Interpretations of an Orthogonal Projection (OP) algorithm for Adaptive Digital Signal Processing

G.P.M Egelmeers

Master's degree thesis

Supervisor : prof.dr.ir. W.M.G. van Bokhoven
Coach : ir. P.C.W. Sommen
Id. number : ITE 253216
Period : January 1991 - August 1991

Eindhoven University of Technology (TUE)
Department of Electrical Engineering (E/ITE)
Electronic Circuit Design Group (EEB)

Eindhoven, august 1991

The Eindhoven University of Technology accepts no responsibility for the contents of theses and reports made by students.
Abstract

In Adaptive Digital Signal Processing discrete-time adaptive filters of a transversal structure with \( N \) variable coefficients (taps) are commonly used. An algorithm to update the coefficients is the Normalized Least Mean Square (NLMS) algorithm. This algorithm has low complexity, but the convergence characteristics of a filter using this algorithm are influenced by the statistical properties of the input-signal. The Recursive Least Squares (RLS) algorithm decorrelates the input signal with help of the \( N \times N \) inverse of the autocorrelation matrix of the input-signal, but its complexity can grow rather hudge depending on the number of taps \( N \) of the adaptive filter. The (Block) Orthogonal Projection ((B)OP) algorithm decouples the dimension \( L \) of the autocorrelation-matrix from the number of taps \( N \) of the adaptive filter. The implementation complexity can be reduced by using the same recursive techniques as with the RLS algorithm.

In order to gain insight into the convergence behaviour of the (B)OP algorithm, the echo-canceller structure, as an example of a general signal estimator, is simulated in the SPOX environment. Verification of the simulation results by theoretical analysis is performed by interpreting the BOP algorithm in diverse fashions. Singular Value Decomposition simplifies the analysis considerably. SVD, Gram-Schmidt orthogonalisation and the frequency domain transformation can all be described with a general orthogonal transform.

The frequency domain interpretation considers the decorrelation of the BOP algorithm as a normalization of the power density spectrum (pds) of the input-signal over \( L \) subbands. If the pds is divided into subbands where the pds is considered flat, then for every subband a corresponding LMS convergence curve can be calculated. Adding these curves multilieated by the correct weights, the convergence curves of the (B)OP algorithm are obtained. The pds of the echo-path impulse response does influence the weights of the curves. A second method, that derives these convergence curves, uses an orthogonal transform to yield a (partially) decorrelated pseudo-pds for the input-signal.

In the future the introduced BOP-interpretations can possibly be evoluted to derive an optimal value for the dimension \( L \) of the decorrelation matrix depending on the input-signal statistical properties and the echo-path impulse response.
Table of contents

1 Introduction

2 Derivation of OP algorithm
  2.1 Introduction
  2.2 The NLMS algorithm
  2.3 The (L-step D)OP algorithm
  2.4 The B(N)LMS algorithm
  2.5 Points of research
  2.6 Further outline of the report

3 Simulation with SPOX
  3.1 Introduction
  3.2 The BOP program
  3.3 Results using the BOP program

4 Efficient implementation
  4.1 Exact implementation
  4.2 Approximations to simplify the implementation

5 Interpretations of BOP
  5.1 Introduction
  5.2 Analysis of BOP algorithm
  5.3 Interpretation with SVD
    5.3.1 The Singular-Value Decomposition (SVD) theorem
    5.3.2 Properties of the SVD
    5.3.3 SVD to analyse BOP
  5.4 Interpretation Fourier transformations
  5.5 Gram-Schmidt orthogonalisation
  5.6 General orthogonal transformation
Chapter 1

Introduction

A part of the Electronic Circuit Design Group (EEB) of the Electrical Engineering Faculty (E/ITE) of the Eindhoven University of Technology (TUE) is doing research in the field of (adaptive) digital signal processing.

A commonly used algorithm in adaptive digital signal processing is the Least Mean Square (LMS) algorithm. This algorithm has low complexity, but the convergence characteristics of a filter using this algorithm are influenced by the statistical properties (such as (auto)correlation) of the input signal. In order to remove this dependency decorrelation can be applied by multiplying the update part of the adaption algorithm with the inverse of the autocorrelation matrix $R$ of the input signal, yielding the so called Recursive Least Squares (RLS) algorithm.

One of the main disadvantages of RLS algorithms is the often hudge complexity caused by the fact that every iteration step needs a (direct or indirect) inversion of a $N \times N$ autocorrelation matrix $R$, where $N$ is the length of the adaptive filter (the number of taps). However, the relevant dimension of the autocorrelation matrix $R$ is determined by the statistical properties of the input signal and is not necessarily equal to $N$.

Using the Orthogonal Projection (OP) algorithm the dimension $L$ of the matrix $R$ can be chosen independent of the number of taps $N$ while the matrix $R$ can still decorrelate the input signal. The dimension $L$ of the autocorrelation matrix $R$ can therefore be derived from the statistical properties of the input signal.
Chapter 1: Introduction

The goal of this report is to gain insight into the behaviour of the Orthogonal Projection algorithm, especially into the effects of the dimension $L$ of the autocorrelation matrix $R$ on that (convergence) behaviour, by interpreting the algorithm in some different ways.

First the OP algorithm for adaptive filters will be derived with help of the echo canceller structure as an example of a general signal estimator. After that a simulation program will be developed to simulate such a filter with help of SPOX (see appendix A). Following that, diverse interpretations of the OP algorithm with some (theoretical) performance analysis of that algorithm are introduced. Last a simple method is presented to construct the convergence curves for the algorithm.
Chapter 2

Derivation of OP algorithm

2.1 Introduction

A signal estimator is a well known structure in many communication applications. The echo canceller structure (figure 2.1) is an example of such a signal estimator. The adaptive filter has to eliminate the leakage (the echo path) from the residual signal. The echo path impulse response $h$ is assumed to have length of at most $N$, the number of taps of the adaptive transversal filter $w$. Filter $h$ can be realised by a so called FIR (Finite Impulse Response) transversal filter, just like the adaptive filter $w$, see figure 3.1 of chapter three. The input signal $x[k]$ produces via this echo path an echo signal $e[k]$. The near-end white noise signal $s[k]$ is added to this echo $e[k]$ yielding $e'[k]$. The residual signal $r[k] = e'[k] - s[k]$ is supposed to be in average equal to $s[k]$ in steady state. For simplicity reasons $s[k]$ is assumed to be equal to zero in the first instance.

![Figure 2.1: Basic echo canceller structure](image)
Chapter 2: Derivation of OP algorithm

The inner product $< \mathbf{x}(k), \mathbf{h} >$ of two $N$ dimensional vectors will be defined as the next convolution sum:

$$
< \mathbf{x}(k), \mathbf{h} > = \sum_{i=0}^{N-1} x[k-i] \cdot h_i
$$

$$
= \mathbf{x}^T[k] \cdot \mathbf{h}
$$

(2.1)

with the vectors:

$$
\mathbf{x}[k] = (x[k], x[k-1], ..., x[k-N+1])^T
$$

(2.2)

$$
\mathbf{h} = (h_0, h_1, ..., h_N)^T
$$

(2.3)

Two vectors are orthogonal if their inner product is zero. The norm (length) $||\mathbf{x}[k]||$ of a vector $\mathbf{x}[k]$ is defined as the square root of the inner product of that vector with itself. With these definitions a geometric interpretation of the echo canceller problem can be derived that will lead to the well known Normalized Least Mean Square (NLMS) algorithm.
2.2 The NLMS algorithm

The update procedure of the NLMS algorithm as given in figure 2.2 performs a projection on the vector \( x[k] \) of the difference vector \( d[k] = h - w[k] \) with \( w[k] = (w_0[k], w_1[k], \ldots, w_N[k])^T \). In other words this means a decomposition of \( d[k] \) as:

\[
d[k] = d^\perp[k] + d^\parallel[k]
\]  \hspace{1cm} (2.4)

with \( d^\perp[k] \) orthogonal and \( d^\parallel[k] \) parallel to \( x[k] \). This implies:

\[
< x[k], d^\perp[k] >= 0
\]  \hspace{1cm} (2.5)

\[
d^\parallel[k] = c \cdot x[k]
\]  \hspace{1cm} (2.6)

where \( c \) is some scalar. As \( s[k] \) is assumed to be zero, figure 2.1 induces for the residual signal \( r[k] \):

\[
r[k] = < x[k], d[k] >
\]  
\[
= x^T[k] \cdot d^\parallel[k]
\]  
\[
= c \cdot < x[k], x[k] >
\]  
\[
= c \cdot \| x[k] \|^2
\]  \hspace{1cm} (2.7)

With this result the scalar \( c \) can be calculated, which yields for the parallel component \( d^\parallel[k] \) of \( d[k] \):

\[
d^\parallel[k] = c \cdot x[k]
\]  
\[
= r[k] \cdot x[k]/\| x[k] \|^2
\]  \hspace{1cm} (2.8)

Figure 2.2: Geometric interpretation of the NLMS algorithm
The update algorithm is supposed to both reduce the length of $d[k]$ and rotate $d[k]$ in such a way that it becomes "more orthogonal" to $x[k]$. This can be achieved by subtracting a small part of the vector $d[k]$ from the vector $d[k]$ as is shown in figure 2.2:

$$d[k+1] = d[k] - 2 \cdot \alpha \cdot d[k]$$

$$= d[k] - 2 \cdot \alpha \cdot x[k] \cdot r[k]/\|x[k]\|^2 \quad (2.9)$$

Remembering that the difference vector $d[k]$ can be written as $h - w[k]$, this leads to the NLMS algorithm:

$$w[k+1] = w[k] + 2 \cdot \alpha \cdot x[k] \cdot r[k]/\|x[k]\|^2 \quad (2.10)$$

Literature [ALE86], [SOM90a] has shown that the convergence behaviour of the NLMS algorithm is influenced by the statistical properties of the input signal $x[k]$. If $x[k]$ is a white noise process (has a flat spectrum) then its geometric interpretation leads to $x[k]$ and $x[k-i]$ being orthogonal to each other (for all $i \neq 0$). For a non white process $x[k]$ in general the vectors $x[k]$ and $x[k-i]$ are not orthogonal. This implies that a new update in the NLMS algorithm is in general not orthogonal to the previous update, so the new update can counteract or fortify the previous one. Therefore convergence behaviour is influenced by the statistical properties of the input signal.

If the signal $s[k]$ is not assumed to be zero, but equal to a random stationary white noise signal with variance $\sigma^2$, the algorithm cannot make the residual signal $r[k]$ infinitely small. A final misadjustment depending on $\sigma^2$ will be the result ([HAY86], [ALE86], [SOM90a], [HAY91]). A short summary of the convergence analysis of the NLMS algorithm with a white noise process as input signal follows below.

The following equation defines a output $\delta[k]$ describing the dynamical behaviour of a filter using the NLMS algorithm:

$$\delta[k] = E((e[k] - e[k])^2) \quad (2.11)$$

The final misadjustment can now be introduced:

$$\delta[\omega] = \lim_{k \to \infty} \delta[k] = \sigma^2 \cdot N/(1 - \sigma^2 \cdot \eta) = N \cdot \sigma^2 \quad (for \ \sigma \ll 1/\eta) \quad (2.12)$$
Chapter 2: Derivation of OP algorithm

in which:

\[ N = N - 1 + K \]  \hspace{1cm} (2.13)
\[ K = E(x'(k))/E(x^2(k)) \]  \hspace{1cm} (2.14)

The initial speed of convergence \( \tau_{\infty} \), defined as the number of samples to reduce the initial value of \( \delta(k) \) by 20 dB, is given in eq. (2.15):

\[ \tau_{\infty} = \frac{-2}{10 \log(1 - 4 \cdot \alpha + 4 \cdot \alpha^2 \cdot \Omega)} = \frac{1.15}{\alpha} \text{ (for } \alpha << 1/\Omega) \]  \hspace{1cm} (2.15)

If normalization is left out, eq. (2.10) must be changed, resulting in the LMS algorithm:

\[ w[k+1] = w[k] + 2 \cdot a' \cdot x[k] \cdot r[k] \]  \hspace{1cm} (2.16)

By filling in \( \alpha = a' \cdot \sigma^2 \text{ with } \sigma^2 = E(x^2[k]) \) in eq. (2.12) till (2.15) the convergence analysis results for the LMS algorithm are obtained that will be used in chapters six and seven.
2.3 The (L-step B)OP algorithm

To diminish the dependency of the convergence behaviour of the algorithm on the statistical properties of the input signal, the geometric approach described in the previous section can be extended to a projection on a L-dimensional hyperplane as is proposed in [FUR87], [OZE84] and [SOM90b] instead of on a line. As complexity can grow rather hudge using this method as can be seen later on, the "block" approach will be introduced, that uses L new samples to produce only one update, the so called Block Orthogonal Projection (BOP) algorithm. A block index m denoting the mth block as introduced below will be needed to describe this algorithm.

