A FAST O(N) ALGORITHM FOR ADAPTIVE FILTER BANK DESIGN

Omid S. Jahromi*, M. A. Masnadi-Shirazi* and M. Fu**

* Dept. of Electrical Engineering, School of Engineering, Shiraz University, Shiraz, 71345, Iran
** Dept. of electrical Engineering, The University of Newcastle, NSW, 2308, Australia

ABSTRACT

Designing optimal filter banks for subband coding applications has recently attracted considerable attention [1]-[5]. In particular, the authors have developed an adaptive algorithm based on stochastic gradient descent (SGD) that enables one to optimize two channel paraunitary filter banks in an on-line fashion [3]. The idea has also been extended to the case of tree-structured filter banks [4]. The computational complexity of the algorithm proposed in [3] is proportional to $N^2$ where $N$ is the number of stages in the paraunitary lattice. In this paper, we derive a fast algorithm which reduces the amount of computation to $O(N)$. We also show that the new algorithm can be implemented using an IIR lattice. Some issues regarding numerical stability of the IIR implementation are also discussed.

1. INTRODUCTION

A two channel paraunitary filter bank can be implemented using QMF lattices that insure perfect reconstruction irrespective of the specific choice of lattice coefficients [7]. For subband coding and some other applications, it is desirable that the filters have high coding gain [1], [5]. Coding gain, nevertheless, depends on the statistical properties of the input signal which is usually unknown or time varying. In [3] an adaptive algorithm was derived to adjust the coefficients of a QMF lattice to maximize its coding gain with respect to the signal at hand. A major part of that adaptive algorithm is computation of the instantaneous gradient of the output signal $y_0(n)$ with respect to the lattice coefficients $\theta_i$ (i.e., computation of $\Psi_i(n) = \frac{\partial y_0(n)}{\partial \theta_i}$ for $i=1...N$). It is shown in [3] that $N$ separate Gradient Computation Lattices (GCLs) are required to compute $\Psi_i(n)$ for $i=1..N$ where the $i$th GCL has $N-i$ stages. Therefore, the total amount of computation required to compute $\Psi_i(n)$ is $O(N^2)$.

In the next sections, we derive an algorithm that enables us to compute the gradient components using $O(N)$ computations per iteration. This fast algorithm uses the structural redundancies in the GCLs to derive a recursive algorithm for computation of $\Psi_i(n)$. The idea is partly stimulated by the method used in [6] to calculate the gradients for IIR adaptive lattices.

Notation: Vector and matrix functions are denoted by capital letters. $[A]_{ij} = (z)$ indicates the $ij$ element of the matrix (or vector) function $A(z)$.

2. THE ORIGINAL ALGORITHM

Consider the basic paraunitary lattice of Fig. 1. In this figure, $Y_i(z)$ is defined as a $2 \times 1$ vector whose elements denote the signals at the beginning of the $(i+1)$th rotation block (i.e., $U_{i+1}$) in the Z-transform domain. Using the notation defined in this figure, it is shown in [3] that:

$$\begin{bmatrix} Y_i(z) \\ L_i(z) \end{bmatrix} = \begin{bmatrix} \Psi_i(z) \\ \Theta_i(z) \end{bmatrix} = U_i \Phi(z) U_{i-1} \Phi(z) \ldots U_1 T(z) Y_{i-1}(z)$$

(1)

where $T(z) = \begin{pmatrix} 0 & 1 \\ -z^{-1} & 0 \end{pmatrix} \Lambda_i(z)$ denote the lower gradients (i.e., $\frac{\partial \Psi_i(n)}{\partial \theta_i}$) in the transform domain. The above relation may be written in a more compact form as:

$$\begin{bmatrix} Y_i(z) \\ L_i(z) \end{bmatrix} = U_N \prod_{j=i+1}^{N-1} (\Phi(z) U_j) \times T(z) Y_{i-1}(z) \quad 1 \leq i \leq N - 1$$

(2)

3. NEW FORMULATION

To begin with, we define the following transfer functions:

$$E_i(z) = \frac{Y_{i+1}(z)}{X(z)} = \left( \prod_{j=i+1}^{N} (\Phi(z) U_j) \right) \left( \frac{1}{z^{-1}} \right)$$

(3)
\[
P_i(z) = \frac{(Y_{ni})_i(z)}{Y_i(z)} = \begin{pmatrix} 1 & 0 \end{pmatrix} U_N \times \left( \prod_{j=N+1}^{i-1} (\Phi(z)U_j) \right) \quad (4)
\]
\[
Q_i(z) = \frac{(Y_{ni})_i(z)}{Y_i(z)} = \begin{pmatrix} 0 & 1 \end{pmatrix} U_N \times \left( \prod_{j=N+1}^{i-1} (\Phi(z)U_j) \right) \quad (5)
\]

