A Mathematical Framework to Describe Interpolated Adaptive Volterra Filters

Eduardo L. O. Batista, Orlando J. Tobias, and Rui Seara

Abstract—The major drawback for using adaptive Volterra filters in practical applications is the large number of adaptive filter weights required. Many research works can be found in the open literature aiming at reducing the computational burden of Volterra filters. Within this group, interpolated structures also have the purpose of reducing the computational overload. Such structures have been presented in an ad hoc manner in previous works. This paper proposes a formal mathematical framework to describe such a class of simplified adaptive Volterra filters. In addition, new insights on such structures are presented leading to a better understanding on interpolated adaptive Volterra filters. Numerical simulations are shown aiming to assess the performance of these computationally less-demanding structures of implementation.

Index Terms—Adaptive filters, interpolated structure, Volterra filters.

I. INTRODUCTION

ADUATE adaptive algorithms for nonlinear applications require a high computational complexity to implement them. However, the increasing processing power of modern digital signal processors (DSPs) makes it now feasible to implement such adaptive algorithms. In this context, Volterra adaptive filters have emerged for dealing with several nonlinear applications, such as active control of nonlinear noise processes [1], acoustic echo canceling [2], and identification and reduction of distortions in loudspeaker systems [3]. Additionally, the applicability of Volterra filters is still largely restricted to simplified versions. In this regard, a lot of research effort has been carried out to reduce the required number of adaptive weights of these filters. For instance, in [4], a parallel structure is proposed for modeling second-order Volterra systems. The complexity reduction in this case is achieved by just using the dominant coefficient of each stage. A simplified (truncated) Volterra filter has been proposed in [5], in which coefficients outside the main diagonal of the second-order block are disregarded. An approach presented in [6] uses a representation in the frequency domain in order to reduce the computational burden of such adaptive structures.

An interpolated structure is included in the class of reduced complexity implementations. In [7], two interpolated structures for implementing adaptive Volterra filters are proposed. The interpolated approach has been originally applied to fixed filters and later extended to adaptive linear filters [8]. For the latter application, the mathematics describing the adaptive algorithm has been well established in [8]. In the Volterra case however, it is only available as an ad hoc procedure. The main goal of this work is to present a mathematical framework, permitting to describe adaptive Volterra filters implemented by using an interpolated approach.

This paper is organized as follows. In Section II, a summarized description of the Volterra filter and its block-structure representation is discussed. Section III presents the foundations and motivation of interpolated structures by using finite impulse response (FIR) filters. Section IV describes mathematically the adaptive interpolated Volterra filter. Section V presents numerical simulations, including performance comparisons. Finally, the remarks and conclusions of this work are presented in Section VII.

II. STANDARD VOLterra FILTER

The mathematical treatment of standard Volterra filters is established in [9]. The input-output relationship of a causal and discrete Volterra filter is given by

\[ y(n) = \sum_{m=0}^{N-1} h_{m}(m) x(n-m) + \sum_{m_1=m_2=0}^{N-1} h_{m_1 m_2}(m_1, m_2) x(n-m_1) x(n-m_2) + \ldots \]  

(1)

where \( x(n) \) and \( y(n) \) represent the input and output signals, respectively; \( h_p(m_1, \ldots, m_p) \) denotes the \( p \)-th order coefficient; \( N \) is the memory size; and \( P \) is the filter order. An interesting feature drawn from (1) is that the Volterra filter can be viewed as a parallel of \( p \)-th order \( h_p \) blocks, with \( p = 1, 2, \ldots, P \) (see Fig. 1). For \( P \geq 2 \) one has the nonlinear blocks.

Manuscript received March 31, 2006. This work was supported in part by the Brazilian National Council for Scientific and Technological Development (CNPq).

Eduardo L. O. Batista, Orlando J. Tobias, and Rui Seara are with the LINSE - Circuits and Signal Processing Laboratory of the Department of Electrical Engineering at the Federal University of Santa Catarina, Florianópolis, Santa Catarina, Brazil (e-mails: dudu@linse.ufsc.br, orlando@linse.ufsc.br; seara@linse.ufsc.br).

