

## Adaptive FIR Filters.

John Aasted Sørensen

### Contents

<b>1</b>	<b>Introduction to FIR Adaptive Filters.</b>	<b>2</b>
<b>2</b>	<b>The Least Mean Squares (LMS) FIR Filter.</b>	<b>4</b>
2.1	Derivation of the LMS FIR Filter. . . . .	4
2.2	Derivation of the NLMS FIR Filter. . . . .	7
2.3	Some Properties of the LMS Algorithm. . . . .	7
2.4	The LMS Filter Algorithm . . . . .	10
2.5	The NLMS Filter Algorithm . . . . .	11
<b>3</b>	<b>The Recursive Least Squares (RLS) FIR filter.</b>	<b>12</b>
3.1	Basic Concepts . . . . .	12
3.2	Derivation of the RLS FIR Filter. . . . .	12
3.3	A Few RLS Properties. . . . .	15
3.3.1	Recursion for the Sum of Weighted Error Squares. . . . .	15
3.3.2	Rank-one Update of the Inverse Correlation Matrix. . . . .	15
3.3.3	The Conversion Factor. . . . .	16
3.4	The RLS FIR Filter Algorithm . . . . .	16
<b>4</b>	<b>From the RLS to the FTF and further to the SFTF algorithm.</b>	<b>16</b>
4.1	From the RLS to the FTF Algorithm. . . . .	17
4.2	The FTF Algorithm . . . . .	19
4.2.1	Input Signals and Internal Variables . . . . .	19
4.2.2	Computations . . . . .	19
4.3	From the FTF to the SFTF algorithm. . . . .	21
4.4	The SFTF Algorithm. . . . .	22
4.4.1	Input Signals and Internal Variables . . . . .	22
4.4.2	Computations . . . . .	23
<b>5</b>	<b>Examples on the Application of the SFTF Algorithm.</b>	<b>25</b>
5.1	Basic System Identification of a FIR Filter. . . . .	25
5.2	Adaptive Line Enhancement (ALE). . . . .	26
5.3	Cancellation of Acoustical Echo in a Car Cabin. . . . .	27
5.4	Cancellation of Noise Interference. . . . .	31
<b>6</b>	<b>Matlab Code Listing.</b>	<b>32</b>
6.1	SFTF Algorithm. . . . .	32
6.2	Basic System Identification of a FIR Filter. . . . .	36
6.3	Adaptive Line Enhancement (ALE). . . . .	37
6.4	Cancellation of Acoustical Echo in a Car cabin. . . . .	38
6.5	Cancellation of Noise Interference. . . . .	38

# 1 Introduction to FIR Adaptive Filters.

The objective of this note is to introduce a selection of adaptive FIR (Finite Impulse Response) filters shown in Fig. 1.

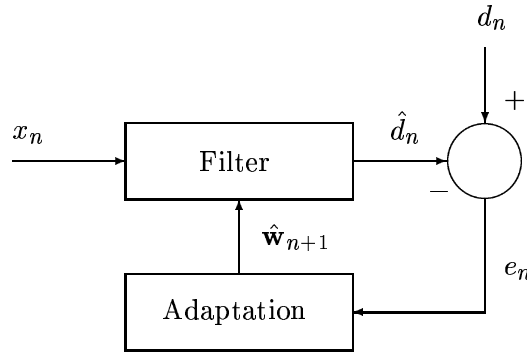


Figure 1: Adaptive FIR Filter.

These adaptive filters [4] (optimizing filters) might be applied when the input signal characteristics are not known a priori or are known to be varying in time. If an adaptive filter operates in a wide-sense stationary environment, then after some time the filter will converge to a solution close to the Wiener filter solution. In a nonstationary environment, the algorithm offers a tracking capability, in that it can track time variations in the statistics of the input signals, provided that the variations are sufficiently slow.

The adaptive filters operate as recursive algorithms when updating the filter coefficients from time step  $n$  to time step  $n + 1$ , thus the parameters are data dependent. This, therefore, means that an adaptive filter is in reality a nonlinear device, in the sense that it does not obey the principle of superposition. Despite of that these filters are often classified as linear or nonlinear. An adaptive filter is said to be linear if the estimate of a quantity of interest is computed adaptively (as the output of the filter) as a linear combination of the available set of observations applied to the filter input, otherwise the adaptive filter is said to be nonlinear. In the following only linear adaptive filters are treated.

A wide variety of recursive algorithms have been developed in the literature for the operation of linear adaptive filters. In the final analysis the choice of one algorithm over another is determined by one or more of the following factors:

- **Rate of convergence.** This is defined as the number of iterations required for the algorithm, in response to stationary inputs, to converge close to the optimal Wiener filter solution in the mean-square sense. A fast rate of convergence allows the algorithm to adapt rapidly to a stationary environment of unknown statistics.
- **Tracking.** When an adaptive filtering algorithm operates in a nonstationary environment, the algorithm is required to track statistical variations in the environment. The tracking performance of the algorithm, however, is influenced by two contradictory features: (1) rate of convergence, and (2) steady-state fluctuation due to algorithm noise. Please notice that rate of convergence and tracking are two very related concepts.
- **Misadjustment.** For an algorithm of interest, this parameter provides a quantitative measure of the amount by which the final value of the mean-squared error, averaged over an

ensemble of adaptive filters, deviates from the minimum mean-squared error that is produced by the Wiener filter.

- **Robustness.** For an adaptive filter to be robust, small disturbances (i.e. disturbances with small energy) can only result in small estimation errors. The disturbances may arise from a variety of factors, internal or external to the filter.
- **Computational requirements.** Here the issues of concern include (1) the number of operations (i.e. multiplications, divisions, and additions/subtractions) required to make one complete iteration of the algorithm, (2) the size of memory locations required to store the data and the program, and (3) the investment required to program the algorithm on a computer.
- **Numerical properties.** When an algorithm is implemented numerically, inaccuracies are produced due to quantization errors. The quantization errors are due to analog-to-digital conversion of the input data and digital representation of internal calculations. Ordinarily, it is the latter source of quantization errors that posed a serious design problem. In particular, there are two basic issues of concern: numerical stability and numerical accuracy. Numerical stability is an inherent characteristic of an adaptive filtering algorithm. Numerical accuracy, on the other hand, is determined by the number of bits (*binary digits*) used in the numerical representation of data samples and filter coefficients. An adaptive filtering algorithm is said to be numerically robust, when it is insensitive to variations in the wordlength used in its digital implementation.

The adaptive filters are characterized by the following groups of parameters, input/output signals and adaptation algorithms.

**Parameters:**

$M$ : the number of FIR filter coefficients.

One or a few adaptation constants, depending of the adaptation algorithm.

**Input signals:**

$x_n$ : input signal.

$\mathbf{x}_n = (x_n, \dots, x_{n-M+1})^T$ : Input signal vector.

$d_n$ : desired response.

**Output signals:**

$\hat{d}_n$ : estimated desired response.

$e_n$ : estimation error.

$\hat{\mathbf{w}}_n = (\hat{w}_{0,n}, \hat{w}_{1,n}, \dots, \hat{w}_{M-1,n})^T$ :  $M$  FIR filter coefficients at time  $n$ .

**Adaptation algorithms:**

Depending of the properties needed by the adaptive FIR filter, it is possible to choose among a selection of adaptation (optimization) criterias spanning a range of computational complexities and performances.

- The Least-Mean-Square and the Normalized Least-Mean-Square Algorithm recursively minimize  $\mathcal{E}_n = E(e_n^2)$ , based on the instantaneous estimates of the auto- and crosscorrelations of the input signals. The computational complexity and the memory requirement of this algorithm is  $\mathcal{O}(M)^1$ , where  $M$  is the number of FIR filter coefficients to be identified.

---

<sup>1</sup> $f(n) = \mathcal{O}(g(n))$  means, that there exist constants  $c$  and  $n_0$  such that  $|f(n)| \leq c|g(n)|$  for all  $n \geq n_0$

- The standard Recursive Least Squares (RLS) FIR filter minimizes recursively the exponentially weighted squared error signal, where  $\mathcal{E}_n = \sum_i \lambda^{n-i} e_i^2$ . The computational complexity and memory requirement of this algorithm is  $\mathcal{O}(M^2)$ , which for large  $M$  can be significant. The convergence is (much) faster than the LMS algorithm.
- The Stable Fast Transversal Filter (SFTF) (Transversal=FIR) is a further development of the standard RLS algorithm, towards lower computational complexity. The SFTF algorithm uses the same optimization criterium as the standard RLS algorithm:  $\mathcal{E}_n = \sum_i \lambda^{n-i} e_i^2$ ; but the computational complexity is reduced to  $\mathcal{O}(M)$  which is the level of the LMS or NLMS algorithm. Note that the number of arithmetic operations in the SFTF algorithm is larger than in the case of LMS or NLMS, but it increases linearly in  $M$ .

## 2 The Least Mean Squares (LMS) FIR Filter.

The LMS algorithm is an important member of the family of stochastic gradient algorithms and is devised by Widrow and Hoff in 1960, [9]. A thorough description and analysis of the LMS algorithm can be found in [4].

The term stochastic gradient is intended to distinguish the LMS algorithm from the method of steepest descent that uses a deterministic gradient in a recursive computation of the Wiener filter for stochastic inputs. A significant feature of the LMS algorithm is its simplicity. Moreover it does not require measurements of the pertinent correlation functions, nor does it require matrix inversion. It is the simplicity of the LMS algorithm that has made it the standard against which other adaptive filtering algorithms are benchmarked.

### 2.1 Derivation of the LMS FIR Filter.

Let the input signal be  $x_n$  and the corresponding vector with the  $M$  most recent values of  $x_n$

$$\mathbf{x}_n = (x_n, \dots, x_{n-M+1})^T. \quad (1)$$

If the desired response is denoted  $d_n$  the the estimation error becomes

$$e_n = d_n - \mathbf{w}_n^T \mathbf{x}_n. \quad (2)$$

In the following the input signals  $x_n$  and  $d_n$  are wide-sense stationary. The performance index expressed as the mean squared error then becomes, letting  $\langle x \rangle$  denote the time average value of  $x$  and  $\mathbf{w}_o$  the optimal value

$$\mathcal{E}_o = \langle e_n^2 \rangle \quad (3)$$

$$= \langle (d_n - \mathbf{w}_n^T \mathbf{x}_n)^2 \rangle \quad (4)$$

$$= \langle d_n^2 \rangle - 2\mathbf{w}_o^T \langle d_n \mathbf{x}_n \rangle + \langle \mathbf{w}_o^T \mathbf{x}_n \mathbf{w}_o^T \mathbf{x}_n \rangle \quad (5)$$

$$= \sigma_d^2 - 2\mathbf{w}_o^T \mathbf{p} + \mathbf{w}_o^T \langle \mathbf{x}_n \mathbf{x}_n^T \rangle \mathbf{w}_o \quad (6)$$

$$= \sigma_d^2 - 2\mathbf{w}_o^T \mathbf{p} + \mathbf{w}_o^T \mathbf{R} \mathbf{w}_o. \quad (7)$$

In the transition from Eq. (5) to Eq. (6) it is used that  $\mathbf{w}_n^T \mathbf{x}_n$  is a scalar and that  $\mathbf{x}_n^T \mathbf{w} = (\mathbf{w}^T \mathbf{x}_n)^T$ . The cross-correlation vector  $\mathbf{p}$  is defined by

$$\mathbf{p} = (\langle d_n x_n \rangle, \dots, \langle d_n x_{n-M+1} \rangle)^T. \quad (8)$$

The correlation-matrix

$$\mathbf{R} = \langle \mathbf{x}_n \mathbf{x}_n^T \rangle \in \mathcal{R}^{M \times M} \quad (9)$$

$$\mathbf{R} = \begin{pmatrix} \langle x_n x_n \rangle & \langle x_n x_{n-1} \rangle & \dots & \langle x_n x_{n-M+1} \rangle \\ \langle x_{n-1} x_n \rangle & \langle x_{n-1} x_{n-1} \rangle & \dots & \langle x_{n-1} x_{n-M+1} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_{n-M+1} x_n \rangle & \dots & \dots & \langle x_{n-M+1} x_{n-M+1} \rangle \end{pmatrix} \quad (10)$$

$$\mathbf{R} = \begin{pmatrix} r_0 & r_1 & \dots & r_{M-1} \\ r_1 & r_0 & \dots & r_{M-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{M-1} & \dots & \dots & r_0 \end{pmatrix}. \quad (11)$$

It is seen that  $\mathbf{R}$  is symmetric  $\mathbf{R}^T = \mathbf{R}$  and Toeplitz (identical elements along the diagonals).

Now the wide sense stationary condition (Wiener filter condition) on the input signals  $x_n$  and  $d_n$  is removed. This means that the auto- and cross-correlations now in general become time varying. This implies that the optimal FIR filter solution will be time varying and must be tracked using the time varying performance index

$$\mathcal{E}_n = \sigma_{d_n}^2 - 2\mathbf{w}_n^T \mathbf{p}_n + \mathbf{w}_n^T \mathbf{R}_n \mathbf{w}_n \quad (12)$$

A computationally very expensive solution to this tracking problem, if the number of FIR filter coefficients is large or the sampling rate of the input signals is high, is to solve the Wiener filter at each time step  $n$ .

