Fast RLS Laguerre Adaptive Filtering *

RICARDO MERCHED AND ALI H. SAYED

Electrical Engineering Department
University of California
Los Angeles, CA 90095

Abstract

The existing derivations of conventional fast RLS adaptive filters are intrinsically dependent on the shift structure in the input regression vectors. This structure arises when a tapped-delay line (FIR) filter is used as a modeling filter. In this paper, we show that a more general data structure is induced by other filter implementations, such as Laguerre-based filters and, more importantly, that an exact fast RLS algorithm can still be derived for such Laguerre-induced data structures. One of the benefits of working with a Laguerre basis is that fewer parameters can be used to model long impulse responses.

1 Introduction

Fast RLS adaptive filtering algorithms represent an attractive way to compute the least squares solution of growing length data efficiently. While the conventional RLS algorithm requires $O(M^2)$ computations per sample, where $M$ is the filter order, its fast counterparts require only $O(M)$ operations. Examples of such fast schemes include the fast aposteriori error sequential technique (FAEST) [1], the fast transversal filter algorithm (FTF) [2], and least-squares lattice algorithms [3, 4, 5]. The latter class of algorithms deal with order-recursive structures, while the first two examples (FTF and FAEST) deal with fixed-order structures; both FTF and FAEST can also be viewed as special cases of a general fast estimation algorithm for state-space models, known as the (extended) Chandrasekhar recursions [6, 7].

The low complexity that is achieved by these algorithms is a direct consequence of the shift structure that is characteristic of regression vectors in FIR adaptive structures. This fact is evident in the conventional derivations of fast adaptive algorithms; all of which rely heavily on the shift structure. The arguments in [6, 7], however, have shown that fast RLS algorithms can still be derived for certain more general structures in the regression vectors, other than the shift structure. In this paper, we show that input regression vectors that arise from Laguerre-based networks satisfy the conditions required in [7] and, therefore, that fast RLS algorithms can still be derived in this context.

*This material was based on work supported in part by the National Science Foundation under Award CCR-9732376. The work of R. Merched was also in part supported by a fellowship from CAPES, Brazil.
There are several important reasons to consider Laguerre basis functions instead of the usual \textbf{FIR} implementations. First, the use of Laguerre models (or more generally, orthonormal basis functions) to describe the dynamical behavior of a wide class of systems has been studied extensively in many recent works on system identification and control [8]-[11]. Second, the orthonormality property of Laguerre models offers many benefits in estimation problems, including better numerical conditioning of the data. Third, one of the primary motivations in using all-pass basis functions for adaptive filtering is the fact that it requires fewer parameters to model systems with long impulse responses.

In echo cancelation applications, for example, a long \textbf{FIR} filter may be necessary to model the echo path and adaptive \textbf{IIR} techniques have been proposed as possible alternatives (e.g., [12]-[13]). These techniques are nevertheless known to face stability problems due to the arbitrary pole locations during filter operation. Laguerre-domain adaptive filtering can offer an attractive alternative since, in this case, the pole location is fixed. It has already been suggested for echo cancelation and equalization applications in the works [14, 15]. However, these earlier contributions rely only on a slow \textbf{RLS}-type form for Laguerre adaptive filtering that requires \mathcal{O}(M^2) operations. Fast \mathcal{O}(M) Laguerre adaptive filters have been proposed before in the literature, but only as extensions of the classical gradient adaptive lattice (GAL) algorithm, which relies on LMS-type recursions and not on RLS-type recursions (see [16]).

In this paper, we show that a fast \mathcal{O}(M) Laguerre adaptive filter can actually be derived in the least-squares domain, thus leading to a fast RLS Laguerre adaptive scheme. One of the advantages of insisting on a least-squares-based adaptive scheme is that these tend to exhibit faster convergence rates and smaller misadjustments than gradient-based adaptive schemes. The algorithm of this paper will be presented in a compact array form, where arrays of numbers are transformed via suitably chosen elementary rotations. Such array forms exhibit better numerical properties than conventional descriptions in terms of explicit sets of equations.