Note that \(E_i(z)\) is a 2x1 transfer matrix while \(P_i(z)\) and \(Q_i(z)\) are 1x2 transfer matrices. Using (2)-(5) we can write \(\Psi_i(z)\) and \(\Lambda_i(z)\) as:
\[
\Psi_i(z) = P_i(z)T(z)E_i(z)X(z) \quad (6)
\]
\[
\Lambda_i(z) = Q_i(z)T(z)E_i(z)X(z) \quad (7)
\]

In the above relations, \(1 \leq i \leq N-1\) since computation of \(\Psi_n(z)\) and \(\Lambda_n(z)\) is trivial. Using (3)-(5) it is also possible to write the following recursive formulas:
\[
E_{i+1}(z) = \Phi(z)U_iE_i(z) \quad \text{for } 1 \leq i \leq N \quad (8)
\]
\[
P_i(z) = P_{i+1}(z)\Phi(z)U_{i+1} \quad \text{for } 0 \leq i \leq N-1 \quad (9)
\]
\[
Q_i(z) = Q_{i+1}(z)\Phi(z)U_{i+1} \quad \text{for } 0 \leq i \leq N-1 \quad (10)
\]

It is obvious that both \(U_i(z)\) and \(\Phi(z)\) are nonsingular.

So, (8) can also be written as:
\[
E_i(z) = U_i^{-1}(z)\Phi^{-1}(z)E_{i+1}(z) \quad \text{for } 1 \leq i \leq N \quad (11)
\]

in which \(\Phi^{-1}(z) = \begin{pmatrix} 1 & 0 \\ 0 & z \end{pmatrix} \), \(U_i^{-1} = \beta_i \begin{pmatrix} 1 & -\alpha_i \\ \alpha_i & 1 \end{pmatrix} \).

Using (3)-(5), it is also easy to show that the following boundary conditions hold in connection with (8)-(10):
\[
E_i(z) = \begin{pmatrix} 1 \\ z^{-1} \end{pmatrix}, \quad P_i(z) = \begin{pmatrix} 1 & 0 \\ 0 & z \end{pmatrix}^T, \quad Q_i(z) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^T \quad (12)
\]

In the following, for the sake of space conservation, we only consider computation of the upper gradient components (i.e. \(\Psi_i'(z)\)). The derivations can be easily modified for \(\Lambda_i(z)\) following the same procedure.

We define the 2x2 matrix \(V_i(z) = E_i(z)P_i(z)\). Then, using (9) and (11) we can write
\[
V_i(z) = U_i^{-1}\Phi^{-1}(z)V_{i+1}(z)\Phi(z)U_{i+1} \quad (13)
\]

Combining both \(\Phi(z)\) and \(\Phi^{-1}(z)\) with \(V_{i+1}(z)\), we can re-write the above equation as
\[
\begin{pmatrix} [V_i]_{11}(z) & [V_i]_{12}(z) \\ [V_i]_{21}(z) & [V_i]_{22}(z) \end{pmatrix} = U_{i+1}^{-1} \begin{pmatrix} [V_{i+1}]_{11}(z) & z^{-2}[V_{i+1}]_{12}(z) \\ z^{-2}[V_{i+1}]_{21}(z) & [V_{i+1}]_{22}(z) \end{pmatrix} U_{i+1} \quad (14)
\]

Using (6), \(\Psi_i(z)\) can be expressed in terms of the anti-diagonal elements of \(V_i(z)\). That is:
\[
\Psi_i(z) = ([V_i]_{12}(z) - z^{-2}[V_i]_{11}(z)) \times x(z) \quad (15)
\]

Equations (14) and (15) are the key results of this section. In particular, (14) indicates that the matrix \(V_i(z)\) can be obtained in terms of \(V_{i+1}(z)\) by means of a quadratic matrix operator \(\Xi_i\) where \(\Xi_i\) is defined so that \(\Xi_i(A) = U_{i+1}^{-1}AU_{i+1}\) for any 2x2 matrix \(A\). Note that a couple of two-sample delay (i.e. \(z^{-2}\)) and advance (i.e. \(z^{-1}\)) elements are also required.

The above idea is shown graphically in Fig. 2(a) in which \(\Xi_i\) is represented as a black-box. Internal details of \(\Xi_i\) are shown in Fig. 2(b). By cascading the operator blocks of Fig. 2(a) and using proper delays and advances between the blocks, it is possible to derive all transfer matrices \(V_i(z) (1 \leq i \leq N)\) starting from \(V_N(z)\). However, the structure of Fig. 2(a) is not practically computable, since the advance elements can not be realized in practice. To solve this problem we can reverse the flow of signals in the two lower branches of the structure. Doing so, we get a new bi-directional block that we call \(\Omega_i\). The new structure is depicted in Fig. 3(a) where its internal details are shown in Fig. 3(b). Using signal flow graph theorems one can easily verify that the two structures shown in Figs. 2(a) and 3(a) are mathematically equivalent.

