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

Research Report N° 94-015

The Fast Subsampled-Updating Stabilized Fast Transversal Filter (FSU SFTF) Algorithm for Adaptive Filtering Based on a Schur Procedure and the FFT

Dirk T.M. Slock   Karim Maouche

November, 1994

Telephone:	+33 93 00 26 26	E-mail:
Dirk T.M. Slock:	+33 93 00 26 06	slock@eurecom.fr
Karim Maouche:	+33 93 00 26 32	maouche@eurecom.fr
Fax:	+33 93 00 26 27	

Abstract

We present a new 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 based on the Stabilized Fast Transversal Filter (SFTF) algorithm, which is a numerically stabilized version of the classical 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 thus appear in the algorithm are done using the Fast Fourier Transform (FFT). For relatively long filters, the computational complexity of the new algorithm is smaller than the one of the well-known LMS algorithm, rendering it especially suitable for applications such as acoustic echo cancellation.

Contents

1 Introduction

Nowadays, adaptive filtering became an important tool in digital signal processing. Recent advances in VLSI technology have rendered this technique very attractive and have led to diverse applications such as equalization, echo cancellation, interference cancelling, signal detection, etc. [3]. The coefficients of an adaptive filter vary periodically 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 builded around the Least-Mean-Square (LMS) algorithm [9]. The LMS algorithm minimizes a mean square 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 modelizes the acoustic path is relatively large and the input signal is highly correlated (speech signal), LMS algorithm does not provide a satisfactory solution because of the very low convergence rate of the filter estimate. The mainstay of the second family is the Recursive Least-Squares (RLS) algorithm that minimizes a deterministic sum of squared errors. The RLS algorithm is known to perform much better than the LMS algoritm [2] but shows a computational complexity of $\mathcal{O}(N^2)$ operations which disqualify it from being used in applications where the FIR filter is relatively long. Fast RLS algorithms such as the Fast Transversal Filter (FTF) algorithm [1] exploit a certain shift invariance structure in the input data vector to reduce the computational complexity to $7N$ for the FTF algorithm.
In [6], we have pursued an alternative way to reduce the complexity of RLS adaptive filtering algorithms. The approach consists of subsampling the filter adaptation, i.e. the LS filter estimate is no longer provided every sample but every $L \geq 1$ samples (subsampling factor L). This leads to 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 w.r.t. 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 resp. which, due to the shift invariance present in the problem, exhibit a low displacement rank. Hence, by using the displacement structure and the FFT (when computing convolutions), we have derived a fast version of SU RLS that we have called the FSU RLS algorithm.
In [7], we have proposed a dual strategy that allowed us to derive the FSU FTF algorithm, see Fig. 1. Namely, we exploit shift-invariance in the RLS algorithm to obtain the FTF algorithm. Hence, 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 algorithm, 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.

$$\begin{tikzcd}
 & \text{RLS} \arrow[dl, "\text{block processing (FFT)}"'] \arrow[dr, "\text{displacement structure}"] & \\
\text{SU RLS} \arrow[d, "\text{displacement structure}"'] & & \text{FTF} \arrow[d, "\text{block processing (FFT)}"] \\
\text{FSU RLS} & & \text{FSU FTF}
\end{tikzcd}$$

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

Applying the L rotation matrices to the filter vectors becomes an issue of multiplying polynomials, which can be efficiently carried out using the FFT. Unfortunately, the FTF algorithm is numerically instable because of round-off error accumulation that arises with finite precision implementation. Inheritting the round-off errors dynamic of the SFTF algorithm, the FSU FTF algorithm is also numerically instable. Recently, the Stabilized FTF (SFTF) algorithm, a numerically stable version of the FTF algorithm, was derived which shows a computational complexity of $8N$ [5]. 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 derivation of the Schur-SFTF procedure that allows the computation of the successive rotation matrices of the SFTF algorithm. The SUS turns out to be especially applicable in the case of very long filters such as occur in the acoustic echo cancellation problem. The computational gain it offers 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.

