AN EFFICIENT BLOCK NEWTON-TYPE ALGORITHM

Kostas Berberidis and Sergios Theodoridis

Computer Technology Institute (C.T.I.)
P.O. Box 1122
26110 Patras
GREECE
E-mail: berberid@cti.gr

ABSTRACT

The algorithm presented in this paper is an exact block processing counterpart of the recently introduced Fast Newton Transversal Filtering (FNTF) algorithm [4]. The main trait of the new algorithm is that the block processing is done in such a way so that the resulting estimates are mathematically equivalent with the respective estimates of the FNTF algorithm. The required by the algorithm blocks can be much smaller than the filter length thus depending on the application the introduced processing delay can be negligible. In cases where the involved filter is of medium to long order the new algorithm offers a substantial saving in computational complexity without sacrificing performance.

1. INTRODUCTION

It is well-known that, among the various issues, concerning the performance of an adaptive filtering algorithm, computational complexity is of paramount importance for real time applications. The task becomes a critical issue in those applications where long filters with a few hundreds or even thousands of taps are involved. Acoustic echo cancellation is a typical application of the kind. Most of the existing recursive schemes (including the Fast Recursive Least Squares schemes [1, Ch. 5] and even the Least Mean Square algorithm in some cases) are disqualified from being used in such applications, with todays technology DSP processors. One line followed to tackle the complexity difficulties was via block adaptive schemes implemented either in the frequency or in the time domain [2], [3]. The step by step counterpart of most of the existing block algorithms is the Least Mean Squares (LMS) algorithm. Recently a significant effort is directed towards developing step-by-step adaptive algorithms whose performance ranges between LMS and Recursive Least Squares (RLS) algorithm, with a corresponding trade-off in complexity [1, Ch. 5]. The FNTF algorithm [4], belongs to this category of algorithms. Specifically, this algorithm exploits the fact that in many cases in practice almost all the predictable information about the input time series can be extracted with predictors whose order can be much lower than that of the corresponding filter. Thus the FNTF algorithm starts from a low order prediction problem of order p and extrapolates the gain vector (equivalently the autocorrelation matrix) from this low order problem up to the filtering order using a saddle point (min-max) approach. The complexity of the FNTF is $2m + 5p$, if the FAEST algorithm [1] is used in the prediction part.

When the filter is long ($m > 256$) the $O(m)$ part of the complexity of FNTF disqualifies the algorithm from being implemented in practice. This paper introduces a novel way to overcome this drawback by using block processing and computing the estimates once every L samples. This is done in such a way so that the resulting estimates are mathematically equivalent with the respective estimates of the FNTF algorithm. That is, the filter (computed every L time steps) as well as the filtering errors (computed for every time instant within the blocks) are exactly equal to those obtained by the step by step FNTF algorithm. Thus the derived algorithm is a Block Exact FNTF (BEFNTF). The involved computations are performed using fast convolutional schemes and the resulting saving in computational complexity is substantial (in the case of long filters is more than 80%). Note also that the size of the block L can be small compared to the filter length thus the processing delay is negligible.

The paper is organized as follows. In Section II the BEFNTF algorithm is developed. Computational issues concerning the new algorithm are discussed in Section III, where also simulation results verifying the performance of BEFNTF are presented.

2. DERIVATION OF THE ALGORITHM

Problem Formulation

A typical system identification problem is to represent the unknown system by a linear filter model based only on input-output observations. Let us denote as $\{x\}$ and $\{z\}$ the input and desired output sequences respectively. Then the taps $c_1, \ldots, c_m$ of the linear filter $\boldsymbol{c}_m$ are computed so that its output $\hat{z}(n)$ given by $\hat{z}(n) = -\sum_{i=1}^{m} c_i x(n-i+1)$ tracks in an optimal way the output $z(n)$ of the unknown system. In the RLS type of algorithms (a-posteriory error formulation) the

Sergios Theodoridis is also with the Dept. of Computer Eng. and Informatics, University of Patras, Patras 26500, GREECE.

