Reducing computational complexity and memory requirements in adaptive filters

Derkx, R.M.M.

Award date:
2000

Link to publication
Reducing Computational Complexity and Memory Requirements in Adaptive Filters

R.M.M. Derkx

SPS 10-00

Verslag van een afstudeerproject verricht binnen de leerstoel Signal Processing Systems onder leiding van Dr.ir. P.C.W. Sommen and Dr.ir. G.P.M. Egelmeers (Philips Nat. Lab.) in de periode november 1999 – augustus 2000

Eindhoven, augustus 2000

De faculteit Elektrotechniek van de Technische Universiteit Eindhoven aanvaardt geen aansprakelijkheid voor de inhoud van dit verslag.
Masters Thesis

Reducing Computational Complexity and Memory Requirements in Adaptive Filters

R.M.M. Derkx

Technical University Eindhoven

August 21, 2000
Abstract

For adaptive filters with many taps, a short algorithmic delay and relatively low computational complexity, the Partitioned Block Frequency Domain Adaptive Filter (PBFDAF) is a good choice. However for practical implementations, the computational complexity is still very large, especially for an increasing number of partitions. A solution is presented which reduces the complexity of the PBFDAF without significant loss in performance.

Furthermore a post-processing device for the adaptive filter is discussed, which suppresses residual echos from the adaptive filter. When using this device, the length of the adaptive filter can be reduced, resulting in a lower computational complexity and smaller memory requirements. The performance of this device depends on accurately chosen acoustical parameters. An efficient tracking algorithm for estimating these parameters during the adaptation is presented.

Samenvatting

Voor adaptieve filters met veel coefficienten, korte algoritmische delay en een gunstige reken-complexiteit, blijkt het gepartitioneerde blok frequentie-domein adaptieve filter (PBFDAF) een goede keuze te zijn. Echter voor praktische implementaties is de reken-complexiteit toch nog vrij hoog, met name wanneer veel partities worden gebruikt. In dit rapport wordt een methode beschreven, waarmee de reken-complexiteit van de PBFDAF kan worden teruggebracht, zonder dat dit leidt tot verlies in kwaliteit.

# Contents

1 Introduction

1.1 Application of adaptive filters ..................................... 1
1.2 Problem definition .................................................. 3
1.3 Assumptions and definitions ........................................ 3
1.4 Properties of Adaptive Filters ...................................... 4
   1.4.1 Computational complexity .................................... 4
   1.4.2 Memory requirements ........................................... 5
   1.4.3 Performance measures .......................................... 6
1.5 Outline of the report ................................................ 8

2 Adaptive Filters

2.1 (N)LMS algorithm .................................................... 11
2.2 Block Frequency Domain algorithms ................................. 12
2.3 FBNLMS algorithm .................................................. 14
2.4 BFDAF algorithm .................................................... 15
2.5 Partitioned BFDAF algorithm ....................................... 17
   2.5.1 Algorithm implementation .................................. 17
   2.5.2 Memory requirements ......................................... 18
   2.5.3 Computational complexity .................................. 19
   2.5.4 Step-size bounds .............................................. 19

3 Efficient constraining mechanisms .................................. 21

3.1 The constraint operator ........................................... 21
3.2 Constraints in the BFDAF algorithm ................................ 23
   3.2.1 Unconstrained BFDAF ....................................... 23
   3.2.2 BFDAF constraint approximation ............................ 24
3.3 Constraints in the PBFDAF algorithm ............................................. 25
  3.3.1 Unconstrained PBFDAF ......................................................... 25
  3.3.2 Alternative PBFDAF ............................................................ 25
  3.3.3 Scheduled constrained PBFDAF ............................................. 26
  3.3.4 PBFDAF constraint approximations ...................................... 27
  3.3.5 Hybrid constrained PBFDAF ............................................... 28
  3.3.6 Reduced overlap PBFDAF .................................................. 28
  3.3.7 Unconstrained PBFDAF with Lattice .................................... 28

4 Constraint approximations .......................................................... 29
  4.1 Properties of constraint approximations .................................... 29
  4.2 Constraint window with a high slope ...................................... 31
    4.2.1 Derivation of the algorithm ......................................... 32
    4.2.2 Convolution of the mean value ...................................... 35
    4.2.3 Initial calculations ..................................................... 36
    4.2.4 Iterative calculations .................................................. 36
    4.2.5 Calculation of the convolution-result ................................ 37
  4.3 Proposed constraining mechanism .......................................... 37
  4.4 Complexity Analysis ........................................................... 38
  4.5 Simulations ........................................................................... 40
    4.5.1 Convergence characteristics for white Gaussian noise ........... 40
    4.5.2 Convergence characteristics for coloured noise .................. 41

5 Compensation method .................................................................... 43
  5.1 Drawbacks in Alternative PBFDAF .......................................... 43
  5.2 Compensation of wrap-around artifacts ................................... 44
    5.2.1 Circular wrap-around artifacts ...................................... 45
    5.2.2 Compensation method ..................................................... 46
    5.2.3 Construction of compensation signals ............................... 49
    5.2.4 Compensation of sinusoidal constrained partitions ............... 50
  5.3 Proposed constraining mechanism ............................................ 54
  5.4 Complexity Analysis .............................................................. 57
  5.5 Simulations ............................................................................ 59
    5.5.1 Influence of the compensation ......................................... 59
Chapter 1

Introduction

A specific application of the adaptive filter in the area of acoustics is discussed. Furthermore the assumptions and definitions are presented and we discuss some properties of adaptive filters. We conclude with the general outline of this report.

1.1 Application of adaptive filters

Adaptive filters can be very useful devices in several applications of digital signal processing. Examples of such applications are: channel equalization, array signal processing and noise and echo cancellation [17]. In this report we use the Acoustic Echo Canceller (AEC) as an example, although we note that not all the algorithms proposed are restricted to this specific application. Acoustic Echo Cancellers are used, for example, in full-duplex teleconferencing systems or in some systems for certain kinds of speech recognition. The structure of the AEC, used in this report, is depicted in Figure 1.1 and will be explained as part of a teleconferencing system.

![Figure 1.1: Use of the Adaptive Filter in an AEC.](image)

In a teleconferencing system with two speakers in two different rooms, an AEC is implemented twice. The speech signal from the far-end speaker $x[k]$ is generated in the
near-end room by the loudspeaker. The transmission between the loudspeaker and the microphone can be modelled by a room impulse response. The echo signal $e[k]$, picked up by the microphone, can be seen as a convolution of the signal $x[k]$ with this room impulse response. In the case that both parties are speaking (i.e. double-talk), the microphone also picks up a near-end speech signal $s[k]$. If there was no adaptive filter present, the signal $\hat{e}[k] = e[k] + s[k]$ would be directly passed to the far-end speaker. Besides the wanted near-end speech signal, the far-end speaker also hears echoes of his own speech signal. For the overall teleconferencing system, this results in a worse communication between the two speakers.

The adaptive filters reduces these unwanted echoes by modelling the real room impulse response with a Finite Impulse Response (FIR) filter. The filter-coefficients $w[k]$ of this FIR filter are updated by correlating the residual signal $r[k]$ with the input signal $x[k]$. By convolving the input signal $x[k]$ with the filter-coefficients $w[k]$, the adaptive filter estimates the unknown acoustic echo signal $e[k]$, indicated by $\hat{e}[k]$. This estimate of the echo is subtracted from the real echo. As a result, the echo level is reduced, leading to improved communication between the speakers.

Another important part of the Acoustic Echo Canceller is the Dynamic Echo and Noise Suppressor (DENS). Figure 1.2 shows an AEC in which the adaptive filter is used in combination with the DENS.

![Figure 1.2: Adaptive Filter in combination with the DENS.](image)

When the filter-length of the adaptive filter is relatively short with respect to the real room impulse response, this signal $r[k]$ still contains much residual echoes. These echoes are suppressed with the DENS. Therefore the DENS can be used to reduce the length of the adaptive filter. Hence we can achieve a reduction in computational complexity and memory requirements of the overall Acoustic Echo Canceller.
1.2 Problem definition

We first concentrate on the adaptive filter. When using adaptive filters for echo cancellation, the room impulse response is often requires several thousands of coefficients long, which results in a large computational complexity. In the past, many structures have been proposed to implement the adaptive filter efficiently. All these structures have certain advantages/disadvantages concerning computational complexity, memory requirements, algorithmic delay and convergence properties. Since we want to use adaptive filters, which have low memory requirements and low algorithmic delay, some adaptive structures are not suitable. Therefore we focus on a specific adaptive filter structure that already has low memory requirements, low algorithmic delay and good performance. As the computational complexity of this specific structure is relatively high, we mainly concentrate on reducing the computational complexity and maintaining the optimal performance.

Secondly we look at the Dynamic Echo and Noise Suppressor. In order to achieve a good performance, the DENS makes certain assumptions about some acoustical parameters. When using the DENS in different environments (e.g. different rooms), we would like to estimate these acoustical parameters. In this report we propose an adaptive parameter tracking algorithm so that in all situations, reduced computational complexity and memory requirements can be achieved within the overall system without degradation in performance.

1.3 Assumptions and definitions

All in- and output signals are real and discrete in time, \( x[k] \) denoting the signal \( x \) at time instance \( k \). We use uppercase symbols to denote frequency-domain variables and lowercase symbols for time-domain variables. A boldface font is used to denote matrices, while vectors are denoted by underlining fonts. Furthermore the superscripts \( t, * \) and \( h \) are used to denote the transpose, complex conjugate and the complex conjugate transpose (Hermitian) respectively. \( O^N \) and \( I^N \) represent a zero and identity matrix, the superscript indicating the dimension of the matrix. Generally, superscripts indicate the dimension and subscripts indicate the element number of the vector or matrix. As most of the vectors and matrices used in this report have dimension \( 2N \times 1 \) and \( 2N \times 2N \), a vector or matrix with omitted superscript indicates a vector of length \( 2N \) or a matrix with dimension \( 2N \times 2N \). The indices between square brackets (e.g. \( x[k] \)), indicate a time-index. \( \text{diag} \{ \} \) denotes the operator that transforms a \( 2N \times 1 \) vector into a \( 2N \times 2N \) matrix containing the vector coordinates as diagonal elements while other elements are zero. The mathematical expectation is denoted by \( E \{ \cdot \} \). A circular shift of the data over \( L \) positions in an \( 2N \) dimensional vector is carried out by the matrix:

\[
D^N_L = \begin{pmatrix}
O & I^{2N-L} \\
I^L & O
\end{pmatrix},
\]

in which the zero matrices \( O \) have appropriate dimensions. Note that \( D^N_0 = I^{2N} \). The data of an \( 2N \) dimensional vector can be mirrored with the \( 2N \times 2N \) mirrored matrix \( J^{2N} \).
that is defined as:

\[ J^{2N} = \begin{pmatrix} 0 & \cdots & 1 \\ \vdots \\ 1 & 0 \end{pmatrix} \]  

(1.2)

A reverse circular shift over \( L \) positions, in opposite direction to \( D^{2N}_L \), is carried out by the matrix

\[ J^{-1}_L^{2N} = \begin{pmatrix} J^L & 0 \\ 0 & J^{2N-L} \end{pmatrix} \]  

(1.3)

We note that \( J_0^{2N} = J^{2N} \).

Besides mathematical descriptions we also use figures based on signal processing blocks. The explanation of these blocks can be found in the glossary of this report. There also an overview is given of the symbols used in this report. The symbols are split into two parts. The first part is concerned with the frequency domain adaptive filters, while the symbols in the second part are used in the parameter estimation algorithm. We note that in this report we only use 50% overlap of block-data, also called the Half Overlap (HOL) of block-data. The FFTs and IFFTs used in this report can be calculated by means of a \( 2N \times 2N \) Fourier matrix \( \mathcal{F} \) and a \( 2N \times 2N \) Inverse Fourier Matrix \( \mathcal{F}^{-1} \). The elements of the Fourier Matrix \( \mathcal{F} \) are defined for all \( 0 \leq a < 2N \) and \( 0 \leq b < 2N \) as:

\[ (\mathcal{F}^{2N})_{a,b} = e^{-j \cdot 2\pi \frac{ab}{2N}}. \]  

(1.4)

The Inverse Fourier Matrix \( \mathcal{F}^{-1} \) is defined as: \( \mathcal{F}^{-1} = \frac{1}{2N} \mathcal{F}^H \).

### 1.4 Properties of Adaptive Filters

In order to make comparisons between the different implementations of the adaptive algorithms in this report, we introduce some performance criteria for adaptive filter algorithms. Although there are several properties of adaptive algorithms, only measures for resources and performance are introduced for using them in this report.

#### 1.4.1 Computational complexity

An important goal in this report is to decrease the computational complexity of the adaptive algorithms. In literature computational complexity is frequently measured by the number of multiplications in the algorithm, as practical implementations for multiplications are often more complex than additions and subtractions.

For most existing DSP architectures, the number of additions and subtractions are just as demanding in terms of clock-cycles. When algorithms are implemented on flexible
architectures (e.g. Application Specific ICs), the number of additions and subtractions in an algorithm are less important for minimization. The divisions are also a point of discussion, as the implementation complexity of this operation is higher compared to the multiplication. Therefore we should differentiate between multiplications and divisions. We also have to consider the case of trivial multiplications. While a multiplication by 1 is neglected in the analysis of computational complexity, multiplications by powers of two (e.g. 0.25, 0.5, 2, 4) can be implemented on fixed point architectures by a simple binary shift operation. For some floating point architectures this binary shift operation can be performed in the exponent.

The above considerations indicate that measuring the computational complexity is far from trivial with respect to different architectures. In order to obtain an accurate picture of the computational complexity, we consider the number of additions/subtractions ($A$), multiplications ($M$), divisions ($D$) and binary shifts ($S$) individually. In order to compare different algorithms, we simply add up the four types of operations, keeping in mind that for a good comparison in computational complexity, the architecture of the signal processing system is important.

For the straightforward implementation of the complex multiplication, one usually uses 4 real multiplications and 2 real additions. There is also the possibility to use a structure with only 3 multiplications and 5 additions [10]. However, as the first structure uses the lowest number of total operations, we implement the complex multiplication in this way.

The computational complexity of the FFT and the IFFT depends on the specific implementation. A split-radix implementation calculates the FFT operation very efficiently. As the data in our algorithms are real-valued, the IFFT/FFT can be further reduced in complexity. The number of multiplications and additions for this real-valued split-radix FFT algorithm is used for the comparisons in our complexity-analysis (Appendix A.1).

Finally we note that indexing operations (i.e. integer additions and subtractions) are neglected in the analysis of the computational complexity. Also it is assumed that load and store operations can be performed in parallel with arithmetic operations, although we note that on existing architectures, the number of loads and stores that can be performed is limited.

### 1.4.2 Memory requirements

Another aim of this report is to achieve algorithms that have low memory requirements. If ICs with embedded memory are used, the memory reduction is even more important than the reduction of the computational complexity. The reason for this is the fact that implementing memory needs a lot of IC area. We denote the memory requirements by the symbol $\Theta$ which represents the minimum number of memory locations needed by the algorithm.
1.4.3 Performance measures

To compare different types of adaptive filters, we make use of different input signals and impulse responses for our simulations. In all cases we perform simulations without the presence of the near-end speech signal \( s[k] \) (i.e. no double-talk). This is done because the double-talk problem is a problem inherent to adaptive filters and is not a topic of this report.

For most comparisons we use stationary white Gaussian and coloured noise with unit-variance and zero-mean, which enables us to calculate an ensemble average of different measurements. In the experiments with noise signals we average 20 independently performed measurements, as it leads to statistically smoother convergence curves. Averaging more measurements would lead to even smoother convergence curves.

The coloured noise is generated by passing zero-mean unit-variance white Gaussian noise through a bandpass colouring filter. The colouring filter used is adopted from Chan [4] and is described in z-domain by Equation 1.5.

\[
G(z) = 0.0899 - 0.4539z^{-2} + 0.7702z^{-4} - 0.4390z^{-6}.
\]  

(1.5)

It is noted that we could use different types of colouring filters for generating the coloured noise. The choice for using this type of colouring filter in our experiments is rather arbitrary.

The power spectral density function (psdf) \( P(e^{j\theta}) \) has a shape as shown in Figure 1.3.

![Psdf of G(z)](image-url)

Figure 1.3: Psdf for the signal coloured with \( G(z) \).

In some cases we perform simulations with real non-stationary highly correlated music signals. These simulations are performed in order to test the adaptive filter in real-life circumstances.

All the simulations in this report use an artificial impulse response. This impulse response is an exponentially decaying randomly generated signal, adopted from Asharif [1]. Since a real impulse response always starts with zeros which represent the propagation delay between loudspeaker and microphone, we precede the impulse response with 20 zeros.
The artificial impulse response is defined as:

\[
(h_{2048}^{2048})_i = \begin{cases} 
0 & \text{for } 0 \leq i < 20 \\
R_i \cdot \frac{1}{10} \cdot e^{-\frac{(i-1)}{2048}} & \text{for } 20 \leq i < 2048
\end{cases}
\]

where \(R_i\) is a uniformly distributed random number on the range \([-1, 1]\). This results in an impulse response which has a flat frequency distribution. The reason for using this impulse response is that it does not contain a direct field component.\(^1\) Impulse responses with a large direct field component might lead to wrong conclusions as only a few samples of the impulse response contain most of the energy. The first 1024 points of the artificial impulse response are depicted in Figure 1.4.

For some simulations in our report we also use a real impulse response with a relatively high direct field component in the first part of the impulse response. The first 1024 values of this real impulse response are depicted in Figure 1.5.

In all our simulations, the acoustic path is modelled by an impulse response of 2048 points, while the adaptive filter models an impulse response of 1024 taps. As a consequence, the residual signal \(r[k]\) always contains some residual echo.

We use 2 performance criteria for comparing adaptive filters. These are the Normalized Mean Square Error (NMSE) adopted from Soo/Pang [32] and the Convergence Time Constant (CTC) adopted from Moulines [25].

For the Normalized Mean Square Error, we sum \(N\) squared samples in the residual signal and in the signal picked up by the microphone and calculate the NMSE in dBs as follows:

\[
NMSE[k] = 10 \cdot \log_{10} \frac{\sum_{i=0}^{N} (e_i[k] - \hat{e}_i[k])^2}{\sum_{i=0}^{N} (e_i[k])^2},
\]

\(^1\)It is noted that we could also measure an impulse response without a significant direct field component. The choice of using an artificial impulse response is arbitrary.
where \( N \) is the length of the block that contains the residual samples for which we calculate the NMSE[\( k \)]. The steady-state NMSE is the value of the NMSE after convergence of the adaptive filter, and is a measure of the residual echo. The lower the steady-state NMSE, the better the performance of the adaptive filter.

The CTC is defined as the slope of the NMSE curve, during the transient of the algorithm. It assumes that the initial part of the learning curve is linear. The CTC is defined in dB/ms. We note that there are several methods for measuring the rate of convergence.

When generating convergence curves in our simulations, we start with adaptive weights equal to zero. For measuring the steady-state NMSE we simulate until the weights have converged. As the CTC and the NMSE depend on the step-size \( 2\alpha \) of the adaptive filter, we construct graphs in which the NMSE is plotted as a function of the CTC. This results in convex curves [28]. These graphs are more appropriate for comparison between the different adaptive algorithms.

1.5 Outline of the report

This report is organized as follows: In Chapter 2 we present an overview of the adaptive filters in general. We mainly concentrate on the frequency domain adaptive filters. Chapter 3 proceeds with discussing the constraint operator, needed in frequency domain adaptive filters. Calculating this constraint requires a lot of computational power, so we try to understand the meaning of this constraint in order to propose a constraining method with reduced computational complexity while maintaining the performance of the adaptive filter.

