A parallelizable recursive least squares algorithm for adaptive filtering, with very good tracking properties

C. Papaodysseus, E. Koukoutsis, C.C. Halkias & G. Roussopoulos

Department of Electrical and Computer Engineering, National Technical University of Athens, Division of Computer Engineering, Athens, GR, 157 73, Greece

Available online: 19 Mar 2007

To cite this article: C. Papaodysseus, E. Koukoutsis, C.C. Halkias & G. Roussopoulos (1998): A parallelizable recursive least squares algorithm for adaptive filtering, with very good tracking properties, International Journal of Computer Mathematics, 67:3-4, 275-292

To link to this article: http://dx.doi.org/10.1080/00207169808804665

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: http://www.tandfonline.com/page/terms-and-conditions

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae, and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand, or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.
A PARALLELIZABLE RECURSIVE LEAST SQUARES ALGORITHM FOR ADAPTIVE FILTERING, WITH VERY GOOD TRACKING PROPERTIES

C. PAPAODYSSEUS, E. KOUKOUTSIS, C. C. HALKIAS and G. ROUSSOPoulos

National Technical University of Athens, Department of Electrical and Computer Engineering, Division of Computer Engineering, GR-157 73 ATHENS, GREECE

(Received 19 July 1996; In final form 10 April 1997)

In this paper, a new Recursive Least Squares (RLS) algorithm for Finite Window Adaptive Filtering is presented, that has a number of interesting and useful properties. First, owing to the specific structure of the updating formulas and due to the fact that the past information is, for the first time, directly dropped by means of a proper inversion Lemma stated and proved in this paper, the proposed algorithm is immediately parallelizable. Second, it is more robust than many RLS Kalman-type schemes, in the sense that it is more resistant to the finite precision error effects. At the same time, the proposed algorithm has very good tracking capabilities. Finally, it can constitute the basis for the development of \(O(m)\) computational complexity algorithms that have very interesting properties, too, i.e. they are robust, parallelizable and they have particularly good tracking properties.

Keywords: Adaptive algorithms; recursive least squares (RLS) filtering; kalman-type parallel algorithms and filters

C.R. Categories: I4.4

1. INTRODUCTION

It is very well known that Recursive Least Squares (RLS) Adaptive Filtering has a wide variety of applications in adaptive control, in high resolution spectrum estimation, in echo cancellation and channel equalization, in system identification in general, in adaptive differential encoding, in interference suppression, in adaptive deconvolution, in biomedical signal processing and in many other fields (e.g. see [1], [4], [9], [17], [21], etc).
Kalman, first, introduced an RLS algorithm of $O(m^2)$ computational complexity, where the updating of the system matrix was performed by means of the Matrix Inversion Lemma at each time instant $n$. Subsequently, the first algorithm, of the so called Fast Kalman-type, that performs Recursive Least Squares Adaptive Filtering with $O(m)$ Multiplications and Divisions (MADs) per recursion was introduced by Ljung, Morf and Falconer [3]; in this scheme, the updating of the system matrix was made by means of the very fruitful concept of rank displacement introduced by Morf [2]. Subsequently, a class of somewhat faster algorithms, the FAEST ones, has been developed by Carayannis, Manolakis, Kalouptsidis, Halkias, et al. [1], [6] e.t.c., while the Fast Transversal Algorithm, similar to the FAEST, and a number of related computational schemes with exact initialization, have been proposed by Cioffi and Kailath, [8]. Also a group of normalized algorithms has been developed by Fabre, Gueguen, [16], etc. A stable version of the algorithm in [8] has been proposed for the first time by Slock and Kailath in an well known paper ([11]). All the aforementioned Adaptive Filtering algorithms use all the available data from the initial time instant $\phi$, up to the present generic instant $n$. They simply use a “forgetting factor” $\lambda$, so that an exponential age weighing of the old data is achieved. Most of the aforementioned computational schemes, clearly except this introduced in [11], suffer from serious numerical problems, owing to the finite precision with which computations are made (see for example [10], [11], [12], [14], [15], [16], [22], [25]). There are, however, cases, where the desired estimate highly depends on the remote data while, it is desirable that this estimate depends only on the $L$, say, most recent data. This is, for example, the case where the data sequence contains short-duration, impulse like, high amplitude interference, which influence the output for a very long period of time. Hence, in this case a “finite or sliding window” or, equivalently, a “finite memory” adaptive filtering is preferable (e.g. [7], [8]). A number of algorithms that perform such a finite window filtering have been developed, e.g. by Cioffi and Kailath, [8], Manolakis, Ling, Proakis [7], Kalouptsidis, Carayannis, Manolakis [6], (see also Honig and Messerchmitt [4]) e.t.c. These algorithms too, demonstrate serious numerical problems due to the finite word length ([7]). An algorithm analogous to the one in [11] for performing finite window RLS filtering with essentially improved numerical properties, has been proposed by Slock and Kailath in [10].

