A FULLY LMS ADAPTIVE INTERPOLATED VOLTERRA STRUCTURE

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

LINSE – Circuits and Signal Processing Laboratory
Department of Electrical Engineering
Federal University of Santa Catarina
88040-900 – Florianópolis - SC – Brazil
E-mails: {dudu, orlando, seara}@linse.ufsc.br

ABSTRACT

The major drawback for using adaptive Volterra filters is the high computational complexity requirement. In this context, a large number of reduced complexity implementations have been proposed to increase the applicability of such filters. Contributing in this sense, this paper presents a fully LMS adaptive approach for implementing interpolated Volterra filters with a very good performance characteristic. The adaptive interpolated Volterra structure is a simplified version of the conventional one which adapts both the interpolator and the sparse filter. Numerical simulations illustrate the usefulness of the proposed approach.

***Index Terms*—**Adaptive filters, adaptive signal processing, interpolation, least mean square methods, nonlinear filters.

1. INTRODUCTION

Adaptive Volterra filters have become an interesting option for several nonlinear adaptive applications, such as active control of nonlinear noise processes [1], acoustic echo canceling, and reduction of distortions on loudspeaker systems, among others [2]. The increasing processing capacity of the modern digital signal processors (DSPs) has contributed widely in this context, permitting to partially overcome the main difficult, for implementing digital Volterra filters, that is, its high computational complexity. In addition, a significant research effort has been carried out aiming to obtain Volterra filter structures with a lower computational burden. Most of such approaches are based on simplified, sparse, and truncated structures [3], [4], or even on the frequency domain [5]. The interpolated structures constitute a class of reduced complexity Volterra implementations [6], which have been originally considered to implement linear FIR filters [7]. The idea then is to use a filter cascade composed of a sparse filter, with a reduced coefficient number, and an interpolator filter, whose purpose is to recreate the missing coefficients in the sparse filter [6], [7]. In the Volterra case, the interpolation is performed with an input linear FIR interpolator filter followed by an output sparse Volterra filter [6]. In the adaptive version, only the coefficients of the sparse filter are adapted; the interpolator coefficients are maintained fixed [6]. However, the use of a fixed interpolator often leads to a poor performance. Then, aiming to overcome such a drawback, this paper proposes a fully adaptive interpolated Volterra structure using the least-mean-square (LMS) algorithm to adapt the filters' coefficients.

This paper is organized as follows. Section 2 presents the Volterra filter and its main characteristics. In Section 3, the interpolated Volterra filter is discussed. In Section 4, the fully LMS adaptive interpolated Volterra structure is derived. Section 5 shows some results of numerical simulations. Finally, Section 6 presents the conclusions of this research work.

This work was supported in part by the Brazilian National Research Council for Scientific and Technological Development (CNPq).

2. VOLTERRA FILTER

The input-output relationship of a causal and discrete Volterra filter is given by [2]

$$\begin{aligned} y(n) = & \sum_{m_1=0}^{N-1} h_1(m_1)x(n-m_1) \\ & + \sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1} h_2(m_1,m_2)x(n-m_1)x(n-m_2) + \dots \\ & + \sum_{m_1=0}^{N-1}\cdots\sum_{m_P=0}^{N-1} h_P(m_1,\cdots,m_P)x(n-m_1)\cdots x(n-m_P) \end{aligned} \quad (1)$$

where $x(n)$ is the input signal, $y(n)$ is the output signal, $h_p(m_1,\dots,m_p)$ denotes the p^{th}-order coefficient, N is the memory size, and P is the filter order. As described in [1], (1) can be rewritten as

$$y(n) = \sum_{p=1}^{P} y_p(n) \quad (2)$$

with

$$y_p(n) = \sum_{m_1=0}^{N-1}\sum_{m_2=0}^{N-1}\cdots\sum_{m_p=0}^{N-1} h_p(m_1,m_2,\dots,m_p)\times\prod_{k=1}^{p} x(n-m_k)\ . \quad (3)$$