0-7803-2431-5/95 $4.00

filter c_m is updated as follows

$$c_m(n) = c_m(n-1) + w_m(n)\epsilon_m(n) \quad (1)$$

where $w_m(n)$ and $\epsilon_m(n)$ are the gain vector and the a-posteriori filtering error respectively, defined as

$$w_m(n) = -\lambda^{-1} R_m^{-1}(n-1)x_m(n) \quad (2)$$

$$\epsilon_m(n) = z(n) + x_m^t(n)c_m(n) \quad (3)$$

where $x_m(n) = [x(n), \ldots x(n-m+1)]^t$ is the current input vector and matrix $R_m(n)$ is the sample input autocorrelation matrix. In the FNTF case this matrix is computed by properly extrapolating the *pth* order sample autocorrelation matrix $R_p(n)$ of the input [4]. Note that in this case $R_m^{-1}(n)$ turns out to be a banded matrix of width p. Writing recursion (1) in L successive time instants and combining the resulting equations we can express $c_m(n)$ in terms of $c_m(n-L)$ as follows

$$c_m(n) = c_m(n-L) + G_{m\times L}(n)\epsilon_L(n) \quad (4)$$

where $G_{m\times L}(n) = [w_m(n-L+1), \ldots, w_m(n)]$ and $\epsilon_L(n) = [\epsilon_m(n-L+1), \ldots, \epsilon_m(n)]^t$. The filter taps vector will be updated L steps ahead. To this end the block updating term $G_{m\times L}(n)\epsilon_L(n)$ must first be computed. This requires a fast computation of the exact filtering error vector $\epsilon_L(n)$ and a proper transformation of matrix $G_{m\times L}(n)$ (called henceforth gain matrix) so that the involved matrix by vector multiplication in the block updating term to be performed efficiently.

Computation of the filtering error vector
We first define the following error variable

$$e_{m,k}(n) = z(n) + x_m^t(n)c_m(n-k) \quad (5)$$

which is the a-priori filtering error at time n using the filter estimated at time $n-k$, i.e. $c_m(n-k)$. In a similar way we define as $e_{m,k}^w(n) = x_m^t(n-1)w_m(n-k)$ the a-priori error associated with the gain vector $w_m(n-k)$. Using the above definitions and Eq. (1) we get

$$e_{m,k-1}(n) = e_{m,k}(n) + \epsilon_m(n-k+1)e_{m,k}^w(n+1) \quad (6)$$

This is a recursive lattice-type formula going backwards with respect to k. For $k=1$ it yields the a-priori filtering error at time n. Using the step-up and step-down recursions of FNTF [4] the respective recursive relation for error $e_{m,k}^w(n)$ can be obtained. This recursion is given in Part 3.2 of Table I. The involved a-priori errors $e_{p,k}^f(n)$ and $e_{p,k}^b(n)$ are associated with the forward and backward predictors respectively. Note that these predictors are of order p while the filter is of order m. Recursive lattice-type relations for these prediction error variables can be similarly obtained using the update recursions of the respective predictors [1, Ch. 5]. In Parts 2.2 and 3.2 the lattice scheme for the prediction errors and the corresponding scheme for the filtering errors are summarized. The initialization phases of these schemes (Parts 2.1 and 3.1) can be formulated as FIR filtering problems and can be performed efficiently using fast convolution techniques. Note also that the filtering errors computed in Part 3.2 are in fact the a-priori errors. The corresponding a-posteriori filtering errors, i.e. the entries of the required error vector $\epsilon_L(n)$, are in turn computed in Part 3.3 of Table I.

Computation of the Block Updating Term
Direct multiplication of the $m\times L$ gain matrix $G_{m\times L}(N)$ by the $L\times 1$ filtering error vector $\epsilon_L(N)$ in Eq. (4) demands mL multiplications per block in order to perform the update recursion. Thus an efficient alternative is required. Let us first define the auxiliary vectors $s_{p+1}(n)$ and $u_{p+1}(n)$ (see Part 1.1 of Table I for their definitions). These two vectors involve the forward and backward prediction quantities respectively and they are computed at every time step. Then by combining the step-up and step-down recursions of FNTF we obtain the following expression for the gain vector

$$w_m(n) = \begin{pmatrix} 0 \\ w_m^{m-1}(n-1) \end{pmatrix} - \begin{pmatrix} s_{p+1}(n) \\ 0_{m-p-1} \end{pmatrix} + \begin{pmatrix} 0_{m-p} \\ u_{p+1}^p(n^d) \end{pmatrix} \quad (7)$$

where $n^d = n-m+p$ and x_M^{M-i} is the $(M-i)\times 1$ vector consisting of the first $M-i$ elements of the $M\times 1$ vector x_M. Now the above recursion will be used repeateadly in order to express all the gain vectors involved in gain matrix $G_{m\times L}(N)$ in terms of the gain vector $w_m(N-L+1)$. Thus after some algebra we get

