Newton-like Nonlinear Adaptive Filters via Simple Multilinear Functionals

Felipe C. Pinheiro *and* Cássio G. Lopes
Dept. of Electronic Systems Engineering, University of São Paulo – São Paulo, SP – Brazil
Emails: felipe.chaud.pinheiro@usp.br; cassio@lps.usp.br

***Abstract*—In the context of nonlinear systems identification new Affine Projection Algorithm (APA) and NLMS adaptive filters (AFs) are developed over the Simple Multilinear model (SML). Such a model is comprised of a product of linear filters and allows for an exponential decrease in complexity when compared to the complete Volterra model. The MSE surface is developed in terms of data statistical moments and its gradient vector is presented, computing the corresponding Hessian matrix in the sequel. The AFs are generated via stochastic approximations for the data moments and a series of non-trivial derivations resulting in an APA implementation structurally similar to the standard APA recursion. The NLMS algorithm is derived as a particular case. Simulations show good convergence properties when identifying unknown SML and Volterra plants.**

I. Introduction

Research in nonlinear adaptive filtering deals with situations where the filtering tasks involve some nonlinear component, such as in some instances of echo cancellation [1], [2], in which linear techniques underperform. The literature in the area is largely focused on polynomial models, such as the Volterra series [3]–[6]. In order to reduce the high complexity of such filters, this approach may be alternatively recast via the Simple Multilinear model (SML) proposed in [7], where we have derived both the model and a Least Mean Squares (LMS) filter. This new model corresponds to a product of K linear filters and involves concepts from multilinear algebra [8].

In this paper we extend previous work on adaptive nonlinear adaptive algorithms by deriving and testing SML-based Newton-like algorithms. This is accomplished by estimating the parameters of a Newton's method recursion. The resulting algorithms—Affine Projections Algorithm (APA) and Normalized Least Mean Squares (NLMS)—compare favorably to existing approaches with respect to computational complexity and/or mean-square performance, as suggested by simulations.

II. SML and Mean-Square Surface

Given an input row vector[1] u_i $(1 \times M)$ that collects samples from a signal $u(i)$ and a collection of K vectors $\{w_1, \ldots, w_K\}$ (each of size $M \times 1$), the simple multilinear model[2] is constructed by imposing decomposability over a homogeneous Volterra kernel and by exploring a tensor formulation. The input-output relation of a nonlinear system represented via SML then becomes

$$\begin{aligned} y(i) &= \underbrace{(u_i \otimes \cdots \otimes u_i)}_{K \text{ times}}(w_1 \otimes \cdots \otimes w_K) \\ &= u_i^{\otimes K} w = (u_i w_1) \cdots (u_i w_K), \end{aligned} \tag{1}$$

where $\otimes$ is the Kronecker (tensor) product, $w \triangleq w_1 \otimes \cdots \otimes w_K$ and $u_i^{\otimes K} \triangleq u_i \otimes \cdots \otimes u_i$ (K times). This allows us to describe the system with only KM coefficients, in contrast with the M^K ones from the full homogeneous Volterra kernel [6].

As commonly done in adaptive filtering, we take a random input regressor $\mathbf{u}$ $(1 \times M)$ and a random "desired" signal $\mathbf{d}$. We want to estimate an SML model that relates $\mathbf{d}$ and $\mathbf{u}$. This may be pursued by minimizing over the variables $\{w_1, \ldots, w_K\}$ the power of the error signal $\mathbf{e} = \mathbf{d} - \mathbf{u}^{\otimes K} w$. The cost function—the mean-square error (MSE)—as previously derived, is

$$\begin{aligned} \text{MSE}(w_1, \ldots, w_K) = \mathbb{E}|\mathbf{e}|^2 = \\ R_d - w^* R_{u^K d}^* - R_{u^K d} w + w^* R_{u^K} w, \end{aligned} \tag{2}$$

where the correlation parameters are

$$R_{u^K} = \mathbb{E}[\mathbf{u}^{\otimes K *} \mathbf{u}^{\otimes K}], \quad R_d = \mathbb{E}|\mathbf{d}|^2, \tag{3}$$

$$R_{u^K d} = \mathbb{E}[\mathbf{u}^{\otimes K} \mathbf{d}^*] = R_{du^K}^*. \tag{4}$$

The cost function (2) may be minimized by computing its gradient, which can be done block-wise:

$$\frac{\partial \text{MSE}}{\partial w_s} = [-R_{u^K d} + w^* R_{u^K}] W^{(s)}, \tag{5}$$

where $W^{(s)} \triangleq (w_1 \otimes \cdots \otimes \widehat{w_s} \otimes \cdots \otimes w_K)$ is a $M^K \times M$ matrix and the symbol $\widehat{w_s}$ implies that the vector w_s has been substituted for the identity matrix I_M of order M.