...
Now, denoting the output of the \( p \)-th order block by \( y_p(n) \), we can rewrite (1) as
\[
y(n) = \sum_{p=1}^{P} y_p(n),
\]
with \( y_p(n) \) given by
\[
y_p(n) = \sum_{m_0=0}^{N-1} \sum_{m_1=0}^{N-1} \cdots \sum_{m_p=0}^{N-1} h_p(m_1, m_2, \ldots, m_p) \prod_{k=1}^{P} x(n-m_k).
\]

Such decomposition is also interesting for implementation purposes, permitting to establish strategies for complexity reduction by manipulating the whole filter or just the coefficient-demanding blocks. The input-output relationship of each block can also be described by a vector representation [10]. For a \( p \)-th order block, we have
\[
y_p(n) = x_p^T(n) h_p,
\]
where \( x_p(n) \) denotes the \( p \)-th order input vector and \( h_p \) the \( p \)-th order coefficient vector. Note that in (4) the output of each block of the Volterra filter is written as an inner vector with \( () \).

As described in [10], the coefficient-reduced realization of FIR filters [8]. The basic idea behind IFIR filters is to remove quite a few impulse response samples and recreate them through an interpolating filter. In Fig. 2, the block diagram of this filter class is illustrated, where \( w_s \) represents the sparse filter, \( i \) denotes the \( M \)-coefficient impulse response \( [i_0, i_1, \ldots, i_{M-1}]^T \) of the interpolator filter, \( y(n) \) represents the output signal of the sparse filter. The input signal and its interpolated version \( x(n) \) and \( x_i(n) \), respectively, are related by
\[
x_i(n) = \sum_{j=0}^{M-1} i_j x(n-j).
\]

The factor determining the FIR filter sparsity is termed interpolation factor and denoted here by \( L \) [8]. The sparse filter vector \( w_s \) is obtained by setting to zero \( (L-1) \) samples from each \( L \) consecutive ones from the original full vector \( w = [w(0) \ 0 \ \cdots \ w(N-1)]^T \). Thus,
\[
w_s = [w(0) \ 0 \ \cdots \ w(L-1) \ 0 \ \cdots \ w((N_s-1)L)]^T
\]
One can then rewrite (1), by using (13) and (14), as follows:
\[
y(n) = \sum_{p=1}^{P} x_p^T(n) h_p.
\]
Note that, from (10) and (11), the number of coefficients for each \( p \)-th order block \( D_p(N) \) is
\[
D_p(N) = N^p.
\]
input vector is now expressed as
\[ x_i(n) = \{x_i(n) \ x_i(n-1) \ x_i(n-2) \cdots x_i[n-(N_s-1)L]\}^T. \]
(23)
The input-output relationship of the sparse filter is then
\[ y(n) = x_s^T(n)w_s. \]  
Finally, (23) can be rewritten as
\[ x_s(n) = I^T u(n), \]  
(25)
with
\[ I = \begin{bmatrix} \hat{i}_0 & 0 & 0 & \cdots & 0 \\ \hat{i}_1 & \hat{i}_0 & 0 & \cdots & 0 \\ \hat{i}_2 & \hat{i}_1 & \hat{i}_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \hat{i}_{M-1} & \cdots & \hat{i}_0 \\
\end{bmatrix}, \]  
(26)
and
\[ u(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-M+2) \end{bmatrix}. \]  
(27)

Note that \( u(n) \) has dimension \( N+M-1 \), and matrix \( I \) is \( (N+M-1) \times N \) with elements taken from the interpolator filter \( [\hat{i}_0 \ \hat{i}_1 \ \cdots \ \hat{i}_{M-1}]^T \). By substituting (25) into (24), the input-output relationship of the interpolated structure becomes
\[ y(n) = u^T(n)Iw_s. \]  
(28)
From (28), it can be noted that the IFIR Volterra filter is equivalent to a FIR version with a memory of \( N+M-1 \) and an interpolated coefficient vector given by \( Iw_s \). For instance, by considering a filter with a memory of 3, \( L = 2 \), and \( I = [0.5 \ 1 \ 0.5]^T \), the sparse coefficient vector is then
\[ w_s = \begin{bmatrix} w(0) \\ 0 \\ w(2) \end{bmatrix}^T, \]  
(29)
and the interpolated coefficient vector is
\[ w_i = Iw_s = \begin{bmatrix} w(0) \\ 0.5w(0) + 0.5w(2) \\ w(2) \\ 0.5w(2) \end{bmatrix}^T. \]  
(30)
In this example, the procedure for recreating coefficients in the IFIR filter is the following one. The boxed coefficient in (29) has changed from 0 to the arithmetic mean of the neighboring coefficients, as indicated by the box in (30). The underlined coefficients in (30) arise from border effects when the interpolation is performed. Since usually \( N > N_s \), a significant reduction of the number of coefficients is obtained. Note that for practical cases, the border effect become negligible.