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 $d(i)$.

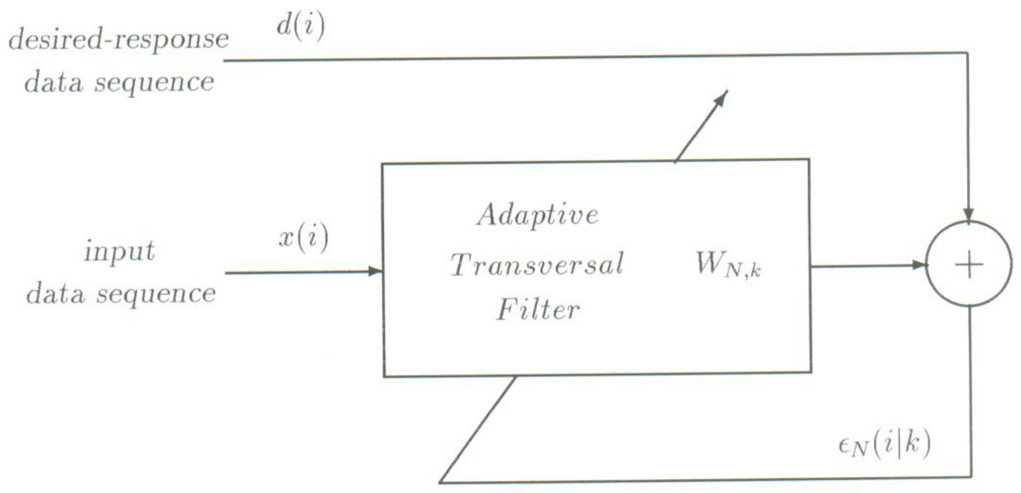

Figure 2: The adaptive FIR filtering scheme.

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

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

where $\lambda \in (0,1]$ is the exponential weighting factor, $\mu > 0$, $\Lambda_N = \operatorname{diag}\left\{\lambda^{N-1}, \ldots, \lambda, 1\right\}$, $\|v\|_\Lambda^2 = v\Lambda v^H$, $\|.\| = \|.\|_I$. 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

$$W_{N,k} \;=\; -P_{N,k}^H R_{N,k}^{-1} \;\;, \tag{3}$$

where

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

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 [4, pg 656] for $R^{-1}_{N,k}$, we obtain the RLS algorithm:

$$\begin{aligned}
\widetilde{C}_{N,k} &= -X^{H}_{N}(k)\lambda^{-1}R^{-1}_{N,k-1} && (5)\\
\gamma^{-1}_{N}(k) &= 1-\widetilde{C}_{N,k}X_{N}(k) && (6)\\
R^{-1}_{N,k} &= \lambda^{-1}R^{-1}_{N,k-1} - \widetilde{C}^{H}_{N,k}\gamma_{N}(k)\widetilde{C}_{N,k} && (7)\\
\epsilon^{p}_{N}(k) &= \epsilon_{N}(k|k-1) = d(k)+W_{N,k-1}X_{N}(k) && (8)\\
\epsilon_{N}(k) &= \epsilon_{N}(k|k) = \epsilon^{p}_{N}(k)\,\gamma_{N}(k) && (9)\\
W_{N,k} &= W_{N,k-1}+\epsilon_{N}(k)\widetilde{C}_{N,k}\,, && (10)
\end{aligned}$$

where $\epsilon^{p}_{N}(k)$ 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 [1]) that they are related by the likelihood variable $\gamma_{N}(k)$ as in (9).
RLS algorithm shows a computational complexity of $\mathcal{O}(N^2)$ operations per sample. This is too much demanding when N is relatively large and with actual technology real-time implementation is not possible. Fast versions of RLS algorithms have been derived using a certain shift invariance property of the adaptive FIR filtering problem. These fast versions constitutes two major families of RLS algorithms. One family uses transversal filters while the other uses treillis filters. Each family shows a computational complexity of $\mathcal{O}(N)$ but the transversal family is the most popular because of its lower complexity. In particular, the SFTF algorithm shows a computational complexity of $8N$.