Stacking the blocks from (5) into a vector yields the total gradient, from which the gradient-descent algorithm is obtained. Proper instantaneous approximations for R_{u^K} and $R_{u^K d}$ give rise to the SML-LMS [7].

A. The Hessian Matrix

Newton's algorithm requires the calculation of the Hessian matrix, which is formed from the second partial derivatives of the MSE in (2). In the complex case, the second derivative must always be in relation to the conjugate of the variables.

The authors were supported by grants from CNPq – Brazil.

[1]The paper follows the methods and notation from [9]. Bold fonts represent random signals, variables of the form $x(i)$ represent scalars, ones of the form y_i represent vectors varying in time and capital letters represent either constants or matrices as dictated by the context.

[2]Here we present only the result. For more details on the model derivations see [7].

978-0-9928-6265-7/16/$31.00 ©2016 IEEE

One could verify that such matrix, when constructed for functions of many vectors, would take the form

$$\nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial w_1^* \partial w_1} & \cdots & \frac{\partial^2 f}{\partial w_1^* \partial w_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_K^* \partial w_1} & \cdots & \frac{\partial^2 f}{\partial w_K^* \partial w_K} \end{pmatrix}. \tag{6}$$

Each of the elements represented in the (6) are matrices themselves and must be computed from the cost function (2).

Proposition. *The blocks that form the Hessian matrix for the MSE function are given by*

$$\frac{\partial^2 MSE}{\partial w_r^* \partial w_s} = W^{*(r)} R_{u^K} W^{(s)}, \tag{7}$$

where $W^{*(r)} = (w_1^* \otimes \cdots \otimes \widehat{w_r^*} \otimes \cdots \otimes w_K^*)$ *is a* $M \times M^K$ *matrix and* $\widehat{w_r^*}$ *implies that* w_r^* *has been substituted for the identity matrix* I_M *of order* M.

Proof. Looking at (5), we see that the only term that depends on the conjugate of the coordinates of some w_r is $w^* R_{u^K} W^{(s)}$. In [7], the coordinates of this term were explicitly calculated. The procedure is to realize that the product $\mathbf{u}^{\otimes K}$ can be indexed as

$$(\mathbf{u}^{\otimes K})_{j_1,\ldots,j_K} = \mathbf{u}(j_1) \cdots \mathbf{u}(j_K),$$

where $\mathbf{u}(r)$ is the r-th coordinate of $\mathbf{u}$. This covers every element of $\mathbf{u}^{\otimes K}$. Similarly, we can index $\mathbf{u}^{\otimes K*}$ with the indexes $(\mathbf{u}^{\otimes K*})^{i_1,\ldots,i_K}$ in a way that we can also index the product of these two vectors as $(\mathbf{u}^{\otimes K*}\mathbf{u}^{\otimes K})^{i_1,\ldots,i_K}_{j_1,\ldots,j_K}$. The matrix $R_{u^K} = \mathbb{E}[\mathbf{u}^{\otimes K*}\mathbf{u}^{\otimes K}]$, therefore, inherits this structure. This is equivalent to looking at this matrix as a multi-index object—a tensor [8]. The upper and lower indexes are a tradition of tensor calculus [10]. One can also index $w = w_1 \otimes \cdots \otimes w_K$ in this way. Under this convention, the coordinates of $w^* R_{u^K} W^{(s)}$ were previously computed as

$$\frac{\partial (w^* R_{u^K} w)}{\partial (w_s)^{j_q}} = \sum_{\substack{i_1,\ldots,i_K \\ j_1,\ldots,j_K}} \prod_p (w_p^*)_{i_p} (R_{u^K})^{i_1,\ldots,i_K}_{j_1,\ldots,j_K} \prod_{\ell \neq s} (w_\ell)^{j_\ell} \delta^{j_r}_{j_q}.$$

