FAST IMPLEMENTATION OF A FAMILY OF MEMORY PROPORTIONATE AFFINE PROJECTION ALGORITHM

Feiran Yang, Jun Yang

State Key Laboratory of Acoustics and Key Laboratory of Noise and Vibration Research,
Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China

ABSTRACT

Previously, a family of memory proportionate affine projection (MPAP) algorithms has been proposed by taking into account the “history” of the proportionate factors for sparse system identification. This paper presents a low-complexity implementation of this family of MPAP algorithms. Two most important ideas are used in the derivation of the fast algorithm. The first one is to update the auxiliary coefficient vector rather than the true coefficient vector. The second interesting idea is to calculate the error vector by using a recursive technique. Simulation results demonstrate the effectiveness of the fast algorithms.

Index Terms— adaptive filtering, proportionate affine projection, sparse impulse response, fast implementation

1. INTRODUCTION

The impulse responses are sparse in nature in many applications such as network echo cancellation. The classical adaptive filters exhibit slow convergence rate in this scenario. The proportionate-type algorithms have been proposed to speed up the convergence rate by assigning different step sizes to different coefficients. The proportionate adaptive algorithms were firstly developed based on the normalized least mean squares (NLMS) algorithm [1, 2, 3, 4]. By extending the proportionate idea to the affine projection (AP) algorithm straightforwardly, the proportionate AP (PAP) [5] and improved PAP (IPAP) [6] are proposed. Recently, Paleologu *et al.* [7] proposed a memory IPAP (MIPAP) algorithm by taking into account the memory of the proportionate coefficients. Compared to the IPAP algorithm, the MIPAP algorithm not only speeds up the convergence rate, but also reduces the computational complexity. Subsequently, based on the memory idea in [7], the μ-law MIPAP (MMIPAP) [8] and individual-activation-factor memory PAP (IAF-MPAP) [9] algorithms were developed. The only difference of the MIPAP, MMIPAP and IAF-MPAP algorithms is the calculation of the proportionate matrix, and thus they are called a family of memory-PAP (MPAP) algorithms. The robustness analysis of the NLMS with diagonal matrix step-size can be found in [10].

This work was supported by Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDA06040501 and also partially supported by National Natural Science Fund of China under Grant 11474307. We thank Dr. Felix Albu in University Politehnica of Bucharest for providing the AMIPAP code.

The complexity of the MPAP algorithms is still expensive, typically $O(4LM)$ operations per sample (L, M being the projection order and filter length). An approximated MIPAP (AMIPAP) algorithm [11] is proposed to reduce the complexity of MIPAP by forcing the matrix to be inverted a symmetric one. However, the approximated method leads to degraded performance for highly colored signals. In [7], it states that “the fact that $\mathbf{P}'(n)$ has the time-shift property [like the data matrix $\mathbf{X}(n)$] could be a possible opportunity to establish a link with the fast APA.” However, all the fast implementations of the MIPAP did not exploit this feature yet. In this paper, we develop a fast version of the MPAP-type algorithms by using the fast exact filtering approach in [12]. The error vector is computed by the auxiliary coefficient vector rather than the true weight vector. Simulation results verify the validity and performance advantage of the proposed fast algorithms.

2. MPAP ALGORITHMS

Consider the desired response $d(n)$ arising from the linear model

$$d(n) = \mathbf{w}_o^T\mathbf{x}(n) + v(n) \tag{1}$$

where $\mathbf{w}_o$ is the weight vector of the unknown system of a length M, $\mathbf{x}(n) = [x(n), x(n-1), ..., x(n-M+1)]^T$ denotes the input signal vector, and $v(n)$ is the system noise. The adaptive weight vector is defined by $\mathbf{w}(n) = [w_0(n), w_1(n), ..., w_{M-1}(n)]^T$. To describe the family of MPAP algorithms, we define the input signal vector $\mathbf{X(n)}$, the desired signal vector $\mathbf{d(n)}$, the filtered-out vector $\mathbf{y(n)}$, and the error vector $\mathbf{e(n)}$ as follows:

$$\mathbf{X}(n) = [\mathbf{x}(n), \mathbf{x}(n-1), ..., \mathbf{x}(n-L+1)], \tag{2}$$

$$\mathbf{d}(n) = [d(n), d(n-1), ..., d(n-L+1)]^T, \tag{3}$$

$$\begin{aligned}\mathbf{y}(n) &= [y_0(n), y_1(n), ..., y_{L-1}(n)]^T \\ &= \mathbf{X}^T(n)\mathbf{w}(n-1),\end{aligned} \tag{4}$$

$$\begin{aligned}\mathbf{e}(n) &= [e_0(n), e_1(n), ..., e_{L-1}(n)]^T \\ &= \mathbf{d}(n) - \mathbf{y}(n).\end{aligned} \tag{5}$$

The update equation of the MPAP algorithms can be written as

$$\begin{aligned}\boldsymbol{\varepsilon}(n) &= [\varepsilon_0(n), \varepsilon_1(n), ..., \varepsilon_{L-1}(n)]^T \\ &= \mu[\mathbf{P}^T(n)\mathbf{X}(n) + \delta\mathbf{I}]^{-1}\mathbf{e}(n),\end{aligned} \tag{6}$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{P}(n)\boldsymbol{\varepsilon}(n) \tag{7}$$

where μ is the step size, δ is a regularization parameter, and $\mathbf{P}(n)$ is computed as

$$\begin{aligned}\mathbf{P}(n) = [&\mathbf{g}(n-1)\bullet\mathbf{x}(n), \mathbf{g}(n-2)\bullet\mathbf{x}(n-1), \\ &..., \mathbf{g}(n-L)\bullet\mathbf{x}(n-L+1)] \\ = [&\mathbf{p}(n), \mathbf{p}(n-1), ..., \mathbf{p}(n-L+1)]\end{aligned} \tag{8}$$

with $\mathbf{g}(n) = [g_0(n), g_1(n), .., g_{M-1}(n)]^T$ is the proportionate vector, where the operator $\bullet$ denotes the Hadamard product. The only difference of the three algorithms is the calculation of the proportionate matrix. In the MIPAP algorithm [7], $g_l(n)$ is calculated as follows

$$g_l(n) = \frac{1-\alpha}{2M} + (1+\alpha)\frac{|w_l(n)|}{2\sum\limits_{i=0}^{M-1}|w_i(n)| + \sigma}, \tag{9}$$

where σ is a small constant. In the MMIPAP algorithm [8], $g_l(n)$ is evaluated as

$$F(|w_l(n)|) = \ln(1 + \mu_{\log}|w_l(n)|), \tag{10}$$

$$g_l(n) = \frac{1-\alpha}{2M} + (1+\alpha)\frac{F(|w_l(n)|)}{2\sum\limits_{i=0}^{M-1}F(|w_l(n)|) + \sigma}, \tag{11}$$

where a value of $\mu_{\log} = 1000$ is used. In the IAF-MPAP algorithm [9], $g_l(n)$ is given by

$$q_l(n) = \begin{cases} 0.5\,|w_l(n)| + 0.5\gamma_l(n-1), \bmod(n, M) = 0 \\ q_l(n-1), \qquad\qquad\qquad \text{otherwise}\end{cases}, \tag{12}$$

$$\gamma_l(n) = \max(q_l(n), |w_l(n)|), \tag{13}$$

$$g_l(n) = \frac{\gamma_l(n)}{\sum\limits_{l=0}^{M-1}\gamma_l(n)}. \tag{14}$$

Note that the difference between the MPAP and traditional PAP algorithms is the calculation of the matrix $\mathbf{P}(n)$. For the traditional PAP algorithm, $\mathbf{P}(n)$ is computed as [6]

$$\begin{aligned}\mathbf{P}(n) = [&\mathbf{g}(n-1)\bullet\mathbf{x}(n), \mathbf{g}(n-1)\bullet\mathbf{x}(n-1), \\ &..., \mathbf{g}(n-1)\bullet\mathbf{x}(n-L+1)].\end{aligned} \tag{15}$$