An alternative method for tracking the FIR filter coefficients is to use the  $\mathbf{w}$  gradient of the performance index in a steepest descent search in the following recursion still assuming that  $\mathbf{R}_n$  is Toeplitz:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \frac{1}{2} \mu \frac{\partial \mathcal{E}_n}{\partial \mathbf{w}_n}. \quad (13)$$

The gradient of the performance index with respect to the FIR filter coefficients is a vector and is determined as follows

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{w}_n} = \left( \frac{\partial \mathcal{E}_n}{\partial w_{0,n}}, \dots, \frac{\partial \mathcal{E}_n}{\partial w_{M-1,n}} \right)^T. \quad (14)$$

Now determine the gradient of the performance index

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{w}_n} = \frac{\partial}{\partial \mathbf{w}_n} \left( \sigma_{d_n}^2 - 2\mathbf{w}_n^T \mathbf{p}_n + \mathbf{w}_n^T \mathbf{R}_n \mathbf{w}_n \right). \quad (15)$$

The derivative of the variance of the desired response is of course  $\mathbf{0}$ .

$$\frac{\partial}{\partial \mathbf{w}_n} \sigma_{d_n}^2 = \mathbf{0}. \quad (16)$$

The derivative of the cross-correlation part becomes

$$\frac{\partial}{\partial \mathbf{w}_n} \mathbf{w}_n^T \mathbf{p}_n = \mathbf{p}_n. \quad (17)$$

The derivative of  $\mathbf{w}_n^T \mathbf{R}_n \mathbf{w}_n$  takes just a few more steps. Let  $r_i = \langle x_n x_{n-i} \rangle$  be the components in the autocorrelation vector

$$\mathbf{r}_i = (r_i, r_{i-1}, \dots, r_0, \dots, r_{M-1-i})^T \quad \text{for } i = 0, \dots, M-1. \quad (18)$$

Then the quadratic form can be represented, omitting the time index  $n$  on  $\mathbf{w}_n$  for convenience

$$\mathbf{w}^T \mathbf{R} \mathbf{w} = [w_0, \dots, w_{M-1}]^T \begin{bmatrix} \mathbf{r}_0^T \\ \mathbf{r}_1^T \\ \vdots \\ \mathbf{r}_{M-1}^T \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{M-1} \end{bmatrix} \quad (19)$$

$$= w_0 \mathbf{r}_0^T \mathbf{w} + w_1 \mathbf{r}_1^T \mathbf{w} + \dots + w_{M-1} \mathbf{r}_{M-1}^T \mathbf{w}. \quad (20)$$

Now the derivative becomes

$$\begin{aligned} \begin{bmatrix} \frac{\partial}{\partial w_0} \mathbf{w}^T \mathbf{R} \mathbf{w} \\ \frac{\partial}{\partial w_1} \mathbf{w}^T \mathbf{R} \mathbf{w} \\ \vdots \\ \frac{\partial}{\partial w_{M-1}} \mathbf{w}^T \mathbf{R} \mathbf{w} \end{bmatrix} &= \begin{bmatrix} \mathbf{r}_0^T \mathbf{w} + w_0 r_0 + w_1 r_1 + \dots + w_{M-1} r_{M-1} \\ \mathbf{r}_1^T \mathbf{w} + w_0 r_1 + w_1 r_0 + \dots + w_{M-1} r_{M-2} \\ \vdots \\ \mathbf{r}_{M-1}^T \mathbf{w} + w_0 r_{M-1} + w_1 r_{M-2} + \dots + w_{M-1} r_0 \end{bmatrix} \\ &= \begin{bmatrix} 2\mathbf{r}_0^T \mathbf{w} \\ 2\mathbf{r}_1^T \mathbf{w} \\ \vdots \\ 2\mathbf{r}_{M-1}^T \mathbf{w} \end{bmatrix} \\ &= 2\mathbf{R} \mathbf{w}. \end{aligned} \quad (21)$$

Finally the FIR filter coefficients gradient of the performance index becomes

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{w}_n} = -2\mathbf{p}_n + 2\mathbf{R}_n \mathbf{w}_n \quad (22)$$

leading to the steepest descent algorithm

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \frac{1}{2} \mu (-2\mathbf{p}_n + 2\mathbf{R}_n \mathbf{w}_n). \quad (23)$$

If the tracking of the filter coefficients is based directly on Eq. (23) it requires a full estimation of both the correlation matrix and the crosscorrelation vector. A rough approximation (which often turns out to be sufficient) then leads to the LMS algorithm.

The adaptive least mean square (LMS) algorithm is now obtained from Eq. (23) by using instantaneous estimates for  $\mathbf{R}_n$  and  $\mathbf{p}_n$  instead of a full estimation.

$$\hat{\mathbf{R}}_n = \mathbf{x}_n \mathbf{x}_n^T \quad (24)$$

$$\hat{\mathbf{p}}_n = d_n \mathbf{x}_n \quad (25)$$

Inserting Eq. (24) and (25) in Eq. (23) leads to the following recursive relation for updating the estimate of the FIR filter coefficients

$$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \mu \mathbf{x}_n (d_n - \mathbf{x}_n^T \hat{\mathbf{w}}_n). \quad (26)$$

The LMS algorithm can now be represented in the following three stages, leaving the discussion of the step-size parameter  $\mu$  to Subsection 2.3.

**Filter output:**

$$\hat{d}_n = \hat{\mathbf{w}}_n^T \mathbf{x}_n \quad (27)$$

**Estimation error:**

$$e_n = d_n - \hat{d}_n \quad (28)$$

**Adaptation of FIR filter coefficients:**

$$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \mu \mathbf{x}_n e_n. \quad (29)$$

## 2.2 Derivation of the NLMS FIR Filter.

In the standard LMS algorithm weight update shown in Eq. (29) the correction  $\mu \mathbf{x}_n e_n$  is applied to the FIR filter coefficients  $\hat{\mathbf{w}}_n$  to obtain the iteration  $n + 1$  step estimate of the coefficients. If  $\mathbf{x}_n$  is large this can lead to a gradient noise amplification problem. To overcome this problem the normalized least-mean-square (NLMS) algorithm is introduced by using a time varying step-size parameter by setting

$$\mu(n) = \frac{\tilde{\mu}}{\mathbf{x}_n^T \mathbf{x}_n}. \quad (30)$$

This leads to a modification of the LMS FIR filter coefficient update equation into the normalized LMS (NLMS) update

$$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \frac{\tilde{\mu}}{\mathbf{x}_n^T \mathbf{x}_n} \mathbf{x}_n e_n. \quad (31)$$

The NLMS algorithm is convergent in the mean square if the adaptation constant

$$0 < \tilde{\mu} < 2. \quad (32)$$

Furthermore the NLMS algorithm exhibits a rate of convergence that is potentially faster than that of the standard LMS algorithm for both uncorrelated and correlated input. To avoid numerical difficulties when  $\mathbf{x}_n^T \mathbf{x}_n$  is small, the coefficient update recursion is modified by the constant  $a > 0$  into

$$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \frac{\tilde{\mu}}{a + \mathbf{x}_n^T \mathbf{x}_n} \mathbf{x}_n e_n. \quad (33)$$

## 2.3 Some Properties of the LMS Algorithm.

In the following, some stability and performance properties of the LMS algorithm will be given in accordance with [4]. They are based on the mean-squared value  $E(e_n^2)$  of the estimation error. These properties are mainly derived from an analysis based on wide-sense stationary input signals  $x_n$  and  $d_n$ , using the Wiener filter solution as a reference. It turns out that two variables are useful for this analysis.

The first one is the LMS algorithm weight error vector  $\boldsymbol{\epsilon}_n$ , which is the error to the Wiener filter solution. This solution  $\mathbf{w}_o = \mathbf{R}^{-1} \mathbf{p}$  is obtained directly from the normal equations by inverting the correlation matrix  $\mathbf{R}$  of the input signal.

$$\boldsymbol{\epsilon}_n = \hat{\mathbf{w}}_n - \mathbf{w}_o. \quad (34)$$

The second variable is the estimation error produced in the Wiener filter solution

$$e_n^o = d_n - \mathbf{w}_o^T \mathbf{x}_n. \quad (35)$$

The main basis for the study of convergence behavior is now derived from the LMS recursion in Eq. (36) by subtracting  $\mathbf{w}_o$  from both sides of the recursion, and using the Wiener filter estimation error from Eq. (35)

$$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \mu \mathbf{x}_n (d_n - \hat{\mathbf{w}}_n^T \mathbf{x}_n) \quad (36)$$

$$\{\hat{\mathbf{w}}_{n+1} - \mathbf{w}_0\} = \{\hat{\mathbf{w}}_n - \mathbf{w}_0\} + \mu \mathbf{x}_n (d_n - \{\hat{\mathbf{w}}_n^T - \mathbf{w}_0^T + \mathbf{w}_0^T\} \mathbf{x}_n) \quad (37)$$

$$\boldsymbol{\epsilon}_{n+1} = \boldsymbol{\epsilon}_n + \mu \mathbf{x}_n (d_n - \boldsymbol{\epsilon}_n^T \mathbf{x}_n - \mathbf{w}_o^T \mathbf{x}_n) \quad (38)$$

$$= \boldsymbol{\epsilon}_n - \mu \mathbf{x}_n \mathbf{x}_n^T \boldsymbol{\epsilon}_n + \mu \mathbf{x}_n (d_n - \mathbf{w}_o^T \mathbf{x}_n) \quad (39)$$

$$= (\mathbf{I} - \mu \mathbf{x}_n \mathbf{x}_n^T) \boldsymbol{\epsilon}_n + \mu \mathbf{x}_n e_n^o \quad (40)$$

In Eq. (38) it is used that  $\boldsymbol{\epsilon}_n^T \mathbf{x}_n = (\boldsymbol{\epsilon}_n^T \mathbf{x}_n)^T$  and in Eq. (40) that  $\mathbf{I}$  is an  $M \times M$  identity matrix. The Eq. (40) is a stochastic difference equation in the weight-error vector  $\boldsymbol{\epsilon}_n$ .

With the purpose of carrying out an analysis of Eq. (40) the following four (restrictive) independence assumptions are used:

1. The input vectors  $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$  constitute a sequence of statistically independent vectors. Here  $\mathbf{x}_n = (x_n, \dots, x_{n-M+1})^T$ .
2. At time  $n$ , the input vector  $\mathbf{x}_n$  is statistically independent of all previous samples of the desired response  $d_0, d_1, \dots, d_{n-1}$ .
3. At time  $n$ , the desired response  $d_n$  is dependent on the corresponding input vector  $\mathbf{x}_n$ , but statistically independent of all previous samples of the desired response.
4. The input vector  $\mathbf{x}_n$  and the desired response  $d_n$  consists of mutually Gaussian-distributed random variables for all  $n$ .

The analysis of the LMS algorithm based on the above four assumptions is called independence theory.

### Excess mean-squared error:

If  $J_n = E(e_n^2)$  is the mean-squared error of the LMS algorithm at iteration  $n$ . Then by using

$$e_n = d_n - \hat{\mathbf{w}}_n^T \mathbf{x}_n \quad (41)$$

$$= d_n - \mathbf{w}_o^T \mathbf{x}_n - \boldsymbol{\epsilon}_n^T \mathbf{x}_n \quad (42)$$

$$= e_n^o - \boldsymbol{\epsilon}_n^T \mathbf{x}_n \quad (43)$$

an expression for  $J_n$  containing the minimum mean-squared error  $J_{min} = E((e_n^o)^2)$  produced by the Wiener filter is obtained as follows

$$J_n = E(e_n^2) \quad (44)$$

$$= E((e_n^o - \boldsymbol{\epsilon}_n^T \mathbf{x}_n)(e_n^o - \mathbf{x}_n^T \boldsymbol{\epsilon}_n)) \quad (45)$$

$$= J_{min} + E((\boldsymbol{\epsilon}_n^T \mathbf{x}_n)(\mathbf{x}_n^T \boldsymbol{\epsilon}_n)). \quad (46)$$

Now using the independence conditions on Eq. (46) together with  $\mathbf{R} = E(\mathbf{x}_n \mathbf{x}_n^T)$  and  $\mathbf{K}_n = E(\boldsymbol{\epsilon}_n \boldsymbol{\epsilon}_n^T)$  leads to

$$J_n = J_{min} + \text{trace}(\mathbf{R} \mathbf{K}_n) \quad (47)$$

where the *trace* of a matrix is the sum of the diagonal elements.

This shows that the estimation error of the LMS algorithm consists of two components: The

minimum mean-squared error  $J_{min}$  corresponding to the Wiener filter solution and a component depending of the transient behavior of the weight-error correlation matrix  $\mathbf{K}_n$ , which is positive definite for all  $n$ . Furthermore this shows that the LMS algorithm always produces a mean-squared error  $J_n$ , that is in excess of the minimum mean-squared error  $J_{min}$  of the Wiener filter solution. This leads to the definition of excess mean-squared error

$$J_{ex}(n) = J_n - J_{min} \quad (48)$$

$$= \text{trace}(\mathbf{R}\mathbf{K}_n). \quad (49)$$

Remembering that this takes place in the case of wide-sense stationary signals  $J_{ex}$  is the error which is added to the Wiener filter solution for obtaining the total error for the LMS algorithm.

**Step size parameter:**

The LMS algorithm is convergent in the mean-square if the step-size parameter  $\mu$  satisfies the following condition

$$0 < \mu < \frac{2}{\lambda_{max}}. \quad (50)$$

In typical applications the eigenvalues of  $\mathbf{R}$  are not available, so an approximation is needed. The *trace* of  $\mathbf{R}$  is a conservative estimate of  $\lambda_{max}$ . Going a step further by utilizing the Toeplitz structure of  $\mathbf{R}$  leads to  $\text{trace}(\mathbf{R}) = Mr_0$ , where  $r_0 = E(x_n^2)$ , and to the following boundary of the step-size parameter  $\mu$

$$0 < \mu < \frac{2}{Mr_0}. \quad (51)$$

**Misadjustment:**

The misadjustment  $\mathcal{M}$  is defined as the ratio of the steady-state value  $J_{ex}(\infty)$  of the excess mean-squared error to the minimum mean-squared error  $J_{min}$

$$\mathcal{M} = J_{ex}(\infty)/J_{min} \quad (52)$$

$$= \sum_{i=1}^M \frac{\mu\lambda_i}{2 - \mu\lambda_i} \quad (53)$$

where  $\lambda_i$  for  $i = 1, \dots, M$  are the eigenvalues of the input signal correlation matrix  $\mathbf{R} = E(\mathbf{x}_n\mathbf{x}_n^T)$ . In typical applications the eigenvalues of  $\mathbf{R}$  are not available. An approximation based on  $\mu \ll \lambda_{max}$  leads to

$$\mathcal{M} \approx \frac{\mu}{2} \sum_{i=1}^M \lambda_i \quad (54)$$

$$= \frac{\mu}{2} Mr_0. \quad (55)$$

Thus the condition of Eq. (51) both assures the convergence of the LMS algorithm in the mean-square sense, but also results in a misadjustment  $\mathcal{M}$  that is less than unity, which normally is desirable.