Then we need only to derivate with respect to the coordinates of the conjugate vectors, resulting in the i_d and j_q coordinates of (7):

$$\left(\frac{\partial^2 \text{MSE}}{\partial w_r^* \partial w_s}\right)^{i_d}_{j_q} = \frac{\partial^2 (w^* R_{u^K} w)}{\partial (w_r^*)_{i_d} \partial (w_s)^{j_q}} = \sum_{\substack{i_1,\ldots,i_K \\ j_1,\ldots,j_K}} \prod_{p \neq r} (w_p^*)_{i_p} \delta^{i_d}_{i_r} (R_{u^K})^{i_1,\ldots,i_K}_{j_1,\ldots,j_K} \prod_{\ell \neq s} (w_\ell)^{j_\ell} \delta^{j_s}_{j_q}. \tag{8}$$

As i_d and j_q take values from 1 through M, (8) indeed renders matrix (7). This can be shown as follows: the Kronecker delta δ^i_j indexes the identity matrix, that is, $(I_M)^i_j = \delta^i_j$. In (8), the two instances of the delta work as if occupying the positions of the coordinates $(w_r^*)_{i_r}$ and $(w_s)^{j_s}$—that is, the coordinates of an identity I_M are occupying the positions of the coordinates of the vectors w_r^* and w_s. This is valid for any pair of indexes i_d and j_q, thus, when reconstructing the matrix form implied by the coordinates computed in (8), the substitutions with I_M in (7) must be made. □

B. Newton's Method

Once the Hessian matrix has been obtained, Newton's algorithm follows, similarly to what is done in [9] for the linear filter. First we stack the vectors $\{w_1, \ldots, w_K\}$ into the $KM \times 1$ vector w^C and the corresponding partial gradients (5) into the $1 \times KM$ total gradient ∇MSE. Then, Newton's algorithm is implemented by

$$w^C[i] = w^C[i-1] - \mu \left(\nabla^2 \text{MSE}\right)^{-1} (\nabla \text{MSE})^*, \tag{9}$$

where both the Hessian and the gradient are computed at $w^C[i-1]$.

Real-time algorithms require instantaneous approximations for the statistical moments and also resort to efficient ways to invert, or avoid inverting, of the Hessian matrix. This is addressed in the sequel.

III. Stochastic Approximations

Adaptive filters make use of real-time data. For example, APA-like algorithms assume that, at time i, we have access to the L last input vectors and samples of the desired signal. In other words, we have $\{u_i, \ldots, u_{i-L+1}\}$ and $\{d(i), \ldots, d(i-L+1)\}$. By utilizing these variables, instantaneous approximations for the quantities R_{u^K} and $R_{u^K d}$ are

$$\widetilde{R}_{u^K} = \frac{1}{L} \sum_{j=i-L+1}^{i} u_j^{\otimes K*} u_j^{\otimes K}, \tag{10}$$

$$\widetilde{R}_{u^K d} = \frac{1}{L} \sum_{j=i-L+1}^{i} u_j^{\otimes K} d(j)^* = \tilde{R}^*_{du^K}. \tag{11}$$

In the linear case (order of nonlinearity $K = 1$), this gives rise to the conventional APA algorithm. In the general case, some extra steps are required.

A. Gradient Estimation

Using the stochastic approximations (10) and (11) in (5) returns an instantaneous estimate for the gradient:

$$\begin{aligned} \widetilde{\frac{\partial \text{MSE}}{\partial w_s}} &= [-\widetilde{R}_{u^K d} + w^* \widetilde{R}_{u^K}] W^{(s)} \\ &= \frac{1}{L} \sum_{j=i-L+1}^{i} \left[-d(j)^* + w^* u_j^{\otimes K*}\right] u_j^{\otimes K} W^{(s)}. \end{aligned} \tag{12}$$