Note, from (1) and (2), that the Volterra filter can be seen as a parallel block structure composed of a first-order (linear block) h_1 and nonlinear blocks with orders ranging from 2 to P. The input-output relationship for each block, given by (3), can also be described in a vector representation [2]. Thus,

$$y_p(n) = \mathbf{x}_p^{\mathrm{T}}(n)\mathbf{h}_p \quad (4)$$

where $\mathbf{h}_p$ denotes the p^{th}-order coefficient vector and $\mathbf{x}_p(n)$ is the p^{th}-order input vector. The latter is obtained as follows [2]:

$$\mathbf{x}_p(n) = \mathbf{x}_1(n)\otimes\mathbf{x}_{p-1}(n) \quad (5)$$

with

$$\mathbf{x}_1(n) = [x(n)\ \ x(n-1)\ \ \cdots\ \ x(n-N+1)]^{\mathrm{T}} \quad (6)$$

where $\otimes$ denotes the Kronecker product.
By defining the Volterra filter input vector as

$$\mathbf{x}_{\mathrm{V}}(n) = [\mathbf{x}_1^{\mathrm{T}}(n),\ \mathbf{x}_2^{\mathrm{T}}(n),\dots,\ \mathbf{x}_P^{\mathrm{T}}(n)]^{\mathrm{T}} \quad (7)$$

and the Volterra coefficient vector by

$$\mathbf{h}_{\mathrm{V}} = [\mathbf{h}_1^{\mathrm{T}},\ \mathbf{h}_2^{\mathrm{T}},\dots,\ \mathbf{h}_P^{\mathrm{T}}]^{\mathrm{T}} \quad (8)$$

expression (1) can be rewritten as

$$y(n) = \mathbf{x}_{\mathrm{V}}^{\mathrm{T}}(n)\mathbf{h}_{\mathrm{V}}. \quad (9)$$

The number of coefficients for each p^{th}-order block is given by

$$D_p(N) = N^p. \tag{10}$$

Therefore, the number of coefficients of the Volterra filter is

$$D_{\text{V}}(N,P) = \sum_{k=0}^{P} N^k. \tag{11}$$

As described in [1], the causal Volterra filter can be implemented by taking into account the symmetry of the kernels. As a consequence, the number of coefficients for each block can then be reduced to

$$D'_p(N) = \frac{(N+p-1)!}{(N-1)!\,p!} \tag{12}$$

and the total number of coefficients to

$$D'_{\text{V}}(N,P) = \frac{(N+P)!}{N!P!} - 1. \tag{13}$$

From (10) and (11), or even from (12) and (13), one can easily verify that the computational complexity required to implement a Volterra filter is very high. Furthermore, the number of coefficients grows exponentially with memory size. Thus, depending on both the required memory size and the order, the implementation of the Volterra filter may even be unfeasible.

3. INTERPOLATED VOLTERRA STRUCTURE

The block diagram of an interpolated Volterra structure is presented in Fig. 1. In this figure, $x(n)$ and $\hat{y}(n)$ represent the input and output signals, respectively, while $\mathbf{i}$ represents an interpolator filter with memory size M and coefficient vector $\mathbf{i} = [i_0 \;\; i_1 \;\; \cdots \;\; i_{M-1}]^{\text{T}}$. The sparse Volterra filter is denoted by $\mathbf{h}_{\text{Vs}}$ and its block structure is highlighted by the dashed box. Each p^{th}-order sparse block is denoted by $\mathbf{h}_{ps}$ with output signal given by $\hat{y}_p(n)$. Vectors $\tilde{\mathbf{x}}_p(n)$ represent the p^{th}-order interpolated input vectors, obtained similar to (5), but now considering the interpolated input signal $\tilde{x}(n)$.

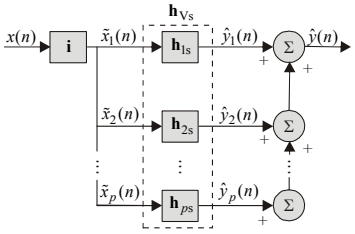

Fig. 1. Block diagram of an interpolated Volterra structure.

