ON DATA-REUSE ADAPTIVE ALGORITHMS

Jacob Benesty[1] *and Tomas Gänsler*[2]

[1]Université du Québec, INRS-EMT, 800 de la Gauchetière Ouest, Suite 6900, Montréal, Canada
[2]Agere Systems, 555 Union Blvd, Allentown, PA 18109, USA
benesty@inrs-emt.uquebec.ca, gaensler@agere.com

ABSTRACT

Adaptive filters play an important role in signal processing and several different categories appear in the literature. In this paper, we discuss a class of adaptive algorithms called *data-reuse*. The hope of a data-reuse adaptive algorithm is to improve the convergence rate of the initial algorithm. We show, in particular, that the so-called block exact normalized LMS is a block version of the Schnaufer and Jenkins data-reuse NLMS, and show that many different forms of the latter exist but will not improve convergence or reduce complexity (compared to the initial form). We also discuss the structure and efficiency of these algorithms.

1. INTRODUCTION

The literature is rich in adaptive algorithms. Many efforts have been made during the last three decades to derive adaptive filters that converge faster and/or are more efficient from a complexity point of view than the classical least mean square (LMS) algorithm [1]. There are different categories of adaptive filtering: sample-by-sample, block, block exact, and data-reuse. The hope is that a data-reuse adaptive algorithm improves the convergence rate of the initial algorithm. While this is true in general, a data-reuse algorithm is, however, very inefficient when it comes to implementation.

In this paper, we discuss different forms of data-reuse adaptive filters and show some links with other algorithms. In particular, we show that the so-called block exact normalized LMS is a block version of the Schnaufer and Jenkins data-reuse NLMS, and show that many different forms of the latter exist but will not improve convergence or reduce complexity (compared to the initial form).

First, let us define the LMS algorithm:

$$
\begin{aligned}
e(n) &= y(n) - \hat{y}(n), && (1)\\
\hat{\mathbf{h}}(n+1) &= \hat{\mathbf{h}}(n) + \mu \mathbf{x}(n) e(n), && (2)
\end{aligned}
$$

where $e(n)$ is the error signal at time n between the system output

$$y(n) = \mathbf{h}^T \mathbf{x}(n) \quad (3)$$

and model filter output

$$\hat{y}(n) = \hat{\mathbf{h}}^T(n) \mathbf{x}(n), \quad (4)$$

$$\mathbf{h} = \begin{bmatrix} h_0 & h_1 & \cdots & h_{L-1} \end{bmatrix}^T$$

is the impulse response of the system,

$$\hat{\mathbf{h}}(n) = \begin{bmatrix} \hat{h}_0(n) & \hat{h}_1(n) & \cdots & \hat{h}_{L-1}(n) \end{bmatrix}^T$$

is the model filter,

$$\mathbf{x}(n) = \begin{bmatrix} x(n) & x(n-1) & \cdots & x(n-L+1) \end{bmatrix}^T$$

is a vector containing the last L samples of the input signal x, superscript ^{T} denotes transpose of a vector or a matrix, and μ is the step-size parameter ($0 < \mu < 2/\lambda_{\mathbf{max}}$, where $\lambda_{\mathbf{max}}$ is the maximum eigenvalue of the input signal's correlation matrix).

The first data-reuse adaptive algorithm was introduced by Shaffer and Williams [2] (and also independently by Nitzberg [3]) for the LMS algorithm and consists of reusing the same data N times. The data-reuse LMS (DR-LMS) algorithm is then given by the following equations:

- **Step 1:** $Initialization: \ i = 0$
  $$e(n) = e_0(n), \ \hat{\mathbf{h}}_0(n) = \hat{\mathbf{h}}(n) \quad (5)$$
- **Step 2:** *Loop* $: \ While\, i \leq N-1$
  $$e_i(n) = y(n) - \hat{\mathbf{h}}_i^T(n)\mathbf{x}(n) \quad (6)$$
  $$\hat{\mathbf{h}}_{i+1}(n) = \hat{\mathbf{h}}_i(n) + \mu \mathbf{x}(n) e_i(n), \ i = i+1 \quad (7)$$
- **Step 3:** $Update$
  $$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}_N(n), \ n = n+1, \ Goto\ Step\ 1. \quad (8)$$