Equation (12) may be conveniently rewritten as follows. If we define $y^w_s(j) \triangleq (u_j w_1) \cdots \widehat{(u_j w_s)} \cdots (u_j w_K)$, where the hat implies a factor being omitted and the w superscript implies that an specific $w^C = [w_1; \ldots; w_K]$ should be used to compute this function, we can see that $u_j^{\otimes K} W^{(s)} = y^w_s(j) u_j$. We then define three variables:

$$d_i \triangleq \begin{pmatrix} d(i) \\ \vdots \\ d(i-L+1) \end{pmatrix}_{(L\times 1)}, \; U_i^K \triangleq \begin{pmatrix} u_i^{\otimes K} \\ \vdots \\ u_{i-L+1}^{\otimes K} \end{pmatrix}_{(L \times KM)},$$
$$y_j^w \triangleq \begin{pmatrix} y_1^w(j) & \cdots & y_K^w(j) \end{pmatrix}_{(1\times K)}. \tag{13}$$

Eq. (12) represents a block gradient. For each j, the terms of this sum will be multiplied only by $y_s^w(j)$—the s-th element of y_j^w—which shows that the total gradient has a Kronecker structure. This follows from the fact that $u_j^{\otimes K} W^{(s)} = y_s^w(j) u_j$. We can, therefore, write

$$\widetilde{\nabla}\text{MSE} = \frac{1}{L} \sum_{j=i-L+1}^{i} \left[-d(j)^* + w^* u_j^{\otimes K *}\right] \left(y_j^w \otimes u_j\right). \quad (14)$$

If we also introduce the notation

$$T_i \triangleq \begin{pmatrix} y_i^w \otimes u_i \\ \vdots \\ y_{i-L+1}^w \otimes u_{i-L+1} \end{pmatrix}_{(L \times KM)}, \quad (15)$$

we can get a compact expression for the instantaneous gradient:

$$\widetilde{\nabla}\text{MSE} = -\frac{1}{L}\left[d_i - U_i^K w\right]^* T_i = -\frac{1}{L} e_i^* T_i, \quad (16)$$

where $e_i \triangleq d_i - U_i^K w$ $(L \times 1)$. And, finally, if we define $y^w(j) \triangleq (u_j w_1) \cdots (u_j w_K)$, computed at w^C, and $y_i \triangleq \begin{pmatrix} y^w(i) & \cdots & y^w(i-L+1) \end{pmatrix}^T$ $(L \times 1)$, then we can compute e_i as

$$e_i = d_i - y_i. \quad (17)$$

B. Hessian Estimation

An instantaneous approximation for the Hessian matrix is obtained by replacing (10) into (7):

$$\begin{aligned} \frac{\partial^2 \widetilde{\text{MSE}}}{\partial w_r^* \partial w_s} &= W^{*(r)} \frac{1}{L} \sum_{j=i-L+1}^{i} u_j^{\otimes K *} u_j^{\otimes K} W^{(s)} \\ &= \frac{1}{L} \sum_{j=i-L+1}^{i} \left(u_j^{\otimes K} W^{(r)}\right)^* \left(u_j^{\otimes K} W^{(s)}\right) \\ &= \frac{1}{L} \sum_{j=i-L+1}^{i} \left(y_r^w(j) u_j\right)^* \left(y_s^w(j) u_j\right). \end{aligned} \quad (18)$$

This follows from $u_j^{\otimes K} W^{(s)} = y_s^w(j) u_j$, as it was carried out in the previous subsection. Similarly, (18) may also be recast in a more compact form.

Take only the j-th term of the sum. It can also be written as $(y_r^w(j)^* y_s^w(j))\left(u_j^* u_j\right)$. If we define a matrix A_s^r whose elements are given by $y_r^w(j)^* y_s^w(j)$, then the complete Hessian matrix becomes a sum over j of the matrices

$$\begin{pmatrix} A_1^1 u_j^* u_j & \cdots & A_K^1 u_j^* u_j \\ \vdots & \ddots & \vdots \\ A_1^K u_j^* u_j & \cdots & A_K^K u_j^* u_j \end{pmatrix} = A \otimes \left(u_j^* u_j\right). \quad (19)$$

Moreover, we have precisely $A = y_j^{w*} y_j^w$, so that

$$\begin{aligned} \widetilde{\nabla}^2\text{MSE} &= \frac{1}{L} \sum_{j=i-L+1}^{i} \left(y_j^{w*} y_j^w\right) \otimes \left(u_j^* u_j\right) \quad (20) \\ &= \frac{1}{L} \sum_{j=i-L+1}^{i} \left(y_j^w \otimes u_j\right)^* \left(y_j^w \otimes u_j\right) \\ &= \frac{1}{L} T_i^* T_i, \quad (21) \end{aligned}$$

where the last equality follows directly from the definition of T_i in (15).