The first main topic is concerned with the development of an efficient constraining method in partitioned frequency domain adaptive filters, discussed in Chapter 4. We introduce a constraining mechanism based on an approximation of the constraint operator. The results show that this mechanism has a non-optimal performance, caused by a phenomenon discussed in Chapter 5. Here we present a method for eliminating this phenomenon, lead-
1.5. OUTLINE OF THE REPORT

ing to an adaptive filter with a performance that is almost equal to the performance of the original partitioned frequency domain adaptive filter, but with reduced complexity. A second topic in this report is concerned with the DENS. As some parameters in this device are fixed, the performance of the device is application dependent. Therefore we present a method for adaptively estimating two parameters for this device. These estimated parameters can be extracted from the impulse response modelled of the adaptive filter. Finally, in Chapter 7, we give conclusions concerning both topics.
Chapter 2

Adaptive Filters

In this chapter we give a global overview of different types of adaptive filters that are used in Acoustic Echo Cancellers. Besides the (N)LMS algorithm in time-domain, we mainly discuss the more efficient frequency-domain algorithms FBNLMS, BFDAF and the PBFDAF. We do not intend to give an extensive description of the algorithms, but present a basic overview of how the algorithms are constructed. We furthermore discuss the properties of the algorithms and the fundamental differences. For mathematical descriptions and derivations of the different algorithms, we refer to the Ph.D. theses of Sommen [31] and Egelmeers [10].

2.1 (N)LMS algorithm

The most widely used adaptive filter algorithm is the (Normalized) Least Mean Squares (LMS) algorithm, proposed by Widrow and Hoff [36]. This algorithm performs a stochastic steepest-descent search and converges to the optimal (Wiener) solution by minimizing the mean squared error of the residual signal. This is done by calculating the stochastic gradient $\nabla[k]$ and updating the adaptive weights depending on this gradient. It is derived in [36] that this gradient can be efficiently estimated by the cross correlation between the input signal vector $x[k]$ and the residual signal $r[k]$ as $\hat{\nabla}[k] = -2x[k]r[k]$. This update method is called the LMS update. As the dynamic behaviour of the LMS update depends on the input-signal variance $\mathbb{E}[x^2[k]] = \sigma_x^2[k]$, a normalization is used to eliminate this dependency. This results in the Normalized LMS update equation:

$$w[k+1] = w[k] + \frac{2\alpha}{\sigma_x^2[k]} x[k] \cdot r[k],$$

where $x[k]$ denotes the vector with input-samples and $w[k]$ denotes the vector with adaptive coefficients. The step-size $2\alpha$ controls the rate of convergence of the algorithm. For convergence, this parameter must lie within fixed bounds. The smaller the step-size parameter, the slower the convergence behaviour of the algorithm.

Reasons for using this type of adaptive algorithm are twofold. First the implementation of
the (N)LMS algorithm is very straightforward and significant research has been done on the algorithm. Secondly, this algorithm has a short algorithmic delay of only one sample and has low memory requirements. However a disadvantage of the (N)LMS algorithm is the slow convergence rate in the presence of a (highly) coloured input signal. The (N)LMS also has a relatively high computational complexity per input-sample, which lies in the order of \( N \) operations, where \( N \) is the number of taps of the adaptive filter. Especially in Acoustic Echo Cancellation, the adaptive filter models a long impulse response. Hence this (N)LMS algorithm is not attractive [10]. A solution for both drawbacks is to use block frequency-domain signal processing.

2.2 Block Frequency Domain algorithms

For efficient implementations of adaptive filters, we can use block signal processing in combination with Fast Fourier Transforms (FFTs) for performing fast convolution (filtering) which permits adaptation of filter parameters in the frequency domain in a computationally efficient manner. To do this, we collect a block of input-samples and perform the adaptive filtering in frequency-domain. In this report the frequency-domain data is calculated from time-domain data by means of Fast Fourier Transforms (FFTs). In literature also other transforms (e.g. (I)DCT transformations) are used. However in this report, we only focus on the IFFT/FFT transformation. Dentino [8] proposed to use the FFT transform in adaptive filters. However, the usual FFT method evaluates circular convolutions, while in adaptive filters a linear convolution is needed. There is however a technique for performing a linear convolution by means of FFT/IFFTs, called the Overlap-Save (OLS) method [3], page 212. Before discussing the block frequency-domain adaptive filter algorithms, we discuss this Overlap-Save method.

![Figure 2.1: Overlap-Save method.](image-url)

We note that the Overlap-Add (OLA) method can also be used for efficiently convolving an infinite length vector with a finite length vector [3], page 217. In the Ph.D. thesis
of Sommen [31] the relation between the OLS and OLA method is described for the Block Frequency Domain Adaptive Filters. Usually adaptive filters using the OLS are less computationally complex compared with adaptive filters that use the OLA method. Therefore all algorithms in this report are based on the OLS method and we do not discuss the Overlap-Add method.

From literature we know that an elementwise multiplication of two frequency-domain vectors yield a convolution in time-domain. If both vectors are transformed with the Fast Fourier Transform this convolution is a so-called circular convolution. This relation is shown in the following theorem:

**Theorem 2.2.1**

If $X^{2N} = F \cdot \tilde{X}^{2N}$ and $G^{2N} = F \cdot \tilde{G}^{2N}$, then:

$$X^{2N} \otimes G^{2N} \iff \tilde{X}^{2N} \odot \tilde{G}^{2N}.$$  \hfill (2.2)

**Proof:** Omitted (see [3], page 112).

Here the $\otimes$ denotes a circular convolution, while the $\odot$ denotes an elementwise multiplication. In adaptive filtering we require linear convolutions/correlations and we need to convolve an infinite length input data stream with a finite length vector. The Overlap-Save (OLS) method is a well known technique that handles with both requirements and uses Fast Fourier Transforms for efficient calculation [3], page 212. The OLS scheme for calculating a convolution is depicted in Figure 2.1. We explicitly note that this OLS scheme uses a Half Overlap (HOL) in the input-blocks. In this report we only use 50% overlap of the input-blocks as the algorithms in this report are implemented most efficient by using Half Overlap.

We see that $N$ samples of the input stream $x[k]$ are collected with the Serial/Parallel converter. Then together with the $N$ samples of the previous collected block these last $N$ samples are combined into a single $2N$ point block. The samples with the lowest time-index (i.e. the oldest samples) are placed in the time-vectors as elements with the smallest index. If we place the samples in a time-mirrored sequence, we talk about reverse-time vectors. The picture below shows three subsequent time-vector input-blocks.

![Figure 2.2: 50% overlap of the input signal blocks.](image-url)
These three subsequent input-blocks contain the samples in the following way:

\[
\tilde{x}^{2N}[kN] = (x[kN - 2N + 1], \ldots, x[kN])^T, \tag{2.3}
\]

\[
\tilde{x}^{2N}[(k - 1)N] = (x[(k - 1)N - 2N + 1], \ldots, x[(k - 1)N])^T, \tag{2.4}
\]

\[
\tilde{x}^{2N}[(k - 2)N] = (x[(k - 2)N - 2N + 1], \ldots, x[(k - 2)N])^T. \tag{2.5}
\]

To perform the linear convolution, the length \(N\) weight vector \(w^N\) (in time-domain) is time-reversed and augmented with \(N\) zeros (see Figure 2.1). After the Fast Fourier Transform of this length \(2N\) vector, the elementwise multiplication with the length \(2N\) input block is performed. The time-domain result vector basically consists of two parts. The left side of this vector contains circular wrap-around artifacts. These artifacts have nothing to do with a linear convolution and need to be disposed of. The right side however contains the linear convolution part and is passed to the output. The output stream is generated by concatenating the subsequent linear results.

### 2.3 FBNLMS algorithm

The Overlap-Save implementation of the Frequency Domain Block Normalized Least Mean Squares (FBNLMS) algorithm [16] is depicted in Figure 2.3. The adaptive filter needs a linear convolution for filtering the input signal with the filter-weights, while a linear correlation calculates the gradient estimation needed for updating these weights. After convergence, the adaptive weights have adjusted to the Wiener solution.

![Figure 2.3: Frequency Domain Block Normalized Least Mean Squares (FBNLMS) algorithm.](image-url)
In Figure 2.3 we can clearly see that the correlation and convolution parts are both implemented by means of the Overlap-Save method. We note that the Overlap-Save used for the correlation part is slightly different compared with Figure 2.1. The reason for this difference is the possibility to create a so-called Gradient Constraint operator, discussed further on in this report. The difference of the correlation-part compared with the convolution-part is that one input block needs to be mirrored in time. In frequency-domain this yields a complex conjugate operator. The adaptive weights are updated in time-domain. The time reverse (mirror) operations before and after the weight update can be combined and left out, but this means that the weights are placed in a time reverse vector.

The NLMS differs from the FBNLMS in that the latter calculates $N$ output values without adaptation of the coefficients. In the FBNLMS, older filter vector values are used to calculate the output values, while for the NLMS the filter vector values are updated each iteration. As a result, a $N$ times smaller maximum value for the adaptation constant $2\alpha$ can be chosen (to ensure stability) compared with the NLMS algorithm of Section 2.1.

With this block implementation of NLMS, the computational complexity per input sample is reduced. Because of the FFTs, we now have a computational complexity in the order of $N \log N$ instead of $N^2$ operations to produce $N$ new output samples. The convergence behaviour however is still worse in cases of highly correlated input signals. In order to improve this convergence behaviour in these situations, we use the BFDAF algorithm.

## 2.4 BFDAF algorithm

The FBNLMS algorithm does not have good convergence behaviour in case of signals which are highly correlated. Therefore one searched for methods for improving the convergence behaviour in this case. Mansour and Gray [23] introduced a so-called power normalization which is also used in the BFDAF algorithm we discuss. The BFDAF algorithm is depicted in Figure 2.4. Much mathematical work has been done in deriving the BFDAF algorithm, but we can also derive the BFDAF algorithm from the FBNLMS in an intuitive manner.

When we would use a highly correlated input signal, for example a speech signal or a music signal, the adaptive weights converge very slowly in case we do not use the power-normalization. When we take a look at the adaptive weights in frequency-domain (by means of a FFT), some frequency bins of the adaptive weights converge more quickly than other frequency components. This is caused due to the fact that some frequency bins contain a higher amount of energy than other frequency bins. When all frequency bins are adapted with the same step-size, the frequency bins with a very low amount of energy are adapted very slowly compared with the high energy frequency bins, leading to a slower overall convergence speed.

A very simple approach to improve the convergence behaviour in this situation is by assuming that the individual frequency components (frequency bins) are independent. Hence, we can perform the power normalization of the residual signal (in frequency-domain) in order to get the same rate of convergence for each individual frequency bin.
We note that this independence-assumption is not quite valid. Especially two near frequency bins have much correlation. In practical situations however, the approach of power normalization works very well.

The individual frequency bins of the residual signal are multiplied by the inverse power of every frequency bin in the input signal. The input signal power vector is defined as:

\[ P_x^{2N}[\kappa N] = \frac{1}{2N} \mathbb{E}\{ |X^{2N}[\kappa N]|^2 \}. \] (2.6)

This input signal power vector can be estimated by means of an exponential averaging filter with a smoothing constant. Stable estimates for stationary input signals are obtained by choosing this smoothing constant close, but not higher than one [28]. We note that normalization of the residual signal by the exponential averaged inverse input-signal power does not lead to the optimal decorrelation. There are adaptive filter algorithms that apply a more exact decorrelation [10].

For frequency bins containing a very low amount of signal energy and a relatively high amount of noise energy, the power-normalization would lead to an adaptation of that frequency bin, mainly driven by the noise. This can be solved by performing the power-normalization only for those frequency bins containing a relatively high amount of signal energy.

Finally we note that we can leave out the two time-reversal operations before and after the adaptive weight update, as was the case with the FBNLMS algorithm.
2.5 Partitioned BFDAF algorithm

2.5.1 Algorithm implementation

A drawback of the (50% overlap) BFDAF algorithm is the delay of $N$ samples due to the block processing. First, a complete block with input samples is collected, before the block can be processed. A solution proposed by Asharif [1] and Sommen [29], is to reduce this block size by partitioning the adaptive filter. The algorithm can be seen as a parallel implementation of BFDAF algorithms which all model a small part of the impulse response. The convolution result is obtained by adding all the smaller convolutions together. In this report, this resulting algorithm is referred as the Partitioned BFDAF (PBFDAF) algorithm, although we are aware of the fact that other abbreviations are used in literature [9]. The implementation of the PBFDAF is shown in Figure 2.5 and 2.6.

---

1. The delay can be reduced by choosing a lower overlap, but this leads to an increase in complexity.
The PBFDAF algorithm can be seen as a parallel implementation of BFDAF algorithms with smaller block-lengths, where the PBFDAF with only one partition is equal to the BFDAF algorithm of Section 2.4.

Moulines [25] provides expressions for the region of convergence and the optimum step-size parameter for an algorithm similar to the Partitioned BFDAF algorithm. Simulations have been done to verify these closed form expressions. Due to the partitioning in PBFDAF algorithm the decorrelation length is decreased compared with the BFDAF algorithm. Research has been done on this topic by several persons [30], [10].

For our research, the computational complexity and memory requirements are the most important properties of the PBFDAF, as we want to reduce both. We analyze the memory occupation, determine the computational complexity and present simulations of the PBFDAF algorithm.

### 2.5.2 Memory requirements

In literature, different numbers for the memory occupation of the PBFDAF algorithm are given [10, 32]. This difference is mainly due to the additional storage for temporary results. We analyzed the minimum number of real-valued memory locations for the PBFDAF. We refer to Appendix A for the quantifications in the different parts of the algorithm. The minimum number of memory locations is expressed as:

$$\Theta_{\text{PBFDAF}} = N K \cdot \frac{4K + 7}{K},$$

(2.7)

where $N$ is the block size and $K$ is the number of partitions. Calculating the memory requirements for different values of $K$ results in the graph depicted in Figure 2.7. This graph is normalized by the filter-length $NK$, to get an independence of the adaptive filter length.

![Figure 2.7: Memory requirements of the PBFDAF algorithm.](image)

From Figure 2.7, we see that for more partitions, we achieve more reduction in memory occupation in the PBFDAF algorithm. The PBFDAF with 4 partitions results in a
2.5. PARTITIONED BFDAF ALGORITHM

reduction of the memory occupation by almost a factor two compared with the BFDAF algorithm \((K = 1)\). This corresponds with the observation of Soo and Pang [32].

2.5.3 Computational complexity

As mentioned in the introduction, we distinguish between different operations in our computational complexity analysis. However, in order to make comparison between different algorithms possible, we have to introduce a single complexity number \(\Psi\). We refer to Section 1.4.1 for measures of the computational complexity. The computational complexity of the half overlap PBFDAF algorithm is given by [10]:

\[
\Psi_{\text{PBFDAF}} = \frac{(3 + 2K) \cdot \Psi_{\text{FFT}}[2N] + 2K \cdot \Psi_{\phi}[2N] + \Psi_{P}[2N]}{N}.
\] (2.8)

We refer to the appendix for the overview of the different operations used in this formula (e.g. FFT). We calculate absolute computational complexity numbers for only one situation. We use a 1024 taps adaptive filter with different partition factors \(K\). Because of the overlap factor of 50%, this yield a value \(N = \frac{1024}{K}\). The table below shows the computational complexity per input sample in this situation:

<table>
<thead>
<tr>
<th>(K)</th>
<th>adds/subs (A)</th>
<th>muls (M)</th>
<th>divs (D)</th>
<th>(\Psi_{\text{PBFDAF}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140.37</td>
<td>62.04</td>
<td>1.00</td>
<td>203.41</td>
</tr>
<tr>
<td>2</td>
<td>180.68</td>
<td>77.53</td>
<td>1.00</td>
<td>259.21</td>
</tr>
<tr>
<td>4</td>
<td>258.59</td>
<td>107.27</td>
<td>1.01</td>
<td>366.86</td>
</tr>
<tr>
<td>8</td>
<td>401.29</td>
<td>160.05</td>
<td>1.02</td>
<td>562.35</td>
</tr>
<tr>
<td>16</td>
<td>651.61</td>
<td>249.44</td>
<td>1.03</td>
<td>902.08</td>
</tr>
</tbody>
</table>

Table 2.1: Computational complexity for the PBFDAF algorithm.

For an more partitions \(K\), the complexity per input sample increases quickly. For \(K = 4\), the computational complexity is almost doubled compared with the BFDAF \((K = 1)\).

2.5.4 Step-size bounds

Since further on in this report we perform simulations for the PBFDAF algorithm, it is useful to know for which values of the step-size \(2\alpha\) the PBFDAF converges. A very extensive analysis on the constrained PBFDAF algorithm is done by Moulines [25], where analytic expressions are derived for the region of convergence and the optimum step-size parameter. We verify these analytical results by means of simulations. As the analytical expressions of Moulines are only derived for the white-noise case, we simulate the PBFDAF algorithm with white Gaussian noise for different step-size parameters and different number of partitions \(K\). The simulation is performed for the impulse response of 2048 points in Figure 1.5; the adaptive filter-length is 1024 taps. Figure 2.8 shows the Convergence Time Constant (CTC) as a function of the step-size parameter \(2\alpha\).
For small values of $2\alpha$, the CTC gets better for increasing step-size. As the steady-state NMSE gets worse for increasing step-size, the rate of convergence can be exchanged with the final misadjustment. For higher values of $2\alpha$, both CTC and NMSE gets worse for increasing step-size. Hence values of $2\alpha$ in this region should not be used.

Moulines [25] gives an expression for the upper-bound of the step-size parameter $2\alpha$, guaranteeing the convergence of the Mean Square Error (MSE):

$$0 < 2\alpha < \frac{4}{1 + K}. \quad (2.9)$$

When we calculate this upper bound for $2\alpha$ for each situation for the number of partitions we used in the measurement, we get for $K = 2, 4, 8$ and 16 respectively the upper bounds: 1.33, 0.8, 0.444 and 0.235. We can see in Figure 2.8, that this upper bound is approximately valid in all cases of $K$. The faster rate of convergence for more partitions $K$ is attributed to the more frequent weight update process.

Also a simple analytical expression is available for the optimum step-size parameter [25]. This optimum step-size $(2\alpha)^*$ is the step-size where the fastest rate of convergence is obtained. For the PBFDAF this optimum step-size is given by:

$$(2\alpha)^* = \frac{2}{1 + K}. \quad (2.10)$$

When we fill in the values of $K = 2, 4, 8$ and 16 for each situation, we get the optimum step-sizes of respectively 0.666, 0.4, 0.222 and 0.117. We can see from the results (Figure 2.8), that these values for the optimum step-sizes are approximately correct.
Chapter 3

Efficient constraining mechanisms

In the previous chapter we have discussed different adaptive filter algorithms that are used in Acoustic Echo Cancellers. The weights of the frequency-domain implementations in the previous chapter are updated in time-domain, but we can also update the weights in frequency-domain. Updating the weights in frequency-domain, leads to so-called constraint operators, which impose restrictions on the adaptive weights. These constraints are computationally expensive operations, but are needed for proper linear convolutions and correlations in the algorithm. In this chapter we give an overview of constraining mechanisms which are discussed in literature. With these constraining mechanisms a more computationally efficient algorithm is obtained, while preserving the convergence behaviour of the algorithm.

3.1 The constraint operator

When we observe the FBNLMS, BFDAF and the PBFDAF algorithm in Figures 2.3, 2.4 and 2.5 respectively, we see that in all cases the adaptive weights are stored in time-domain. In the figure below, this time-domain weight update with the two windows, FFT and IFFT is depicted.