It is clear that $N = 1$ reduces to the standard LMS update. It can easily be shown (see [4]) that the above algorithm can be rewritten as:

$$e_i(n) = e(n)[1 - \mu \mathbf{x}^T(n)\mathbf{x}(n)]^i, \ i = 0, ..., N-1, \quad (9)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu \mathbf{x}(n) \sum_{i=0}^{N-1} e_i(n), \quad (10)$$

but

$$\sum_{i=0}^{N-1} e_i(n) = \frac{e(n)\{1 - [1 - \mu \mathbf{x}^T(n)\mathbf{x}(n)]^N\}}{\mu \mathbf{x}^T(n)\mathbf{x}(n)}. \quad (11)$$

Hence

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \frac{\mathbf{x}(n)}{\mathbf{x}^T(n)\mathbf{x}(n)} e(n)\{1 - [1 - \mu \mathbf{x}^T(n)\mathbf{x}(n)]^N\},$$

so when $N \to \infty$, the DR-LMS is a special case ($\alpha = 1$) of the normalized LMS (NLMS) algorithm which is defined as follows [5]:

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \frac{\alpha \mathbf{x}(n)}{\mathbf{x}^T(n)\mathbf{x}(n)} e(n), \quad (12)$$

where α is a normalized step size. (From here on, for simplicity, we will suppress the step size α, with the understanding that it can always be reintroduced in any of the algorithms we discuss.) Thus, the DR-LMS algorithm has a convergence rate that lies between the LMS and NLMS algorithms. Obviously, from a practical point of view, the DR-LMS is not very attractive since it requires an infinite complexity to attain the performance of the NLMS. The only interesting thing about it is that it does not require any division unlike the NLMS.

2. THE BLOCK EXACT NLMS ALGORITHM

The block exact NLMS (BENLMS) algorithm is a block adaptive algorithm; it updates the coefficients of the filter only once per block N. The following equations summarize the algorithm (for more details, see [6]):

$$\mathbf{e}_\mathrm{a}(n) = \mathbf{y}(n) - \mathbf{X}^T(n)\hat{\mathbf{h}}(n-N+1), \quad (13)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n-N+1) + \mathbf{X}(n)\mathbf{S}^{-1}(n)\mathbf{e}_\mathrm{a}(n), \quad (14)$$

where

$$\mathbf{e}_\mathrm{a}(n) = \begin{bmatrix} e_\mathrm{a}(n) & e_\mathrm{a}(n-1) & \cdots & e_\mathrm{a}(n-N+1) \end{bmatrix}^T,$$

$$e_\mathrm{a}(n-i) = y(n-i) - \mathbf{x}^T(n-i)\hat{\mathbf{h}}(n-N+1),$$

$$\mathbf{y}(n) = \begin{bmatrix} y(n) & y(n-1) & \cdots & y(n-N+1) \end{bmatrix}^T,$$

$$\mathbf{X}(n) = \begin{bmatrix} \mathbf{x}(n) & \mathbf{x}(n-1) & \cdots & \mathbf{x}(n-N+1) \end{bmatrix},$$

$$\mathbf{S}(n) = \begin{bmatrix} s_0(n) & \cdots & 0 \\ \vdots & \vdots & \vdots \\ s_{N-1}(n) & \cdots & s_0(n-N+1) \end{bmatrix},$$

$$s_i(n) = \mathbf{x}^T(n)\mathbf{x}(n-i),\ i = 0, 1, ..., N-1.$$

The BENLMS and NLMS algorithms are mathematically equivalent. Hence, they have the same performance. The only difference between the two is that the BENLMS calculates the coefficients of the filter every N samples (so it has to wait for N new samples before starting to process them) instead of every sample for the NLMS. The advantage of the block approach, though, is that because of the redundancy [note that $\mathbf{X}(n)$ used in (13) and (14) is a Hankel matrix], the arithmetic complexity can be significantly reduced compared to a sample-by-sample approach by using divide-and-conquer techniques or the fast Fourier transform as an intermediary step [6].