The BOP algorithm can be geometrically described as follows (see figure 2.3): make a projection of $d[m]$ on an L-dimensional plane spanned by the vectors $x_0[m], \ldots, x_L[m]$ with:

$$x_i[m] = x[m \cdot L - i] = (x[m \cdot L - i], \ldots, x[m \cdot L - i - N + 1])^T$$

(2.17)

Remembering (2.4) (with of course the block index $m$ instead of the sample index $k$) this yields:

$$<x_i[m], a_i[m]> = 0 \text{ for } i \in \{0, \ldots, L-1\}$$

(2.18)

$$d^i[m] = \sum_{i=0}^{L-1} c_i[m] \cdot x_i[m]$$

$$= x[m] \cdot c[m]$$

(2.19)

![Figure 2.3: Geometric interpretation of the BOP method](image-url)
Chapter 2: Derivation of OP algorithm

with:

\[ \mathbf{g}[m] = (c_0[m], \ldots, c_{L-1}[m])^\top \]  \hspace{1cm} (2.20)

\[ \mathbf{X}[m] = (\mathbf{x}_0[m], \ldots, \mathbf{x}_{L-1}[m]) \]  \hspace{1cm} (2.21)

As again \( s[k] \) is assumed to be zero, figure 2.3 induces for the residual signal vector \( \mathbf{z}[m] \):

\[ \mathbf{z}[m] = (z_0[m], \ldots, z_{L-1}[m]) = \mathbf{X}[m]'\mathbf{g}[m] \]  \hspace{1cm} (2.22)

Here the typical block behaviour can be observed, the same adaptive weights are used to calculate all the residual signals in the vector \( \mathbf{z}[m] \). As with the NLMS algorithm the vector \( \mathbf{g}[m] \) can be eliminated from (2.22):

\[ \mathbf{z}[m] = \mathbf{X}[m]'\mathbf{d}[m] = \mathbf{X}[m]'\mathbf{d}^\parallel[m] \]  \hspace{1cm} (2.23)

Calculating the vector \( \mathbf{d}[m] \) yields:

\[ \mathbf{d}[m] = \mathbf{d}^\parallel[m] - 2 \cdot \alpha \cdot \mathbf{X}[m]' \mathbf{R}^\parallel[m] \mathbf{X}[m] \]  \hspace{1cm} (2.24)

with the L-by-L estimate of the autocorrelation matrix of the input signal:

\[ \mathbf{R}^\parallel[m] = \mathbf{X}[m]' \mathbf{X}[m] \]  \hspace{1cm} (2.25)

This result makes it possible to express the parallel component \( \mathbf{d}^\parallel[m] \) of \( \mathbf{d}[m] \) in terms of \( \mathbf{X}[m] \), \( \mathbf{R}^\parallel[m] \) and \( \mathbf{X}[m] \) and use it to reduce the length of \( \mathbf{d}[m] \) and rotate \( \mathbf{d}[m] \) in such a way that it becomes more orthogonal to the hyperplane spanned by the columns of \( \mathbf{X}[m] \):

\[ \mathbf{d}[m+1] = \mathbf{d}[m] - 2 \cdot \alpha \cdot \mathbf{X}[m]' \mathbf{R}^\parallel[m] \mathbf{X}[m] \]  \hspace{1cm} (2.26)

This leads to the BOP update equation:

\[ \mathbf{w}[m+1] = \mathbf{w}[m] + 2 \cdot \alpha \cdot \mathbf{X}[m]' \mathbf{R}^\parallel[m] \mathbf{X}[m] \]  \hspace{1cm} (2.27)

The procedure using this equation yields one new adaptive weight vector after every \( L \) input samples, meanwhile calculating \( L \) residual signals (\( L \) convolutions) with the old adaptive weight vector.
Chapter 2: Derivation of OP algorithm

The sliding approach on the contrary is based on the following equation:

\[ y[k+1] = y[k] + 2 \cdot \alpha \cdot X[k] \cdot R_L^{-1}[k] \cdot e[k] \quad (2.28) \]

with:

\[ X[k] = (x_0[k], \ldots, x_{L-1}[k]) \quad (2.29) \]
\[ e[k] = (r_0[k], \ldots, r_{L-1}[k]) \quad (2.30) \]

As \( r_i[k] = x_i^T[k] \cdot e[k] \neq r[k-i] \), using this equation implies calculating \( L \) new residual signals during each iteration. This results in a (probably) large complexity increase compared to the block approach used above. In [SOM89b] however an efficient implementation of this algorithm is given when the input signal is generated by an Auto Regressive (AR) process (see figure 3.2 of chapter three). In the first instance only the block approach will be used. Afterwards an efficient way of implementing the sliding approach will be discussed in chapter four.

The well known Recursive Least Squares (RLS) algorithm of equation (2.31) is strongly related to the (sliding) OP algorithm:

\[ y[k+1] = y[k] + 2 \cdot \alpha \cdot R_{NN}^{-1}[k] \cdot X[k] \cdot e[k] \quad (2.31) \]

The OP algorithm decouples the dimension of the autocorrelation matrix from the number of taps of the adaptive filter. Therefore the order of the matrices \( X \) and \( R_L^{-1} \) (or \( R_{NN}^{-1} \)) is reversed. The inverse autocorrelation matrix \( R_{NN}^{-1}[k] \) can be updated using a simple recursive algorithm [ALE86], which can [SCH88, page 69] be used for the (B)OP case as will be shown in chapter four.
Chapter 2: Derivation of OP algorithm

2.4 The B(N)LMS algorithm

In [SOM90] the Block Least Mean Square (BLMS) algorithm is mentioned as a method to improve the estimation of the difference vector $g[k]$. This is done by projecting (geometrical interpretation) $g[k]$ separately on the last $L$ not necessarily orthogonal signal vectors instead of only on the last vector $x_m[k]$. This algorithm in fact is an intermediate stage between the LMS algorithm and the BOP algorithm of the previous paragraph:

$$ w[m+1] = w[m] + 2 \cdot \alpha / L \cdot X[m] \cdot z[m] \quad (2.32) $$

The matrix $X[m]$ and the vector $z[m]$ are defined here the same way as in the previous paragraph, eq. (2.21) and (2.22).

For the BLMS algorithm also a sliding approach can be used instead of a block approach, which results in the same algorithm, with again the block index $m$ replaced by the sample index $k$:

$$ w[k+1] = w[k] + 2 \cdot \alpha / L \cdot X[k] \cdot z[k] \quad (2.33) $$

The difference between the BLMS algorithm and the BOP algorithm is that in the latter case the projection is made on a plane spanned by the signal vectors $x_m[m]$ until $x_{L-1}[m]$ while in the BLMS case this projection in general is not the projection of the vector $g[m]$ on the plane spanned by the signal vectors $x_m[m]$ until $x_{L-1}[m]$ (but the sum of the projections on the vectors $x_0[m]$ until $x_{L-1}[m]$). This is caused by the fact that those vectors are normally not orthogonal to each other, as the signal $x[k]$ is in general not a white noise signal.

In [SOM90] an analysis of the convergence behaviour of the BLMS algorithm is given for the case that $x[k]$ is a white noise signal. In [FEU85] a more complete performance analysis of the BLMS algorithm is presented. The results presented there provide exact indication of how the block size affects convergence rate and steady state performance. Also a (new) measure for the rate of convergence is introduced:

$$ J = \sum_{i=0}^{\infty} (\delta(i) - \delta(\infty)) \quad (2.34) $$

15
This measure however, is not very useful as two algorithms with the same rate of convergence \( J \), can have enormously different convergence properties, as is shown in chapter six. For this measure for the rate of convergence a closed expression is found, which is still not yet the case for the initial speed \( \tau_w \) introduced in paragraph 2.2. This is probably the reason why Feuer introduced that measure. Although this measure can probably also be calculated for the BOP-case, it will not be used here, for the reason mentioned above.

The BLMS algorithm can be normalised just like the LMS algorithm to yield the first Block Normalised least Mean Square (BNLMS-1) algorithm:

\[
\tilde{w}[m+1] = \tilde{w}[m] + 2 \cdot \alpha / (L \cdot \hat{\sigma}_x^2) \cdot \tilde{x}[m] \cdot \tilde{e}[m] \tag{2.35}
\]

with \( \hat{\sigma}_x^2 \) an estimate of \( \sigma_x^2 = \mathbb{E}\{x^2[k]\} \).

One could however also try to normalise the L signal vectors separately yielding the BNLMS-L algorithm:

\[
\tilde{w}[m+1] = \tilde{w}[m] + 2 \cdot \alpha \cdot (N/L) \cdot \tilde{x}[m] \cdot \tilde{P}^{-1}[m] \cdot \tilde{e}[m] \tag{2.36}
\]

in which:

\[
\tilde{P}[m] = \text{diag}(\|\tilde{x}_0[m]\|^2, ..., \|\tilde{x}_L[m]\|^2) \tag{2.37}
\]

The first algorithm has a lower complexity than the second algorithm, as only one average power estimate has to be calculated, which can be done by the following recursive scheme:

\[
\hat{\sigma}_x^2[m] = (1-\mu) \cdot \hat{\sigma}_x^2[m-1] + \mu / L \cdot \|\tilde{x}_0[m]\|^2 \tag{2.38}
\]

in which:

\[
\tilde{x}[m] = (x[m\cdot L-i], ..., x[m\cdot L-i-L+1])^T \tag{2.39}
\]

In most cases the parameter \( N \) is large enough to assure that for all \( i \):

\[
\|\tilde{x}[m]\|^2 = N \cdot \hat{\sigma}_x^2 \tag{2.40}
\]

In that case the performance of both algorithms will be equal. Some more information on the \( B(N) \) LMS algorithm is given in paragraph 5.6, where the BNLMS-L version is used.
2.5 Points of research

The aim of this report is to gain insight into the behaviour of the (B)OP algorithm introduced in paragraph 2.3. Some very interesting questions are:

- How does the parameter L (the dimension of the autocorrelation matrix) influence the convergence behaviour. How can the parameter L be chosen as small as possible (to reduce complexity) with still a 'good' convergence behaviour?
- (How) does the echo path impulse response influence the choice of the parameter L?
- How can the (implementation) complexity of the (B)OP algorithm be reduced?
- What is the influence of the near-end signal $s[k]$ on the convergence behaviour of the algorithm?
- How can the algorithm be interpreted (simplified?) with:
  - Single Value Decomposition (SVD)
  - Transformations to the Frequency Domain
  - Transforming the input matrix to an orthogonal base (Gram-Schmidt orthogonalisation)
2.6 Further outline of the report

At first some experiments with the OP algorithm will be performed to gain insight on the properties of the BOP algorithm described in paragraph 2.3. Therefore an echo canceller using this algorithm is simulated using the SPOX-program as will be explained in chapter three.

Chapters four and five describe diverse theoretical aspects of the BOP algorithm. In chapter four an efficient implementation of the BOP algorithm is discussed. Chapter five contains an analysis of the convergence behaviour of the BOP algorithm and interprets the algorithm in diverse fashions to get a better view of the BOP algorithm and simplify analysis.

Chapters six and seven introduce a method to construct the convergence curves of the BOP algorithm from the input-signal statistical properties and the echo-path impulse response (and the parameters of the algorithm).
Chapter 3
Simulation with SPOX

3.1 Introduction

Since the (Block) Orthogonal Projection algorithm is rather complex, no easy theoretical analysis can be performed on the convergence behaviour of a filter using this algorithm. Therefore at the first instance a program is developed that simulates a filter based on that algorithm. The filter structure chosen to simulate, is based on echo cancellation. The main reason for choosing an echo canceller structure is the fact that it is a well known structure in many communication applications. A second reason is that a lot of theoretical analysis on the echo canceller structure using all kinds of different adaption algorithms has been performed ([ALE86], [SOM89], [STE88], [SOM89b], [TRE87], [HON84], [HAY86], [HAY91], [SOM90a], [WID85]). This may prove useful later on when trying to theoretically analyse the filter convergence behaviour.

The filter will be simulated using the SPOX soft- and hardware that is shortly described in appendix A and thoroughly studied in [WIT91].
3.2 The BOP program

The BOP program listed in Appendix B, consists of a group of functions followed by the main program. The main program is built from an initialization part and a filter part. The random generator used in the program comes from [CLU91]. The program is listed because this way easy can be seen that only minor adaptations are needed to change the program for other block adaptive filter algorithms (such as the Gram-Schmidt orthogonalization algorithm discussed in paragraph 4.6). In most cases only one or few functions have to be added or changed.

The results (the \( r[k] \) values, see chapter two) are stored in a file that can be read by MATLAB. Loading this file into MATLAB and processing the data in it (square it, take an average, take the logarithm) and plot it will yield the picture of the average squared error value against the sample number (the misadjustment as defined in the previous chapter).

In chapter two the echo path impulse response \( h \) is assumed to have finite length of at most \( N \) and can therefore realized by a FIR (Finite Impulse Response) transversal filter of figure 3.1 (with as input signal \( x[k] \) instead of \( n[k] \), and as output \( e[k] \)). If a (white) innovation signal \( n[k] \) is used as input, then the output signal \( x[k] \) is said to be modelled as a MA(\( N \)) signal.

![Figure 3.1: Transversal FIR filter to realise MA-signal model.](image)
Chapter 3: Simulation with SPOX

Another well known signal-model is the AR(N) (Auto-Regressive) model, realised by the filter of figure 3.2. The combination of these two models yields the ARMA(M,N) model, realised in figure 3.3. All time-discrete stationary stochastic signals can be modelled by the ARMA(M,N) model by proper choice of M and N [SOK90]. The program of appendix B therefore models all its input signals (x[k] and s[k]) by the ARMA-model. The adaptive filter w is realised by a FIR filter. The echo-path is (more general) realised by the filter of figure 3.3.

![Figure 3.2: Filter to realise AR-signal.](image)

![Figure 3.3: Filter to realise ARMA(M,N) signal.](image)
3.3 Results using the BOP program

To gain insight in the BOP algorithm some tests have been performed with the program introduced in the previous paragraph. In the first instance the echo-path impulse response is considered to be of length one (delta-pulse). For the first experiment a (normalised) AR(1) signal is used as input:

\[ x[k] = 0.312 \cdot (n[k] - 0.95 \cdot x[k-1]) \]  \hspace{1cm} (3.1)

The number of filter-coefficients is chosen to be 16. In figure 3.4 the average squared error is plotted against the sample number for \( \alpha = 0.04 \), no additional noise and different values of \( L \). The curves drawn are obtained by manually 'smoothing' (drawing a line) the output of the program. The curves should be obtained by ensemble averaging instead of taking the time average, but this would cost a lot of time (a lot of simulations have to be made) and a lot of computations.

Figure 3.4: Average squared error for AR(1) signal without noise.
Chapter 3: Simulation with SPOX

The second experiment takes a (normalised) MA(2) signal as input:

\[ x[k] = 0.577 \cdot (n[k] + n[k-1] + n[k-2]) \]  

(3.2)

Here also 16 is chosen for the number of filter coefficients. The average squared error is plotted against the sample number in figure 3.5 for \( \alpha=0.04 \), no additional noise and different values of L.

![Figure 3.5: Average squared error for MA(2) signal without noise.](image)

Clearly can be seen that the second signal needs a larger value of L to yield the same decorrelation properties than the first signal. From this it can be concluded that the statistical properties of the input signal will influence the convergence behaviour. If for example a misadjustment of -30 dB is allowed, the first signal (figure 3.4) needs only L=2 as decorrelation parameter to get the same speed as L=16 (full decorrelation). In the second case L has to be at least equal to 8 to yield the same results.

The third experiment uses the same input signal as the second experiment, but now noise is added with \( \sigma_n^2 = 0.01 \). The results are plotted in figure 3.6.
Chapter 3: Simulation with SPOX

Figure 3.6: Average squared error for MA(2) signal with additional noise.

The noise level defines a final misadjustment for the convergence curves. This misadjustment is studied in chapter five.

The echo-path impulse response will also effect the choice of the parameter L, this is investigated in chapter six.
4.1 Exact implementation

In paragraph 2.3 already the suggestion is made that the inverse autocorrelation matrix $R^{-1}_{uu}[m]$ can be calculated by using a recursive scheme. From (2.25) and (2.21) it follows that:

$$(Ru.[m])_{p,q} = < x_p[m], x_q[m] >$$  \hspace{1cm} (4.1)

In this equation (4.1) the indices $p$ and $q$ indicate the element in the matrix in the $(p+1)'$th row and $(q+1)'$th column. For simplicity reasons now the block-index $m$ will be eliminated by re-introducing the sample-index $k$ into (4.1):

$$(R_{uu}[m])_{p,q} = (R_{uu}[m\cdot L])_{p,q}$$  \hspace{1cm} (4.2)

The sample-indexed matrix elements can now be expressed as:

$$(R_{uu}[k])_{p,q} = < x[k-p], x[k-q] >$$

$$= \sum_{i=0}^{N-1} x[k-p-i] \cdot x[k-q-i]$$

$$= ( \sum_{i=0}^{N-1} x[k-p-i-1] \cdot x[k-q-i-1] ) + x[k-p] \cdot x[k-q] - x[k-p-N] \cdot x[k-q-N]$$  \hspace{1cm} (4.3)
Chapter 4: Efficient implementation

This implies:

\[ R_{ll}[k] = R_{ll}[k-1] + \mathbf{z}[k] \cdot \mathbf{z}^T[k] - \mathbf{z}[k-N] \cdot \mathbf{z}^T[k-N] \]  

(4.4)

The vectors \( \mathbf{z}[k] \) are defined as vectors of length \( L \) containing the last \( L \) samples \( x[k] \) till \( x[k-L+1] \).

