The Fast Subsampled-Updating Stabilized Fast Transversal Filter (FSU SFTF) RLS Algorithm for Adaptive Filtering

Karim Maouche Dirk T.M. Slock

Institut EURECOM

2229, route des Crêtes, B.P. 193, 06904 Sophia Antipolis Cedex, FRANCE

Tel: +33 4 93 00 26 32 (Maouche)/ 06 (Slock)/ 27 (fax)

email: {maouche,slock}@eurecom.fr

Abstract

We present a new, doubly fast algorithm for Recursive Least-Squares (RLS) adaptive filtering that uses displacement structure and subsampled-updating. The Fast Subsampled-Updating Stabilized Fast Transversal Filter (FSU SFTF) algorithm is mathematically equivalent to the classical Fast Transversal Filter (FTF) algorithm. The FTF algorithm exploits the shift invariance that is present in the RLS adaptation of a FIR filter. The FTF algorithm is in essence the application of a rotation matrix to a set of filters and in that respect resembles the Levinson algorithm. In the subsampled-updating approach, we accumulate the rotation matrices over some time interval before applying them to the filters. It turns out that the successive rotation matrices themselves can be obtained from a Schur type algorithm which, once properly initialized, does not require inner products. The various convolutions that appear in the algorithm are done using the Fast Fourier Transform (FFT). The resulting algorithm is doubly fast since it exploits FTF and FFTs. The roundoff error propagation in the FSU SFTF algorithm is identical to that in the SFTF algorithm, a numerically stabilized version of the classical FTF algorithm. The roundoff error generation on the other hand seems somewhat smaller. For relatively long filters, the computational complexity of the new algorithm is smaller than that of the well-known LMS algorithm, rendering it especially suitable for applications such as acoustic echo cancellation.

EDICS: SP 2.2.5

1 Introduction

Nowadays, adaptive filtering is an important tool in digital signal processing. Recent advances in VLSI technology have rendered this tool very attractive and have led to diverse applications such as equalization, echo cancellation, interference cancellation, signal detection, etc. [1]. In an adaptive filter, the coefficients are periodically updated according to an adaptive filtering algorithm in order to minimize a certain cost function. There exist two major families of adaptive algorithms. The first family is built around the Least-Mean-Square (LMS) algorithm [2],[3]. The LMS algorithm minimizes mean square filtering error by using a gradient search type algorithm and is very popular because of its low computational complexity which is $2N$ (N is the FIR filter length) and its robustness. However, the convergence rate of the LMS depends on the length of the filter and on the input statistics. In applications such as acoustic echo cancellation where the FIR filter which models the acoustic path is relatively large and the input signal is highly correlated (speech signal), the LMS algorithm does not provide a satisfactory solution because of the very low convergence rate of the filter estimate. The second family is based upon the Recursive Least-Squares (RLS) algorithm that minimizes a deterministic sum of squared errors. The RLS algorithm is known to be capable of performing much better than the LMS algorithm [4] but suffers from a computational complexity of $\mathcal{O}(N^2)$ operations. This complexity restricts its use in applications where the FIR filter is relatively long. Fast RLS algorithms such as the Fast Transversal Filter (FTF) algorithm [5],[6] and the Fast Lattice/Fast QR (FLA/FQR) algorithms [7] efficiently exploit the shift invariance structure present in the RLS approach to the adaptive FIR filtering problem. They reduce the computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ operations per sample. In order to further reduce the computational complexity of these algorithms, it appears that the sampling rate at which the LS filter estimate is provided has to be reduced from the signal sampling rate to a subsampled rate with a subsampling factor of $L \geq 1$. Two strategies emerge in order to accomplish this. One consists of a block processing approach in which the normal equations governing the LS problem are solved every L samples. This leads to Block RLS (BRLS) algorithms [8],[9]. An alternative approach (especially applicable when $L < N$) consists of

using the same strategy as the RLS algorithm and to compute the new filter estimate and auxiliary quantities from the same quantities that were available L samples before. In [10], we have applied this strategy and derived the Subsampled-Updating RLS (SU RLS) algorithm, which nevertheless provides exactly the same filtering error signal as the RLS algorithm. The computational complexity of the SU RLS algorithm is certainly not reduced with respect to that of the RLS algorithm. However, in the SU RLS algorithm the Kalman gain and the likelihood variable are $L \times N$ and $L \times L$ matrices respectively which, due to the shift invariance present in the problem, exhibit a low displacement rank. Hence, by using the displacement structure [11] and the FFT (when computing convolutions), we have derived a fast version of SU RLS that we have called the FSU RLS algorithm.

In [12], we have proposed to handle the problem of reducing the computational complexity of the RLS algorithm by employing a dual strategy. This allowed us to derive the FSU FTF algorithm, see Fig. 1. Namely, after having exploited shift-invariance in the RLS algorithm to obtain the FTF algorithm, we apply the Subsampled-Updating Strategy (SUS) to the estimation of the filters involved. The starting point is an interpretation of the FTF algorithm as a rotation applied to the vectors of filter coefficients. Using the filter estimates at a certain time instant, we compute the filter outputs over the next L time instants. Using what we have called a Schur-FTF procedure, it becomes possible to compute from these multi-step ahead predicted filter outputs the one step ahead predicted filter outputs, without updating or using the filters. These quantities allow us to compute the successive rotation matrices of the FTF algorithm for the next L time instants. Because of the presence of a shift operation in the FTF algorithm, it turns out to be most convenient to work with the z-transform of the rotation matrices and the filters. One rotation matrix is then a polynomial matrix of order one, and the product of L successive rotation matrices is a polynomial matrix of order L. Applying the L rotation matrices to the filter vectors becomes an issue of multiplying polynomials (convolution), which can be efficiently carried out using the FFT. Unfortunately, the FTF algorithm is numerically unstable because of round-off error accumulation that arises with finite precision implementation. Inheriting the round-off errors dynamics of the FTF algorithm, the FSU FTF algorithm is also numerically unstable. The Stabilized FTF (SFTF)

algorithm, a numerically stabilized version of the FTF algorithm, has been introduced to alleviate this problem, at the cost of a marginal increase of the computational complexity from $7N$ to $8N$ [13]. Here, we extend the FSU FTF idea to the Stabilized FTF (SFTF) algorithm. The starting point is still an interpretation of the SFTF algorithm as a rotation applied to the vectors of filter coefficients. The key ingredient is the computation of the rotation parameters in a way that mimicks exactly the operations performed bu the SFTF algorithm. The resulting FSU SFTF algorithm turns out to be especially applicable in the case of very long filters such as those that are used in the acoustic echo cancellation problem. The gain it offers in computational complexity is obtained in exchange for some processing delay, as is typical of block processing.

In order to formulate the RLS adaptive filtering problem and to fix notation, we shall first recall the RLS algorithm in section 2. In section 3, we briefly present the (multi-channel) SFTF algorithm and introduce its rotation formulation. The Schur-SFTF procedure that allows the computation of L filtering errors without updating the filter estimate at each input data sample is presented in section 4. In order to update the filters by convolution, the FFT based Overlap-Save technique is presented in detail in section 5. In section 6, we show how the z-transform yields an easy formulation of the FSU SFTF algorithm whose computational complexity is discussed in section 7. Finally, concluding remarks are given in section 8.

2 The RLS Algorithm

An adaptive transversal filter $W_{N,k}$ forms a linear combination of N consecutive input samples $\{x(i-n), n = 0, \ldots, N-1\}$ to approximate (the negative of) the desired-response signal $d(i)$. The resulting error signal is given by (see Fig. 2)

$$\epsilon_N(i|k) \;=\; d(i) + W_{N,k}\, X_N(i) \;=\; d(i) + \sum_{n=0}^{N-1} W_{N,k}^{n+1}\, x(i-n) \;\;, \tag{1}$$

where $X_N(i) \;=\; \left[x^H(i)\, x^H(i-1) \cdots x^H(i-N+1)\right]^H$ is the input data vector and superscript ^{H} denotes Hermitian (complex conjugate) transpose. In the RLS algorithm, the set of N transversal filter coefficients $W_{N,k} = \left[W_{N,k}^1 \cdots W_{N,k}^N\right]$ are adapted so as to minimize recursively

the following LS criterion

$$\begin{aligned}\xi_N(k) &= \min_{W_N}\left\{\sum_{i=1}^{k}\lambda^{k-i}\left\|d(i)+W_N\,X_N(i)\right\|^2 \;+\; \lambda^{k+1}\mu\left\|W_N-W_0\right\|^2_{\Lambda_N}\right\} \\ &= \sum_{i=1}^{k}\lambda^{k-i}\left\|\epsilon_N(i|k)\right\|^2 \;+\; \lambda^{k+1}\mu\left\|W_{N,k}-W_0\right\|^2_{\Lambda_N} \quad ,\end{aligned} \tag{2}$$

where $\lambda \in (0,1]$ is the exponential weighting factor, $\mu > 0$, $\Lambda_N = \text{ diag}\left\{\lambda^{N-1},\ldots,\lambda,1\right\}$, $\|v\|^2_\Lambda = v\Lambda v^H$, $\|.\| = \|.\|_I$. The second term in the LS criterion represents a priori information. For instance, prior to measuring the signals, we may assume that W_N is distributed as $W_N \sim \mathcal{N}\left(W_0, R_0^{-1}\right)$, $R_0 = \mu\lambda\Lambda_N$. The particular choice for R_0 will become clear in the discussion of the initialization of the FSU SFTF algorithm.