3. THE SJ-DR-NLMS AND BENLMS ALGORITHMS

The Schnaufer and Jenkins DR-NLMS (SJ-DR-NLMS) algorithm [7] is an improved version of the DR-LMS algorithm where instead of iterating with an LMS on the same present data, the SJ-DR-NLMS algorithm iterates with data from the past and present with an NLMS. The SJ-DR-NLMS algorithm is defined as follows:

- **Step 1:** $Initialization:\ i = 0$

$$e(n) = e_0(n),\ \hat{\mathbf{h}}_0(n) = \hat{\mathbf{h}}(n) \quad (15)$$

- **Step 2:** $Loop:\ While\ i \le N-1$

$$e_i(n) = y(n-i) - \hat{\mathbf{h}}_i^T(n)\mathbf{x}(n-i) \quad (16)$$

$$\hat{\mathbf{h}}_{i+1}(n) = \hat{\mathbf{h}}_i(n) + \frac{\mathbf{x}(n-i)}{\mathbf{x}^T(n-i)\mathbf{x}(n-i)}e_i(n) \quad (17)$$

$$i = i+1 \quad (18)$$

- **Step 3:** $Update$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}_N(n),\ n = n+1,\ Goto\ Step\ 1. \quad (19)$$

We can see from the previous equations that the only two differences with the DR-LMS, is that, at iteration i, we use $\mathbf{x}(n-i)$ [resp. $y(n-i)$] instead of $\mathbf{x}(n)$ [resp. $y(n)$] and $1/\left[\mathbf{x}^T(n-i)\mathbf{x}(n-i)\right]$ instead of μ.

We now propose to rewrite the SJ-DR-NLMS algorithm differently. At iteration $i = 0$, we have:

$$e_0(n) = y(n) - \hat{\mathbf{h}}^T(n)\mathbf{x}(n), \quad (20)$$

$$\hat{\mathbf{h}}_1(n) = \hat{\mathbf{h}}(n) + \frac{\mathbf{x}(n)}{\mathbf{x}^T(n)\mathbf{x}(n)}e_0(n). \quad (21)$$

At iteration $i = 1$, we have:

$$e_1(n) = y(n-1) - \hat{\mathbf{h}}_1^T(n)\mathbf{x}(n-1), \quad (22)$$

$$\hat{\mathbf{h}}_2(n) = \hat{\mathbf{h}}_1(n) + \frac{\mathbf{x}(n-1)}{\mathbf{x}^T(n-1)\mathbf{x}(n-1)}e_1(n). \quad (23)$$

Replacing $\hat{\mathbf{h}}_1(n)$ in (22) and (23) by (21), we get respectively:

$$e_1(n) = y(n-1) - \hat{\mathbf{h}}^T(n)\mathbf{x}(n-1) - \frac{\mathbf{x}^T(n-1)\mathbf{x}(n)}{\mathbf{x}^T(n)\mathbf{x}(n)}e_0(n), \quad (24)$$

$$\hat{\mathbf{h}}_2(n) = \hat{\mathbf{h}}(n) + \frac{\mathbf{x}(n)}{\mathbf{x}^T(n)\mathbf{x}(n)}e_0(n) + \frac{\mathbf{x}(n-1)}{\mathbf{x}^T(n-1)\mathbf{x}(n-1)}e_1(n). \quad (25)$$