![Figure 3.1: Adaptive weights in time-domain.](image)

We note that we use the index $i$ for indicating the partition number. The FBNLMS and the BFDAF can be seen as having only 1 partition. Furthermore we note that we use a
variable $W_{2N}[\kappa N]$, defined as:

$$W_{2N}[\kappa N] = \mathcal{F} \left( \begin{bmatrix} I^N \\ O_N \end{bmatrix} J^N w_{2N}[\kappa N] \right).$$  \hspace{1cm} (3.1)

After transforming the vector $Z^2_{2N}[\kappa N]$ to time-domain, the first window in this figure discards $N$ points of the correlation result. This is needed to remove the circular wrap-around artifacts from the Overlap-Save method. The remaining $N$ samples are used to update the adaptive weights. After this update $N$ zeros are augmented. This is needed for the efficient convolution of the adaptive weights with the input signal, by means of the Overlap-Save method. Finally, the result is transformed back to frequency-domain.

We can clearly see that the two mirroring operators $J^N$ can be eliminated by storing the vector $w_{2N}[\kappa N]$ in time-reverse order. Furthermore, we can combine the two time-domain windows, the FFT and the IFFT in a time-domain gradient constraint operator [21]. From now on, we will talk about constraints instead of gradient constraints. As a consequence of the constraint operator, the adaptive coefficients need to be stored and updated in frequency domain instead of time-domain. These frequency-domain coefficients will be denoted by $W_{2N}$. We can illustrate the separation of the constraint operator and the adaptive coefficients by the following deduction:

$$W_{2N}[(\kappa + 1)N] = \mathcal{F} \left( \begin{bmatrix} I^N \\ O_N \end{bmatrix} J^N \left( w_{2N}[\kappa N] \right) + J^N \left( \begin{bmatrix} I^N \\ O_N \end{bmatrix} \mathcal{F}^{-1}Z^2_{2N}[\kappa N] \right) \right)$$

$$= W_{2N}[\kappa N] + \mathcal{F} \left( \begin{bmatrix} I^N \\ O_N \end{bmatrix} \begin{bmatrix} I^N \\ O_N \end{bmatrix} \mathcal{F}^{-1}Z^2_{2N}[\kappa N] \right)$$

$$= W_{2N}[\kappa N] + \mathcal{F} G^{2N \times 2N} \mathcal{F}^{-1}Z^2_{2N}[\kappa N],$$

where $G^{2N \times 2N}$ is the constraint window matrix which has a diagonal form. The vector on this diagonal is called the constraint window vector $g^{2N}$. In this report, the constraint window is also referred to as the rectangular constraint window. Figure 3.2 shows the frequency-domain coefficient update with the rectangular constraint window in block representation.

![Figure 3.2: Adaptive weights in frequency-domain.](image-url)

We note that there are reasons for updating and storing the adaptive weights in frequency-domain, despite the fact that frequency-domain weights need more storage elements compared with the time-domain weights. The most important reason is the possibility to omit the gradient constraint, in order to reduce the computational complexity.
In most articles, the rectangular constraint is placed before the coefficient update as formulated in Equation 3.2, where the gradient constraint is calculated before the coefficient update takes place. However, it is advisable to apply the gradient constraint after the coefficients are updated [11]. Although it is quite easy to see that this is allowed, we can show this mathematically:

\[ \mathcal{F}G\mathcal{F}^{-1}W_{i}^{2N}[(\kappa + 1)N] = \mathcal{F}G\mathcal{F}^{-1}W_{i}^{2N}[\kappa N] + \mathcal{F}G\mathcal{F}^{-1}\mathcal{F}G\mathcal{F}^{-1}Z_{i}^{2N}[\kappa N]. \]  

(3.3)

The relation \( \mathcal{F}G\mathcal{F}^{-1}\mathcal{F}G\mathcal{F}^{-1} = \mathcal{F}G\mathcal{F}^{-1} \) is obvious, because \( \mathcal{F}^{-1}\mathcal{F} \) equals the unity matrix containing all ones and \( GG = G \) is valid. Therefore equation 3.3 can be rewritten as:

\[ W_{i}^{2N}[(\kappa + 1)N] = \mathcal{F}G\mathcal{F}^{-1}\left(W_{i}^{2N}[\kappa N] + Z_{i}^{2N}[\kappa N]\right). \]  

(3.4)

Again we note that \( G^{2N \times 2N} \) is the diagonal constraint window matrix which contains the constraint window vector \( g^{2N} \) on its diagonal. In block representation the weight update with the constraint after the update can be depicted as follows:

![Figure 3.3: Constraining the coefficients after the weight update.](image)

Calculating the rectangular constraint with infinite precision gives results which are the same for both situations (Figure 3.2 and 3.3). However when calculating the weight update in Figure 3.2 with finite precision (e.g. fixed-point arithmetic), numerical round-off errors are accumulated in the weight update. Eventually, after many iterations of the weight update, this leads to large errors in the last \( N \) points of the coefficients. Correction of these accumulated errors is prevented by the right half part of the constraint. When placing the constraint after the weight update as depicted in Figure 3.3, the result after calculating the constraint also contains numerical round-off errors in cases of finite precision. However, as the constraint operation is performed on the actual coefficients, there is no accumulation of these errors which could lead to long-term problems. Hence, rectangular constraint operations need to be placed after the weight update.

### 3.2 Constraints in the BFDAF algorithm

#### 3.2.1 Unconstrained BFDAF

The BFDAF algorithm as shown in Figure 2.4 needs 5 FFT/IFFT operations per block iteration, which is in fact most of the total computational complexity of the algorithm. It
is proposed to reduce the computational complexity by skipping the constraint. Therefore we use the constraint window defined as: $g_{2N}^i = 1$, for $0 \leq i < 2N$. Hence only 3 FFT/IFFT operations per block iteration remain. Dentino was the first who described this unconstrained structure, but used an overlap of 0% instead of the half overlap of the input-data blocks. Mansour and Gray [23] showed by means of extensive mathematical analysis, that for the 50% overlap case, the removal of the constraint in the adaptive filter does lead to convergence to the Wiener solution under the condition that in the case of $2N$ point FFTs, the length of the identified impulse response needs to be less or equal to $N$ points. The only effect is that $N$ weights are fluctuating around zero, which increases the final misadjustment by a factor 2 (3 dB) without any effect on the rate of convergence [28]. When the unknown system impulse response is longer than $N$, the unconstrained algorithm no longer converges to the Wiener solution [23].

If a $2N$ point unconstrained adaptive filter is used, the question arises, why this adaptive filter is not capable of effectively modelling a higher ($> N$) order impulse response. The reason lies in the fact that the last $N$ coefficients are in fact circular wrap-around artifacts which are the result of a circular convolution. Li and Jenkins [21] showed by means of simulation that after convergence, this $2N$ point unconstrained adaptive filter frequently obtains a lower MSE compared with the $N$ point constrained adaptive filter. However the variance of the MSE after convergence is much higher than the $N$ point constrained adaptive filter. Also the convergence speed and the step-size upper bound is lower in the case of the unconstrained BFDAF. The contribution of Petraglia [26] also shows similar results. The experimental results of Li and Jenkins [21] and Petraglia [26] on the unconstrained BFDAF are very hard to prove by means of mathematical analysis. The reason for this difficulty is due to the presence of the circular wrap-around artifacts.

### 3.2.2 BFDAF constraint approximation

Some work has been done in eliminating the constraint in the BFDAF algorithm, while preserving the convergence speed and the step-size upper bound of the constrained algorithm.

Sommen [28] proposed an efficient window function for the BFDAF algorithm. Here the rectangular constraint operation in time-domain is replaced by a raised cosine window denoted by:

$$ (g_{2N})_i = \frac{1}{2} \left(1 + \cos \left(\frac{\pi \cdot i}{N}\right)\right). $$

In frequency-domain this window results in a circular convolution with only one real and two mutually conjugate imaginary coefficients. Hence, this circular convolution can be performed with very low computational complexity. Using this constraint approximation in the BFDAF algorithm leads to a near optimum convergence behaviour compared with the constrained BFDAF in cases where the system to identify is shorter than $N$. 
3.3 Constraints in the PBFDAF algorithm

3.3.1 Unconstrained PBFDAF

In the Partitioned BFDAF algorithm, $3 + 2K$ FFTs need to be computed, which is the largest part of the total computational complexity (formula 2.8). The $2K$ FFTs are the FFTs/IFFTs needed for the computation of the constraint operations. We could simply leave out the constraints as was also proposed for the BFDAF algorithm in the previous section, leading to the Unconstrained PBFDAF. This leads to very slow convergence or even divergence of the adaptive weights [10], but reduces the computational complexity as indicated by formula 3.6.

$$\Psi_{PBFDAF} = \frac{3 \cdot \Psi_{FFT}[2N] + 2K \cdot \Psi_\otimes[2N] + \Psi_P[2N]}{N}. \quad (3.6)$$

When we calculate the computational complexity for the situation where we use an 1024 taps adaptive filter with different partition factors $K$, the results are as follows:

<table>
<thead>
<tr>
<th>$K$</th>
<th>adds/subs $A$</th>
<th>muls $M$</th>
<th>divs $D$</th>
<th>$\Psi_{PBFDAF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89.42</td>
<td>41.22</td>
<td>1.00</td>
<td>131.65</td>
</tr>
<tr>
<td>2</td>
<td>89.43</td>
<td>41.23</td>
<td>1.00</td>
<td>131.67</td>
</tr>
<tr>
<td>4</td>
<td>97.40</td>
<td>45.27</td>
<td>1.01</td>
<td>143.68</td>
</tr>
<tr>
<td>8</td>
<td>121.29</td>
<td>57.30</td>
<td>1.02</td>
<td>179.60</td>
</tr>
<tr>
<td>16</td>
<td>176.61</td>
<td>85.44</td>
<td>1.03</td>
<td>263.08</td>
</tr>
</tbody>
</table>

Table 3.1: Computational complexity for the Unconstrained PBFDAF algorithm.

We clearly see that compared with the constrained PBFDAF algorithm (see Table 2.1), the reduction in complexity is substantially large, as $2K$ FFTs are eliminated.

In the previous section we mentioned that for the unconstrained BFDAF a mathematical analysis was available. For unconstrained PBFDAF however (in the case of multiple partitions), we could not find any extensive mathematical analysis in literature.

Despite the appealing computational complexity of the unconstrained PBFDAF algorithm, it is not advisable to leave out the constraints in the PBFDAF algorithm, as the performance of the unconstrained PBFDAF is poor.

3.3.2 Alternative PBFDAF

As the unconstrained PBFDAF is found to be substantially slower and to have a larger misadjustment, some effort in literature is done in finding a constraining mechanism with low computational complexity that maintains the convergence behaviour of the fully constrained PBFDAF. The Alternative constraining mechanism, proposed by Soo and Pang [32], tries to achieve these objectives. They also found this constraining mechanism to
behave poorly in cases where the adaptation constant is large or when the alternative constraining is not done quickly enough. Some more articles are concerned with the Alternative constraining mechanism [6, 19, 37]. In this report, we use the abbreviation 'Alternative PBFDAF' to indicate that this method of constraining is used. The global algorithm of the Alternative PBFDAF differs slightly from the PBFDAF algorithm (Figure 2.5). This difference is within the update part, which is depicted below:

\[
\begin{align*}
&X_{t}^{2N}[\kappa N] \\
&\times \quad + \quad 1 \\
&(X_t^{2N})^*[\kappa N] \\
&\quad \quad 0 \\
&\quad \quad S_i \\
&\quad \quad 1 \\
&\quad \quad 0 \\
&\quad \quad S_i \\
&\quad \quad T_N \\
&\quad \quad W_t^{2N}[\kappa N]
\end{align*}
\]

**Figure 3.4:** Update block $A_i$ of the Alternative PBFDAF algorithm.

The idea of the alternative constraining method is to constrain the coefficient vector of a particular partition only once in a while. Usually one should use a linear way of alternative constraining. By performing a linear alternative constraining, the two switches of Figure 3.4 are described by the following formula:

\[
S_i = \begin{cases} 
1 & \text{if } \kappa \mod KP = i \\
0 & \text{otherwise}.
\end{cases}
\]  

(3.7)

In this formula the parameter $K$ indicates the number of partitions in the Alternative PBFDAF algorithm. The parameter $P$ is the alternative constraining period in number of block iterations of the algorithm.

This linear alternative constraining means that the frequency of which a particular partition is constrained is equal for all partitions. The article of Joho [19] and Estermann [14] use such a linear way of constraining. The computational complexity of the Alternative PBFDAF approaches that of the Unconstrained PBFDAF for high values of $K$ and $P$. However, for higher values of $K$ and $P$ there is a degradation in performance of the Alternative PBFDAF algorithm [32]. Further-on in this report we present a method where this degradation is much less for increasing values of $K$ and $P$.

### 3.3.3 Scheduled constrained PBFDAF

The Scheduled constrained PBFDAF [24] [5] is proposed by McLaughlin. Basically, the Scheduled PBFDAF is equal to the Alternative PBFDAF, discussed in the previous section. The only difference is that the frequency at which every partition is constrained is related to the power in the impulse response. Hence the rate of convergence is faster than in the Alternative PBFDAF. The method has a disadvantage over the Alternative PBFDAF algorithm. When constructing the schedule in which some partitions are more frequently constrained than others, the power in each partition needs to be calculated. As the power in the partitions could fluctuate in time, the schedule might need to be updated during the adaptation process, resulting in additional computational complexity.
3.3.4 PBFDAF constraint approximations

In section 3.1 we saw that an IFFT and FFT operation and an elementwise multiplication are needed to calculate the constraint in order to null half of the coefficients in time-domain. This requires $O(N \log N)$ operations, where $2N$ is the length of the FFT/IFFT. We can also perform this constraint by calculating a circular convolution directly in frequency-domain. This is indicated by the following theorem:

Theorem 3.3.1

If $X^{2N} = \mathcal{F}_T x^{2N}$ and $G^{2N} = \mathcal{F}_T g^{2N}$, then:

$$x^{2N} \otimes g^{2N} \leftrightarrow \frac{1}{2N} \left( X^{2N} \otimes G^{2N} \right).$$

(3.8)

Proof: Omitted (see [3], page 113).

The straightforward calculation of the circular convolution requires $O(N^2)$ operations. In the report of Egelmeers [11] the frequency-domain coefficients $(G^{2N})_i$ for the circular convolution with the rectangular constraint are deduced:

$$(G^{2N})_i = \begin{cases} 2N \cdot \frac{1}{2} & \text{for } i = 0 \\ 0 & \text{for } 0 < i < 2N, \ i \ \text{even} \\ \frac{-2N \cdot i}{2N \cdot \tan \left( \frac{\pi}{2N} \right)} & \text{for } 0 < i < 2N, \ i \ \text{odd.} \end{cases} \quad (3.9)$$

The factor $2N$ in the coefficients of $G^{2N}$ are compensated by factor $\frac{1}{2N}$ of theorem 3.3.1. However for reasons of consistency this factor is not omitted in formulas in this report.

The first three orders of the constraint window $g^{2N}$ are depicted in Figure 3.5.

![Figure 3.5: Different constraint approximation windows.](image)

When using a first order approximation, we only use the real coefficient and the first two mutually conjugate imaginary coefficients. This results in a sinusoid constraint window that can be computed in frequency-domain with much less computational complexity.
than by means of an FFT/IFFT. We note that this window is different from the window as formulated in Section 3.2.2.

Constraint windows with other than 50% overlap (half overlap) contain frequency components that have both real and complex values. Hence such a calculation needs additional computational complexity compared with the 50% overlap windows. As mentioned before, in most practical cases half overlap constraint windows are used. Hence, we only focus on these cases.

It is also possible to combine the constraint approximations with the Alternative PBFDAF or Scheduled PBFDAF [11]. The rectangular constraints need to be placed after the update as proposed in section 3.1. This is however different for the constraint approximations. When we place the approximations after the update, the time-domain adaptive weights are constantly multiplied by the weights of the approximation window. As most of the weights in the approximation window are generally not equal to one, this would lead to adjustments of the weights in every block-iteration and the adaptive filter would not converge [11]. Hence we apply all constraint approximations before the update.

### 3.3.5 Hybrid constrained PBFDAF

In the Hybrid structure proposed by Asharif [1], only the first $K_t$ partitions are constrained, while the other partitions remain unconstrained. As the first $K_t$ partitions contain more power than the last partitions, better convergence is achieved. However due to the last unconstrained partitions, there is a presence of circular wrap-around artifacts. Hence, a near optimal performance with respect to the fully constrained method cannot be achieved.

### 3.3.6 Reduced overlap PBFDAF

In the paper of Farhang-Boroujeny [15] it is stated that the performance of the Unconstrained PBFDAF is degraded by increasing the overlap of the partitions, while for the constrained PBFDAF this overlap factor does not have serious effects on the convergence behaviour. As a result, the paper of Farhang-Boroujeny [15] describes a reduced overlap between the partitions in Unconstrained PBFDAF. This method is questionable while in this method circular wrap-around effects are not eliminated at all. Hence, the performance of the fully constrained method cannot be achieved.

### 3.3.7 Unconstrained PBFDAF with Lattice

Chan [4] proposed the Half Overlap (HOL) of the partitions in the Unconstrained PBFDAF and introduced a lattice structure in order to remove the correlation between successive partitions. The lattice structure is computationally complex and the results presented in the paper concerning the new lattice method indicate that the convergence is far from optimal with respect to the fully constrained method.
Chapter 4

Constraint approximations

As a starting point in developing a constraining method with a low computational complexity and a good performance, we investigate the behaviour of the adaptive filter when approximating the rectangular constraint. By means of simulations, we show that certain properties of these constraint approximations are required and others are highly recommended. As a result we propose a so-called high-slope constraint window and a constraining method based upon this high-slope constraint window.

4.1 Properties of constraint approximations

In the report of Egelmeers [11] it is stated that negative values in the time-domain window might need to be avoided to avoid divergence of the adaptive weights. While a mathematical analysis to prove this for the PBFDAF algorithm will not be easy, we show this by means of an experiment, where we use the following constraint window:

\[
(g^{2N})_i = \begin{cases} 
1 & \text{for } 0 \leq i < N \\
\epsilon & \text{for } N \leq i < 2N.
\end{cases}
\] (4.1)

We perform the experiment with the PBFDAF algorithm, where the filter-length equals 1024 taps partitioned into 8 partitions \((K = 8)\). We use the real impulse response of 2048 points (see Figure 1.5), which is convolved with a zero-mean unit-variance white Gaussian noise input-signal. We perform the experiment for the values \(\epsilon = 0, -0.05, -0.1\) and use a step-size \(2\alpha = \frac{1.0}{K}\), which lies within the region of convergence.

We see in Figure 4.1 that for negative values of \(\epsilon\), the coefficients of the adaptive filter eventually start to diverge. The more negative the value of \(\epsilon\), the faster the divergence of the coefficients. Hence we can conclude that negative values in the constraint window should be avoided. The reason for the divergence in the case of negative values in the constraint window can be explained in an intuitive way. Because of the negative values, the correlation part produces results that are negatively correlated with the residual signal. Hence the adaptive weights are updated in the wrong direction. As the adaptive weights
become less accurate, there is more negative correlation and worsening of the weights in every next block-iteration.

![Graph showing influence of negative-valued constraint windows.](image)

Figure 4.1: Influence of negative-valued constraint windows.

As the frequency-domain coefficients of Equation 3.9 result in time-domain coefficients that can be negative, we need to scale the frequency coefficients. Using more frequency components in the constraint approximation, results in better performance, but enlarges the computational complexity. To achieve an optimal performance with low computational complexity, we investigate which properties of the constraint are important. Therefore we perform an experiment with constraint windows as depicted in Figure 4.2.

![Graph showing different types of constraint windows.](image)

Figure 4.2: Different types of constraint windows.