$$G_{m\times L}(N) = W_{m\times L}(N) - S_{m\times L}(N) + U_{m\times L}(N) \quad (8)$$

where the $k-th$ column of matrix $W_{m\times L}(N)$ is

$$\begin{pmatrix} 0_{k-1} \\ w_m^{m-k+1}(N-L+1) \end{pmatrix}$$

Also the $k-th$ columns of matrices $S_{m\times L}(N)$ and $U_{m\times L}(N)$ are given by $\tilde{s}_m(N-L+k)$ and $\tilde{u}_m(N-L+k)$ respectively, defined as

$$\tilde{s}_m(N-L+k) = \sum_{i=0}^{k-2} \begin{pmatrix} 0_i \\ s_{p+1}(N-L+k-i) \\ 0_{m-p-1-i} \end{pmatrix} + 0_m$$

$$\tilde{u}_m(N-L+k) = \sum_{i=0}^{k-2} \begin{pmatrix} 0_{m-p+i} \\ u_{p+1}^{p-i}(N^d-L+k-i) \end{pmatrix} + 0_m$$

for $k = 1, 2, \ldots, L$. To compute the block updating term each one of the right hand side terms of Eq. (8) must be multiplied by vector $\epsilon_L(N)$, which has been computed as in the previous subsection. The first matrix by vector multiplication, due to the structure of $W_{m\times L}(N)$, can be performed using fast convolutional schemes. The other two matrix by vector products are computed directly. Note however that matrices $S_{m\times L}(N)\epsilon_L(N)$ and $U_{m\times L}(N)\epsilon_L(N)$ consist mainly of zero elements while their nonzero parts cover only an $O(pL)$ area (recall that in practice it is $p \leq L << m$).

The computation of matrices $S_{m\times L}(N)$ and $U_{m\times L}(N)$ by using the above definitions for their columns requires a large amount of additions. This complexity can be reduced by an order of magnitude by using the recursions given in Part 4.1 of Table I. Using these relations only $O(pL)$ additions per block (i.e. $O(p)$ additions per iteration) are required.

3. COMPLEXITY - SIMULATIONS

The main computational burden stems from the convolutions involved in Parts 2.1, 3.1 and 4.2.1 of the BEFNTF algorithm. Two distinct techniques for computing these convolutions can be followed. One performes the respective computations via time domain FIR filtering arguments and the other via FFT based operations. The relative computational efficiency of the two approaches depends on the specific values of the parameters involved in the problem, i.e. m, L and p. In this paper, due to the limited space we present only the former approach. The latter approach as well as more details on the derivation of the algorithm are presented in [6]. The time domain technique adopted for computing the convolutions is the one suggested in [5]. The main advantage of this approach is that it retains the basic FIR filtering structure (multiply-accumulate) which lends itself for DSP implementation [5].

Next we summarize the required number of multiplications per time instant for each part of the algorithm.

Part 1.1: This part is executed at each time instant. If the Stabilized FAEST as given in [1, Ch. 5] is used (only the prediction part) then $M_{1.1} = 6p$ mults per time instant are needed.

Part 2.1: For the three convolutions involved in this part the required number of mults per time instant is $M_{2.1} = 3\left(\frac{3}{2}\right)^q$, where $2^q = p$.

Part 2.2: The approximate complexity of the recursive scheme of this part is $M_{2.2} = 2L + 4p + \frac{2p^2}{L}$ mults per time instant.

Part 3.1: The complexity for the two convolutions in this part is approximately $M_{3.1} = 2\left(\frac{3}{2}\right)^l \frac{m}{L}$ mults per time instant, where $2^l = L$.

Part 3.2: The recursive scheme here has an approximate complexity equal to $M_{3.2} = \frac{1}{2}L$ mults per time instant.

Part 4.2.1: For the convolution of this part $M_{4.2.1} = \left(\frac{3}{2}\right)^l$ mults per time instant are needed.

Part 4.2.2: Performing directly the involved matrix by vector multiplication the respective complexity is $M_{4.2.2} = p + \frac{L}{2}$ mults per time instant.

Part 4.2.3: The matrix by vector multiplication in this part requires $M_{4.2.3} = p$ mults per time instant.