Calculation of $\mathbf{P}(n)$ by using (15) requires LM multiplications per sample while it only needs M multiplications per sample in (8) due to the time-shift property of $\mathbf{P}(n)$. In the MPAP algorithms, the matrices $\mathbf{P}(n)$ and $\mathbf{X}(n)$ both have the time-shift character which is the motivation behind the fast algorithms in the following.

3. FAST IMPLEMENTATION OF MPAP ALGORITHMS

3.1. Fast adaption of the weight vector

In this section, we will develop a fast method to update the weight vector by using the property that $\mathbf{P}(n)$ has the time-shift structure. Expanding the matrix/vector multiplications in (7), the weight vector $\mathbf{w}(n)$ can be rewritten as:

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \sum_{i=0}^{L-1}\mathbf{p}(n-i)\varepsilon_i(n). \tag{16}$$

Continuing to recursively expand (16) yields

$$\mathbf{w}(n) = \mathbf{w}(0) + \sum_{j=0}^{n-1}\sum_{i=0}^{L-1}\mathbf{p}(n-j-i)\varepsilon_i(n-j). \tag{17}$$

Assuming that $\mathbf{x}(n) = \mathbf{0}$ for $n < 0$, (17) can be rewritten as

$$\begin{aligned}\mathbf{w}(n) = \mathbf{w}(0) &+ \sum_{j=0}^{L-1}\mathbf{p}(n-j)\sum_{i=0}^{j}\varepsilon_i(n-j+i) \\ &+ \sum_{j=L}^{n-1}\mathbf{p}(n-j)\sum_{i=0}^{L-1}\varepsilon_j(n-j+i).\end{aligned} \tag{18}$$

The weight vector $\mathbf{w}(n)$ can be alternatively rewritten as [12]:

$$\mathbf{w}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{P}(n)\boldsymbol{\varphi}(n) \tag{19}$$

where

$$\hat{\mathbf{w}}(n-1) = \mathbf{w}(0) + \sum_{j=L}^{n-1}\mathbf{p}(n-j)\sum_{i=0}^{L-1}\varepsilon_j(n-j+i), \tag{20}$$

$$\boldsymbol{\varphi}(n) = [\varphi_0(n), \varphi_1(n), ..., \varphi_{L-1}(n)]^T \tag{21}$$

with $\varphi_m(n) = \sum\limits_{i=0}^{m}\varepsilon_i(n-m+i)$. Both $\hat{\mathbf{w}}(n)$ and $\boldsymbol{\varphi}(n)$ can be recursively computed, respectively,

$$\boldsymbol{\varphi}(n) = \boldsymbol{\varepsilon}(n) + \begin{bmatrix} 0 \\ \bar{\boldsymbol{\varphi}}(n-1)\end{bmatrix}, \tag{22}$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \mathbf{p}(n-L+1)\varphi_{L-1}(n) \tag{23}$$

where $\bar{\boldsymbol{\varphi}}(n)$ consists of the upper $L-1$ elements of $\boldsymbol{\varphi}(n)$.