Continuing the same process until iteration $N-1$, we obtain exactly:

$$\mathbf{e}_\mathrm{a}(n) = \mathbf{y}(n) - \mathbf{X}^T(n)\hat{\mathbf{h}}(n), \quad (26)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mathbf{X}(n)\mathbf{S}^{-1}(n)\mathbf{e}_\mathrm{a}(n), \quad (27)$$

where all the variables used here were defined in the previous section. Comparing the rewritten SJ-DR-NLMS [(26)–(27)] and the BENLMS [(13)–(14)], we can see that the two algorithms have the same structure; the only difference is that the adaptive weight is held constant over a block for the BENLMS, whereas it is allowed to update on each sample for the SJ-DR-NLMS. Hence, the SJ-DR-NLMS converges to the Wiener solution and converges, in principle, faster than NLMS (or equivalently BENLMS). Since the NLMS is a one-dimensional affine projection algorithm (APA), so is the SJ-DR-NLMS. The multi-dimensional APA [8] is:

$$\mathbf{e}_\mathrm{a}(n) = \mathbf{y}(n) - \mathbf{X}^T(n)\hat{\mathbf{h}}(n), \quad (28)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mathbf{X}(n)\left[\mathbf{X}^T(n)\mathbf{X}(n)\right]^{-1}\mathbf{e}_\mathrm{a}(n). \quad (29)$$

We can easily see that the lower triangular parts of the two matrices $\mathbf{S}(n)$ and $\mathbf{X}^T(n)\mathbf{X}(n)$ are identical. Note that $\mathbf{S}(n)$ is not symmetric and all the elements above the main diagonal are zeroes. So we should not expect the SJ-DR-NLMS [(26)–(27)] to perform as well as the APA [(28)–(29)].

The comparison between the BENLMS and SJ-DR-NLMS is similar to the comparison between the multi-delay filter (MDF) [9] and generalized MDF (GMDF) [10]. Recall that the MDF is a frequency-domain adaptive algorithm using a block size smaller than the length of the adaptive filter, where the coefficients of this filter are adjusted once per block, while in the GMDF these coefficients can be adjusted more often within the same block, depending on the overlapping factor. This results in an increase of the convergence rate. In the BENLMS, the coefficients are updated once per block while they are updated N times per block for the SJ-DR-NLMS. In other words, the BENLMS is a block version of the SJ-DR-NLMS, like the partial rank (PRA) algorithm [11], defined as

$$\mathbf{e}_{\mathrm{a}}(n) = \mathbf{y}(n) - \mathbf{X}^T(n)\hat{\mathbf{h}}(n-N+1), \quad (30)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n-N+1) + \mathbf{X}(n)\left[\mathbf{X}^T(n)\mathbf{X}(n)\right]^{-1}\mathbf{e}_{\mathrm{a}}(n), \quad (31)$$

is a block version of the APA. While BENLMS is a block version of SJ-DR-NLMS, it is however a block exact version of NLMS.

The SJ-DR-NLMS as given at the beginning of this section is highly inefficient from a complexity point of view. It requires $2NL$ operations per time sample compared to $2L$ for the NLMS. In any application, it will be preferable to use BENLMS, PRA, or APA since there are many ways to implement these algorithms with much less complexity. For example, the BENLMS and PRA can be implemented with fewer number of operations than NLMS while exhibiting the same or better performance. However, the SJ-DR-NLMS as shown in (26)–(27) can be computed rather efficiently in $2L + O(N^2)$ operations by borrowing some of the ideas used in the fast affine projection algorithm [12] and noticing that $\mathbf{S}(n)$ is a triangular matrix and all its elements can be computed recursively. So when $N \ll L$, which is the case with long adaptive filters, the complexity of the SJ-DR-NLMS is comparable to the NLMS but with a faster convergence rate.