3 The SFTF Algorithm

In what follows we shall consider the single-channel case. However the generalization to the multichannel case can easily be done. The SFTF algorithm can be described in the following way, which emphasizes its rotational structure:

$$\begin{bmatrix} [\,\widetilde{C}_{N,k}\;0\,] \\ A_{N,k} \\ B_{N,k} \\ [\,W_{N,k}\;0\,] \end{bmatrix} = \Theta_k \begin{bmatrix} [\,0\;\widetilde{C}_{N,k-1}\,] \\ A_{N,k-1} \\ B_{N,k-1} \\ [\,W_{N,k-1}\;0\,] \end{bmatrix}$$

$$\begin{aligned}
e^{p}_{N}(k) &= A_{N,k-1}X_{N+1}(k)\\
e_{N}(k) &= e^{p}_{N}(k)\gamma_{N}(k-1)\\
\gamma^{-s}_{N+1}(k) &= \gamma^{-1}_{N}(k-1)-\widetilde{C}^{0}_{N+1,k}e^{p}_{N}(k)\\
\gamma^{-s}_{N}(k) &= \gamma^{-s}_{N+1}(k)+\widetilde{C}^{N}_{N+1,k}r^{pf}_{N}(k) && (11)\\
r^{ps}_{N}(k) &= -\lambda\beta_{N}(k-1)\widetilde{C}^{N\,H}_{N+1,k}\\
r^{pf}_{N}(k) &= B_{N,k-1}X_{N+1}(k)\\
\alpha^{-1}_{N}(k) &= \lambda^{-1}\alpha^{-1}_{N}(k-1)-\widetilde{C}^{0\,H}_{N+1,k}\gamma^{s}_{N+1}(k)\widetilde{C}^{0}_{N+1,k}
\end{aligned}$$

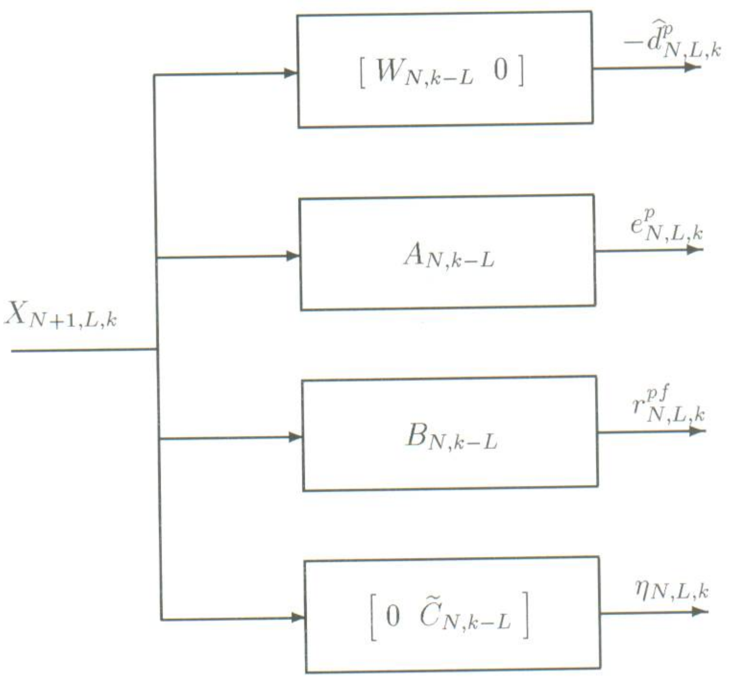

Figure 3: Filtering operations in the FSU SFTF algorithm.