Yet equation (4.4) wouldn’t be of any use without the famous matrix inversion lemma [HAY91], defined as follows:

Let \( A \) and \( B \) be two positive-definite \( M \)-by-\( M \) matrices, \( D \) a \( N \)-by-\( N \) matrix, \( C \) a \( M \)-by-\( N \) matrix and \( E \) a \( N \)-by-\( M \) matrix related by:

\[ A = B + C \cdot D \cdot E \]  

(4.5)

The matrix inversion lemma expresses the inverse of matrix \( A \) as follows:

\[ A^{-1} = B^{-1} - B^{-1} \cdot C \cdot (D^{-1} + E \cdot B^{-1} \cdot C)^{-1} \cdot E \cdot B^{-1} \]  

(4.6)

The proof of this lemma is established by multiplying equation (4.5) by (4.6).

The matrix inversion lemma can be used by rewriting it as follows:

\[ A = B - d \cdot d^T \]
\[ B = C + e \cdot e^T \]  

(4.7)

The inversion lemma (4.6) expresses the inverse of matrix \( A \) as follows:

\[ A^{-1} = B^{-1} + B^{-1} \cdot d \cdot (1 - d^T \cdot B^{-1} \cdot e) \cdot e^T \cdot B^{-1} \]
\[ B^{-1} = C^{-1} - C^{-1} \cdot e \cdot (1 + e^T \cdot C^{-1} \cdot e) \cdot e^T \cdot C^{-1} \]  

(4.8)

The lemma (4.8) can be used making the following identifications:

\[ A = R_{ll}(k) \]
\[ C = R_{ll}(k-1) \]
\[ d = \mathbf{z}[k-N] \]
\[ e = \mathbf{z}[k] \]

Then, substituting these definitions in eq. (4.8) the following recursive scheme for updating the inverse of the correlation matrix is obtained:
Chapter 4: Efficient implementation

\[ P = R_{Lx}^{[k-1]} \cdot x[k] \]  
(4.9)

\[ f = 1/(1 + x^T[k] \cdot P) \]  
(4.10)

\[ Q = R_{Lx}^{[k-1]} - P \cdot (P^T/f) \]  
(4.11)

\[ v = Q \cdot x[k-N] \]  
(4.12)

\[ q = 1/(1 - x^T[k-N] \cdot v) \]  
(4.13)

\[ R_{Lx}^{[k]} = Q + v^T \cdot (v^T/g) \]  
(4.14)

The equations (4.9) untill (4.14) define an algorithm to update the inverse of the autocorrelation matrix without matrix-inversion. Initialization of \( R_{Lx}^{[0]} \) can be done by making the following assumptions:

\[ x[k] = \delta \text{ if } k \leq 0 \text{ and } (k \mod L) = 0 \]

\[ x[k] = 0 \text{ if } k < 0 \text{ and } (k \mod L) \neq 0 \]  
(4.15)

With (4.3) this yields:

\[ (R_{Lx}^{[0]})_{m,n} = \sum_{i=0}^{N-1} x[-p-i] \cdot x[-q-i] \]

\[ = \delta \cdot ((N \div L) + 1) \text{ if } p = q \text{ and } p < N \mod L \]  
(4.16)

\[ = \delta \cdot (N \div L) \text{ if } p = q \text{ and } p \geq N \mod L \]

This matrix has only non-zero entries on its diagonal and can therefore very easily be inverted (by inverting its diagonal entries!). Eq. (4.15) introduces a history of the signal \( x[k] \). For continuity purposes the parameter \( \delta \) can be chosen to obtain the same average of \( x[k] \) for \( k \) smaller than one as for \( k \) greater than (or equal to) one. This yields \( \delta_m = (L \cdot \sigma^2)^m \).

Easily can be computed that the number of multiplications in (4.9) till (4.14) is: \( 2 \cdot (L^2 + (1 + L) + (L^2 + L)) = 4 \cdot L^2 \). At first the reduction in complexity seems enormous, as directly inverting a matrix is of order \( L^3 \) and computing the elements of the matrix directly costs order \( N \cdot L \) multiplications (with (4.4) order \( L^2 ! \)). This will indeed be the case for the sliding-OP algorithm, but one has to realize that the above introduced update equation is on sample-base, and not on block base.

Using the block approach inverting a matrix of \( L \)-by-\( L \) once every \( L \) samples is of order \( L^3 \). The advantage of the above introduced method for matrix inversion is the fact that the computations do not all have to take place in the same time interval (the matrix is updated after every new sample, so computations are divided over \( L \) time-intervals). Directly inverting the
matrix once every L samples requires a lot of computations in the same
time-interval, therefore a long time-interval or a lot of parallelism and
will thus result in a large implementation complexity or a lot of delay
compared to the method using the above introduced algorithm. Of course more
computations than inverting a matrix have to be done using the BOP algo-

rithm. These computations include a matrix-matrix and a matrix-vector
product but are not considered here.

In the case of the sliding-OP algorithm still another problem has to be
solved: the computation of the residual signal $z[k]$ seems rather difficult
(as was indicated in chapter two). One can however simplify the computa-
tions [KUR87]. From (2.30) can be concluded that (for $1 \leq i \leq N-1)$:

$$r_i[k] = x_i[k]^T (h - x[k])$$
$$= x_i[k-1]^T (h - x[k-1]) - 2 \cdot \alpha \cdot x_i[k-1]^T \cdot x[k-1] \cdot r_i[k-1]$$
$$= r_i[k-1] - 2 \cdot \alpha \cdot r_i[k-1]$$
$$= (1 - 2 \cdot \alpha) \cdot r_i[k-1]$$

(4.17)

A simple delay-line with multiplicators can be used to realize this
equation.
4.2 Approximations to simplify the implementation

The complexity of the algorithm used in the previous paragraph can be reduced by using another (approximate) update algorithm for the autocorrelation-matrix:

$$R_{\mu}[k] = (1-\tau) \cdot R_{\mu}[k-1] + \tau \cdot (x[k] \cdot x^T[k])$$ (4.18)

Implementation complexity can roughly be reduced by a factor 2 using this algorithm. The effects on the performance of the algorithm are not investigated here, but will probably have a low-pass character.
Chapter 5

Interpretations of BOP

5.1 Introduction

In order to gain insight into different properties of the (B)OP algorithm, at first an analysis of the algorithm is presented. Thereafter the algorithm is interpreted in diverse ways with diverse mathematical tools, such as Singular Value Decomposition (SVD), transformation to the frequency domain and Gram-Schmidt orthogonalisation. At last a general orthogonal transform is defined that includes the tools mentioned above.
5.2 Analysis of BOP algorithm

In this paragraph an analysis of the average misadjustment of the BOP algorithm of equation (2.27) is given. This average misadjustment is defined as follows (compare to \[ (SOM90, \text{part on BLMS}) \]):

$$\delta[m] = (1/L) \cdot ( \sum_{i=0}^{L-1} \| \mathbf{x}_i^T[m] \cdot \mathbf{h} - \mathbf{s}_i^T[m] \cdot \mathbf{x}[m] \|^2 )$$  \hspace{1cm} (5.1)

$$= (1/L) \cdot \mathbb{E}\{ \| \mathbf{x}_m^T[m] \cdot \mathbf{d}[m] \|^2 \}$$

$$= (1/L) \cdot \mathbb{E}\{ (\mathbf{x}_m^T[m] \cdot \mathbf{d}[m])^T \cdot (\mathbf{x}_m^T[m] \cdot \mathbf{d}[m]) \}$$  \hspace{1cm} (5.2)

The rest of this paragraph it is assumed that the signal \( \mathbf{g}[k] \) has zero mean and that its variance \( \mathbb{E}\{\mathbf{g}^T[k] \mathbf{g}[k]\} = \sigma_g^2 \). Also is assumed that there exists no correlation between the signals \( \mathbf{s}[k] \) and \( \mathbf{x}[k] \). Equations (5.2) and (2.19) then yield:

$$\delta[m] = (1/L) \cdot \mathbb{E}\{ (\mathbf{x}_m^T[m] \cdot \mathbf{d}[m])^T \cdot (\mathbf{x}_m^T[m] \cdot \mathbf{d}[m]) \}$$

$$= (1/L) \cdot \mathbb{E}\{ (\mathbf{x}_m^T[m] \cdot (\mathbf{g}[m-1] - 2 \cdot \mathbf{x}(m-1) \cdot \mathbf{R}_m[m-1] \cdot \mathbf{X}_m^T[m-1] \cdot \mathbf{d}[m-1] - \mathbf{g}[m-1]))^T \cdot (\mathbf{x}_m^T[m] \cdot (\mathbf{g}[m-1] - 2 \cdot \mathbf{x}(m-1) \cdot \mathbf{R}_m[m-1] \cdot \mathbf{X}_m^T[m-1] \cdot \mathbf{d}[m-1] + \mathbf{g}[m-1])) \}$$

$$= (1/L) \cdot (a_1 + a_2 + a_3 + a_4)$$  \hspace{1cm} (5.3)

with:

$$a_1 = \mathbb{E}\{\mathbf{g}^T[m-1] \cdot (\mathbf{I}_m - 2 \cdot \mathbf{x}(m-1) \cdot \mathbf{R}_m[m-1] \cdot \mathbf{X}_m^T[m-1] \cdot \mathbf{d}[m-1]) \cdot \mathbf{x}[m] \cdot \mathbf{x}_m^T[m] \cdot (\mathbf{I}_m - 2 \cdot \mathbf{x}(m-1) \cdot \mathbf{R}_m[m-1] \cdot \mathbf{X}_m^T[m-1] \cdot \mathbf{g}[m-1]) \}$$  \hspace{1cm} (5.4)

$$a_2 = -2 \cdot \alpha \cdot \mathbb{E}\{\mathbf{g}^T[m-1] \cdot \mathbf{R}_m[m-1] \cdot \mathbf{x}_m^T[m-1] \cdot \mathbf{x}[m] \cdot \mathbf{x}_m^T[m] \cdot (\mathbf{I}_m - 2 \cdot \mathbf{x}(m-1) \cdot \mathbf{R}_m[m-1] \cdot \mathbf{X}_m^T[m-1] \cdot \mathbf{d}[m-1]) \}$$

$$a_3 = -2 \cdot \alpha \cdot \mathbb{E}\{\mathbf{g}^T[m-1] \cdot (\mathbf{I}_m - 2 \cdot \mathbf{x}(m-1) \cdot \mathbf{R}_m[m-1] \cdot \mathbf{X}_m^T[m-1] \cdot \mathbf{x}[m] \cdot \mathbf{x}_m^T[m] \cdot \mathbf{x}[m-1] \cdot \mathbf{R}_m[m-1] \cdot \mathbf{g}[m-1]) \}$$

$$a_4 = 4 \cdot \alpha^2 \cdot \mathbb{E}\{\mathbf{g}^T[m-1] \cdot \mathbf{R}_m[m-1] \cdot \mathbf{x}_m^T[m-1] \cdot \mathbf{x}[m] \cdot \mathbf{x}_m^T[m] \cdot \mathbf{x}[m-1] \cdot \mathbf{R}_m[m-1] \cdot \mathbf{g}[m-1]) \}$$  \hspace{1cm} (5.5)

The variables \( a_2 \) and \( a_4 \) are both equal to zero as there exists no correlation between the signals \( \mathbf{s}[k] \) and \( \mathbf{x}[k] \) so \( \mathbb{E}\{\mathbf{x}_m^T[k] \cdot \mathbf{g}[k]\} = \mathbb{E}\{\mathbf{g}[k]\} \cdot \mathbb{E}\{\mathbf{g}[k]\} \). As both the expected values are equal to zero also \( \mathbb{E}\{\mathbf{x}_m^T[k] \cdot \mathbf{g}[k]\} \) is equal to zero. Further analysis on \( a_1 \) yields:
Chapter 5: Performance analysis

\[ a_i = \mathbb{E}\{d^T[m-1] \cdot R_{NN} \cdot d[m-1]\} - \\
2 \cdot \alpha \cdot \mathbb{E}\{d^T[m-1] \cdot (R'_{NN} \cdot R_{NN} + R_{NN} \cdot R'_{NN}) \cdot d[m-1]\} + \\
4 \cdot \alpha^2 \cdot \mathbb{E}\{d^T[m-1] \cdot (R'_{NN} \cdot R_{NN} \cdot R'_{NN}) \cdot d[m-1]\} \quad (5.8) \]

with:

\[ R_{NN} = \mathbb{E}\{X[m] \cdot X^T[m]\} = \mathbb{E}\{X[m-1] \cdot X^T[m-1]\} \quad (5.9) \]
\[ R'_{NN} = \mathbb{E}\{X[m-1] \cdot R'_{LL}[m-1] \cdot X^T[m-1]\} \quad (5.10) \]

Equation (5.8) can be rewritten as:

\[ a_i = \delta[m-1] - 2 \cdot \alpha \cdot (\delta'[m-1] + 2 \cdot \alpha \cdot \delta''[m-1]) \quad (5.11) \]

with:

\[ \delta'[m-1] = \mathbb{E}\{d^T[m-1] \cdot (R'_{NN} \cdot R_{NN} + R_{NN} \cdot R'_{NN}) \cdot d[m-1]\} \quad (5.12) \]
\[ 2 \cdot \alpha \cdot \delta''[m-1] = 2 \cdot \alpha \cdot \mathbb{E}\{d^T[m-1] \cdot (R'_{NN} \cdot R_{NN} \cdot R'_{NN}) \cdot d[m-1]\} = 0 \quad (2 \cdot \alpha \cdot \delta''[m-1] \ll \delta'[m-1]) \quad (5.13) \]

The last approximation can be made for small \( \alpha \ll 1 \) (in most applications this is indeed the case). In the above equations the matrix \( R'_{NN} \) in fact defines in combination with the autocorrelation matrix \( R_{NN} \) the behaviour of \( \delta[m] \). As \( L \) increases towards \( N \) this matrix \( R'_{NN} \) will more and more resemble the \( NxN \) identity matrix. \( \delta'[m-1] \) will therefore with increasing \( L \) towards \( N \) converge to \( \delta[m-1] \). In that case the convergence behaviour of the BOP algorithm would be the same as the BNLMS algorithm, with a white noise input signal (see paragraph 2.4). For \( L=1 \) the convergence behaviour will be that of the BNLMS (Block Normalised Least Mean Square) algorithm, with the original (possibly colored) signal as input. More investigations on the properties of \( \mathbb{E}\{X[m-1] \cdot R'_{LL}[m-1] \cdot X^T[m-1]\} \) are performed in chapter seven.

