processing, vol. ASSP-33, no. 3, pp. 607–625, June 1985.
[14] S. Haykin, "Adaptive Filter Theory," Prentice-Hall, pp. 570–585, 1991.

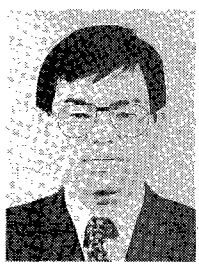


Masashi Tanaka was born in Sapporo, Japan on January 5, 1966. He received B.E. and M.E. from Hokkaido University in 1988 and 1990. He joined Nippon Telegraph and Telephone Corporation (NTT) in 1990. Since then, He has been engaged in signal processing in microphone array and acoustic echo cancellation. He is a member of the Acoustical Society of Japan.



Yutaka Kaneda was born in Osaka, Japan, on February 20, 1951. He received the B.E., M.E. and Doctor of Engineering degrees from Nagoya University, Nagoya, Japan, in 1975, 1977 and 1990. In 1977, he joined the Electrical Communication Laboratory of Nippon Telegraph and Telephone Corporation (NTT), Musashino, Tokyo, Japan. He has since been engaged in research on acoustic signal processing. He is now a Senior Research Engineer at the Speech and Acoustics Laboratory of the NTT Human Interface Laboratories. His recent research interests include microphone array processing, adaptive filtering and sound field control. He received the IEEE ASSP Senior Award in 1990 for an article on inverse filtering of room acoustics, and paper awards from the Acoustical Society of Japan in 1990 and 1992 for articles on adaptive microphone arrays and active noise control. Dr. Kaneda is a member of the Acoustical Society of Japan, the Acoustical Society of America.

Shoji Makino was born in Nikko, Japan, on June 4, 1956. He received the B.E., M.E., and Doctor of Engineering degrees from Tohoku University, Sendai, Japan, in 1979, 1981 and 1993. He joined the Electrical Communication Laboratory of Nippon Telegraph and Telephone Corporation (NTT) in 1981. Since then, he has been engaged in research on electroacoustic transducers and acoustic echo cancellers. He is now a Senior Research Engineer, 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 IEEE, the Acoustical Society of Japan.



Junji Kojima was born in Tokyo, Japan on May 25, 1949. He received B.E. degree from Waseda University in 1973. He joined NTT in 1973. He is a Group Leader of NTT Human Interface Laboratories. His current research interest is acoustical signal processing. He is a member of the Acoustical Society of Japan and the Information Processing Society of Japan.