3.2. Fast exact filtering

In this section, we will present a fast exact filtering approach [12] based on the time-shift property of the matrix $\mathbf{X}(n)$. Substituting (7) into (4) yields

$$\begin{aligned}\mathbf{y}(n) &= \mathbf{X}^T(n)\mathbf{w}(n-1) \\ &= \mathbf{X}^T(n)[\mathbf{w}(n-2) + \mathbf{P}(n-1)\boldsymbol{\varepsilon}(n-1)] \\ &= \mathbf{z}(n) + \mathbf{G}(n)\boldsymbol{\varepsilon}(n-1)\end{aligned} \tag{24}$$

where $\mathbf{G}(n) = \mathbf{X}^T(n)\mathbf{P}(n-1)$, and $\mathbf{z}(n) = \mathbf{X}^T(n)\mathbf{w}(n-2)$. Taking (4) into account, $\mathbf{z}(n)$ can be rewritten as

$$\begin{aligned}\mathbf{z}(n) &= \mathbf{X}^T(n)\mathbf{w}(n-2) \\ &= [z_0(n), y_0(n-1), ..., y_{L-2}(n-1)]^T\end{aligned} \tag{25}$$

where

$$z_0(n) = \mathbf{x}^T(n)\mathbf{w}(n-2). \tag{26}$$

Having expressed $\mathbf{w}(n)$ in a special way in (19), we can use the auxiliary coefficient vector $\hat{\mathbf{w}}(n)$ instead of $\mathbf{w}(n)$ to calculate $z_0(n)$. Substituting (19) into (26), we obtain

$$\begin{aligned}z_0(n) &= \mathbf{x}^T(n)\mathbf{w}(n-2) \\ &= \mathbf{x}^T(n)[\hat{\mathbf{w}}(n-3) + \mathbf{P}(n-2)\boldsymbol{\varphi}(n-2)] \\ &= \mathbf{x}^T(n)\hat{\mathbf{w}}(n-3) + \mathbf{r}^T(n)\boldsymbol{\varphi}(n-2)\end{aligned} \tag{27}$$

where $\mathbf{r}(n) = \mathbf{P}^T(n-2)\mathbf{x}(n)$. Note that (27) is the exact calculation of $z_0(n)$ without any assumption. Defining $\rho^m_{px}(n) = \mathbf{p}^T(n)\mathbf{x}(n-m)$ and $\rho^m_{xp}(n) = \mathbf{x}^T(n)\mathbf{p}(n-m)$, the matrices $\mathbf{R}(n)$, $\mathbf{G}(n)$, and $\mathbf{r}(n)$ can be written as

$$\begin{aligned}\mathbf{R}(n) &= \mathbf{P}^T(n)\mathbf{X}(n) \\ &= \begin{bmatrix} \rho^0_{px}(n) & \rho^1_{px}(n) & ... & \rho^{L-1}_{px}(n) \\ \rho^1_{xp}(n) & \rho^0_{px}(n-1) & ... & \rho^{L-2}_{px}(n-1) \\ ... & ... & ... & ... \\ \rho^{L-1}_{xp}(n) & \rho^{L-2}_{xp}(n-1) & ... & \rho^0_{px}(n-L+1) \end{bmatrix}\end{aligned} \tag{28}$$

$$\begin{aligned}\mathbf{G}(n) &= \mathbf{X}^T(n)\mathbf{P}(n-1) \\ &= \begin{bmatrix} \rho^1_{xp}(n) & \rho^2_{xp}(n) & ... & \rho^L_{xp}(n) \\ \rho^0_{xp}(n-1) & \rho^1_{xp}(n-1) & ... & \rho^{L-1}_{xp}(n-1) \\ ... & ... & ... & ... \\ \rho^{L-2}_{px}(n-1) & \rho^{L-1}_{px}(n-2) & ... & \rho^1_{xp}(n-L+1) \end{bmatrix}\end{aligned} \tag{29}$$

$$\begin{aligned}\mathbf{r}(n) &= \mathbf{P}^T(n-2)\mathbf{x}(n) \\ &= [\rho^2_{xp}(n), \rho^3_{xp}(n)(n), ..., \rho^{L+1}_{xp}(n)]^T.\end{aligned} \tag{30}$$

From (28)-(30) it is noted that the update of $\mathbf{R}(n)$, $\mathbf{G}(n)$, and $\mathbf{r}(n)$ only need to calculate $2L+1$ elements. Note that $\rho^m_{px}(n)$ and $\rho^m_{xp}(n)$ cannot be updated by a recursive technique in [12] because the vector $\mathbf{p}(n)$ does not hold the time-shift structure as $\mathbf{x}(n)$. Update of the three matrices needs $(2L+1)M$ multiplications and $(2L+1)M$ additions per sample.