The total complexity of BEFNTF in multiplications per iteration (time instant) equals

$$M_{tot} = 12p + 3L + \left(\frac{3}{2}\right)^l \left(2\frac{m}{L} + 1\right) + 3\left(\frac{3}{2}\right)^q + \frac{2p^2}{L}$$

The longer the filter length the more the saving in computations offered by BEFNTF as compared to FNTF. For long filters ($m \geq 1024$) the complexity of BEFNTF is almost the 1/5 of that of FNTF. Note that in such a case its complexity is comparable to that of Fast Exact LMS [3] offering at the same time a performance very close to that of RLS.

Simulation Results

To show the performance of the algorithm the results of a typical system identification experiment are given in Fig. 1. A typical impulse response (truncated to 2048) of the acoustic echo path of a room was used as the unknown FIR system. The input time series was an AR stationary process of order 20 (corresponding to the greek vowel "ϵ"). A white gaussian noise was added to the system output resulting to an SNR=20dB. Three algorithms were used, namely, the RLS, the BEFNTF and the Normalized LMS (NLMS). Curve 1 (solid line) corresponds to the RLS algorithm with filtering order 2048 and forgetting factor $\lambda = 0.9998$. Curve 2 (dashed) corresponds to the BEFNTF algorithm whose filtering and prediction order was set equal to 2048 and 16 respectively, and the forgetting factor λ was equal to 0.9998. Curve 3 (dotted) corresponds to the NLMS algorithm with order 2048. The step size was chosen so as the algorithm to converge to the same error power level as that of RLS and BEFNTF. The results of this experiment show that the BEFNTF algorithm exhibits a performance very close to that of RLS while NLMS, as it was expected, converges much slower.

Other experiments with different system orders were also carried out and the results (reported in [6]) were similar to those presented here.

ACKNOWLEDGEMENTS

This work was supported by Computer Technology Institute, P.O. Box 1122, Patras 26110, CREECE. The authors would like also to thank Prof. G.V. Moustakides and Dr. E. Psarakis for useful discussions concerning fast convolutional schemes.

4. REFERENCES

[1] N. Kalouptsidis, S. Theodoridis, (Eds.), "Adaptive System Identification and Signal Processing Algorithms", *Prentice-Hall International (UK) Ltd*, 1993.

[2] J.J. Shynk, "Frequency-Domain and Multirate Adaptive Filtering", IEEE Signal Processing Magazine, pp. 15-37, Jan. 1992.

[3] J. Benesty, P. Duhamel, "A Fast Exact Least Mean Square Adaptive Algorithm", *IEEE Trans. on Signal Processing*, vol. 40, pp. 2904-2920, Dec. 1992.

[4] G.V. Moustakides, S. Theodoridis, "Fast Newton Transversal Filters - A New Class of Adaptive Estimation Algorithms", *IEEE Trans. on Signal Processing*, vol. 39, pp. 2184-2193, Oct. 1991.

[5] Z.J. Mou, P. Duhamel, "Fast FIR Filtering: Algorithms and Implementation", *Signal Processing*, vol. 13, pp. 377-384, Dec. 1987.

[6] K. Berberidis, S. Theodoridis, "A Block Exact Fast Newton Algorithm", *C.T.I. Technical Report #940420*, April 1994.

TABLE I : The Block Exact FNTF Algorithm

- Let $[N-L+1, N]$ be the current data block.

Part 1 : Sample by sample computations

1.1 For $n = N-L+1 : N$ using the prediction part of the Stabilized FAEST compute:

$$\boldsymbol{s}_{p+1}(n) = \frac{e_p^f(n)}{\lambda \alpha_p^f(n-1)} \begin{pmatrix} 1 \\ \boldsymbol{a}_p(n-1) \end{pmatrix}$$

$$\boldsymbol{u}_{p+1}(n) = \frac{e_p^b(n)}{\lambda \alpha_p^b(n-1)} \begin{pmatrix} \boldsymbol{b}_p(n-1) \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \boldsymbol{w}_m(n) \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \boldsymbol{w}_m(n-1) \end{pmatrix} - \begin{pmatrix} \boldsymbol{s}_{p+1}(n) \\ \mathbf{0}_{m-p} \end{pmatrix} + \begin{pmatrix} \mathbf{0}_{m-p} \\ \boldsymbol{u}_{p+1}(n^d) \end{pmatrix}$$

$$\alpha_m(n) = \alpha_m(n-1) + s_{p+1}^1(n) e_p^f(n) - u_{p+1}^{p+1}(n^d) e_p^b(n^d)$$

with $n^d = n - m + p$ and x^i the i^{th} element of $\boldsymbol{x}$

Part 2 : Block computation of the prediction errors

2.1 : Using fast convolution techniques compute the following seed errors:

- $e_{p,1}^f(N-L+1), \ldots, e_{p,L+p}^f(N+p)$
- $e_{p,1}^b(N-L+1), \ldots, e_{p,L+p}^b(N+p)$
- $e_{p,1}^w(N-L+1), \ldots, e_{p,L+p}^w(N+p)$

2.2 : Recursive scheme

For $n = N-L+2 : N+p$

For $k = n-N+L : -1 : 2$

$$e_{p,k}^w(n+1) = e_{p,k}^w(n) - s_{p+1}^1(n-k+1) e_{p,k}^f(n) + u_{p+1}^{p+1}(n-k+1) e_{p,k}^b(n)$$

$$e_{p,k-1}^f(n) = e_{p,k}^f(n) + \epsilon_p^f(n-k+1) e_{p,k}^w(n)$$

$$e_{p,k-1}^b(n) = e_{p,k}^b(n) + \epsilon_p^b(n-k+1) e_{p,k}^w(n+1)$$

End

$$e_{p,1}^w(n+1) = e_{p,1}^w(n) - s_{p+1}^1(n) e_{p,1}^f(n) + u_{p+1}^{p+1}(n) e_{p,1}^b(n)$$

End

Part 3 : Block computation of the filtering errors

3.1 : Using fast FIR filtering techniques compute the following seed errors

- $e_{m,1}(N-L+1), \ldots, e_{m,L}(N)$
- $e_{m,1}^w(N-L+1), \ldots, e_{m,L}^w(N)$

3.2 : Recursive scheme

For $n = N-L+2 : N$

For $k = n-N+L : -1 : 2$

$$e_{m,k}^w(n+1) = e_{m,k}^w(n) - s_{p+1}^1(n-k+1) e_{p,k}^f(n) + u_{p+1}^{p+1}(n^d-k+1) e_{p,k}^b(n^d)$$

$$e_{m,k-1}(n) = e_{m,k}(n) + \epsilon_m(n-k+1) e_{m,k}^w(n+1)$$

End

$$e_{m,1}^w(n+1) = e_{m,1}^w(n) - s_{p+1}^1(n) e_{p,1}^f(n) + u_{p+1}^{p+1}(n^d) e_{p,1}^b(n^d)$$

End

3.3:

$$\epsilon_L(N) = \begin{pmatrix} \alpha_m^{-1}(N-L+1) & \ldots & \alpha_m^{-1}(N) \end{pmatrix} \cdot \begin{pmatrix} e_m(N-L+1) \\ \vdots \\ e_m(N) \end{pmatrix}$$

Part 4 : Filter block updating

4.1 : Formation of the involved matrices

For $j = 1 : L-1$

$$\tilde{\boldsymbol{s}}_m(N-L+j+1) = \begin{pmatrix} 0 \\ \tilde{\boldsymbol{s}}_m^{m-1}(N-L+j) \end{pmatrix} + \begin{pmatrix} \boldsymbol{s}_{p+1}(N-L+j+1) \\ \mathbf{0}_{m-p-1} \end{pmatrix}$$

$$\tilde{\boldsymbol{u}}_m(N-L+j+1) = \begin{pmatrix} 0 \\ \tilde{\boldsymbol{u}}_m^{m-1}(N-L+j) \end{pmatrix} + \begin{pmatrix} \mathbf{0}_{m-p} \\ \boldsymbol{u}_{p+1}^p(N^d-L+j+1) \end{pmatrix}$$

End

- Form matrices $W_{m\times L}(N)$, $S_{m\times L}(N)$, $U_{m\times L}(N)$

4.2 : Computation of the block updating term

4.2.1 : Using fast convolution techniques perform the multiplication $W_{m\times L}(N)\epsilon_L(N)$

4.2.2 : Perform the multiplication: $S_{m\times L}(N)\epsilon_L(N)$

4.2.3 : Perform the multiplication: $U_{m\times L}(N)\epsilon_L(N)$

4.3 : Filter updating

$$\boldsymbol{c}_m(N) = \boldsymbol{c}_m(N-L) + W_{m\times L}(N)\epsilon_L(N) - S_{m\times L}(N)\epsilon_L(N) + U_{m\times L}(N)\epsilon_L(N)$$

FIGURE 1