2 \hspace{1em} The \textbf{RLS} Algorithm

Given a column vector \( y \in \mathbb{C}^{N+1} \) and a data matrix \( H \in \mathbb{C}^{(N+1) \times M} \), the exponentially-weighted least squares problem seeks the column vector \( w \in \mathbb{C}^M \) that solves

\[
\min_w \left[ \lambda^{N+1} w^* \Pi_0^{-1} w + (y - Hw)^* W (y - Hw) \right] .
\]  

The matrix \( \Pi_0 \) is a positive-definite regularization matrix, and \( W = (\lambda^N \oplus \lambda^{N-1} \oplus \cdots \oplus 1) \) is a weighting matrix that is defined in terms of a forgetting factor \( \lambda \). In tracking problems, typically we have \( 0 \ll \lambda < 1 \). The symbol \( \ast \) denotes complex conjugate transposition.

The individual entries of \( y \) will be denoted by \( \{d(i)\} \), and the individual rows of \( H \) will be denoted by \( \{u_i\} \).

\[
y = \begin{bmatrix} d(0) \\ d(1) \\ \vdots \\ d(N) \end{bmatrix}, \quad H = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{bmatrix} .
\]

The RLS algorithm computes the optimal solution of problem (1) recursively as follows:

\[
w_{i+1} = w_i + g_{i+1}[d(i + 1) - u_{i+1} w_i] , \quad w_1 = 0 , \tag{2}
\]

\[
g_{i+1} = \lambda^{-1} P_i u_{i+1}^* \gamma(i + 1) , \tag{3}
\]

\[
\gamma^{-1}(i + 1) = 1 + \lambda^{-1} u_{i+1} P_i u_{i+1}^* , \tag{4}
\]

\[
P_{i+1} = \lambda^{-1} P_i - g_{i+1} \gamma^{-1}(i + 1) g_{i+1}^* , \quad P_{-1} = \Pi_0 . \tag{5}
\]
The computation of the gain vector $g_{i+1}$ in the above solution relies on the propagation of the Riccati variable $P_i$. This method of computation requires $O(M^2)$ operations per iteration. Fast $O(M)$ RLS schemes, on the other hand, avoid the propagation of $P_i$ and therefore evaluate the necessary gain vector in an alternative more efficient manner. It turns out that the choice of $\Pi_0$ plays a crucial role in the derivation of such fast schemes, as we now explain.

Assume for the time being that there exists a square matrix $\Phi$ that relates two successive regression vectors as

$$u_i = u_{i+1} \Phi .$$

Then using Eq. (2) we can relate the two successive gain vectors $\{g_i, g_{i+1}\}$ as follows. Note that

$$\Phi g_i \gamma^{-1}(i) = \lambda^{-1} \Phi P_{i-1} u_i^* = \lambda^{-1} \Phi P_{i-1} \Phi^* u_{i+1}^* .$$

Subtracting this equality from $g_{i+1} \gamma^{-1}(i + 1) = \lambda^{-1} P_i u_{i+1}^*$, we find that

$$g_{i+1} \gamma^{-1}(i + 1) = \Phi g_i \gamma^{-1}(i) + \lambda^{-1} (P_i - \Phi P_{i-1} \Phi^*) u_{i+1}^* .$$

This equation shows that in order to update the (scaled) gain vector from time $i$ to time $i + 1$, it is not necessary to evaluate the individual $\{P_i\}$ but rather the differences

$$\nabla \{P_i, \Phi\} = P_i - \Phi P_{i-1} \Phi^* .$$

It turns out that such differences can be updated fast for certain choices of $\Pi_0$ and $\Phi$ (as explained in [6, 7] in a more general state-space context), and this fact can be used to derive fast RLS schemes for input data vectors with or without shift structure.