Constraint window 1 includes the real and first 3 mutually complex conjugate components according to formula 3.9. We apply a scaling by a factor $\frac{5}{6}$ in order to avoid negative val-
ues in the window. The second window contains the same gradient in the slope compared with the first window type, but is more accurate (with respect to the rectangular constraint window) in between the slopes. The step-size parameter $2\alpha$ is set to the value of $0.5$. Furthermore in this experiment we use the same input-signal and the same impulse response as in the previous experiment in this section. The results are depicted in Figure 4.3.

As we can see, both constraint approximation windows lead to a less accurate steady-state NMSE compared to the fully constrained case. The two approximation windows however do not differ very much. Apparently, the values in between the slopes are of little importance for the convergence behaviour. This can be explained due to the fact that the time-domain adaptive weights in between the slopes are updated with random (noisy) values. Hence the weights do not converge and are zero on average. This is not true for the circular artifacts closer to the slope. The adaptive weights close to these slopes converge to a non-zero value and introduce errors in the convolution part of the PBFDAR algorithm, as the neighbour partitions are coupled [10]. To eliminate as much as possible of the circular artifacts that converge to a non-zero value, we concentrate on creating high slopes on the two places where there are transitions in the rectangular constraint window. This is the topic of the next section.

### 4.2 Constraint window with a high slope

This section describes an efficient algorithm to convolve a complex-valued Hermitian signal with a high-slope constraint window. This constraint approximation window will be
used in the PBFDAF algorithm instead of the rectangular constraint, in order to reduce the computational complexity of the algorithm. Firstly, the algorithm for this efficient convolution is derived formally. Secondly, we introduce some simplifications for achieving an even lower computational complexity.

### 4.2.1 Derivation of the algorithm

As we showed that high slopes are important for obtaining a good convergence behaviour, we should use as many frequency components as possible. When using a regular circular convolution algorithm, incorporating more frequency components leads to a linear increase in computational complexity. Therefore we should not use a regular circular convolution algorithm.

An efficient method for convolving a signal with an approximation of the rectangular constraint is obtained by assuming that the odd frequency components in Equation 3.9 are related with the neighbouring odd components by a multiplicative factor. As this window also contains higher-order frequency components, the slope of this constraint approximation is very steep. Hence this window is called the high-slope constraint window.

The high-slope constraint window \( g_{2N} \) in frequency domain for 50% overlap partitions is defined as:

\[
(g_{2N})_i = \begin{cases} 
2N \cdot aj & \text{for } i = 0 \\
0 & \text{for } 0 < i < 2N, i \text{ even} \\
-\frac{2N \cdot aj}{2} \left( m^{-\left(\frac{i}{2}\right)} - m^{-\left(\frac{2N-1-i}{2}\right)} \right) & \text{for } 0 \leq i < 2N, i \text{ odd.}
\end{cases}
\]  

(4.2)

Because of the relation between two subsequent frequency components of this above defined constraint-window, the frequency domain circular convolution of this window with a complex valued Hermitian signal can be calculated in an efficient way.

The experiments in the previous section showed that good results can be obtained when the constraint-window \( g_{2N} \) in time domain qualifies the following conditions:

1. \((g_{2N})_i \geq 0\), \(\forall i\),
2. High-slope on \(i = 0\) and \(i = N\).

From the experiments in Section 4.1 it follows that a steep transition on \(i = 0\) and \(i = N\) (i.e. high-slope) is of more importance than the nulling in the range \(N \leq i < 2N\). By choosing \(m = 2.166\) and \(a = 0.57\), the high-slope constraint window \(g_{2N} = \mathcal{F}^{-1} \cdot G_{2N}\) just qualifies the first condition above, while maximizing the slope on the positions \(i = 0\) and \(i = N\), as proposed in condition 2. This optimal value of \(m\) is found experimentally. With the value \(a\), the area between the zero level and the high-slope window can be adjusted. For \(a = 0.57\), the mean amplitude in the range \(0 \leq i < N\) is normalized to 1. The constraint-window can be calculated for \(2N = 256\) as is depicted in Figure 4.4.

To derive an efficient algorithm for convolving the high-slope window with the complex-valued Hermitian input-signal, we start with the definition of the circular convolution:
4.2. CONSTRAINT WINDOW WITH A HIGH SLOPE

Figure 4.4: High-slope constraint window.

Definition 4.2.1

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G^{2N})_k = \frac{1}{2N} \sum_{i=0}^{2N-1} (X^{2N})_{(k-i) \mod 2N} \cdot (G^{2N})_i.
\] (4.3)

As the circular convolution is a linear operation, we can apply the superposition theorem and split up the circular convolution with \(G^{2N}\) into three parts:

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G^{2N})_k = \frac{1}{2N} \cdot \left( (X^{2N} \otimes G^{2N})_k + (X^{2N} \otimes G^{2N}_2)_k + (X^{2N} \otimes G^{2N}_3)_k \right),
\] (4.4)

where:

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G^{2N}_1)_k = \sum_{i=0, i \text{ odd}}^{2N-1} (X^{2N})_{(k-i) \mod 2N} \cdot \frac{-aj}{2} \cdot m^{-\left(\frac{i-1}{2}\right)},
\] (4.5)

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G^{2N}_2)_k = \sum_{i=0, i \text{ odd}}^{2N-1} (X^{2N})_{(k-i) \mod 2N} \cdot \frac{aj}{2} \cdot m^{-\left(\frac{2N-1-i}{2}\right)},
\] (4.6)

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G^{2N}_3)_k = (X^{2N})_k \cdot a.
\] (4.7)

As there is a direct relation between formula 4.5 and formula 4.6, we only need to calculate one of these circular convolutions in the algorithm. This relation between the two formulas is expressed in the following lemma:

Lemma 4.2.1 If \((G^{2N}_2)_k = (G^{2N}_1)^*_{2N-k}\) and \((X^{2N})_k = (X^{2N})^*_{2N-k}\)

(i.e. \(X^{2N}\) Hermitian), then:

\[
(X^{2N} \otimes G^{2N}_2)_k = (X^{2N} \otimes G^{2N}_1)^*_{2N-k}.
\] (4.8)
Proof:

\[
(X^{2N} \otimes G_2^{2N})_k = \sum_{i=0, i \text{ odd}}^{2N-1} ((X^{2N})_{(k-1) \mod 2N} \cdot (G_2^{2N})_i)
\]

\[
= \sum_{i=0, i \text{ odd}}^{2N-1} ((X^{2N})_{(k-1) \mod 2N} \cdot (G_1^{2N})_{2N-i}^*)
\]

\[
= \sum_{i=0, i \text{ odd}}^{2N-1} ((X^{2N})_{(2N-(k-1)) \mod 2N} \cdot (G_1^{2N})_{2N-i}^*)
\]

\[
= \sum_{i=0, i \text{ odd}}^{2N-1} ((X^{2N})_{(2N-(k-1)) \mod 2N} \cdot (G_1^{2N})_{i}^*)
\]

\[
= (X^{2N} \otimes G_1^{2N})_{2N-k}.
\]

Hence, in the next derivation of the algorithm, we only consider the circular convolution of formula 4.5.

Because each next convolution-element \((G_1^{2N})_{k+2}\) can be expressed as a function of the actual convolution-element \((G_1^{2N})_k\), an efficient convolution algorithm can be derived. The iterative expression is formulated in the following lemma:

**Lemma 4.2.2** If \(G_1^{2N} = \begin{cases} 0 & \text{for } 0 \leq i < 2N, i \text{ even} \\ -\frac{2N-aj}{2} \cdot m^{-\frac{(i-1)}{2}} & \text{for } 0 \leq i < 2N, i \text{ odd} \end{cases}\)

and \((X^{2N})_k = (X^{2N})_{2N-k}^*\) (i.e. \(X^{2N}\) Hermitian), then:

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G_1^{2N})_{k+2} = m^{-1} \left( \frac{1}{2N} \cdot (X^{2N} \otimes G_1^{2N})_k - (X^{2N})_{k+1} \cdot \frac{aj}{2} \cdot (m - m^{-(N-1)}) \right). \quad (4.9)
\]

Proof:

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G_1^{2N})_k = \sum_{i=0, i \text{ odd}}^{2N-1} ((X^{2N})_{(k-1) \mod 2N} \cdot -\frac{aj}{2} \cdot m^{-\frac{(i-1)}{2}})
\]

\[
\frac{1}{2N} \cdot (X^{2N} \otimes G_1^{2N})_{k+2} =
\]
4.2. CONSTRAINT WINDOW WITH A HIGH SLOPE

\[
\sum_{i=0, \text{i odd}}^{2N-1} \left( (X^{2N})_{(k+2-i) \mod 2N} \cdot \frac{-aj}{2} \cdot m^{-\left(\frac{i+1}{2}\right)} \right)
\]

\[
= \sum_{i=-2, \text{i odd}}^{2N-3} \left( (X^{2N})_{(k-i) \mod 2N} \cdot \frac{-aj}{2} \cdot m^{-\left(\frac{i+1}{2}\right)} \right)
\]

\[
= \sum_{i=-2, \text{i odd}}^{2N-1} \left( (X^{2N})_{(k-i) \mod 2N} \cdot \frac{-aj}{2} \cdot m^{-\left(\frac{i+1}{2}\right)} \right) + \frac{aj}{2} \cdot (X^{2N})_{(k-2N+1) \mod 2N} \cdot m^{-N}
\]

\[
= m^{-1} \sum_{i=0, \text{i odd}}^{2N-1} \left( (X^{2N})_{(k-i) \mod 2N} \cdot \frac{-aj}{2} \cdot m^{-\left(\frac{i+1}{2}\right)} \right) - \frac{aj}{2} \cdot (X^{2N})_{k+1} \cdot (m^0 - m^{-N})
\]

\[
= m^{-1} \left( \frac{1}{2N} \cdot (X^{2N} \oplus G^{2N}^2)_k - (X^{2N})_{k+1} \cdot \frac{aj}{2} \cdot \left( m - m^{-(N-1)} \right) \right).
\]

In a similar way as in the above derivation, we can derive an expression for calculating a convolution element \((G^{2N})_k\) as a function of the convolution element \((G^{2N})_k\). Without proof, this results in the following expression:

\[
\frac{1}{2N} \cdot (X^{2N} \oplus G^{2N}^2)_{k-2} = m^{-1} \left( \frac{1}{2N} \cdot (X^{2N} \oplus G^{2N}^2)_k + (X^{2N})_{k-1} \cdot \frac{aj}{2} \cdot \left( m - m^{-(N-1)} \right) \right).
\]

(4.10)

Note that the above two expressions are numerically stable iterations, because of the multiplication with the factor \(m^{-1}\). One can also derive an expression where in each iteration a multiplication with factor \(m\) is done; however, such an expression leads to less accurate results.

In order to calculate a convolution with the high-slope constraint in frequency-domain, we basically need to do the following sub-calculations explained in the next sections:

1. Calculation of \(\frac{1}{2N} \cdot (X^{2N} \oplus G^{2N}^2)\) (A scaling operation),
2. Initial calculation of \(\frac{1}{2N} \cdot (X^{2N} \oplus G^{2N}^2)_0\) and \(\frac{1}{2N} \cdot (X^{2N} \oplus G^{2N}^2)_1\),
3. Iterative calculation of the convolution by means of lemma 4.2.2,
4. Calculation of \(\frac{1}{2N} \cdot (X^{2N} \oplus G^{2N})_k\) out-of the calculations above.

4.2.2 Convolution of the mean value

The convolution with the vector with \(G^{2N}_3\) containing the mean value \(a\), can be calculated by scaling the input-signal \(X^{2N}\) with the factor \(a\). We note that when performing more than one circular convolution with the high-slope constraint window (as is the case in the PBFDAF algorithm), this convolution with the mean value \((2N)^{-1} \cdot (G^{2N})_0\) can be performed at the cost of only 1 multiplication (see Section 4.3).
4.2.3 Initial calculations

The calculation of $\frac{1}{2N} \cdot (X^{2N} \circledast G_1^{2N})_0$ and $\frac{1}{2N} \cdot (X^{2N} \circledast G_1^{2N})_1$, is needed for the first iteration of the iterative calculation of the circular convolution, one for the odd points and one for the even points. As the sum of the first few mutually complex conjugate frequency components of Equation 3.9 is almost equivalent to the total sum of all mutually complex conjugate frequency components, there is no need to calculate $\frac{1}{2N} \cdot (X^{2N} \circledast G_1^{2N})_0$ and $\frac{1}{2N} \cdot (X^{2N} \circledast G_1^{2N})_1$ based on all the coefficients of $G_1^{2N}$. Therefore we perform the initial calculation based upon the following approximation:

$$\frac{1}{2N} \cdot (X^{2N} \circledast G_1^{2N})_k \approx \frac{1}{2N} \cdot \sum_{i=0}^{Q} (X^{2N})_{(k-i) \mod 2N} (G_1^{2N})_i.$$  \hfill (4.11)

In the calculation of $\frac{1}{2N} \cdot (X^{2N} \circledast G_1^{2N})_0$ ($k = 0$) and $\frac{1}{2N} \cdot (X^{2N} \circledast G_1^{2N})_1$ ($k = 1$), only 1 or 2 coefficients of $G_1^{2N}$ need to be considered (i.e. $Q = 1, 3$) in order to get accurate results.

4.2.4 Iterative calculations

The factor $\frac{1}{2N} \cdot (m - m^{-(N-1)})$ in lemma 4.2.2 can be pre-calculated beforehand by the input-data $X^{2N}$. As the multiplication with $a$ can be performed elsewhere, as described in 4.2.2, the pre-calculation factor is $\frac{1}{2} \cdot (m - m^{-(N-1)})$. For large $N$, this factor is approximately $\frac{1}{2} \cdot m$. In Section 4.2 we have chosen $m = 2.166$, resulting in a factor of 1.083. We can approximate this value by 1.0, resulting in a reduced complexity of $2N$ multiplications, without violating the condition $(\bar{g}_i^{2N})_i \geq 0, \forall i$.

![Figure 4.5: Influence of the pre-multiply approximation.](image)

The complexity of the iteration-steps can be reduced even further, as a multiplication by two can be performed by a shift operation as explained in Section 1.4.1. When using a factor of 1.0 and $m = 2$, the constraint-window slightly drops below zero, thus violating
the condition \((g^{2N}_i)_j \geq 0\), for \(0 \leq i < 2N\). As we explained in Section 4.1 that negative values in the constraint should be avoided, it is not advisable to use \(m = 2\).

When we perform the iterative calculations with the pre-multiplication factor of 1.0, a single iteration in the convolution algorithm is done at the cost of two additions and one multiplication.

### 4.2.5 Calculation of the convolution-result

Because of the relation between \(G_1^{2N}\) and \(G_2^{2N}\) according to lemma 4.2.1, only the convolution with \(G_1\) needs to be calculated. Also half of the output-points of \(\frac{1}{2N} (X^{2N} \circ G^{2N})\) needs to be calculated, as the frequency-domain vectors are Hermitian.

### 4.3 Proposed constraining mechanism

With the high-slope constraint window as a basis, we developed a constraining mechanism for the PBFDAF algorithm. The objective is to achieve a lower computational complexity, but a near optimal convergence behaviour, compared with the fully constrained PBFDAF algorithm.

When using constraint approximations alone, not all the circular wrap-around artifacts are removed. This leads to degradation in the convergence behaviour, as the different partitions in the PBFDAF are coupled. In order to obtain better convergence behaviour, we use a combination of constraint approximations with the Alternative PBFDAF.

The global structure of the PBFDAF algorithm is basically the same as the full-constrained PBFDAF algorithm in Figure 2.5. The only difference is that for an efficient implementation, the elementwise multiplication by \(a\) is performed in the step-size parameter \(2\alpha\). This is done for reducing the computational complexity in calculating the convolutions of the mean values of approximation constraints in the different partitions (see Section 4.2.2). By doing this pre-scaling, only 1 instead of \(2NK\) real multiplications are needed when all the partitions are constrained with the approximation constraint. This single multiplication is eliminated in the complexity analysis of Section 4.4. In cases where we apply a rectangular constraint after the coefficient update, the effect of the multiplication with \(a\) must be eliminated by an elementwise multiplication by \(a^{-1}\). This is done at a cost of \(2N\) real multiplications, as can be seen in Figure 4.7, where the update block \(A_i\) is depicted.

We apply a linear way of rectangular constraining, where the state of the switches are described by the following formula:

\[
S_i = \begin{cases} 
1 & \text{if } \kappa \mod (KP) = i \\
0 & \text{otherwise}
\end{cases}
\]  

(4.12)

We note that we can also use the Scheduled PBFDAF instead of the Alternative PBFDAF. As the Scheduled PBFDAF is more complex, we only looked at the Alternative PBFDAF.
4.4 Complexity Analysis

While the memory usage of this proposed PBFDAF algorithm is identical to the fully constrained PBFDAF, we show that the computational complexity is less for the proposed algorithm. The computational complexity is measured by means of additions, multiplications, divisions and shift operations individually. Formula 4.13 shows the computational complexity per input sample of the proposed PBFDAF. We note that the first formula-part is the computational complexity for the unconstrained PBFDAF, while the second part is concerned with the proposed constraining mechanism.

\[
\Psi_{\text{PBFDAF}} = \frac{3 \cdot \Psi_{\text{FFT}}(2N) + 2K \cdot \Psi_{\oplus}(2N) + \Psi_{\Phi}(2N)}{N} + \]

\[
\]
4.4. COMPLEXITY ANALYSIS

\[
2^{P-1} \cdot \Psi_{FFT} \{2N\} + P^{-1} \Psi_M \{2N\} + (K - P^{-1}) \cdot \Psi_{HS} \{2N\},
\]  

(4.13)

where \( P \) is the alternating constraint period. The computational complexity of the high-slope constraint window is indicated with \( \Psi_{HS} \{2N\} \) and may be approximated as follows:

\[
\Psi_{HS} \{2N\} \approx 4 \cdot \Psi_A \{2N\} + \Psi_M \{2N\}.
\]  

(4.14)

We note that the initial convolution calculation of formula 4.11 is neglected in this analysis. Furthermore, we note that \( P^{-1} \Psi_M \{2N\} \) in formula 4.13 is concerned with the \( N \) point multiplication by \( a^{-1} \). We calculate this computational complexity for the PBFDAF with the proposed constraining mechanism for a 1024 taps filter with different partitions \( K \). We assume that \( P = 1 \) and \( Q = 1 \). Table 4.1 and Figure 4.8 shows the computational complexity per input sample:

<table>
<thead>
<tr>
<th>( K )</th>
<th>adds/subs ( A )</th>
<th>muls ( M )</th>
<th>divs ( D )</th>
<th>( \Psi_{PBFD} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140.37</td>
<td>64.04</td>
<td>1.00</td>
<td>205.41</td>
</tr>
<tr>
<td>2</td>
<td>143.06</td>
<td>63.38</td>
<td>1.00</td>
<td>207.44</td>
</tr>
<tr>
<td>4</td>
<td>161.70</td>
<td>68.77</td>
<td>1.01</td>
<td>231.47</td>
</tr>
<tr>
<td>8</td>
<td>212.29</td>
<td>86.14</td>
<td>1.02</td>
<td>299.45</td>
</tr>
<tr>
<td>16</td>
<td>326.30</td>
<td>127.69</td>
<td>1.03</td>
<td>455.02</td>
</tr>
</tbody>
</table>

Table 4.1: Computational complexity for the Alternative PBFD A with high-slope constraint approximations.

