Fast constant modulus adaptive algorithm

J. Benesty
P. Duhamel

Abstract: An exact block formulation of the constant modulus algorithm (CMA) is presented, in which a reduction of arithmetic complexity is achieved. Two types of fast algorithms are explained, either in time-domain or in frequency-domain. The first one is of greater interest for small block length and the second one, using the FFT as an intermediate step, has greater advantage for large blocks. Due to the equivalence between the original CMA formulation and this one, the convergence properties of the CMA are maintained, which is not the case in the Treichler et al. implementation in frequency domain of this algorithm. Furthermore, this approach allows the use of very small block lengths (e.g. $N = 2$), the reduction of the arithmetic complexity increasing with the block size.

1 Introduction

The constant modulus adaptive algorithm (CMA) is a special case of a more general algorithm that was first proposed by D.N. Godard [1] as a method for blind equalisation for data modems. Another similar algorithm was proposed by Benveniste et al. [2]. The CMA was extensively studied by Treichler et al. [3, 4] for a communication application. Indeed, in many modulation schemes, such as frequency modulation (FM) and phase modulation (PM), the signal to be transmitted possesses a constant envelope. The received signal, however, has lost this property due to multipath and interference effects. The CMA restores the constant envelope property of the signal and increases the SNR. This algorithm thus employs just the a priori knowledge about the envelope of the transmitted signal and has the nice characteristic that no reference signal is required.

Nevertheless, the CMA has some shortcomings. First, it involves the minimisation of a nonconvex cost function [1]. This non-convexity implies the existence of local minima, and a satisfactory convergence of the algorithm does not imply a true minimisation of the cost function. Secondly, the algorithm may capture a constant modulus interferer rather than the constant modulus signal of interest [4]. These two problems can be overcome by a simple filter initialisation [1, 4] and will not be considered here. Another drawback, is the large number of arithmetic operations required for this algorithm. To reduce this load, Treichler et al. [5] proposed to compute the nonlinear error in the time-domain while updating weights and filtering in the frequency-domain. Unfortunately, this algorithm is only an approximation of the initial one, and has been observed to have very slow convergence [5].

The main result of this paper is that it is possible to both reduce the arithmetic complexity of the CMA by working in blocks that may be very small, if required and to maintain convergence properties. From the mathematical point of view, the algorithm thus obtained is strictly equivalent to the CMA.

Section 2 briefly recalls the initial version of the CMA, and provides an evaluation of the arithmetic complexity in two cases of implementation. Section 3 provides the basis of our approach: merging the computations of two successive CMA outputs allowing a reduction of the required number of operations per output point.

This is generalised in Section 4 in which we establish an exact block formulation of the CMA on which a reduction of the arithmetic complexity is feasible by using the 'fast FIR' technique [6, 7]. Two special cases are studied in more detail. The first one is the recursive application of the fast FIR of length 2, which has the advantage of allowing in improvement of the arithmetic complexity even for very small block lengths. The second one, which uses FFT as an intermediate step, is more efficient for larger blocks.

Note that the blocksize never depends on the filter's length, and that the usual constraint that the FFT length should be at last twice the filter's length does not hold here. This fact has a lot of advantages when thinking of memory requirements or overall system delay.

2 Initial CMA

2.1 Derivation of the algorithm

The organisation of the constant modulus algorithm's is shown in Fig. 1, where $y(n)$ is the output of a complex FIR filter:

$$
y(n) = X^*_n H = H^T X_n = \sum_{k=0}^{L-1} x(n - k)h_k
$$

L is the length of the filter, $X_n$ the vector of the past $L$ complex data at time $n$, and $H$ the vector of complex

![Fig. 1 Overall organisation of the CMA](image)

Paper 8008F (ES), first received 25th May 1990 and in revised form 2nd January 1991

The authors are with CNET/PAR/APE, 38-40 rue du Général Leclerc, 92313 Issy-Les-Moulineaux, France

IEE PROCEEDINGS-F, Vol. 138, No. 4, AUGUST 1991
weights:
\[ X_s = [x(n) \ x(n-1) \ \cdots \ x(n - L + 1)]' \]
\[ H = [h_0 \ h_1 \ \cdots \ h_{L-1}]' \]

The purpose of the adaptation process is to find a weight vector \( H \) that minimises fluctuations in the complex envelope of the output \( x(n) \). Hence, a natural criterion \( J \) measures the distance between the modulus of \( x(n) \) and the constant modulus of the transmitted signal:
\[ J = \frac{1}{4} E[[|x(n)|^2 - 1]^2] \]

where \( E \) denotes statistical expectation and where the modulus of the signal is assumed to be equal to 1. A possible algorithm for the coefficient updating is as follows:
\[ H_{n+1} = H_n - \mu \nabla J(n) \]

where \( \mu \) is a positive step size and \( \nabla \) the gradient operator:
\[ \nabla J(n) = \frac{\partial J(n)}{\partial H_s} = E[[|x(n)|^2 - 1]|x(n)X_s'] \]

where * denotes complex conjugation.