### 3 Fast RLS Laguerre Algorithm

Thus consider the Laguerre-based model of Figure 1 where

$$H_0(z) = \frac{\sqrt{1 - a^2}}{1 - az^{-1}} \quad \text{and} \quad H(z) = \frac{z^{-1} - a}{1 - az^{-1}}, \quad |a| < 1 .$$ (9)

Note that $H(z)$ is a first-order all-pass system and that, unlike a general IIR structure, the poles of the Laguerre-based model are fixed at $a$. $^1$ [The choice of $a$ can be optimized off-line according to some criterion.] The input to the Laguerre filter at time $i$ is denoted by $x(i)$, and the coefficients that combine the outputs of the successive sections $\{H_0(z), H(z)\}$ are denoted by $\{w_k\}$.

$^1$The case of complex poles can also be handled. For simplicity, we assume real $a$. 


Let

\[ u_i = \frac{1}{a} u(i) - \frac{\sqrt{1 - a^2}}{a} x(i + 1) \]

\[ u_1 = \frac{1}{a} u(1) + \left(1 - \frac{1}{a^2}\right) u(0) + \frac{\sqrt{1 - a^2}}{a^2} x(i + 1) \]

\[ u_2 = \frac{1}{a} u(2) + \left(1 - \frac{1}{a^2}\right) u(1) - \frac{1}{a} \left(1 - \frac{1}{a^2}\right) u(0) + \frac{\sqrt{1 - a^2}}{a^3} x(i + 1) \]

In matrix form, we can express the above difference equations as (say, for \( M = 4 \)),

\[ u_i = [u_{i+1} \ x(i + 1)] \begin{bmatrix} T \\ v \end{bmatrix}, \tag{10} \]

where \( T \) is the \((M \times M)\) upper triangular Toeplitz matrix

\[ T = \begin{bmatrix} \frac{1}{a} & \frac{1}{a} & \frac{1}{a} & \frac{1}{a} \\ 0 & 1 - \frac{1}{a^2} & \frac{1}{a} & \frac{1}{a} \\ 0 & 0 & 1 - \frac{1}{a^2} & \frac{1}{a} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{11} \]

and \( v \) is the \((M \times 1)\) row vector

\[ v = \begin{bmatrix} -\frac{\sqrt{1-a^2}}{a} & \frac{\sqrt{1-a^2}}{a^2} & -\frac{\sqrt{1-a^2}}{a^3} & \frac{\sqrt{1-a^2}}{a^4} \end{bmatrix}. \tag{12} \]

Except for the additional term \( x(i + 1) \), relation (10) is of the same form as (6). This slight difference in the nature of the relations can be handled easily by properly defining extended quantities.
Thus note that using Eq. (2), we can write

\[
\begin{bmatrix}
g_{i+1}^{-1}(i + 1) \\
0
\end{bmatrix} = \lambda^{-1} \begin{bmatrix}
P_i & 0 \\
0 & 0
\end{bmatrix} \begin{bmatrix}
u_i^* \\
x^*(i + 1)
\end{bmatrix}.
\] (13)

Likewise, we have

\[
\begin{bmatrix}
T \\
v
\end{bmatrix} g_i^{-1}(i) = \lambda^{-1} \begin{bmatrix}
T \\
v
\end{bmatrix} P_{i-1} u_i^* = \lambda^{-1} \begin{bmatrix}
P_i & 0 \\
0 & 0
\end{bmatrix} P_{i-1} \begin{bmatrix}
T^* & v^*
\end{bmatrix} \begin{bmatrix}
u_i^* \\
x^*(i + 1)
\end{bmatrix}.
\] (14)

Subtracting (13) from (14), we obtain

\[
\begin{bmatrix}
g_{i+1}^{-1}(i + 1) \\
0
\end{bmatrix} - \begin{bmatrix}
T \\
v
\end{bmatrix} g_i^{-1}(i) = \lambda^{-1} \left( \begin{bmatrix}
P_i & 0 \\
0 & 0
\end{bmatrix} - \begin{bmatrix}
P_i & 0 \\
0 & 0
\end{bmatrix} \begin{bmatrix}
T^* & v^*
\end{bmatrix} \right) \begin{bmatrix}
u_i^* \\
x^*(i + 1)
\end{bmatrix}
\] (15)