4. OTHER VERSIONS OF THE SJ-DR-NLMS ALGORITHM

In the loop (Step 2) of the SJ-DR-NLMS algorithm, the iterative process starts with $\mathbf{x}(n)$ [resp. $y(n)$] and finishes with $\mathbf{x}(n-N+1)$ [resp. $y(n-N+1)$]. Actually, the order in which we use this data should not change the behaviour of the algorithm as long as the impulse response that we want to identify is stationary. We can, for example, start the iterative process with $\mathbf{x}(n-N+1)$ [resp. $y(n-N+1)$] and finish with $\mathbf{x}(n)$ [resp. $y(n)$]. In this case, the algorithm becomes:

- **Step 1:** $Initialization:\ i = 0$

  $$\hat{\mathbf{h}}_0(n) = \hat{\mathbf{h}}(n) \quad (32)$$

- **Step 2:** $Loop:\ While\ i \le N-1$

  $$e_i(n) = y(n-N+1-i) - \hat{\mathbf{h}}_i^T(n)\mathbf{x}(n-N+1-i) \quad (33)$$

  $$\hat{\mathbf{h}}_{i+1}(n) = \hat{\mathbf{h}}_i(n) + \frac{\mathbf{x}(n-N+1-i)}{\mathbf{x}^T(n-N+1-i)\mathbf{x}(n-N+1-i)}e_i(n) \quad (34)$$

  $i = i+1$

- **Step 3:** $Update$

  $$e(n) = e_{N-1}(n) \quad (35)$$

  $$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}_N(n),\ n = n+1,\ Goto\ Step\ 1. \quad (36)$$

It can be checked that the above algorithm is equivalent to:

$$\mathbf{e}_{\mathrm{a}}(n) = \mathbf{y}(n) - \mathbf{X}^T(n)\hat{\mathbf{h}}(n), \quad (37)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mathbf{X}(n)\left[\mathbf{S}^T(n)\right]^{-1}\mathbf{e}_{\mathrm{a}}(n). \quad (38)$$

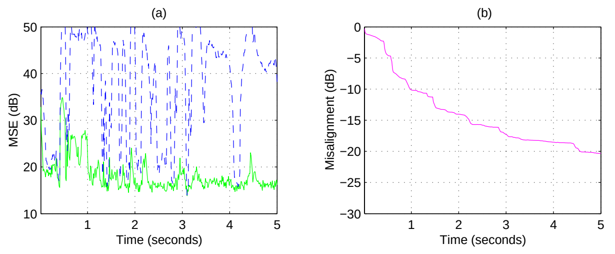

Fig. 1. Behavior of the NLMS algorithm with $\alpha = 0.5$. (a) MSE (–) as compared to the output signal level (– –). (b) Misalignment.

In the version of the previous section, the algorithm is normalized with a lower triangular matrix $[\mathbf{S}(n)]$, while in the version given above, the algorithm is normalized with an upper triangular matrix $[\mathbf{S}^T(n)]$. This is the only difference between the two algorithms. Amazingly, even with this difference of normalization, the algorithms should have the same performance.

In the general case, depending in which order the data in the loop is processed, the adaptive algorithm will have the following form:

$$\mathbf{e}_{\mathrm{a}}(n) = \mathbf{y}(n) - \mathbf{X}^T(n)\hat{\mathbf{h}}(n), \quad (39)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mathbf{X}(n)\mathbf{S}_{\mathrm{p}}^{-1}(n)\mathbf{e}_{\mathrm{a}}(n), \quad (40)$$

where the matrix $\mathbf{S}_{\mathrm{p}}(n)$ has its elements organized in the same order as the data is processed. Note, in general, that matrix $\mathbf{S}_{\mathrm{p}}(n)$ is not only constrained to be upper or lower triangular, but can be more generally obtained from $\mathbf{S}(n)$ by simple permutation of its elements. First, matrix $\mathbf{S}(n)$ has exactly $N(N-1)/2$ elements (above the main diagonal) that are equal to zero. The general matrix $\mathbf{S}_{\mathrm{p}}(n)$ must be built according to the following rules:

- The main diagonal of $\mathbf{S}_{\mathrm{p}}(n)$ must be the same as the main diagonal of $\mathbf{S}(n)$,
- $N(N-1)/2$ elements of $\mathbf{S}_{\mathrm{p}}(n)$ must be equal to zero,
- Two or more lines of the matrix $\mathbf{S}_{\mathrm{p}}(n)$ must not have the same number of zeroes.