In this paper, a new Recursive Least Squares algorithm for finite window Adaptive Filtering is introduced, which, however, can be used in many instances as a good alternative of the growing (or “infinite”) window schemes. The proposed algorithm has an $O(m^2)$ computational complexity, but it has a number of interesting properties:
First, the involved quantities are updated by means of new recursive relations, so that the resulting computational scheme can be parallelized. When executed in parallel by two or more processors is essentially faster than the finite window Kalman algorithm.

Second, the algorithm presented here manifests a good combination of tracking capabilities and robustness, i.e., resistance to the finite precision problems. In fact, it must be pointed out that, in general, the tracking capabilities of an RLS algorithm are intimately connected to its robustness: if one pushes the algorithm to have optimum tracking properties, then this action, very frequently, causes a drastic loss of robustness and vice versa. Therefore, one must compare two algorithms in respect with the strongly interrelated notions of tracking performance and robustness simultaneously. If this comparison is made, then the introduced algorithm has a good performance, better than most other RLS algorithms that use either finite or "infinite" window, as it is confirmed by a large number of simulation experiments performed by the authors.

And, finally, the $O(m^2)$ algorithm introduced in this paper can constitute the basis for the development of $O(m)$ computational complexity Recursive Least Squares Algorithms, that demonstrate some very interesting properties too (see [29]; these results will be extended to forthcoming publications).

2. A BRIEF DESCRIPTION OF THE RECURSIVE LEAST SQUARES FILTERING IN THE FINITE WINDOW CASE

Suppose that a system is given, which for a specific input data sequence $x(n)$, $n = M, M + 1, \ldots$, produces a certain output signal $z(n)$, $n = M, M + 1, \ldots, M \in \mathbb{Z}$. The purpose of Recursive Least Squares (RLS) Filtering is to determine an estimate $\hat{z}(n)$ of the given output signal $z(n)$ so that $\hat{z}(n)$ is a linear combination of $m$ previous elements $x(n), x(n-1), \ldots, x(n-m+1)$ of the input signal. Or, equivalently, to determine the so called filter coefficients $c_1(n), c_2(n), \ldots, c_m(n)$, so that

$$\hat{z}(n) = -\sum_{i=1}^{m} c_i(n)x(n+1-i)$$  \hspace{1cm} (2.1)

is close to $z(n)$ in the sense of the Least-Squares (LS) criterion. This LS criterion consists in minimizing the sum

$$E_m(n) = \sum_{j=M}^{n} \lambda^{n-j}(z(j) - \hat{z}(j))^2$$  \hspace{1cm} (2.2)
over the discrete time interval \((M, n)\), where \(\lambda\) is a constant such that
\[
0 < \lambda \leq 1.
\] (2.3)

Formula (2.2) is quite frequently stated as follows:
\[
E_m(n) = \sum_{j=M}^{n} \lambda^{n-j} \cdot (\varepsilon_m(j))^2
\] (2.4)

where \(\varepsilon_m(j)\), the prediction error, is by definition,
\[
\varepsilon_m(j) = z(j) - \hat{z}(j)
\] (2.5)

If, moreover, one defines the vectors
\[
x_m(n) = [x(n) \ x(n-1) \ldots x(n-m+1)]^T
\] (2.6)
\[
c_m(n) = [c_1(n) \ c_2(n) \ldots c_m(n)]^T
\] (2.7)

then (2.2) becomes, equivalently,
\[
E_m(n) = \sum_{j=M}^{n} \lambda^{n-j} \cdot (z(j) + c_m(n)x_m(j))^2.
\] (2.8)