Note that the matrix $\mathbf{R}(n)$ has the time-shift property but it is not symmetric because $\rho^m_{px}(n) \neq \rho^m_{xp}(n)$. In [11], an approximation is used to force the matrix $\mathbf{R}(n)$ a symmetric one. Although this approximation leads to important computational saving, the convergence performance is also degraded. Thus, the approximation method is not used in this paper.

3.3. Calculation of $\mathbf{g}(n)$

The proposed fast filtering method uses $\hat{\mathbf{w}}(n)$ rather than $\mathbf{w}(n)$ to calculate the filtered-out vector $\mathbf{y}(n)$. Therefore, the true weight vector $\mathbf{w}(n)$ is not available in the proposed method. However, the calculation of $\mathbf{g}(n)$ requires $\mathbf{w}(n)$. If the weight vector $\mathbf{w}(n)$ is computed by using (19), the complexity is LM multiplications per sample and no complexity reduction is obtained. To address this problem, we can periodically update $\mathbf{w}(n)$ and $\mathbf{g}(n)$ every L samples:

```
if    mod $(n, L) = 0$
      calculate   $\mathbf{w}(n)$   by   using   (19)
      calculate   $\mathbf{g}(n)$   by   using   (9) – (14)
else
      $\mathbf{g}(n) = \mathbf{g}(n-1)$
end
```
(31)

Thus, calculation of $\mathbf{g}(n)$ only needs M multiplications per sample. Simulation in the following confirms that this simplify cannot lead to significant performance lost.

3.4. Computational complexity

In fact, the MPAP algorithms have the same asymptotic computational complexity. To simplify the presentation, we just list the computational complexity of the improved proportionate-type AP algorithms in Table 1, i.e., the IPAP, MIPAP, AMIPAP and the proposed fast MIPAP algorithms. We use p_m indicate the multiplications associated with solving the linear systems of equations in (6). Note that the MIPAP algorithm achieves major complexity saving compared to the IPAP algorithm thanks to the recursive implementation of the "proportionate history." The fast MIPAP saves $(2L-4)M$ multiplications per sample than the original MIPAP algorithm. In practical applications such as echo cancellation, L is between 2 and 10 and $L \ll M$, so the obtained gain $(2L-4)M$ with respect to the value of p_m is still high. The complexity reduction is significant especially for system with a very long impulse response.

Table 1. Complexity of improved proportionate-type AP algorithms

Algorithm	Number of multiplications per sample
IPAP [6]	$(L^2+3L+1)M + p_m$
MIPAP [7]	$(4L+1)M + p_m$
AMIPAP [11]	$(3L+2)M + L + p_m$
Fast MIPAP	$(2L+5+1/L)M + L^2 + p_m$

4. SIMULATION RESULTS

Computer simulations are performed in the context of network echo cancellation to examine the proposed fast algorithms. The sampling rate is 8 kHz. The first impulse response from ITU-T G.168 Recommendation is padded with zeros in order to have $M = 512$ coefficients. An abrupt change of the echo path was introduced at the middle of the iterations, by shifting the impulse response to the right by 10 samples. An independent white noise signal is added to the

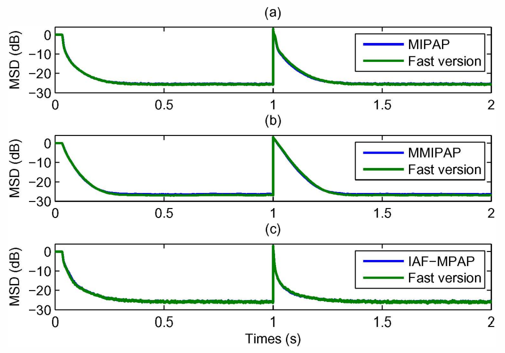

Fig. 1. Misalignment performance of MPAP algorithms and their fast versions with $L = 4$ and $\mu = 0.5$. Input signal is an AR(1) process. (a) MIPAP. (b) MMIPAP. (c) IAF-MPAP.