Minimization of the LS criterion leads to the following minimizer

$$W_{N,k} \;=\; -P^H_{N,k}R^{-1}_{N,k} \quad , \tag{3}$$

where

$$\begin{aligned}R_{N,k} &= \sum_{i=1}^{k}\lambda^{k-i}X_N(i)X_N^H(i) \;+\lambda^{k+1}\mu\Lambda_N \\ &= \lambda R_{N,k-1} + X_N(k)X_N^H(k)\;, \qquad\qquad R_{N,0} = R_0 = \mu\lambda\Lambda_N \\ P_{N,k} &= \sum_{i=1}^{k}\lambda^{k-i}X_N(i)d^H(i) \;-\lambda^{k+1}\mu\Lambda_N W_0^H \\ &= \lambda P_{N,k-1} + X_N(k)d^H(k)\;, \qquad\qquad P_{N,0} = -R_0W_0^H \quad ,\end{aligned} \tag{4}$$

are the sample second order statistics. Substituting the time recursions for $R_{N,k}$ and $P_{N,k}$ from (4) into (3) and using the matrix inversion lemma [14, pg 656] for $R^{-1}_{N,k}$, we obtain the RLS algorithm:

$$\widetilde{C}_{N,k} \;=\; -X_N^H(k)\lambda^{-1}R^{-1}_{N,k-1} \tag{5}$$

$$\gamma_N^{-1}(k) \;=\; 1-\widetilde{C}_{N,k}X_N(k) \tag{6}$$

$$R^{-1}_{N,k} \;=\; \lambda^{-1}R^{-1}_{N,k-1} \;-\; \widetilde{C}^H_{N,k}\gamma_N(k)\widetilde{C}_{N,k} \tag{7}$$

$$\epsilon_N^p(k) \;=\; \epsilon_N(k|k-1) \;=\; d(k)+W_{N,k-1}X_N(k) \tag{8}$$

$$\epsilon_N(k) \;=\; \epsilon_N(k|k) \;=\; \epsilon_N^p(k)\,\gamma_N(k) \tag{9}$$

$$W_{N,k} \;=\; W_{N,k-1}+\epsilon_N(k)\widetilde{C}_{N,k} \quad , \tag{10}$$

where $\epsilon_N^p(k)$ and $\epsilon_N(k)$ are the a priori and a posteriori error signals (resp. predicted and filtered errors in the Kalman filtering terminology) and one can verify (or see [5]) that they

are related by the likelihood variable $\gamma_N(k)$ as in (9).

3 The SFTF Algorithm

The FTF algorithm efficiently exploits the shift-invariance present in the adaptive FIR filtering problem, which translates into a low displacement rank of $R_{N,k}^{-1}$. The computational complexity of the FTF algorithm is $7N$ in its most efficient form. In this algorithm, $A_{N,k}$ and $B_{N,k}$ are the forward and backward prediction filters, $e_N^p(k)$ and $e_N(k)$ are the a priori and a posteriori forward prediction errors, $r_N^p(k)$ and $r_N(k)$ are the a priori and a posteriori backward predition errors, $\widetilde{C}_{N+1,k} = \left[\begin{array}{ccc} \widetilde{C}_{N+1,k}^0 & \cdots & \widetilde{C}_{N+1,k}^N \end{array} \right]$ is the Kalman gains of (increased) order N+1, and $\alpha_N(k)$ and $\beta_N(k)$ are the forward and backward prediction error variances. The key ingredient of the stabilization of the FTF algorithm is the introduction of redundancy in the algorithm by computing the backward prediction error in two ways: $r_N^{pf}(k)$ and $r_N^{ps}(k)$. Hence, the difference between the two computed values constitutes a measurement of the numerical errors in the implemented algorithm. This numerical error is fed back to the algorithm in order to stabilize the numerical error propagation system associated with the algorithm [15]. This feedback operation can be done by simply taking as final value of $r_N^p(k)$ a certain convex combination of the two computed values as was suggested independently in [16] and [13]. The convex combination coefficients can be interpreted as feedback coefficients. The signal $r_N^p(k)$ appears essentially in two places in the algorithm, for which the two values $K_1 = 1.5$ and $K_2 = 2.5$ of the feedback gains appear to stabilize well for most applications. It was also shown in [13] that the likelihood variable recursion of the FTF algorithm is also numerically unstable, but that this problem can be circumvented by eliminating $\gamma_N(k)$ in terms of $\alpha_N(k)$ and $\beta_N(k)$.

In Table 1, the SFTF algorithm is presented for the general complex multichannel case. However, the remainder of this paper will concentrate on the single-channel case for notational simplicity, extensions to the multichannel case are immediate though. The algorithm can be

described in the following way, which emphasizes its rotational structure:

$$\begin{bmatrix} \begin{bmatrix} \widetilde{C}_{N,k} & 0 \end{bmatrix} \\ A_{N,k} \\ B_{N,k} \\ \begin{bmatrix} W_{N,k} & 0 \end{bmatrix} \end{bmatrix} = \Theta_k \begin{bmatrix} \begin{bmatrix} 0 & \widetilde{C}_{N,k-1} \end{bmatrix} \\ A_{N,k-1} \\ B_{N,k-1} \\ \begin{bmatrix} W_{N,k-1} & 0 \end{bmatrix} \end{bmatrix} , \tag{11}$$

with Θ_k a 4×4 rotation matrix given by

$$\Theta_k = \Theta_k^1 \; \Theta_k^2 \; \Theta_k^3 \; \Theta_k^4 \; , \tag{12}$$

where the four 4×4 matrices $\Theta_k^i \; , \; i = 1, 2, 3, 4$ are

$$\Theta_k^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \epsilon_N(k) & 0 & 0 & 1 \end{bmatrix} \quad , \quad \Theta_k^2 = \begin{bmatrix} 1 & 0 & -\widetilde{C}_{N+1,k}^N & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Theta_k^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ r_N^{(1)}(k) & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad , \quad \Theta_k^4 = \begin{bmatrix} 1 & \dfrac{-e_N^{pH}(k)}{\lambda \alpha_N(k-1)} & 0 & 0 \\ e_N(k) & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{13}$$

In fact, one can straightforwardly see that the different steps in the SFTF algorithms consists of the following vector transformations:

$$\Theta_k \begin{bmatrix} \begin{bmatrix} 0 & \widetilde{C}_{N,k-1} \end{bmatrix} \\ A_{N,k-1} \\ B_{N,k-1} \\ \begin{bmatrix} W_{N,k-1} & 0 \end{bmatrix} \end{bmatrix} = \Theta_k^1 \Theta_k^2 \begin{bmatrix} \begin{bmatrix} \widetilde{C}_{N,k} & 0 \end{bmatrix} \\ A_{N,k} \\ B_{N,k-1} \\ \begin{bmatrix} W_{N,k-1} & 0 \end{bmatrix} \end{bmatrix} = \Theta_k^1 \begin{bmatrix} \begin{bmatrix} \widetilde{C}_{N,k} \, 0 \end{bmatrix} \\ A_{N,k} \\ B_{N,k} \\ \begin{bmatrix} W_{N,k-1} & 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \widetilde{C}_{N,k} & 0 \end{bmatrix} \\ A_{N,k} \\ B_{N,k} \\ \begin{bmatrix} W_{N,k} & 0 \end{bmatrix} \end{bmatrix} . \tag{14}$$

Apart from the computation of the filters, the prediction error variances $\alpha_N(k)$ and $\beta_N(k)$ also need to be updated. In order to compute the rotation matrices, one must obtain the a priori errors $e_N^p(k) \, , r_N^{pf}(k)$ and $\epsilon_N^p(k)$ which are the outputs at time k of the filters $A_{N,k-1}, B_{N,k-1}$ and $W_{N,k-1}$. In the FTF algorithm, these quantities are computed via inner products.