The demand that \(E_m(n)\) in (2.8) is minimum, is equivalent to the requirement of minimizing the sum of squares of the prediction error times \(\lambda^{n-j}\), over the window of observation \((M, n)\); the “forgetting” factor \(\lambda\) is introduced, in order to exponentially reduce the influence of the past prediction error in the whole process, and, therefore, make the algorithm adaptive. In the growing window case, the lower summation end in (2.8) is kept fixed, while in the case of finite (or sliding) window filtering which is examined in this paper, the length of the time interval \((M, n)\) over which summation in (2.8) occurs is kept fixed. To be more specific, in the finite window filtering case one uses only the output and input signal samples \([x(j), z(j)]\), \(j = \alpha, \alpha + 1, \ldots, n\), where \(n - \alpha = \beta\) is fixed at all times, and one completely ignores the samples previous to the time instant \(\alpha\). On this occasion, one determines 
\[c_1(n), c_2(n), \ldots, c_m(n)\] and therefore \(\hat{z}(n)\) for every time instant \(n\), by minimizing \(E_m(n)\) in (2.8). Since the right hand side of (2.8) is a second degree polynomial in \(c_i(n), i = 1, 2, \ldots, n\), in order to minimize it, one suffices to differentiate \(E_m(n)\) with respect to \(c_m(n)\) and set the derivative to zero. In this way, one obtains the normal equations:
\[
R_m(n)c_m(n) = -r_m(n)
\] (2.9)

where the \(m \times m\) matrix \(R_m(n)\) is defined via the formula
\[
R_m(n) = \sum_{j=M}^{n} \lambda^{n-j}x_m(j)x_m^T(j)
\] (2.10)
and the vector \( r_m(n) \) is

\[
    r_m(n) = \sum_{j=M}^{n} \lambda^{n-j} z(j)x_m(j),
\]

where it is stressed once more that for every time instant \( n \), the difference \( n - M \) is fixed. Hereafter we will express this difference by the constant \( L \):

\[
    L = n - M + 1,
\]

which we will call "the finite window length".

3. NEW RECURSIONS FOR FINITE WINDOW ADAPTIVE FILTERING

3.1. New Formulae for Recursive Updating of the System Matrix \( R_m(n) \)

So far, in the sliding window case, the updating of the system matrix \( R_m(n) \) in the \( O(m^2) \) computational complexity algorithms, is performed by two successive applications of the matrix Inversion Lemma, i.e. (see [8]) through the pair of formulae: relation (3.1)

\[
    R_{m,L+1}^{-1}(n + 1) = \frac{1}{\lambda} \left\{ R_{m,L}^{-1}(n) - \frac{R_{m,L}^{-1}(n)x_m(n+1)x_m^T(n+1)R_{m,L}^{-1}(n)}{\lambda + x_m^T(n+1)R_{m,L}^{-1}(n)x_m(n+1)} \right\}
\]

and

\[
    R_m^{-1}(n + 1) = R_{m,L+1}^{-1}(n + 1) + \lambda L \cdot \frac{R_{m,L+1}^{-1}(n)x_m(M)x_m^T(M)R_{m,L+1}^{-1}(n)}{1 - \lambda^2 x_m^T(M)R_{m,L+1}^{-1}(n+1)x_m(M)}. \tag{3.1.2}
\]

In this way, one obtains a finite memory Least Squares filter, that consists of two growing memory filters running in parallel (e.g. [4], [6], [7], [8]). In this case, in practice, one carries most of the problems that are intrinsic to the infinite window RLS algorithms, to the finite window computational schemes too, frequently amplified. For example, in most of the existing finite window schemes the two growing memory filters running in parallel contribute additively to the generation of the finite word length error (e.g. [4], [6], [7], [25]). A major exception is the scheme by Slock and Kailath introduced in [10].

On the contrary in this paper, the updating of the system matrix \( R_m(n) \) is achieved through a single formula, by means of a Lemma which is stated and proved here. This, together with a class of alternative definitions and
new recursive formulas, will lead to a finite window Recursive Least Squares algorithm, that has the afore-mentioned interesting properties.