Analysing the effect of the signal \( s[k] \) yields:

\[ a_4 = 4 \cdot \alpha^2 \cdot \mathbb{E}\{s^T[m-1] \cdot X'^T[m-1] \cdot X[m] \cdot X^T[m] \cdot X'^T[m-1] \cdot s[m-1]\} \quad (5.14) \]

in which \( X'^T[m-1] \) is defined as the pseudo-inverse [HAY91] of \( X[m-1] \):

\[ X'^T[m-1] = R'_{LL}[m-1] \cdot X^T[m-1] \quad (5.15) \]
Chapter 5: Performance analysis

Remembering that there is no correlation between $s[k]$ and $x[k]$, eq. (5.11) can be written as follows:

$$a_4 = 4 \cdot \alpha^2 \cdot E\{s'[m-1] \cdot E\{Y[m] \cdot Y'[m]\} \cdot g[m-1]\}$$

$$= 4 \cdot \alpha^2 \cdot \sigma^2 \cdot \text{trace}(E\{Y[m] \cdot Y'[m]\})$$

(5.16)

with:

$$Y[m] = X'[m-1] \cdot X[m]$$

(5.17)

If one assumes the signals $s[k]$ and $x[k]$ to be time-stationary, eq. (5.16) introduces a final misadjustment into $\delta[m]$ ($a_4$ will in general not become infinitely small). The terms $a_4$ and $a_4$ can be calculated if the statistical properties of $x[k]$ and $s[k]$ are known, and with that the misadjustment $\delta[m]$, but this is a very complex procedure. By interpreting the (B)OP algorithm differently, more insight into the convergence behaviour of the algorithm is obtained as will be shown in the next paragraphs.
5.3 Interpretation with SVD

5.3.1 The Singular-Value Decomposition (SVD) theorem

The Singular-Value Decomposition (SVD) theorem is defined as follows [HAY91]: given the data matrix \( A \), there are two unitary matrices \( V \) and \( U \), such that:

\[
U^H A V = \Sigma
\]

where \( \Sigma \) is a diagonal matrix:

\[
\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_W)
\]

and (the unitarity-property):

\[
U^H U = U U^H = I_K
\]

\[
V^H V = V V^H = I_M
\]

The \( \sigma \)'s are ordered as \( \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_W > 0 \). The theorem above is also referred to as the Autonne-Eckart-Young theorem in recognition of its originators [KLE80]. Figure 5.1 presents a diagrammatic interpretation of the above described SVD theorem. In this diagram is assumed that the number of rows \( K \) contained in the data matrix \( A \) is larger than the number of columns \( N \), and that the number of nonzero singular values \( W \) is less than \( M \).

![Diagrammatic interpretation of the SVD theorem](image-url)
Let the number of rows of \( A \) be equal to \( K \) and the number of columns equal to \( M \). Then \( U \) will be a unitary \( K \times K \) matrix, and \( V \) a unitary \( M \times M \) matrix. \( U^H \) is the Hermitian transpose of the matrix \( U \). The subscript \( W \) in (5.28) is the rank of matrix \( A \), written as \( \text{rank}(A) \). It is defined as the number of linearly independent columns in the matrix \( A \). Of course \( W = \text{rank}(A^H) = \text{rank}(A) \leq \min(K,M) \).

### 5.3.2 Properties of the SVD

The following properties can be deducted from the SVD theorem (5.18) as is proved in [HAY91], assuming that the rank \( W \) of the data matrix \( A \) is equal to its number of columns \( M \) and smaller than its number of rows \( K \):

\[
A = U \cdot [\Sigma] \cdot V^H
\]

(5.22)

\[
U^H \cdot A \cdot V = [\Sigma]
\]

(5.23)

The squares of the diagonal values of the matrix \( \Sigma \) are the eigenvalues of the matrix \( A^T \cdot A \). These eigenvalues are equal to the \( M \) non-zero eigenvalues of \( A^T \cdot A \). The corresponding set of orthonormal eigenvectors of \( A^T \cdot A \) form the columns of \( V \). The set of orthonormal eigenvectors of \( A \cdot A^T \) form the columns of \( U \).

### 5.3.3 SVD to analyse BOP

The SVD theorem from the above paragraph can be used to simplify the BOP update equation from equation (2.27):

\[
x[m+1] = x[m] + 2 \cdot \alpha \cdot x[m] \cdot R_{\hat{L}}^L[m] \cdot x[m]
\]

(5.24)

The data-matrix \( X[m] \) can according to eq. (5.22) be expressed as follows (\( A = X[m] \), \( M = L \) and \( K = N \)):

\[
X[m] = U[m] \cdot [\Sigma^m] \cdot V^T[m]
\]

(5.25)

The inverse of the estimated autocorrelation matrix \( R_{\hat{L}}^L[m] \) can now be written as:
\[ R_{11}^m = (X^T[m] \cdot X[m])^{-1} \]
\[ = ((U[m] \cdot [\Sigma^m] \cdot \Psi^T[m])^T \cdot U[m] \cdot [\Sigma^m] \cdot \Psi^T[m])^{-1} \]
\[ = (U[m] \cdot [\Sigma^m] \cdot \Psi^T[m])^T \cdot U[m] \cdot (U[m] \cdot [\Sigma^m] \cdot \Psi^T[m])^{-1} \]
\[ = U[m] \cdot [\Sigma^2[m] \cdot \Psi^T[m]] \]
\[(5.26)\]

As the matrix \( \Sigma[m] \) is a diagonal matrix it is very easy to invert (by inverting its diagonal entries). The residual signal \( x[m] \) can also be rewritten:

\[ x[m] = X^T[m] \cdot g[m] + s[m] \]
\[ = V[m] \cdot [\Sigma[m] \cdot 0] \cdot U^T[m] \cdot g[m] + s[m] \]
\[(5.27)\]

Filling in equations (5.25) till (5.27) in eq. (5.24) gives the following result:

\[ x[m+1] = x[m] + 2 \cdot \alpha \cdot U[m] \cdot [\Sigma^m] \cdot \Psi^T[m] \cdot V[m] \cdot \Sigma^2[m] \cdot \Psi^T[m] \cdot (U[m] \cdot [\Sigma[m] \cdot 0] \cdot g[m] + s[m]) \]
\[ = x[m] + 2 \cdot \alpha \cdot U[m] \cdot [U[m] \cdot \Sigma^2[m] \cdot [\Sigma[m] \cdot 0] \cdot g[m] + s[m]) \]
\[(5.28)\]

with:

\[ s'[m] = [\Sigma^2[m] \cdot \Psi^T[m] \cdot g[m]] \]
\[(5.29)\]

The signal-matrix \( X[m] \) is decomposed with SVD, yielding a 'new' signal-matrix \( U[m] \). The product \( U[m] \cdot [U[m] \cdot \Psi^T[m] \cdot g[m] + s[m]) \) forms a scattering matrix, decreasing \( L \) induces more scattering (less columns of \( U[m] \) are used to construct the product).

With eq. (5.18) untill (5.29) one can try to develop an expression for the misadjustment:

\[ \delta[m] = E\{|X^T[m] \cdot g[m]|^2\} \]
\[ = E\{\text{trace}(X^T[m] \cdot g[m] \cdot g^T[m] \cdot X[m])\} \]
\[ = E\{\text{trace}(X[m] \cdot X^T[m] \cdot g[m] \cdot g^T[m])\} \]
\[ = \text{trace}(E\{X[m] \cdot X^T[m]\} \cdot E\{g[m] \cdot g^T[m]\}) \]
\[ = E\{\text{trace}(R_{NN} \cdot E\{g[m] \cdot g^T[m]\})\} \]
\[(5.30)\]

In the above equation it is assumed that \( x[k] \) is time-stationary, so that \( E\{X[m] \cdot X^T[m]\} \) is independent of the block-index \( m \). In the first instance the signal \( s[k] \) is assumed to be zero, and a matrix \( R'[m] \) is defined:
Chapter 5: Performance analysis

\[
R'[m] = U[m] \cdot [\phi_{UL} \phi_{U}] \cdot U'[m] 
\]  
(5.31)

From eq. (5.28) and the above assumptions it follows:

\[
g[m] = g[m-1] - 2 \cdot \alpha \cdot R'[m-1] \cdot g[m-1] 
= (I_NN - 2 \cdot \alpha \cdot R'[m-1]) \cdot g[m-1] 
\]  
(5.32)

Combining eq. (5.30) and (5.32) yields:

\[
\delta[m] = \text{trace}(R_{NN} \cdot E((I_{NN} - 2 \cdot \alpha \cdot R'[m-1]) \cdot g[m-1] \cdot g'[m-1] \cdot (I_{NN} - 2 \cdot \alpha \cdot R'[m-1]))) 
= \text{trace}(R_{NN} \cdot E\{g[m-1] \cdot g'[m-1]\}) 
- 2 \cdot \alpha \cdot \text{trace}(R_{NN} \cdot E\{R'[m-1] \cdot g[m-1] \cdot g'[m-1]\}) 
- 2 \cdot \alpha \cdot \text{trace}(R_{NN} \cdot E\{g[m-1] \cdot R'[m-1]\}) 
+ 4 \cdot \alpha^2 \cdot \text{trace}(R_{NN} \cdot E\{R'[m-1] \cdot g[m-1] \cdot g'[m-1] \cdot R'[m-1]\}) 
= \delta[m-1] - 2 \cdot \alpha \cdot (\delta'[m-1] + 2 \cdot \alpha \cdot \delta''[m-1]) 
\]  
(5.33)

with:

\[
\delta'[m-1] = \text{trace}(R_{NN} \cdot R'[NN] + R'[NN] \cdot R_{NN}) \cdot E\{g[m-1] \cdot g'[m-1]\} 
R'[NN] = E\{R'[m-1]\} 
2 \cdot \alpha \cdot \delta''[m-1] = 2 \cdot \alpha \cdot \text{trace}(R'[NN] \cdot R_{NN} \cdot R'[NN] \cdot E\{g[m-1] \cdot g'[m-1]\}) 
\]  
(5.34) \quad (5.35) \quad (5.36)

When the parameter \( \alpha \) is small enough (\( \alpha \ll 1 \)), \( 2 \cdot \alpha \cdot \delta''[m-1] \) may again be neglected. If the dimension \( L \) of the OP algorithm equals \( N \), then the pseudo-autocorrelation matrix \( R'_{NN} \) equals \( I_{NN} \) and the algorithm will converge as if it were a BNLMS algorithm with an input signal with a flat spectrum ('white' noise). If however the parameter \( L \) grows increasingly more smaller than \( N \), the pseudo-autocorrelation matrix \( R'_{NN} \) will more and more resemble the autocorrelation matrix \( \Sigma \), as:

\[
R_{NN} = E\{X[m] \cdot X'[m]\} 
= E\{U[m] \cdot [\Sigma[m] \cdot V[m] \cdot V[m]' \cdot \Sigma[m] \cdot 0] \cdot U'[m]\} 
= E\{U[m] \cdot [\Sigma[m]' \Sigma[m]] \cdot U'[m]\} 
\]  
(5.37)

If \( L \) equals one, then \( [\Sigma[m]] \cdot U[m]' \) equals \( \phi_{UL}^T \cdot [\Sigma[m]] \), so the matrices \( \phi_{UL}^T \cdot R_{NN} \) and \( R'_{NN} \) are equal then. This behaviour will strongly influence the convergence behaviour. The product matrix \( R_{NN} \cdot R'_{NN} \) will grow increasingly less diagonally dominant with decreasing \( L \), which results in a convergence behaviour that diverges more and more from the curve of a BNLMS algorithm with a 'white' noise input signal. Further analysis on this product matrix is performed in chapter seven.
5.4 Interpretation with Fourier transformations

By transforming the input signal $x[k]$ (or strictly speaking the columns of the matrix $X[k]$) to the frequency domain a new class of adaptive algorithms is introduced. This transformation to the frequency domain can take place by means of a Discrete Fourier Transform (DFT) (or the more efficient Fast Fourier Transform (FFT) if the length of the transform is chosen to be a power of two) ([HAY91], [WIL91], [SOM90]).

To perform such a transform a $M \times M$ Fourier matrix $F_{MM}$ is introduced of which the element of the $(p+1)$'th row and $(q+1)$'th column is given by:

$$ (F_{MM})_{pq} = e^{-j2\pi pq/M} $$

This matrix has the following useful properties:

$$ F_{MM} = F_{MM}^T $$
$$ F_{MM}^{-1} = (1/M) \cdot F_{MM}^* $$

For verification of these properties see [SOM90].

To perform decorrelation in the frequency domain we use the relationship between the discrete autocorrelation function $p_s(t)$ defined as:

$$ p_s(t) = E\{x[k] \cdot x[k-t]\} $$

and the periodic power density spectrum (pds) $P_s(e^{j\theta})$ of the discrete input signal $x[k]$ known as the Wiener-Khintchine relation:

$$ P_s(e^{j\theta}) = \sum_{t=-\infty}^{\infty} p_s(t) \cdot e^{-j\theta t} $$

Eq. (5.42) states that $P_s(e^{j\theta})$ and $p_s(t)$ form a Fourier pair. When $x[k]$ is uncorrelated in time (white), its pds is flat (and its autocorrelation function is a delta-pulse). The more correlation occurs in the input signal, the less smooth the pds becomes. Decorrelation can be performed by splitting the (periodic) pds into several subbands (or bins) and normalising every subband by its own power. Making the number of bins large enough makes it possible to normalise the whole pds.
Chapter 5: Performance analysis

Combination of the above theory and the NLMS algorithm is a way to derive a Frequency Domain Adaptive Filter (FDAF) algorithm, see for further information [SOM90] and [WIL91]). With the BNLMS algorithm instead of the NLMS algorithm also a Block Frequency Domain Adaptive Filter (BFDAF) algorithm can be derived ([SOM90], [CLA81]). In both these cases however the Fourier-transformation length is equal to the number of taps N. Just like in the BOP case, one would like to reduce implementation complexity by reducing the length of the Fourier-transform on the input signal, as this is the only transform that is actually performed. This is done by another class of filters, the so-called (Block) Partitioned Frequency Domain Adaptive Filters ((B)PFDAF). These filters also decouple the dimension of the filter and the dimension of decorrelation. In [WIL91] more can be found about the PFDAF structures.

The theory introduced above makes it possible to look at the BOP algorithm in another way by introducing a Fourier transform of length L on the input signal matrix. In [SOM90] is derived that under certain conditions, which are not discussed or derived here, the estimate of the inverse autocorrelation matrix can be approximated by:

\[ R^{-1}_{L}[m] = F_L \cdot P^*_L[m] \cdot F_L^* \]  

(5.43)

with:

\[ P_L[m] = \text{diag}(P_0[m], P_1[m], \ldots, P_{L-1}[m]) \]  

(5.44)

\[ P_i[m] = E\{D(\tilde{x}_0[m]), D^*(\tilde{x}_0[m])\} \]  

(5.45)

where \( D(\tilde{x}_0[m]) \) denotes the \( i \)'th component of the discrete Fourier transform of \( \tilde{x}_0[m] \). \( D(\tilde{x}_0[m]) = (F_L^*/N) \cdot \tilde{x}_0[m] \). \( \tilde{x}_0[m] \) is again defined as the signal vector containing the last \( L \) values of \( x[m\cdot L] \).