4 The Schur-SFTF Procedure

Now we apply the SUS to the SFTF algorithm. From the filters at time instant $k-L$, we want to obtain the filters at time instant k. This will require the rotation matrices and hence the a priori errors in that time range. We shall show that these quantities can be computed without generating (completely) the intermediate filter estimates using a Schur-SFTF algorithm. Let us introduce the negative of the filter output

$$\widehat{d}_N^{\,p}(k) = d(k) - \epsilon_N^p(k) \quad , \quad \widehat{d}_N(k) = d(k) - \epsilon_N(k) \quad . \tag{15}$$

Consider now the following set of filtering operations

$$F_L(k) \triangleq \begin{bmatrix} \eta_{N,L,k}^H \\ e_{N,L,k}^{p\,H} \\ r_{N,L,k}^{pf\,H} \\ -\widehat{d}_{N,L,k}^{\,p\,H} \end{bmatrix} \triangleq \begin{bmatrix} [\,0 \;\; \widetilde{C}_{N,k-L}\,] \\ A_{N,k-L} \\ B_{N,k-L} \\ [\,W_{N,k-L} \;\; 0\,] \end{bmatrix} X_{N+1,L,k}^H \tag{16}$$

where

$$X_{N+1,L,k} = \begin{bmatrix} X_{N+1}^H(k-L+1) \\ \vdots \\ X_{N+1}^H(k) \end{bmatrix} = [\, x_{L,k} \; x_{L,k-1} \cdots x_{L,k-N} \,] \quad , \tag{17}$$

is the $L \times (N+1)$ Toeplitz input data matrix and $x_{L,k} = [\, x_{k-L+1} \cdots x_k \,]^H$. $F_L(k)$ is a $4 \times L$ matrix, the rows of which are the result of the filtering of the data sequence $\{x(j) \;,\; j = k-N-L+1, \ldots, k\}$ by the four filters of the SFTF algorithm, see Fig. 3. $\eta_{N,L,k}$ is the output of the Kalman gain and $e_{N,L,k}^p$ and $r_{N,L,k}^{pf}$ are respectively the vectors of forward and backward (multistep ahead) prediction errors

$$e_{N,L,k}^p = \begin{bmatrix} e_N^H(k-L+1|k-L) \\ \vdots \\ e_N^H(k|k-L) \end{bmatrix} \quad , \quad r_{N,L,k}^{pf} = \begin{bmatrix} r_N^{f\,H}(k-L+1|k-L) \\ \vdots \\ r_N^{f\,H}(k|k-L) \end{bmatrix} \quad . \tag{18}$$

The last row of $F_L(k)$ corresponds to the (multi-step ahead predicted) adaptive filter outputs

$$\widehat{d}^{\,p}_{N,L,k} = d_{L,k} - \epsilon^{p}_{N,L,k} = \begin{bmatrix} d^H(k-L+1) \\ \vdots \\ d^H(k) \end{bmatrix} - \begin{bmatrix} \epsilon^H_N(k-L+1|k-L) \\ \vdots \\ \epsilon^H_N(k|k-L) \end{bmatrix} = \begin{bmatrix} \widehat{d}^{\,H}_N(k-L+1|k-L) \\ \vdots \\ \widehat{d}^{\,H}_N(k|k-L) \end{bmatrix} . \tag{19}$$

The first column of $F_L\,(k)$ turns out to be

$$F_L\,(k)\;\; u_{L,1} \;\; = \;\; \begin{bmatrix} 1 - \gamma_N^{-1}(k-L) \\ e^p_N\,(k-L+1) \\ r^{pf}_N\,(k-L+1) \\ -\widehat{d}^{\,p}_N\,(k-L+1) \end{bmatrix} , \tag{20}$$

where $u_{L,n}$ is the $L \times 1$ vector with 1 in the n^{th} position and 0 elsewhere. The quantities in (20) give after some straightforward operations (using the appropriate recursions of the SFTF algorithm in Table 1) the scalars $\epsilon_N(k-L+1)\,, e_N(k-L+1)$ and $e^p_N(k-L+1)/\lambda\alpha_N(k-L)$, that are elements of the rotation matrix Θ_{k-L+1}. In order to obtain the rest of the elements that define Θ_{k-L+1} , we must compute $r^{ps}_N(k-L+1)$ (this allows the computation of $r^{(1)}_N(k-L+1)$ that appears in the matrix Θ^3_{k-L+1}) and $\widetilde{C}^N_{N+1,k-L+1}$ (in Θ^2_{k-L+1}). Since $r^{ps}_N(k-L+1) = -\lambda\beta_N(k-L)\widetilde{C}^N_{N+1,k-L+1}$ (see Table 1), we just need to compute $\widetilde{C}^N_{N+1,k-L+1}$ in order to obtain the rotation matrix Θ_{k-L+1}. In the SUS, our aim is to compute the successive rotation matrices over an interval of L samples. To do this, we need the different $\widetilde{C}^N_{N+1,k-L+j}$ for $j = 1, \ldots, L$. In fact, it turns out that these quantities can be obtained in an efficient manner by carrying out the SFTF recursions on the last $L-j$ entries of the filters in the SFTF prediction part:

For $j = 1, \ldots, L$:

$$\begin{aligned} n_j \;\; &= \;\; N - L + j \;\; , \;\; k_j \;\; = \;\; k - L + j \\ \widetilde{C}^{n_j:N}_{N+1,k_j} \;\; &= \;\; \widetilde{C}^{n_j-1:N-1}_{N,k_j-1} - \lambda^{-1}\alpha^{-1}(k_j-1)\, e^{p\,H}_N(k_j)\, A^{n_j:N}_{N,k_j-1} \\ r^{ps}_N(k_j) \;\; &= \;\; -\lambda\beta_N(k_j-1)\, \widetilde{C}^{N\,H}_{N+1,k_j} \end{aligned}$$

If $j < L$:

$$A^{n_j+1:N}_{N,k_j} \;\; = \;\; A^{n_j+1:N}_{N,k_j-1} + e_N(k_j)\, \widetilde{C}^{n_j:N-1}_{N,k_j-1}$$

$$\begin{aligned}
\left[\ \widetilde{C}_{N,k_j}^{n_j:N-1}\ \ 0\ \right] &= \widetilde{C}_{N+1,k_j}^{n_j:N} - \widetilde{C}_{N+1,k_j}^{N}\ B_{N,k_j-1}^{n_j:N} \\
B_{N,k_j}^{n_j+1:N} &= B_{N,k_j-1}^{n_j+1:N} + r_N^{(1)}(k_j)\left[\ \widetilde{C}_{N,k_j}^{n_j+1:N-1}\ \ 0\ \right]
\end{aligned}$$

End if (21)

End for .

Counting only the most significant term as we often do, the computational complexity of these recursions is $2L^2$. So with the quantities in $F_L\,(k)\ \ u_{L,1}$, some recursions from Table 1 and the recursions (21), it is possible to construct the successive $\Theta_{k-L+j}\ ,\ j = 1, \ldots, L$.