**Average time constant and misadjustment:**

If an average eigenvalue of the input signal correlation matrix  $\mathbf{R}$  is defined

$$\lambda_{av} = \frac{1}{M} \sum_{i=1}^M \lambda_i \quad (56)$$

then it is possible to derive the following average time constant for the LMS algorithm

$$\tau_{av} = \frac{1}{2\mu\lambda_{av}}. \quad (57)$$

Furthermore by using Eq. (56) and Eq. (57) in Eq. (54) the misadjustment  $\mathcal{M}$  can be expressed approximately

$$\mathcal{M} \approx \frac{1}{2}\mu M\lambda_{av} \quad (58)$$

$$\approx \frac{M}{4\tau_{av}}. \quad (59)$$

On the basis of Eq. (59) the following observations can be made

1. The misadjustment increases linearly with the FIR filter length  $M$  for a fixed  $\tau_{av}$ .
2. The settling time for the LMS algorithm (the time it takes for the transients to disappear) is proportional to the average time constant. It follows therefore that the misadjustment  $\mathcal{M}$  is inversely proportional to the settling time.
3. The misadjustment  $\mathcal{M}$  is directly proportional to the step size parameter  $\mu$ , whereas the average time constant  $\tau_{av}$  is inversely proportional to  $\mu$ . Therefore conflicting requirements exists in that if  $\mu$  is reduced so as to reduce the misadjustment, then the settling time of the LMS algorithm is increased. Conversely, if  $\mu$  is increased so as to reduce the settling time, then the misadjustment is increased.

## 2.4 The LMS Filter Algorithm

Main parameters and variables:

$M$ : The number of coefficients in the FIR filter.

$x_n$ : Input signal.

$E_x = \sum_{k=0}^{M-1} E(x_{n-k}^2)$

$d_n$ : Desired response.

$\mathbf{x}_n = (x_n, x_{n-1}, \dots, x_{n-M+1})^T$ : Input signal vector.

$\mathbf{w}_n = (w_{0,n}, w_{1,n}, \dots, w_{M-1,n})^T$ : FIR filter.

$0 < \mu < 2/E_x$ : Step-size parameter.

Complexity:

The number of multiplications per iteration:  $2M + 1$ .

The number of additions/subtractions per iteration:  $2M$ .

Initialization:

$\hat{\mathbf{w}}_0 = \mathbf{0}$  or use prior knowledge of  $\hat{\mathbf{w}}(0)$ .

The LMS algorithm:

*for*  $n = 1, 2, \dots$

$e_n = d_n - \hat{\mathbf{w}}_n^T \mathbf{x}_n$

$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \mu \mathbf{x}_n e_n$

*endfor*  $n$

## 2.5 The NLMS Filter Algorithm

Main parameters and variables:

$M$ : The number of coefficients in the FIR filter.

$x_n$ : Input signal.

$d_n$ : Desired response.

$\mathbf{x}_n = (x_n, x_{n-1}, \dots, x_{n-M+1})^T$ : Input signal vector.

$\mathbf{w}_n = (w_{0,n}, w_{1,n}, \dots, w_{M-1,n})^T$ : FIR filter.

$0 < \tilde{\mu} < 2$ : Adaptation constant.

$a$ : Positive constant.

Complexity:

The number of multiplications per iterations:  $2M + 4$ .

The number of divisions per iteration: 1.

The number of additions/subtractions per iteration:  $2M + 1$ .

Initialization:

$\hat{\mathbf{w}}_0 = \mathbf{0}$  or use prior knowledge of  $\hat{\mathbf{w}}(0)$ .

The *NLMS* algorithm:

*for*  $n = 1, 2, \dots$

$$e_n = d_n - \hat{\mathbf{w}}_n^T \mathbf{x}_n$$

$$\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_n + \frac{\tilde{\mu}}{a + \mathbf{x}_n^T \mathbf{x}_n} \mathbf{x}_n e_n$$

*endfor*  $n$

### 3 The Recursive Least Squares (RLS) FIR filter.

#### 3.1 Basic Concepts

In the following the method of least squares is used in a recursive algorithm for the computation of an adaptive FIR filter. The structure of the algorithm is such, that if the estimate of the FIR filter coefficients is given at time  $n - 1$ , then this estimate is used in the computation of the time  $n$  coefficients. The resulting algorithm is denoted the recursive least squares (RLS) algorithm.

A derivation of the RLS algorithm is given in the following for the real input signal  $x_n$  and the desired response  $d_n$  for  $n = 1, 2, \dots$ . It is assumed that  $x_n = 0$  for  $n < 1$  and nothing is assumed about  $x_n$  for  $n > N$ . This is called the prewindowing method.

The following outline is used in the derivation of the RLS algorithm:

- Define a cost function which is minimized for obtaining the least squares (LS) solution for each time instance  $n$ .
- Find the normal equations which relates the input signal  $x_n$  and the desired response  $d_n$  and the weighted error signal  $e_n$ .
- Form a recursive expression for updating the correlation matrix of the above normal equations from time instance  $n - 1$  to  $n$ . Form similarly a recursive expression of the time update of the weighted cross correlation between the input signal vector  $\mathbf{x}_n$  and the desired response  $d_n$ .
- To avoid the inversion of the correlation matrix of the normal equations at each time instance  $n$ , the Woodbury's identity also called the matrix inversion lemma is used. This leads to the efficient computation of the inverse correlation matrix at time  $n$  when the inverse matrix is known at time  $n - 1$  together with the input signal vector at time  $n$  denoted  $\mathbf{x}_n$ .
- Finally a set of initial conditions for the RLS algorithm is given such that numerical problems are avoided during the start of the algorithm.

#### 3.2 Derivation of the RLS FIR Filter.

Let the input signal vector be denoted

$$\mathbf{x}_n = (x_n, x_{n-1}, \dots, x_{n-M+1})^T \quad (60)$$

and the desired response  $d_n$ . The FIR filter with  $M$  coefficients is represented by

$$\mathbf{w}_n = (w_{0,n}, w_{1,n}, \dots, w_{M-1,n})^T. \quad (61)$$

The estimation error at time  $n$  is

$$e_n = d_n - \mathbf{w}_n^T \mathbf{x}_n. \quad (62)$$

If a cost function, depending of  $e_n$ , is selected such that the most recent errors are emphasized, then the commonly used exponentially weighted least squares becomes

$$\mathcal{E}_n = \sum_{i=1}^n \lambda^{n-i} (d_i - \mathbf{w}_n^T \mathbf{x}_i)^2. \quad (63)$$

Here  $0 < \lambda < 1$  and is often denoted the forgetting factor or exponential weighting factor. The effective length of the exponential weighting window is approximately  $1/(1 - \lambda)$  for  $\lambda \simeq 1$ .

The normal equation becomes

$$\Phi_n \hat{\mathbf{w}}_n = \boldsymbol{\theta}_n. \quad (64)$$

Where the  $M \times M$  correlation matrix  $\Phi_n$  (which is symmetric but not Toeplitz, and different from the time averaged correlation matrix) is determined by

$$\Phi_n = \sum_{i=1}^n \lambda^{n-i} \mathbf{x}_i \mathbf{x}_i^T \quad (65)$$

and the  $M \times 1$  cross correlation vector  $\boldsymbol{\theta}_n$  between the FIR filter input and the desired response  $d_n$  is given by

$$\boldsymbol{\theta}_n = \sum_{i=1}^n \lambda^{n-i} \mathbf{x}_i d_i. \quad (66)$$

With the purpose of obtaining recursive expressions for  $\Phi_n$  and  $\boldsymbol{\theta}_n$ , Eq. (65) can be rewritten

$$\Phi_n = \lambda \sum_{i=1}^{n-1} \lambda^{n-1-i} \mathbf{x}_i \mathbf{x}_i^T + \mathbf{x}_n \mathbf{x}_n^T \quad (67)$$

which can be rewritten into

$$\Phi_n = \lambda \Phi_{n-1} + \mathbf{x}_n \mathbf{x}_n^T. \quad (68)$$

Furthermore Eq. (66) can be rewritten into a recursive update of the cross correlation vector as follows

$$\boldsymbol{\theta}_n = \lambda \boldsymbol{\theta}_{n-1} + \mathbf{x}_n d_n. \quad (69)$$

Now the LS estimate must be computed in accordance with Eq. (64). To avoid a number of floating point addition and multiplication operations which increase in accordance with  $\mathcal{O}(M^3)$  if a standard matrix inversion algorithm is used at each time step, the Woodbury identity, also denoted the matrix inversion lemma, is applied.

**The matrix inversion lemma** takes the following form for the  $M \times M$  positive definite matrices  $\mathbf{A}$ ,  $\mathbf{B}$  and the  $N \times N$  positive definite matrix  $\mathbf{D}$

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C} \mathbf{D}^{-1} \mathbf{C}^T \quad (70)$$

where  $\mathbf{C}$  is an  $M \times N$  matrix. Then the inverse of the  $\mathbf{A}$  matrix can be written:

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B} \mathbf{C} (\mathbf{D} + \mathbf{C}^T \mathbf{B} \mathbf{C})^{-1} \mathbf{C}^T \mathbf{B}. \quad (71)$$

If it is assumed that  $\Phi_n$  is nonsingular, then the Eq. (68) can be rewritten into a computational more efficient form by using the matrix inversion lemma.

The application of the lemma is implemented directly by identifying matrices in Eq. (68) with the matrices in the lemma:

$$\mathbf{A} = \Phi_n \quad (72)$$

$$\mathbf{B}^{-1} = \lambda \Phi_{n-1} \quad (73)$$

$$\mathbf{C} = \mathbf{x}_n \quad (74)$$

$$\mathbf{D} = \mathbf{I} \quad (75)$$

where  $\mathbf{I}$  is the identity matrix. By substituting these 4 identifications into the recursive equation for the correlation matrix in Eq. (68), a recursive expression for the **inverse correlation matrix** is obtained

$$\Phi_n^{-1} = \lambda^{-1} \Phi_{n-1}^{-1} - \frac{\lambda^{-2} \Phi_{n-1}^{-1} \mathbf{x}_n \mathbf{x}_n^T \Phi_{n-1}^{-1}}{1 + \lambda^{-1} \mathbf{x}_n^T \Phi_{n-1}^{-1} \mathbf{x}_n}. \quad (76)$$

To enlighten the notation let

$$\mathbf{P}_n = \Phi_n^{-1}. \quad (77)$$

Furthermore let the gain vector  $\mathbf{k}_n$  be defined as follows

$$\mathbf{k}_n = \frac{\lambda^{-1} \mathbf{P}_{n-1} \mathbf{x}_n}{1 + \lambda^{-1} \mathbf{x}_n^T \mathbf{P}_{n-1} \mathbf{x}_n}. \quad (78)$$

Using the Equations (77) and (78) the recursive expression given in Eq. (76) can be rewritten into

$$\mathbf{P}_n = \lambda^{-1} \mathbf{P}_{n-1} - \lambda^{-1} \mathbf{k}_n \mathbf{x}_n^T \mathbf{P}_{n-1}. \quad (79)$$

The expression for the gain vector  $\mathbf{k}_n$  can be rewritten and simplified through the following steps

$$\mathbf{k}_n = \lambda^{-1} \mathbf{P}_{n-1} \mathbf{x}_n - \lambda^{-1} \mathbf{k}_n \mathbf{x}_n^T \mathbf{P}_{n-1} \mathbf{x}_n \quad (80)$$

$$= [\lambda^{-1} \mathbf{P}_{n-1} - \lambda^{-1} \mathbf{k}_n \mathbf{x}_n^T \mathbf{P}_{n-1}] \mathbf{x}_n. \quad (81)$$

Because the bracketed expression in Eq. (81) is equal to the right side of Eq. (79), the expression for  $\mathbf{k}_n$  can be simplified into

$$\mathbf{k}_n = \mathbf{P}_n \mathbf{x}_n \quad (82)$$

which immediately through the use of Eq. (77) leads to the following expression, where the gain vector is determined as the transformation of the input signal vector  $\mathbf{x}_n$  through the inverse correlation matrix  $\Phi_n$

$$\mathbf{k}_n = \Phi_n^{-1} \mathbf{x}_n. \quad (83)$$

The next step in the derivation of the RLS FIR filter algorithm is to develop a recursive equation for updating the least squares estimate of the filter coefficients  $\hat{\mathbf{w}}_n$ . Initially the Equations (64), (69) and (77) can be rewritten into

$$\hat{\mathbf{w}}_n = \Phi_n^{-1} \boldsymbol{\theta}_n \quad (84)$$

$$= \mathbf{P}_n \boldsymbol{\theta}_n \quad (85)$$

$$= \lambda \mathbf{P}_n \boldsymbol{\theta}_{n-1} + \mathbf{P}_n \mathbf{x}_n d_n. \quad (86)$$

If the Eq. (79) is substituted for the  $\mathbf{P}_n$  in the first term only of the right side of Eq. (86) the following is obtained

$$\hat{\mathbf{w}}_n = \mathbf{P}_{n-1} \boldsymbol{\theta}_{n-1} - \mathbf{k}_n \mathbf{x}_n^T \mathbf{P}_{n-1} \boldsymbol{\theta}_{n-1} + \mathbf{P}_n \mathbf{x}_n d_n \quad (87)$$

$$= \Phi_{n-1}^{-1} \boldsymbol{\theta}_{n-1} - \mathbf{k}_n \mathbf{x}_n^T \Phi_{n-1}^{-1} \boldsymbol{\theta}_{n-1} + \mathbf{P}_n \mathbf{x}_n d_n \quad (88)$$

$$= \hat{\mathbf{w}}_{n-1} - \mathbf{k}_n \mathbf{x}_n^T \hat{\mathbf{w}}_{n-1} + \mathbf{P}_n \mathbf{x}_n d_n. \quad (89)$$

Finally the recursive equation for  $\hat{\mathbf{w}}$  is obtained by using the identity  $\mathbf{P}_n \mathbf{x}_n = \mathbf{k}_n$

$$\hat{\mathbf{w}}_n = \hat{\mathbf{w}}_{n-1} + \mathbf{k}_n (d_n - \mathbf{x}_n^T \hat{\mathbf{w}}_{n-1}) \quad (90)$$

$$= \hat{\mathbf{w}}_{n-1} + \mathbf{k}_n \alpha_n \quad (91)$$

where  $\alpha_n$  is defined by

$$\alpha_n = d_n - \hat{\mathbf{w}}_{n-1}^T \mathbf{x}_n. \quad (92)$$

Here  $\alpha_n$  represents an estimate of the error, based on the old least squares estimate of the  $\hat{\mathbf{w}}_{n-1}$  computed at time  $n - 1$ . Because of that,  $\alpha_n$  is denoted the **a priori estimation error** also sometimes referred to as **the innovation**.