Of course, since eqn. 4 involves a mathematical expectation, it cannot be used as it is. It has been proposed [1, 3] that it be replaced by an instantaneous gradient estimate, as given in eqn. 5.
\[ \nabla J(n) = [|x(n)|^2 - 1]|x(n)X_s^* \]

The CMA is thus described by the following set of equations:
\[ y(n) = H_s X_s \]
\[ z(n) = \mu[|x(n)|^2 - 1]|x(n) \]
\[ H_{n+1} = H_n - \alpha(n)X_s^* \]

2.2 Arithmetic complexity of the CMA

2.2.1 Initial version (CMA1): Assuming the usual 4 multi-2 add complex multiplication scheme is used, the arithmetic complexity of the initial CMA, as described by eqn. 6, is as follows: Eqn. 6a requires 4L real multiplications and 4L-2 real additions. The computation of eqn. 6b requires 5 real multiplications and 2 real additions, while eqn. 6c requires 4L real multiplications and 4L real additions. This results in the following total number of real operations per output point:

\[ 8L + 5 \text{ multiplications} \]
\[ 8L + 4 \text{ additions} \]

Note that since \( \mu \) has not been constrained to be a negative power of 2, it appears in this count.

2.2.2 'Fast' complex multiply-based version: It is well known that a complex multiplication can be computed by any of the following two eqns. 8:
\[ (x_s + jx_v)(h_r + jh_i) = [(x_s + x_v)h_r - x_s(h_r + h_i)] \]
\[ + [(x_s + x_v)h_i - x_v(h_r - h_i)] \] (9a)
\[ = [(h_r + h_i)x_s - h_i(x_s + x_v)] \]
\[ + [(h_r + h_i)x_v - h_i(x_s + x_v)] \] (9b)

In the case of fixed coefficient FIR filtering, eqn. 9a is preferred, because \( h_r \pm h_i \) can be precomputed, so that the overall computational load is 3 multiplications and 3 additions, which results in an exchange of one multiplication for one addition. When \( h_r \) and \( h_i \) are not fixed, the apparent cost is 3 multiplications, 5 additions. However, it is shown below that the use of eqn. 9b in the CMA eqns. 6a-c allows for a reduction in the total number of operations compared to the initial algorithm. Using eqn. 9b, eqn. 6 is rewritten as follows:
\[ y(n) = [[H_s^* + H_s^*X_s^* - (X_s^* + X_v^*)Y_s^*] + [(H_s^* + H_s^*X_s^* - (X_s^* - X_v^*)Y_s^*]] \]
\[ z(n) = \mu[|x(n)|^2 - 1]|x(n) \]
\[ H_{n+1} = H_n - [(\alpha(n) - \alpha(n)]X_s^* + \alpha(n)X_s^* \]
\[ H_{n+1} = H_n - [(\alpha(n) - \alpha(n)]X_s^* - \alpha(n)X_s^* - X_v^*] \]

A straightforward operation count would be as follows: 3L multiplications and 6L-1 adds in eqns. 10a, 5 multiplies and 2 adds in eqns. 10b and 3L multiplies and (4L + 1) adds in eqns. 10c.

Nevertheless, remembering that \((X_s^* + X_v^*)\) has a single additional term compared to \((X_s^* - X_v^*)\) and that identical terms are stored in the filtering process, it is easy seen (Fig. 2) that 2L-1 additions can be saved.

![Diagram](https://via.placeholder.com/150.png)

**Fig. 2** Efficient implementation of the complex filter found in the CMA in terms of real operations

The overall CMA process, based on a complex FIR scheme as depicted in Fig. 2 hence requires:

\[ 6L + 5 \text{ multiplications} \]
\[ 8L + 4 \text{ additions} \]

which reduces the total number of operations by about 2L. In other words, one fourth of the number of multiplications has been saved, at the cost of 25% additional memory locations in some implementations. This second algorithm will be referred to as the CMA2.

With these two versions of the CMA as starting points, we shall derive an exact block formulation of this algorithm which allows a reduction of the arithmetic complexity. The tools we use are the same ones as for the fixed coefficients case [7], and this derivation follows the same lines as for the LMS case [9], on which similar work was already performed. The main difference between the treatment of LMS and CMA algorithms are found in the type of signals (complex for the CMA, real-valued in Reference 9), and in the non-linearity of the instantaneous gradient expression eqn. 5. These differences do not bring major drawbacks for the derivation of the algorithms.

3 Example of CMA with reduced number of operations

Let us first consider the required computations in the CMA at two successive time samples \( n - 1 \) and \( n \). By appropriately rearranging the corresponding equations,
we shall obtain an exact equivalent of eqn. 6 requiring a lower number of operations per output point.

Consider eqn. 6, written at time $n-1$:

$$y(n-1) = X_{n-1}^T H_{n-1}$$

$$s(n-1) = \mu [y(n-1)]$$

$$H_n = H_{n-1} - \alpha(n-1)X_n^*$$

Substituting eqn. 13c into eqn. 6a results in:

$$y(n) = X_n^T H_{n-1} - \alpha(n-1)X_n^* X_{n-1}^T$$

$$= X_n^T H_{n-1} - \alpha(n-1)s(n)$$

where:

$$s(n) = X_n^* X_{n-1}^T$$

Eqs. 14 and 13a can be combined in matrix form, thus providing two successive outputs of the system:

$$\begin{bmatrix} y(n-1) \\ s(n-1) \end{bmatrix} = \begin{bmatrix} X_{n-1}^T \\ X_n^T \end{bmatrix} H_{n-1} - \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} \alpha(n-1) \\ s(n) \end{bmatrix}$$