In fact, if for simplicity, one symbolizes the vector $x_m(n + 1)$ containing the new coming information with $x$ and the vector $x_m(M)$ including the dropped information $x(M)$ with $y$, then it follows from (2.10) that the system matrix $R_m(n + 1)$ is equal to

$$R_m(n + 1) = \lambda R_m(n) + xx^T - \mu yy^T$$  \hfill (3.1.3)

where

$$\mu = \lambda^L. \hfill (3.1.4)$$

At this point, the inverse of the system matrix $R_m(n + 1)$ will be computed, by means of a single formula recursively, as a function of the inverse of the system matrix at the time instant $n R_m^{-1}(n)$, and of the vectors $x_m(n + 1)$ and $x_m(M)$. This will be achieved by means of the following:

**LEMMA** Consider the $m \times m$ symmetric matrices $A$ and $B$, as well as the $m \times 1$ vectors $x$ and $y$. Suppose, moreover, that the matrices $A$, $B$ and $(B-yy^T)$ have an inverse and that the following relation holds:

$$A = B + xx^T - yy^T. \hfill (3.1.5)$$

Then, the inverse of $A$ equals:

$$A^{-1} = B^{-1} - \frac{B^{-1}xx^T B^{-1}(1 - y^T B^{-1}y) - B^{-1}yy^T B^{-1}(1 + x^T B^{-1}x)}{(1 - y^T B^{-1}y)(1 + x^T B^{-1}x) + y^T B^{-1}xx^T B^{-1}y} + \frac{B^{-1}xx^T B^{-1}yy^T B^{-1} + B^{-1}yy^T B^{-1}xx^T B^{-1}}{(1 - y^T B^{-1}y)(1 + x^T B^{-1}x) + y^T B^{-1}xx^T B^{-1}y}. \hfill (3.1.6)$$

**Proof** One may get a lengthy but rather straightforward proof, by applying the Matrix Inversion Lemma twice. Equivalently, one may deduce the formula if one applies the inversion Lemma for a matrix polynomial of the form

$$A = B + CD \hfill (3.1.7a)$$

by setting

$$C = [-y \ x], \quad D = [y \ x]^T. \hfill (3.1.7b)$$

Q.E.D.

If one uses the above theorem and if, moreover, in relation (3.1.6), one makes the substitutions:

$$A = R_m(n + 1) \hfill (3.1.8)$$
$$B = \lambda \cdot R_m(n) \hfill (3.1.9)$$
$$y \rightarrow \mu^{1/2}y \hfill (3.1.10)$$
then, one obtains immediately

\[(R_m(n + 1) + x x^T - \mu y y^T)^{-1}\]

\[= \frac{1}{\lambda} R_m^{-1}(n) - \frac{1}{\lambda} R_m^{-1}(n + 1) x x^T R_m^{-1}(n) \left( 1 - \frac{\mu}{\lambda} y y^T R_m^{-1}(n) y \right) \Delta(n)\]

\[+ \frac{\mu}{\lambda^3} R_m^{-1}(n) y y^T R_m^{-1}(n) \Delta(n)\]

\[+ \frac{\mu}{\lambda^3} R_m^{-1}(n) x x^T R_m^{-1}(n) \Delta(n),\]  (3.1.11)

where, relation (3.1.12)

\[\Delta(n) = \left( 1 + \frac{1}{\lambda} x^T R_m^{-1}(n) x \right) \left( 1 - \frac{\mu}{\lambda} y^T R_m^{-1}(n) y \right)\]

\[+ \mu y^T \frac{1}{\lambda} R_m^{-1}(n) x x^T R_m^{-1}(n) y \frac{1}{\lambda} \ldotp\]

### 3.2 A New Formula for Updating the Filter Coefficients \(c_m(n)\)

The normal equations at the time instant \((n + 1)\) are:

\[R_m(n + 1) c_m(n + 1) = -r_m(n + 1),\]  (3.2.1)

where, from the minimization of (2.8) it holds that

\[R_m(n + 1) = \sum_{j=M+1}^{n+1} \lambda^{n-j} x_m(j) x_m^T(j)\]  (3.2.2)

which implies, after a straightforward calculation, that

\[R_m(n + 1) = \lambda R_m(n) + x_m(n + 1) x_m^T(n + 1) - \mu x_m(n - L + 1) x_m^T(n - L + 1).\]  (3.2.3)

Moreover,

\[r_m(n + 1) = \sum_{j=M+1}^{n+1} \lambda^{n-j} z(j) x_m(j)\]  (3.2.4)
which gives the following recursive relation for the vector $r_m(n + 1)$

$$r_m(n + 1) = \lambda r_m(n) + z(n + 1)x_m(n + 1) - \mu z(n - L + 1)x_m(n - L + 1). \quad (3.2.5)$$