This shows that in order to update the (scaled) gain vector from \( g_i \) to \( g_{i+1} \), we only need to know how to update the difference

\[
\nabla_{\{P_i, \Phi\}} \triangleq \begin{bmatrix}
P_i & 0 \\
0 & 0
\end{bmatrix} - \Phi \begin{bmatrix}
P_{i-1} & 0 \\
0 & 0
\end{bmatrix} \Phi^*,
\]

where we are now defining the \((M + 1) \times (M + 1)\) square matrix \( \Phi \) as

\[
\Phi = \begin{bmatrix}
T & 0 \\
v & 0
\end{bmatrix}.
\]

We shall now verify that it is possible to update the differences \( \nabla_{\{P_i, \Phi\}} \) efficiently. For this purpose, we first note that for the case of pre-windowed input data (i.e., \( x(i) = 0 \) for \( i \leq 0 \) and zero initial conditions), we obtain at the initial time instant \( i = 0 \),

\[
\nabla_{\{P_0, \Phi\}} = \lambda^{-1} \Pi_0 \\
0
\]

where \( \Pi_0 \) is chosen such that the above difference has low rank (say \( r \), where \( r \) is independent of \( M \) — see Sec. 5). We can then factor \( \nabla_{\{P_0, \Phi\}} \) as

\[
\nabla_{\{P_0, \Phi\}} = L_0 J L_0^*,
\] (17)

where \( L_0 \) is \((M + 1) \times r\) and \( J \) is a signature matrix with as many \( \pm 1 \)'s as \( \nabla_{\{P_0, \Phi\}} \) has positive or negative eigenvalues. It will then follow that the rank of all successive differences, \( \nabla_{\{P_i, \Phi\}} \) for \( i > 0 \), will not exceed \( r \) and, more importantly, that the inertia of all these successive differences can also be taken as \( J \). In other words, by forcing the initial difference to have low rank, and a certain inertia, we end up forcing all successive differences to have a similar property. This fact is essential to the derivation of a fast algorithm.

To establish the above claims we proceed by induction. Thus assume that the difference \( \nabla_{\{P_i, \Phi\}} \) at time \( i \) can be factored as \( \nabla_{\{P_i, \Phi\}} = L_i J L_i^* \) for some \((M + 1) \times r\) matrix \( L_i \). Define further, for compactness of notation, the extended quantities:

\[
\bar{a}_{i+1} \triangleq [u_{i+1} \ x(i + 1)], \quad \bar{g}_i \triangleq \begin{bmatrix}
g_i \\
0
\end{bmatrix}, \quad \bar{k}_i = \bar{g}_i \gamma_i^{-1/2}(i), \quad \text{and} \quad \bar{P}_i \triangleq \begin{bmatrix}
P_i & 0 \\
0 & 0
\end{bmatrix}.
\]
Now implement a \((1 \oplus J)\)-unitary transformation matrix \(\Theta_i\) (i.e., \(\Theta_i\) satisfies \(\Theta_i(1 \oplus J)\Theta_i^* = (1 \oplus J)\)) that transforms the following pre-array to the form
\[
\begin{bmatrix}
\gamma^{-1/2}(i) & \frac{1}{\sqrt{\lambda}} \left[ u_{i+1} \ x(i+1) \right] L_i \\
\Phi^* \hat{k}_i & \frac{1}{\sqrt{\lambda}} L_i
\end{bmatrix}
\Theta_i =
\begin{bmatrix}
s & 0 \\
m & C
\end{bmatrix}, \tag{18}
\]
where \(s\) is a scalar, \(m\) is a column vector, and \(C\) is a matrix. Then we claim that we can make the identifications
\[
s = \gamma^{-1/2}(i + 1), \quad m = \hat{k}_{i+1}, \quad C = L_{i+1},
\]
and also show that \(\nabla_{\{P_{i+1}, \Phi\}} = JC^*\). [This last equation means that the inertia of \(\nabla_{\{P_{i+1}, \Phi\}}\) can also be taken as \(J\), and that the above array algorithm provides the desired low rank factor \(L_{i+1}\) as well.]