Under this scenario, we will always have:

$$\mathbf{X}^T(n)\mathbf{X}(n) = \mathbf{S}_{\mathrm{p}}(n) + \mathbf{S}_{\mathrm{p}}^T(n) - \mathrm{diag}\{\mathbf{S}_{\mathrm{p}}(n)\}. \quad (41)$$

Again, the adaptive algorithm given by (39)–(40) with any matrix $\mathbf{S}_{\mathrm{p}}(n)$ following the previous rules should have the same performance as the original SJ-DR-NLMS.

5. SIMULATIONS

In this section, we compare by way of simulations, the NLMS, APA, and two versions of the SJ-DR-NLMS. The impulse response $\mathbf{h}$ to be identified is of length $L = 128$. The same length is used for all the adaptive filters $\hat{\mathbf{h}}(n)$. The input signal $x(n)$ is a 5 second speech signal sampled at 8 kHz. The signal-to-noise ratio is equal to 30 dB. All the algorithms were slightly modified. Indeed, for stability, we add a regularization parameter $\delta = 20\sigma_x^2$ to the diagonal of $\mathbf{S}$ and multiply the error signal with a step-size α ($0 < \alpha \le 1$).

Figures 1, 2, 3, and 4 show the mean-squared error (MSE) and the normalized misalignment, $\|\mathbf{h} - \hat{\mathbf{h}}(n)\|/\|\mathbf{h}\|$, for all the

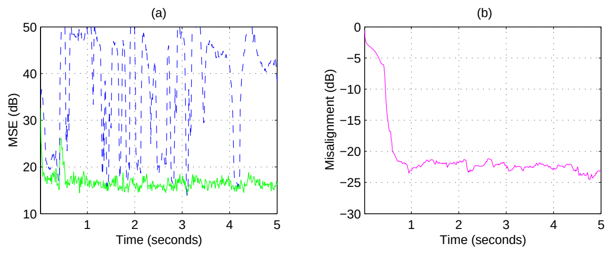

Fig. 2. Behavior of the APA algorithm with $\alpha = 0.2$ and $N = 10$. (a) MSE (–) as compared to the output signal level (– –). (b) Misalignment.

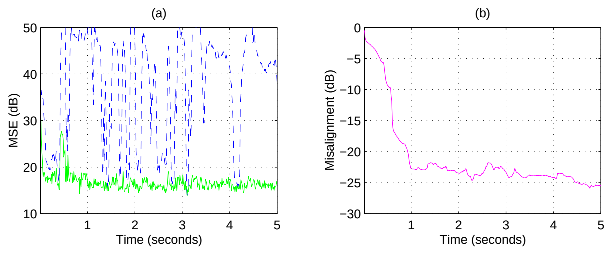

Fig. 3. Behavior of the SJ-DR-NLMS algorithm with $\alpha = 0.2$, $N = 10$, and normalized with $\mathbf{S}(n)$. (a) MSE (–) as compared to the output signal level (– –). (b) Misalignment.

algorithms. In Fig. 1, the NLMS algorithm is used with $\alpha = 0.5$. In Fig. 2, we used the APA with $\alpha = 0.2$ and $N = 10$. Figures 3 and 4 show the SJ-DR-NLMS when it is normalized respectively with $\mathbf{S}(n)$ and $\mathbf{S}^T(n)$; for both algorithms, we have chosen $\alpha = 0.2$ and $N = 10$. As expected, we can see that the APA and SJ-DR-NLMS perform better than NLMS. There is little difference between the APA and SJ-DR-NLMS. Also, the two versions of SJ-DR-NLMS are almost identical.