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 defines θ_{k-L+1} , we must compute $r^{ps}_{N}(k-L+1)$ (this allows the computation of $r^{(1)}_{N}(k-L+1)$) and $\widetilde{C}^{N}_{N+1,k-L+1}$. Since $r^{ps}_{N}(k-L+1) = -\lambda\beta_N(k-L)\widetilde{C}^{N}_{N+1,k-L+1}$ (see 11), 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 SFTF prediction part filters

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

$$\begin{aligned}
m &= N-L+j \\
K &= k-L+j \\
\widetilde{C}^{m:N}_{N+1,K} &= \widetilde{C}^{m-1:N-1}_{N,K-1} - \lambda^{-1}\alpha^{-1}(K-1)\, e^{p\,H}_{N}(K)\, A^{m:N}_{N,K-1} \\
\left[\,\widetilde{C}^{m:N-1}_{N,K}\;\;0\,\right] &= \widetilde{C}^{m:N}_{N+1,K} - \widetilde{C}^{N+1}_{N,K}\; B^{m:N}_{N,K-1} \\
A^{m+1:N}_{N,K} &= A^{m+1:N}_{N,K-1} + e_N(K)\,\widetilde{C}^{m:N-1}_{N,K-1} \\
B^{m+1:N}_{N,K} &= B^{m+1:N}_{N,K-1} + r^{(1)}_{N}(K)\left[\,\widetilde{C}^{m+1:N-1}_{N,K}\;\;0\,\right]
\end{aligned}$$

$$r_N^{ps}(K) \;=\; -\lambda\beta_N(K-1)\,\tilde{C}^{N\,H}_{N+1,K}\;. \tag{20}$$

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 in (11) and the order down-date recursions (20), it is possible to construct the successive Θ_{k-L+j} , $j=1,\ldots,L$.
Now we rotate both expressions for $F_L(k)$ in (15) with Θ_{k-L+1} to obtain $\Theta_{k-L+1}F_L(k)$ which equals

$$\begin{bmatrix} \left[\,\tilde{C}_{N,k-L+1}\ 0\,\right] \\ A_{N,k-L+1} \\ B_{N,k-L+1} \\ \left[\,W_{N,k-L+1}\ 0\,\right] \end{bmatrix} X^H_{N+1,L,k} = \begin{bmatrix} \boxed{\eta^H_{N,L-1,k}} & * \\ e_N(k-L+1) & \boxed{e^{p\,H}_{N,L-1,k}} \\ r^f_N(k-L+1) & \boxed{r^{pf\,H}_{N,L-1,k}} \\ -\hat{d}_N\,(k-L+1) & \boxed{-\hat{d}^{p\,H}_{N,L-1,k}} \end{bmatrix}\;. \tag{21}$$

One can see from (21) that quantities in boxes are the four rows of $F_{L-1}(k)$. This can be written more compactly as

$$\mathcal{S}\left(\Theta_{k-L+1}\,F_L(k)\right) \;=\; F_{L-1}(k)\;, \tag{22}$$

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 prcedure, the one step ahead output errors $\epsilon^p_N(k-L+j)$ are computed during this time span without updating the estimate filter. Inner products (filtering operations) are needed for the computation of $F_L(k)$. This is the Schur-SFTF algorithm, which contrasts with the Levinson-style SFTF algorithm in (11). Taking into account the fact that a rotation matrix in factored form as in (13) 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 algorithm 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 [8]. 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 (15), 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:

$$v_{N+1,k} = \left[\,v^1_{N+1,k}\ \cdots\ v^M_{N+1,k}\,\right]\;, \tag{23}$$

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

$$X_{N+1,L,k} = [\, X_{L,L,k} \; X_{L,L,k-L} \; \cdots X_{L,L,k-N+L-1} \,] \;\; , \tag{24}$$

then

$$v_{N+1,k}\, X^H_{N+1,L,k} = \sum_{j=1}^{N_L} v^j_{N+1,k} X^H_{L,L,k-(j-1)L} \;\; . \tag{25}$$