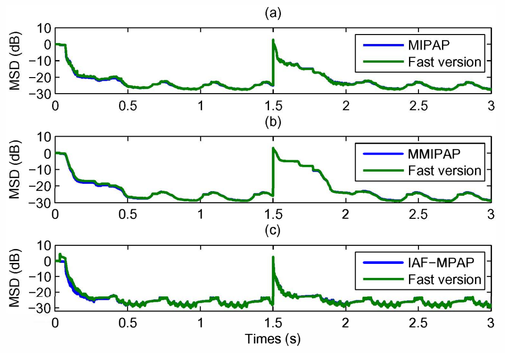

Fig. 2. Misalignment performance of MPAP algorithms and their fast versions with $L = 4$ and $\mu = 0.5$. Input signal is CSS. (a) MIPAP. (b) MMIPAP. (c) IAF-MPAP.

microphone signal, with 30-dB signal-to-noise ratio (SNR). The input signals are an AR process, the composite source signal (CSS), or a speech signal. The performance is evaluated in terms of mean square deviation (MSD) defined as $10\log_{10}\left[\|\mathbf{w}_{\rm o} - \mathbf{w}(n)\|^2/\|\mathbf{w}_{\rm o}\|^2\right]$. The results are obtained by averaging over 50 Monte Carlo trials. The regularization parameter is $\delta_{\rm AP} = 20\sigma_x^2$ for the AP algorithm. The parameters used for the MPAP algorithms are set as follows: MIPAP ($\alpha = 0$, $\sigma = 10^{-6}$, $\delta = \delta_{\rm AP}(1-\alpha)/2M$); MMIPAP ($\alpha = 0$, $\sigma = 10^{-6}$, $\delta = \delta_{\rm AP}(1-\alpha)/2M$); IAF-MPAP ($q_l(0) = 0.01/M$, $\delta = \delta_{\rm AP}/M$).

Fig. 1 compares the misalignment performance of the MIPAP, MMIPAP, IAF-MPAP algorithms and their corresponding fast versions for AR(1) input with coefficients (1, -0.9). The projection order is $L = 4$ and the step size is set to $\mu = 0.5$. Fig. 2 uses the same set of parameters as Fig. 1 except that the CSS is used as input. It can be observed from Fig. 1 and Fig. 2 that the proposed fast algorithms achieve almost the same performance with their standard ones. Thus, computing the proportionate vector every L samples cannot degrade the performance significantly.

In Fig. 3, we compare the performance of the proposed fast MIPAP with that of the AP, IPAP, MIPAP, and AMIPAP algorithms for $L = 8$ and $\mu = 0.5$. In Fig. 3(a), the input signal is AR(2) input with coefficients (1, -0.1, -0.8) while in Fig. 3(b) the input is a speech signal. It is noted that the AMIPAP algorithm has poor convergence performance due to the approximation on the matrix $\mathbf{R}(n)$, while the performance of the proposed fast MIPAP is comparable to that of the original MIPAP. This example demonstrates the robustness of the proposed fast algorithm.