Now, we define two Kalman-type gains through the formulas (see [4], too):

$$R_m(n + 1)u_m^*(n + 1) = -x_m(n + 1) \quad (3.2.6)$$

$$R_m(n + 1)v_m^*(n + 1) = -x_m(n - L + 1) \equiv -x_m(M) \quad (3.2.7)$$

Moreover, we define the “leading prediction error”

$$f_m(n + 1) = z(n + 1) + c_m^T(n)x_m(n + 1) \quad (3.2.8)$$

and the “trailing prediction error”

$$g_m(n + 1) = z(n - L + 1) + c_m^T(n)x_m(n - L + 1) \equiv z(M) + c_m^T(n)x_m(M). \quad (3.2.9)$$

Then, (3.2.1) is transformed into relation (3.2.10)

$$R_m(n + 1)\{c_m(n) + f_m(n + 1)u_m^*(n + 1) - \mu g_m(n + 1)v_m^*(n + 1)\} = -r_m(n + 1).$$

A direct comparison of (3.2.1) and (3.2.10) implies that the filter coefficients vector is updated through the formula:

$$c_m(n + 1) = c_m(n) + f_m(n + 1)u_m^*(n + 1) - \mu g_m(n + 1)v_m^*(n + 1). \quad (3.2.11)$$

### 3.3. Novel Formulas for the Updating of the Inverse of the System Matrix $R_m^{-1}(n + 1)$ and of Two Pairs of Kalman-type Gains

Subsequently, we define two alternative gains through the formulas:

$$R_m(n)u_m(n + 1) = -\frac{1}{\lambda}x_m(n + 1) \quad (3.3.1)$$

$$R_m(n)v_m(n + 1) = -\frac{1}{\lambda}x_m(n - L + 1) \iff$$

$$R_m(n)v_m(n + 1) = -\frac{1}{\lambda}x_m(M). \quad (3.3.2)$$

The updating of the two defined above alternative Kalman gains, may be immediately achieved. In fact, (3.3.1) and (3.3.2) imply, immediately, that

$$u_m(n + 1) = -\frac{1}{\lambda}R_m^{-1}(n)x_m(n + 1) \quad (3.3.3)$$

$$v_m(n + 1) = -\frac{1}{\lambda}R_m^{-1}(n)x_M(M) \quad (3.3.4)$$
Next, the formulas for the recursive updating of the two gains $u_m(n+1)$ and $v_m(n+1)$ will be derived. In fact, if one uses the definitions (3.3.1) and (3.3.2), then (3.1.1) is transformed into

$$R_m^{-1}(n+1) = \frac{1}{\lambda} R_m^{-1}(n)$$

$$- \frac{\delta_x u_m(n+1) u_m^T(n+1) - \mu \delta_y v_m(n+1) v_m^T(n+1)}{\Delta(n)}$$

$$+ \mu \delta_{xy} \cdot \frac{u_m(n+1) v_m^T(n+1) + v_m(n+1) u_m^T(n+1)}{\Delta(n)}$$

(3.3.5)

where,

$$\delta_{xx} = u_m^T(n+1) x_m(n+1)$$

(3.3.6)

$$\delta_{xy} = u_m^T(n+1) x_m(M) = v_m^T(n+1) x_m(n+1)$$

(3.3.7)

$$\delta_{yy} = v_m^T(n+1) x_m(M)$$

(3.3.8)

$$\delta_x = 1 - \delta_{xx}$$

(3.3.9)

$$\delta_y = 1 + \mu \delta_{yy}.$$  

(3.3.10)

Upon right multiplying both sides of (3.5.19) by $-(1/\lambda) x_{m-1}(n+1)$, and since

$$\frac{1}{\lambda} R_m^{-1}(n+1) x_m(n+1) = u_m^* (n+1), \forall n,$$

(3.3.11)

$$x_m^{-1}(n+1) = [x(n+1) x_m(n)]^T$$

(3.3.12)

it follows, after a certain elaboration, that

$$u_m^*(n+1) = \frac{\delta_y}{\Delta(n)} u_m(n+1) - \mu \cdot \frac{\delta_{xy}}{\Delta(n)} v_m(n+1)$$

(3.3.13)

where, now, $\Delta(n)$ is expressed as

$$\Delta(n) = \delta_x \delta_y + \mu \delta_{xy}^2.$$  

(3.3.14)