The first term of this equation appears to be the computation of two successive outputs of a fixed coefficient filter. Thus, we can apply the fixed part of eqn. 15 the techniques explained in Reference 7.

$$\begin{bmatrix} X_{n-1}^T \\ X_n^T \end{bmatrix} H_{n-1} = \begin{bmatrix} x(n-1) x(n-2) \cdots x(n-L) \\ x(n) x(n-1) \cdots x(n-L+1) \end{bmatrix}$$

$$\begin{bmatrix} h_0(n-1) \\ h_1(n-1) \\ \vdots \\ h_{L-1}(n-1) \end{bmatrix}$$

$$\begin{bmatrix} h_0(n-1) \\ h_2(n-1) \\ \vdots \\ h_{L-2}(n-1) \end{bmatrix}$$

$$\begin{bmatrix} h_0(n-1) \\ h_2(n-1) \\ \vdots \\ h_{L-2}(n-1) \end{bmatrix}$$

$$\begin{bmatrix} x(n) x(n-2) \cdots x(n-L+2) \\ x(n-1) x(n-2) \cdots x(n-L+1) \end{bmatrix}$$

$$\begin{bmatrix} \alpha(n-1) \\ s(n) \end{bmatrix}$$

Now, the following set of equations is exactly equivalent to the definition of the CMA for a block of two outputs:

$$\begin{bmatrix} y(n-1) \\ y(n) \end{bmatrix} = \begin{bmatrix} A_1 \\ \alpha(n-1) \end{bmatrix}$$

$$\begin{bmatrix} \alpha(n) \\ \alpha(n) \end{bmatrix}$$

$$\begin{bmatrix} H_{0,1}^o + H_{0,1}^1 \\ H_{1,1}^o + H_{1,1}^1 \end{bmatrix}$$

Note that from a computational point of view, eqn. 20b gives a problem: the computation of $y(n)$ seems to require the knowledge of $\alpha(n)$ which itself is defined in terms of $y(n)$. Nevertheless, since the matrix involved in eqn. 20b is strictly lower triangular, $y(n)$ depends on $\alpha(n-1)$ and this equation can be solved as follows: $y(n-1)$ is readily obtained, compute $\alpha(n-1)$ by its definition (eqn. 13b), then obtain $y(n)$ by the second line of eqn. 20b, and finally compute $\alpha(n)$ from $y(n)$. Although $\alpha(n)$ and $\alpha(n)$ are related in a non-linear manner, this kind of equation will always be solvable by substitution, due to the nature of the matrix involved in eqn. 20b (strictly lower triangular).

(16)

(17)

(18)

(19)

(20a)

(20b)

(20c)

(21a)

(21b)

(21c)

(22a)

(22b)

(22c)
Step (a) computation of \( y(n-1) \) and \( y(n) \) from eqn. 21a.

Step (b) recursive computation of \( s(n) \):

\[
s(n) = (n-2) + [x(n)x(n-1) + x(n-1)x(n-2) - x(n-L)x(n-1-L)]
\]

\[
- x(n-L-1)x(n-L-2)
\]  

(22)

Step (c) \( y(n-1) = y(n) \). Compute \( a(n-1) \) from eqn. 13b, then substitute in eqn. 21b to get \( y(n) \), and finally use eqn. 6b to obtain \( a(n) \).

Step (d) compute the update of \( H = \text{eqn. 21c} \).

Step (e) incrementation of \( n \) by 2 then go to Step (a).

Several considerations similar to the ones in Reference 9, allow precise evaluation of the number of complex arithmetic operations involved in eqn. 21:

Step (a) The filtering operation \( A_1^*H_{n+1}^ + H_{n+1}^* \) is common to the two terms of eqn. 21a, which requires 3 length \( L/2 \) filters, two of which are applied to combinations of the input samples, namely:

\[
A_2 - A_4 = [x(n-2) - x(n-1) \cdots \ x(n-L) - x(n-L+1)]
\]

\[
A_1 - A_5 = [x(n-1) - x(n) \cdots \ x(n-L+1) - x(n-L+2)]
\]  

(23)

As in Section 2 the apparent number of additions involved in eqn. 23 can be reduced by noticing that the previous set of scalar products involved in the computation of \( y(n-3), y(n-2) \) already require nearly the same combinations of the input samples, and that only two new complex additions are needed: \( x(n-2) - x(n-1) \) and \( x(n-1) - x(n) \).

Step (b) involves two complex multiplications and two complex additions (half the complex multiplications were already computed previously).

Step (c) involves the computation of \( y(n) \) with one complex addition and one complex multiplication, plus two equations of the type eqn. 13b, which require a total of six multiplications.

Finally, Step (d) requires the complex product \( A_1^*(a(n-1) + a(n)) \), which is common to the two equations eqn. 21c. Moreover, \( (A_1 - A_0) \) and \( (A_2 - A_1) \) were already calculated in eqn. 21a.

The above considerations allow evaluation of the reduction in the number of complex operations per output point required for implementing the CMA. Nevertheless, the precise improvement in terms of real operations depends on the way the complex multiplications are performed (see Section 2.2).