Now, we should consider boundary relations. The first boundary condition in (12) implies that
\[
V_1(z) = E_1(z)P_1(z) = \begin{pmatrix} [P_1]_{11}(z) & [P_1]_{12}(z) \\ z^{-1}[P_1]_{11}(z) & z^{-1}[P_1]_{12}(z) \end{pmatrix} \quad (16)
\]

which in turn shows that
\[
[V_1]_{21}(z) = z^{-1}[V_1]_{11}(z)
\]
and
\[
[V_1]_{22}(z) = z^{-2}[V_1]_{11}(z) \quad (17)
\]

The above relations permit us to connect the two upper outputs of the rightmost block (i.e. \(\Omega_N\)) to its own lower inputs via two delay elements (See Fig. 4(a)). Now, to complete the structure, it is only required to provide proper input signals for the leftmost block (i.e. \(\Omega_{N-1}\)). This can be done easily. From (12) we can write:
\[ V_N(z) = E_N(z)P_N(z) = E_N(z) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} E_{N1}(z) \\ E_{N2}(z) \end{pmatrix} \]  

\[ E_N(z) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} E_{N1}(z) \\ E_{N2}(z) \end{pmatrix} \]  

Therefore, the input to \( \Omega_{N-1} \) block should be provided as shown in Fig. 4(b). Fortunately, \( E_N(z) \) is just the transfer function from X(z) to the output of the (N-1)th stage of the main adaptive lattice (See the definition of \( E(z) \) in (3)), hence it can be obtained from the main lattice as depicted in Fig. 5.

The complete structure for the new gradient computation scheme is also shown in Fig. 5. As seen in this figure, only N-1 bi-directional blocks are required to compute all the gradient components for the N stage adaptive lattice; hence, the computational complexity of this structure is proportional to N-1.

If the gradient components for the lower lattice output (i.e. \( \Lambda_1(z) \)) are also required, a derivation completely similar to the one given above can be used. The only difference is that, in all of the relations, we should use \( Q_i \) instead of \( P_i \). Also, instead of (18), the following relation should be used as the boundary condition at the rightmost stage:

\[ V_N(z) = E_N(z)Q_N(z) = \]

\[ E_N(z) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} E_{N1}(z) \\ E_{N2}(z) \end{pmatrix} \]  

\[ \text{(19)} \]

4. SOME CONSIDERATIONS

Two points should be noticed regarding the fast gradient computing method presented above:

1. All derivations are based on the assumption that the paraunitary lattice of Fig. 1 is time-invariant and the input signal \( x(n) \) is stationary. As a result, the new gradient computing algorithm is not exactly equivalent to the original one when the lattice coefficients are being adapted.

2. The new gradient computation structure contains feedback paths and hence is IIR. Using Mason’s gain formula, it can be shown that the determinant of the signal flow graph corresponding to the structure shown in Fig. 5 is equal to \( |E_N(z)| \). Therefore, it is inherently stable provided that, in the z-plane, all zeros of \( |E_N(z)| \) are located inside the unit circle.

In other words, the structure is stable if \( |E_N(z)| \) is a minimum phase transfer function. It is not difficult to show that a sufficient condition for this is that \( |\alpha_i| < 1 \) for all \( i \) [4]. This condition is not necessarily satisfied by the paraunitary lattices, however, in many practical designs \( |\alpha_i| \) is less than one for \( i \geq 2 \) [3], [4], [7]. So, a reliable performance is usually obtained.

5. CONCLUSION

We derived a fast gradient computing algorithm to reduce the computational complexity of adaptation from \( O(N^2) \) to \( O(N) \) for adaptive QMF lattices. In several computer experiments, no essential difference is observed between this algorithm and the one previously developed in [3]. Nevertheless, we leave it untouched to completely investigate the behavior of the proposed fast algorithm in various realistic applications. A thorough mathematical investigation of the convergence dynamics of this algorithm is also left open.

6. ACKNOWLEDGMENTS

The first author wishes to appreciate kind cooperation of Prof. B. Francis, Department of Electrical and Computer Engineering, University of Toronto, who provided him with appropriate computer facilities during the final stages of completion this work in Toronto, Canada.

7. REFERENCES


Fig. 1. Basic paraunitary lattice. $U_i = \begin{bmatrix} 1 & \alpha_i \\ -\alpha_i & 1 \end{bmatrix}$ are the lattice sells.

Fig. 2. (a) Computation of $V_i(z)$ in terms of $V_{i+1}(z)$ using the operator $\Xi_i$. (b) Details of implementing $\Xi_i$.

Fig. 3. (a) The bi-directional operator $\Omega_i$. (b) Details of implementing $\Omega_i$.

Fig. 4. (a) Applying boundary conditions to the left-most block (i.e. $\Omega_1$). (b) Boundary relations for the right-most block ($\Omega_N$).