To determine the \(\{s, m, C\}\) as above, all we need to do is simply compare entries on both sides of the following equality (which holds in view of the \((1 \oplus J)\)-unitarity of \(\Theta_i\):
\[
\begin{bmatrix}
\gamma^{-1/2}(i) & \frac{1}{\sqrt{\lambda}} \left[ u_{i+1} \ x(i+1) \right] L_i \\
\Phi^* \hat{k}_i & \frac{1}{\sqrt{\lambda}} L_i
\end{bmatrix}
\begin{bmatrix}
1 \\
J
\end{bmatrix}
\begin{bmatrix}
\gamma^{-1/2}(i) & \frac{1}{\sqrt{\lambda}} \left[ u_{i+1} \ x(i+1) \right] L_i \\
\Phi^* \hat{k}_i & \frac{1}{\sqrt{\lambda}} L_i
\end{bmatrix}^*
= \begin{bmatrix}
s & 0 \\
m & C
\end{bmatrix}
\begin{bmatrix}
1 \\
J
\end{bmatrix}
\begin{bmatrix}
s & 0 \\
m & C
\end{bmatrix}^*. \tag{19}
\]
The resulting fast algorithm can be summarized as follows.

Algorithm 1 (Fast Array Laguerre Algorithm) Consider input regression vectors \(u_i\) arising from the Laguerre structure of Figure 1. The solution to the minimization problem (1) can be recursively computed as follows. Start with \(w_{-1} = 0, \gamma^{-1/2}(0) = 1, g_0 = 0, L_0\) and \(J\) from the factorization (17), and repeat for each \(i > 0\),
\[
\begin{bmatrix}
\gamma^{-1/2}(i) & \frac{1}{\sqrt{\lambda}} [u_{i+1} \ x(i+1)] L_i \\
\Phi^* \hat{k}_i & \frac{1}{\sqrt{\lambda}} L_i
\end{bmatrix}
\Theta_i =
\begin{bmatrix}
\gamma^{-1/2}(i+1) & 0 \\
g_{i+1} \gamma^{-1/2}(i+1) & L_{i+1}
\end{bmatrix},
\]
where \(\Theta_i\) is a \((1 \oplus J)\)-unitary matrix that produces the zero entries in the post-array, and \(L_i\) is \((M + 1) \times r\). The transformation matrix \(\Theta_i\) can be implemented according to the procedure described below. The upper triangular Toeplitz matrix \(T\) and the row vector \(v\) are as defined in Eqs. (11) and (12). Moreover,
\[
w_{i+1} = w_i + g_{i+1}[d(i+1) - u_{i+1} w_i].
\]

4 Implementation Issues

Although from a theoretical point of view, any \((1 \oplus J)\)-unitary matrix \(\Theta_i\) that produces the zero entries in the first row of the post-array will do, we have noticed that different implementations lead to different numerical behavior. To see this, consider for simplicity the case \(M = 3\) and \(J = (1 \oplus -1)\). Then the pre- and post-arrays will be of the generic forms:
\[
\begin{bmatrix}
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times
\end{bmatrix} \quad \Theta = \begin{bmatrix}
\times & 0 & 0 \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
0 & \times & \times
\end{bmatrix}.
\]
In order to create the zero pattern in the first row of the post array, the \((1 \oplus J)\)-unitary rotation \(\Theta\) can be evaluated based only on the first row of the pre-array, which means that only the information needed to update \(\gamma^{-1}(i)\) to \(\gamma^{-1}(i + 1)\) is required in order to generate the matrix \(\Theta\). In other words, in this case no information from the other equations is used to update the rest of the entries of the array. We have observed in simulations (see Sec. 6) that even for \(\lambda = 1\), this type of transformation can cause the algorithm to diverge in finite precision.