When the complex multiplications are performed with the usual 4 multi-2 add scheme, the total number of operations for computing two outputs is:

- 12L + 22 real multiplications
- 14L + 22 real additions

We denote this algorithm by FCMA1, for a block length \( N = 2 \).

When the 'fast' complex multiply scheme is used for computing the length \( L/2 \) filters, as explained in Section 2.2.2, the corresponding algorithm is called FCMA2, and its computational load for a blocklength \( N = 2 \) is:

- 9L + 22 real multiplications
- 14L + 36 real additions

Table 1 gives the number of operations per output point for all the algorithms explained up to now: CMA1, CMA2, FCMA1 and FCMA2.

It can be observed that each 'fast' algorithm reduces by about 25% the number of multiplications compared to their initial counterpart, while slightly reducing the number of additions. The total number of real operations is seen to be 20% less than that required by a straightforward implementation of the CMA.

Note that this reduction is obtained only by a rearrangement of the initial equations, and that there is an exact equivalence, mathematically speaking, between the initial algorithm and our block version of it. Hence, all these algorithms have the same convergence rate.

The above explanations follow closely the work we have performed on the LMS algorithm. Nevertheless, because of the nonlinear error of the CMA, some equations (eqn. 21b) need more inspection than in the LMS case.

This method has been explained in a rather specific manner, by merging the computations of two successive CMA outputs. We show in the next Section that this approach is much more general: grouping the computations of more outputs results in greater computational savings.

4 Generalisation to arbitrary \( N \)

We shall follow the same lines as in Section 3. First, we provide an exact block formulation of the CMA for arbitrary \( N \), which is in the form of a fixed filtering followed by a correction of the outputs and an update of the coefficients that are to be used in the next block. For the fixed coefficients filtering, we shall refer essentially to References 6 and 7, and only recall some results. We shall rather concentrate on the adaptive part of this algorithm.

4.1 Exact block formulation of the CMA

Let us write the fixed FIR filter output equations at time \( n = N + 1, n + N + 2, \ldots, n - 1, n \):

\[
\begin{bmatrix}
y(n(N+1)) \\
y(n(N+2)) \\
\vdots \\
y(n-1) \\
y(n)
\end{bmatrix} =
\begin{bmatrix}
X_{n-N} \\
X_{n-N+2} \\
\vdots \\
X_{n-1} \\
X_n
\end{bmatrix}
\begin{bmatrix}
H_{n-N+1} \\
\vdots \\
H_1 \\
H_0
\end{bmatrix}
\]  

(28)

In the same manner as for the example \( N = 2 \), we may write the exact output equations at time \( n = N + 1, n + N + 2, \ldots, n - 1, n \) of the CMA:

\[
\begin{bmatrix}
y(n(N+1)) \\
y(n(N+2)) \\
\vdots \\
y(n-1) \\
y(n)
\end{bmatrix} =
\begin{bmatrix}
X_{n-N} \\
X_{n-N+2} \\
\vdots \\
X_{n-1} \\
X_n
\end{bmatrix}
\begin{bmatrix}
S(n) \\
\vdots \\
S(0)
\end{bmatrix}
\]  

(29)

Table 1: Comparison of the number of operations per output point of the CMA and the FCMA for a blocksize of \( N = 2 \)

<table>
<thead>
<tr>
<th>Number of operations</th>
<th>CMA1</th>
<th>CMA2</th>
<th>FCMA1</th>
<th>FCMA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of additions</td>
<td>6L</td>
<td>6L + 4</td>
<td>7L + 11</td>
<td>7L + 18</td>
</tr>
<tr>
<td>Number of multiplications</td>
<td>6L + 5</td>
<td>6L + 5</td>
<td>6L + 11</td>
<td>6L + 11</td>
</tr>
<tr>
<td>Total number of operations</td>
<td>14L + 5</td>
<td>14L + 9</td>
<td>13L + 22</td>
<td>11L + 5 + 29</td>
</tr>
</tbody>
</table>

IEE PROCEEDINGS-F, Vol. 138, No. 4, AUGUST 1991
with
\[
\begin{bmatrix}
0 & 0 & \cdots & 0 \\
s_1(n-N+2) & 0 & \cdots & 0 \\
\vdots & \vdots & & \vdots \\
s_{N-1}(n) & s_{N-1}(n) & \cdots & s_1(n)
\end{bmatrix}
\]
(30)

where
\[s_i(n) = X'_i X'_{i-1} \quad i = 1, 2, \ldots, N-1\]

The remaining part of the algorithm is the coefficient updating which is expressed as follows:
\[
H_{n+1} = H_{n-N+1} - \left[ X'_i X'_{i-1} \cdots X'_2 X'_1 \right] \left[ \begin{array}{c}
\alpha(n-N+1) \\
\vdots \\
\alpha(n-N+2) \\
\vdots \\
\alpha(n-1) \\
\alpha(n)
\end{array} \right]
\]
(31)

Eqs. 28, 29 and 31 can be written in matrix form as follows:
\[
Y'_n = X(n)H_{n-N+1}
\]
(32a)
\[
Y_n = Y'_n - S(n)\alpha_n
\]
(32b)
\[
H_{n+1} = H_{n-N+1} - X(n)\alpha_n
\]
(32c)

where \(\dagger\) denotes the transpose conjugate, \(Y'_n\) is a vector of \(N\) successive outputs of a fixed filter, \(X(n)\) represents \(N\) successive outputs of the complex CMA filter, \(\alpha(n)\) is a matrix \((N \times L)\) of the \(N\) last input vectors and \(\alpha_n\) the vector \((N \times 1)\) formed from \(\alpha(n-i)\) for \(i = 0\) to \(i = N-1\).