The sparse first-order coefficient vector of the interpolated Volterra filter is obtained by setting to zero $L-1$ of each L coefficients of the conventional Volterra first-order block [6], [7], which by considering $\mathbf{h}_{\text{l}} = [h_{\text{l}}(0) \;\; h_{\text{l}}(1) \;\; \cdots \;\; h_{\text{l}}(N)]^{\text{T}}$ results in

$$\mathbf{h}_{1s} = \{h_{\text{l}}(0) \;\; 0 \;\; \cdots \;\; h_{\text{l}}(L) \;\; 0 \;\; \cdots \;\; h_{\text{l}}[(N_{\text{s}}-1)L] \;\; 0 \;\; \cdots \;\; 0\}^{\text{T}}. \tag{14}$$

Thus, the number of nonzero coefficients in (14) is

$$N_{\text{s}} = \lfloor (N-1)/L \rfloor + 1 \tag{15}$$

where $\lfloor \cdot \rfloor$ represents the truncation operation.

The remaining p^{th}-order sparse coefficient vectors are obtained as described in [6]. The first-order input vector is given by

$$\tilde{\mathbf{x}}_1(n) = [\tilde{x}(n) \;\; \tilde{x}(n-1) \;\; \tilde{x}(n-2) \;\; \cdots \;\; \tilde{x}(n-N+1)]^{\text{T}}. \tag{16}$$

By considering that

$$\tilde{x}(n) = \mathbf{x}_{\text{M}}^{\text{T}}(n)\mathbf{i} \tag{17}$$

with

$$\mathbf{x}_{\text{M}}(n) = \left[x(n) \;\; x(n-1) \;\; x(n-2) \;\; \cdots \;\; x(n-M+1)\right]^{\text{T}} \tag{18}$$

expression (16) is rewritten as

$$\tilde{\mathbf{x}}_1(n) = \mathbf{X}_1^{\text{T}}(n)\mathbf{i} \tag{19}$$

for

$$\mathbf{X}_1(n) = [\mathbf{x}_{\text{M}}(n) \;\; \mathbf{x}_{\text{M}}(n-1) \;\; \cdots \;\; \mathbf{x}_{\text{M}}(n-N+1)]. \tag{20}$$

Thus, the output of the sparse first-order block is

$$\hat{y}_1(n) = \tilde{\mathbf{x}}_1^{\text{T}}(n)\mathbf{h}_{1s} = \mathbf{i}^{\text{T}}\mathbf{X}_1(n)\mathbf{h}_{1s}. \tag{21}$$

The input vectors for the other blocks are obtained recursively by

$$\tilde{\mathbf{x}}_p(n) = \tilde{\mathbf{x}}_1(n) \otimes \tilde{\mathbf{x}}_{p-1}(n). \tag{22}$$

By considering (17), (22), and the Kronecker mixed-product rule [2], the output of the sparse second-order block can be written as

$$\begin{aligned} \hat{y}_2(n) &= \tilde{\mathbf{x}}_2^{\text{T}}(n)\mathbf{h}_{2s} = [\mathbf{i}^{\text{T}}\mathbf{X}_1(n) \otimes \mathbf{i}^{\text{T}}\mathbf{X}_1(n)]\mathbf{h}_{2s} \\ &= (\mathbf{i}^{\text{T}} \otimes \mathbf{i}^{\text{T}})\left[\mathbf{X}_1(n) \otimes \mathbf{X}_1(n)\right]\mathbf{h}_{2s} = \mathbf{i}_2^{\text{T}}\mathbf{X}_2(n)\mathbf{h}_{2s} \end{aligned} \tag{23}$$

with $\mathbf{i}_2 = \mathbf{i} \otimes \mathbf{i}$ and $\mathbf{X}_2(n) = \mathbf{X}_1(n) \otimes \mathbf{X}_1(n)$.