To mitigate this problem, we propose to apply \(\Theta\) as follows. First create one zero in the first row of the post-array by means of a circular rotation, using the entries \((0, 0)\) and \((0, 1)\) (as indicated by the arrows) in the pre-array:

\[
\begin{bmatrix}
\otimes & \otimes & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\end{bmatrix}
\quad \xrightarrow{\text{Givens}}
\begin{bmatrix}
\times & 0 & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\end{bmatrix}.
\]

Now, note that the additional hyperbolic rotation that is needed to zero out the remaining entry in the first row of the post-array, also needs to create a zero in the entry \((6, 5)\) of the post-array. Rather than determining this hyperbolic rotation by using the entries \((0, 0)\) and \((0, 3)\) of the first row of the pre-array, we propose instead to determine the hyperbolic rotation by using the entries \((6, 5)\) and \((6, 6)\) of the last row of the pre-array. That is,

\[
\begin{bmatrix}
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\otimes & \otimes & \otimes \\
\end{bmatrix}
\quad \xrightarrow{\text{Hyperbolic}}
\begin{bmatrix}
\times & 0 & 0 \\
\times & \times & \times \\
\times & \times & \times \\
0 & \times & \times \\
\end{bmatrix}.
\]

This choice is actually more reasonable, since in this case, the rotation matrix \(\Theta\) is determined by using all the equations that constitute the fast recursions. We have verified by simulations that this method of constructing \(\Theta\) presents no signs of instability for the case \(\lambda = 1\).

On another matter, the algorithm also requires the evaluation of a matrix-vector product of the form \(Tp\) for some vector \(p\). Now since the matrix \(T\) is upper triangular Toeplitz, this product amounts to a convolution operation. Actually, the entries of \(Tp\) can be obtained by filtering the (reversed) entries of \(p\) through the maximum-phase all-pass filter \(Q(z) = (z^{-1} + a^{-1})/(1 + a^{-1}z^{-1})\).

Another possibility for the evaluation of \(Tp\) is to rely on fast transform techniques. One well-known technique (known as the \textit{extension approach}) is to embed the Toeplitz matrix into a larger \(n \times n\) circulant matrix, which can then be diagonalized by the DFT matrix. A second technique, known as the \textit{decomposition approach}, is to express \(T\) as the sum of circulant and skew-circulant matrices. Both techniques can also be extended to trigonometric and Hartley transforms (see [17]), which are better suited for cases with real-valued data.

### 5 Choice of the Regularization Matrix

We now examine choices for the positive-definite matrix \(\Pi_0\) such that the rank of the initial difference \(\nabla_{\{R_0, \Phi\}}\) is low (2 or 3). For any such choice of \(\Pi_0\), all successive differences \(\nabla_{\{P, \Phi\}}\) will also have low rank with the same inertia. [The actual value of \(\Pi_0\) is not needed in the fast
algorithm, but rather the corresponding factors \( \{L_0, J\} \). This initial calculation is performed off-line and can be assigned to overhead costs.\]

Assume first that \( \lambda = 1 \). Then the choice \( \Pi_0 = \delta I \), for any \( \delta > 0 \), leads to a difference \( \nabla_{\{P_0, \Phi\}} \) of the form

\[
\nabla_{\{P_0, \Phi\}} = \delta \left( \begin{bmatrix} \Pi_0 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} T & 0 \\ v & 0 \end{bmatrix} \begin{bmatrix} \Pi_0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} T^* & v^* \\ 0 & 0 \end{bmatrix} \right) = \delta \begin{bmatrix} I - TT^* & -Tv^* \\ -vT^* & -\|v\|^2 \end{bmatrix}.
\]

This difference can be easily seen to have rank 2. This is because \( I - TT^* \) has rank one, viz.,

\[
I - TT^* = -\alpha^2 cc^* \text{ with } \alpha = \frac{\sqrt{1 - a^2}}{a^M}, \quad c = \text{col}\{1, -a, a^2, -a^3, \ldots, (-a)^{M-1}\},
\]

and the vector \( Tv^* \) is collinear with \( c \), since \( Tv^* = -\beta c \) with \( \beta = \alpha/a^M \). Therefore, \( \nabla_{\{P_0, \Phi\}} \) has rank 2, with

\[
J = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad L_0 = \sqrt{\delta} \cdot \begin{bmatrix} 0 & \frac{\alpha c}{a^M} \\ 1 & \frac{\alpha c}{a^M} \end{bmatrix}.
\]