IV. INTERPOLATED ADAPTIVE Volterra FILTER

The idea behind the interpolated Volterra filter is to use a sparse Volterra filter having a reduced number of coefficients. An interpolator then attempts to recreate the removed samples of that filter. Thus, from (23), the first-order input vector of the sparse Volterra filter is represented as
\[ x_{1s}(n) = I^T u(n) \]  
(31)
with a corresponding sparse coefficient vector given by
\[ h_{1s} = \{h(0) \ 0 \ \cdots \ h(L) \ \cdots \ h[(N_s-1)L]\}^T. \]  
(32)
For the second-order block, from (8) and (31) the input vector is obtained as
\[ x_{2s}(n) = x_{1s}(n) \otimes x_{1s}(n) = I^T u(n) \otimes I^T u(n). \]  
(33)
One of the properties of the Kronecker product is the mixed-product rule [10], which establishes
\[ AC \otimes BD = (A \otimes B)(C \otimes D), \]  
(34)
with \( A, B, C, \) and \( D \) representing generic matrices. By applying (34) in (33), we obtain the second-order input vector. Then,
\[ x_{2s} = [u^T(n) \otimes u^T(n)](I \otimes I) = u_2^T(n)I_2, \]  
(35)
with
\[ u_2(n) = u(n) \otimes u(n), \]  
(36)
and
\[ I_2 = I \otimes I. \]  
(37)
The derivation process for the second-order sparse coefficient vector is better understood by expressing such a vector in a matrix form. For instance, a Volterra filter with a memory equal to 3 has the following second-order kernel:
\[ H_2 = \begin{bmatrix} h(0,0) & h(0,1) & h(0,2) \\ h(1,0) & h(1,1) & h(1,2) \\ h(2,0) & h(2,1) & h(2,2) \end{bmatrix}. \]  
(38)
Now, considering its sparse version with \( L = 2 \), one has
\[ H_{2s} = \begin{bmatrix} h(0,0) & 0 & h(0,2) \\ 0 & 0 & 0 \\ h(2,0) & 0 & h(2,2) \end{bmatrix}, \]  
(39)
or, equivalently, the following coefficient vector:
\[ h_{2s} = [h(0,0) \ 0 \ h(0,2) \ 0 \ 0 \ h(2,0) \ 0 \ h(2,2)]^T. \]  
(40)
By considering (35) and (40), the input-output relationship of the second-order block is given by
\[ y_2(n) = u_2^T(n)I_2h_{2s}, \]  
(41)
where the \( (N+M-1) \)-coefficient vector is
\[ h_{2s} = I_2h_{2s}. \]  
(42)
Thus, from (39), (42), and \( i = [0.5 \ 1 \ 0.5]^T \) we can express, for \( N = 3 \) and \( L = 2 \), the second-order coefficient in the following matrix form:
Note from (43), by disregarding again the border effects, how the coefficients are recreated after the interpolation process. For example, the center coefficient results from the arithmetic mean of four neighboring coefficients. Naturally, the border effect becomes less significant as the memory size increases. For the third-order block, we have the following expressions:

### a) Input vector

\[
x_{3s}(n) = I^T \mathbf{u}(n) \otimes x_{2s}(n)
\]

\[
= \left[ I^T \mathbf{u}(n) \right] \otimes \left[ I^T \mathbf{u}_2(n) \right]
\]

\[
= \left[ I^T \otimes I^T \right] \left[ \mathbf{u}(n) \otimes \mathbf{u}_2(n) \right]
\]

\[
= I^T_3 \mathbf{u}_3(n),
\]

with \( I_3 = I \otimes I_2 \) and \( \mathbf{u}_3(n) = \mathbf{u}(n) \otimes \mathbf{u}_2(n) \).

### b) Input-output relationship

\[
y_{3s}(n) = u^T_3(n) I_3 h_{3s}.
\]

The coefficient recreation process is better illustrated by considering Fig. 3. The cubic structure is because of the number of indexes that each third-order kernel presents. The black dots (at the cube vertices) represent the sparse third-order kernel, while the remaining gray dots are the ones recreated by interpolation. For example, the coefficient at the cube center (indicated by 1) results from the arithmetic mean of eight cube vertices; the coefficient indicated by 2 arises from the arithmetic mean of two vertical neighbors. In this case, we have used an interpolating filter given by \( I = [0.5 \ 1.0 \ 0.5]^T \).