IV. Adaptive Algorithms

APA and NLMS-like algorithms can be derived from Newton's algorithm and from the stochastic approximations for the gradient vector (16) and the Hessian matrix (21).

A. APA-like Algorithm

The update equation follows directly from (9), with the appropriate substitutions.

$$\begin{aligned} w^C[i] &= w^C[i-1] - \mu \left(\widetilde{\nabla}^2\text{MSE}\right)^{-1} \left(\widetilde{\nabla}\text{MSE}\right)^* \\ &= w^C[i-1] + \mu \left(T_i^* T_i\right)^{-1} T_i^* e_i \end{aligned} \quad (22)$$

We cannot invert $T_i^* T_i$ directly, because it is not invertible in most cases. Thus, we add a regularization term ϵI and apply the Matrix Inversion Lemma to get

$$\left[\epsilon I + T_i^* T_i\right]^{-1} T_i^* = T_i^* \left[\epsilon I + T_i T_i^*\right]^{-1}. \quad (23)$$

The matrix $T_i T_i^*$ $(L \times L)$, not only is smaller in the usual case of $L < M$, but also has better inversion properties. We can compute this matrix directly as

$$\begin{aligned} \left(T_i T_i^*\right)_s^r &= \left(y_{i-r+1}^{w[i-1]} \otimes u_{i-r+1}\right) \left(y_{i-s+1}^{w[i-1]} \otimes u_{i-s+1}\right)^* \\ &= \left(y_{i-r+1}^{w[i-1]} y_{i-s+1}^{w[i-1]*}\right) \otimes \left(u_{i-r+1} u_{i-s+1}^*\right) \\ &= \left(y_{i-r+1}^{w[i-1]} y_{i-s+1}^{w[i-1]*}\right) \left(u_{i-r+1} u_{i-s+1}^*\right). \end{aligned} \quad (24)$$

The two factors in (24) are scalars, so, if we introduce the notation

$$Y_i \triangleq \begin{pmatrix} y_i^{w[i-1]} \\ \vdots \\ y_{i-L+1}^{w[i-1]} \end{pmatrix}_{(L \times K)}, \quad U_i \triangleq \begin{pmatrix} u_i \\ \vdots \\ u_{i-L+1} \end{pmatrix}_{(L \times M)}, \quad (25)$$

then we can say that

$$T_i T_i^* = \left(Y_i Y_i^*\right) \circ \left(U_i U_i^*\right), \quad (26)$$

where $\circ$ denotes the Hadamard (or entry-wise) product. This leads us into the final form of the APA recursion:

$$w^C[i] = w^C[i-1] + \mu T_i^* \left[\epsilon I + \left(Y_i Y_i^*\right) \circ \left(U_i U_i^*\right)\right]^{-1} e_i. \quad (27)$$

An implementation of the algorithm, with suitable initialization (as discussed in [7]), is described in Algorithm 1.

B. NLMS-like Algorithm

By setting $L = 1$, that is, by using only one past sample, an NLMS-like algorithm can be obtained. This would imply, for example, $Y_i = y_i^{w[i-1]}$ and $U_i = u_i$. Therefore, $U_i U_i^* = \|u_i\|^2$ and $Y_i Y_i^* = \|y_i^{w[i-1]}\|^2$. It can also be seen that $e_i = d(i) - y^{w[i-1]}(i) = e(i)$.