Let us now consider the case \( \lambda \neq 1 \), for which the difference \( \nabla_{\{P_0, \Phi\}} \) becomes

\[
\nabla_{\{P_0, \Phi\}} = \begin{bmatrix} \lambda^{-1} \Pi_0 - T\Pi_0 T^* & -T\Pi_0 v^* \\ -v\Pi_0 T^* & -v\Pi_0 v^* \end{bmatrix} \quad (20).
\]

We want to argue that there are choices of \( \Pi_0 \) that lead to at least a rank 3 difference. This can be seen as follows. Assume first that \( \sqrt{\lambda}/a < 1 \). Then any vector \( b \) such that the pair \( (\lambda^{1/2}T, b) \) is controllable will result in a positive-definite solution \( \Pi_0 \) for the Lyapunov equation

\[
\lambda^{-1} \Pi_0 - T\Pi_0 T^* = bb^* \quad (21).
\]

If, on the other hand, \( \sqrt{\lambda}/a > 1 \), then any vector \( b \) such that the pair \( (\lambda^{1/2}T, b) \) is controllable, will result in a positive-definite solution \( \Pi_0 \) for the following Lyapunov equation

\[
\lambda^{-1} \Pi_0 - T\Pi_0 T^* = -bb^* \quad (22).
\]

In either case, the point is that we can choose a \( b \) (and consequently, \( \Pi_0 \)) such that the difference \( \lambda^{-1} \Pi_0 - T\Pi_0 T^* \) has rank one (and inertia 1 or \( -1 \) depending on whether \( \sqrt{\lambda}/a \) is smaller or larger than one). It then follows that the rank of \( \nabla_{\{P_0, \Phi\}} \) in (20) will be 3 and its inertia will be \( \{\mu, -1, 1\} \), where \( \mu = +1 \) or \( \mu = -1 \). [We may remark that once a vector \( b \) has been chosen, it can be scaled arbitrarily by a number \( \sqrt{\delta} \). This helps guarantee that the resulting \( \Pi_0 \) will be sufficiently large, as is often required in such least-squares applications.]

[Also, if one insists on a rank 2 matrix \( \nabla_{\{P_0, \Phi\}} \), then we need to find a \( b \) such that \( -T\Pi_0 v^* \) is collinear with \( b \) as well.]

6 Simulation Results

The simulation results in this section were obtained by averaging over 10 runs in a system identification scenario with 50 dB signal-to-noise ratio. The input to the unknown model was taken as colored noise.

Figure 2 shows the results of two experiments with \( \lambda = 1 \) and a Laguerre filter with 5 taps and poles at \( a = 0.5 \). The regularization matrix was chosen as \( \Pi_0 = 10^5 I \). The figure
Figure 2: Learning curves for the fast RLS Laguerre using two forms of hyperbolic transformations.

shows two learning curves corresponding to the two different implementations of the rotation matrices $\Theta_i$, as explained in Sec. 4. We can see from the figure that the classical method of implementation of the rotations leads to divergence in a few iterations. Figure 3 compares the performance of the fast Laguerre filter with the fast FIR filter (both implemented in array forms). The model used for the system identification example in Figure 3 is the same from [16]):

$$G(z) = 0.0017 \frac{z^{-1}(1 + 0.673z^{-1})}{(1 - 0.368z^{-1})(1 - 0.819z^{-1})(1 - 0.995z^{-1})}.$$  

The Laguerre filter was implemented with 6 taps and the FIR filter with 500 taps. Both filters used $\lambda = 0.999$. We see from the figure that the Laguerre-RLS algorithm presents faster convergence and achieves a lower MSE level compared to the fast FIR-RLS algorithm.

We should also mention that for $\lambda < 1$, the fast recursions may encounter convergence difficulties. For $\lambda = 1$, and by using the proposed method for implementing the rotations, we did not notice such problems even over long simulations. [Actually, even the fast array algorithm for regression vectors with shift structure is also unstable for $\lambda = 1$ if the rotations are not implemented with care.] Methods to stabilize the recursions in the general case are under investigation.

References


Figure 3: Comparison of a 6-tap fast Laguerre filter with a 500-tap fast FIR filter.