We see that compared with the fully constrained PBFD A a large reduction in computational complexity is achieved.
4.5 Simulations

The algorithm as described in Section 4.3 is implemented in the Acoustic Performance and Enhancement System (APES), which is a library with signal processing routines used at the Philips Research. To illustrate the convergence behaviour of the proposed algorithm, we compare the results with the full-constrained algorithm. All simulations are performed with a 1024 taps PBFDAF with 8 partitions. Furthermore we varied the step-size $2\alpha$ in the range $0.5$ to $2.5\frac{1}{K}$ with steps of $0.05\frac{1}{K}$, thus performing the simulation with 41 different step-sizes. The results of the simulations are depicted in graphs where the steady-state NMSE is plotted as a function of the CTC.

4.5.1 Convergence characteristics for white Gaussian noise

In the first simulation we use zero-mean white Gaussian noise. The simulation is performed with two different values of the exact-constraint period (i.e. 1 rectangular constraint every 1 respectively 2 block iterations). For this simulation we use the artificial impulse response of Figure 1.4. The results are shown in Figure 4.9.

![Figure 4.9: Simulation results of the Alternative PBFDAF with high slope constraint approximations in the case of white noise.](image)

In the above figure the steady-state NMSE is plotted as a function of the Convergence Time Constant (CTC). These curves are convex and the maximum CTC lies at the right side of the figure.

We see that the PBFDAF with the proposed constraining mechanism has similar results compared to the constrained PBFDAF.
4.5.2 Convergence characteristics for coloured noise

We also performed the simulation in the case of coloured noise. We generated this coloured noise by passing it through the bandpass colouring filter and used the artificial exponentially decaying signal as impulse response, both mentioned in Section 1.4.3. We obtained the following results:

![Figure 4.10: Simulation results of the Alternative PBFDAF with high slope constraint approximations in the case of coloured noise.](image)

Observing the results of the simulations for white and coloured noise, we clearly see that the convergence behaviour is almost as good as the constrained PBFDAF. The maximum CTC of the new proposed constraining method is slightly lower than the maximum CTC of the fully constrained method. We also see that the steady-state NMSE is slightly impaired for high values of the step-size.

The simulations show that a quite good performance can be obtained with the constraining method based on high-slope constraint windows and there is only a small degradation compared with the fully constrained PBFDAF. We however note that this performance degradation of the new constraining method is larger in the case where the real impulse response of Figure 1.5 is used, instead of the artificial one used in the simulations presented in this chapter. The next chapter describes the reason for this degradation and proposes a second constraining mechanism with even better performance.
Chapter 5

Compensation method

*In the previous chapter we proposed a constraining mechanism based on constraint approximations and found that this constraining mechanism does not perform as well as the fully constrained PBFDAF. This performance degradation is due to a drawback in the alternative constraining method. In this chapter we investigate this drawback and propose a solution for eliminating it, which we call the compensation method. Finally we present an efficient constraining mechanism for the PBFDAF based on a first-order approximation constraint and the compensated alternative constraining mechanism. By means of simulations we show that a near optimal performance is achieved.*

5.1 Drawbacks in Alternative PBFDAF

In the previous chapter we proposed a method for efficient constraining in the PBFDAF algorithm by means of the Alternative PBFDAF in combination with constraint approximations. The results of this method are non-optimal compared with the fully constrained PBFDAF. In this section we show that this non-optimal performance is caused by the alternative constraining mechanism.

In order to show the disadvantages of the Alternative PBFDAF, we first take a look at the convergence curve of the alternative PBFDAF with a zero-mean, unit-variance white Gaussian noise as input signal. Furthermore we use the real measured impulse response of Figure 1.5 for this experiment, as this impulse response contains more power in the direct-field component and the drawbacks in the Alternative PBFDAF are more visible. The constraints are applied as indicated in formula 3.7, but we note that the exact sequence of the constrained partitions is not relevant. One rectangular constraint is applied every 25 block-iterations of the PBFDAF algorithm \((P = 25)\). The convergence curve of this simulation is depicted in Figure 5.1. In block-iteration \(25 \cdot i\), a rectangular constraint is applied on partition \(i\). After the last partition is constrained, the sequence is repeated. At block-iteration 225 for example, the first partition is constrained for the second time.

We clearly see that there are problems at the moments where rectangular constraints are applied. The same phenomenon is also slightly visible in the article of Estermann [14], but
this effect is not noticed by the author. Since the degradation in the NMSE occurs exactly after the moment that the rectangular constraint is applied, the removal of the circular-errors is responsible for this phenomenon. Apparently the accumulated circular-errors in the adaptive weights somehow contribute to the model of the room impulse response, and removing these circular-errors by means of rectangular constraining results in a temporary degradation of the NMSE in the next block-iteration.

![Graph showing NMSE vs. K for different configurations of the Alternative PBFDAF algorithm.](image)

**Figure 5.1: Convergence curve of the Alternative PBFDAF algorithm.**

We also see that the jumps in the convergence curve are very large in the first two partitions. This can be explained due to the fact that these first partitions model that specific part of the room impulse response which contains the most energy.

We note that a real impulse response is used for the generation of the convergence curve in Figure 5.1. When we use the artificial exponential decaying impulse response of Figure 1.4, the temporary jumps are smaller. This is caused by the fact that the real impulse response contains a large direct-field energy part.

### 5.2 Compensation of wrap-around artifacts

We start with an analysis, where we demonstrate that it is useful to compensate some of the wrap-around artifacts. This analysis is done by means of graphical construction of the convolution part in the adaptive filter. After this analysis we show how the compensation signals can be constructed. Finally we combine the compensated alternative constraining method with constraint approximations.
5.2. COMPENSATION OF WRAP-AROUND ARTIFACTS

5.2.1 Circular wrap-around artifacts

We can eliminate the temporary degradation in the convergence curve of the Alternative PBFDAF by compensating the removed circular-errors in the the other partitions after a rectangular constraint is applied. As a result, the output of the partitioned convolution is approximately the same as when the constraint was not applied. For explaining this method, we consider Figure 5.2 which shows the input-blocks in time-domain.

![Figure 5.2: Input blocks for the different partitions.](image)

After the circular correlation of the residual signal with the input signal blocks is performed, the result \( x_i^{2N}[kN] \) contains the wanted linear correlation in the range \([0..N - 1]\), while the range \([N..2N - 1]\) is polluted with circular wrap-around artifacts. In the fully constrained PBFDAF all these wrap-around artifacts are eliminated. In this analysis we assume that the results near the correlation outputs \( N \) and \( 2N \) can be considered to approximate a linear correlation. In the PBFDAF without power normalization, this can be easily understood by correlating the residual signal in time-domain (augmented with \( N \) zeros) with the input signal blocks of Figure 5.2. For the correlation output points higher but close to \( N \), the amount of wrap-around artifacts is low. This is also true for the points lower but close to \( 2N \).

For correlation output points in the direction of \( \frac{3}{2}N \) the approximation gets worse as the amount of wrap-around artifacts increases. Therefore we refer the correlation output points higher but close to \( N \) as being the left tail and the points lower but close to \( 2N \) as being the right tail. As the right tail approximates the linear correlation for negative time lags, it is better to change the principal interval as is depicted in Figure 5.3.

![Figure 5.3: Principal interval of the adaptive weights that correspond with the linear correlation.](image)

1We note that the element \( x_i^{2N}[kN] \) is a value that represents a linear correlation result, as the Overlap-Save with FFT size \( 2N \) is capable of generating \( N + 1 \) linear results [31].
As both tails are approximate results of a linear correlation, the coefficients of the tails in the adaptive weights $w_i^{2N}[\kappa N]$ (in time-domain) converge to a value that is approximately correct. The tails of these adaptive weights can be successfully used for eliminating the temporary degradation in the convergence curve in case we use the Alternative PBFDAF.

On the other hand, the correlation results in the neighbourhood of $\frac{3}{2}N$ (with a high amount of wrap-around artifacts) are useless, due to the fact that there is an approximately 50% wrap-around between the two input blocks of length $N$. Hence, these correlation results have nothing to do with linear correlations. As these correlation results can be assumed random, the adaptive weights $w_i^{2N}[\kappa N]$ in the adaptive filter are also updated randomly. As a result, these parts in the adaptive weights are fluctuating and do not converge to a stable value. Therefore we refer to these values in-between the two tails as the noisy part.

The circular wrap-around artifacts introduce errors in the convolution part of the adaptive filter, as neighbour partitions in this convolution are coupled [10]. Because the two tails contain values that are approximate results of linear correlations, we can reuse these two tails on the moments where the rectangular constraint is applied in the Alternative PBFDAF. This is done in such a way that the output of the block convolution in the adaptive filter remains approximately unchanged, while all circular wrap-around artifacts (i.e. the two tails and the noisy part) in that specific partition are eliminated. A method for achieving this, is by compensating the left tail into the next partition and compensating the right tail into the previous partition.

We explain the method of compensation graphically. In Figure 5.4 we see the two weight vectors of the adaptive filter.

![Figure 5.4: Weight vectors with the input blocks.](image)

For the adaptive weight vector $w_i^{2N}[\kappa N]$, the left part represents the weights based on the correct linear correlations. The right side of this vector contains weights polluted with circular wrap-around artifacts. The weights are constructed out of the two tails as discussed earlier. The left tail is indicated with the number 1, while the right tail is indicated with number 2. In between these tails, the weights are noisy and therefore useless.

### 5.2.2 Compensation method

In the previous section we discussed the weights in the adaptive filter as being constructed out of results related with linear correlations and the circular wrap-around artifacts. It is
also explained that a part of these wrap-around artifacts (the two tails) are useful results. The compensation discussed in this section explains how these tails can be reused. As we have two useful tails we have a backward and a forward compensation. Figure 5.5 and 5.6 are used for explaining the forward and backward compensation.

\[
\begin{align*}
&x_i^{2N}[\kappa N] \\
&J_0^{2N} \cdot w_i^{2N}[\kappa N] \\
&J_{2N}^{2N} \cdot w_i^{2N}[\kappa N] \\
&J_{2N-1}^{2N} \cdot w_i^{2N}[\kappa N]
\end{align*}
\]

Figure 5.5: Destination of the backward compensation signal.

The circular convolution of \(x_i^{2N}[\kappa N]\) with \(w_i^{2N}[\kappa N]\) can be performed graphically \([13]\) by mirroring one signal (e.g. the adaptive weights) and moving this signal to the right; this is indicated in Figure 5.5.

The total sum of products of both signals in one convolution-iteration is the output-element of that specific iteration. As we perform a circular convolution, shifting the adaptive weights is actually a rotation of the \(2N\) length vector. The convolution output can be described by the following formula:

\[
(x_i^{2N}[\kappa N] \otimes w_i^{2N}[\kappa N])_k = \sum_{m=0}^{2N-1} (x_i^{2N}[\kappa N] \otimes (J_k^{2N} \cdot w_i^{2N}[\kappa N]))_m.
\]

In Figure 5.5, three stages of graphical construction of the convolution-output are shown. We note that this figure shows the circular convolution in time-domain while in our efficient implementation, the circular-convolution is performed in frequency-domain.

In Figure 5.5, we see the graphical construction of the correlation-output for \(k = 0, N\) and \(2N-1\). The convolution-results of \(k = 0\) to \(k = N - 1\) don't play any role, as the window after the IFFT in the PBFDARF algorithm discards these points (see Figure 2.5). For the points \(N\) to \(2N - 1\) in \(w_i^{2N}[\kappa N]\), we see that convolution of tail 1 is largely concerned with input block \(B2\). When we assume that this tail is small, this convolution can be approximated by the convolution of input block \(B2\) with this tail in the next partition. This is achieved by removing tail 1 in the right side of the adaptive weights and placing this tail inside the first \(N\) points of the adaptive weight of the next partition, depicted in Figure 5.6. We see that for small tails, the convolution output concerning the original tail 1 in this partition is approximately equal to the convolution output of this tail in the next partition.
On the other hand, we see that in Figure 5.6 there is also a right tail indicated by the number 2. For the points $N$ to $2N - 1$ of the convolution output, we see that the convolution of tail 2 is also largely concerned with block $B2$. For convolution outputs close to $2N - 1$, there is only a small wrap-around with block $B3$. Again, for small tails, this convolution of tail 2 with input block $B2$ can also be obtained (approximately) in the previous partition. The destination of this tail 2 is depicted in 5.5.

![Figure 5.6: Destination of the forward compensation signal.](image)

When the adaptive weights are shown with modified principal intervals as in Figure 5.3, we can visualize the compensation in the partition $i$ as follows:

![Figure 5.7: Compensation of circular wrap-around artifacts.](image)

We note that this may work well for the middle $K - 2$ partitions, but there are problems for the first and the last partition. This topic is discussed in Section 5.2.4.
5.2.3 Construction of compensation signals

For every weight vector constrained with a rectangular constraint, the forward and backward compensation signal $C_{f_i}^{2N} [\kappa N]$ and $C_{b_i}^{2N} [\kappa N]$ is generated. The weight vector and compensation signals can be constructed in the following way:

![Diagram](image)

Figure 5.8: Generation of $W_i^{2N} [\kappa N]$, $C_{f_i}^{2N} [\kappa N]$ and $C_{b_i}^{2N} [\kappa N]$.

For reasons of transparency, all signals are constructed in time-domain. If we consider the unconstrained weight-vector as consisting of a linear convolution part and tailed circular wrap-around artifacts (as in Figure 5.3), the signals 1 to 6 of Figure 5.8 can be visualized as depicted in Figure 5.9.

![Time-domain signals](image)

Figure 5.9: Time-domain signals of Figure 5.8.

When we now perform the same experiment as in Section 5.1, but with the compensation of the circular wrap-around artifacts, we get a convergence curve as shown in Figure 5.10. This figure also shows the Alternative PBFDAF (without compensation). The positive effect of the compensation is clearly visible on the moments rectangular constraints are applied. The Alternative PBFDAF shows large degradation in NMSE on these moments, whereas the compensated version shows an improvement. This improvement is due to the fact that the the circular wrap-around artifacts are constrained to zero, while the useful information of the approximately linear correlation results is saved.

We clearly see that the compensation of the circular-errors from one partition into the neighbour partitions has a positive effect on the convergence time-constant. We note that for the other simulations in this chapter, we do not use one rectangular constraint every 25 block-iterations ($P = 25$), but use a smaller value of $P$. We compensate $\frac{N}{2}$ time-domain points out of the circular wrap-around artifacts into the next and $\frac{N}{2}$ time-domain points into the previous partition. Only a part of these $\frac{N}{2}$ points are approximately linear (i.e. the tails in the figures) and thus useful, while the points between the tails are polluted with artifacts that have approximately 50% wrap-around, and can be assumed to be noisy and useless. When we also compensate the noisy artifacts
as depicted in Figure 5.8, the steady-state NMSE is slightly worsened. A solution for this problem is not to compensate the noisy part and only compensate the two tails. However, it is not obvious how many coefficients need to be taken into account for these tails. We could make an arbitrary choice for the number of coefficients that is actually compensated, but we can better use the compensated alternative constraining method in combination with the use of sinusoidal windows as an approximation of the rectangular constraint. The sinusoidal windows constrain the noisy parts in the update, while two tails remain for the compensation. This topic is discussed in the next section.

5.2.4 Compensation of sinusoidal constrained partitions

Using the unconstrained PBFDAF, the convergence behaviour is impaired and the region of convergence is smaller than with the fully constrained method. When applying a sinusoidal constraint approximation $g^{2N}$ in combination with the alternative constraining method, improved results are obtained, compared with the unconstrained algorithm. The sinusoidal approximation constraint $g^{2N}$ is defined as:

$$
(g^{2N})_i = \frac{1}{2} \left(1 + \sin \left(\frac{\pi \cdot i}{N}\right)\right), \quad 0 \leq i < 2N. \quad (5.2)
$$

When applying sinusoidal constraints, the noisy parts in the neighbourhood of $\frac{3}{2}N$ are reduced. On the other hand, the tails are not effectively reduced by the approximation constraint, but can be successfully compensated into the neighbour partitions with the method described in the previous section.
The application of the rectangular windows of length $\frac{N}{2}$ in Figure 5.8 requires two extra FFTs. This complexity can be reduced by replacing the rectangular windows by a raised cosine and a raised inverse cosine window, as indicated in Figure 5.11.

\[ (g^{2N} \otimes z^{2N}[\kappa N]) + w^{2N}[\kappa N] + C^{2N}_r[\kappa N] + C^{2N}_i[\kappa N] \]

Figure 5.11: Efficient generation of $W^{2N}[\kappa N]$, $C^{2N}_r[\kappa N]$ and $C^{2N}_i[\kappa N]$.

Again all signals in Figure 5.11 are constructed in time-domain for transparency reasons. The raised cosine windows can be easily implemented in frequency-domain without the use of extra FFTs (see Section 5.3). The signals 1 to 6 are the same as in Figure 5.9 except that the noisy part is largely reduced by the sinusoidal constraint approximations. The windows for the backward compensation in the range $N < i < 2N$ can be calculated by multiplying the sinusoidal constraint window and the raised inverse cosine window. Figure 5.13 shows the window for the backward compensation.

Figure 5.13: Construction of the backward compensation window.

Figure 5.13 shows the construction of the time-domain window for extracting the backward compensation signal. The sinusoidal constraint window and the raised cosine window are depicted and the compensation window is generated by multiplying these two
windows. The construction of the forward compensation window is depicted in Figure 5.14.

![Figure 5.14: Construction of the forward compensation window.](image)

The adaptive weights in the different partitions are sinusoidal shaped compared with the weights in the fully constrained PBFDAF. Furthermore we have tails that are useful and have the shapes as depicted in Figure 5.13 and 5.14. When we add these two parts together, we get the overall time-domain weighting in the PBFDAF update, as depicted in Figure 5.15. In this figure, four partitions are depicted ($K = 4$). We note that for the original PBFDAF algorithm, the time-domain weightings are equal to 1 for all coefficients. When we look at Figure 5.15, the middle $K - 2$ partitions have quite flat time-domain weightings, while for the first and last partition there are differences.

![Figure 5.15: Overall weighting in the PBFDAF update ($K = 4$).](image)

\(^2\)We note that the minima in the range $\frac{1}{2}N$ to $3\frac{1}{2}N$ are quite optimistic, as these adaptive weights are polluted with a relatively high amount of circular wrap-around artifacts. For better overall weightings one should use higher order constraint approximations (e.g. 3rd order). However the sinusoidal constraint approximations already give satisfactory results (see Section 5.5).
For the first partition, we do not want to compensate the right tail into the previous partition, as this tail is concerned with correlations that are not causal with respect to the input-signal \(^3\). Hence we can omit the compensation of the right tail of the first partition into the non-existent previous partition. Due to this non-existent partition, there also is no forward compensation of the left tail into the first partition. As a consequence, the first \(\frac{N}{2}\) time-domain points of the weights in Figure 5.15 have the shape of the sinusoidal constraint. Usually this would lead to reduced convergence speed of the adaptive filter. However, in most cases the first points of the impulse response to be modelled by the adaptive filter contain zeros, due to the transmission delay between the loudspeaker and the microphone. As a result, the first time-domain weights of the adaptive filter are fluctuating around zero with a smaller variance and there is a small improvement in steady-state NMSE.

For the last partition \(K - 1\) there is a similar problem. The compensation of the right tail of partition \(K\) into partition \(K - 1\) cannot be performed, as partition \(K\) is non-existent. Again, this leads to a sinusoidal shape in the time-domain points \(KN - \frac{N}{2} - 1\) to \(KN - 1\). As usually the impulse response to be modelled by the adaptive filter contains less energy in the last time-domain points due to the energy decay, this does not lead to reduced convergence speed. The forward compensation of the left tail in partition \(K - 1\) also cannot be performed due to the non-existence of partition \(K\). We can choose to omit this forward compensation. However, this means that a useful forward tail would be thrown away. It is might better be to maintain this useful tail in the last partition \(^4\). This leads to the sinusoidal shape of the last \(\frac{N}{2}\) points in Figure 5.15. With these extra time-domain points, the order of the adaptive filter is increased, as will be shown in a simulation of Section 5.5.5. We note however that maintaining the left tail in the weights of the last partition introduces errors in the convolution part, that might lead to audible artifacts.