Now we rotate both expressions for $F_L(k)$ in (16) with Θ_{k-L+1} to obtain $\Theta_{k-L+1}F_L(k)$ which equals

$$\begin{bmatrix} \left[\ \widetilde{C}_{N,k-L+1}\ 0\ \right] \\ A_{N,k-L+1} \\ B_{N,k-L+1} \\ \left[\ W_{N,k-L+1}\ 0\ \right] \end{bmatrix} X_{N+1,L,k}^H = \begin{bmatrix} \boxed{\eta_{N,L-1,k}^H} & * \\ e_N(k-L+1) & \boxed{e_{N,L-1,k}^{p\,H}} \\ r_N^f(k-L+1) & \boxed{r_{N,L-1,k}^{pf\ H}} \\ --\ \widehat{d}_N\,(k-L+1) & \boxed{-\widehat{d}_{N,L-1,k}^{p\,H}} \end{bmatrix} . \quad (22)$$

One can see from (22) that quantities in boxes are the four rows of $F_{L-1}(k)$. This can be written more compactly as

$$\mathcal{S}\,(\Theta_{k-L+1}\,F_L(k)) \ = \ F_{L-1}(k)\ , \quad (23)$$

where the operator $\mathcal{S}(M)$ stands for: shift the first row of the matrix M one position to the right and drop the first column of the matrix thus obtained. Now this process can be repeated until we get $F_0(k)$ which is a matrix with no dimensions. So the same rotations that apply to the filters at times $k-L+j,\ j = 1, \ldots, L$, also apply to the set of filtering error vectors $F_{L-j}(k)$ over the same time span. With this procedure, the one step ahead output errors (rotation parameters) are computed during this time span without updating the filters. Inner products (filtering operations) are just needed for the computation of $F_L(k)$ and constitutes the initialization part of the procedure. The Schur-SFTF procedure is given in Table 2. This

procedure contrasts with the Levinson-style [17] SFTF algorithm in (11). Taking into account the fact that a rotation matrix in factored form as in (12) only contains five non-trivial entries, this takes $2.5L^2$ operations per L samples. The innner products need $4N$ operations, so the successive rotation matrices can be obtained via the Schur-SFTF procedure with a computational complexity of $4.5L^2 + 4N$ operations per L samples. The amount of operations needed for the inner products can be further reduced by using the FFT as is explained in the next section.

5 Fast computation using the FFT

It is possible to reduce the computational complexity of the Schur-SFTF procedure by introducing FFT techniques as explained in [18]. In what follows, we shall often assume for simplicity that L is a power of two and that $N_L = (N+1)/L$ is an integer. To get $F_L(k)$ in (16), we need to compute products of the form $v_{N+1,k}\ X^H_{N+1,L,k}$ where $v_{N+1,k}$ is a row vector of $N+1$ elements.

Consider a partitioning of $v_{N+1,k}$ in N_L subvectors of length L:

$$v_{N+1,k} = \left[\ v^1_{L,k}\ \cdots\ v^{N_L}_{L,k}\ \right]\ , \tag{24}$$

and a partitioning of $X_{N+1,L,k}$ in N_L submatrices of order $(L \times L)$:

$$X_{N+1,L,k} = [\ X_{L,L,k}\ X_{L,L,k-L}\ \cdots X_{L,L,k-N+L-1}\ ]\ , \tag{25}$$

then

$$v_{N+1,k}\ X^H_{N+1,L,k} = \sum_{j=1}^{N_L} v^j_{L,k} X^H_{L,L,k-(j-1)L}\ . \tag{26}$$

In other words, we have essentially N_L times the product of a vector of length L with a $(L \times L)$ Toeplitz matrix. Such a product can be efficiently computed in basically two different ways [18]. One way is to use fast convolution algorithms , which are interesting for moderate values of L. Another way is to use the overlap-save method in which one embeds the $L \times L$ Toeplitz

matrix $X_{L,L,k}$ into a $2L \times 2L$ circulant matrix, viz.

$$\underline{X}^H_{L,L,k} \;=\; \left[\begin{array}{cc} * & X^H_{L,L,k} \\ & \\ X^H_{L,L,k} & * \end{array}\right] \;=\; \mathcal{C}\left(x^H_{2L,k}\right) \tag{27}$$

where $\mathcal{C}(c^H)$ is a right shift circulant matrix with c^H as first row. Then we get for the vector-matrix product

$$v^j_{L,k} X^H_{L,L,k-(j-1)L} = \left[\begin{array}{cc} 0_{1\times L} & v^j_{L,k} \end{array}\right] \mathcal{C}\left(x^H_{2L,k-(j-1)L}\right) \left[\begin{array}{c} I_L \\ \\ 0_{L\times L} \end{array}\right] \quad . \tag{28}$$

Now consider the Discrete Fourier Transform (DFT) $\mathcal{V}^j_{L,k}$ of $v^j_{L,k}$

$$\mathcal{V}^j_{L,k} = v^j_{L,k}\; F_L \quad , \tag{29}$$

F_L is the $L \times L$ DFT matrix whose generic element is $(F_L)_{p,q} = e^{-i2\pi\frac{(p-1)(q-1)}{L}}$, $i^2 = -1$. The inverse of F_L is $\frac{1}{L}F^H_L$. It defines the inverse DFT transformation (IDFT)

$$v^j_{L,k} = \mathcal{V}^j_{L,k}\; \frac{1}{L} F^H_L \quad . \tag{30}$$

The product of a row vector v with a circulant matrix $\mathcal{C}(c^H)$ where v and c are of length m can be computed efficiently as follows. Using the property that a circulant matrix can be diagonalized *via* a similarity transformation with a DFT matrix, we get

$$v\;\mathcal{C}(c^H) \;=\; v\;F_m\;\mathrm{diag}\left(c^H F_m\right)\frac{1}{m}F^H_m \;=\; \left[\left(v F_m\right)\;\;\mathrm{diag}\left(c^H F_m\right)\right]\;\frac{1}{m}F^H_m \quad , \tag{31}$$

where $\mathrm{diag}(w)$ is a diagonal matrix with the elements of the vector w as diagonal elements. So the computation of the vector in (28) requires the padding of v with L zeros, the DFT of the resulting vector, the DFT of $x_{2L,k-(j-1)L}$, the product of the two DFTs, and the (scaled) IDFT of this product. When the FFT is used to perform the DFTs, this leads to a computationally more efficient procedure than the straightforward matrix-vector product which would require L^2 multiplications. Note that at time k, only the FFT of $x_{2L,k}$ needs to be computed; the FFTs of $x_{2L,k-jL}, j = 1, \ldots, M-1$ have been computed at previous time instants. This reduces

the $4N$ computations per sample that are needed for the initialization of the Schur-SFTF procedure to

$$4N\left[\frac{\mathrm{FFT}(2L)}{L^2}+\frac{2}{L}\right]+\frac{5\mathrm{FFT}(2L)}{L} \tag{32}$$

computations per sample (FFT(L) signifies the computational complexity associated with a FFT of length L) or basically $\mathcal{O}\left(N\,\frac{\log_2(L)}{L}\right)$ operations.

6 The FSU SFTF Algorithm

Once we have computed the L consecutive rotation matrices with the Schur-SFTF algorithm, we want to apply them all at once to obtain the filters at time k from the filters at time $k-L$. Due to the shift of the Kalman gain in (11), we need to work in the z-transform domain. So we shall associate polynomials with the filter coefficients as follows

$$\begin{bmatrix} \widetilde{C}_k(z) \\ A_k(z) \\ B_k(z) \\ W_k(z) \end{bmatrix} = \begin{bmatrix} \left[\,\widetilde{C}_{N,k}\;0\,\right] \\ A_{N,k} \\ B_{N,k} \\ \left[\,W_{N,k}\;0\,\right] \end{bmatrix} \begin{bmatrix} 1 \\ z^{-1} \\ \vdots \\ z^{-N} \end{bmatrix}. \tag{33}$$

Hence (11) can be written in the z-transform domain as

$$\begin{bmatrix} \widetilde{C}_k(z) \\ A_k(z) \\ B_k(k) \\ W_k(z) \end{bmatrix} = \Theta_k \begin{bmatrix} z^{-1} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \widetilde{C}_{k-1}(z) \\ A_{k-1}(z) \\ B_{k-1}(z) \\ W_{k-1}(z) \end{bmatrix}. \tag{34}$$

Let us introduce the following polynomial matrix

$$\Theta_k(z) \;=\; \Theta_k \begin{bmatrix} z^{-1} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}. \tag{35}$$

Now, in order to adapt the filters at time k from the ones at time $k-L$, we get straightforwardly

$$\begin{bmatrix} \widetilde{C}_k(z) \\ A_k(z) \\ B_k(k) \\ W_k(z) \end{bmatrix} = \Theta_{k,L}(z) \begin{bmatrix} \widetilde{C}_{k-L}(z) \\ A_{k-L}(z) \\ B_{k-L}(z) \\ W_{k-L}(z) \end{bmatrix} \tag{36}$$

where

$$\Theta_{k,L}(z) = \Theta_k(z)\,\Theta_{k-1}(z)\cdots\Theta_{k-L+1}(z) \quad . \tag{37}$$

Now also remark that $\Theta_{k,L}(z)$ has the following structure

$$\Theta_{k,L}(z) = \begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 1 \end{bmatrix} \tag{38}$$

where the stars stand for polynomials in z^{-1} of degree at most L. The accumulation of the successive rotation matrices is done as follows