With equation (5.43) the BOP equation of (2.27) can be rewritten as:

\[ \tilde{x}[m+1] = \tilde{x}[m] + 2 \cdot \alpha \cdot X^*[m] \cdot P^*_L[m] \cdot \tilde{x}[m] \]  

(5.46)

with:

\[ X^*[m] = X[m] \cdot F_L \]  

(5.47)

\[ \tilde{x}^*[m] = F_L \cdot \tilde{x}[m] \]  

(5.48)

\[ \tilde{x}^*[m] = (F_L^* \cdot X^*[m])^N \]  

(5.49)

39
Eq. (5.46) in combination with eq. (5.47) and (5.48) introduces a new way of looking at the BOP algorithm. Partial decorrelation of the input signal \( L < N \) takes place by normalising the pds of the input signal by dividing it in only \( L \) subbands instead of \( N \) subbands (the full decorrelation or RLS case) and normalising the subbands. Looking at the algorithm this way, one can see that the pds of the input signal influences the choice of the parameter \( L \).

The Fourier transforms are in fact just orthogonal transforms of the original input-base. Further analysis on orthogonal transformations in general is performed paragraph 5.6. The implementation of (5.46) can be made more efficient if FFT's can be used and \( P_{\Lambda}[m] \) is calculated efficiently ([SOM90], [HAY91]). In implementing this algorithm some extra work is needed because of the fact that the autocorrelation matrix used is not Toeplitz but cyclic [SOM90], so a FFT (or DFT) that is twice as big as one would like will be needed.

In the algorithm (5.46) the adaption itself still takes place in the time-domain (the filter-coefficients are not translated) and is therefore not directly related to the earlier mentioned (B)(P)FDAF's. Transformation of the whole algorithm to the frequency domain would need again (Discrete) Fourier-Transforms of length (two times) \( N \) on the input signal, yielding a larger complexity in implementing the transforms.
5.6 Gram-Schmidt orthogonalisation

The BOP concept of paragraph 2.3 was to make a projection of the vector \( g[m] \) on the space which was spanned by the basis:

\[
\{x_0[m], \ldots, x_L[m]\}
\]  

(5.50)

In this paragraph the same BOP concept is used, but first an orthogonal basis applying the Gram-Schmidt procedure will be derived [SOM89a]. For \( i=0 \) to \( L-1 \) this procedure leads to the next set of orthogonal vectors:

\[
\begin{align*}
\mathcal{X}^{i-1}[m] &= x_i[m] - \sum_{q=0}^{i-1} \langle x_i[m], x_q[m] \rangle x_q[m] \\
&= x_i[m] - \sum_{q=0}^{i-1} \frac{\langle x_i[m], x_q[m] \rangle}{\|x_q[m]\|^2} x_q[m] \\
&\quad \text{where the sum is defined as zero if } i=0.
\end{align*}
\]  

(5.51)

where one has to realize that these orthogonal vectors are in general not shifted versions of each other, in contrast to the input signal vectors. Following the Gram-Schmidt procedure (5.51) the original basis (5.50) can be rotated to an orthogonal basis, spanning the same space. This leads for \( i=0 \) to \( L-1 \) to:

\[
\mathcal{X}^i[m] = \sum_{q=0}^{i} \tau_{i[q]}[m] x_q[m] 
\]  

(5.52)

with \( \tau_{i[q]}[m] = 1 \) for all \( i \in \{0,1,\ldots,L-1\} \). In matrix notation this yields:

\[
\mathcal{X}^i[m] = X[m] \cdot \Gamma[m]
\]  

(5.53)

where the \( L \times L \) upper triangular matrix is defined as:
Chapter 5: Performance analysis

Now the BOP concept is to make a projection of the vector $g[m]$ on the orthogonal basis. This leads to:

\[ < g_l[m], X^\top[m] > = 0 \text{ for } i \in \{0, \ldots, L-1\} \]
\[ g^\top[m] = X^\top[m] \cdot g[m] \]

By defining a rotated (transformed) residual signal vector as:

\[ X^\top[m] = \Gamma^\top[m] \cdot X[m] \]
\[ = \Gamma^\top[m] \cdot X^\top[m] \cdot g[m] \]
\[ = \Gamma^\top[m] \cdot X^\top[m] \cdot X^\top[m] \cdot g[m] \]
\[ = (X^\top[m])^\top \cdot X^\top[m] \cdot g[m] \]
\[ = R^\top[m] \cdot g[m] \]

Analogously to the derivation of the BOP-equation in chapter two, the next Gram-Schmidt BOP update algorithm (GS-BOP) can be derived:

\[ y[m+1] = y[m] + 2 \cdot \alpha \cdot X^\top[m] \cdot (R^\top[m])^{-1} X^\top[m] \]
\[ = y[m] + 2 \cdot \alpha \cdot X^\top[m] \cdot X^\top[m] \cdot g[m] \]

where the $L \times L$ transformed inverse autocorrelation matrix $(R^\top[m])^{-1}$ is diagonal because of the orthogonality property of the vectors $X^\top_i[m]$. Thus:

\[ (R^\top[m])^{-1} = \text{diag}(\|X^\top_0[m]\|^2, \ldots, \|X^\top_{L-1}[m]\|^2) \]

Instead of the $L \times L$ inverse transformed autocorrelation matrix also the $L \times L$ identity matrix can be taken, if one chooses a transform to an orthonormal base by dividing for every $i$ all elements $\tau_{ai}[m]$ of $\Gamma[m]$ by $\|X^\top_i[m]\|$. This yields the Modified GS-BOP (MGS-BOP) algorithm:

\[ y[m+1] = y[m] + 2 \cdot \alpha \cdot X^\top[m] \cdot I_{LL} \cdot X^\top[m] \]
\[ = y[m] + 2 \cdot \alpha \cdot X^\top[m] \cdot X^\top[m] \]
Chapter 5: Performance analysis

with \( r'[m] = 1 \)

\[
\begin{pmatrix}
\tau_{a0}[m]/|x_0'[m]| & \tau_{a1}[m]/|x_1'[m]| & \ldots & \tau_{aL-1}[m]/|x_{L-1}'[m]| \\
0 & \tau_{l0}[m]/|x_0'[m]| & \ldots & \tau_{lL-1}[m]/|x_{L-1}'[m]| \\
& & \ddots & \vdots \\
& & & \ddots & \ddots \\
0 & 0 & \ldots & \tau_{L-1L-1}[m]/|x_{L-1}'[m]| \\
\end{pmatrix}
\] (5.63)

\[
x'[m] = x[m] \cdot r'[m]
\] (5.64)

\[
x'[m] = (r'[m])^T \cdot x'[m]
\] (5.65)

If \( s[k] \) is zero then:

\[
x'[m] = (r'[m])^T \cdot x'[m] \cdot d[m]
= (x'[m])^T \cdot d[m]
\] (5.66)

The behaviour of the MGS-BOP algorithm is the same as the behaviour of the GS-BOP algorithm. The GS-BOP algorithm has probably a lower complexity, the MGS-BOP algorithm however, fits perfectly in the general orthogonal transformation theory presented in the next paragraph. Equation (5.66) is of course not really implemented in the filter as the input to the filter is not transformed, but only the input to the update part! This equation however is needed for the general orthogonal transform theory of the next paragraph.

Of course the (H)GS-BOP can also be transformed to a version using the sliding approach by replacing the block-index \( m \) by the sample-index \( k \).
5.6 General orthogonal transformation

A strong resemblance between the BOP analysed with SVD of eq. (5.28) and the MGS-BOP of eq. (5.61) can be observed. The first algorithm replaces the data-matrix $X[m]$ by the matrix $U[m]$ containing its left singular vectors (an orthonormal matrix) and using only $L$ of them (corresponding to the $L$ non-zero eigenvalues of $X[m] \cdot X[m]^T$). The second algorithm transforms the data-matrix $X[k]$ to an orthonormal matrix $X'[m]$.

Both the algorithms can be taken into one equation by defining a transformation $T$, transforming the basis (5.50) (the matrix $X[m]$) into an orthonormal basis ($X'[m]$ or the first $L$ columns of $U[m]$). This yields the next equation for the General BOP (GBOP) case (with $s[k] = 0$):

$$ x[m+1] = x[m] + 2 \cdot \alpha \cdot T(X[m]) \cdot (T(X[m]))^T \cdot g[m] $$

(5.67)

If one looks closely at equation (5.67) one observes directly that this is in fact the update equation of the BLMS- and also BNLMS-algorithm, with in the latter case the matrix $X[m]$ replaced by its transformed version (remembering that for the BNLMS case $z[m] = X[m]^T \cdot g[m]$, a transform $T(X[m]) = X[m] \cdot \text{diag}(\|z_0[m]\|^{-1}, \ldots, \|z_L[m]\|^{-1})$ can be introduced).

In eq. (5.67) the effect of the noise signal $s[k]$ is not yet taken into account. This noise signal can be re-introduced:

$$ x[m] = X'[m] \cdot g[m] + g[m] = X'[m] \cdot (g[m] + X[m] \cdot (X'[m] \cdot X[m])^{-1} \cdot g[m]) $$

(5.68)

By defining:

$$ g'[m] = g[m] + X[m] \cdot (X'[m] \cdot X[m])^{-1} \cdot g[m] $$

(5.69)

and filling in $g'[m]$ instead of $g[m]$ in (5.67) the equation is completed.

This all can also be done for the sliding block OP algorithms, which will yield the same result with the block index $m$ replaced by the sample index $k$.

More information about general orthogonal transforms for noise canceller structures can be found in [AMI88].
Chapter 6

Influence of echo-path

6.1 Introduction

Variation of the impulse response of the echo-path in the echo-canceller structure does influence the convergence behaviour of the update algorithm of the echo canceller. This can be concluded from the results of paragraph 3.3. In [FEU85] the influence of the echo-path impulse-response (and the initial adaptive filter tap-values) on the convergence behaviour of the BLMS algorithm is studied. In order to make it possible for the adaptive filter to imitate the echo path, the echo path impulse response $h$ is assumed to have a finite impulse response length of at most $N$ (number of taps of the adaptive filter).
6.2 Spectral view

In paragraph 5.5 a definition is given of the pds of a signal \( x[k] \). For the echo-path impulse response also a pds can be calculated. In figure 6.1 the power spectral density functions for diverse MA(1) signals (see chapter three) are drawn. The signals are defined as:

\[
x[k] = \sqrt{(1 - a^2)} \cdot n[k] + a \cdot n[k-1]
\]

(6.1)

with \( n[k] \) a white noise signal. An echo-path impulse response of eq. (6.2) yields the same pds (\( h \) is a two element vector):

\[
h = \{ h_0, h_1 \}^T = \{ \sqrt{(1 - a^2)}, a \}^T
\]

(6.2)

Figure 6.1: Power spectral density for MA(1) signals.
Chapter 6: Influence of echo-path

The adaptive filter has to imitate the echo-path, in other words: the pds of its impulse response has to converge towards the pds of the echo-path impulse response. The adaptive filter impulse response is adapted by the input signal vectors multiplied by the error values. Every subband of the pds of the adaptive filter impulse response is adapted by the same subband of the input signal pds.

If for a certain (very small) subband the value of the pds of the echo-path is relatively small and the value of the pds of the input-signal is relatively large, then that (very small) part of the pds of the echo-path will be imitated quickly (but inaccurately) by the adaptive filter pds.

If however the reverse is the case, the value of the pds of the echo-path for a certain (very small) subband is relatively large and the pds of the input signal is relatively small, then that part of the pds of the echo-path will take more time to be imitated. That imitation however, will be more accurate.

Normalising the subbands yields the (P)FDAF algorithm to decorrelate the input signal (see the previous chapter and [WIL91]).
Chapter 6: Influence of echo-path

6.3 Experiments

The effect of the echo-path impulse response can clearly be observed in the next experiment. For the input signal a MA(1) signal with the parameter \( \alpha \) equal to \( \frac{1}{2} \) is chosen (see figure 6.1). The echo-path impulse response is chosen to have the same pds as the input signal. Two experiments are carried out, one with no additional noise, the other with \( \sigma^2 = 0.01 \). In both cases the following choice for the other parameters is made: \( N = 16 \) and \( \alpha = 0.04 \). The results of the two experiments are shown in figures 6.2 and 6.3.

![Figure 6.2: Effect of echo-path without additional noise.](image)

The pds for the signal \( x[k] \) and the pds for the echo-path impulse response do match, meaning that where the pds of the echo-path is large also the pds of \( x[k] \) is large, resulting in a faster (initial) rate of convergence for a large amount of power. This effect can be observed clearly in figure 6.2. Decorrelation yields a (initially) slower algorithm. As soon as the average squared error resulting from the frequencies where both pds's are large is
Chapter 6: Influence of echo-path

reduced to the level of the (initially much smaller) average squared error resulting from the frequencies where both pds's are small, the smaller rate of convergence for that part will get more important, resulting in a slower algorithm.

![Figure 6.3: Effect of echo-path with additional noise.](image)

If the near-end ('noise') signal $s[k]$ is introduced and a desired signal to noise ratio is chosen, an optimal value for $L$ can be chosen. If for example a signal to noise ratio of 5 dB is desired, a final misadjustment of 25 dB is needed. In this case no decorrelation would yield an algorithm of low complexity and optimal convergence behaviour. Decorrelation therefore does not always yield better results!
6.4 Theoretical model

In the previous paragraphs some effects of the echo-path impulse response on the convergence behaviour of the BOP algorithm are shown. The combination of the input-signal psd, the echo-path impulse response psd and the decorrelation parameter $L$ together determine with the other parameters the convergence behaviour of the algorithm. This paragraph introduces a possible method of deriving (theoretically) the convergence behaviour of a (N)LMS algorithm from the psd of the input-signal and the psd of the echo-path impulse response. The next chapter will develop an (approximate) method to combine the input-signal psd and the parameter $L$ into a new psd.

In the previous paragraphs it was already mentioned that a certain subband of the adaptive filter impulse response psd is adapted by the corresponding subband of the input-signal psd. If the input-signal psd is flat, then the convergence behaviour of the NLMS algorithm is easy to calculate. If that psd however is not flat, one could try to split the psd of the input signal into $K$ subbands for which the psd is (almost) flat, see figure 6.4.