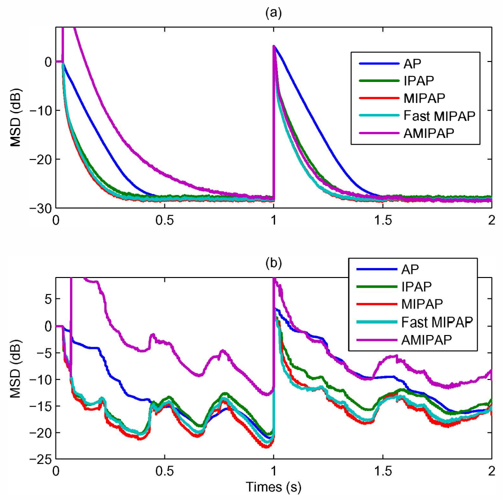

Fig. 3. Misalignment performance of the AP, IPAP, MIPAP, AMIPAP, and the proposed fast MIPAP algorithms with $L = 8$ and $\mu = 0.5$. (a) AR(2) input. (b) Speech signal as input.

5. CONCLUSION

This paper has presented a low-complexity implementation of a family of MPAP algorithms based on the time-shift property of the matrices $\mathbf{X}(n)$ and $\mathbf{P}(n)$. The proposed fast algorithms are the fast exact implementation of the original MPAP algorithms except that the proportionate vector of the fast versions is updated every L samples. Typically, the proposed fast algorithms saves about $(2L-4)M$ multiplications per sample than their original counterparts. The complexity reduction becomes more apparent when the projection order is large. Simulation results confirm the convergence performance of the proposed methods is comparable to that of the original algorithm.

6. REFERENCES

[1] D. L. Duttweiler, "Proportionate normalized least-mean-squares adaptation in echo cancellers," *IEEE Trans. Speech Audio Process.*, vol. 8, pp. 508–518, Feb. 2000.

[2] J. Benesty and S. L. Gay, "An improved PNLMS algorithm," in *Proc. IEEE ICASSP*, 2002, pp. 1881–1884.

[3] H. Deng and M. Doroslovacki, "Improving convergence of the PNLMS algorithm for sparse impulse response identification," *IEEE Signal Process. Lett.*, vol. 12, pp. 181–184, Mar. 2005.

[4] F. C. D. Souza, O. J. Tobias, R. Seara, and D. R. Morgan, "A PNLMS algorithm with individual activation factors," *IEEE Trans. Signal Process.*, vol. 58, pp. 2036–2047, Apr. 2010.

[5] T. Gansler, J. Benesty, S. L. Gay, and M. M. Sondhi, "A robust proportionate affine projection algorithm for network echo cancellation," in *Proc. IEEE ICASSP*, 2000, pp. 793–796.

[6] O. Hoshuyama, R. A. Goubran, and A. Sugiyama, "A generalized proportionate variable step-size algorithm for fast changing acoustic environments," in *Proc. IEEE ICASSP*, 2004, pp. IV–161–IV–164.

[7] C. Paleologu, S. Ciochină, and J. Benesty, "An efficient proportionate affine projection algorithm for echo cancellation," *IEEE Signal Process. Lett.*, vol. 17, pp. 165–168, Feb. 2010.

[8] J. Yang and G. E. Sobelman, "Efficient μ-law improved proportionate affine projection algorithm for echo cancellation," *Electron. Lett.*, vol. 47, pp. 73–74, Jan. 2011.

[9] H. Zhao, Y. Yu, S. Gao, X. Zeng, and Z. He, "Memory proportionate APA with individual activation factors for acoustic echo cancellation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 22, pp. 291–294, June 2014.

[10] R. Dallinger and M. Rupp, "On the robustness of lms algorithms with time-variant diagonal matrix step-size," in *Proc. IEEE ICASSP*, 2013, pp. 5691–5695.

[11] F. Albu, C. Paleologu, J. Benesty, and S. Ciochină, "A low complexity proportionate affine projection algorithm for echo cancellation," in *Proc. EUSIPCO*, 2010, pp. 6–10.

[12] F. Yang, M. Wu, J. Yang, and Z. Kuang, "A fast exact filtering approach to a family of affine projection-type algorithms," *Signal Processing*, vol. 101, pp. 1–10, Aug. 2014.