Now generalizing, the output of the p^{th}-order sparse block is

$$\hat{y}_p(n) = \mathbf{i}_p^{\text{T}}\mathbf{X}_p(n)\mathbf{h}_{ps} \tag{24}$$

with

$$\mathbf{i}_p = \mathbf{i} \otimes \mathbf{i}_{p-1} \tag{25}$$

and

$$\mathbf{X}_p(n) = \mathbf{X}_1(n) \otimes \mathbf{X}_{p-1}(n). \tag{26}$$

Then, considering

$$\tilde{\mathbf{x}}_{\text{V}}(n) = [\mathbf{i}^{\text{T}}\mathbf{X}_1(n) \;\; \mathbf{i}_2^{\text{T}}\mathbf{X}_2(n) \;\; \cdots \;\; \mathbf{i}_P^{\text{T}}\mathbf{X}_P(n)]^{\text{T}} \tag{27}$$

and

$$\mathbf{h}_{\text{Vs}} = [\mathbf{h}_{1s}^{\text{T}} \;\; \mathbf{h}_{2s}^{\text{T}} \;\; \cdots \;\; \mathbf{h}_{Ps}^{\text{T}}]^{\text{T}} \tag{28}$$

the output of the interpolated Volterra filter is given by

$$\hat{y}(n) = \sum_{p=1}^{P} \hat{y}_p(n) = \tilde{\mathbf{x}}_{\text{V}}^{\text{T}}(n)\mathbf{h}_{\text{Vs}}. \tag{29}$$

The interpolated Volterra filter can also be implemented by considering sparsity and interpolation only in the nonlinear blocks [6]. Such an approach is known as the partially interpolated Volterra (PIV) structure, presenting almost the same number of coefficients of the fully interpolated Volterra (FIV) approach, since the nonlinear blocks are the most coefficient demanding ones [6].

4. FULLY LMS ADAPTIVE INTERPOLATED VOLTERRA STRUCTURE

The implementations of adaptive interpolated Volterra structures [6] [adaptive fully interpolated Volterra (AFIV) structure and adaptive partially interpolated Volterra (APIV) structure] are based on adapting only the sparse filter [6] in the structure. The advantage of such an approach is that all standard adaptive algorithms can be considered. However, the use of a fixed input interpolator results in a poor performance. Then, to make use of an adaptive interpolator may improve such a situation. In this section, the expressions for a fully LMS adaptive interpolated Volterra structure are derived. Starting from a similar approach to [8], the error signal is given by

$$e(n) = d(n) - \hat{y}(n) = d(n) - \sum_{p=1}^{P} \hat{y}_p(n) = d(n) - \tilde{\mathbf{x}}_{\text{V}}^{\text{T}}(n)\mathbf{h}_{\text{Vs}} \tag{30}$$

where $d(n)$ represents the signal to be estimated by the adaptive filter. The cost function for deriving the LMS algorithm is the instantaneous estimate of the mean-square error (MSE), given by

$$\hat{J}_{\mathrm{MSE}}(n) = e^2(n). \tag{31}$$

The coefficients of the interpolated structure are updated proportionally by using the gradient of the cost function (31) [8], [9]. Thus, for the sparse Volterra filter the following update expression is obtained:

$$\mathbf{h}_{\mathrm{Vs}}(n+1) = \mathbf{h}_{\mathrm{Vs}}(n) - \mu_{\mathrm{V}} \nabla_{\mathbf{h}_{\mathrm{Vs}}} e^2(n) \tag{32}$$

where μ_{V} denotes the step-size parameter and $\nabla_{\mathbf{h}_{\mathrm{Vs}}} e^2(n)$ is the gradient of the cost function with respect to $\mathbf{h}_{\mathrm{Vs}}(n)$. By using the derivative chain rule, the gradient vector is written as

$$\nabla_{\mathbf{h}_{\mathrm{Vs}}} e^2(n) = \frac{\partial e^2(n)}{\partial \mathbf{h}_{\mathrm{Vs}}(n)} = \frac{\partial e^2(n)}{\partial e(n)} \frac{\partial e(n)}{\partial \mathbf{h}_{\mathrm{Vs}}(n)}. \tag{33}$$