Additionally, it must be that $T_i = y_i^{w[i-1]} \otimes u_i$. The NLMS recursion is, therefore,

$$w^C[i] = w^C[i-1] + \mu T_i \left[\epsilon + \|y_i^{w[i-1]}\|^2 \|u_i\|^2\right]^{-1} e(i). \quad (28)$$

Algorithm 1 APA-like algorithm

Initialization
for $j = 1$ to $K-1$ **do**
  $w_j[0] = [2^{1-j}\, 0 \cdots 0\, 0]^T$
end for
$w_k[0] = [0\, 0 \cdots 0\, 0]^T$
Iteration
for $i = 1$ to END **do**
  for $\ell = 1$ to L **do**
    for $r = 1$ to K **do**
      Compute $y_{or\ell}(i) = u_{i-\ell+1} w_r[i-1]$
    end for
  end for
  for $\ell = 1$ to L **do**
    for $r = 1$ to K **do**
      Compute $y_{r\ell}(i) = y_{o1\ell}(i) \cdots \widehat{y_{or\ell}(i)} \cdots y_{oK\ell}(i)$
    end for
    Compute $y_\ell(i) = y_{K\ell}(i) y_{oK\ell}(i)$
    Set $Y_i(\ell, :) = [y_{1\ell}(i) \quad \cdots \quad y_{K\ell}(i)]$
    Set $T_i(\ell, :) = Y_i(\ell, :) \otimes u_{i-\ell+1}$
  end for
  Set $y_i = [y_1(i); \quad \cdots ; \quad y_L(i)]$
  Compute $e_i = d_i - y_i$
  Compute $Q_i = \epsilon I + \left(Y_i Y_i^*\right) \circ \left(U_i U_i^*\right)$
  Compute $w^C[i] = w^C[i-1] + \mu T_i^* Q_i^{-1} e_i$
end for

Algorithm 2 NLMS-like algorithms

Initialization
for $j = 1$ to $K-1$ **do**
  $w_j[0] = [2^{1-j}\, 0 \cdots 0\, 0]^T$
end for
$w_k[0] = [0\, 0 \cdots 0\, 0]^T$
Iteration
for $i = 1$ to END **do**
  for $r = 1$ to K **do**
    Compute $y_{or}(i) = u_i w_r[i-1]$
  end for
  for $r = 1$ to K **do**
    Compute $y_r(i) = y_{o1}(i) \cdots \widehat{y_{or}(i)} \cdots y_{oK}(i)$
  end for
  Compute $y(i) = y_K(i) y_{oK}(i)$
  Set $y_i = [y_1(i) \quad \cdots \quad y_K(i)]$
  Compute $e(i) = d(i) - y(i)$
  Compute $f_i = \mu e(i) \frac{u_i^*}{\epsilon + \|y_i\|^2 \|u_i\|^2}$
  for $r = 1$ to K **do**
    Compute $w_r[i] = w[i-1] + f_i y_r(i)^*$
  end for
end for

This can be further simplified by separating each w_r from w^C. This is done by separating T_i: its first M coordinates are $y_1^{w[i-1]}(i) u_i$, the next M terms are $y_2^{w[i-1]}(i) u_i$, and so on that the r-th M terms are $y_r^{w[i-1]}(i) u_i$. This leads to the following update equation, for every r, with $1 \le r \le K$:

$$w_r[i] = w_r[i-1] + \mu e(i) \frac{y_r^{w[i-1]}(i)^* u_i^*}{\epsilon + \|y_i^{w[i-1]}\|^2 \|u_i\|^2}. \qquad (29)$$

C. Computational Complexity

It is possible to compute the number of multiplications these algorithms take, for $K \ge 2$. The results are on the Table I. This assumes $O(L^3)$ multiplications to invert an $L \times L$ matrix.

TABLE I
COMPUTATIONAL COMPLEXITY OF THE ALGORITHMS.

Algorithm	No. of multiplications
SML-NLMS	$KM + 2M + K^2 + 4$
SML-APA	$(2KL + L^2 + KL^2)M + (KL + L^2 - 2L)K + O(L^3)$

V. SIMULATIONS

For cases I-IV, we have tested the two new algorithms (NLMS and APA) against polynomial versions of the linear NLMS and APA algorithms [6], [9]. These algorithms consist of taking all the possible K-order Volterra monomials $u(i)^{\alpha_1} \cdots u(i-M+1)^{\alpha_M}$, $\alpha_1 + \cdots + \alpha_M = K$, and putting them in the regressor u_i. The SML algorithms, conversely, assume u_i to have a delay line structure, because the nonlinearity is in the structure of the filter itself. In Cases III and IV we also compare the new algorithms with the SML-LMS [7].