\(^3\)In theory, the residual samples cannot be correlated with input-samples that are not collected yet, so this tail is not useful.

\(^4\)For the middle \(K - 2\) partitions the tails cannot be maintained as they cause a coupling between neighbour partitions. However as the left tail of the last partition does not have a coupling with the next non-existent partition, this left tail can be maintained.
5.3 Proposed constraining mechanism

In Section 5.2.4 the method of compensating wrap-around artifacts into the next and previous partitions is explained. The proposed constraining mechanism is depicted below:

In Figure 5.16 we can see that the compensation leads to connections between the update blocks $A_i$ of neighbour partitions. For the first partition, there is no compensation to the previous partition, as explained in Section 5.2.4. For the last partition we omit the compensation to the next (non-existent) partition. Omitting this forward compensation for...
the last partition leads to removal of circular wrap-around artifacts which could be useful for modelling a higher order impulse response (see Section 5.2.4). The update blocks \( A_i \) in Figure 5.16 exploit the compensation method as also explained in the previous section. The structure of these update blocks is depicted in Figure 5.17.

The sinusoidal constraint is placed before the coefficient update, while the rectangular constraint is placed after the coefficient update; this is explained in Section 3.1. The compensation signals are indicated with \( C_{f,2N}[k,N] \) (forward) and \( C_{b,2N}[k,N] \) (backward). When no rectangular constraint is applied the compensation signals are omitted (i.e. state 0 of the switches). A linear way of rectangular constraining is applied, where the switches are described by the following formula:

\[
S_i = \begin{cases} 
1 & \text{if } \kappa \mod (K + P) = i \\
0 & \text{otherwise.}
\end{cases}
\]  

(5.3)

The use of an approximation constraint when also a rectangular constraint is applied seems to be needless. In the case of the un-compensated alternative constraining in Section 4.3 this is true. However the compensation after applying a rectangular constraint also compensates the extra noise in the wrap-around artifacts (due to the omitting of the sinusoidal constraint). Hence this leads to degradation in convergence behaviour. Therefore it is preferred to also apply the approximation constraint when applying a rectangular constraint with the compensation (see Figure 5.17).

When implementing the rectangular constraint together with the compensation, we need additional computational complexity for the compensation part. This is because of the need of splitting up the two tails for the compensation compensation in the next and previous partition and doing a time-domain shift of the two individual tails. A direct implementation as in Figure 5.11 is possible, but needs an abundant amount of extra computational complexity as all operations are performed in time-domain. A more efficient implementation can be obtained when generating the signals in frequency-domain.

The shift in the time-domain is a \( N \) point shift of a \( 2N \) point signal \( h_{2N} \), resulting in the \( 2N \) point signal \( h_{2N}^{2N} \). In frequency-domain these signals are represented by \( H_{1,2N} \) and \( H_{2,2N} \) respectively. This shift can be easily implemented in the frequency-domain via lemma 5.3.1 which uses the shift-theorem [3], page 111.

**Lemma 5.3.1** If \( h_{2N}^{2N} = D_{2N}^{2N} \cdot h_{2N}^{2N} \), then:

\[
(H_{2,2N})_k = \begin{cases} 
+ (H_{1,2N})_k & \text{for } k \text{ even} \\
- (H_{1,2N})_k & \text{for } k \text{ odd.}
\end{cases}
\]  

(5.4)

**Proof:**

\[
(x_2^{2N})_{n-i} \leftrightarrow (x_2^{2N})_k \cdot e^{-j\frac{\pi k i}{2N}},
\]
where \( i \) is an integer value

\[
(h_2^{2N})_n = (h_1^{2N})_{n-N}
\]

\[
(h_2^{2N})_k = e^{-j\frac{2\pi kn}{2N}} (h_1^{2N})_k
\]

\[
e^{-j\pi k} (H_1^{2N})_k
\]

\[
= \begin{cases} 
+ (H_1^{2N})_k & \text{for } k \text{ even} \\
- (H_1^{2N})_k & \text{for } k \text{ odd}.
\end{cases}
\]

The generation of the time-domain signals 2, 3 and 4 of Figure 5.11 can be easily calculated in frequency-domain. The figure below shows this efficient implementation.

![Figure 5.18: Efficient calculation of the compensation signals and the constrained weight vector.](image)

Figure 5.18: Efficient calculation of the compensation signals and the constrained weight vector.

We note that \((-1)^{2N})_i = (-1)^i\). The elementwise multiplication of a frequency domain vector by this vector implements the time-domain shift of \(N\) points. As the compensation signals need to be added to the neighbour partitions, these elementwise multiplications by \(-1\) and \(1\) can be omitted if we use both additions and subtractions. The raised cosine window is now applied in frequency-domain. The other (raised inverse cosine window) is calculated by simple subtraction operations and saves computational complexity. It is also noted that the multiplications by \(\frac{1}{2}\) and \(\frac{1}{4}j\) can be implemented as simple shift operations.

We already mentioned that it is not possible to perform the forward compensation in the last partition. Usually the circular wrap-around artifacts of this left tail are nulled by the rectangular constraint. As the wrap-around artifacts of this tail can be used for a higher order (> \(KN\)) adaptive filter, we could leave this tail unconstrained. The implementation of the compensation in the last partition is depicted in Figure 5.19.

We note that the computational complexity of this implementation is slightly lower compared to the implementation of Figure 5.18.

We already mentioned that the first partition in the PBFDAF does not retrieve a forward compensation signal from its predecessor partition, and the first partition does not have to generate a backward compensation signal. Hence the implementation of the update-block in the first partition can also be implemented with reduced computational complexity.
5.4 Complexity Analysis

While the memory usage of this PBFDAF algorithm with the proposed constraining mechanism is exactly the same as in the case of the fully constrained PBFDAF, we show that the computational complexity of the proposed constraining mechanism is lower. As explained before, the computational complexity is measured by means of additions/subtractions, multiplications, divisions and shift operations individually. The computational complexity per input sample of the proposed PBFDAF is as follows:

\[
\Psi_{\text{PBFDAF}} = \frac{3 \cdot \Psi_{\text{FFT}}[2N] + 2K \cdot \Psi_{\Theta}[2N] + \Psi_p[2N]}{N} + \frac{P^{-1} \cdot (2 \cdot \Psi_{\text{FFT}}[2N] + 4 \cdot \Psi_A[2N] + \Psi_{\text{COS}}[2N]) + K \cdot \Psi_{\text{SIN}}[2N]}{N},
\]

(5.5)

where \( P \) is the period for performing the rectangular constraints. Compared with the fully constrained PBFDAF, we see that the \( 2K \) FFTs are eliminated, just as in the case of the Unconstrained PBFDAF. However, we have additional computational complexity for the proposed constraining mechanism. The first part of the formula is in fact the computational complexity of the Unconstrained PBFDAF, while the second part in the formula is concerned with the proposed constraining mechanism. The \( \Psi_{\text{SIN}}[2N] \) and \( \Psi_{\text{COS}}[2N] \) are respectively the sinusoidal constraint window (for the constraint approximation) and the raised cosine window (for the compensation). The computational complexity of these windows is given by:

\[
\Psi_{\text{SIN}}[2N] = 3 \cdot \Psi_A[2N] + \Psi_S[2N], \quad (5.6)
\]

\[
\Psi_{\text{COS}}[2N] = 3 \cdot \Psi_A[2N] + 2 \cdot \Psi_S[2N]. \quad (5.7)
\]

We note that the single shift operation needed for convolution of the mean value (0.5) in the sinusoidal constraint window is omitted in the complexity analysis.

We calculate this computational complexity for the PBFDAF with the proposed constraining mechanism in the case of a 1024 taps adaptive filter with different partition factors \( K \). Furthermore we assume that the rectangular constraining period \( P = 1 \). The table below
Table 5.1: Computational complexity for the Compensated Alternative BFDAF with sinusoidal constraint approximations.

<table>
<thead>
<tr>
<th>$K$</th>
<th>adds/subs $A$</th>
<th>muls $M$</th>
<th>divs $D$</th>
<th>shifts $S$</th>
<th>$\Psi_{PBFDAF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>160.37</td>
<td>62.04</td>
<td>1.00</td>
<td>6.00</td>
<td>229.41</td>
</tr>
<tr>
<td>2</td>
<td>161.06</td>
<td>59.38</td>
<td>1.00</td>
<td>8.00</td>
<td>229.44</td>
</tr>
<tr>
<td>4</td>
<td>175.70</td>
<td>60.77</td>
<td>1.01</td>
<td>12.00</td>
<td>249.47</td>
</tr>
<tr>
<td>8</td>
<td>218.29</td>
<td>70.14</td>
<td>1.02</td>
<td>20.00</td>
<td>309.45</td>
</tr>
<tr>
<td>16</td>
<td>316.30</td>
<td>95.69</td>
<td>1.03</td>
<td>36.00</td>
<td>449.02</td>
</tr>
</tbody>
</table>

shows the computational complexity per input sample in this situation:

We see that compared with the fully constrained PBFDAF a great reduction in computational complexity is achieved for small block length $N$. The total number of operations (i.e. $\Psi_{PBFDAF}$) can be plotted in a graph as a function of the FFT-length (i.e. $2N$). This graph is depicted in Figure 5.20.

![Figure 5.20: Number of operations in the PBFDAF for different partitions.](image)

Compared with the Unconstrained PBFDAF, the computational complexity for the PBFDAF algorithm with the proposed constraining mechanism is only a constant number of operations higher for every number of partitions $K$.

---

5For higher values of the alternative constraining period $P$, even a higher reduction in computational complexity is achieved.
5.5 Simulations

The algorithm as described in Section 5.3 is implemented in the Acoustic Performance and Enhancement System (APES). To illustrate the convergence behaviour of the proposed proposed algorithm, we compare the results with the fully constrained algorithm. All simulations are performed with a 1024 taps PBFDAF with 8 partitions.

5.5.1 Influence of the compensation

As a first impression, we first do a simulation of the proposed constraining method, with and without the compensation, but both with the approximation constraint included. We use the real impulse response of Figure 1.5, as the temporary jumps in the NMSE are larger in this case. The input-signal is a zero mean, unit variance white Gaussian noise signal. Furthermore we choose the step-size parameter $2\alpha = \frac{0.5}{4}$ and the Alternative constraining period $P = 2$. The results of the simulation are depicted below:

![Graph showing influence of compensation](image)

Figure 5.21: Influence of the compensation in the Compensated Alternative PBFDAF with sinusoidal constraint approximations.

In the simulation results we clearly see that the compensation is very effective for obtaining good convergence results. As can be seen in the figure, the convergence curve of the compensated situation nearly coincides with the convergence curve of the fully constrained situation. After adaptation nearly the same steady-state NMSE is obtained for all situations.
5.5.2 Convergence curve for white Gaussian noise

This simulation is also performed in the case of zero-mean unit-variance white Gaussian noise, with two different values of the rectangular constraining period (i.e. 1 rectangular constraint every 1 respectively 2 block iterations). Furthermore, we use the artificial impulse response of Figure 1.4.

![Simulation results of the Compensated Alternative PBFDAF with sinusoidal constraint approximations in the case of white noise.](image)

Figure 5.22: Simulation results of the Compensated Alternative PBFDAF with sinusoidal constraint approximations in the case of white noise.

We see that the Compensated Alternative PBFDAF nearly coincides with the fully constrained PBFDAF for both values of the rectangular constraining period, but a slight shift can be observed with respect to the steady-state NMSE of the convergence curve. For higher values of the constraint period \( P \), we get better values for the steady-state NMSE. The reason for this phenomenon lies in the fact that the un-compensated left tail of the last partition leads to a higher-order model in the adaptive filter. The lower the alternative constraining period \( P \), the faster these higher-order coefficients are constrained by means of the rectangular constraint. As the useful (approximately) linear correlation result is thrown away with a faster rate, this leads to a lower steady-state NMSE compared with higher values of \( P \). Therefore it seems to be better that we leave the left tail of the last partition unconstrained. More extensive simulations on this topic are discussed in Section 5.5.5.

We also see that the region of convergence for the proposed constraining mechanism is slightly improved compared with the fully constrained PBFDAF. The main reason for this is the fact that the first partition does not receive the forward compensation of the previous partition. As mentioned at the end of Section 5.3, this would generally lead to a small degradation compared to the fully constrained PBFDAF. However, as the impulse
5.5. SIMULATIONS

response model used in this simulation is preceded with 20 zeros (see formula 1.6), we do not suffer from this degradation and even obtain better convergence behaviour, as the first 20 coefficients are less fluctuating around the zero level. We note that better convergence behaviour is also obtained due to the local minima overall weighting (see Figure 5.15).

5.5.3 Convergence curve for coloured noise

We also performed the simulation in the case of coloured noise. We generated this coloured noise by passing it through the bandpass colouring filter and used the artificial impulse response of Figure 1.4, both as mentioned in Section 1.4.3. We obtained the following results:

![Simulation results of the Compensated Alternative PBFDAF with sinusoidal constraint approximations in the case of coloured noise.](image)

Figure 5.23: Simulation results of the Compensated Alternative PBFDAF with sinusoidal constraint approximations in the case of coloured noise.

We see that approximately the same results are obtained as in the white noise situation. Again we see the slight shift in steady-state NMSE due to the higher order modelling in the last partition. We also see the improvement in the region of convergence which can be explained due to the absence of the forward compensation into the first partition.

From the simulation in this and previous section we can conclude that the Compensated Alternative PBFDAF in combination with the constraint approximations works very well. As these simulations both use stationary noise signals as input signals, we also perform an experiment which uses a non-stationary input signal. Therefore we use a real-life music signal, which is assumed to be non-stationary and highly correlated.
5.5.4 Simulation with real music signal

In the case of white noise and coloured noise input signals the above simulation show that the convergence curve of the proposed constraining method is approximately equal to the fully constrained method. These simulations were both performed with an artificial exponentially decaying randomly generated signal for the impulse response. As a final verification we perform a simulation with a real (highly correlated) non-stationary music signal for the input signal. In this simulation we also use a real impulse response of a room (as used in previous simulations). The results are obtained for a step-size of $2\alpha = \frac{0.5}{K}$.

Contrary to the previous simulations with stationary noise signals, we cannot perform an ensemble averaging of several independent simulation-results. Therefore, to obtain smooth results, the convergence curve is smoothed with a moving average filter of order 64. The coefficients of this filter are all chosen equally. Furthermore, we note that the constraining period is set to the arbitrary value of $P = 2$. The simulation results are depicted in Figure 5.24.

![Figure 5.24: Comparison of different constraining methods for a music input-signal.](image-url)

The simulation shows that in the case of a highly correlated music signal the proposed constraining method also gives equal results as the fully constraining method. We also see that without the compensation the convergence speed is significantly impaired. Hence we can conclude that the proposed PBFDAF algorithm is a good and computationally efficient alternative over the fully constrained PBFDAF algorithm.
5.5.5 **Simulation with unconstrained left tail**

In the simulations of Section 5.5.2 and 5.5.3 it is stated that the plots are slightly shifted in the direction of the steady-state NMSE. For higher values of the alternative constraining period $P$, we get improved values for the steady-state NMSE. This is caused by the left tail in the last partition which models an additional part of the impulse response. The fully constrained PBFDAF has an adaptive filter of order $KN$ time-domain points, while with the extra left tail, the adaptive filter is capable of modelling a higher order impulse response. In order to prove that the slightly shifted curves are due to this phenomenon, we perform a similar simulation as in Section 5.5.3 with coloured noise. In this simulation we first perform a measurement, where we zero the higher order coefficients $w_{K-1}^{2N}$, $N \leq i < \frac{3}{2}N$, all the time. As a result, there is no build up of circular wrap-around errors in the left tail of the last partition. In the second measurement, we perform a simulation where we do not constrain the coefficients $w_{K-1}^{2N}$, $N \leq i < \frac{3}{2}N$, leading to an unconstrained left tail of the last partition. In both measurements we use the same conditions as in Section 5.5.3. The simulation results are depicted in Figure 5.25.

![Graph showing influence of constrained and unconstrained left tail of the last partition.](image)

Figure 5.25: Influence of constrained and unconstrained left tail of the last partition.

We see that with the constraining of the left tail in the last partition, the graph is almost equal to the fully constrained PBFDAF. Without this constraining of the left tail in the last partition, we achieve a lower steady-state NMSE. We can conclude that lower steady-state NMSE is achieved due to the higher order modelling in the last partition.  

---

6We note that the unconstrained left tail in the last partition might introduces audible artifacts as the convolution part is only correct in case the last $N$ values of the weight vector are nulled. Hence it may be safer to perform the alternative constraining for both tails in the last partition.
Chapter 6

Acoustical parameter estimation

The Dynamic Echo and Noise suppressor (DENS) is used as a post-processing device for the adaptive filter. It causes the residual echos from the adaptive filter to be reduced even further. As the current implementation of the DENS is based upon two fixed acoustical parameters, the DENS does not perform equally well in different acoustic circumstances. In this chapter we present a method for estimating these two parameters by means of a backward integration method.

6.1 Evaluation of the impulse response

In the Dynamic Echo and Noise Suppressor (DENS) [18, 11], two acoustical room parameters are now fixed. These parameters are related to respectively the reverberation time $T_{60}$ and the Clarity Index $C$, which are well known acoustical parameters [20]. These two parameters can be extracted from the impulse response. Figure 6.1 shows the real measured impulse response, which is also depicted in Figure 1.5.

![Acoustic impulse response](image)

Figure 6.1: Acoustic impulse response.

A short period after the direct sound between the loudspeaker and the microphone, there
is generally a set of well defined and directional reflections that are directly related to the shape and size of the room. These are called the early reflections. After the early reflections, the rate of the arriving reflections increase greatly. These reflections are more random and difficult to relate to the physical characteristics of the room. This is called the diffuse reverberation or the late reflections, which decays exponentially in time.

The Reverberation time $T_{60}$ is defined as the time needed for the sound energy to decay to $1/1000$th (-60 dB) of the original energy. For the late reflections in the room, this decay in energy is exponential. For the early reflections, however, this is not true.

The Clarity Index $C$ is closely related to the early to late power ratio in the impulse response. The Clarity Index is defined as follows:

$$C = 10 \cdot \log_{10} \left( \frac{\text{total power in impulse response}}{\text{diffuse power in impulse response}} \right).$$

In order to make estimates of these two acoustical parameters, we make use of the impulse response of the room available in the adaptive filter. However, problems occur when the adaptive-filter has not converged properly to the Wiener solution; hence the estimate will be wrong. As this problem is inherent to the adaptive filter, we assume that the impulse response is correctly available (without errors).

Another problem is the length of the available impulse response. When the impulse response is very short, it is more difficult to make a good estimate of the reverberation time. As the DENS will be used especially for short adaptive filter lengths, we also want to make a good estimate (as good as possible) for short impulse responses.

The reverberation time of a specific impulse response can be estimated by calculating the average decay in energy in the impulse response. Therefore we can plot the squares of the individual samples of the impulse response $h[i]$ of Figure 1.5 on a logarithmic scale. This leads to the graph in Figure 6.2.

![Figure 6.2: Squared samples of the impulse response on log-scale.](image)

As we can see in this graph (on logarithmic scale), there is a linear decrease in average squared amplitude for increasing values of $i$. On absolute scale, this means that the energy
decays exponentially. However it is quite difficult to make an accurate estimation of this decay, as there are many random fluctuations visible in the graph. Therefore, many algorithms for estimating the reverberation time, are based upon the backward integration [27]. Here a single impulse response is integrated backwardly, starting with the tail. The result plot is often called the 'Energy Decay Curve (EDC)', which is free of random fluctuations.

\[ EDC[i] = \sum_{k=i}^{\infty} h^2[k]. \]  

(6.2)

In Figure 6.3, the Energy Decay Curve of the impulse response in Figure 6.1 is shown \(^1\). We can see that a linear regression line can be fitted in order to estimate the energy-decay. The reverberation time is calculated from the slope of this regression line.