![Diagram](image_url)

Fig. 3. Spatial representation of the coefficient recreation process for a third-order block.

Note that the second-order kernel can be represented by a plane structure. By generalizing for a \( p^p \)-order block, the input-output relationship is given by

\[
y_{ps}(n) = u^T_p(n) I_p h_{ps},
\]

with

\[
u_p(n) = \mathbf{u}(n) \otimes \mathbf{u}_{p-1}(n),
\]

and

\[
I_p = I \otimes I_{p-1}.
\]

For block order higher than three the coefficient recreation follows the same logic as presented and its graphical interpretation is no longer possible. Finally, the overall input vector is given by

\[
x_{Vi}(n) = \left[ u^T(n) \quad u^T_1(n) \quad \cdots \quad u^T_p(n) \right]^T,
\]

with its corresponding full coefficient vector given by

\[
h_{Vi} = \left[ h^T_1 I^T \quad h^T_2 I^T \quad \cdots \quad h^T_p I^T \right]^T.
\]

On the right side of Fig. 4, the equivalent interpolated Volterra filter denoted by \( h_{Vi} \) is illustrated. It is obtained from the association of interpolator \( I \) with sparse Volterra filter \( h_{Vs} \).

![Diagram](image_url)

Fig. 4. Block diagram of the interpolated Volterra filter.

Now, considering an interpolated Volterra filter adapted by the LMS algorithm, the coefficient update expression is then

\[
h_{Vs}(n+1) = h_{Vs}(n) + 2\mu e(n) x_{Vs}(n),
\]

with

\[
h_{Vs}(n) = [h^T_1(n) \quad h^T_2(n) \quad \cdots \quad h^T_p(n)]^T,
\]

and

\[
x_{Vs}(n) = [x^T_1(n) \quad x^T_2(n) \quad \cdots \quad x^T_p(n)]^T.
\]

In (51), \( e(n) \) denotes the error signal. Table 1 shows the number of coefficients to be adapted for both the standard Volterra filter and its interpolated version.

<table>
<thead>
<tr>
<th>Memory (N)</th>
<th>Order (P)</th>
<th>Interpolation factor (L)</th>
<th>Number of coefficients</th>
<th>Coefficient reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>44.44%</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>65</td>
<td>69.23%</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>2</td>
<td>350</td>
<td>70.29%</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>2</td>
<td>815</td>
<td>79.88%</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>3</td>
<td>5455</td>
<td>85.06%</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>4</td>
<td>5455</td>
<td>96.99%</td>
</tr>
</tbody>
</table>

By simply making sparse an ordinary adaptive Volterra filter one has a sub-optimal solution, resulting in a higher minimum MSE condition. The use of an interpolated approach amends the sparsity introduced, making the minimum MSE value to be kept within an acceptable range.
V. SIMULATIONS

To illustrate the behavior of the proposed approaches, some numerical simulations, considering a system identification problem, are presented. We consider two strategies for implementing an interpolated adaptive Volterra filter. The first one presented in Section IV, and another obtained by just interpolating the higher-order blocks (greater or equal to two), maintaining all coefficients of the linear block unchanged [7]. This strategy is reasonable since most of the applications present a predominant linear part and most coefficient-demanding blocks are the nonlinear ones. The proposed structures are compared with the standard Volterra structure (full) and some implementations of the simplified Volterra approach introduced in [5]. Here, the simple sparse structure is only considered to verify the effect of the interpolator filter in the process. The simplified version is a sparse-like implementation of a second-order Volterra filter. Such an approach consists of setting to zero the coefficients outside the main diagonal. By examining, for instance, the input-output relationship of the second-order block, one verifies that

\[ y_2(n) = \sum_{k=0}^{N-1} \sum_{m=0}^{K-1} h_2(m,m+k)x(n-m)x(n-m-k), \]  

(54)

where \( y_2(n) \) is the second-order block output signal, \( x(n) \) denotes the input signal, \( h_2(m,m+k) \) represents the second-order coefficients, and \( N \) is the memory size. In (54), the factor \( K \) determines the number of coefficients to be set to zero. By choosing \( K = N - 1 \) it results in the standard Volterra structure.