For $j = 2, \ldots, L$

$$\begin{aligned} k_j &= k-L+j \\ \Theta_{k_j,j}(z) &= \Theta^1_{k_j}\,\Theta^2_{k_j}\,\Theta^3_{k_j}\,\Theta^4_{k_j} \begin{bmatrix} z^{-1} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \Theta_{k_j-1,j-1}(z) \quad . \end{aligned} \tag{39}$$

The computation of $\Theta_{k,L}(z)$ takes $7.5L^2$ operations. As a result of the structure displayed in (38), the product in (36) represents 12 convolutions of a polynomial of order L with a polynomial of order N. These convolutions can be done using fast convolution techniques. In the case we consider, in which the orders of the polynomials are relatively large, we will implement the convolutions using the FFT technique. Consider one of those convolution products: it has the form $\mathcal{P}_L \star \Phi_{N+1,k}$ where $\mathcal{P}_L$ is one of the 12 order L vectors that appear in the accumulated rotation matrix and $\Phi_{N+1,k}$ is one of the four SFTF filters. As in section 5, the product is splitted in N_L parts

$$\mathcal{P}_L \star \Phi_{N+1,k} = \mathcal{P}_L \star \left[\, \Phi^1_{L,k} \cdots \Phi^{N_L}_{L,k} \,\right] = \left[\, \mathcal{P}_L \star \Phi^1_{L,k} \cdots \mathcal{P}_L \star \Phi^{N_L}_{L,k} \,\right] \quad , \tag{40}$$

every subproduct in (40) is done using the Overlap-Save method. Note that at this stage, we do not need to compute the FFTs of the filters $A_{N,k}, B_{N,k}, \widetilde{C}_{N,k}$ and $W_{N,k}$ because they were already used when computing $F_L(k)$ in the Schur-SFTF procedure. The update of each filter need 3 times such product. Taking in particular the update of the adaptive filter, we have

$$W_k(z) = (\Theta_{k,L}(z))_{4,1}\,\widetilde{C}_{k-L}(z) + (\Theta_{k,L}(z))_{4,2}\,A_{k-L}(z) + (\Theta_{k,L}(z))_{4,3}\,B_{k-L}(z) + W_{k-L}(z) \quad , \quad (41)$$

each product in (41) is done as explained before. The additons are done in the frequency domain, reducing hence the number of needed IDFTs. The complexity associated with the update of the adaptive filter is $(\frac{N+1}{L}+3)FFT\,(2L)+6\,(N+1)$ operations per L samples. The resulting FSU SFTF algorithm is summarized in Table 3.

The initialization of the algorithm is done with the soft constraint initialization technique [5]. The additon of the soft constraint to the LS cost function as shown in (2) can be interpreted as the result of an unconstrained LS problem where the input signal is equal to $\sqrt{\mu}$ at time $k = -N$ and zero at all other time instants before time $k = 0$. Hence the FSU SFTF algorithm departs from the following initial conditions

$$\begin{array}{rcll} W_{N,0} & = & W_0 & \\ A_{N,0} & = & [\,1\;0\cdots 0\,]\;, & \alpha_N(0) = \lambda^N\mu \\ B_{N,0} & = & [\,0\cdots 0\;1\,]\;, & \beta_N(0) = \mu \\ \widetilde{C}_{N,0} & = & [\,0\cdots 0\,]\;, & \gamma_N(0) = 1\;\;. \end{array} \quad (42)$$

With this initialization at $k = 0$, the corresponding initial sample covariance matrix is indeed $R_0 = \mu\lambda\Lambda_N$ for some diagonal matrix Λ_N of powers of λ.

7 Computational Complexity

The complexity of the FSU SFTF algorithm is

$$C_{FSUSFTF} = (8\frac{N+1}{L}+17)\frac{FFT(2L)}{L} + 32\frac{N}{L} + 12L \quad (43)$$

operations per sample. This can be very interesting for long filters. For example, when $(N, L) = (4095, 256)$, $(8191, 256)$ and the FFT is done via the split radix ($FFT(2m) =$

$mlog_2(2m)$ real multiplications for real signals) the multiplicative complexity is respectively $1.2N$ and $0.8N$ per sample. This should be compared to $8N$ for the SFTF algorithm, the currently fastest stable RLS algorithm, and $2N$ for the LMS algorithm. The number of additions is somewhat higher. The cost we pay is a processing delay which is of the order of L samples. In fact, there exists an optimal value of L for every filter length N and the computational complexity per filter coefficient is decreasing as a function of N. In Table 4, we give the optimal multiplicative complexities for different values of N. Note that the computational complexity falls below that of the SFTF algorithm when N is greater than 127 and similarly below that of the NLMS algorithm when N becomes greater than 1023. This can be considered as a relatively small filter length when dealing with acoustic echos that appear in teleconference applications.

8 Concluding Remarks

We have simulated the algorithm to verify its correctness. In Fig. 4, we compare the evolution in dB of the numerical errors that can be measured in the backward prediction part: $10 \log < (r_N^{pf}(k) - r_N^{ps}(k))^2 >$, where the mean is taken over 100 samples. The input is white noise of unit variance, the filter length is $N = 100$, the initial backward prediction error energy is chosen to be $\mu = 0.01$, the forgetting factor is $\lambda = 1 - \frac{1}{3N} = 0.9967$ and the downsampling factor is $L = 32$. As we see, the FSU SFTF algorithm is numerically stable (simulations were run for more than 10^7 samples). Moreover, when comparing with the numerical errors of the SFTF algorithm, it appears that the FSU SFTF algorithm is more accurate.

In [10], we have introduced the FSU RLS algorithm, an alternative algorithm with a very different internal structure, but a very similar computational complexity

$$C_{FSURLS} = (8\frac{N+1}{L}+20)\frac{FFT(2L)}{L} + 35\frac{N}{L} + 5.5L \quad , \tag{44}$$

see Fig. 5. These developments lead us to conjecture that perhaps a lower bound on computational complexity has been reached for RLS algorithms when the subsampled updating strategy is applied and when the filters to be adapted are relatively long. We have also applied the SUS to other adaptive filters such as the FNTF algorithm [19],[20] and the FAP algorithm

[21],[22] which respectively leads to the FSU FNTF algorithm [23] and the FSU FAP algorithm [24]. As a perspective for further research, we may remark that since the FSU SFTF algorithm updates in blocks of L samples; numerical errors on the backward prediction filter can be observed in an L dimensional subspace, opening up the perspective of correcting the errors in an L dimensional subspace. This would increase the range of λ for stable operation by a factor L. At this point, these issues are subject of ongoing research.

References

[1] S. Haykin. *Adaptive Filter Theory.* Prentice-Hall, Englewood Cliffs, NJ, 1991. second edition.

[2] B. Widrow and S.D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[3] O. Macchi. *The Least Mean Squares Approach with Applications in Transmission.* John Wiley & Sons, New York, 1995.

[4] E. Eleftheriou and D. Falconer, "Tracking properties and steady state performance of RLS adaptive filter algorithms". *IEEE Trans. on ASSP*, ASSP-34(5):821–823, July 1987.

[5] J.M. Cioffi and T. Kailath. "Fast, recursive least squares transversal filters for adaptive filtering". *IEEE Trans. on ASSP*, ASSP-32(2):304–337, April 1984.

[6] D.T.M. Slock and T. Kailath. "A Modular Prewindowing Framework for Covariance FTF RLS Algorithms". *Signal Processing*, 28(1):47–61, July 1992.

[7] D.T.M. Slock. "Reconciling Fast RLS Lattice and QR Algorithms". In *Proc. ICASSP 90 Conf.*, pages 1591–1594, Albuquerque, NM, April 3–6 1990.

[8] X.-H. Yu and Z.-Y. He. "Efficient Block Implementation of Exact Sequential Least-Squares Problems". *IEEE Trans. Acoust., Speech and Signal Proc.*, ASSP-36:392–399, March 1988.

[9] J.M. Cioffi. "The Block-Processing FTF Adaptive Algorithm". *IEEE Trans. on ASSP*, ASSP-34(1):77–90, Feb. 1986.

[10] D.T.M. Slock and K. Maouche. "The Fast Subsampled-Updating Recursive Least-Squares (FSU RLS) for Adaptive Filtering Based on Displacement Structure and the FFT". *Signal Processing*, Vol. 40, No. 1, Oct. 1994, pp. 5–20.

[11] T. Kailath, S.Y. Kung, and M. Morf. "Displacement ranks of matrices and linear equations". *J. Math. Anal. Appl.*, 68(2):295–407, 1979. (See also *Bull. Amer. Math. Soc.*, vol. 1, pp. 769–773, 1979.).