In other words, we have essentially N_L times the product of a a vector of length L with a $(L \times L)$ Toeplitz matrix. Such a product can be efficiently computed in basically two different ways [8]. 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. We can embed the $L \times L$ Toeplitz matrix $X_{L,L,k}$ into a $2L \times 2L$ circulant matrix, viz.

$$\underline{X}^H_{L,L,k} \;=\; \begin{bmatrix} * & X^H_{L,L,k} \\ X^H_{L,L,k} & * \end{bmatrix} \;=\; \mathcal{C}\left(x^H_{2L,k}\right) \tag{26}$$

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

$$v^j_{N+1,k} X^H_{L,L,k-(j-1)L} = \begin{bmatrix} 0_{1\times L} \; v^j_{N+1,k} \end{bmatrix} \mathcal{C}\left(x^H_{2L,k-(j-1)L}\right) \begin{bmatrix} I_L \\ 0_{L\times L} \end{bmatrix} \;\; . \tag{27}$$

Now consider the Discrete Fourier Transform (DFT) $\mathcal{V}^j_{N+1,k}$ of $v^j_{N+1,k}$

$$\mathcal{V}^j_{N+1,k} = v^j_{N+1,k}\, F_L \;\; , \tag{28}$$

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 $\frac{1}{L}F_L^H$. It defines the inverse DFT transformation (IDFT)

$$v^j_{N+1,k} = \mathcal{V}^j_{N+1,k}\, \frac{1}{L} F_L^H \;\; . \tag{29}$$

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

$$v\,\mathcal{C}(c^H) \;=\; v\,F_m \;\mathrm{diag}\left(c^H F_m\right) \frac{1}{m} F_m^H \;=\; \left[(vF_m)\;\; \mathrm{diag}\left(c^H F_m\right)\right] \frac{1}{m} F_m^H \;\; , \tag{30}$$

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 (27) 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-FNTF procedure to

$$4N \left[\frac{\mathrm{FFT}(2L)}{L^2} + \frac{2}{L}\right] + \frac{5\mathrm{FFT}(2L)}{L} \tag{31}$$

computations per sample ($\mathrm{FFT}(L)$ 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

$$\begin{bmatrix} \widetilde{C}_k(z) \\ A_k(z) \\ B_k(z) \\ W_k(z) \end{bmatrix} = \begin{bmatrix} [\,\widetilde{C}_{N,k}\ 0\,] \\ A_{N,k} \\ B_{N,k} \\ [\,W_{N,k}\ 0\,] \end{bmatrix} \begin{bmatrix} 1 \\ z^{-1} \\ \vdots \\ z^{-N} \end{bmatrix} . \tag{32}$$

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

$$\begin{bmatrix} \widetilde{C}_k(z) \\ A_k(z) \\ B_k(k) \\ W_k(z) \end{bmatrix} = \Theta_k \begin{bmatrix} z^{-1} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \widetilde{C}_{k-1}(z) \\ A_{k-1}(z) \\ B_{k-1}(z) \\ W_{k-1}(z) \end{bmatrix} . \tag{33}$$

Let's introduce the following polynomial matrix

$$\Theta_k(z) = \Theta_k \begin{bmatrix} z^{-1} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} . \tag{34}$$

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

$$\begin{bmatrix} \widetilde{C}_k(z) \\ A_k(z) \\ B_k(k) \\ W_k(z) \end{bmatrix} = \Theta_{k,L}(z) \begin{bmatrix} \widetilde{C}_{k-L}(z) \\ A_{k-L}(z) \\ B_{k-L}(z) \\ W_{k-L}(z) \end{bmatrix} \tag{35}$$

where

$$\Theta_{k,L}(z) = \Theta_k(z)\,\Theta_{k-1}(z)\cdots\Theta_{k-L+1}(z) . \tag{36}$$

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

$$\Theta_{k,L}(z) = \begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 1 \end{bmatrix} \tag{37}$$