![Energy Decay Curve](image)

Figure 6.3: Energy Decay Curve obtained via the backward integration method.

Available literature basically describe two problems. The first is the problem of background noise. When the impulse response is very long, the tail of the impulse response contains much more noise than actual decay component. This leads to errors in the estimation of the energy-decay, as at the end of the 'Energy Decay Curve' the decay is biased due to the noise power. Xiang [35] proposed a nonlinear iterative regression approach to deal with this background noise, but within this report, the problem of background noise is left out of consideration, as we mainly focus on short impulse responses.

The second problem is that of deformation, caused by the fact that impulse responses have finite lengths. Hence, for the first iterations of the backward integration method (i.e. the right part of the Energy Decay Curve), the decay seems to be very high. The problem of the deformation at the end of the Energy Decay Curve is avoided by omitting the last part of the Energy Decay Curve. However in cases of short impulse responses, we throw away information which can be used for a better estimate of the reverberation time. Therefore we estimate the rest energy \(E_{\text{comp}}\) [22], which eliminate the deformation effects. The estimation of \(E_{\text{comp}}\) is discussed in Section 6.2.2.

\(^1\)The integration starts with the 1024th input-sample of the impulse-response instead of the 00'th input-sample indicated by formula 6.2
When fitting a linear regression curve in the diffuse part of the Energy Decay Curve on log scale, the complexity would be quite low if we had the values available on log scale, but this is not the case. As we want to estimate the acoustical parameter on-the-fly and the implementation of the log-operator is very expensive, we use another method which requires less computations. This method is discussed in Section 6.2.

6.2 Proposed estimation algorithm

6.2.1 Kernel algorithm

The kernel algorithm proposed in this Section is first explained in a heuristical way. After this description we present the implementation of this algorithm.

The individual values of the Energy Decay Curve (EDC) are used in the kernel algorithm, where the $i$th value is indicated with $E[i]$. The ratio $\frac{E[i]}{E[i-1]}$ is approximately equal for values of $i$ in the diffuse part of the impulse response. This ratio is actually a multiplicative factor that indicates the energy decay between two values in the Energy Decay Curve. In this chapter this ratio is indicated with the symbol $\delta[i]$. More generally, the ratio $\frac{E[i]}{E[i-R]}$ is indicated with $\delta R[i]$ \(^2\). With all energy ratios $\delta R[i]$ in the diffuse part of the impulse response, we are capable of calculating the reverberation time, as indicated in Section 6.2.4. First we concentrate on calculating the energy ratios in an efficient manner.

When the energy ratio $\delta R[i]$ and the energy value $E[i-R]$ is known, we are able to make a prediction of $E[i-2R]$. The prediction can be used with the real energy value $E[i-2R]$ to define an error value. With the error value the energy ratio $\delta R[i]$ can be corrected in order to obtain an approximation of the energy ratio $\delta R[i - 1]$. This correction is performed by multiplying the error with a step-size parameter $\mu R$. By repeating this procedure, we are capable of estimating energy ratios $\delta R[i]$ between subsequent values in the Energy Decay Curve, without using expensive log- and division-operations. The complete kernel algorithm is depicted in Figure 6.4.

\[^2\text{As the energy decay curve is a monotonically decreasing function, the parameter } \delta R[i] \text{ will always be smaller than or equal to one.}\]

Figure 6.4: Kernel algorithm for the estimation of the reverberation time.
complete backward integration method and increases by one for each iteration. The index \( i \) is updated (decreased) during the iterations of the kernel algorithm. The backward integration method starts with \( i = \frac{N}{R} - 1 \), where \( \frac{N}{R} \) is an integer, and stops at \( i = 0 \). Parameter \( N \) is the length of the impulse response and \( R \) is a decimation factor. If \( i = 0 \) is reached, the index \( k \) is increased and the index \( i \) is again initialized with \( i = \frac{N}{R} - 1 \).

The backward integration method starts with the input of the squared values of the impulse response samples as input. These samples are integrated (see Figure 6.4). The result of the backward integration is indicated with \( E^R[i, k] \). The decimation of the output samples with the value \( R \) can be useful when we want to reduce the computational complexity of the algorithm. Especially if the algorithm is used for an on-the-fly estimation of the acoustical parameters, the computational complexity can be an issue.

We assume that there is no background noise and no deformation. The parameter \( \delta^R[i, k-1] \) in Figure 6.4 is the arithmetic mean of \( \delta^R[i, k-1] \) for \( i_{\text{late}} \leq i \leq \frac{N}{R} \), where \( i_{\text{late}} \) is the transition in the impulse response between the non-diffuse and the diffuse sound field.

The update for \( \delta^R[i, k] \) in Figure 6.4 is described by the following update equation:

\[
\delta^R[i-1, k] = \delta^R[i, k] + \mu^R[i, k] \cdot (E^R[i, k] - \delta^R[i, k] \cdot E^R[i-1, k]). \tag{6.3}
\]

The parameter \( \mu^R[i, k] \) is the step-size parameter. For a higher \( \mu^R[i, k] \), we get faster fluctuation of the parameter \( \delta^R[i, k] \).

Since the value of \( E^R[i, k] \) gets higher for decreasing \( i \), the adaptation-speed increases with decreasing \( i \). As we want to maintain a constant adaptation, we have to lower the step-size parameter each iteration \( i \) in a way that the adaptation-speed is normalized. This is done as follows:

\[
\mu^R[i-1, k] := \mu^R[i, k] \cdot \delta^R[k-1]. \tag{6.4}
\]

The mean energy ratio \( \delta^R[k-1] \) of the previous iteration \( k-1 \) of the backward integration is used to adjust the step-size parameter during iteration \( k \) of the backward integration. This factor \( \delta^R[k-1] \) is approximately the same factor as the multiplicative factor of which \( E^R[i, k] \) is growing every iteration of \( i \).

We can deduce a bound for the step-size in each iteration \( i \) of the algorithm, starting with formula 6.3:

\[
\delta^R[i-1, k] = \delta^R[i, k] + \mu^R[i, k] \cdot (E^R[i, k] - \delta^R[i, k] \cdot E^R[i-1, k]) = (1 - \mu^R[i, k] \cdot E^R[i-1, k]) \cdot \delta^R[i, k] + \mu^R[i, k] \cdot E^R[i, k].
\]

For a stable development of \( \delta^R[i, k] \) the term \((1 - \mu^R[i, k] \cdot E^R[i-1, k])\) in the previous formula must lie between the values -1 and 1. Hence, the bound to guarantee a stable development of \( \delta^R[i, k] \) is given by the formula:

\[
0 < \mu^R[i, k] < \frac{2}{E^R[i-1, k]}. \tag{6.5}
\]
Parameter $E^R[i - 1, k]$ is the energy integration value, increasing for decreasing $i$. Therefore we decrease the step-size $\mu^R[i, k]$ with decreasing value of $i$. We note that formula 6.5 which describes the stable region, is only valid in case of a diffuse sound field.

### 6.2.2 Choice of initial values

Each time a backward integration step $k$ is performed, the parameters $\mu^R[(N/R) - 1, k]$, $E^R[(N/R) - 1, k]$ and $\delta^R[(N/R) - 1, k]$ need to be initialized, as follows:

$$
\mu^R[(N/R) - 1, k] := c \cdot (E^R[i_{late}, k - 1])^{-1} \cdot (\delta^R[k - 1])^{-\left(\frac{N}{R} - i_{late}\right)}, \\
E^R[(N/R) - 1, k] := \hat{E}_{comp}^R[k - 1], \\
\delta^R[(N/R) - 1, k] := \bar{\delta}^R[k - 1],
$$

where $c$ is a constant for controlling the adaptation-rate. In order to determine these initial values, we must have $\delta^R[k - 1]$ available. This value has to be determined on basis of iteration $k - 1$ of the kernel algorithm.

Also we have to make a prediction $\hat{E}_{comp}^R[k]$ of $E_{comp}^R[k]$. This is the energy estimate of the tail defined by the total sum of samples:

$$
E_{comp}^R[k] = \sum_{i = N}^{\infty} \sum_{j = 0}^{R - 1} h^2[i \cdot R + j, k].
$$

This parameter $E_{comp}^R[k]$ can be estimated on basis of $\delta^R[k - 1]$ and the total power of the last $S$ samples in the impulse response (of iteration $k - 1$) in the following way:

$$
\hat{E}_{comp}^R[k] = \sum_{i = N - S}^{N - 1} h^2[i, k] \cdot \left(\delta^R[k - 1]\right)^{\frac{S}{R}} \approx E_{lastS}^R[k - 1] \cdot \left(\delta^R[k - 1]\right)^{\frac{S}{R}},
$$

where $\frac{S}{R}$ is an integer. We note that the last $S$ samples in the impulse response need to be part of the late (diffuse) reflections. Otherwise the estimation $\hat{E}_{comp}^R[k]$ is biased.

The constant $c$ in the above initialization values needs to be as large as possible, but always below the stability condition of formula 6.5. The larger the step-size $c$, the more the parameter $\delta^R[i, k]$ fluctuates, and the less iterations for the backward integrations are needed for convergence to good estimates for the acoustical parameters. In Section 6.2.5 we introduce an automatic step-size control, which adapts to a step-size $c$ that is as high as possible.
6.2.3 Estimation of the Clarity Index

The Clarity Index $C$ follows directly from $i_{late}$. The parameter $i_{late}$ is the value of $i$ where there is a transition between the non-diffuse and the diffuse part. The Clarity Index $C$ can be calculated as follows:

$$C = 10 \cdot \log_{10} \left( \frac{E^R[0, k]}{E^R[i_{late}, k]} \right).$$

(6.8)

The parameter $E^R[0, k]$ is simply the last value of $E^R[i, k]$ in the kernel algorithm and represents the total energy of the impulse response. The parameter $E^R[i_{late}, k]$ is the energy of the late diffuse sound field in the impulse response.

In order to determine $i_{late}$ and therefore also $E^R$, we need to use a threshold parameter, as mentioned in Section 6.1. This threshold is able to detect the transition between the non-diffuse and the diffuse part in the impulse response. We consider two thresholds $\gamma_{min}$ and $\gamma_{max}$ and detect $i_{late}$ as follows:

$$i_{late} = \left( \max_i \in [0, i_{range}] : (\delta^R[i, k] < \gamma_{min} \vee \delta^R[i, k] > \gamma_{max}) \right).$$

(6.9)

Calculation of $\gamma_{min}$

We consider the parameter $\delta^R[i, k]$ for $0 \leq i < \frac{N}{R}$ in three cases. In the first case (Figure 6.5), the initial parameter $E^R[(N/R) - 1, k]$ is chosen equal to the optimum value $E^R_{comp}[k]$.

![Figure 6.5: Case for good initial compensation energy.](image)

Hence there is no deformation visible in the running estimate of $\delta^R[i, k]$. We see that this estimate remains approximately constant until the non-diffuse part of the impulse response is reached. When the part of the impulse response is reached that only contains approximately zeros, the estimation curve tends to go smoothly from $\delta^R_{min2}[k]$ to 1.
To detect the non-diffuse and diffuse field transition, we introduce three variables:

\[
\begin{align*}
\delta^R_{\text{min1}}[k] &= \min\{\delta^R[i, k]\} \\
\delta^R_{\text{max1}}[k] &= \max\{\delta^R[i, k]\} \\
\delta^R_{\text{min2}}[k] &= \min\{\delta^R[i, k]\}
\end{align*}
\]

\(i_{\text{range}} < i \leq (N/R) - 1, \quad \text{(6.10)}\)

\(i_{\text{range}} < i \leq (N/R) - 1, \quad \text{(6.11)}\)

\(0 \leq i \leq i_{\text{range}}, \quad \text{(6.12)}\)

We define a threshold \(\gamma_{\text{min1}}\) with parameter \(\gamma_1\) to detect the transition between non-diffuse and diffuse field, where \(i_{\text{late}}\) is detected if \(\delta^R[i, k] < \gamma_{\text{min1}}\). In Figure 6.5, \(\gamma_1 = 4\).

\[
\gamma_{\text{min1}} = \delta^R_{\text{min1}}[k] - (\delta^R_{\text{max1}}[k] - \delta^R_{\text{min1}}[k]) \cdot \gamma_1.
\]

\(\text{(6.13)}\)

In the second case (Figure 6.6), we have chosen the initial parameter \(E^R[0, k]\) higher than the optimum value \(E^R_{\text{comp}}[k]\), while in the third case (Figure 6.7), we have chosen this initial parameter lower than the optimum value \(E^R_{\text{comp}}[k]\) (e.g. \(E^R[0, k] = 0\)).

---

Figure 6.6: Case for too high initial compensation energy.

Figure 6.7: Case without initial compensation energy.

When we consider the second case, we see that the variable \(\delta^R_{\text{max1}}\) is larger compared with the first case. In the third case, the variable \(\delta^R_{\text{min1}}\) is smaller compared with the first case.
When we apply formula 6.13, with \( \gamma_1 = 4 \) in the second and third case, we cannot detect \( i_{late} \). The solution for this problem is the introduction of a second threshold parameter \( \gamma_2 \) and detect \( i_{late} \) as soon as \( \delta^R[i,k] < \gamma_{min2} \):

\[
\gamma_{min2} = \delta_{min1}[k] - (\delta_{min1}[k] - \delta_{min2}[k - 1]) \cdot \gamma_2.
\]

(6.14)

We note that \( \delta_{min2}[k - 1] \), determined in the previous iteration of \( k \), is used. We assume that this value is approximately equal to \( \delta^R_{min2}[k] \).

In the second and third case (Figure 6.6 and 6.7) the \( i_{late} \) is determined with formula 6.14 as soon as \( \delta^R[i,k] < \gamma_{min2} \). For the second and third case, the threshold \( \gamma_2 = 0.5 \). The threshold \( \gamma_2 \) must be in the range \( 0 < \gamma_2 < 1 \). For the minimum threshold \( \gamma_{min} \) we use:

\[
\gamma_{min} = \max{\{\gamma_{min1}, \gamma_{min2}\}}.
\]

**Calculation of \( \gamma_{max} \)**

Until now we showed how the detection of \( i_{late} \) can be established in cases where \( \delta^R[i,k] \) gets below a certain threshold. There are however cases where the Early to Late ratio is very small and the values of \( \delta^R[i,k] \) are approximately the same for every value of \( i \). In these cases the parameter \( i_{late} \) is not detected with \( \gamma_1 \) and \( \gamma_2 \).

Therefore we introduce another threshold parameter \( \gamma_{max} \) and detect \( i_{late} \) as soon as \( \delta^R[i,k] > \gamma_{max} \), where:

\[
\gamma_{max} = \delta_{max1}[k] + (\delta_{max1}[k] - \delta_{min1}[k]) \cdot \gamma_3.
\]

(6.15)

We note that in Figures 6.5, 6.6 and 6.7 the variable \( \delta^R[i,k] \) does not get above \( \delta_{max1}[k] \), as the step-size \( c \) is chosen very low. When the step-size is maximized with the step-size control (see Section 6.2.5), the variable \( \delta^R[i,k] \) increases rapidly from \( i_{late} \) down to 0 due to the zeros in the impulse response. Hence the detection by means of \( \gamma_{max} \) can be performed.

**6.2.4 Estimation of the reverberation time**

The reverberation time \( T_{60} \) is directly related with the mean energy ratio \( \bar{\delta}^R[k] \) by the following equation:

\[
T_{60} = \frac{-6R}{F_s \cdot \log_{10}(\bar{\delta}^R[k])},
\]

(6.16)

where \( F_s \) is the sampling frequency and \( R \) is the decimation factor.

When \( i_{late} \) is detected (See Section 6.2.3), the range \( i_{late} \leq i < \frac{N}{R} \) is used to determine \( \bar{\delta}^R[k] \). The most practical way is to calculate the arithmetic mean of the values \( \delta^R[i,k] \) over this range:

\[
\bar{\delta}^R[k] = \left( \frac{N}{R} - i_{late} \right)^{-1} \sum_{i=i_{late}}^{N-1} \delta^R[i,k].
\]

(6.17)
We note that with the calculation of this arithmetic mean, the data in the energy decay curve is not fitted in a least-squares sense, as the lowest values in the energy decay curve are equally weighted compared to the high valued. When we want to minimize the sum of squares, a weighting must be applied, with higher weighting for the higher values. As in the energy decay curve the high values might be part of early reflections, we do not want to impose high weights to these values. On the other hand, when the energy decay curve contains noise, the estimate of the decay is biased. However we assume that the impulse response does not contain a significant noise-part and apply no weighting.

6.2.5 Automatic step-size control

When using a fixed step-size parameter \( c \) in the estimation algorithm, the parameter \( c \) needs to be chosen very small, in order to guarantee that the algorithm works robustly for different impulse responses. However, a small value for \( c \) leads to a slow convergence speed to the steady state estimates. Therefore we introduce a method for automatically adapting the step-size parameter.

During the kernel algorithm of Section 6.2.1 the estimate of \( \delta^R[i, k] \) sometimes exceeds the value of 1. As a value higher than 1 clearly does not represent an energy-decay, we need to limit the parameter \( \delta^R[i, k] \) to 1. Therefore we check if \( \delta^R[i, k] > 1 \). If this happens, we reduce the value of parameter \( \mu^R[i, k] \). The reduction should be based upon the degree of excess of parameter \( \delta^R[i, k] \) with respect to unity. We could use the following reduction of the parameter \( \mu \):

\[
\mu^R[i, k] := \mu^R[i, k] \cdot \frac{1}{\max\{1, \delta^R[i, k]\}}. \tag{6.18}
\]

When we assume that parameter \( \delta^R[i, k] \) exceeds the value of 1 only slightly and stays well below the value of 2, we can also use formula 6.19. This trades an expensive division operation off against a less expensive multiplication and addition.

\[
\mu^R[i, k] := \mu^R[i, k] \cdot \max\{1, 2 - \delta^R[i, k]\}. \tag{6.19}
\]

Note that we do not change \( c \) directly during the kernel algorithm. Instead we only alter the value of \( c \) at the end of the algorithm with the following update:

\[
c := c \cdot (2 - \delta^R_{\text{min}}[k]). \tag{6.20}
\]

With this method of automatic step-size control, the running estimate of \( \delta^R[i, k] \) maximizes the deviation between the values 0 and 1. This is done with relatively low computational complexity, as for most of the iterations \( i \) in the kernel algorithm, \( \delta^R[i, k] \) stays below the value of 1.
6.3 Computational complexity

We already mentioned that the proposed algorithm does not use the log-operator which is expensive in terms of multiplications and additions. Instead it computes a running estimate $\delta^R[i, k]$ and does this in an adaptive manner.

The computational complexity of the proposed algorithm is scalable via the parameter $R$. The higher the value $R$, the lower the computational complexity. The estimate for the reverberation time $T_{60}$ works fine for higher values of $R$. However for higher values of $R$, the detection for the non-diffuse sound field is performed at a lower resolution, leading to less accurate estimates for the Clarity Index $C$. The choice for the parameter $R$ is a trade-off between the computational complexity and the accuracy of the Clarity Index $C$.

There is also a possibility to perform the scaling of the computational complexity during the iterations $(k)$ of the algorithm. This could be useful in cases where the computational load available for the algorithm is fluctuating.