![Figure 6.4: Subband division of MA(1) signal.](image)
Chapter 6: Influence of echo-path

For every subband now a LMS algorithm (normalised with the power of the total input-signal) can be derived with a 'white' input signal by extending the (average) level of that certain subband over the whole spectrum. The weighted sum of the K convergence curves now yields one convergence curve for the whole algorithm (every curve is weighted by the width of its subband (divided by the spectrum width)). Every curve can be described by:

\[ 10^{-10} \log(\delta_i(k)) = 10^{-10} \log(\sigma_i^2 \cdot 10^k) \]
\[ = 10^{-10} \log(\sigma_i^2) + 10 \cdot k_i \]  

with:

\[ k_i = \frac{2 \cdot k}{\tau_{\alpha_i}} \]  
\[ \sigma_i^2 = \left( \frac{b_i}{\pi} \right) \cdot \text{pds}(i) \]  

in which \( b_i \) is the width of the subband \( i \), \( \text{pds}(i) \) is the average value of the \( \text{pds} \) in subband \( i \), \( k \) is the sample number and \( \tau_{\alpha_i} \) is calculated according to (2.15), with \( \alpha_i = \alpha \cdot \text{pds}(i) \). In this case still the echo-path impulse response is assumed to be flat. Figure 6.5 gives an example, by dividing the MA(1) signal of figure 6.4 into three subbands with \( \alpha=0.0025 \):

- \( \text{pds}(1) = 1.82 \Rightarrow \sigma_1^2 = 0.61 \Rightarrow 10^{-10} \log(\sigma_1^2) = -2.2 \text{ dB} \)
- \( \text{pds}(2) = 1.00 \Rightarrow \sigma_2^2 = 0.33 \Rightarrow 10^{-10} \log(\sigma_2^2) = -4.8 \text{ dB} \)
- \( \text{pds}(3) = 0.164 \Rightarrow \sigma_3^2 = 0.055 \Rightarrow 10^{-10} \log(\sigma_3^2) = -12.6 \text{ dB} \)

\[ 10 \cdot k_1 = -0.0767 \cdot k \]
\[ 10 \cdot k_2 = -0.0419 \cdot k \]
\[ 10 \cdot k_3 = -0.0070 \cdot k \]

The \( \text{pds} \) of the echo-path impulse response can be brought in by multiplying the convergence curve for every subband of the input-signal by the level of the corresponding subband of the echo-path impulse response \( \text{pds} \):

\[ \sigma_i^2 = \left( \frac{b_i}{\pi} \right) \cdot \text{pds}_\alpha(i) \cdot \text{pds}_\beta(i) \]  

The rest of the procedure does not change, see figure 6.6, where an echo-path is chosen with an impulse response \( \text{pds} \) that is equal to the input-signal \( \text{pds} \) of figure 6.4.

By making the subbands infinitely small (integrating) the approximations made above can be made more accurate. If one approximates the B(N)LMS algorithm by a (N)LMS algorithm (with the correct parameter \( \alpha \), see [SOM90]), the above procedure can also be used for the B(N)LMS case.
Chapter 6: Influence of echo-path

Figure 6.5: Theoretical convergence curve for 3 subbands on MA(1) signal with $\alpha=0.0025$, no noise.

Figure 6.6: Convergence curve of MA(1) signal with MA(1) echo-path impulse response.
Chapter 7

Influence of dimension L

7.1 Introduction

In chapter five it was derived that the dimension L of the (estimated) autocorrelation matrix does influence the convergence behaviour of the BOP algorithm by the matrix $E\{X^T (X^T X)^{-1} X\}$. The behaviour of this matrix as a function of the parameter L and the input signal $x[k]$ will determine the convergence behaviour. At first therefore the influence of L on that matrix will be simulated and a method to calculate a new psd for a pseudo-input signal is presented. After that a second method to combine the effects of the parameter L and the input signal psd into one new pseudo-input-signal psd is introduced.
Chapter 7: Influence of dimension \( L \)

7.2 The matrix \( \mathbf{E}\{X(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\} \)

From chapter two it is known that the BOP update equation can be written as follows:

\[
\mathbf{y}[m+1] = \mathbf{y}[m] + 2\cdot\alpha\cdot\mathbf{X}[m]\cdot(\mathbf{X}^T[m]\cdot\mathbf{X}[m])^{-1}\cdot\mathbf{X}^T[m] \cdot \mathbf{d}[m] \tag{7.1}
\]

If it is possible to rewrite the matrix \( \mathbf{X}[m]\cdot(\mathbf{X}^T[m]\cdot\mathbf{X}[m])^{-1}\cdot\mathbf{X}^T[m] \) to a product of a matrix \( \mathbf{Y}[m] \) and its transpose:

\[
\mathbf{X}[m]\cdot(\mathbf{X}^T[m]\cdot\mathbf{X}[m])^{-1}\cdot\mathbf{X}^T[m] = \mathbf{Y}[m]\cdot\mathbf{Y}^T[m] \tag{7.2}
\]

then the equation (7.1) can be rewritten as:

\[
\mathbf{y}[m+1] = \mathbf{y}[m] + 2\cdot\alpha\cdot\mathbf{Y}[m]\cdot\mathbf{Y}^T[m] \cdot \mathbf{d}[m] \tag{7.3}
\]

This eq. (7.3) satisfies the general orthogonal transform theory of paragraph 5.6. Now a pseudo-autocorrelation matrix can be defined:

\[
\mathbf{R}' = \mathbf{E}\{\mathbf{Y}[m]\cdot\mathbf{Y}^T[m]\} = \mathbf{E}\{\mathbf{X}[m]\cdot(\mathbf{X}^T[m]\cdot\mathbf{X}[m])^{-1}\cdot\mathbf{X}^T[m]\} \tag{7.4}
\]

From [SOM90] is known that the elements of the autocorrelation matrix are determined by the autocorrelation function. Knowing this one could try to reverse the process of forming the autocorrelation matrix from the autocorrelation function. With the autocorrelation function formed this way, the psd of a fictive signal \( \mathbf{y}[k] \) can be calculated with help of the Wiener-Khintchine relation (5.50). With these results the process of chapter six can be used to determine the convergence behaviour of the algorithm.

One however should not forget that the matrix \( \mathbf{R}' \) is not a real autocorrelation matrix as the columns of the matrix \( \mathbf{Y}[m] \) are in general not shifted versions of each other. Therefore the matrix will be not Toeplitz (Toeplitz = on every diagonal the entries are equal), but just near-Toeplitz. The exact order of the deviation is not investigated here, but seems to be in the order of one percent.

The effects of the fact that the matrix is not Toeplitz can be decreased by taking the average of the diagonals to form the pseudo-autocorrelation function from the pseudo-autocorrelation matrix.
The elements of the function are then constructed by averaging the corresponding elements of the matrix. These approximations will introduce an error into the final result, that is not further investigated here. This method however is much more simple than the methods of chapter five to determine the convergence behaviour of the BOP adaptive filter algorithm and is therefore presented here.

A simulation with this method on the MA(1) signal of the previous chapter yields the following results for the autocorrelation functions for diverse values of L of the signal $y[k]$ (N=16):

<table>
<thead>
<tr>
<th>$L$</th>
<th>$p_0$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$p_5$</th>
<th>$p_6$</th>
<th>$p_7$</th>
<th>$p_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>0.50</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>0.33</td>
<td>0.15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>0.20</td>
<td>0.13</td>
<td>0.09</td>
<td>0.07</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>0.12</td>
<td>0.09</td>
<td>0.07</td>
<td>0.06</td>
<td>0.05</td>
<td>0.04</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>16</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 7.1: Autocorrelation functions for diverse values of parameter $L$.

For $L=1$ the original function is retrieved and for full decorrelation ($L=16$) a delta pulse is the result. Increasing $L$ does make the autocorrelation function $p_0(t)$ 'wider', but does decrease the values for $t$ not equal to zero. Increasing $L$ will therefore smoothen the signal as can be seen in figure 7.1 where the corresponding psd's are drawn.

Figure 7.1: Psd's of MA(1) signal for diverse decorrelations
Chapter 7: Influence of dimension L

7.3 Influence of L on psd

In chapter five is derived that a way of looking at the mechanism of the BOP algorithm is to normalize L subbands of the psd of the input-signal. This can be used to develop a second method of constructing the convergence characteristics from the input-signal psd. With help of the theory of paragraph 6.4 the convergence curve of the (B)(N)LMS algorithm can be derived from the psd of the input-signal and the psd of the echo-path impulse response. Paragraph 6.5 introduces a frequency domain interpretation of the BOP algorithm that is in fact a BLMS algorithm with a transformed input-signal. This transformation is done by normalising L subbands of that signal. This transformed signal spectrum can be used in the method of the previous chapter as is illustrated in figures 7.1 and 7.2. Using this method it is best to make the number of subbands K as large as possible (infinitely large?), in order to have the best approximation. For illustration purposes this is not done here.

\[ \begin{align*}
&0 & \frac{\pi}{3} & \frac{\pi}{2} & \frac{2\pi}{3} & \pi & \frac{4\pi}{3} & \frac{3\pi}{2} & \frac{5\pi}{3} & 2\pi \\
&3 & 1.5 & & & & & & & \\
\end{align*} \]

Figure 7.1: Subband division of normalized spectrum (L=2, K=6).
Figure 7.2: Theoretical convergence curve of MA(1) signal (L=2, K=6).
Chapter 8

Conclusions

In this report diverse interpretations of the Block Orthogonal Projection ((B)OP) algorithm were presented. These interpretations give insight into the convergence behaviour of the (B)OP algorithm as a function of its parameters, such as the dimension $L$ of the autocorrelation-matrix, and the echo-path impulse response.

Interpretation of the BOP algorithm with the Singular Value Decomposition (SVD) theory simplifies the analysis of the algorithm considerably compared to the analysis of the not-transformed interpretation.

The frequency domain interpretation and the orthogonal transformation theory yield, together with the convergence characteristics of the Normalized Least Mean Square (NLMS) algorithm for white noise input signals, a simple method to derive the convergence curves for the (B)OP algorithm. The results of this method still have to be compared to 'practical' results (simulations) in order to estimate the quality of this method.

The implementation-complexity of the (B)OP algorithm can possibly be reduced using the frequency domain interpretation or the matrix inversion lemma for the time domain interpretation.

Further investigations on the Block Orthogonal Projection algorithm in order to find a method for estimating an optimal value for the dimension of the autocorrelation-matrix can be performed with the help of one or more of the interpretations of the algorithm presented in this report.
Appendix A: SPOX(/DOS)

At first two quotations from [SPO90a] and [SPO90b] are presented to introduce SPOX/DOS and SPOX:

"SPOX/DOS [SPO90a] is an implementation of the SPOX ([SPO90a], [SPO90b], [SPO90c]) operating system for the PC (PC, PC-AT, PS/2) with MS-DOS. SPOX applications can be developed, debugged and tested on the PC using SPOX/DOS and the Microsoft development tools before being ported to other processors."

"SPOX is an integrated system of portable software components targeted for high-end DSP (Digital Signal Processor) microprocessors such as the Texas Instruments TMS320C30. SPOX presents a high-level software interface to the underlying DSP hardware, improving the productivity of application developers as well as enhancing program portability from one hardware platform to the next."

In [SPO90b, page 1-3] the main features of SPOX are listed. As SPOX provides vector math operations, matrix math operations and linear filtering, it seems perfectly (?) fit for implementation of adaptive filters. For more information on the above mentioned features, see [SPO90c].

Some notes have to be made on using SPOX(/DOS). Although the programming guide seems rather simple, one will encounter some "strange" things programming in SPOX. Especially regarding memory management one has to be very careful with using variables and always declare space for them in the main program. One also must never pass actual numbers as an argument to a SPOX function, but always a variable that contains that number. For more information on SPOX(/DOS) see [WIT91]. At the moment of writing this appendix still portability between SPOX/DOS and the SPOX system using the TMS320C30 is not guaranteed.
Appendix B: SPOX(/DOS) program listing

```c
/* opf6.c
  * orthogonal projection
  */

#include <spox.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>

#define EPSILON 1E-100
#define BUFSIZE 100
#define SQROOT3 sqrt(3)

Char buf[BUFSIZE];  /* buffer for ungetch */
Int bufp = 0;  /* next free position in buf */

typedef struct {
  Int l;
  Int n;
  Int m;
  Int f;
} Paramtp;

typedef Paramtp *Paramptr;

typedef struct {
  Uns x[7];
  Uns s[7];
} Seedstp;

typedef Seedstp *Seedsptr;

Int getch()
{
  return ((bufp > 0) ? buf[--bufp] : getchar());
}

Void ungetch( un_c )
Char un_c;
if (bufp >= BUFSIZE)
  printf( "ungetch: too many characters\n" );
else
  buf[bufp++] = un_c;
}

Int getint( gi_inptr )
Int *gi_inptr;
{
  Int gi_c, gi_sign;
  while (isspace( gi_c = getch() ))
  /* skip white space */
  if (!isdigit( gi_c ) && gi_c != EOF && gi_c != '+' && gi_c != '-')
    /* it's not a number, so skip & retry */
    return 0;

  gi_sign = (gi_c == '-') ? -1 : 1;
  if (gi_c == 'f' || gi_c == 'F')
    gi_c = getch();
  for ( *gi_inptr = 0 ; isdigit( gi_c ) ; gi_c = getch() )
```
Appendix B: SPOX-program listing

```c
while (isspace( gf_c = getch() )
1 /* skip white space */
if (!isdigit( gf_c ) && gf_c != EOF && gf_c != '+' && gf_c != '-')
1 /* it's not a number, so skip & retry */
return 0;
}
gi_sign = (gf_c == '‑') ? -1 : 1;
if (gf_c == '+' || gf_c == '‑')
gf_c = getc();
for (*gl_inptr = 0; isdigit( gl_c ) ; gl_c = getc() )
*gl_inptr = 10 * *gl_inptr + (gl_c - '0');
if (gl_c != EOF)
ungetc( gl_c );
return gl_c;
}
93
94 Int
getfloat( gf_flptr )
Float
*gf_flptr;
96 {
97 Int
gf_c , gf_sign , gf_exp , gf_aint;
98 Float
gf_fac;
99
while (isspace( gf_c = getc() )
1 /* skip white space */
if (!isdigit( gf_c ) && gf_c != EOF && gf_c != '+' && gf_c != '‑' &&
gf_c != 'E' )
/* it's not a float, so skip & retry */
return 0;
}

gf_sign = (gf_c == '‑') ? -1 : 1;
if (gf_c == '+' || gf_c == '‑')
gf_c = getc();
for ( ; isdigit( gf_c ) ; gf_c = getc() )
*gf_flptr = 10 * *gf_flptr + (gf_c - '0');
if (gf_c == '.')
for ( gf_fac = .1 , gf_c = getc() ; isdigit( gf_c ) ; gf_c =
getc() , gf_fac /= 10 )
*gf_flptr *= (gf_c - '0') * gf_fac;
if ( (gf_aint = getint( &gf_exp )) != EOF || gf_aint == 0)
ungetc( gf_c );
else
*gf_flptr *= pow( 10 , gf_exp );
else
if (gf_c != EOF)
ungetc( gf_c );
*gf_flptr *= gf_sign;
```
Appendix B: SPOX-program listing