The **a posteriori estimation error** is given by

$$e_n = d_n - \hat{\mathbf{w}}_n^T \mathbf{x}_n \quad (93)$$

and is based on the current least squares estimate of  $\hat{\mathbf{w}}_n$ .

Now the RLS FIR algorithm is obtained from the Equations (78), (92), (91) and (79) in this order:

$$\mathbf{k}_n = \frac{\lambda^{-1} \mathbf{P}_{n-1} \mathbf{x}_n}{1 + \lambda^{-1} \mathbf{x}_n^T \mathbf{P}_{n-1} \mathbf{x}_n} \quad (94)$$

$$\alpha_n = d_n - \hat{\mathbf{w}}_{n-1}^T \mathbf{x}_n \quad (95)$$

$$\hat{\mathbf{w}}_n = \hat{\mathbf{w}}_{n-1} + \mathbf{k}_n \alpha_n \quad (96)$$

$$\mathbf{P}_n = \lambda^{-1} \mathbf{P}_{n-1} - \lambda^{-1} \mathbf{k}_n \mathbf{x}_n^T \mathbf{P}_{n-1}. \quad (97)$$

It is seen that the number of floating point arithmetic operations, also denoted *flops* cf. [2] used in this RLS algorithm increases as  $\mathcal{O}(M^2)$ , where  $M$  is the number of coefficients in the FIR filter. Notice that most digital signal processors with floating point arithmetic carry out 2 flops per machine cycle; one multiplication and one addition. This is to be compared with the number of flops  $\mathcal{O}(M)$  necessary for the LMS algorithm.

### 3.3 A Few RLS Properties.

#### 3.3.1 Recursion for the Sum of Weighted Error Squares.

Based on the least squares normal equations and the derivation of the RLS FIR filter, the following recursion [3] for the update of the minimum value of the sum of weighted error squares is obtained

$$\mathcal{E}_n^{\min} = \lambda \mathcal{E}_{n-1}^{\min} + \alpha_n e_n. \quad (98)$$

As defined above,  $\alpha_n$  is the a priori estimation error and  $e_n$  is the a posteriori estimation error.

#### 3.3.2 Rank-one Update of the Inverse Correlation Matrix.

With the purpose of later reference it is noticed that the recursive computation of the inverse correlation matrix in Eq. (76) is determined by a **rank-one update**.

A matrix is said to be of **rank**  $r$ , if the maximum number of linearly independent rows or columns is  $r$ .

If  $\mathbf{x} = (x_1, \dots, x_n)^T$  and  $\mathbf{X} = \mathbf{x}\mathbf{x}^T$  then

$$\mathbf{X} = \begin{bmatrix} x_1 \mathbf{x} & x_2 \mathbf{x} & \dots & x_n \mathbf{x} \end{bmatrix} \quad (99)$$

It is seen that any pair of columns of  $\mathbf{X}$  are linearly dependent, which means that the maximum number of linearly independent columns is 1. Thus the rank of  $\mathbf{x}\mathbf{x}^T$  is 1.

### 3.3.3 The Conversion Factor.

It is convenient to define the **conversion factor**  $\gamma_n^M$  [3] as the ratio between the a posteriori  $e_n$  and the a priori  $\alpha_n$  estimation errors.

$$\gamma_n^M = \frac{e_n}{\alpha_n} = \frac{1}{1 + \lambda^{-1} \mathbf{x}_n^T \mathbf{\Phi}_{n-1}^{-1} \mathbf{x}_n}. \quad (100)$$

The following property is used later in the discussion of the *Fast Transversal Filter* algorithm:

$$0 \leq \gamma_n^M \leq 1. \quad (101)$$

### 3.4 The RLS FIR Filter Algorithm

In summary the *RLS* algorithm obtains the structure shown in the following, which is an extension of the classical algorithm such that the symmetry of the covariance matrix  $\mathbf{P}_n$  is preserved.

Main variables:

- $M$ : The number of coefficients in the FIR filter.
- $x_n$ : Input signal.
- $d_n$ : Desired response.
- $\mathbf{x}_n = (x_n, x_{n-1}, \dots, x_{n-M+1})^T$ : Input signal vector.
- $\mathbf{w}_n = (w_{0,n}, w_{1,n}, \dots, w_{M-1,n})^T$ : FIR filter.
- $0 < \lambda < 1$ : Forgetting factor.
- $\mathbf{P}_n$ :  $M \times M$  inverse covariance matrix for the input signal vector.

Initialization:

- $\mathbf{P}_0 = \delta^{-1} \mathbf{I}$  where  $\delta$  is a small positive constant.
- $\hat{\mathbf{w}}_0 = \mathbf{0}$

Operator:

- Tri* Computes the upper or lower triangular matrix and fill the rest to preserve symmetry.

*RLS FIR* algorithm with symmetry preservation of the covariance matrix.

```

for n = 1, 2, ...
     $\mathbf{q}_n = T\mathbf{P}_{n-1}\mathbf{x}_n$ 
     $r_n = \frac{1}{\lambda + \mathbf{x}_n^T \mathbf{q}_n}$ 
     $\mathbf{k}_n = r_n \mathbf{q}_n$ 
     $\alpha_n = d_n - \hat{\mathbf{w}}_{n-1}^T \mathbf{x}_n$ 
     $\hat{\mathbf{w}}_n = \hat{\mathbf{w}}_{n-1} + \mathbf{k}_n \alpha_n$ 
     $\mathbf{P}_n = Tri\{\lambda^{-1}[\mathbf{P}_{n-1} - \mathbf{k}_n \mathbf{q}_n^T]\}$ 
endfor n

```

## 4 From the RLS to the FTF and further to the SFTF algorithm.

The notation used is primarily in accordance with [3].

## 4.1 From the RLS to the FTF Algorithm.

From Section 3 it is seen, that the number of flops required by the RLS algorithm is increasing in accordance with  $\mathcal{O}(M^2)$ . If  $M$  is large, then, because of the hardware costs, this computational load could be prohibitive, if the algorithm is used in real time systems. As an example it can be mentioned, that the number of filter coefficients necessary in some acoustical echo cancellation problems vary from several hundred to several thousands.

Thus for practical applications the computational complexity is important. This has initiated the work towards RLS algorithms with a lower complexity than  $\mathcal{O}(M^2)$ , and have resulted in a family of algorithms denoted *fast algorithms* with a computational complexity of  $\mathcal{O}(M)$  corresponding to the LMS algorithm. This has a significant impact on the possible digital signal processing applications, where the recursive least squares adaptive FIR filters can be used.

In the following, the necessary fundamental concepts for these algorithms are given. This leads to the *fast transversal filter algorithm* [1] from 1984 which is unstable and needs a rescue variable. Further developments lead to the *stable, fast transversal filter algorithm* denoted *SFTF* [8] from 1988 and [7] from 1991.

The structures of the classical RLS and the FTF, SFTF algorithms are very different. As described in Section 3 the RLS relies heavily on the matrix inversion lemma, where the inversion of the correlation matrix of the input signal  $x_n$  is carried out recursively by a rank-one update.

The approach taken in the fast transversal FIR filter FTF and also the SFTF algorithm is based on the following structure of the time shifted input signal vector  $\mathbf{x}_n^M = (x_n, \dots, x_{n-M+1})^T$ , where the superscript  $M$  denotes the dimension.

$$\begin{bmatrix} x_n \\ \mathbf{x}_{n-1}^M \end{bmatrix} \stackrel{E1}{=} \mathbf{x}_n^{M+1} \stackrel{E2}{=} \begin{bmatrix} \mathbf{x}_n^M \\ x_{n-M} \end{bmatrix}. \quad (102)$$

The first equal sign marked *E1* implies both an update of time (from  $n - 1$  to  $n$ ) and an update of the dimension (also denoted order) from  $M$  to  $M + 1$ , when transforming from  $\mathbf{x}_{n-1}^M$  to  $\mathbf{x}_n^{M+1}$  after the arrival of a new sample  $x_n$ . The second equal sign implies a reduction of the order by one (also denoted a downdating), leaving the time unchanged when transforming from  $\mathbf{x}_n^{M+1}$  to  $\mathbf{x}_n^M$  discarding the oldest sample  $x_{n-M}$ .

A set of FIR filter relations, which can be used for the derivation of the order up- and downdates and the time update according to Eq. (102), can be derived based on the general RLS FIR filter solution on the *forward linear prediction error recursive least squares filter* (FLP) and equivalently the *backward linear prediction error recursive least squares filter* (BLP).

The following Table from [3] summarizes the correspondences between the variables of the three algorithms:

<b>Variable</b>	<b>RLS</b>	<b>FLP</b>	<b>BLP</b>
Input signal vector	$\mathbf{x}_n$	$\mathbf{x}_{n-1}^M$	$\mathbf{x}_n^M$
Desired response	$d_n$	$x_n$	$x_{n-M}$
<i>FIR</i> filter coefficients	$\mathbf{w}_n$	$\mathbf{a}_n$	$\mathbf{c}_n$
A priori estimation error	$\alpha_n$	$\eta_n$	$\psi_n$
A posteriori estimation error	$e_n$	$f_n$	$b_n$
Gain vector	$\mathbf{k}_n$	$\mathbf{k}_{n-1}^M$	$\mathbf{k}_n^M$
Minimum value of sum of weighted errors squares	$\mathcal{E}_n^{min}$	$\mathcal{F}_n$	$\mathcal{B}_n$

The update relations for FLP and BLP are given below together with the relations for updating the gain vector and conversion factor. The detailed derivations of these relations can be found in [3].

**Forward linear prediction error least squares filter:**

$\mathbf{a}_n = (a_{0,n}, \dots, a_{M,n})^T$ : Forward prediction error filter.  $a_{0,n} = 1$ .

$\eta_n$ : Forward a priori prediction error.

$$\mathbf{a}_n = \mathbf{a}_{n-1} - \begin{bmatrix} 0 \\ \mathbf{k}_{n-1}^M \end{bmatrix} \eta_n$$

$$\eta_n = \mathbf{a}_{n-1}^T \mathbf{x}_n^{M+1}$$

$f_n$ : Forward a posteriori prediction error.

$\mathcal{F}_n$ : Weighted forward a posteriori prediction squared errors.

$$\mathcal{F}_n = \lambda \mathcal{F}_{n-1} + \eta_n f_n$$

**Backward linear prediction error least squares filter:**

$\mathbf{c}_n = (c_{0,n}, \dots, c_{M,n})^T$ : Backward prediction error filter.  $c_{M,n} = 1$ .

$\psi_n$ : Backward a priori prediction error.

$$\mathbf{c}_n = \mathbf{c}_{n-1} - \begin{bmatrix} \mathbf{k}_n^M \\ 0 \end{bmatrix} \psi_n$$

$$\psi_n = \mathbf{c}_{n-1}^T \mathbf{x}_n^{M+1}$$

$b_n$ : Backward a posteriori prediction error.

$\mathcal{B}_n$ : Weighted backward a posteriori prediction squared errors.

$$\mathcal{B}_n = \lambda \mathcal{B}_{n-1} + \psi_n b_n$$

**Gain vector:**

$\mathbf{k}_n^M = (k_{0,n}^M, \dots, k_{M-1,n}^M)^T$ : Gain vector.

$\mathbf{k}_n^{M+1}$ : Extended gain vector.

$$\mathbf{k}_n^{M+1} = \begin{bmatrix} 0 \\ \mathbf{k}_{n-1}^M \end{bmatrix} + (f_n / \mathcal{F}_n) \mathbf{a}_n$$

$$\mathbf{k}_n^{M+1} = \begin{bmatrix} \mathbf{k}_n^M \\ 0 \end{bmatrix} + (b_n / \mathcal{B}_n) \mathbf{c}_n$$

**Conversion factor, defining relations:**

$\gamma_n^M$ : Ratio between a posteriori and a priori estimation error.

Conversion factor for the general RLS:

$$\gamma_n^M = e_n / \alpha_n$$

Conversion factor for forward linear prediction errors least squares:

$$\gamma_{n-1}^M = f_n / \eta_n$$

Conversion factor for backward linear prediction errors least squares:

$$\gamma_n^M = b_n / \psi_n$$

**Conversion factor, update formulas:**

$$\gamma_n^{M+1} = \gamma_{n-1}^M - f_n^2 / \mathcal{F}_n$$

$$\gamma_n^{M+1} = \gamma_n^M - b_n^2 / \mathcal{B}_n$$

$$\gamma_n^{M+1} = \lambda (\mathcal{F}_{n-1} / \mathcal{F}_n) \gamma_{n-1}^M$$

$$\gamma_n^{M+1} = \lambda (\mathcal{B}_{n-1} / \mathcal{B}_n) \gamma_n^M$$

In accordance with the above, the operations of the FTF algorithm are carried out in 3 stages as follows:

- Stage 1: Order and time update the gain vector from  $\tilde{\mathbf{k}}_{n-1}^M$  to  $\tilde{\mathbf{k}}_n^{M+1}$  using  $\mathbf{a}_{n-1}$ , where  $\tilde{\mathbf{k}}_n^M = \mathbf{k}_n^M / \gamma_n^M$ .  
Time update the forward linear prediction error least squares filter from  $\mathbf{a}_{n-1}$  to  $\mathbf{a}_n$ .
- Stage 2: Reduce the order of the gain vector from  $\tilde{\mathbf{k}}_n^{M+1}$  to  $\tilde{\mathbf{k}}_n^M$ .  
Time update the backward linear prediction error least squares filter from  $\mathbf{c}_{n-1}$  to  $\mathbf{c}_n$ .
- Stage 3: Update the joint process estimator from  $\mathbf{w}_{n-1}$  to  $\mathbf{w}_n$ .

The stages are shown in the next section which defines the FTF algorithm. Further details on the derivation of this algorithm can be found in [3].

## 4.2 The FTF Algorithm

### 4.2.1 Input Signals and Internal Variables

#### Input Signals and Parameters of the FTF Algorithm:

- $x_n$ : Input signal at time instance  $n$ .
- $d_n$ : Desired response at time instance  $n$ .
- $M$ : The number of coefficients of the transversal filter.
- $\lambda$ : Weighting factor for the exponential window.
- $\mu$ : Initialization constant, cf. [1].