Eqs. 32 with the expressions relating \(\alpha(n)\) and \(Y_n\) (eqn. 6b at time \(n = N + 1, \ldots, n\)) forms an exact equivalent of eqn. 6 for a whole block of output \(y(n-N+1), \ldots, y(n)\). Eqn. 28 is an FIR filtering, whose coefficients remain unchanged during the whole block of outputs, and is thus amenable to a reduction of the arithmetic complexity through the techniques explained in Reference 7.

Eqn. 29 requires more inspection, since both vectors on each side of the equation depend on the same unknowns \(Y_n\) through eqn. 6b. Nevertheless, since \(S(n)\) is strictly lower triangular, eqn. 29 can be solved in a manner strictly parallel to the computation of the solution of a linear system with a lower-triangular matrix: First initialize \(\alpha(n-N+1) = y(n-N+1)\) then obtain \(\alpha(n-N+1)\) by eqn. 6b at time \(n - N + 1\). Solve in \(y(n-N+2)\) using the second line of eqn. 29 from which \(\alpha(n-N+1)\) is obtained (eqn. 6b). Then, \(\alpha(n-N+1)\) and \(\alpha(n-N+2)\) allow the computation of \(\alpha(n-N+3)\) from the third line of eqn. 29. Iterating the process provides both \(Y_n\) and \(\alpha_n\) in eqn. 32b. Eqn. 32c then provides the values of \(H\) to be used in the next iteration. It is seen that the filter weights are updated once per data block instead of once per data sample. Nevertheless, this updating is performed in such a manner that the weights are equal to those that could have been found in the initial CMA at the same time. The only drawback of this blockwise adaptation, apart from the explicit knowledge of a single weight vector per block, is a delay in the output, which is equal to the block length, hence is easily under control.

In this way, the expressions in eqn. 32 are exact block formulation of the CMA with the advantage that arithmetic complexity can be saved by using the same techniques as described in Reference 9.

4.2 Block Toeplitz formulation

Let us assume that \(L = NM\), where \(M\) is a positive integer. A formulation of eqns. 28-31 using subsampled versions of the different signals involved allows the derivation of the fast algorithm for any \(N\).

Define:
\[
A_j = \begin{bmatrix} \alpha(n-j) & \alpha(n-N-j) & \cdots \\
\vdots & & \vdots \\
\alpha(n-L-j) & \cdots & \alpha(n-L-N-j) \\
\end{bmatrix}
\]
(33)

a vector of length \(L/N\); and
\[
H_{n-N+1} = \begin{bmatrix} h_0 h_{k+1} & h_{k+1} h_{k+2} & \cdots & h_{k+L-1} \end{bmatrix}
\times (n-N+1)
\]
(34)

Then, eqn. 28 becomes:
\[
Y'_n = \begin{bmatrix} A_{N-1} & A_N & \cdots & A_{2N-3} & A_{2N-2} \end{bmatrix} H_{n-N+1}
\]
(35)

and eqn. 31 gives:
\[
\begin{bmatrix} H_{n+1}^N \cr H_{n+1}^N \cr \vdots \cr H_{n+1}^N \cr H_{n+1}^N \cr \vdots \cr A_{N-1}^* & A_{N-2}^* & \cdots & A_1^* & A_0^* \end{bmatrix} \begin{bmatrix} \alpha(n-N+1) \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\alpha(n-N+1) \\
\vdots \\
\alpha(n)
\end{bmatrix}
\]
(36)

Eqs. 35 and 36 play a key role for reducing the arithmetic complexity of the algorithm, since they can be seen as a filtering equation with elements replaced by vectors. Hence, all non-commutative fast FIR filtering algorithms such as those explained in References 6, 7 and 9 apply to this kind of matrix-vector product. Eqn. 29 has not been changed by this block formulation, and it seems that the most efficient way to compute the matrix \(S\) is the use of the following recursions. Eqn. 37 provides the expression of the first column of matrix \(S\):
\[
s_i(n-N+i+1) = s_i(n-N) + \sum_{j=0}^{i-1} x^*(n-N-j+1) \times x(n-L-N+i-j+1)
\]
(37)
Eqn. 38 provides the computations to be performed along the diagonals:

\[ s(n+1) = s(n) + x(n+1) \times (n-1) \]

The above considerations allow precise evaluation of the number of arithmetic operations required for operating this block-CMA, whatever the block size \( N \). It is important to note that the number of operations to be performed per output point can be decomposed into two terms. The first term is due to the 'fixed' coefficient filtering and to the update of the algorithm. This term decreases with \( N \), working with larger blocks results in more efficient algorithms. The second term is due to the computation of matrix \( S \) and this term increases with \( N \). Therefore, for a given filter length, there exist an optimum blocksize that also depends on the type of fast algorithm that is used. The next two Sections study two special cases of interest, where both the block-size and filter length are powers of two.