Thus from (30), the right-hand side (r.h.s.) terms of (33) are

$$\frac{\partial e^2(n)}{\partial e(n)} = 2e(n) \tag{34}$$

and

$$\frac{\partial e(n)}{\partial \mathbf{h}_{\mathrm{Vs}}(n)} = -\tilde{\mathbf{x}}_{\mathrm{V}}(n). \tag{35}$$

Therefore, the LMS update equation for the sparse filter is given by

$$\mathbf{h}_{\mathrm{Vs}}(n+1) = \mathbf{h}_{\mathrm{Vs}}(n) + 2\mu_{\mathrm{V}} e(n) \tilde{\mathbf{x}}_{\mathrm{V}}(n). \tag{36}$$

Note that (36) is similar to the update expression of the sparse filter of the AFIV and APIV structures [6]. When a fixed interpolator is used, many of the elements of $\tilde{\mathbf{x}}_{\mathrm{V}}(n)$ can be reused from the previous iteration. This condition is not possible in the fully adaptive approach since the interpolator filter is time varying, increasing thus the computational complexity of the structure. A solution to this problem is to assume that the interpolator filter has a slow adaptation rate and therefore $\tilde{\mathbf{x}}_{\mathrm{V}}(n)$ can be approximately obtained in a similar way as the fixed interpolator case.

The updating process for the interpolator filter is given by

$$\mathbf{i}(n+1) = \mathbf{i}(n) - \mu_{\mathrm{i}} \nabla_{\mathbf{i}} e^2(n) \tag{37}$$

where μ_{i} is the step-size parameter. Then, from (30) and considering the derivative chain rule, we have

$$\nabla_{\mathbf{i}} e^2(n) = \frac{\partial e^2(n)}{\partial \mathbf{i}(n)} = \frac{\partial e^2(n)}{\partial e(n)} \frac{\partial e(n)}{\partial \mathbf{i}(n)}. \tag{38}$$

The first r.h.s. term of (38) results in (34), while the second one is given by

$$\frac{\partial e(n)}{\partial \mathbf{i}(n)} = -\frac{\partial y(n)}{\partial \mathbf{i}(n)} = -\sum_{p=1}^{P} \frac{\partial \hat{y}_p(n)}{\partial \mathbf{i}(n)}. \tag{39}$$

Then, from (21) and considering the vector differentiation rules, described in [10], we get

$$\frac{\partial \hat{y}_1(n)}{\partial \mathbf{i}(n)} = \frac{\partial \mathbf{i}^{\mathrm{T}}(n)}{\partial \mathbf{i}(n)} \mathbf{X}_1(n) \mathbf{h}_{1\mathrm{s}}(n) = \mathbf{X}_1(n) \mathbf{h}_{1\mathrm{s}}(n). \tag{40}$$

From (24), (25) and [10], we can write

$$\frac{\partial \hat{y}_p(n)}{\partial \mathbf{i}(n)} = \frac{\partial \mathbf{i}_p^{\mathrm{T}}(n)}{\partial \mathbf{i}(n)} \mathbf{X}_p(n) \mathbf{h}_{ps}(n) \tag{41}$$

with

$$\frac{\partial \mathbf{i}_p^{\mathrm{T}}(n)}{\partial \mathbf{i}(n)} = \mathbf{I}_{\mathrm{M}} \otimes \mathbf{i}_{p-1}^{\mathrm{T}}(n) + \mathbf{i}^{\mathrm{T}}(n) \otimes \frac{\partial \mathbf{i}_{p-1}^{\mathrm{T}}(n)}{\partial \mathbf{i}(n)}. \tag{42}$$