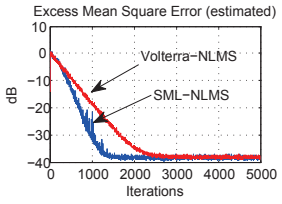
The Cases I-IV were run with $K = 2$, $N = 5{,}000$ iterations, 1,000 realizations, $M = 10$, and $L = 4$ for the APA, with an uncorrelated, zero-mean, unitary power Gaussian input, trying to identify an unknown SML plant with additive noise of power 10^{-3}—except Case IV, which is run through 40,000 iterations and with a colored input. The additional parameters of the simulation were empirically chosen so to equalize the steady-state Excess Mean Square Error. The resulting EMSE curves are presented in Fig. 1.

Case V compares the SML-NLMS with some algorithms present in the literature. These are all Volterra-based algorithms applied on an NLMS-like framework. These are the Power Filter (PF) [11], Simplified Volterra with three diagonals (SV) [1], [2], Sparse-Interpolated Volterra (IV) [3], [4] and the common Volterra Filter (V). We chose $M = 21$, $N = 10{,}000$ iterations, $K = 2$ and the rest being the same as in the previous cases. The plant was chosen to be a smooth and highly correlated one. The specific parameters were chosen to equalize the convergence rates, although only approximately in the case of the SML, since it exhibited a very distinct path of convergence. The resulting MSE curves are in Fig. 2a.

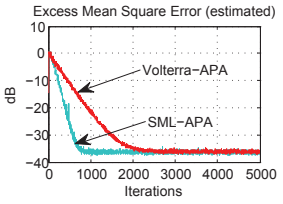
Case VI does a direct comparison with the Parallel Cascade Filter (CF) [12], single branch version. For $K = 2$, the filters would produce quite similar algorithms, with the product of two linear filters. So $K = 3$ was chosen, which makes the SML a product of three linear filters and CF a product of a second order Volterra filter with a linear filter. The unknown plant was a third order SML model. The algorithms have been simulated for $M = 10$, $N = 30{,}000$ iterations averaged through 2,000 realizations. The rest are the same as the other cases.

VI. DISCUSSION

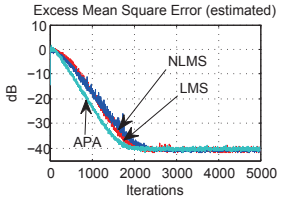
Fig. 1a shows the SML-NLMS and Volterra-NLMS algorithms converging. We can also see that the Volterra algorithm converges slower than the SML. The same happens in Fig 1b, for the APA algorithms. This behavior makes sense, as it is natural that the SML algorithms would be better at identifying an SML plant.



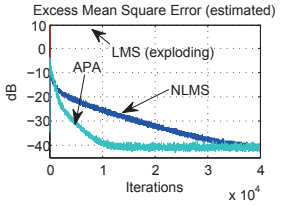
(a) Case I: NLMS filters.



(b) Case II: APA filters.

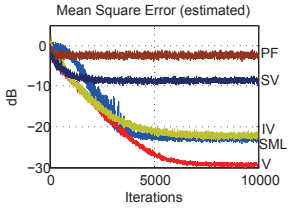


(c) Case III: Comparison with the LMS; uncorrelated input.

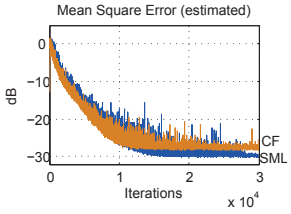


(d) Case IV: Comparison with the LMS; colored input.

Fig. 1. Plot of the EMSE curves from the SML algorithms.



(a) Case V: SML and various filters.



(b) Case VI: SML and P. Cascade.

Fig. 2. Comparing with the literature.

In Fig. 1c we have all algorithms behaving in similar ways, with the APA being marginally faster. We also can see, mostly in APA, the mitigation of the effect of slower convergence speed at the beginning of the process, as it was noted in [7] for the LMS. On the other hand, in Fig. 1d we see that the LMS has become unstable (something that happens when we set its μ greater than 0.0005, but even if we set it lower it would show higher steady-state EMSE). This shows that the choice of μ for this algorithm depends on the statistics of the input signal, something that does not happen for the Newton-based algorithms. The APA algorithm was the least affected by the correlated input, but it still shows a decrease in speed and the existence of modes of convergence throughout the process.