Similarly, if one right multiplies (3.3.5) with $x_m(M)$ and uses (3.3.11), (3.3.1) and (3.3.2), then one may deduce that

$$v_m^*(n+1) = \frac{\delta_{xy}}{\Delta(n)} u_m(n+1) + \frac{\delta_x}{\Delta(n)} v_m(n+1).$$

(3.3.15)

Therefore, both pairs of gains have been updated and by means of them the system matrix has been updated as well.
3.4 The Proposed RLS Adaptive Algorithm

1. Quantities known at the time instant $n$.

\[ x^m(M), x^m(n), R^{-1}_m(n), c^m(n). \]

2. Recursive computation of the two pairs of Kalman-type gains and of the system matrix.

2.1. Updating of the pair of alternative gains $u^m(n + 1)$ and $v^m(n + 1)$

\[ u^m(n + 1) = -\frac{1}{\lambda} R^{-1}_m(n)x^m(n + 1) \quad \text{#MADS : } m^2 + m \quad (3.4.1) \]

\[ v^m(n + 1) = -\frac{1}{\lambda} R^{-1}_m(n)x^m(M) \quad \text{#MADS : } m^2 + m \quad (3.4.2) \]

2.2. Computation of useful intermediate quantities.

\[ \delta_{xx} = u^T_m(n + 1)x^m(n + 1) \quad \text{#MADS : } m \quad (3.4.3) \]

\[ \delta_{xy} = u^T_m(n + 1)x^m(M) = v^T_m(n + 1)x^m(n + 1) \quad \text{#MADS : } m \quad (3.4.4) \]

\[ \delta_{yy} = v^T_m(n + 1)x^m(M) \quad \text{#MADS : } m \quad (3.4.5) \]

\[ \delta_x = 1 - \delta_{xx} \quad \text{#MADS : } 0 \quad (3.4.6) \]

\[ \delta_y = 1 + \mu \delta_{yy} \quad \text{#MADS : } 1 \quad (3.4.7) \]

\[ \Delta(n) = \delta_x \delta_y + \mu \delta_{xy}^2 \quad \text{#MADS : } 3 \quad (3.4.8) \]

2.3. Updating of the pair of gains $u^*_m(n+1)$ and $v^*_m(n+1)$

\[ u^*_m(n + 1) = \frac{\delta_y}{\Delta(n)} u^m(n + 1) - \mu \cdot \frac{\delta_{xy}}{\Delta(n)} v^m(n + 1) \quad \text{#MADS : } 2m + 3 \quad (3.4.9) \]

\[ v^*_m(n + 1) = \frac{\delta_{xy}}{\Delta(n)} u^m(n + 1) + \frac{\delta_x}{\Delta(n)} v^m(n + 1) \quad \text{#MADS : } 2m + 2 \quad (3.4.10) \]

2.4. Updating of the system matrix

\[ R^{-1}_m(n + 1) = \frac{1}{\lambda} R^{-1}_m(n) - \mu \delta_{xy} u^T_m(n + 1) - \mu \delta_y v^T_m(n + 1) \quad \frac{\delta_{xy}}{\Delta(n)} \quad (3.4.12) \]

\[ \Delta(n) \]


\[ x \times \#MADS : 2m^2 + 2m + 3 \]
\[ + \mu \delta_{xy} \cdot \frac{u_m(n+1)w_m^T(n+1) + v_m(n+1)u_m^T(n+1)}{\Delta(n)} \]
\[ \times \#MADS : m^2 + m + 2 \]  \hspace{1cm} (3.4.11)

3. Recursive computation of the FIR Filter.

\[ f_m(n+1) = z(n+1) + c_m^T(n)x_m(n+1) \quad \#MADS : m \]  \hspace{1cm} (3.4.12)
\[ g_m(n+1) = z(n - L + 1) + c_m^T(n)x_m(n - L + 1) \quad \#MADS : m \]  \hspace{1cm} (3.4.13)
\[ c_m(n+1) = c_m(n) + f_m(n+1)u_m^* (n+1) - \mu g_m(n+1)v_m^* (n+1) \]
\[ \times \#MADS : 2m + 1 \]  \hspace{1cm} (3.4.14)

4. PROPERTIES OF THE PROPOSED ALGORITHM

4.1 The Proposed Algorithm Combines Good Tracking Capabilities and Noteworthy Robustness

Clearly, one may expect that any finite window adaptive algorithm has essentially better tracking properties than the infinite window RLS schemes, since the burdening information previous to a certain, application dependent, time instant \( n - L - m + 1 \) is necessary dropped.