Thus, by considering (26), (42), and the Kronecker mixed-product rule [2], (41) can be rewritten as

$$\frac{\partial \hat{y}_p(n)}{\partial \mathbf{i}(n)} = \left[\mathbf{X}_1(n) \otimes \tilde{\mathbf{x}}_{p-1}^{\mathrm{T}}(n) + \tilde{\mathbf{x}}_1^{\mathrm{T}}(n) \otimes \frac{\partial \mathbf{i}_{p-1}^{\mathrm{T}}(n)}{\partial \mathbf{i}(n)} \mathbf{X}_{p-1}(n) \right] \mathbf{h}_{ps}(n). \tag{43}$$

From (43), the gradient of the second-order block output signal with respect to $\mathbf{i}(n)$ is

$$\frac{\partial \hat{y}_2(n)}{\partial \mathbf{i}(n)} = [\mathbf{X}_1(n) \otimes \tilde{\mathbf{x}}_1^{\mathrm{T}}(n) + \tilde{\mathbf{x}}_1^{\mathrm{T}}(n) \otimes \mathbf{X}_1(n)] \mathbf{h}_{2\mathrm{s}}(n). \tag{44}$$

Due to the symmetric characteristic of $\mathbf{h}_{2\mathrm{s}}$ [$h_{2\mathrm{s}}(j,k) = h_{2\mathrm{s}}(k,j)$, $\forall$ k and j], one verifies that $[\mathbf{X}_1(n) \otimes \tilde{\mathbf{x}}_1^{\mathrm{T}}(n)] \mathbf{h}_{2\mathrm{s}}(n) = [\tilde{\mathbf{x}}_1^{\mathrm{T}}(n) \otimes \mathbf{X}_1(n)] \mathbf{h}_{2\mathrm{s}}(n)$ and that also (44) can be rewritten as

$$\frac{\partial \hat{y}_2(n)}{\partial \mathbf{i}(n)} = 2[\mathbf{X}_1(n) \otimes \tilde{\mathbf{x}}_1^{\mathrm{T}}(n)] \mathbf{h}_{2\mathrm{s}}(n). \tag{45}$$

The symmetric characteristic is also observed in the other nonlinear blocks, resulting in similar expressions. Thus, the generalized expression for the gradient of the output of a p^{th}-order block with respect to $\mathbf{i}(n)$ is given by

$$\frac{\partial \hat{y}_p(n)}{\partial \mathbf{i}(n)} = p[\mathbf{X}_1(n) \otimes \tilde{\mathbf{x}}_{p-1}^{\mathrm{T}}(n)] \mathbf{h}_{ps}(n). \tag{46}$$

Finally, the update expression for the interpolator coefficients is

$$\mathbf{i}(n+1) = \mathbf{i}(n) + 2\mu_{\mathrm{i}} e(n) \sum_{p=1}^{P} p[\mathbf{X}_1(n) \otimes \tilde{\mathbf{x}}_{p-1}^{\mathrm{T}}(n)] \mathbf{h}_{ps}(n). \tag{47}$$

Since the interpolator is time varying, the element $\tilde{\mathbf{x}}_{p-1}^{\mathrm{T}}(n)$ in (47) should be entirely determined at each iteration. However, due to complexity questions, the same approximation adopted for evaluating $\tilde{\mathbf{x}}_{\mathrm{V}}(n)$ in (36) is used here.

Fully adaptive interpolated Volterra structures can be implemented either by considering a fully interpolated Volterra (FIV) structure or a partially interpolated Volterra (PIV) one. The differences in the adaptive process between the fully adaptive FIV (FAFIV) and fully adaptive PIV (FAPIV) structures are very small and the update expressions are easily obtained from (36) and (47).

Regarding the computational burden, Fig. 2 shows the number of operations per sample as a function of the memory size for implementing a conventional Volterra filter, the AFIV and FAFIV structures. From this figure, it is evident that the proposed approach provides considerable computational savings when compared with the conventional Volterra implementation.