126     return gf_c;
127 }
128
129 Void  matpr( ma_matptr )
130 SM_Matrix *ma_matptr;
131 {
132     Int    ma_i , ma_j;
133
134     printf( "\n" );
135     for ( ma_i = 0 ; ma_i < SM_getlength( *ma_matptr , SM_COL ) ;
136                ma_i++ ) {
137         for ( ma_j = 0 ; ma_j < SM_getlength( *ma_matptr , SM_ROW ) ;
138             ma_j++ )
139             printf( "%f " , *(Float *)SM_loc( *ma_matptr , ma_i , ma_j ) );
140     printf( " done\n" );
141 }
142
143 Void  vecpr( ve_vecptr )
144 SV_Vector *ve_vecptr;
145 {
146     Int    ve_i;
147
148     printf( "\n" );
149     for ( ve_i = 0 ; ve_i < SV_getlength( *ve_vecptr ) ; ve_i++ )
150     printf( "%f " , *(Float *)SV_loc( *ve_vecptr , ve_i ) );
151     printf( " done\n" );
152 }
153
154 Void  rin2( ri_matptr )
155 SM_Matrix *ri_matptr;
156 {
157     Int    ri_dim , ri_i , ri_j , ri_temin;
158     SM_Cursor ri_acur , ri_bcur;
159     SV_Vector ri_avec , ri_bvec;
160     Float  ri_aptr;
161     Float  ri_fac , ri_temfl;
162     /* initialising of inverse matrix to identity matrix */
163     ri_dim = SM_getlength( *ri_matptr , SM_COL );
164     SM_setlength( *ri_matptr , SM_ROW , ri_dim );
165     SM_setbase( *ri_matptr , SM_ROW , ri_dim );
166     ri_temfl = 0;
167     ri_temin = 0;
168     SM_fill( *ri_matptr , ri_temfl );
169     for ( ri_i = 0 ; ri_i < ri_dim ; ri_i++ ) *(Float *)SM_loc( *ri_matptr , ri_i , ri_i ) = 1;
170     SM_setbase( *ri_matptr , SM_ROW , ri_temin );
171     SM_setlength( *ri_matptr , SM_ROW , 2*ri_dim );
172     /* no pivoting is carried out as the matrix will be diagonally
173        dominant
174        * for the OP-problem! */
175     for ( ri_i = 0 ; ri_i < ri_dim ; ri_i++ ) {
176         ri_aptr = *(Float *)SV_loc( *ri_bvec , ri_i );
177     }
Appendix B: SPOX-program listing

187 if ((*ri_aptr < EPSILON) && (*ri_aptr > -EPSILON)) {
188     printf("Underflow\n");
189     exit(0);
190 }
191 else {
192     SM_muls( ri_bvec , ri_bvec , ri_fac / *ri_aptr );
193     SM_sub2( ri_avec , ri_bvec );
194 }
195     /* matpr( ri_matptr ); */
196     */
197 }
198 }
199 SM_scan( *ri_matptr , SM Row , &ri_acur );
200 for ( ri_i = 0 ; ri_i < ri_dim ; ri_i++ ) {
201     ri_avec = SM_next( &ri_acur );
202     SM_divs( ri_avec , ri_avec , *(Float *)SM_loc( ri_avec , ri_i ) );
203 }
204 }
205
206 SF_Filter crfilter();
207 {
208 SF_Filter cr_filt;
209 SA_Array cr_aarray , cr_barray , cr_carray , cr_darray;
210 SV_Vector cr_avec , cr_bvec;
211 Int cr_i , cr_n , cr_d , cr_aint;
212 SV_Cursor cr_acur , cr_bcur;
213 Float cr_afl , cr_tfl;
214
215 printf( "\nCreate filter:\n" );
216 printf( "\nN-1 \n" );
217 printf( "P(z) = A(z)/P(z) with A(z) = S num[n]*z \n" );
218 printf( "n=0 \n" );
219 printf( "D-1 \n" );
220 printf( "and P(z) = S denom[n]*z \n" );
221 printf( "n=0 \n" );
222 do
223     printf( "Type order of numerator A(z) (N): " );
224     while ( cr_aint = getint( &cr_n ) == EOF || cr_aint == 0 );
225     do
226     printf( "Type order of denominator P(z) (N): " );
227     while ( cr_aint = getint( &cr_d ) == EOF || cr_aint == 0 );
228     cr_aarray = SA_create( SG FAST , (cr_n + 1) * sizeof( Float ) ,
229     NULL );
230     cr_carray = SA_create( SG FAST , (cr_n + 1) * sizeof( Float ) ,
231     NULL );
232     cr_barray = SA_create( SG FAST , (cr_d + 1) * sizeof( Float ) ,
233     NULL );
234     cr_darray = SA_create( SG FAST , (cr_d + 1) * sizeof( Float ) ,
235     NULL );
236     cr_avec = SV_create( FLOAT , cr_aarray , NULL );
237     cr_bvec = SV_create( FLOAT , cr_barray , NULL );
238     for ( cr_i = 0 ; cr_i <= cr_n ; cr_i++ ) {
239     do
240     printf( "Type num[%i]:" , cr_i );
241     while ( cr_aint = getfloat( &cr_afl ) == EOF || cr_aint == 0 );
242     *(Float *)SM_next( &cr_acur ) = cr_afl;
243 }
244     SV_scan( cr_bvec , &cr_acur );
245     SV_scan( cr_avec , &cr_bcur );
246     for ( cr_i = 0 ; cr_i <= cr_d ; cr_i++ ) {
do
    printf( "Type denom[%d]:", cr_i );
    while
    cr_f1 = cr_afl;
    *(Float *)S_Next( &cr_acur ) = cr_afl / cr_f1;
    *(Float *)S Next( &cr_bcur ) /= cr_f1;
    cr_filt = SF_create( cr_carray , cr_darray , NULL );
    SF bind( cr_filt , cr_avec , cr_bvec );
    return cr_filt;
}
SF Filter
getxfilt()
{
 printf( "\nChoose form of input-signal:\n" );
 printf( " X[z] = F[z]*N[z] with n[k] is white noise.\n" );
 printf( "Choose numerator 1 (order 0) for AR-signal,\n" );
 printf( "choose denominator 1 for MA-signal,\n" );
 printf( "choose both numerator and denominator 1 for white noise.\n" );
 return crfilter();
}
SF Filter
gethfilt()
{
 printf( "\nChoose transfer-function F(z) of echopad.\n" );
 return crfilter();
}
SF Filter
getxfilt()
{
 printf( "\nChoose form of noise-signal:\n" );
 printf( " S(z) = F(z)*N(z) with n[k] is white noise.\n" );
 printf( "Choose numerator 1 (order 0) for AR-signal,\n" );
 printf( "choose denominator 1 for MA-signal,\n" );
 printf( "choose both numerator and denominator 1 for white noise.\n" );
 return crfilter();
}
Paramptr
cr_p= (Paramptr) malloc( sizeof( Paramtp ) );
printf( "\n" );
do
    printf( "Type number of coefficients (order minus 1) n of adaptive filter w:" );
    while
    ((cr_p = getint( &(cr_p).n ) == EOF || cr_p = 0);
    do
        printf( "Type dimension l of decorrelation matrix:" );
        while
        ((cr_p = getint( &(cr_p).l ) == EOF || cr_p = 0);
        do
            printf( "Number of rows f of X-matrix:" );
            while
            ((cr_p = getint( &(cr_p).f ) == EOF || cr_p = 0);
Appendix B: SPOX-program listing

```c
printf( "Type number of samples m to process: " );
while ((gp_aaint = getint( &(*gp_parptr).m )) == EOF || gp_aaint == 0);
return gp_parptr;

Seedsptr getseeds()
{
    Int gt_aaint;
    Seedsptr gt_seed;
    gt_seed = (Seedsptr) malloc( sizeof( Seedstp ) );
    (*gt_seed).x[0] = 1234598754;
    (*gt_seed).x[1] = 657893546;
    (*gt_seed).s[0] = 43281000;
    (*gt_seed).s[1] = 1467634822;
    return ( gt_seed );
}

Float getalfa()
{
    Int ga_aaint;
    Float ga_alfa;
    printf( "ln " );
    do
        printf( "Type alfa (stepsize parameter): " );
        while ((ga_aaint = getfloat( &ga_alfa )) == EOF || ga_aaint == 0 );
    return ga_alfa;
}

Float getrand( gr_s )
{
    Uns *gr_s;
    Float gr_f1;
    Long gr_v;
    gr_v = (((*gr_s) >> 16 ^ (*((gr_s + 3)))) & 0x07FFFF)
             | (*(*gr_s) ^ ((gr_s + 1)) >> 17);
    *(gr_s + 1) = (((*gr_s + 1)) << 15 & 0xFFFF8000)
             | (0x07FFFF & (*(gr_s + 2)) >> 17);
    *(gr_s + 2) = (((*gr_s + 2)) << 15 & 0xFFFF8000)
             | (0x07FFFF & (*(gr_s + 3)) >> 17);
    *(gr_s + 3) = (((*gr_s + 3)) << 15 & 0xFFFF8000)
             | gr_v;
    gr_v = (((((*gr_s + 4)) >> 8) ^ (((*gr_s + 5)) << 11 & 0xFFFF8000)
             | (((*gr_s + 6)) >> 21 & 0x0007FF)) & 0x0001FFFF;
    *(gr_s + 4) = (((*gr_s + 4)) << 17 & 0xFFE00000)
             | (0x0001FFFF & *(gr_s + 5)) >> 15);
    *(gr_s + 5) = (((*gr_s + 5)) << 17 & 0xFFE00000)
             | (0x0001FFFF & *(gr_s + 6)) >> 15);
    *(gr_s + 6) = (((*gr_s + 6)) << 17 & 0xFFE00000)
             | gr_v;
    gr_v = (((*gr_s + 1)) ^ ((gr_s + 6))) % 16777216;
```

Appendix B: SPOX-program listing

```c
/* initialisation of parameters */
{
    gr_fl = (Float) gr_v;
    gr_fl /= 16777216.0;
    return( 2 * SQROOT3 * (gr_fl - 0.5) );
} /* random spread over [-sqrt(3),+sqrt(3)] */

Void smain()
{
    Paramptr parptr;
    Seedsptr seedptr;
    Int i, j, k, max;
    Float alfa, tafl, tbfl, tcfl;
    Ptr atr;

    SM_Matrix mat, xmat, ymat;
    SA_Array marray, xarray, yarray;
    SH_Attrs attr;
    SM_View matview;
    SM_Cursor xmcurs, ymcurs, rcolcurs, yacolcurs, ybcolcurs;
    SF_Filter xfilt, hfilt, sfilt, wfilt;
    SA_Array delarray;
    SV_Vector wnvec, talvec, tblvec, tanvec, tbnvec, tnfvec, tbnvec, tucvec, evvec;
    SA_Array wnarray, talarray, tblarray, tnfarray, tbnarray, tanarray, tbnarray, earray;
    SV_Cursor ecur, flicurs;
    SV_View vecview;
    SS_Stream output;
    SA_Array oarray;
    String str, stu;

    /* initialisation of parameters */
    {
        xfilt = getxfilt();
        hfilt = gethfilt();
        sfilt = getsfilt();
        parptr = getparam();
        seedptr = getseeds();
        alfa = 2 * getalfa();
    }

    /* creation of autocorrelation matrix (and its inverse) frame */
    {
        marray = SA_create( SG_FAST, 2 * (*parptr).l * (*parptr).l *
            sizeof( Float ), NULL );
        matview.base[ SM_ROW ] = (*parptr).l;
        matview.stride[ SM_ROW ] = 0;
        matview.base[ SM_COL ] = 1;
        matview.length[ SM_ROW ] = (*parptr).l;
        matview.length[ SM_COL ] = (*parptr).l;
        attr.view = &matview;
        mat = SM_create( FLOAT, marray, (*parptr).l, 2 * (*parptr).l, &attr );
    }

    /* creation and initialisation of xmat frame */
    {
        xarray = SA_create( SG_FAST, (*parptr).l * (*parptr).n *
            sizeof( Float ), NULL );
        matview.base[ SM_ROW ] = (*parptr).l - 1;
        matview.length[ SM_COL ] = (*parptr).n;
    }
```
Appendix B: SPOX-program listing

425 matview.stride[ SM_ROW ] = -1;
426 attr.view = &matview;
427 xmat = SM_create( FLOAT, xarray, (*parptr).n, (*parptr).l, &attr);
428 SM_scan( xmat, SM_COL, &xmcur );
429 }
430 /* creation and initialisation of ymat frame */
431 {
432 yarray = SA_create( SG_FAST, (*parptr).f * (*parptr).l * sizeof( Float), NULL);
433 matview.length[ SM_COL ] = (*parptr).f;
434 attr.view = &matview;
435 ymat = SM_create( FLOAT, yarray, (*parptr).f, (*parptr).l, &attr);
436 SM_scan( ymat, SM_COL, &ymcur );
437 }
438 /* creation of adaptive filter wfilt */
439 {
441 wnarray = SA_create( SG_FAST, (((*parptr).n > (*parptr).f) ? (*parptr).n : (*parptr).f) * sizeof( Float), NULL);
442 wnvec = SV_create( FLOAT, wnarray, NULL);
443 tal = 0;
444 SV_fill( wnvec, tal );
445 wfilt = SF_create( delarray, NULL, NULL);
446 SF_bind( wfilt, wnvec, NULL);
447 }
448 /* creation and initialisation of error vector evec */
449 {
450 earray = SA_create( SG_FAST, (*parptr).l * sizeof( Float), NULL);
451 evec = SV_create( FLOAT, earray, NULL);
452 SV_getview( evec, &vecview );
453 vecview.stride = -1;
454 vecview.base = (*parptr).l - 1;
455 SV_setview( evec, &vecview );
456 SV_scan( evec, &ecur );
457 }
458 /* creation of temporary vectors */
459 {
460 tanarray = SA_create( SG_FAST, (*parptr).n * sizeof( Float), NULL);
461 tanvec = SV_create( FLOAT, tanarray, NULL);
462 tbnarray = SA_create( SG_FAST, (*parptr).n * sizeof( Float), NULL);
463 tbnvec = SV_create( FLOAT, tbnarray, NULL);
464 talarray = SA_create( SG_FAST, (*parptr).l * sizeof( Float), NULL);
465 talvec = SV_create( FLOAT, talarray, NULL);
466 tblarray = SA_create( SG_FAST, (*parptr).l * sizeof( Float), NULL);
467 tblvec = SV_create( FLOAT, tblarray, NULL);
469 tnfvec = SV_create( FLOAT, tnfarray, NULL);
471 tbnfvec = SV_create( FLOAT, tbnfarray, NULL);
472 }
473 /* preparation of output-stream */
474 {
475 output = SS_create( SS_NULL, 0, -1, NULL);
476 SS_open( output, "\file:val.mat", SS_WRITE );
477 }
Appendix B: SPOX-program listing