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 $j = 2, \dots, L$

$$\begin{aligned} l &= k{-}L{+}j \\ \Theta_{l,j}(z) &= \Theta_l^4\, \Theta_l^3\, \Theta_l^2\, \Theta_l^1 \begin{bmatrix} z^{-1} & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \Theta_{l-1,j-1}(z) \ . \end{aligned} \tag{38}$$

The computation of $\Theta_{k,L}(z)$ takes $7.5L^2$ operations. As a result of the structure displayed in (37), the product in (35) 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 L order 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

$$\mathcal{P}_L \star \Phi_{N+1,k} = \mathcal{P}_L \star \left[\, \Phi^1_{N+1,k} \cdots \Phi^{N_L}_{N+1,k} \,\right] = \left[\, \mathcal{P}_L \star \Phi^1_{N+1,k} \cdots \mathcal{P}_L \star \Phi^{N_L}_{N+1,k} \,\right] \ , \tag{39}$$

every subproduct in (39) 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 estimate filter, we have

$$W_k(z) = \left(\Theta_{k,L}(z)\right)_{4,1} \widetilde{C}_{k-L}(z) + \left(\Theta_{k,L}(z)\right)_{4,2} A_{k-L}(z) + \left(\Theta_{k,L}(z)\right)_{4,3} B_{k-L}(z) + W_{k-L}(z) \ , \tag{40}$$

each product in (40) 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 estimate 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 I.
The initialization of the algorithm is done with the soft constraint initialization technique [1]. 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 FTF algorithm departs from the following initial conditions

$$\begin{aligned} W_{N,0} &= W_0 \\ A_{N,0} &= [\, 1\, 0 \cdots 0\,]\ , \quad \alpha_N(0) = \lambda^N \mu \\ B_{N,0} &= [\, 0 \cdots 0\, 1\,]\ , \quad \beta_N(0) = \mu \\ \widetilde{C}_{N,0} &= [\, 0 \cdots 0\,]\ , \quad \gamma_N(0) = 1\ . \end{aligned} \tag{41}$$

With this initialization at $k = 0$, the corresponding initial sample covariance matrix is indeed $R_0 = \mu \lambda \Lambda_N$.

7 Concluding Remarks

The complexity of the FSU SFTF is $\mathcal{O}((8\frac{N+1}{L}+17)\frac{FFT(2L)}{L}+32\frac{N}{L}+12L)$ 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. We have simulated the algorithm and have verified that it works. In [6], we have introduced the FSU RLS algorithm, an alternative algorithm with a very similar computational complexity, but a very different internal structure. These developments leads 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.

Table I: the FSU SFTF Algorithm

#	Computation	Cost per L samples
1	$\begin{bmatrix} \eta^{H}_{N,L,k} \\ e^{p\,H}_{N,L,k} \\ r^{pf\,H}_{N,L,k} \\ -\widehat{d}^{\,p\,H}_{N,L,k} \end{bmatrix} = \begin{bmatrix} \left[\, 0\ \widetilde{C}_{N,k-L} \,\right] \\ A_{N,k-L} \\ B_{N,k-L} \\ [W_{N,k-L}\ 0] \end{bmatrix} X^{H}_{N+1,L,k}$	$(5+4\frac{N+1}{L})FFT(2L)+8N$
2	SFTF-Schur Algorithm: Input: $\eta_{N,L,k},\ e^{p}_{N,L,k},\ r^{pf}_{N,L,k},\ -\widehat{d}^{\,p}_{N,L,k}$ Output: $\Theta_{k-i}(z)\ ,\ i=L-1,\cdots,0$	$4.5L^2$
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	$(17+8\frac{N+1}{L})\frac{FFT(2L)}{L}+32\frac{N}{L}+12L$

References

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] B. Widrow et al., "Stationary and nonstationary learning charactristics of the LMS adaptive filter", *Proc. IEEE*, vol. 64, No. 8, August 1976, pp. 1151–1162.