#### Output Signals (also given in the Joint-Process Estimator):

- $\epsilon_n$ : A posteriori estimation error.
- $\hat{\mathbf{w}}_n = (\hat{w}_{0,n}, \dots, \hat{w}_{M-1,n})^T$ : Transversal filter.

#### Internal Variables of the FTF Algorithm:

##### Input signal vectors:

- $\mathbf{x}_n^M = (x_n, \dots, x_{n-M+1})^T$ : Input signal vector.
- $\mathbf{x}_n^{M+1} = (x_n, \dots, x_{n-M})^T$ : Extended input signal vector.

##### Joint-Process Estimator:

- $\alpha_n$ : A priori estimation error.
- $\epsilon_n$ : A posteriori estimation error.
- $\hat{\mathbf{w}}_n = (\hat{w}_{0,n}, \dots, \hat{w}_{M-1,n})^T$ : Transversal filter.

### 4.2.2 Computations

#### INITIALIZATION PART:

$$\begin{aligned}
\mathbf{a}_0 &= (1, \mathbf{0}^T)^T \\
\mathcal{F}_0 &= \lambda^M \mu \\
\mathbf{c}_0 &= (\mathbf{0}^T, 1)^T \\
\mathcal{B}_0 &= \mu \\
\tilde{\mathbf{k}}_0^M &= \mathbf{0} \\
\gamma_0^M &= 1 \\
\mathbf{w}_0 &= \mathbf{0}
\end{aligned}$$

for  $n = 1, 2, \dots$

PREDICTION PART (Stage 1):

Forward a priori prediction filter:

$$\eta_n = \mathbf{a}_{n-1}^T \mathbf{x}_n^{M+1}$$

Forward a posteriori prediction error:

$$f_n = \eta_n \gamma_{n-1}^M$$

Sum of forward prediction weighted squared errors:

$$\mathcal{F}_n = \lambda \mathcal{F}_{n-1} + \eta_n f_n$$

Update the extended gain vector:

$$\tilde{\mathbf{k}}_n^{M+1} = (0, \tilde{\mathbf{k}}_{n-1}^{M,T})^T + \lambda^{-1} (\eta_n / \mathcal{F}_{n-1}) \mathbf{a}_{n-1}$$

Update the conversion factor:

$$\gamma_n^{M+1} = \lambda (\mathcal{F}_{n-1} / \mathcal{F}_n) \gamma_{n-1}^M$$

Forward prediction error filter:

$$\mathbf{a}_n = \mathbf{a}_{n-1} - f_n (0, \tilde{\mathbf{k}}_{n-1}^{M,T})^T$$

PREDICTION PART (Stage 2):

Backward a priori prediction error:

$$\psi_n = \lambda \mathcal{B}_{n-1} \tilde{k}_{M,n}^{M+1}$$

Downdate the conversion factor:

$$\gamma_n^M = (1 - \psi_n \gamma_n^{M+1} \tilde{k}_{M,n}^{M+1})^{-1} \gamma_n^{M+1}$$

Rescue variable:

$$res = (1 - \psi_n \gamma_n^{M+1} \tilde{k}_{M,n}^{M+1})$$

*if res < 0 then keep the current value of  $\hat{\mathbf{w}}_n$  and reinitialize the algorithm.*

Backward a posteriori prediction error:

$$b_n = \psi_n \gamma_n^M$$

Sum of backward prediction weighted squared errors:

$$\mathcal{B}_n = \lambda \mathcal{B}_{n-1} + b_n \psi_n$$

Downdate the order of the extended gain vector:

$$(\tilde{\mathbf{k}}_n^{M,T}, 0)^T = \tilde{\mathbf{k}}_n^{M+1} - \tilde{k}_{M,n}^{M+1} \mathbf{c}_{n-1}$$

Backward prediction error filter:

$$\mathbf{c}_n = \mathbf{c}_{n-1} + b_n (\tilde{\mathbf{k}}_n^{M,T}, 0)^T$$

JOINT PROCESS ESTIMATOR PART (Stage 3):

A priori estimation error:

$$\alpha_n = d_n - \hat{\mathbf{w}}_{n-1}^T \mathbf{x}_n^M$$

A posteriori estimation error:

$$\epsilon_n = \alpha_n \gamma_n^M$$

Transversal filter update:

$$\hat{\mathbf{w}}_n = \hat{\mathbf{w}}_{n-1} + \epsilon_n \tilde{\mathbf{k}}_n^M$$

endfor  $n$

### 4.3 From the FTF to the SFTF algorithm.

An analysis of the propagation of numerical errors of the FTF algorithm reveals instability (that's why the rescue variable is used) as already mentioned in the former section.

A solution to this problem is given in [8] and [7] and is based on the idea of introducing **computational redundancy** by computing certain quantities in two different ways. This redundancy can then be used for specific measurements of the numerical errors available. Furthermore the errors can be fed back to appropriately modify the propagation of errors.

It turns out, that the FTF algorithm, with computational complexity of  $7N$ , exhibits redundancy, and by exploiting this properly, a *stable fast transversal filter (SFTF)* algorithm with a computational complexity of  $9N$  is obtained. Thus the stabilization is carried out on the expense of a moderately increase of the complexity.

The stabilization, which is described in detail in [7], is based on computing certain variables by scalar operations, in the following description of the algorithm denoted by superscript  $s$  and by filtering operations, denoted by superscript  $f$  and finally forming combinations of these variables for error control.

## 4.4 The SFTF Algorithm.

### 4.4.1 Input Signals and Internal Variables

#### Notation:

- Variable with superscript  $f$ : Generated by a filtering operation.
- Vector with superscript  $M$  or  $M + 1$ : Dimension of vector.
- Variable with superscript  $s$ : Generated by a scalar operation.
- Variable with superscript  $T$ : Transposed vector.

#### Input Signals and Parameters of the SFTF Algorithm:

- $x_n$ : Input signal at time instance  $n$ .
- $d_n$ : Desired response at time instance  $n$ .
- $M$ : Dimension of transversal filter.
- $\lambda$ : Weighting factor for the exponential window.  $\lambda \cong 1 - 0.4/M$ , cf. [7].
- $\mu$ : Constant for the adjustment of tracking properties. Here  $\mu = 1$ .  
The tracking adjustment is not implemented.
- $K_i$  for  $i = 1, \dots, 6$ : Constants for stabilizing of the algorithm.

#### Output Signals (also given in the Joint-Process Estimator):

- $\epsilon_n$ : A posteriori estimation error.
- $\hat{\mathbf{w}}_n = (\hat{w}_{0,n}, \dots, \hat{w}_{M-1,n})^T$ : Transversal filter.

#### Complexity:

The number of *flops* per iteration:  $9N + 23$  includes 3 divisions.

#### Internal Variables of the SFTF Algorithm:

##### Input signal vectors:

- $\mathbf{x}_n^M = (x_n, \dots, x_{n-M+1})^T$ : Input signal vector.
- $\mathbf{x}_n^{M+1} = (x_n, \dots, x_{n-M})^T$ : Extended input signal vector.

##### Forward prediction error transversal filter:

- $\mathbf{a}_n = (a_{0,n}, \dots, a_{M,n})^T$ : Forward prediction error filter.  $a_{0,n} = 1$ .
- $\eta_n$ : Forward a priori prediction error.
- $f_n$ : Forward a posteriori prediction error.
- $\mathcal{F}_n$ : Weighted forward a posteriori prediction squared errors.

##### Backward prediction error transversal filter:

- $\mathbf{c}_n = (c_{0,n}, \dots, c_{M,n})^T$ : Backward prediction error filter.  $c_{M,n} = 1$ .
- $\psi_n^f$ : Backward a priori prediction error based on filtering.
- $\psi_n^s$ : Backward a priori prediction error based on scalar computation.
- $b_n$ : Backward a posteriori prediction error.
- $\mathcal{B}_n$ : Weighted backward a posteriori prediction squared errors.

##### Gain vector transversal filter:

- $\tilde{\mathbf{k}}_n^M = (\tilde{k}_{0,n}^M, \dots, \tilde{k}_{M-1,n}^M)^T$ : Gain vector.
- $\tilde{\mathbf{k}}_n^{M+1}$ : Extended gain vector.

$\tilde{k}_{0,n}^{M+1}$ : Extended gain vector component number 0.

$\tilde{k}_{M,n}^{M+1,f}$ : Extended gain vector component number  $M$ , determined by a filtering operation.

$\tilde{k}_{M,n}^{M+1,s}$ : Extended gain vector component number  $M$ , determined by a scalar operation.

$\gamma_n^{M+1}$ : Conversion factor of the extended filter.

$\gamma_n^{M+1,f}$ : Conversion factor of the extended filter, based on filtering operation.

$\gamma_n^{M+1,j}$ : Conversion factor of the extended filter, based on a convex combination of scalar and filter estimates.

$\gamma_n^{M+1,s}$ : Conversion factor of the extended filter, based on scalar operation.

### Joint-Process Estimator:

$\alpha_n$ : A priori estimation error.

$\epsilon_n$ : A posteriori estimation error.

$\hat{\mathbf{w}}_n = (\hat{w}_{0,n}, \dots, \hat{w}_{M-1,n})^T$ : Transversal filter.

## 4.4.2 Computations

INITIALIZATION PART:

$$\mathbf{a}_0 = (1, \mathbf{0}^T)^T$$

$$\mathcal{F}_0 = \lambda^M \mu$$

$$\mathbf{c}_0 = (\mathbf{0}^T, 1)^T$$

$$\mathcal{B}_0 = \mu$$

$$\tilde{\mathbf{k}}_0 = \mathbf{0}$$

$$\gamma_0 = 1$$

$$\hat{\mathbf{w}}_0 = \mathbf{0}$$

for  $n = 1, 2, \dots$

PREDICTION PART:

Forward a priori prediction filter:

$$\eta_n = \mathbf{a}_{n-1}^T \mathbf{x}_n^{M+1}$$

First element of the extended gain vector:

$$\tilde{k}_{0,n}^{M+1} = -\lambda^{-1} \mathcal{F}_{n-1}^{-1} \eta_n$$

Update the extended gain vector except the last component  $\tilde{k}_{M,n}^{M+1}$ :

$$\tilde{\mathbf{k}}_n^{M+1} = (0, \tilde{\mathbf{k}}_{n-1}^{M,T})^T + \tilde{k}_{0,n}^{M+1} \mathbf{a}_{n-1}$$

Update the inverse, extended conversion factor:

$$(\gamma_n^{M+1})^{-1} = (\gamma_{n-1}^M)^{-1} - \tilde{k}_{0,n}^{M+1} \eta_n$$

Last element in the extended gain vector (scalar):

$$\tilde{k}_{M,n}^{M+1,s} = \tilde{k}_{M-1,n-1}^M + \tilde{k}_{0,n}^{M+1} a_{M,n-1}$$

Backward a priori prediction error (filtering):

$$\psi_n^f = \mathbf{c}_{n-1}^T \mathbf{x}_n^{M+1}$$

Backward a priori prediction error (scalar):

$$\psi_n^s = -\lambda \mathcal{B}_{n-1} \tilde{k}_{M,n}^{M+1,s}$$

Three convex combinations of backward a priori prediction errors:

$$\psi_n^{(i)} = K_i \psi_n^f + (1 - K_i) \psi_n^s \quad \text{for } i = 1, 2, 5$$

Last element in the extended gain vector (filtering):

$$\tilde{k}_{M,n}^{M+1,f} = -\lambda^{-1} \mathcal{B}_{n-1}^{-1} \psi_n^f$$

Convex combination of last elements of extended gain vector:

$$\tilde{k}_{M,n}^{M+1} = K_4 \tilde{k}_{M,n}^{M+1,f} + (1 - K_4) \tilde{k}_{M,n}^{M+1,s}$$

Downdate the order of the extended gain vector:

$$(\tilde{\mathbf{k}}_n^{M,T}, 0)^T = \tilde{\mathbf{k}}_n^{M+1} - \tilde{k}_{M,n}^{M+1} \mathbf{c}_{n-1}$$

Conversion factor (scalar):

$$(\gamma_n^{M,s})^{-1} = (\gamma_n^{M+1})^{-1} + \tilde{k}_{M,n}^{M+1,s} \psi_n^{(5)}$$

Conversion factor (filter):

$$(\gamma_n^{M,f})^{-1} = 1 - \tilde{\mathbf{k}}_n^{M,T} \mathbf{x}_n^M$$

Convex combination of conversion factors:

$$(\gamma_n^{M,j})^{-1} = K_3 (\gamma_n^{M,f})^{-1} + (1 - K_3) (\gamma_n^{M,s})^{-1}$$

Forward a posteriori prediction error:

$$f_n = \eta_n \gamma_{n-1}^M$$

Forward prediction error filter:

$$\mathbf{a}_n = \mathbf{a}_{n-1} + f_n (0, \tilde{\mathbf{k}}_{n-1}^{M,T})^T$$

Sum of inverse forward prediction weighted squared errors:

$$\mathcal{F}_n^{-1} = \lambda^{-1} \mathcal{F}_{n-1}^{-1} - (\tilde{k}_{0,n}^{M+1})^2 \gamma_n^{M+1}$$

Backward a posteriori prediction error:

$$b_n^{(i)} = \psi_n^{(i)} \gamma_n^{M,s} \quad \text{for } i = 1, 2.$$

Backward prediction error filter:

$$\mathbf{c}_n = \mathbf{c}_{n-1} + b_n^{(1)} (\tilde{\mathbf{k}}_n^{M,T}, 0)^T$$

Sum of backward prediction weighted squared errors:

$$\mathcal{B}_n = \lambda \mathcal{B}_{n-1} + b_n^{(2)} \psi_n^{(2)}$$

Conversion factor from convex combination:

$$\gamma_n^M = K_6 \lambda^M \mathcal{B}_n \mathcal{F}_n^{-1} + (1 - K_6) \gamma_n^{M,j}$$

## JOINT PROCESS ESTIMATOR PART:

A priori estimation error:

$$\alpha_n = d_n + \hat{\mathbf{w}}_{n-1}^T \mathbf{x}_n^M$$

A posteriori estimation error:

$$\epsilon_n = \alpha_n \gamma_n^M$$

Transversal filter update:

$$\hat{\mathbf{w}}_n = \hat{\mathbf{w}}_{n-1} + \epsilon_n \tilde{\mathbf{k}}_n^M$$

endfor  $n$

## 5 Examples on the Application of the SFTF Algorithm.

In the following section the application of the SFTF algorithm is exemplified by four closely related cases; basic system identification of an FIR filter, adaptive line enhancement (ALE), cancellation of acoustical echo in a car cabin, and cancellation of noise interference. The examples can all be represented by the adaptive FIR filter shown in Fig. 2.