In all presented examples the plant follows the Wiener model illustrated in Fig. 5, consisting of a linear system followed by a memoryless nonlinearity. Such a system can model several practical situations such as acoustic echo path [13] and sensor saturation [14]. For all examples, the interpolation factor is \( L = 2 \), corresponding to interpolator \( I = [0.5 1.0 0.5]^T \). The measurement noise variance is \( \sigma_z^2 = 0.001 \) (SNR = 60 dB).

\[ x(n) \]  

\[ h(n) \]  

\[ d(n) \]  

Fig. 5. Block diagram of the Wiener model.

A. Example 1

In this example, the linear part of the plant is a FIR filter given by \([-0.05 -0.10 0.00 0.15 0.32 0.40 0.32 0.15 0.00 -0.10 -0.05]^T\], followed by a second-order nonlinear block whose output function is \( d(n) = y_1(n) + 0.3y_2^2(n) \), where \( y_1(n) \) is the linear block output. The input signal is white with unit variance. The step-size value used is \( \mu = 0.2 \mu_{\text{max}} \), where \( \mu_{\text{max}} \) is the maximum step-size value for algorithm convergence (experimentally determined). The number of coefficients in Table 2 are obtained for a plant having a memory size equal to 11.

The MSE curves are obtained from Monte Carlo simulations (average of 200 independent runs). In Fig. 6(a), the MSE curves obtained from a linear filter and the standard, sparse, interpolated, and partially interpolated Volterra structures, are shown. Fig. 6(b) depicts the MSE results from the partially interpolated Volterra filter compared with three implementations of the simplified Volterra filter from [5]. Fig. 6(a) shows the good performance of the IFIR Volterra structure as compared with the coefficient-demanding standard Volterra one. One observes a better performance of the structure that uses input interpolator in comparison with that of the sparse Volterra approach. Considering the implementations using the simplified Volterra filter [Fig. 6(b)], the partially interpolated Volterra structure shows a better performance.

### Table 2. Number of coefficients for Example 1

<table>
<thead>
<tr>
<th>Filter</th>
<th>Number of coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>11</td>
</tr>
<tr>
<td>Volterra</td>
<td>77</td>
</tr>
<tr>
<td>Sparse</td>
<td>27</td>
</tr>
<tr>
<td>Interpolated (proposed)</td>
<td>27</td>
</tr>
<tr>
<td>Partially interpolated (proposed)</td>
<td>32</td>
</tr>
<tr>
<td>Simplified (K = 2) [5]</td>
<td>32</td>
</tr>
<tr>
<td>Simplified (K = 3) [5]</td>
<td>41</td>
</tr>
<tr>
<td>Simplified (K = 4) [5]</td>
<td>49</td>
</tr>
</tbody>
</table>

![Fig. 6. Example 1. MSE curves. (a) Performance comparison of some Volterra implementations. (b) Performance comparison between some implementations with the simplified Volterra filter [5] and the partially interpolated one.](image)
B. Example 2

In this example, the linear part of the plant of Example 1 is changed to $[0.40 \ 0.20 \ 0.10 \ 0.15 \ 0.05 \ 0.10 \ 0.00 \ 0.30 \ -0.25 \ -0.02]^T$. The nonlinearity and step-size value remain unchanged. In contrast to the previous example, the linear part of the plant now has lower correlation between neighboring coefficients. Thus, it is expected that this fact affects the performance of the interpolated approach, since the interpolator works better with plants having a smooth impulse response. The simplified Volterra is implemented with 67 coefficients for $K = 7$, and 74 coefficients for $K = 9$. Fig. 7 illustrates the results obtained by Monte Carlo simulations (average of 200 independent runs). In this case, the interpolated approach exhibits a worse MSE performance as compared with the linear and sparse Volterra ones. This is due to the characteristics of the plant used. Observe the better performance of the partially interpolated structure as compared with that of the purely linear one (see Fig. 7(a)). By assessing the results obtained through the implementations considering the simplified Volterra approach and those with the proposed schemes [Fig. 7(b)], one verifies that the partially interpolated structure with $L = 2$ (32 coefficients) has a performance close to that of the simplified Volterra filter with 67 coefficients.

VI. REMARKS AND CONCLUSIONS

In this paper, a mathematical framework to describe interpolated structures to implement adaptive Volterra filters is presented. These structures lead to a significant computational overload reduction, opening the possibility of using adaptive Volterra filters in practical applications. From the presented simulations, it has been observed that in some cases the proposed algorithms exhibit an equivalent performance with respect to other simplified implementations along with a lower complexity. Thus, interpolated structures constitute an interesting alternative to implement adaptive Volterra filters.

REFERENCES