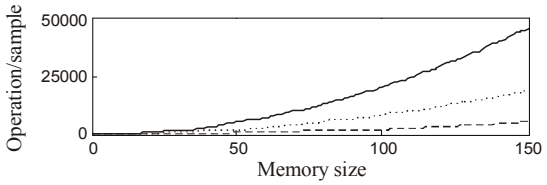

Fig. 2. Computational burden for a second-order LMS Volterra implementation. (Solid line) conventional adaptive Volterra filter. (Dashed line) AFIV structure. (Dotted line) FAFIV approach.

5. SIMULATION RESULTS

In this section, some simulation results are shown, illustrating the performance of the fully adaptive interpolated structure in comparison with the standard adaptive interpolated approaches. The presented examples consider a system identification problem [9]. The performance of the structures is assessed in terms of the MSE obtained from Monte Carlo simulations (average of 100 runs). The simulated structures are second-order AFIV and FAFIV ones. The interpolation factor is $L = 2$ and the coefficients of the

fixed interpolator used by the AFIV structure is given by $\mathbf{i}=[0.5\ 1\ 0.5]^{\mathrm{T}}$. In the case of the FAFIV structure, the interpolator filter is initialized with $\mathbf{i}=[0.5\ 1\ 0.5]^{\mathrm{T}}$. The input signal is a white Gaussian with variance $\sigma_x^2=1$, and the additive noise added to $d(n)$ is a white Gaussian with variance $\sigma_z^2=10^{-4}$.

Example 1: In this example, the memory size of the plant as well as length sparse filters are equal to 11 ($N=11$). The step-size parameter is $\mu=\mu_{\max_1}/2$ for the AFIV structure, and $\mu_{\mathrm{i}}=\mu_{\mathrm{V}}=\mu_{\max_2}/2$ for the FAFIV structure ($\mu_{\max_1}$ and $\mu_{\max_2}$ are the maximum step-size values, obtained experimentally). The MSE results are shown in Fig. 3, from which one observes a better performance of the FAFIV structure as compared with the AFIV one. Fig. 4 shows the first-order filter coefficients of the plant and the steady-state equivalent ones for each adaptive structure. Figs. 5 and 6 show the second-order kernels, illustrating the surface plots. In this case, one observes the existing match between the plant and the estimated kernels, which is much better for the FAFIV structure than for the AFIV one.

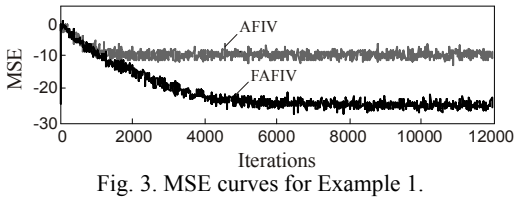

Fig. 3. MSE curves for Example 1.

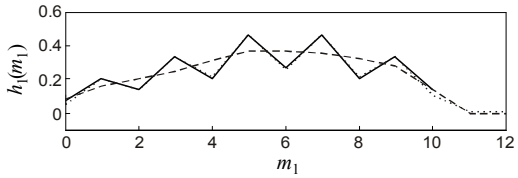

Fig. 4. Coefficient curves for the first-order block from Example 1. (Solid line) plant. (Dashed line) AFIV structure. (Dotted line) FAFIV structure.

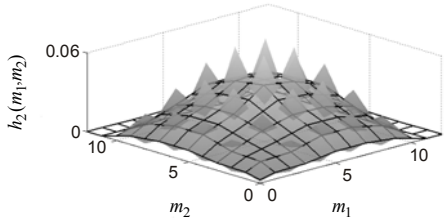

Fig. 5. Superposition of second-order coefficient surfaces for Example 1. (Solid surface) plant. (Wireframe surface) AFIV structure.

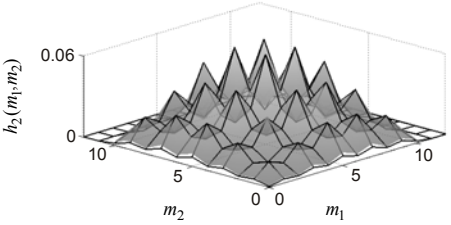

Fig. 6. Superposition of second-order coefficient surfaces for Example 1. (Solid surface) plant. (Wireframe surface) FAFIV structure.