The Matlab implementation of the SFTF algorithm is listed in Section 6.1. In the following sections the Matlab code listings of each of the four application examples are also given.

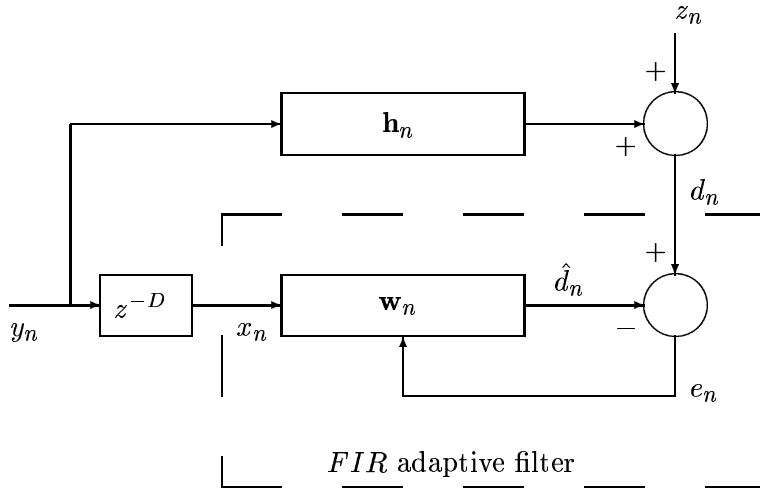


Figure 2: Adaptive *FIR* filter.

### 5.1 Basic System Identification of a FIR Filter.

#### Problem:

Identify the FIR filter  $\mathbf{h}_n$  using the input signal  $y_n$  and the desired response  $d_n$ .

#### Main parameters and signals:

$y_n$ : Input noise signal, wide sense stationary.

$D$ : If the system knowledge permits  $D$  is equal to the system delay, otherwise  $D = 0$ .

$z_n$ : Measurement noise; wide sense stationary.

$\mathbf{h}_n = \mathbf{h}$ : Constant impulse response to be identified.

#### Result:

$$\mathbf{w}_n \rightarrow \hat{\mathbf{h}} \text{ for } n \rightarrow \infty$$

#### Example:

In the identification experiment, given by the Matlab program in Section 6.2, the following values are used:

$D = 0$ .

$z_n = 0$ . Noise free measurement.

$\mathbf{h}_n = \mathbf{h}$ : Bandpass filter with  $f_{lower} = 0.30 \times f_{s/2}$  and  $f_{upper} = 0.40 \times f_{s/2}$ , where  $f_{s/2}$  is the half sampling frequency.

The number of coefficients in  $\mathbf{h}$  is set to 50, as seen in Fig. 3.

The number of coefficients in  $\mathbf{w}_n$  is 100.

$y_n$ : White gaussian signal with unity variance (power),  $R_x(0) = 1$ . A segment of the signal is seen in Fig. 4.

The resulting FIR filter is shown in Fig. 5. In Fig. 7 is shown the error  $\mathbf{h} - \hat{\mathbf{h}}$  on the final estimated filter. Finally in Fig. 6 is shown the mean value, within a window of length 100, of the squared a posteriori estimation error normalized by the first mean value obtained.

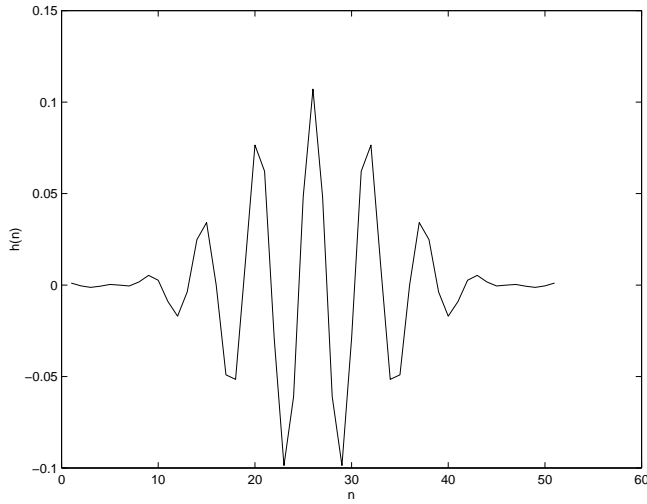


Figure 3: FIR filter to be identified

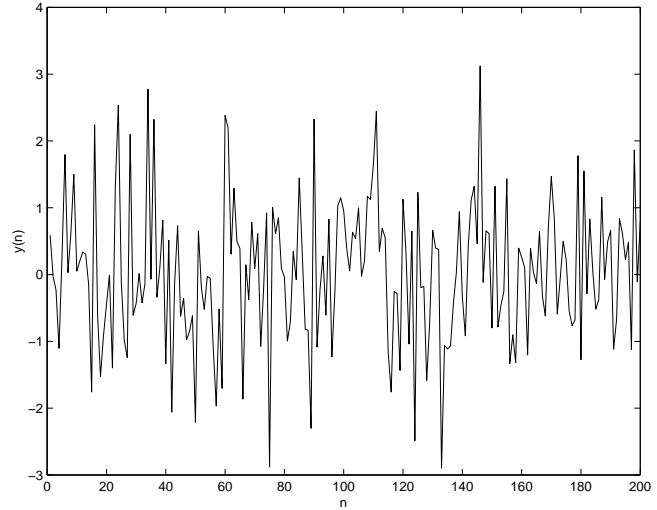


Figure 4: Gaussian input signal

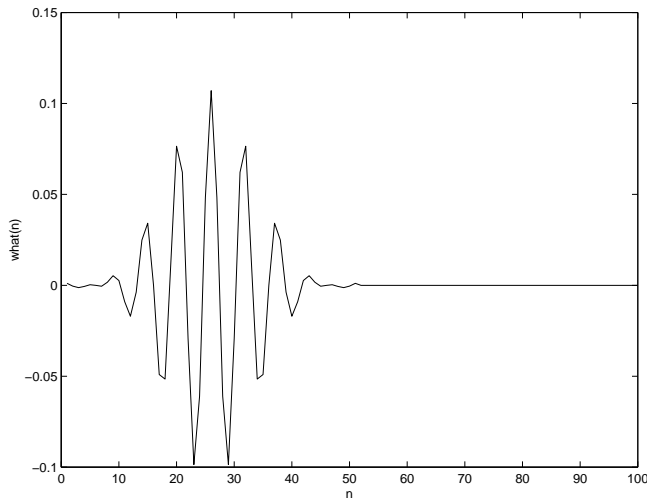


Figure 5: Estimated FIR filter

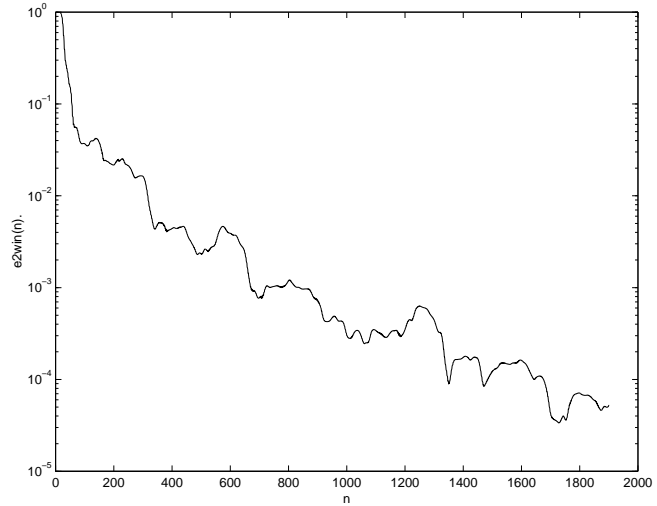


Figure 6: Convergence of estimation

## 5.2 Adaptive Line Enhancement (ALE).

### Problem:

If the input signal  $y_n = s_n + p_n$  is the sum of a noise signal  $s_n$  with a broad band power

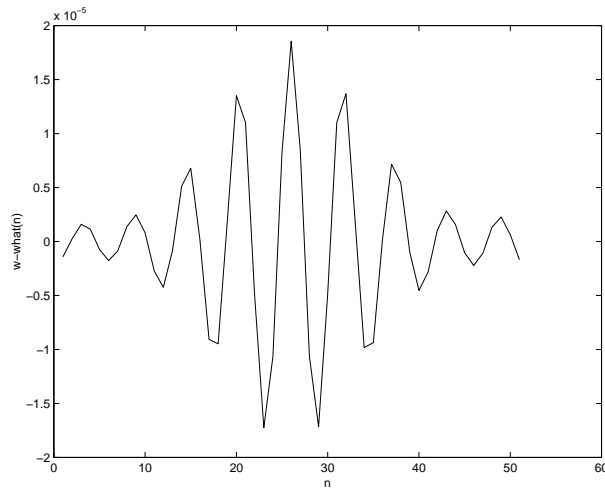


Figure 7:  $\mathbf{h} - \hat{\mathbf{h}}$

spectrum, and a harmonic signal  $p_n$  with a narrow band power spectrum, then the ALE algorithm estimate the noise  $\hat{s}_n$  and the harmonic  $\hat{p}_n$  signals, cf. [10] and [11].

### Main parameters and signals:

In the experiment on adaptive line enhancement, given by the Matlab program in Section 6.3, the following values are used:

$y_n$ : Input signal.

$D$ : Delay selected such that the autocorrelation of  $R_s(k) = 0$  for  $k > D$ .

$z_n = 0$ . Noise free measurement.

$\mathbf{h}_n = 1$ , because the adaptive FIR filter is used for linear prediction.

### Results:

$\hat{d}_n$ : Estimate of the harmonic component  $p_n$  in the input signal  $y_n$ .

$e_n$ : Estimate of the broad band component  $s_n$  in the input signal  $y_n$ .

### Example:

$y_n$ : Input signal which is the sum of a sinusoid and a white gaussian signal, with a sinusoid to noise ratio on  $-23dB$ . In Fig. 8 is shown the power spectrum of  $y_n$ . The spectrum is estimated by Welch method, cf. [6]. The frequency axis is normalized by the half sampling frequency.

$M = 700$ . The number of coefficients in the SFTF FIR filter.

In Fig. 9 is shown the power spectrum of  $\hat{d}_n$  after the convergence of the SFTF filter.

Then in Fig. 10 is shown the power spectrum of  $e_n$  after the convergence of the SFTF filter.

Finally in Fig. 11 the first 2000 samples of the  $\hat{d}_n$  signal is shown. Notice how the power of the signal is reduced during the period of convergence, in accordance with the harmonic to noise ratio.

## 5.3 Cancellation of Acoustical Echo in a Car Cabin.

### Problem:

In some car telephone systems it is necessary to have the loudspeaker and microphone open simultaneously. To avoid that the loudspeaker output is returned to the far end speaking

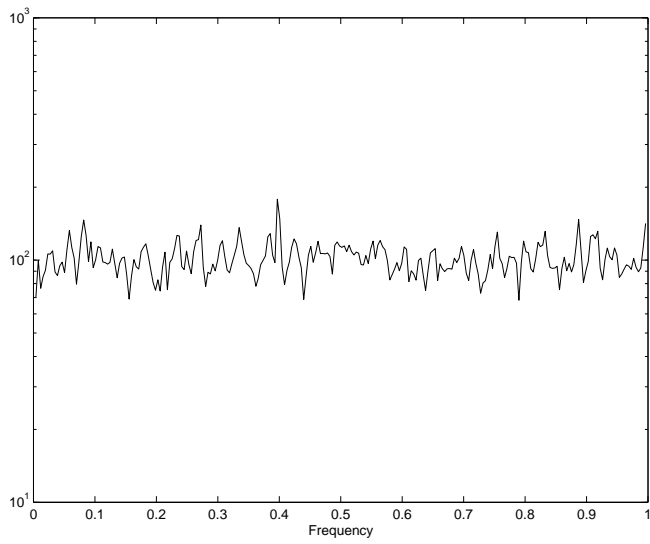


Figure 8: Power spectrum of input signal

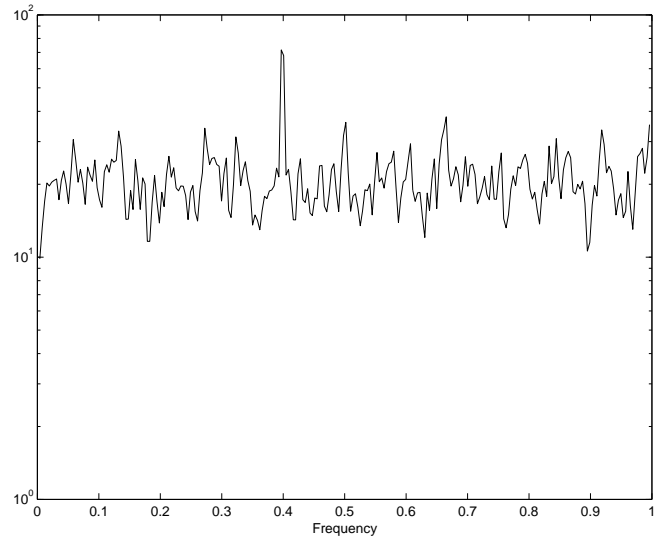


Figure 9: Power spectrum of estimated harmonic signal

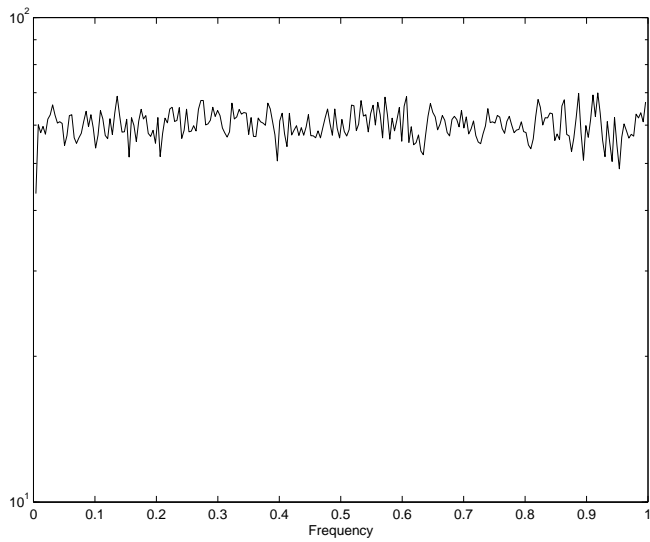


Figure 10: Power spectrum of estimated noise signal

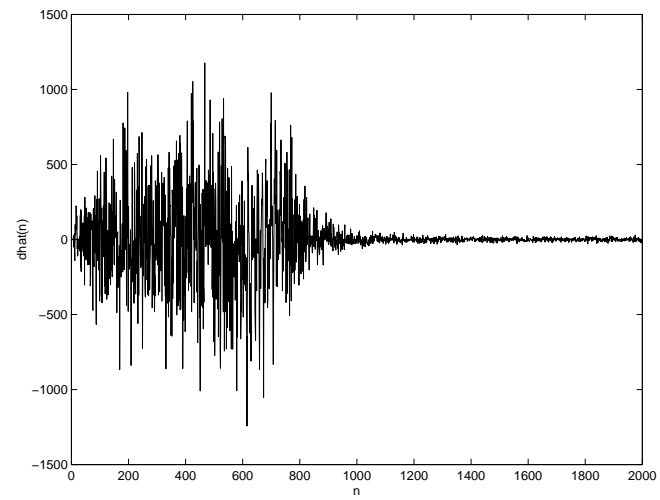


Figure 11: Estimated harmonic signal

person as an echo signal, an echo canceller is used in the near end telephone. This problem is treated in [5].