### 4.3 FCMA based on short-length FIR algorithms

The first case of interest is the recursive application of the computation used in Section 3. We showed that for a block size of \( N = 2 \), the fast algorithm requires three subfilters of length \( L/2 \). If each of these subfilters is in turn decomposed, the next iteration will result in nine subfilters subsampled by four. Hence, if the block length is a power of 2 (\( N = 2^p \)), a fast FIR algorithm can be obtained by applying \( n \) times the decomposition (eqn. 21a), thus resulting in \( 2^p \) subfilters of size \( L/N \), the inputs and outputs of which are sub-sampled by a factor \( N \) compared to the input of the system. A precise description of the resulting schemes is provided in Reference 6 in the case of fixed coefficient FIR filtering.

An evaluation of the number of operations required by this algorithm for computing a block of \( N = 2^p \) outputs is provided in the Appendix. It is shown that, if the 4 multi:2 add complex multiply scheme is used in the filtering part (FCMA1), the number of operations to be performed per output point for a filter of length \( L = 2^p M \) is:

\[ 8(3/2)^p M + 62^p - 1 \] multiplications

\[ 4(3/2)^p - 1)M + 72^p + 8(3/2)^p - 15 \] additions

If the fast complex multiply scheme is used (FCMA2, see Section 2.2.2), the resulting number of operations per output point are:

\[ 6(3/2)^p M + 62^p - 1 \] multiplications

\[ 4(3/2)^p - 1)M + 72^p + 12(3/2)^p - 13 - 2^p \] additions

Note that as long as \( M \geq 4 \) (i.e. the filter is at least four times as long as the block length) and whatever the block size may be, the FCMA requires fewer operations than the CMA. This means that a reduction of the arithmetic complexity is feasible even for such short filters as \( L = 8 \).

Furthermore, if we suppose that \( M = 2^p \), we see that the arithmetic complexity of FCMA varies with \( 0(3^p) \) instead of \( 0(4^p) \) for the CMA. This shows the efficiency of this approach.

The important point concerning these numbers is that the precise arithmetic complexity involves a term growing with \( N \) (updating of \( S \)) and another one diminishing with \( N \) (fast FIR). Hence, eqns. 39 and 41 have a minimum. The zeros of the derivations of these functions provide the approximate value of the optimum block length:

\[ n \approx -0.6 + 0.7 \log_2 L \text{ for FCMA1} \]

\[ n \approx -0.9 + 0.7 \log_2 L \text{ for FCMA2} \]

Table 2 provides a comparison of the number of operations per output point required by the various algorithms for the approximate optimum blocksize given by eqns. 43 and 44. A reduction by a factor of two of the total number of operations is seen to be very easily obtained for filters longer than \( L = 64 \), and a block length as small as \( N = 8 \).

### 4.4 FFT-based implementation of FCMA

It is well known that the FFT can be used for a fast implementation of an FIR filter, through the use of overlap-add or overlap-save techniques. Since eqn. 35 has the form of an FIR filter equation, the FFT technique can be applied. The main differences with the classical technique [5] are that the FFT length is twice the block length instead of twice the filter's length, and that sufficient care has been taken in the block formulation of the algorithm to maintain the rate of convergence of the CMA.

A simple way of understanding this method consists in extending the size of the block-Toeplitz matrix of eqn. 35 in such a way that the resulting matrix is cyclic. The resulting equation is:

\[ \begin{bmatrix} Y_s \\ \end{bmatrix} = \begin{bmatrix} T(n) & T(n) & H_{s-N+1} & 0 \\ \end{bmatrix} \]

where \( T(n) \) is the block-Toeplitz matrix of eqn. 35, a null vector of size \( N \), a set of outputs that do not need to be computed (overlap-save technique).

\[ T(n) = \begin{bmatrix} A_{N-1} & A_0 & \cdots & A_{N-3} & A_{N-2} \\ A_{2N-2} & A_{N-1} & \cdots & A_{N-3} & A_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ A_{N+1} & A_0 & \cdots & A_{N-2} \end{bmatrix} \]

Eqn. 45 involves inner products of the form \( A_i H_p \). Let us denote by \( C(n) \) the matrix \( 2N \times 2N \) made from the \( i \)th term of the blocks of matrix of eqn. 45. When developing all inner products in terms of the individual components eqn. 45 is rewritten as:

\[ \begin{bmatrix} Y_s \\ \end{bmatrix} = \sum_{i=0}^{(2N-1)} C(n) \begin{bmatrix} R_{i-N+1} \\ \end{bmatrix} \]

where

\[ R_{i-N+1} = [h_{n_1}, h_{n_1+1}, \cdots, h_{n_1+N-2}, h_{n_1+N-1}] \times (n-N+1) \]

Each matrix \( C(n) \) is cyclic, hence can be diagonalized by a Fourier matrix of size \( 2N \):

\[ \begin{bmatrix} Y_s \\ \end{bmatrix} = F_{2N} \sum_{i=0}^{(2N-1)} [F_{2N} C(n) F_{2N}^*] F_{2N} \begin{bmatrix} R_{i-N+1} \\ \end{bmatrix} \]

where

\[ F_{2N} C(n) F_{2N}^* = D(n) \]
is a diagonal matrix, whose elements are the DFT of the first column of $C(n)$.