Example 2: In this example, the used plant is the same given in [11, Example 2]. The memory sizes and step-size values are the same as in Example 1. The MSE curves are shown in Fig. 7. Now, one notices that the performance difference between the AFIV and the FAFIV structures is smaller than in the previous example. This is due to the fact that the fixed interpolator of the AFIV structure is in a better agreement with the characteristics of the plant used in this example, resulting in an improved performance. However, even in this case, a slightly better performance of the FAFIV structure is still verified, thus confirming the effectiveness of the proposed approach.

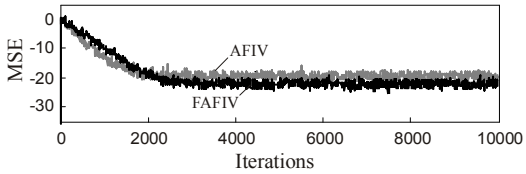

Fig. 7. MSE curves for Example 2.

6. CONCLUSIONS

This paper presents the derivation of the LMS algorithm for a fully adaptive interpolated Volterra structure. The proposed approach allows improving the performance of standard adaptive interpolated Volterra structures, providing computational savings in comparison with the standard adaptive Volterra filter. The presented simulation results verify the effectiveness of the proposed algorithm.

7. REFERENCES

[1] L. Tan and J. Jiang, "Adaptive Volterra filters for active control of nonlinear noise processes," *IEEE Trans. Signal Processing*, vol. 49, no. 8, pp. 1667-1676, Aug. 2001.

[2] V. J. Mathews and G. L. Sicuranza, *Polynomial Signal Processing*, John Wiley & Sons Inc., 2000.

[3] L. Tan and J. Jiang, "An adaptive technique for modeling second-order Volterra systems with sparse kernels," *IEEE Trans. Circ. and Syst. II-Analog and Digital Signal Processing,* vol. 45, no. 12, pp. 1610-1615, Dec. 1998.

[4] A. Fermo, A. Carini, and G. L. Sicuranza, "Simplified Volterra filters for acoustic echo cancellation in GSM receivers," in *Proc. European Signal Processing Conf.* (*EUSIPCO*), Tampere, Finland, Sep. 2000.

[5] M. J. Reed and M. O. J. Hawksford, "Efficient implementation of the Volterra filter", *IEE Proc.-Vis. Image Signal Processing*, vol. 147, no. 2, pp. 109-114, Apr. 2000.

[6] E. L. O. Batista, O. J. Tobias, and R. Seara, "A mathematical framework to describe interpolated adaptive Volterra filters," in *Proc. IEEE Int. Telecomm. Symp*., Fortaleza, Brazil, Sep. 2006, pp. 144-149.

[7] O. J. Tobias and R. Seara, "Analytical model for the first and second moments of an adaptive interpolated FIR filter using the constrained filtered-X LMS algorithm," *IEE Proceedings – Vision, Image, Signal Process.*, vol. 148, no. 5, pp. 337-347, Oct. 2001.

[8] E. L. O. Batista, O. J. Tobias, and R. Seara, "New insights in adaptive cascaded FIR structure: application to fully adaptive interpolated FIR structures," in *Proc. 15th European Signal Processing Conf. (EUSIPCO)*, Poznan, Poland, vol. 1, Sep. 2007, pp. 370-374.

[9] S. Haykin, *Adaptive Filter Theory*, 4th ed., Prentice-Hall, 2002.

[10] J. W. Brewer, "Kronecker Products and Matrix Calculus in System Theory," *IEEE Trans. on Circuits and Systems*, vol. CAS-25, no. 9, pp. 772-781, Sep. 1978.

[11] E. L. O. Batista, O. J. Tobias, and R. Seara, "Border effect removal for IFIR and interpolated Volterra filters," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP)*, Honolulu, USA, vol. 3, Apr. 2007, pp. 1329-1332.