6. CONCLUSIONS

The way the data-reuse adaptive algorithms are presented is highly inefficient from a complexity point of view. There is always a strong need to rewrite these algorithms in a way that shows redundancy and make links with other existing algorithms. For example, we showed that the BENLMS is a block version of the SJ-DR-NLMS while it is also a block exact version of the NLMS. We also

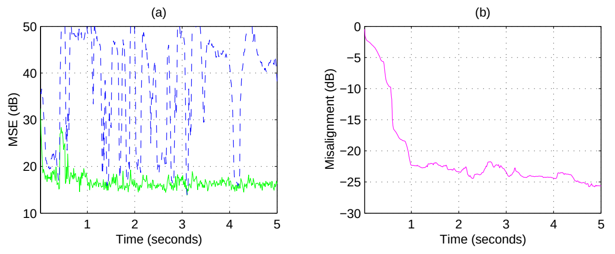

Fig. 4. Behavior of the SJ-DR-NLMS algorithm with $\alpha = 0.2$, $N = 10$, and normalized with $\mathbf{S}^T(n)$. (a) MSE (–) as compared to the output signal level (– –). (b) Misalignment.

showed that the SJ-DR-NLMS can be written in many other forms with different normalization, depending in which order the data is processed. Amazingly, even this change of normalization will not change the behavior of the algorithm. A data-reuse adaptive filter might be sometimes useful in practice but only if it can be implemented efficiently.

Our conclusions are that data-reuse algorithms present some interest in theory but are almost useless in practice if their structure can not be modified. Many existing algorithms that are able to perform better with much less complexity will be preferred. However, if a data-reuse algorithm can be rewritten in a way that shows redundancy and allows a derivation of an efficient version, it might find some practical interest.

7. REFERENCES

[1] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1985.

[2] S. Shaffer and C. S. Williams, “Comparison of LMS, alpha LMS, and data reusing LMS algorithms,” in *Conference Record of the Seventeenth Asilomar Conference on Circuits, Systems and Computers*, Nov. 1983, pp. 260–264.

[3] R. Nitzberg, “Application of the normalized LMS algorithm to MSLC,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 21, pp. 79–91, Jan. 1985.

[4] S. Roy and J. J. Shynk, “Analysis of the data-reusing LMS algorithm,” in *Proc. of the 32nd Midwest Symposium on Circuits and Systems*, 1990, pp. 1127–1130.

[5] S. Haykin, *Adaptive Filter Theory*. Third Edition, Prentice Hall, Englewood Cliffs, N.J., 1996.

[6] J. Benesty and P. Duhamel, “A fast exact least mean square adaptive algorithm,” *IEEE Trans. Signal Processing*, vol. 40, pp. 2904–2920, Dec. 1992.

[7] B. A. Schnaufer and W. K. Jenkins, “New data-reusing LMS algorithms for improved convergence,” in *Conference Record of the Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, Nov. 1993, pp. 1584–1588.

[8] K. Ozeki and T. Umeda, “An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties,” *Elec. Comm. Japan*, vol. J67-A, pp. 126–132, Feb. 1984.

[9] J.-S. Soo and K. K. Pang, “Multidelay block frequency domain adaptive filter,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 373–376, Feb. 1990.

[10] E. Moulines, O. Ait Amrane, and Y. Grenier, “The generalized multidelay adaptive filter: structure and convergence analysis,” *IEEE Trans. Signal Processing*, vol. 43, pp. 14–28, Jan. 1995.

[11] D. R. Morgan and S. K. Kratzer, “On a class of computationally efficient, rapidly converging, generalized NLMS algorithms,” *IEEE Signal Processing Lett.*, vol. 3, pp. 245–247, Aug. 1996.

[12] S. L. Gay and S. Tavathia, “The fast affine projection algorithm,” in *Proc. IEEE ICASSP*, 1995, pp. 3023–3026.