### Main parameters and signals:

$M$ : Length of car cabin impulseresponse.

$\mathbf{h}_n$ : Time varying impulse response of car cabin (to be estimated).

$D$ : Possible system delay.

$y_n$ : Input signal from far end speaking person.

$z_n$ : Input signal from the near end speaking person: Also sometimes denoted the double talk signal.

$e_n$ : Output signal with reduced echo.

### Example:

In the echo cancellation experiment, given by the Matlab program in Section 6.4, the following values are used:

$D = 0$ .

$y_n$ : Segment of speech signal sampled at  $f_s = 8 \text{ kHz}$ , shown in Fig. 12.

$\mathbf{h}^T = (h_1, \dots, h_{300})$ : Impulse response of car cabin, shown in Fig. 13.

$z_n = 0$ : Input signal from the near end speaking person.

The echo return loss enhancement (ERLE), defined as

$$ERLE_n = -10 \log(\langle e_n^2 \rangle_{win} / \langle d_n^2 \rangle_{win})$$

is shown in Fig. 14. The mean value is computed in a window around time  $n$ .

The estimated car impulse response is shown in Fig. 15. It must be noticed that this example on echo cancellation does not include any dynamics related to the change of the car cabin impulse response  $\mathbf{h}$ ; for instance from moving persons in the car cabin. Finally the error of the estimated car cabin impulse response is shown in Fig. 16

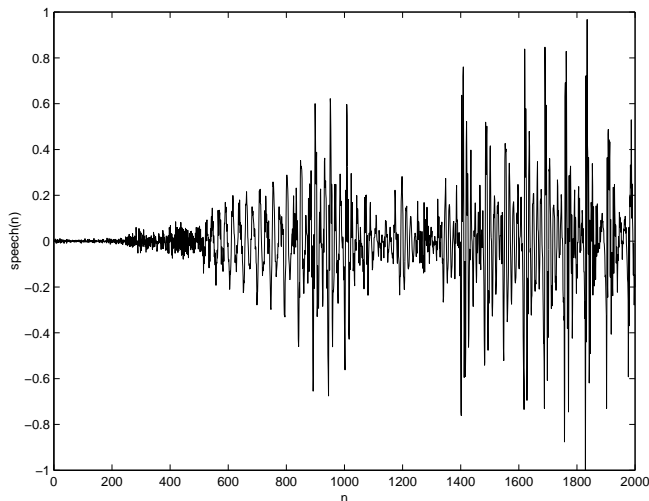


Figure 12: Speech signal segment

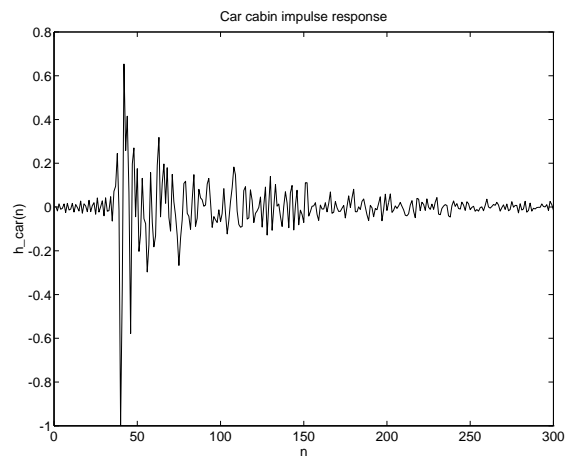


Figure 13: Car impulse response

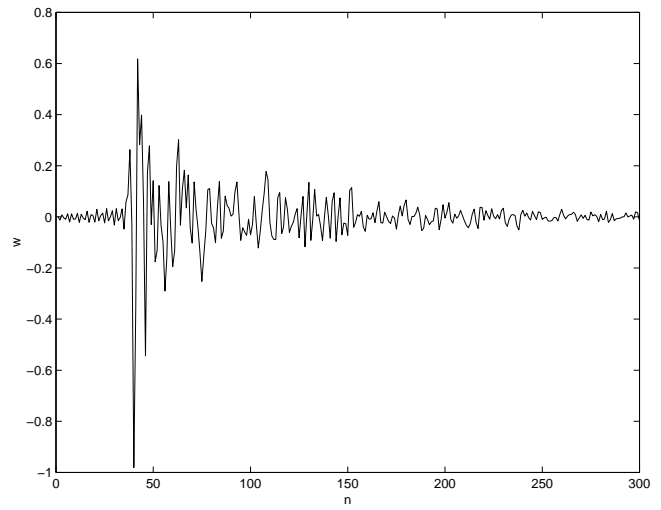
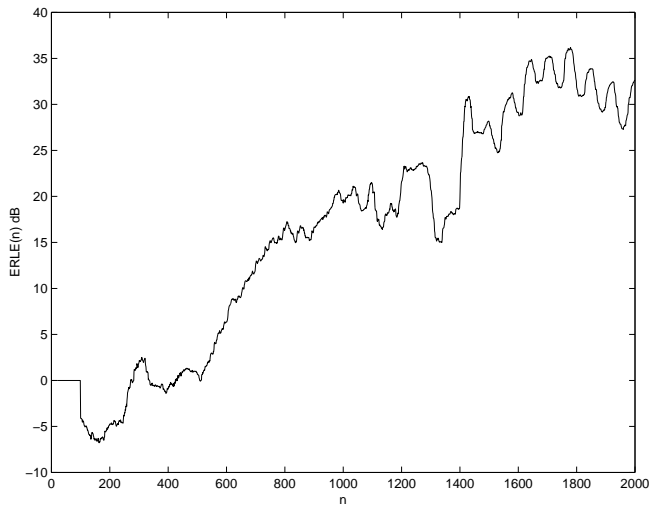


Figure 14: ERLE: Echo Return Loss Enhancement    Figure 15: Identified car cabin impulse response

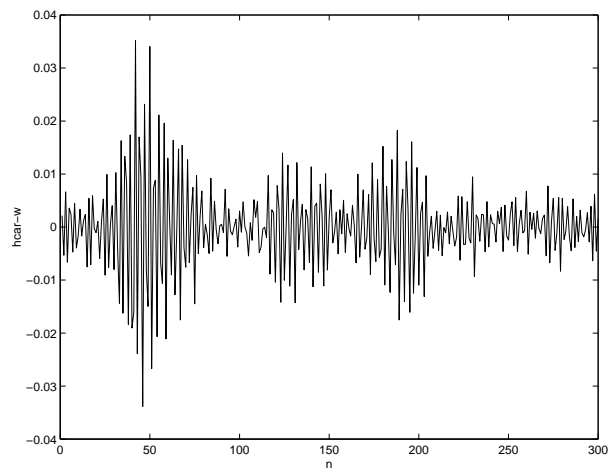


Figure 16: Error in estimated impulse response

## 5.4 Cancellation of Noise Interference.

### Problem:

If a signal is contaminated with additive noise and it is possible to separately measure another signal which is correlated with the noise, then an adaptive FIR filter might offer a possibility for noise reduction.

### Main parameters and signals:

$D = 0$  if no explicit information on system delay is available.

$d_n$ : Signal contaminated by additive noise.

$y_n$ : Separately measured noise signal correlated with the noise component in  $d_n$ .

$z_n$ : Original signal which is estimated.

### Results:

$e_n$ : Noise cancelled estimate of  $z_n$ .

### Example:

In the noise cancellation experiment, given by the Matlab program in Section 6.5, the following values are used:

$D = 0$ .

$z_n$ : Speech signal shown in Fig. 17.

$h$ : Bandpass filter which represents the correlation between the noise signal added to the speech signal and the separately measured noise signal  $y_n$ .

$d_n = y_n \otimes h_n + z_n$ : Desired response input signal. Convolution is represented by  $\otimes$ . The speech + noise signal is shown in Fig. 18.

The estimated speech signal  $e_n$  is shown in Fig. 19 and a segment of the original speech signal, the speech + noise and the estimated speech signal is shown in Fig. 20.

The signal to noise ratio before and after the cancellation are  $-9dB$  and  $13dB$  respectively.

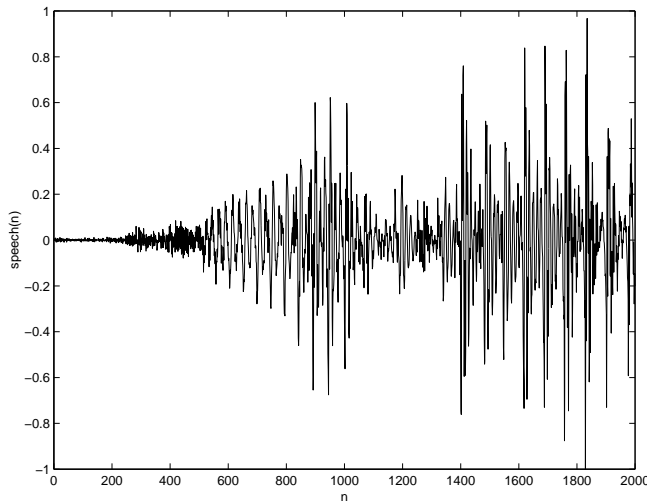


Figure 17: Noise free speech input signal

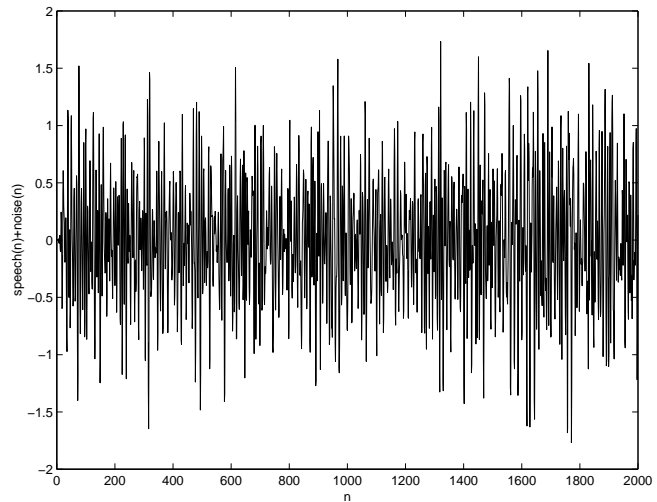


Figure 18: Speech signal + noise

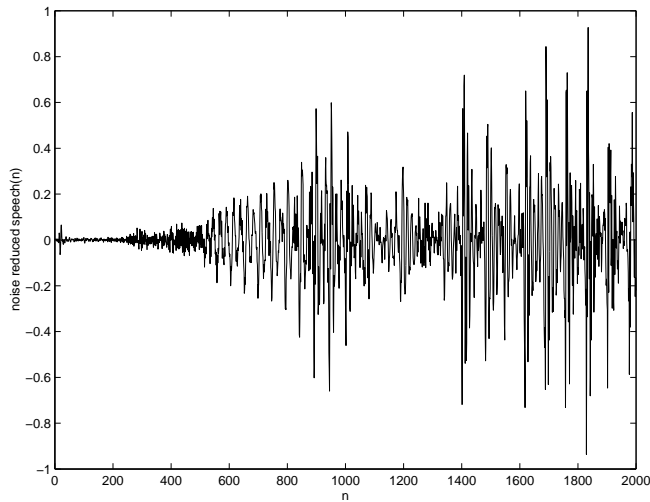


Figure 19: Noise reduced speech

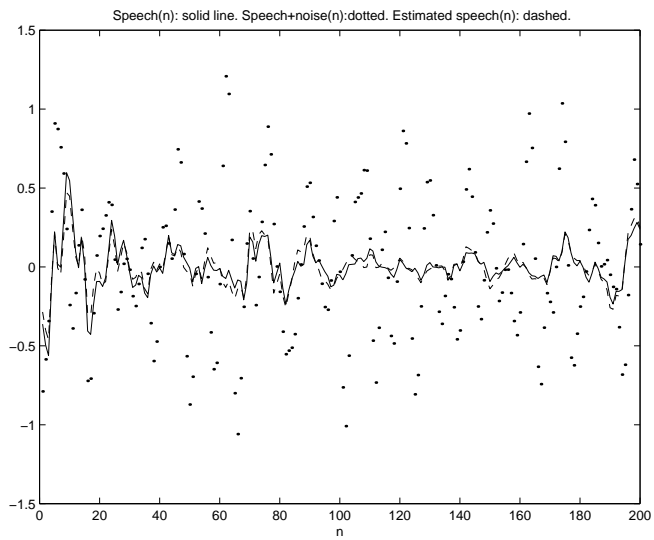


Figure 20: Segment of input speech, speech + noise and estimated speech

## 6 Matlab Code Listing.

### 6.1 SFTF Algorithm.

```

\begin{verbatim}
function [w,ep,d_hat,e2w]=sftf(x,d,M,lambda,my,K,win)
%
% The Stable Fast Transversal Filter (SFTF):
%
% Input signals and parameters:
% x: Input signal vector of length L.
% d: Desired response.
% M: The number of FIR filter coefficients.
% lambda: Exponential weighting factor. Typical value 0.99.
% my: Constant for adjustment of tracking capability. Here used only only
%     for initialization.
% K(i): Vector with stabilizing coefficients, i=1,...,6.
%     Suggested values: K=(1.5, 2.5, 1.0, 0.0, 1.0, 0.0).
% win: Target FIR filter; if available for convergence experiments.
%
% Output signals (also given in the Joint-Process Estimator):
% d_hat: Estimato of d (vector).
% ep: A posteriori prediction errors (vector).
% w: Estimated FIR coefficients (vector).
% e2w: Squared differences between estimated w and win (vector).
%     Used only for convergence experiments.
%
% Internal variables:
%
% Input signal vectors:

```

```

% xe(n:n-M+1): Input signal vector at time instance n. (Length M).
% x1=x(n:n-M): Extended input signal vector at time instance n. (Length M+1).
%
% Forward prediction error transversal filter:
% a: Forward prediction error filter.
% eta: Forward a priori prediction errors.
% f: A posteriori prediction errors.
% Fi: Inverse weighted forward a posteriori prediction errors.
%
% Backward prediction error transversal filter:
% c: Backward prediction error filter.
% psi_f: Backward a priori prediction error based on filtering.
% psi_s: Backward a priori prediction error based on scalar computation.
% b1: Backward a posteriori prediction errors, estimator number 1.
% b2: Backward a posteriori prediction errors, estimator number 2.
% B: A posteriori prediction squared errors.
%
% Gain vector transversal filter:
% k: Gain vector.
% k1: Extended gain vector.
% k1Mf: Component number M (last component) in filtering estimated extended
%       gain vector.
% k1Ms: Component number M (last component) in scalar estimated extended gain vector.
% gamma: A posteriori estimation error.
% gammai: Inverse a posteriori estimation error.
% gammai_f: Inverse a posteriori estimation error (filtering).
% gammai_s: Inverse a posteriori estimation error (scalar).
% gammai_j: Convex combination of inverse a posteriori estimation errors.
% gammai1: Inverse a posteriori estimation error of extended filter.
%
% Joint process estimator:
% alpha: A priori estimation error.
% ep: A posteriori estimation error.
%
%*****
% Allocate variables:
%
L=min(max(size(x)),max(size(d))); % Number of samples in input signals.
lambdai=1/lambda;
lambdaM=lambda^(M);
xi=zeros(M,1); % Current input signal vector.
xe=zeros(M+1,1); % Current extended input signal vector.

% Forward prediction error transversal filter:
a=[1;zeros(M,1)];
eta=zeros(L,1);
f=zeros(L,1);
Fi=0; % Inverse sum of squared errors of forward pred.

```

```

% Backward prediction error transversal filter:
c=zeros(M,1);1];
psi_f=0;
psi_s=0;
b1=0;
b2=0;
B=0;

% Gain vector transversal filter:
k=zeros(M,1);
k1=zeros(M+1,1);
k1Mf=0;
k1Ms=0;

gamma=0;
gammai=0;
gammai_f=0;
gammai_s=0;
gammai_j=0;

gammai1=0;

% Joint-process estimator:
d_hat=zeros(L,1);
alpha=zeros(L,1);
ep=zeros(L,1);
w=zeros(M,1);
e2w=zeros(L,1);
%*****
% Initialize internal variables:
%
% a: as allocated.
% c: as allocated.
% k: as allocated.
% w: as allocated.
B=my;
Fi=1/(lambdaM*my);
gamma=my;
gammai=1;
%*****
% Main loop of SFTF:
%
for n=1:L

    % FORM THE INPUT VECTORS:

    xe=[x(n);xe(1:M)];           % Extended input signal vector.

```

```

xi=xe(1:M); % Input signal vector.

% PREDICTION PART:

eta(n)=a'*xe; % Forward a priori prediction filter.

k1(1)=-lambdai*Fi*eta(n); % First element of extended gain vector.
k1=[0;k]+k1(1)*a; % Update the extended gain vector except
% the last component.

gammai1=gammai-k1(1)*eta(n); % Inv. a posteriori estimation error.

k1Ms=k(M)+k1(1)*a(M+1); % Last element in the extended gain vector (scalar).

psi_f=c'*xe; % Backward a priori prediction error (filtering).
psi_s=-lambda*B*k1Ms; % Backward a priori prediction error (scalar).
psi_1=K(1)*psi_f+(1-K(1))*psi_s; % Convex combination number 1.
psi_2=K(2)*psi_f+(1-K(2))*psi_s; % Convex combination number 2.
psi_5=K(5)*psi_f+(1-K(5))*psi_s; % Convex combination number 3.

k1Mf=-lambdai*B^(-1)*psi_f; % Last element in the extended gain vector
% (filtering).
k1(M+1)=K(4)*k1Mf+(1-K(4))*k1Ms; % Convex combination of last elements.
kold=k;
k=k1(1:M)-k1(M+1)*c(1:M); % Downdate the order of the extended gain
% vector.

gammai_s=gammai1+k1Ms*psi_5; % Inverse estimation error (scalar).
gammai_f=1-k'*xi; % Inverse estimation error (filtering).
gammai_j=K(3)*gammai_f+(1-K(3))*gammai_s; % Convex combination.

f(n)=eta(n)*gamma; % Forward a posteriori prediction error.
a=a+f(n)*[0;kold]; % Forward prediction error filter.
Fi=lambdai*Fi-k1(1)^2*gammai1^(-1); % Sum of forward prediction weighted squared
% errors.

b1=psi_1*gammai_s^(-1); % Backward a posteriori prediction error (1).
b2=psi_2*gammai_s^(-1); % Backward a posteriori prediction error (2).
c=c+b1*[k;0]; % Backward prediction error filter.
B=lambda*B+b2*psi_2; % Backward prediction weighted squared errors.

gamma=K(6)*lambdaM*B*Fi+(1-K(6))*gammai_j^(-1); % Estimation error.
gammai=1/gamma; % Update.

% JOINT PROCESS ESTIMATOR PART:

d_hat(n)=-w'*xi;
alpha(n)=d(n)-d_hat(n); % A priori estimation part.

```

```

    ep(n)=alpha(n)*gamma;           % A Posteriori estimation part.
    w=w+ep(n)*k;                   % Transversal filter update.
    %
    ew=-w-win;
    e2w(n)=ew'*ew;                 % Store squared errors of w.
end
w=-w;                              % Change sign convention of filter.
\end{verbatim}

```

## 6.2 Basic System Identification of a FIR Filter.

```

% Example: Basic system identification of FIR filter.
% *****
%
% Use the Matlab Signal Processing Toolbox fir1 function for the design of
% an FIR bandpass filter. The design is based on a Hamming window.
% Ref. Alan V. Oppenheim, Ronald Schaffer. "Digital Signal processing".
%   Prentice-Hall, 1975.
%
% Arguments of the fir1 filter design function:
%   M: Order of the filter.
%   fn=(f1,f2) where 0 < f1 < f2 < 1 are the cut-off frequencies of the
%       band-pass filter. Here 1 corresponds to the half sampling
%       frequency Fs.
%
Mf=50;                               % Order of FIR filter to be identified.
fn=[0.300, 0.40];                   % Define band-pass frequencies.
h=fir1(Mf,fn);                       % Length M+1 vector with FIR filter coefficients.
%
%
L=2000;                               % Length of input signal.
seed=107;                             % Seed for random generator (optional).
randn('seed',seed);
y=randn(L,1);                         % Input signal: Zero mean gaussian white noise
%                                     % with variance 1.
d=filter(h,1,y);                     % Generate the desired response from FIR filter.
%
% Initialize SFTF algorithm:
%
M=100;                                 % The number of coefficients in the SFTF FIR filter.
w1=zeros(M,1);                       % Not used in this application.
lambda=0.999;
my=1.0;
K=[1.5 2.5 1.0 0.0 1.0 0.0];        % Stabilization constants.
[w_hat,e,e2w]=sftf(y,d,M,lambda,my,K,w1);
%
% w_hat: Estimated FIR filter coefficients.
% e: Vector with a posteriori prediction errors.
%

```

```

e2win=meanwin(e.*e,100);      % Mean value with window length 100.
e2win=e2win/e2win(1);        % Normalize.

plot(h,'-')
xlabel('n')
ylabel('h(n)')
print -deps c:\matlab\work\04362\note10_2\firw.eps % Generate eps file.

figure                        % Change to the next figure.

```

### 6.3 Adaptive Line Enhancement (ALE).

```

% Example: Adaptive line enhancement (ALE).
%*****
%
M=700;                        % The number of coefficients in the SFTF FIR filter.
%
%
L=20480;                      % Length of input signal.
y=zeros(L,1);
d=zeros(L,1);
seed=107;                     % Seed for normal random generator (optional).
A=1;
f0=0.2;
SNR=0.005;
y=model7(y,seed,A,f0,SNR); %
%ymax=max(y);
%y=1/ymax*y;
d=y;
y=[0;y(1:L-1)];              % Create a new x signal delayed 1 time step.
%
%
lambda=0.9995;
my=1.0;
K=[1.5 2.5 1.0 0.0 1.0 0.0];
w=zeros(M,1);
[w_hat,e,d_hat,e2w]=sftf(y,d,M,lambda,my,K,w);
%
% w_hat: Estimated FIR filter coefficients.
% e: A posteriori prediction errors.
% d_hat: Estimate of d.
% e2w: Squared errors of differences between w_hat and w.
%
Py=spectrum(y(L-14*1024+1:L),512,256); % Power spectrum of the
spplot(Py);                    % harmonic signal + noise.
print -deps c:\matlab\work\04362\note10_2\ale1.eps;

```

## 6.4 Cancellation of Acoustical Echo in a Car cabin.

```
% Example: Cancellation of acoustical echo in a car cabin.
%*****
%
load speech                % Load speech signal.
load h_car                 % Load car cabin impulse response.
%
d=filter(h_car,1,speech);
%
%
M=300;                    % The number of FIR filter coefficients in canceller.
lambda=0.999;
my=1.0;
K=[1.5 2.5 1.0 0.0 1.0 0.0];
[w,e,d_hat,e2w]=sftf(speech,d,M,lambda,my,K,h_car);
%
% w: Estimated impulse response of car.
% e: Vector with a posteriori prediction errors.
% d_hat: Vector with estimate of car cabin signal.
% e2w: Vector with relative squared estimation errors of w.
%
% Estimate the echo return loss enhancement: ERLE.
%
ERLE=zeros(size(e));
win=100;
for i=win:length(e);
    ERLE(i)=-10*log10((e(i-win+1:i)'*e(i-win+1:i))/...
        (speech(i-win+1:i)'*speech(i-win+1:i)));
end
%
%
plot(speech,'-')
xlabel('n')
ylabel('speech(n)')
print -deps c:\matlab\work\04362\note10_2\speech.eps
```

## 6.5 Cancellation of Noise Interference.

```
% Example: Cancellation of noise interference.
% *****
% Use the Matlab Signal Processing Toolbox fir1 function for the design of
% a FIR bandpass filter. The design is based on a Hamming window.
% Use the bandpass filter for adding noise to a speech signal.
% Remove the noise by noise cancellation by the SFTF algorithm.
%
% Arguments of the fir1 filter design function:
% M: Order of the filter.
% fn=(f1,f2) where  $0 < f1 < f2 < 1$  are the cut-off frequencies of the
```

```

%      band-pass filter. Here 1 corresponds to the half sampling
%      frequency fs.
%
Mf=30;           % Order of FIR filter.
fn=[0.1,0.4];   % Define band-pass frequencies.
h=fir1(Mf,fn);  % Length Mf+1 vector with FIR filter coefficients.
%
%
load speech;    % Load a segment of speech signal.
%
L=length(speech); % Length of input signal.
seed=107;      % Seed for normal random generator (optional).
randn('seed',seed);
y=randn(L,1);  % Input noise signal.
dn=filter(h,1,y); % Additive noise signal.
d=dn+speech;   % Desired response as speech + noise.
%
M=50;          % The number of coefficients in the adaptive FIR filter.
w=zeros(M,1);
lambda=0.999;
my=1.0;
K=[1.5 2.5 1.0 0.0 1.0 0.0];
[w_hat,e,d_hat,e2w]=sftf(y,d,M,lambda,my,K,w);
%
% w_hat: Estimated FIR filter coefficients.
% e: Vector with a posteriori prediction errors; estimate of speech signal.
% d_hat: Estimate of the noise component of the speech signal.
% e2w: Squared errors of w.
%
SNR_in=10*log10((speech'*speech)/(dn'*dn));
es=speech-e;   % Difference between noise free speech and noise
               % reduced speech.
SNR_out=10*log10((speech'*speech)/(es'*es));
%
% Plot original speech, speech + noise, noise reduced speech,
%      and speech estimation error.
%
plot(speech,'-')
xlabel('n')
ylabel('speech(n)')
print -deps c:\matlab\work\04362\note10_2\noise1.eps

```

## References

- [1] John M. Cioffi and Thomas Kailath. Fast, recursive-least-squares transversal filters for adaptive filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(6):–, April 1984.
- [2] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The John Hopkins University Press, 2 edition, 1989.
- [3] Simon Haykin. *Adaptive Filter Theory*. Prentice-Hall International, 2 edition, 1991.
- [4] Simon Haykin. *Adaptive Filter Theory*. Prentice-Hall International, 3 edition, 1996.
- [5] Søren Holdt Jensen. *Algorithms for Echo Cancellation and Noise Reduction in Digital Hands-Free Mobile Telephony*. PhD thesis, Electronics Institute, Tech. Univ. of Denmark, 1994.
- [6] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing*. Prentice-Hall International, 3 edition, 1996.
- [7] Dirk M. Slock and Thomas Kailath. Numerically stable fast transversal filters for recursive least squares adaptive filtering. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 39(1):92–114, January 1991.
- [8] D.T.M. Slock and Thomas Kailath. Numerically stable fast recursive least squares transversal filters. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1373–1376, April 1988.
- [9] B. Widrow and Jr. M.E. Hoff. Adaptive switching circuits. In *IRE WESCON Conv. REC.*, volume 4, pages 96–104, 1960.
- [10] Bernard Widrow and Samuel D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, 1985.
- [11] James R. Zeidler. Performance analysis of lms adaptive prediction filters. *Proceedings of IEEE*, 78(12):–, December 1990.