[12] D.T.M. Slock and K. Maouche. "The fast subsampled-updating fast transversal filter (FSU FTF) RLS Algorithm" *Annals of telecommunications*, Vol. 49, No. 7-8, 1994, pp. 407-413.

[13] D.T.M. Slock and T. Kailath. "Numerically Stable Fast Transversal Filters for Recursive Least-Squares Adaptive Filtering". *IEEE Trans. Signal Proc.*, ASSP-39(1):92–114, Jan. 1991.

[14] T. Kailath. *Linear Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[15] J-L Botto and G.V. Moustakides. "Stabilization of Fast Recursive Least-Squares Transversal Filters for Adaptive Filtering". In *Proc. ICASSP 87 Conf.*, volume 1, pages 403–407, Dallas, Texas, April 1987.

[16] A. Benallal and A. Gilloire. "A New Method to Stabilize Fast RLS Algorithms based on a First-Order Model of the Propagation of Numerical Errors". In *Proc. ICASSP 88 Conf.*, volume 3, pages 1373–1376, New York, April 1988.

[17] N. Levinson. "The Wiener r.m.s. (root-mean-square) error criterion in filter design and prediction". *J. Math. Phys.*, 25:261–278, 1947.

[18] M. Vetterli. "Fast Algorithms for Signal Processing". In M. Kunt, editor, *Techniques modernes de traitement numérique des signaux*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1991. ISBN 2-88074-207-2.

[19] G.V. Moustakides and S. Theodoridis. "Fast Newton Transversal Filters – A New Class of Adaptive Estimation Algorithms". *IEEE Trans. SP*, SP-39(10):2184–2193, Oct. 1991.

[20] T. Petillon, A. Gilloire, and S. Theodoridis. "The Fast Newton Transversal Filter: an Efficient Scheme for Acoustic Echo Cancellation in Mobile Radio". *IEEE Trans. ASSP*, ASSP-42(3):509–518, March 1994.

[21] K. Ozeki and T. Umeda. "An Adaptive Algorithm Using Orthogonal Projection to an Affine Subspace and its Properties". *Trans. IECE Japan*, J67-A:126–132, 1984.

[22] S. L. Gay. "A Fast Converging Low Complexity Adaptive Filtering Algorithm". In *Proc. ICASSP Conf.*, pages 3023–3026, Detroit, USA, May 1995.

[23] K. Maouche and D.T.M. Slock. "The Fast Subsampled-Updating Fast Newton Transversal Filter (FSU FNTF) for Adapting Long FIR Filters". In *Proc. 28th Asilomar Conf. on Signals, Systems and Computers*, pages 1488–1492, Pacific Grove, CA, Oct. 31 - Nov. 2 1994.

[24] K. Maouche. *Algorithmes des Moindres Carrés Récursifs Doublement Rapides: Application à l'identification de Réponses impulsionnelles Longues.* PhD thesis, Ecole Nationale Supérieure des Télécommunications, Paris, France, March 1996.

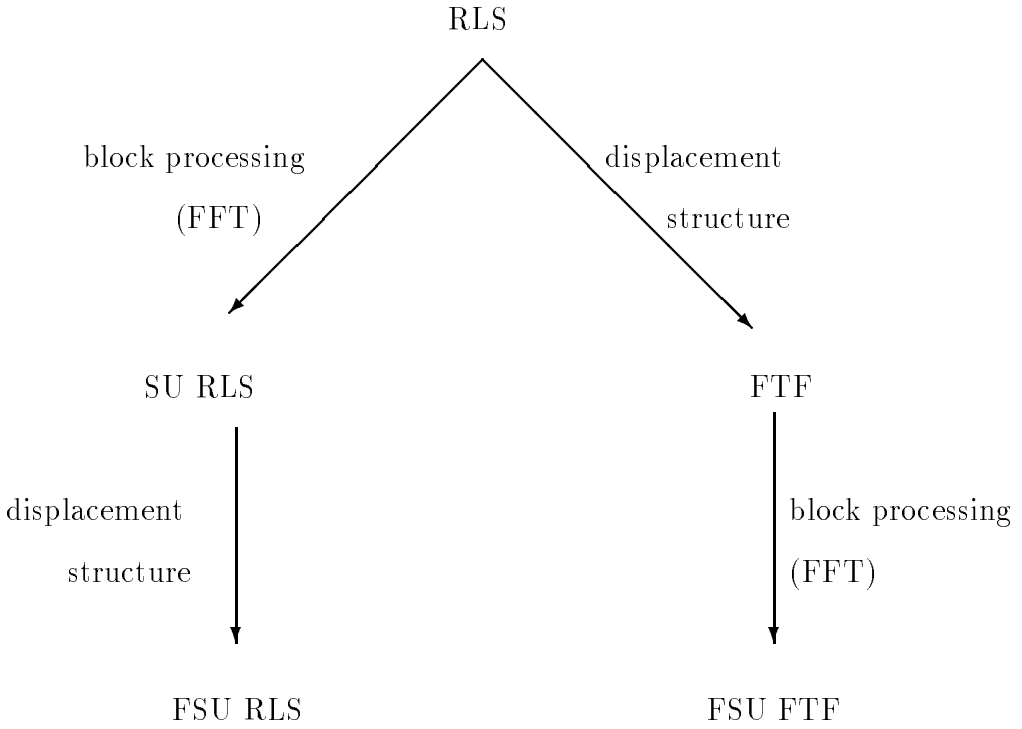

Figure 1: Dual strategies for the derivation of the FSU FTF and FSU RLS algorithms.

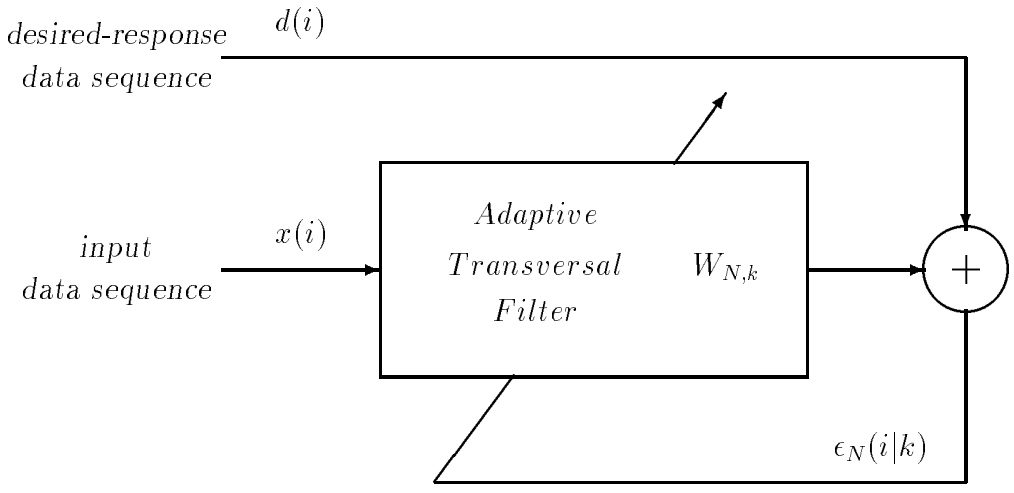

Figure 2: The adaptive FIR filtering scheme.

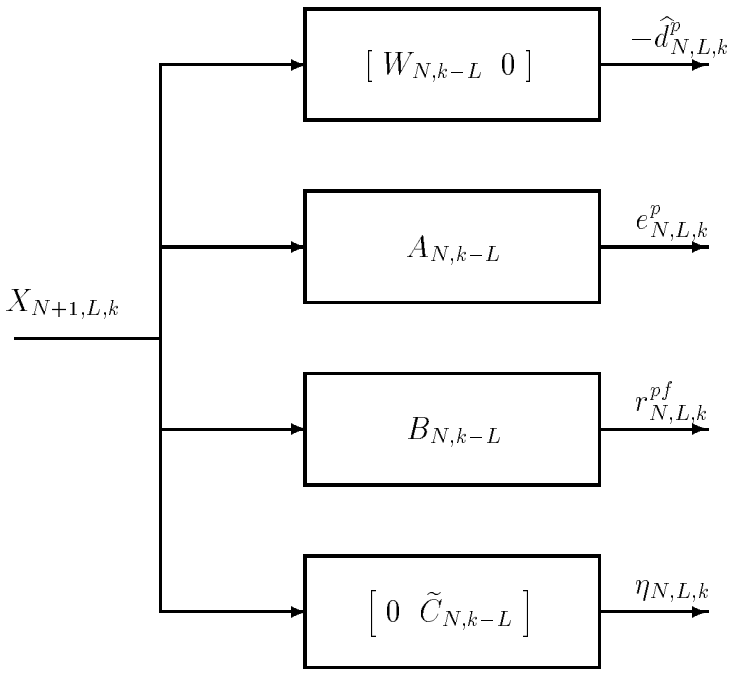

Figure 3: Filtering operations in the FSU SFTF algorithm.