Furthermore, it can easily be seen that $D_i(n) = D_{i-1}(n - N)$. This implies that eqn. 48 represents $2N$ complex filters of length $L/N$ in the Fourier domain, subsampled by a factor $l/N$. The overall organisation of this scheme is provided in Fig. 3. It is seen to require $(L/N) + 2$ FFTs of length $2N$ per block of data of size $N$, plus $2N$ complex filters of length $L/N$ which run at a rate divided by $N$.

![Diagram](image)

**Fig. 3** Implementation of the complex filter based on shorter FFT’s

The same kind of work has to be performed for the updating of the coefficients: eqn. 36 is first extended to become block-cyclic. Considering separately each $i$th term of the vector $H^i$ and $A_i$ results in the following set of equations:

$$
\begin{align*}
\begin{bmatrix}
R_i(n+1) \\
0
\end{bmatrix} &=
\begin{bmatrix}
R_i(n) \\
0
\end{bmatrix} - W C[n] a_i \\
& + W F_{2N} D_i(n) F_{2N}^{-1}
\end{align*}
$$

where $n = 0, 1, 2, \ldots, (L/N) - 1$

and finally:

$$
\begin{align*}
\begin{bmatrix}
R_i(n+1) \\
0
\end{bmatrix} &=
\begin{bmatrix}
R_i(n) \\
0
\end{bmatrix} - W F_{2N} D_i(n) F_{2N}^{-1}
\end{align*}
$$

where $D_i(n)$ is the complex conjugate of the matrix $D_i(n)$.

Eqns. 48 and 50, together with eqn. 29 are seen to represent the FFT-based implementation of the BCMA. A precise count of the required number of operations is provided in the Appendix and compared in Table 2 for a number of filter lengths of interest. It is seen that this method requires the lowest number of multiplications among all considered methods for lengths greater than $32$, and a lower number of operations (adds plus multiplies) above $L = 128$. For a filter of length $512$, the FFT-based implementation requires a number of operations divided by $4.5$ compared to the initial algorithm. Note that this performance is obtained for quite small blocklengths. Table 2 makes the distinction between two versions of FFT-based CMA, depending on the algorithm chosen for the complex filters of length $L/N$, as seen in Section 2.2.2. Note that the number of operations for both FFT-based algorithms are similar. Hence, the choice between them will rely on structural considerations.

5 Simulations

Some of these algorithms have been simulated, to verify our affirmations concerning:

(a) The exact equivalence between the CMA and the FCMA and
(b) the speed of our algorithm in relation to the initial one.

The complex input signal is (see Fig. 1):

$$x(n) = s(n) + \sigma(n) + \eta(n)$$

where $s(n) = \exp(j \phi(n))$ is the constant modulus transmitted signal ($|s(n)| = 1$) and $\eta = 1$, $\sigma(n) = \theta(n) - \tau$ the signal due to the effects of multipath propagation and $\eta(n)$ a zero-mean white noise. The objective is to use the CMA to provide an output $y(n)$ which is an estimation of the transmitted signal $x(n)$:

$$y(n) = \hat{x}(n)$$

Our aim here is not to concentrate on properties of the CMA itself, but to check the equivalence between the initial and fast versions of this algorithm. Fig. 4 provides the convergence curve $|\hat{x}(n)|$ averaged on 64 points of both algorithms CMA1 and FCMA1 in the case of a complex FIR filter of length $L = 256$ and a blocksize of $N = 32$. These curves are identical. As for the speeding up of the algorithm, we observed that the FCMA1 saved 40% computation time compared with the CMA1, which is nearly the ratio of the total number of operations, while exhibiting exactly the same convergence behaviour. This gives an indication on the accuracy issue of the FCMA; the updating of coefficients $\alpha_i$ being performed recursively, one may wonder if this recursivity could introduce any major drawback. We can show that in the LMS case, and for a fixed-point computation, this way of

![Diagram](image)

**Fig. 4** Error curve of the CMA and the FCMA

Table 2: Comparison of the number of operations per output point required by the various algorithms

<table>
<thead>
<tr>
<th>Filter</th>
<th>Number of additions</th>
<th>Number of multiplications</th>
<th>Number of additions</th>
<th>Number of multiplications</th>
<th>Number of additions</th>
<th>Number of multiplications</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>256</td>
<td>256</td>
<td>250</td>
<td>197</td>
<td>214</td>
<td>156</td>
</tr>
<tr>
<td>64</td>
<td>512</td>
<td>512</td>
<td>389</td>
<td>389</td>
<td>346</td>
<td>253</td>
</tr>
<tr>
<td>128</td>
<td>1024</td>
<td>1024</td>
<td>773</td>
<td>773</td>
<td>614</td>
<td>519</td>
</tr>
<tr>
<td>256</td>
<td>2048</td>
<td>2048</td>
<td>154</td>
<td>154</td>
<td>104</td>
<td>104</td>
</tr>
</tbody>
</table>

**Table 2:** Comparison of the number of operations per output point required by the various algorithms

computing only results in a slight increase of the residual error. In any realistic case, these errors cannot result in an instability of the algorithm. The same demonstration holds for the FCMA case, and will not be repeated here, due to lack of space.

It is interesting to note that the matrix S depends only on the input signal, and some approximations are feasible [9], depending on some knowledge of the input signal properties.