477   oarray = SA_create( SS_memseg( output ) , 1 +
        SS_sizeof( output ) , NULL );
478   SA_setlength( oarray , SS_sizeof( output ) );
479   str = (String) SA_getbuf( oarray );
480   stu = str;
481 
482   /* main loop of program */
483   for( j = 1 , max = ((*parptr).m / (*parptr).1 ) * (*parptr).l;
     j <= max ; j++ ) {
484       /* get new x,s-values through filters */
485       { 
486         taf1 = getrand( &(*seedptr).x[0] );
487         SF_lapply( xfilt , &tafl , &tafl );
488         SF_lapply( hfilt , &tafl , &tf1 );
489         SF_lapply( wfilt , &tafl , &tf1 );
490         taf1 = getrand( &(*seedptr).s[0] );
491         SF_lapply( sfilt , &tafl , &tafl );
492         } 
493         /* calculate error-value and perform output */
494         { 
495           *(Float *)SV_next( &eeur ) = taf1 + tf1 - tcfl;
496           sprintf( "%16.16g", tf1 - tcfl );
497           stu = stu + 16;
498           if ( ((j * 16) % (SA_getlength( oarray ))) == 0 ) {
499             SS_put( output , oarray );
500             stu = str;
501         }
502       }
503       /* update x-matrix , y-matrix */
504       { 
505         SF_getstate( wfilt , tnfvec , NULL );
506         SV_setlength( tnfvec , (*parptr).n );
507         SV_assign( tnfvec , SM_next( &xmcur ) );
508         SV_setlength( tnfvec , (*parptr).f );
509         SV_assign( tnfvec , SM_next( &ymcur ) );
510       }
511       /* update of adaptive filter vector & calculation of inverse
        autoc.m. */
512       if ( j % (*parptr).l == 0 ) {
513         /* set views of error-vector and matrices correct */
514         { 
515           SV_getview( evvec , &vecview );
516           vecview.base = 0;
517           vecview.stride = +1;
518           SV_getview( evvec , &vecview );
519           SM_getview( xmat , &matview );
520           matview.base[ SM_ROW ] = 0;
521           matview.stride[ SMROW ] = +1;
522           SM_getview( xmat , &matview );
523           SM_getview( ymat , &matview );
524           matview.base[ SM_ROW ] = 0;
525           matview.stride[ SMROW ] = +1;
526           SM_getview( ymat , &matview );
527           SM_setbase( mat , SM_ROW , 0 );
528           SM_setlength( mat , SM_ROW , (*parptr).l );
529         }
530         /* calculate LxL estimate of autocorrelation matrix */
531         { 
532           SM_scan( ymat , SM_COL , &yacolcur );
533           for ( i = SM_scan( mat , SM_COL , &rcolcur ) ; i > 0 ; i-- ) {
534             tvec = SM_next( &yacolcur );
535             SM_scan( ymat , SM_COL , &bcolcur );
536             for ( k = SM_next( &rcolcur ) , &flcur ) ; k > 0 ;
537             k-- ) { 
538           
539         }  
540       }  
541     }
537    SV_ddot( tuvec , SM_next( &ybcolcur ) , aptr );
538    *(Float *)SV_next( &flcur ) = *(Float *)aptr;
539  }
540  }
541  }
542  /* inversion of matrix */
543  rin2( &mat );
544  /* setting view of mat to the inverse matrix */
545  {
546    SM_setlength( mat , SM_ROW , (*parptr).l );
547    SM_setbase( mat , SM_ROW , (*parptr).l );
548  }
549  /* update calculations */
550  {
551    SM_prodv( mat , evec , talvec );
552    SV_muls( talvec , tblvec , alfa );
553    SM_prodv( xmat , tblvec , tanvec );
554    SV_setlength( tnfvec , (((*parptr).n > (*parptr).f) ?
555                   (*parptr).n : (*parptr).f) );
556    tbnfvec = SF_getnum( wfilt );
557    SV_setlength( tbnfvec , (*parptr).n );
558    SV_add2( tanvec , tbnfvec );
559    SV_setlength( tbnfvec , (((*parptr).n > (*parptr).f) ?
560                   (*parptr).n : (*parptr).f) );
561    SF_bind( wfilt , tbnfvec , NULL );
562  }
563  /* resetting of views */
564  {
565    SM_setbase( mat , SM_ROW , 0 );
566    SV_getview( evec , &vecview );
567    vecview.stride = -1;
568    vecview.base = (*parptr).l - 1;
569    SV_setview( evec , &vecview );
570    SV_scan( evec , &ecur );
571    SM_getview( xmat , &matview );
572    matview.stride[ SM_ROW ] = -1;
573    matview.base[ SM_ROW ] = (*parptr).l - 1;
574    SM_getview( ymat , &matview );
575    SM_getview( ymat , &matview );
576    matview.stride[ SM_ROW ] = -1;
577    matview.base[ SM_ROW ] = (*parptr).l - 1;
578    SM_scan( xmat , SM_COL , &xmcur );
579    SM_scan( ymat , SM_COL , &ymcur );
580  }
581  }
Text conventions

1. Underlined lowercase letters are used to denote column vectors.
2. Boldfaced uppercase letters are used to denote matrices.
3. Time-dependent scalars, vectors and matrices are printed in italics (and given a time-index k, e.g.: c[k], X[k], or a block index m).
4. A subscript to a scalar or vector is used to denote a time delayed version of that scalar or vector.
5. A (first) subscript to a matrix is used to denote its format (e.g.: R_{NN} for the N x N autocorrelation-matrix), a (eventual) second subscript is used to denote weather a estimate is based on sample basis or on block basis. (A)_{p,q} denotes the element of A in the (p+1)th row and (q+1)th column.
6. The symbol \| \| denotes the Euclidean norm of the vector enclosed within.
7. The inverse of a (square, non-singular) matrix A is denoted by A^{-1}, the pseudo-inverse [HAY91] of a (not necessarily square) matrix B is denoted by B^{-1}.
8. Transposition of a vector or matrix is denoted by superscript T. Complex conjugation is denoted by superscript *. Hermitian transposition (that is complex conjugation and transposition combined) of a vector or matrix is denoted by superscript H.
9. The symbol A^T denotes the transpose of the (pseudo-)inverse of a nonsingular matrix A.
10. The symbol diag(\sigma_1, \sigma_2, ..., \sigma_N) denotes a diagonal matrix whose elements on the main diagonal equal \sigma_1, \sigma_2, ..., \sigma_N.
11. The statistical expectation operator is denoted by E\{ \}, where the quantity enclosed is the random variable, vector or matrix of interest. The variance of a signal s(k) is denoted by \sigma_s^2.
12. The inner product of two vectors \( \mathbf{x} \) and \( \mathbf{y} \), \( \langle \mathbf{x}, \mathbf{y} \rangle \), is defined as \( \mathbf{x}^T \cdot \mathbf{y} \) (a scalar). The outer product is defined as \( \mathbf{x} \cdot \mathbf{y}^T \) (a matrix).

13. The trace of a square matrix \( \mathbf{A} \) is denoted by \( \text{trace}(\mathbf{A}) \), it is defined as the sum of the diagonal elements of \( \mathbf{A} \).

14. The rank of a matrix \( \mathbf{A} \) is denoted by \( \text{rank}(\mathbf{A}) \), it is defined as the number of independent columns (or rows) of \( \mathbf{A} \).

15. The ensemble-averaged autocorrelation matrix of a random vector \( \mathbf{x}[k] \) of length \( N \) is defined by:

\[
\mathbf{R}_{nn} = \mathbb{E}\{\mathbf{x}[k] \cdot \mathbf{x}^T[k]\}
\]

16. The discrete-time Fourier transform (FTD) of a time function \( x[k] \) is denoted by \( F\{x[k]\} \) \( (X[w] = F\{x[k]\}) \). The inverse discrete-time Fourier transform of a frequency function \( X[w] \) is denoted by \( F^{-1}\{X[w]\} \).

17. The discrete discrete-time Fourier transform (DFT) of a periodic time function is denoted by \( D\{x[k]\} \) \( (X[m] = D\{x[k]\}) \). The inverse discrete discrete-time Fourier transform of a periodic function \( X[m] \) is denoted by \( D^{-1}\{X[m]\} \).

18. The minimum of two scalars \( k \) and \( l \) is denoted as \( \min(k,l) \).
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td>Auto-Regressive</td>
</tr>
<tr>
<td>ARMA</td>
<td>Auto-Regressive-Moving Average</td>
</tr>
<tr>
<td>BFDAF</td>
<td>Block Frequency Domain Adaptive Filtering</td>
</tr>
<tr>
<td>BPFDFAF</td>
<td>Block Partitioned Frequency Domain Adaptive Filtering</td>
</tr>
<tr>
<td>BLMS</td>
<td>Block Least Mean Squares</td>
</tr>
<tr>
<td>BNLMS</td>
<td>Block Normalized Least Mean Square</td>
</tr>
<tr>
<td>BOP</td>
<td>Block Orthogonal Projection</td>
</tr>
<tr>
<td>BRLS</td>
<td>Block Recursive Least Squares</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td>FDAF</td>
<td>Frequency Domain Adaptive Filtering</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>GBOP</td>
<td>Generalized Block Orthogonal Projection</td>
</tr>
<tr>
<td>GS-BOP</td>
<td>Gram-Schmidt Block Orthogonal Projection</td>
</tr>
<tr>
<td>LMS</td>
<td>Least Mean Square</td>
</tr>
<tr>
<td>MA</td>
<td>Moving Average</td>
</tr>
<tr>
<td>MGS-BOP</td>
<td>Modified Gram-Schmidt Block Orthogonal Projection</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean-Squared Error</td>
</tr>
<tr>
<td>NLMS</td>
<td>Normalized Least Mean Square</td>
</tr>
<tr>
<td>OP</td>
<td>Orthogonal Projection</td>
</tr>
<tr>
<td>psd</td>
<td>power density spectrum</td>
</tr>
<tr>
<td>psd</td>
<td>power spectral density</td>
</tr>
<tr>
<td>RLS</td>
<td>Recursive Least Squares</td>
</tr>
<tr>
<td>SVD</td>
<td>Singular Value Decomposition</td>
</tr>
</tbody>
</table>
Glossary

Principal symbols

\( \alpha \)  
Adaption constant.

\( d[k] \)  
Difference-channel \( h - y[k] \).

\( d^\perp[k] \)  
Orthogonal component of \( d[k] \).

\( d^\parallel[k] \)  
Parallel component of \( d[k] \).

\( \delta[k], \delta[m] \)  
Misadjustment, echo canceller quadratic output error.

\( \delta[\infty] \)  
Final misadjustment.

\( \tau_\infty \)  
Initial speed of convergence, number of samples needed for a 20 dB reduction in \( 10^{10\log(\delta[m])} \).

\( h \)  
Vector of length \( N \) of the tap-coefficients of the transversal-filter model of the echo-path.

\( k \)  
Time index.

\( L \)  
Dimension of decorrelation(-matrix), dimension of autocorrelation matrix.

\( I \)  
Identity matrix

\( m \)  
Block index.

\( N \)  
Number of taps of the adaptive filter, vector length.

\( r[k] \)  
Residual signal \( y[k] \cdot d[k] \).

\( R_{x\times x} \)  
Autocorrelation matrix of \( x[k] \) (here \( N \times N \)).

\( R_{x\times x}[m] \)  
LxL estimate of the autocorrelation matrix of \( x[k] \).

\( s[k] \)  
Random (white) noise signal.

\( \sigma \)  
Singular value.

\( \sigma^2 \)  
Variance.

\( \Sigma \)  
Diagonal matrix with at the diagonal in decreasing order the singular values.

\( T \)  
Input signal sample period.

\( U \)  
Square matrix with the left-singular vectors.

\( V \)  
Square matrix with the right-singular vectors.

\( w[k] \)  
Vector of coefficients of the taps of the adaptive filter at time \( k \cdot T \).

\( x[k] \)  
Input signal at time \( k \cdot T \).

\( X[m] \)  
Input signal matrix.
References

[GOL70] Golub, G.H. and Reinsch, C.
"Singular value decomposition and least squares problems".
1970.
page 403-420.

[GER79] Gerbronds, J.J.
"SVD in relation to KLT and PCA".

[CLA81] Clark, G.A. et al.
"Block Implementation of Adaptive Digital Filters".
1981.
page 744-752.

[HON84] Honig, M.L. and Messerschmitt D.G.
"Adaptive filters: structures, algorithms and applications".
The Kluwer international series in eng. ...: vol.: 1

[OZE84] Ozeki, K. and Umeda T.
"An adaptive Filtering Algorithm Using an Orthogonal Projection to an Affine Subspace and its properties".
page 19-27.
1984.
page 126-132.
References

[FEU85a] Feuer, A.
"Performance Analysis of the Block Least Mean Square Algorithm".
1985.
page 960-963.

[WID85] Widrow, B. and Stearns, S.D.
"Adaptive signal processing".
Prentice-Hall signal processing series: 8501.

[ALE86a] Alexander, S.T.
"Adaptive signal processing: theory and applications".

[HAY86] Haykin, S.
"Adaptive filter theory".

[KUR87] Kurosawa, K. and Furusawa, T.
"A geometric interpretation of adaptive algorithms".
In: Globecom conference, Tokyo, nov. 15-18 1987.
page 49.7.1-49.7.5.

[TRE87] Treichler, J.R. et al.
"Theory and design of adaptive filters".
Topics in digital signal processing.

[AMI88] Amin, M.G. and Allen Jr., F.D.
"A generalized noise canceller using orthogonal transformations".
In: J.L. Lacoume et al.
"Signal Processing IV: Theories and Applications".

[KER88] Kernighan, B.W. and Ritchie, D.M.
References

[STE88] Steinhardt, A.O.
"Householder Transforms in Signal Processing".
page 4-12.

"The orthogonal projection algorithm for block adaptive signal processing".
page 1059-1062.

[OHI89] Oh'Isahi, K. et al.
"A new block adaptation algorithm realizing rapid identification system".
page 976-979.

[PER89] Perez, H. and Tsujii, S.
"Adaptive filtering using orthogonal functions".
page 575-578.

[SOM89a] Sommen, P.C.W.
"On the Orthogonal Projection method for Adaptive Filters".
In: [MTN90].
page 283-290.

[SOM89b] Sommen, P.C.W. and Valburg, C.J. van
"Efficient realisation of adaptive filter using an Orthogonal Projection method".
Glasgow, Scotland: ICASSP 1989

[MTN90] Ed. by Kaashoek, M.A. et al.
Progress in systems and control theory 5.
References

[SOM90] Sommen, P.C.W.
"Tijddiscrete signaalbewerking (5L090)".
Dictaat van de Technische Universiteit Eindhoven, fac. Electro-
techniek, nr. 5739.

DN 80109-C.

[SPO90b] "SPOX Application Programming Guide SPOX/XDS, SPOX/DOS, SPOX/-
SUN Version".
DN 82064-0.

Version".
DN 82032-C.

[CLU91] Cluitmans, L.
"Een zoektocht naar algoritmen voor multilayer perceptrons
vanuit een hardware-perspectief".
Stageverslag van de Technische Universiteit Eindhoven, fac.
Electrotechniek, vakgr. Electronische Schakelingen.

[EGE91] Egelmeers, G.P.M.
"Implementation of an adaptive equalizer on the TMS320C25
Signal Processor".
Stageverslag van de Technische Universiteit Eindhoven, fac.
Electrotechniek, vakgr. Electronische Schakelingen.

[HAY91] Haykin, S.

[SUN91] Sunwoo, J.S. and Un C.K.
"Recursive modified Gram-Schmidt algorithm for linear phase
filtering".
References

[WIL91] Wilde, E. de
"On the influence of the partitioning method on the convergence behaviour of Partitioned Frequency Domain Adaptive Filters (PFDAF)".

[WIT91] Withagen, H.C.A.M.
"Evaluatie van een SPOX ontwerp omgeving".