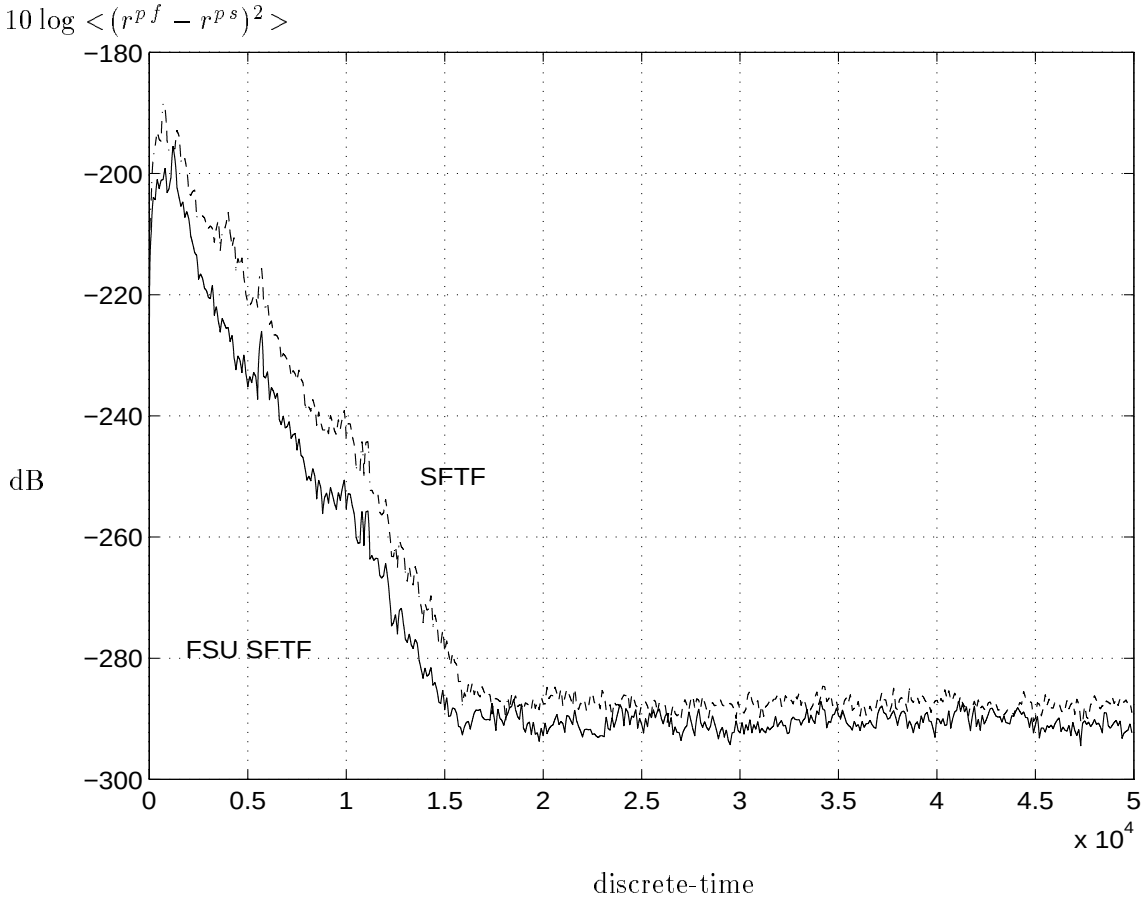

Figure 4: Evolution of numerical errors for the SFTF algorithm ('– –') and the FSU SFTF algorithm ('—')

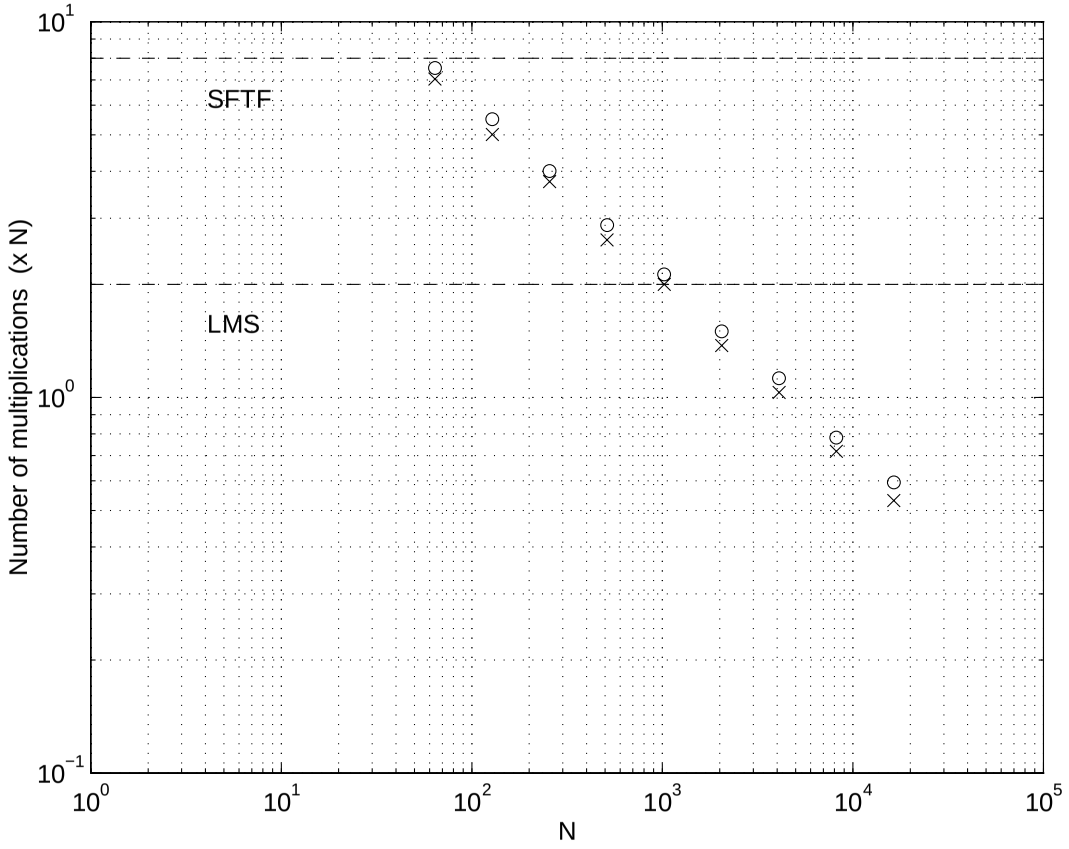

Figure 5: FSU SFTF ('$\circ$') and FSU RLS ('$\times$') complexities *vs.* filter length.