6 Conclusion

In this paper we provided a new algorithm which allows for a reduction of arithmetic complexity of the CMA. This reduction is possible whatever the blocksize is, and even for the smallest blocklength \( N = 2 \).

Furthermore, we showed that it was also possible to work in the frequency-domain and that the obtained algorithm is strictly equivalent to the initial CMA.

All these algorithms share the same advantages: same convergence as CMA with a lower arithmetic complexity, and small blocksize. The small blocksize allows the memory requirements to remain reasonable, and reduces the overall system delay.

The algorithms based on short-length FIR algorithms are efficient for very small blocklengths, while FFT-based algorithms are more efficient for medium size ones.

Furthermore, there is a possibility that the convergence rate can be improved using the FFT. This work is under consideration and will be reported.

7 Acknowledgments

The authors would like to thank S. Mayrargue and O. Rioul from the CNET for careful reading of the manuscript.

8 References


8 BLAHUTH, R.E.: ‘Fast algorithms for signal processing’ (Addison-Wesley, 1985)


9 Appendix: Arithmetic complexity of the proposed algorithms for blocklengths \( N = 2^n \)

The precise evaluation is cumbersome, and we provide here only partial results that can easily be checked. The total number of operations is derived from these partial results.

9.1 Based on short-length FIR filters

This algorithm turns the fixed coefficient filtering (eqn. 28) of length \( L \) into \( 3^n \) complex filters of length \( L/N \), that are used for computing a block of \( N \) outputs. The precise number of real operations required by these filters depends on the chosen type of implementation (see Section 2.2.2). This transformation is obtained by linear combinations of subsampled input sequences (recursive application of eqn. 35) which cost a total of

\[ 4^{3^n} \cdot N \] complex additions \( \text{(51)} \)

The second step is the correction of \( Y_e \) to obtain \( Y_c \). This first requires the computation of the elements of the matrix \( S(n) \), by application of eqns. 40 and 41. This requires

\[ 2N(N - 1) \] complex multiplications \( \text{(52)} \)

Once \( S(n) \) is obtained, eqn. 29 is solved by substitution, as explained in Section 4.1, which requires a total of

\[ N \] real multiplications

\[ N(N + 1) \] complex multiplications \( \text{(53)} \)

\[ N^2 \] complex additions

The final step is the updating of \( H \) to be used in the next block computation. This first requires the computation of \( X^T(n)Y_e \), which is computed in the same manner as \( Y_e \).

By taking into account the fact that the combinations of the input samples need not to be computed again, this requires:

\[ 2^{3^n}L \] complex multiplications \( \text{(54)} \)

Finally, once the impulse response is obtained, the linear combinations of the coefficients \( H^e \) need to be computed. This requires:

\[ 3^{3^n}L - N \] complex additions \( \text{(55)} \)

and the final computation of (eqn. 36) requires \( L \) more complex additions.

When the 4-mul 2-add complex multiplication scheme is used in the FIR filtering, the computation of a full block of outputs by the resulting algorithm (FCMA1) requires a total of:

\[ 3^{3^n}L/N + N(6N - 1) \] real mults

\[ 12^{3^n}L/N - 4L + 83^N + 7N - 15 \] real adds \( \text{(56)} \)

Finally, for the so-called FCMA2, where the complex filter scheme is that of Fig. 2, we obtain the following number of operations (for a full block of outputs):

\[ 6^{3^n}L/N + N(6N - 1) \] real mults

\[ 12^{3^n}L - 4L + 123^N + N(2N - 13) - 2 \] real adds \( \text{(57)} \)

9.2 FFT-based implementation

The overall organisation of the algorithm is the same one as before, the differences being found in the complex filtering scheme and in the updating of the coefficients:

In fact, the FFT scheme transforms the length-\( L \) filter into \( 2^n \) complex filters of length \( L/N \), at the cost of \( L/N \) length-\( 2^n \) FFT's for computing the weights, one FFT for
the determination of $D_i(n)$ (note that $D_i(n)$, $i = 1, 2, \ldots, \left\lfloor L/N \right\rfloor - 1$, have already been computed, since $D_i(n) = D_{\lfloor i/L \rfloor}(n - N)$) and one length-2N inverse FFT for recovering the outputs.

As for the updating of the weights, the overall computation, as given in eqn. 50 requires $(L/N) + 1$ length-2N FFT's plus $2L$ complex multiplications and $3L$ complex additions.

Furthermore, let us assume that the FFT is computed using the split radix algorithm [10], we obtain as a result:

For the computation of a set of $N$ outputs by the FFT-based FCMA1:

$$4L(\log_2 N + 2) + 8L/N + 6N^2$$
$$+ 6N \log_2 N - 13N + 12 \text{ real mults}$$

$$2L(6 \log_2 N + 7) + 8L/N + 7N^2$$
$$+ 18N \log_2 N - 9N + 12 \text{ real adds} \quad (58)$$

and for the computation of a set of $N$ outputs by the FFT-based FCMA2:

$$4L(\log_2 N + 1) + 8L/N + 6N^2$$
$$+ 6N \log_2 N - 13N + 12 \text{ real mults}$$

$$2L(6 \log_2 N + 7) + 8L/N + 7N^2$$
$$+ 18N \log_2 N - N + 12 \text{ real adds} \quad (59)$$