It must be stressed, once more however, that, in general, the tracking properties of a RLS algorithm are intimately connected to its robustness, namely its resistance to the finite precision problems. Therefore, one must compare the tracking properties of two RLS schemes only if they manifest very similar robustness and vice versa, one must test their robustness, only when they exhibit very similar tracking properties.

Concerning robustness, it will be stressed once more that, ([6], [7], [8], etc), the previous finite window algorithms make use of two strongly coupled growing memory LS filters; thus, in most of them, the finite precision error generated in the computation of those two filters, adds up. In the algorithm introduced here, since the computation of the quantities related to the incoming novel information is completely uncoupled from the computation of the quantities related to the dropped information, one may expect that no adding up of the generated finite precision error occurs. This can be demonstrated on the basis of the general methodology introduced in [23], [24], [25], [26], [27], [28], etc. Consequently, the proposed algorithm proves to be more robust than...
many classical RLS schemes that perform adaptive filtering by means of either a finite or an infinite window. Two major exceptions are the algorithms by Slock and Kailath namely the one in [11] which is practically stable, and the one in [10] that is essentially robust.

The aforementioned results have been confirmed by a considerable number of simulation experiments. Notice, that when the excellent tracking capabilities are not a primary demand, then the proposed algorithm may prove to be very robust, since it may run in a completely satisfactory way at least two hundred fifty times longer than the classical growing Kalman, Fast Kalman and the FAEST schemes, still having equal or better tracking capabilities. Notice that the computational scheme introduced here can be stabilized by means of rather simple techniques, according to the information.

The noteworthy robustness and the good tracking capabilities of the proposed algorithm have been tested by a large number of simulation experiments and are obvious too in the graphs of the estimated output $\hat{y}(n)$ produced by the new algorithm depicted in Figure 1, 2 and 3, where the results of two system identification simulation experiments are presented. In fact:

In Figures 1 and 2, the way the introduced algorithm tracks a very difficult to follow by most RLS algorithms signal is presented. This signal is depicted in subplot (1a), and is the actual digitized recording of a musical instrument (a guitar). In subplot (1b) the estimated response of the proposed scheme is shown. A white noise was chosen as input for the algorithm, a system order $m = 50$, a window length $L = 200$ and a forgetting factor $\lambda = 0.97$. In Figure 2, the two signals, the original one and the estimated by the introduced algorithm are shown for a small time interval, so that the tracking capabilities of the proposed algorithm can be clearly appreciated. In general, the performed experiments manifest that the introduced scheme can

![FIGURE 1 Subplot (a): The actual digitized recording of a musical instrument (a guitar).](image-url)
FIGURE 1 Subplot (b): The way the proposed algorithm approximates the very fast varying signal depicted in subplot (a) of the present figure. The input is a white noise. System order $m = 50$, window length $L = 200$, $\lambda = 0.97$.

FIGURE 2 The two signals of Figure 1, plotted together for a relatively small time interval, so that the way the introduced algorithm tracks the actual signal can be looked at in detail. Continuous line: the output of the introduced algorithm. Dashed line: the actual fast varying musical signal.

be used for processing of such very-difficult-to-track signals as music and speech.

In Figure 3, the way the proposed algorithm tracks a fast varying signal is shown. The input is a cosine with period 800, while the system order is $m = 25$, the window length is $L = 150$ and the forgetting factor $\lambda = 0.97$. 
4.2 The Proposed Algorithm is Parallelizable and its Parallel Form is Faster than the Original Kalman RLS Algorithm

It follows from the definition of the two pairs of gains \((u_m(n), v_m(n))\) and \((u'_m(n), v'_m(n))\) and from all the related definitions, that the formulas that perform their computation are essentially uncoupled. Moreover, all the computations related with the dropped information are also uncoupled from those connected with the incoming one. This suggests the idea that the algorithm introduced here is parallelizable. In fact this is true, and as it will become evident from the subsequent analysis, when the proposed scheme is executed by two or more processors, then the resulting parallel form is essentially faster than the finite window Kalman RLS algorithm. Indeed, with 4 processors, the new algorithm performs the FIR filtering in \(2m^2 + 3m + 4\) Multiplications and Divisions per recursion as compared to the \(5m^2 + 11m\) MADs required by the
finite window Kalman scheme. In fact, the algorithm introduced in this paper may be executed by 4 processors, as follows:

STEP 1

Processor number

#1 \( u_m(n + 1) \) computation (formula (3.4.1))  
# MADS: \( m^2 + m \)

#2 \( v_m(n + 1) \) computation (formula (3.4.2))  
# MADS: \( m^2 + m \)

#3 computation of \( \frac{1}{\lambda} R_m^{-1}(n) \) (first term of formula (3.4.11))  
# MADS: \( (m^2 + m)/2 \)

#4 \( f_m(n + 1) \) and \( g_m(n + 1) \) computation (formulas (3.4.12) and (3.4.13))  
# MADS: \( 2m \)

Maximum number of MADS in this step \( m^2 + m \).

STEP 2

Processor number

#1 \( \delta_{xx} \) computation (formula (3.4.3))  
# MADS: \( m \)

#2 \( \delta_{yy} \) computation (formula (3.4.5))  
# MADS: \( m \)

#3 \( \delta_{xy} \) computation (formula (3.4.4))  
# MADS: \( m \)

Maximum number of MADS in this step \( m \).

STEP 3

Processor number

#1 \( (\delta_y/\Delta(n))u_m(n + 1) \) computation (in formula (3.4.9))  
# MADS: \( m + 1 \)

#2 \( (\mu \delta_{xy}/\Delta(n))w_m(n + 1) \) computation (in formula (3.4.9))  
# MADS: \( m + 2 \)

#3 \( (\delta_{xy}/\Delta(n))u_m(n + 1) \) computation (in formula (3.4.10))  
# MADS: \( m + 1 \)

#4 \( (\delta_x/\Delta(n))v_m(n + 1) \) computation (in formula (3.4.10))  
# MADS: \( m + 1 \)

Maximum number of MADS in this step \( m + 2 \).
STEP 4

Processor number

#1 \( \frac{\delta_y}{\Delta(n)} u_m(n + 1) u_m^T(n + 1) \) computation (in formula (3.4.11))
# MADS: \((m^2 + m + 1)\)

#2 \( \mu \frac{\delta_y}{\Delta(n)} v_m(n + 1) v_m^T(n + 1) \) computation (in formula (3.4.11))
# MADS: \((m^2 + m + 2)\)

#3 \( \mu \frac{\delta_x}{\Delta(n)} u_m(n + 1) v_m^T(n + 1) \) computation (in formula (3.4.11))
# MADS: \((m^2 + m + 2)\)

#4 \( f_m(n + 1) u_m^T(n + 1) \) (formula (3.4.14))
\( \mu g_m(n + 1) v_m^T(n + 1) \) (formula (3.4.14))
# MADS: \(2m + 1\)

Maximum number of MADS in this step \((m^2 + m + 2)\)
Total number of Multiplications and Divisions (MADS) for FIR filtering:
\[2m^2 + 3m + 4\]

Notice, that for a system order \(m\) greater than four \((m > 4)\), one may exploit much better the processors and reduce the number of steps to two.

4.3 The Scheme Introduced Here Can Form the Basis for the Development of \(O(m)\) Computational Complexity RLS Adaptive Algorithms

In order to achieve the development of such fast schemes for performing adaptive filtering by means of a finite window, one must employ the same new definitions of the various quantities given in this paper, but, in addition, one must further exploit the structural properties of the system matrix \(R_m(n)\) and must employ the powerful concept of “rank displacement” introduced by Morf [2] and successfully used by Ljung, Morf, Falconer, [3], Carayannis et al. [1], [6], Fabre, Gueguen [16], etc. This is briefly presented in [29] and it will be treated extensively in forthcoming publication.

CONCLUSION

In this paper a new computational scheme is introduced for performing Recursive Least Squares Adaptive Filtering. This algorithm uses a finite window and has a number of interesting properties. First, it has very good tracking capabilities and, due to its particular structure and the mathematical definitions behind it, is parallelizable. When it is executed in parallel, is essentially faster.
than the finite window Kalman scheme. In addition, the proposed algorithm is robust and it may be the basis for the development of $O(m)$ RLS schemes with some particularly good properties, too.

References