Fig. 2a shows a simulation on a non-SML plant. As expected, there is some degradation in the steady-state MSE as compared to the complete Volterra filter (231 coefficients), but the SML-NLMS performs better than both FP and SV and shows itself competitive in relation to IV, converging at the same time to almost the same plateau. Our algorithm initially seems to converge slowly, but soon speeds up enough to catch up with the others. In relation to the Sparse Interpolated Volterra (60 coefficients), our algorithm (42 coefficients) shows comparable performance with 30% fewer filter coefficients. The Simplified Volterra used 66 coefficients, and the Power Filter only 21.

In Case VI, we have simulated the SML-NLMS against its most similar filter of the set, the Parallel Cascade filter. We are able to see some differences, albeit minor, in convergence, but the main distinction is in complexity: the SML required only 30 coefficients while the CF required 66. For third order models, we would have complexities $O(3M)$ for the SML and $O(M^2)$ for the CF.

VII. Conclusion

We have derived two new algorithms based on Newton's method when applied to the mean square surface of the SML model. This extends the results from [7] to APA and NLMS-like algorithms and maintains the model's characteristic low complexity, but now with better tolerance to colored input signals than the previously derived LMS.

In terms of complexity, conventional Volterra algorithms take $O(L^2C)$ operations per iteration, for the APA, and $O(C)$, for the NLMS, where $C = \binom{M+K-1}{K}$ is the number of Volterra coefficients, which shows how complex they can get. From Table I, we see that the SML-NLMS is only a $O(KM)$ algorithm and the SML-APA is $O(KL^2M)$. We have also verified a reduction on the number of filter coefficients in relation to other low-complexity algorithms in the literature, while maintaining comparable adaptation properties.

Complete understanding of the algorithms still requires convergence and parameter analysis. Future work also involves extensions of the model to more general Volterra plants. All of this will be covered in future publications, together with some applications and new comparisons.

References

[1] A. Fermo, A. Carini, and G. L. Sicuranza, "Simplified volterra filters for acoustic echo cancellation in gsm receivers," in *Signal Processing Conference, 2000 10th European*, Sept 2000, pp. 1–4.

[2] ——, "Low-complexity nonlinear adaptive filters for acoustic echo cancellation in gsm handset receivers," *European Transactions on Telecommunications*, vol. 14, no. 2, pp. 161–169, 2003. [Online]. Available: http://dx.doi.org/10.1002/ett.908

[3] E. Batista, O. Tobias, and R. Seara, "A sparse-interpolated scheme for implementing adaptive volterra filters," *Signal Processing, IEEE Transactions on*, vol. 58, no. 4, pp. 2022–2035, April 2010.

[4] E. Batista, O. J. Tobias, and R. Seara, "A fully lms adaptive interpolated volterra structure," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, March 2008, pp. 3613–3616.

[5] V. Mathews, "Adaptive polynomial filters," *Signal Processing Magazine, IEEE*, vol. 8, no. 3, pp. 10–26, July 1991.

[6] T. Ogunfunmi, *Adaptive Nonlinear System Indentification: The Volterra and Wiener Model Approaches*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[7] F. C. Pinheiro and C. G. Lopes, "A nonlinear adaptive filter based on the model of simple multilinear functionals," *CoRR*, vol. abs/1603.00427, 2016. [Online]. Available: http://arxiv.org/abs/1603.00427

[8] S. Roman, *Advanced Linear Algebra*. Springer, 2007.

[9] A. H. Sayed, *Adaptive Filters*. Wiley-IEEE Press, 2008.

[10] A. J. McConnell, *Applications of Tensor Analysis*. Dover Publications, 2011.

[11] F. Kuech, A. Mitnacht, and W. Kellermann, "Nonlinear acoustic echo cancellation using adaptive orthogonalized power filters," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, vol. 3, March 2005, pp. iii/105–iii/108 Vol. 3.

[12] T. M. Panicker, V. J. Mathews, and G. L. Sicuranza, "Adaptive parallel-cascade truncated volterra filters," *IEEE Transactions on Signal Processing*, vol. 46, no. 10, pp. 2664–2673, Oct 1998.