The SFTF Algorithm		
#	**Computation**	**Cost per sample**
1	$e_N^p(k) = A_{N,k-1} X_{N+1}(k)$	N
2	$\widetilde{C}^{0\,H}_{N+1,k} = -\lambda^{-1}\alpha_N^{-1}(k-1) e_N^p(k)$	
3	$\widetilde{C}_{N+1,k} = \left[\begin{array}{cc} 0 & \widetilde{C}_{N,k-1} \end{array}\right] + \widetilde{C}^0_{N+1,k} A_{N,k-1}$	N
4	$\gamma_{N+1}^{-s}(k) = \gamma_N^{-1}(k-1) - \widetilde{C}^0_{N+1,k} e_N^p(k)$	
5	$e_N(k) = e_N^p(k)\gamma_N(k-1)$	
6	$A_{N,k} = A_{N,k-1} + e_N(k) \left[\begin{array}{cc} 0 & \widetilde{C}_{N,k-1} \end{array}\right]$	N
7	$\alpha_N^{-1}(k) = \lambda^{-1}\alpha_N^{-1}(k-1) - \widetilde{C}^{0\,H}_{N+1,k}\gamma^s_{N+1}(k)\widetilde{C}^0_{N+1,k}$	
8	$r_N^{p\,f}(k) = B_{N,k-1} X_{N+1}(k)$	N
9	$r_N^{p\,s}(k) = -\lambda\beta_N(k-1)\widetilde{C}^{N\,H}_{N+1,k}$	
10	$r_N^{p\,(1)}(k) = 1.5\, r_N^{p\,f}(k) - 0.5\, r_N^{p\,s}(k)$	
11	$r_N^{p\,(2)}(k) = 2.5\, r_N^{p\,f}(k) - 1.5\, r_N^{p\,s}(k)$	
12	$\left[\begin{array}{cc} \widetilde{C}_{N,k} & 0 \end{array}\right] = \widetilde{C}_{N+1,k} - \widetilde{C}^N_{N+1,k} B_{N,k-1}$	N
13	$\gamma_N^{-s}(k) = \gamma_{N+1}^{-s}(k) + \widetilde{C}^N_{N+1,k} r_N^{p\,f}(k)$	
14	$r_N^{(j)}(k) = r_N^{p\,(j)}(k)\gamma_N^s(k)\ ,\ j = 1, 2$	
15	$B_{N,k} = B_{N,k-1} + r_N^{(1)}(k) \left[\begin{array}{cc} \widetilde{C}_{N,k} & 0 \end{array}\right]$	N
16	$\beta_N(k) = \lambda\beta_N(k-1) + r_N^{(2)}(k) r_N^{p\,(2)H}(k)$	
17	$\gamma_N^{-1}(k) = \lambda^{-N}\alpha_N(k)\beta_N^{-1}(k)$	
18	$\epsilon_N^p(k) = d(k) + W_{N,k-1} X_N(k)$	N
19	$\epsilon_N(k) = \gamma_N(k)\epsilon_N^p(k)$	
20	$W_{N,k} = W_{N,k-1} + \epsilon_N(k)\widetilde{C}_{N,k}$	N
Total Cost per Sample:		$8N$

Table 1: The SFTF Algorithm.

The Schur-SFTF Procedure

0	$F_L(k)$	$=$	$\begin{bmatrix} [\,0 \;\; \widetilde{C}_{N,k-L}\,] \\ A_{N,k-L} \\ B_{N,k-L} \\ [\,W_{N,k-L} \;\; 0\,] \end{bmatrix} \quad X^H_{N+1,L,k} = \begin{bmatrix} \eta^H_{N,L,k} \\ e^{p\,H}_{N,L,k} \\ r^{pf\,H}_{N,L,k} \\ -\widehat{d}^{\,p\,H}_{N,L,k} \end{bmatrix}$
	For j	$=$	$1:L\ ,\ k_j = k-L+j\ ,\ n_j = N-L+j$
1	$F_{L-j+1}(k)\ u_{L-j+1,1}$	$=$	$\begin{bmatrix} 1-\gamma_N^{-1}(k_j-1) \\ e^p_N(k_j) \\ r^{pf}_N(k_j) \\ -\widehat{d}^{\,p}_N(k_j) \end{bmatrix}$
2	$\widetilde{C}^{n_j:N}_{N+1,k_j}$	$=$	$\widetilde{C}^{n_j-1:N-1}_{N,k_j-1} - \lambda^{-1}\alpha_N^{-1}(k_j-1)\, e^{p\,H}_N(k_j)\, A^{n_j:N}_{N,k_j-1}$
3	$r^{ps}_N(k_j)$	$=$	$-\lambda\beta_N(k_j-1)\,\widetilde{C}^{N\,H}_{N+1,k_j}$
4	$e_N(k_j)$	$=$	$\gamma_N(k_j-1)e^p_N(k_j)$
5	$A^{n_j+1:N}_{N,k_j}$	$=$	$A^{n_j+1:N}_{N,k_j-1} + e_N(k_j)\,\widetilde{C}^{n_j:N-1}_{N,k_j-1}$
6	$\left[\,\widetilde{C}^{n_j:N-1}_{N,k_j} \;\; 0\,\right]$	$=$	$\widetilde{C}^{n_j:N}_{N+1,k_j} - \widetilde{C}^{N}_{N+1,k_j}\ B^{n_j:N}_{N,k_j-1}$
7	$\gamma^{-s}_{N+1}(k_j)$	$=$	$\gamma_N^{-1}(k_j-1) + \lambda^{-1}e^p_N(k_j)\alpha_N^{-1}(k_j-1)e^{p\,H}_N(k_j)$
8	$\alpha_N(k_j)$	$=$	$\lambda\alpha_N(k_j-1) + e^p_N(k_j)\, e^H_N(k_j)$
9	$r^{p\,(1)}_N(k_j)$	$=$	$1.5\, r^{pf}_N(k_j) - 0.5\, r^{ps}_N(k_j)$
10	$r^{p\,(2)}_N(k_j)$	$=$	$2.5\, r^{pf}_N(k_j) - 1.5\, r^{ps}_N(k_j)$
11	$\gamma^{-s}_N(k_j)$	$=$	$\gamma^{-1}_{N+1}(k_j) - \lambda^{-1}r^p_N(k_j)\beta_N^{-1}(k_j-1)r^{p\,H}_N(k_j)$
12	$r^{(j)}_N(k_j)$	$=$	$r^{p\,(j)}_N(k_j)\gamma^s_N(k_j)\ ,\ j=1,2$
13	$B^{n_j+1:N}_{N,k_j}$	$=$	$B^{n_j+1:N}_{N,k_j-1} + r^{(1)}_N(k_j)\left[\,\widetilde{C}^{n_j+1:N-1}_{N,k_j} \;\; 0\,\right]$
14	$\beta_N(k_j)$	$=$	$\lambda\beta_N(k_j-1) + r^{(2)}_N(k_j)r^{p\,(2)^H}_N(k_j)$
15	$\gamma_N^{-1}(k_j)$	$=$	$\lambda^{-N}\alpha_N(k_j)\beta_N^{-1}(k_j)$
16	$\epsilon_N(k_j)$	$=$	$\gamma_N(k_j)\epsilon^p_N(k_j)$
17	$\mathcal{S}\left(\Theta_{k_j}\, F_{L-j+1}(k)\right)$	$=$	$F_{L-j}(k)$

Table 2: The Schur-SFTF Procedure.

The FSU SFTF Algorithm		
#	**Computation**	**Cost per L sample**
1	$\begin{bmatrix} \eta^H_{N,L,k} \\ e^{p\,H}_{N,L,k} \\ r^{p\,f\,H}_{N,L,k} \\ -\widehat{d}^{\,p\,H}_{N,L,k} \end{bmatrix} = \begin{bmatrix} \left[0\ \widetilde{C}_{N,k-L}\right] \\ A_{N,k-L} \\ B_{N,k-L} \\ [W_{N,k-L}\ 0] \end{bmatrix} X^H_{N+1,L,k}$	$(5+4\frac{N+1}{L})FFT(2L)+8N$
2	Schur-SFTF Procedure: Input: $\eta_{N,L,k},\ e^p_{N,L,k},\ r^{p\,f}_{N,L,k},\ -\widehat{d}^{\,p}_{N,L,k}$ $A^{N-L+1:N}_{N,k_1-1}, B^{N-L:N-1}_{N,k_1-1}, \widetilde{C}^{N-L+1:N}_{N,k_1-1}$ Output: $\Theta_{k-i}(z)\ \ i=L-1,\ldots,0$	$4.5L^2$
3	$\Theta_{k,L}(z) = \prod_{i=0}^{L-1}\Theta_{k-i}(z)$	$7.5L^2$
4	$\begin{bmatrix} \widetilde{C}_k(z) \\ A_k(z) \\ B_k(z) \\ W_k(z) \end{bmatrix} = \Theta_{k,L}(z) \begin{bmatrix} \widetilde{C}_{k-L}(z) \\ A_{k-L}(z) \\ B_{k-L}(z) \\ W_{k-L}(z) \end{bmatrix}$	$(12+4\frac{N+1}{L})FFT(2L)+24N$
Total Cost per Sample:		$(17+8\frac{N+1}{L})\frac{FFT(2L)}{L}+32\frac{N}{L}+12L$

Table 3: The FSU SFTF Algorithm.

N	63	127	255	511	1023	2047	4095	8191	16383
L (optimal)	16	32	32	64	64	128	128	256	512
Complexity ($\times N$)	7.53	5.51	4.01	2.88	2.13	1.50	1.13	0.78	0.59

Table 4: Optimal Complexity of the FSU SFTF Algorithm.