The largest part of the computational complexity is located in the kernel algorithm. When considering multiplications and additions, the computational complexity of a single iteration $k$ of the backward integration is:

$$\Psi_{\text{ESTIM}} = M\{N + 3 \cdot R^{-1} \cdot N\} + A\{N + 2 \cdot R^{-1} \cdot N\}$$

$$= 2N + 5 \cdot R^{-1} \cdot N,$$  \hspace{1cm} (6.21)

where $R$ is the decimation factor and $N$ is the length of the impulse response. We note that the kernel algorithm uses also minimum and maximum operators and a step-size control. Furthermore the estimation algorithm uses additional computational complexity in order to calculate the initial parameters and the estimates of the acoustical parameters. However for large values of $N$ and small values of $R$, this additional computational complexity is small compared with formula 6.21.

6.4 Simulations

We first implemented the estimation algorithm in Matlab. The Matlab program is very helpful for checking how the algorithm behaves when using particular impulse responses. Furthermore we implemented the algorithm in the Acoustic Performance and Enhancement System (APES) environment, within an Acoustic Echo Canceller similar to Figure 1.2.

We found that after double-talk situations, the high-frequency components of the impulse response in the adaptive filter are damped very slowly. These high-frequency components lead to wrong estimates for the acoustical parameters. Therefore we filtered the impulse response with a bandpass pre-filter before using the impulse response for the backward integration. We used a 3rd order bandpass filter with a passband from 200 Hz to 3600 Hz (at 8 kHz sample frequency).
The following simulations are performed with the Matlab implementation, where the estimation is based upon 1 specific impulse response. As many parameters can be adjusted, the simulations are performed for one specific set of parameters. These parameters are listed in Table 6.4.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Impulse response length</td>
<td>1024 @ 8 KHz</td>
</tr>
<tr>
<td>$S$</td>
<td>Compensation Energy block length</td>
<td>128</td>
</tr>
<tr>
<td>$R$</td>
<td>Decimation Factor</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma_1$</td>
<td>Threshold 1</td>
<td>4</td>
</tr>
<tr>
<td>$\gamma_2$</td>
<td>Threshold 2</td>
<td>0.5</td>
</tr>
<tr>
<td>$\gamma_3$</td>
<td>Threshold 3</td>
<td>0.1</td>
</tr>
<tr>
<td>niter</td>
<td>Number of iterations</td>
<td>5</td>
</tr>
</tbody>
</table>

We have performed the simulations for three different impulse responses (measured in one specific chamber) for three distances between the loudspeaker and microphone (i.e. 1, 3 and 5 metres). These impulse responses do not contain any significant background noise. We present the Energy Decay Curve (EDC) (without tail-energy compensation) in each of the three situations and list the Reverberation Time $T_{60}$, Clarity Index $C$ and $i_{late}$. These three values are also depicted in the EDC by means of dashed lines.

**Distance: 1 metre**

![Energy Decay Curve](image)

The results of the estimation algorithm are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$T_{60}$ (ms)</th>
<th>$C$ (dB)</th>
<th>$i_{late}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>316.1</td>
<td>8.349</td>
<td>42</td>
</tr>
</tbody>
</table>
6.4. SIMULATIONS

Distance: 3 metres

The results of the estimation algorithm are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$T_{60}$ (ms)</th>
<th>$C$ (dB)</th>
<th>$i_{late}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>393.3</td>
<td>6.053</td>
<td>112</td>
</tr>
</tbody>
</table>

Distance: 5 metres

The results of the estimation algorithm are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$T_{60}$ (ms)</th>
<th>$C$ (dB)</th>
<th>$i_{late}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>363.9</td>
<td>2.907</td>
<td>133</td>
</tr>
</tbody>
</table>

The estimates for these situations are fairly accurate. The estimated reverberation time $T_{60}$ is indicated by the linear regression line, which fits the diffuse part of the impulse response accurately. The transition between non-diffuse and diffuse energy, indicated by the vertical dashed line, is detected on the moment the decay in energy is much too high compared with the exponential energy decay of the diffuse field.
Chapter 7

Conclusions

Constraint approximations for the Partitioned BFDAF that can be calculated efficiently, do not result in a near-optimal convergence behaviour compared with the fully constrained PBFDAB. This is caused by partly unconstrained circular wrap-around artifacts, which are accumulated in the weight-update and cause problems in the convolution part of the PBFDAB, as the different partitions are coupled. When combining the constraint approximations with the Alternative constraining mechanism, an improvement is obtained, but still achieve a non-optimal convergence behaviour. The parts of the circular wrap-around artifacts, not removed by the constraint approximations, is removed once in a while with the Alternative constraining mechanism. Some of these removed parts are approximately results of a linear correlation, that can be successfully re-used (compensated) in the neighbour partitions. When we combine this method of compensation with the simplest form of constraint approximations (i.e. sinusoidal constraint), we obtain a near-optimal convergence behaviour compared to the fully constrained PBFDAB, while obtaining a substantial reduction in computational complexity. Simulation results show that the performance of this constraining mechanism is near-optimal for highly correlated, non-stationary input signals.

For acoustic applications, the Dynamic Echo and Noise Canceller (DENS) can be used in combination with the adaptive filter. This post-processing device suppresses residual echoes from the adaptive filter and can be used to reduce the filter-length of the adaptive filter, resulting in a reduction in computational complexity and memory requirements. In this report we present a method for estimating two acoustical parameters from the room impulse response, modelled by the adaptive filter. These acoustical parameters are used in the DENS. Due to this parameter estimation, the performance of the DENS is good in different situations where the acoustical parameters can vary. The estimation method presented in this report is based on the well known backward integration method. As the estimation method has a low computational complexity, the estimation of the parameters can be updated on-the-fly during the adaptation process of the adaptive filter. The parameter estimation, however, fails when the impulse response is polluted with background noise or when double-talk in the adaptive filter occurs. When using the estimation algorithm for the Echo and Noise Suppressor, this issue needs further investigation.
Bibliography


[34] H.V.Sorensen, C.Sidney Burrus Efficient Computation of the DFT with Only a Subset of Input or Output Points (1993)


## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEC</td>
<td>Acoustic Echo Canceller</td>
</tr>
<tr>
<td>AF</td>
<td>Adaptive Filter</td>
</tr>
<tr>
<td>APES</td>
<td>Acoustic Performance Enhancement System</td>
</tr>
<tr>
<td>BFDAF</td>
<td>Block Frequency Domain AF</td>
</tr>
<tr>
<td>BNLMS</td>
<td>Block Normalized Least Mean Square</td>
</tr>
<tr>
<td>CTC</td>
<td>Convergence Time Constant</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>DENS</td>
<td>Dynamic Echo and Noise Suppressor</td>
</tr>
<tr>
<td>EDC</td>
<td>Energy Decay Curve</td>
</tr>
<tr>
<td>PBFDAF</td>
<td>Partitioned BFDAF</td>
</tr>
<tr>
<td>FBNLMS</td>
<td>FD Block Normalized Least Mean Square</td>
</tr>
<tr>
<td>FD</td>
<td>Frequency Domain</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>HOL</td>
<td>Half Overlap</td>
</tr>
<tr>
<td>IDCT</td>
<td>Inverse Discrete Cosine Transform</td>
</tr>
<tr>
<td>IFFT</td>
<td>Inverse Fast Fourier Transform</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>NMSE</td>
<td>Normalized Mean Squared Error</td>
</tr>
<tr>
<td>OLA</td>
<td>Overlap and Add</td>
</tr>
<tr>
<td>OLS</td>
<td>Overlap and Save</td>
</tr>
</tbody>
</table>
Notation

\( x \) \hspace{1cm} \text{Signals (Lower case character).}

\( \underline{x} \) \hspace{1cm} \text{Time domain vector (Lower case underlined).}

\( \underline{X} \) \hspace{1cm} \text{Frequency domain vector (Upper case underlined).}

\( \underline{x}^{2N} \) \hspace{1cm} \text{Vector of length } 2N.

\( \mathcal{F}, \mathbf{I}^N \) \hspace{1cm} \text{Matrices (Bold upper case or calligraphic).}

\( w_i \) \hspace{1cm} \text{A subscript } i \text{ denotes the } i^{\text{th}} \text{ version.}

\( (w)_i \) \hspace{1cm} \text{The } i^{\text{th}} \text{ element of } w.

\( [k] \) \hspace{1cm} \text{Denotes the time index.}

\( (\underline{x})' \) \hspace{1cm} \text{Transpose of } \underline{x}.

\( (x)^* \) \hspace{1cm} \text{Complex conjugate of } x.

\( \otimes \) \hspace{1cm} \text{Elementwise multiplication of two vectors.}

\( \mathbf{diag}\{\underline{x}^{2N}\} \) \hspace{1cm} 2N \times 2N \text{ matrix with on its main diagonal } \underline{x}^{2N}.

\( \max\{a, b\} \) \hspace{1cm} \text{Maximum of } a \text{ and } b.

\( \min\{a, b\} \) \hspace{1cm} \text{Minimum of } a \text{ and } b.

\( \mathcal{E}\{x[k]\} \) \hspace{1cm} \text{Ensemble average of } x[k].
Symbols

Symbols in Chapters 1 to 5

\(a\) \hspace{1cm} \text{Mean value for constraint approximations.}
\(2\alpha\) \hspace{1cm} \text{Adaptation constant.}
\(A_i\) \hspace{1cm} \text{Update block for partition } i.
\(\mathcal{A}\{\cdot\}\) \hspace{1cm} \text{Number of additions.}
\(C_{b_i}[\kappa N]\) \hspace{1cm} \text{Backward compensation signal from partition } i.
\(C_{f_i}[\kappa N]\) \hspace{1cm} \text{Forward compensation signal from partition } i.
\(\mathbf{D}^2_N\) \hspace{1cm} \text{Rotation matrix.}
\(\mathcal{D}\{\cdot\}\) \hspace{1cm} \text{Number of divisions.}
\(e[k]\) \hspace{1cm} \text{Desired signal (echo in AEC).}
\(\hat{e}[k]\) \hspace{1cm} \text{Estimate of the desired signal.}
\(\tilde{e}[k]\) \hspace{1cm} \text{Desired signal corrupted with } s[k].
\(\mathcal{F}\) \hspace{1cm} \text{Fourier transform matrix.}
\(\mathcal{G}^2_N\) \hspace{1cm} \text{Generalised constraint window.}
\(\mathcal{V}[k]\) \hspace{1cm} \text{Stochastic gradient vector.}
\(h[k]\) \hspace{1cm} \text{Impulse response vector.}
\(\mathbf{I}^N\) \hspace{1cm} \text{Identity matrix.}
\(j\) \hspace{1cm} \text{Imaginary unit.}
\(\mathbf{J}^N\) \hspace{1cm} \text{Mirror matrix.}
\(K\) \hspace{1cm} \text{Number of partitions in the PBFDAF algorithm.}
\(K_i\) \hspace{1cm} \text{Threshold partition for Hybrid constrained PBFDAF.}
\(k\) \hspace{1cm} \text{Time index.}
\(\kappa\) \hspace{1cm} \text{Block time index.}
\(m\) \hspace{1cm} \text{Multiplicative factor for high-slope constraint approximation.}
\(\mathcal{M}\{\cdot\}\) \hspace{1cm} \text{Number of real multiplications.}
\(\Theta\) \hspace{1cm} \text{Memory occupation.}
\(N\) \hspace{1cm} \text{Block length.}
\(\mathbf{0}, \mathbf{O}\) \hspace{1cm} \text{All zero vector, matrix.}
\(\mathcal{P}\) \hspace{1cm} \text{Alternating constraining period.}
\(\mathcal{P}_x[k]\) \hspace{1cm} \text{Normalisation (power) vector.}
\(\hat{\mathcal{P}}_x[k]\) \hspace{1cm} \text{Estimate of } \mathcal{P}_x.
\(\Psi\) \hspace{1cm} \text{Computational complexity measure.}
\(r[k]\) \hspace{1cm} \text{Residual signal.}

87
\( s[k] \) Near-end signal.

\( S \) Two state switch function.

\( S\{\cdot\} \) Number of binary shift operations.

\( \sigma^2_x[k] \) Variance of \( x[k] \).

\( w[k] \) Adaptive filter vector.

\( x[k] \) Input signal (Far-end signal).

\( y[k] \) Normalised residual signal vector.

\( \tilde{z}[k] \) Weight update vector (without the constraint).

\( -1 \) Vector with 1 on even and -1 on odd elements.

### Symbols in Chapter 6

\( C \) Clarity index.

\( c \) Parameter for the estimation algorithm.

\( \delta^R[i, k] \) Running estimate of \( \delta^R \) in the kernel algorithm.

\( \delta^R_{\text{max}}[k] \) Maximum in region \( i_{\text{range}} < i \leq (N/R) - 1 \).

\( \delta^R_{\text{min}}[k] \) Minimum in region \( i_{\text{range}} < i \leq (N/R) - 1 \).

\( \delta^R_{\text{max}}[k] \) Maximum in region \( 0 \leq i \leq i_{\text{range}} \).

\( \delta^R[k] \) Estimate of \( \delta^R \) in 1 iteration of the estimation algorithm.

\( E^R[i, k] \) Energy of accumulated samples of the reverse integration method.

\( E^R_{\text{comp}}[k] \) Compensation energy for the remaining tail of the impulse response.

\( E^R_{\text{last}}[k] \) Energy of the last \( S \) samples in the impulse response.

\( F_s \) Sample frequency.

\( \gamma_1 \) Relative threshold parameter 1.

\( \gamma_2 \) Relative threshold parameter 2.

\( \gamma_3 \) Relative threshold parameter 2.

\( \gamma_{\text{min}} \) Absolute threshold parameter \((\text{max}(\gamma_{\text{min}1}, \gamma_{\text{min}2}))\).

\( \gamma_{\text{max}} \) Absolute threshold parameter.

\( h[k] \) Sample of the impulse response.

\( i \) Index for 1 iteration in the kernel algorithm.

\( i_{\text{late}} \) Index of the transition between diffuse/non-diffuse field.

\( i_{\text{range}} \) Index indicating the evaluation range of the minimum/maximum operators.

\( k \) Index for 1 iteration of the estimation algorithm.

\( N \) Adaptive filter-length.

\( \Psi_{\text{ESTIM}} \) Computational complexity measure.

\( R \) Decimation factor.

\( S \) Number of samples of the tail in the impulse response.

\( T \) Delay for 1 value.

\( T_{60} \) Reverberation time where the energy has dropped 60 dB.

\( \mu^R[i, k] \) Step-size parameter for the kernel algorithm.
Figures

**Mirror vector**
\[ x^N[\kappa N] \xrightarrow{J^N} y^N[\kappa N] \]
\[ y^N[\kappa N] = J^N \cdot x^N[\kappa N]. \]

**Elementwise addition**
\[ x^{2N}[\kappa N] + z^{2N}[\kappa N] \]
\[ y^{2N}[\kappa N] = y^{2N}[\kappa N] - x^{2N}[\kappa N]. \]

**Elementwise multiplication**
\[ x^{2N}[\kappa N] \times z^{2N}[\kappa N] \]
\[ y^{2N}[\kappa N] = x^{2N}[\kappa N] \otimes y^{2N}[\kappa N]. \]

**Scale vector**
\[ x^{2N}[\kappa N] \]
\[ y^{2N}[\kappa N] = \alpha \cdot x^{2N}[\kappa N]. \]

**Complex conjugate**
\[ x^{2N}[\kappa N] \]
\[ y^{2N}[\kappa N] = (x^{2N}[\kappa N])^*. \]

**Block delay**
\[ x^{2N}[\kappa N] \]
\[ y^{2N}[\kappa N] = x^{2N}[(\kappa - 1)N]. \]
**FFT**

\[ x^{2N}[\kappa N] \xrightarrow{\mathcal{F}} X^{2N}[\kappa N] \]

\[ X^{2N}[\kappa N] = \mathcal{F} \cdot x^{2N}[\kappa N]. \]

**Inverse FFT**

\[ X^{2N}[\kappa N] \xrightarrow{\mathcal{F}^{-1}} x^{2N}[\kappa N] \]

\[ x^{2N}[\kappa N] = \mathcal{F}^{-1} \cdot X^{2N}[\kappa N]. \]

**Half Overlap**

\[ x^{N}[\kappa N] \xrightarrow{\text{50\% overlap}} x^{2N}[\kappa N] \]

\[ y^{2N}[\kappa N] = \left( \begin{array}{c} x^{N}[(\kappa - 1)N] \\ x^{N}[\kappa N] \end{array} \right). \]

**Circular shift**

\[ x^{2N}[\kappa N] \xrightarrow{\mathcal{N}} x^{2N}[\kappa N] \]

\[ y^{2N}[\kappa N] = D_{N}^{2N} \cdot x^{2N}[\kappa N]. \]

**Serial-Parallel converter**

\[ x[k] \xrightarrow{S} y^{N}[\kappa N] \]

\[ y^{N}[\kappa N] = \left( \begin{array}{c} x[\kappa N - N + 1] \\ \vdots \\ x[\kappa N] \end{array} \right). \]

**Parallel/Serial converter**

\[ x^{N}[\kappa N] \xrightarrow{P S} y[k] \]

\[ y[k\in \{ \kappa N - N + 1, \ldots, \kappa N \}] = (x^{N}[\kappa N])_{i}, \ 0 \leq i < N. \]

**1-to-2 switch**

\[ Z^{2N}[\kappa N] \xrightarrow{S_{i}} \left( \begin{array}{c} Z^{2N}[\kappa N] \\ 0^{2N} \end{array} \right) \text{ for } S_{i} = 0 \]

\[ Z^{2N}[\kappa N] \xrightarrow{S_{i}} \left( \begin{array}{c} 0^{2N} \\ Z^{2N}[\kappa N] \end{array} \right) \text{ for } S_{i} = 1. \]

**2-to-1 switch**

\[ X^{2N}[\kappa N] \xrightarrow{S_{i}} Z^{2N}[\kappa N] \]

\[ Z^{2N}[\kappa N] = \left( \begin{array}{c} X^{2N}[\kappa N] \\ Z^{2N}[\kappa N] \end{array} \right) \text{ for } S_{i} = 0 \]

\[ Z^{2N}[\kappa N] = \left( \begin{array}{c} Z^{2N}[\kappa N] \\ X^{2N}[\kappa N] \end{array} \right) \text{ for } S_{i} = 1. \]

**Append zeros on the right**

\[ x^{N}[\kappa N] \xrightarrow{0} y^{2N}[\kappa N] \]

\[ y^{2N}[\kappa N] = \left( \begin{array}{c} I^{N} \\ O_{N} \end{array} \right) \cdot x^{N}[\kappa N]. \]
Figures

Append zeros on the left
\[ x^N[kN] \rightarrow 0 \rightarrow y^{2N}[kN] \]
\[ y^{2N}[kN] = \begin{pmatrix} 0^N \\ 1^N \end{pmatrix} \cdot x^N[kN]. \]

Nulling right part
\[ x^{2N}[kN] \rightarrow y^{2N}[kN] \]
\[ y^{2N}[kN] = \begin{pmatrix} 1^N \\ 0^N \end{pmatrix} \cdot x^{2N}[kN]. \]

Disposal of elements
\[ x^{2N}[kN] \rightarrow y^{2N}[kN] \]
\[ y^N[kN] = \begin{pmatrix} 1^N \\ 0^N \end{pmatrix} \cdot x^{2N}[kN]. \]

Raised Cosine window
\[ x^{2N}[kN] \rightarrow y^{2N}[kN] \]
\[ y^{2N}[kN] = \frac{1}{2N} (X^{2N}[kN] \oplus G^{2N}), \]
where
\[ (G^{2N})_i = \begin{cases} 
2N \cdot \frac{1}{2} & \text{for } i = 0 \\
2N \cdot \frac{1}{4} & \text{for } i = 1, 2N - 1 \\
0 & \text{elsewhere.}
\end{cases} \]

Rectangular constraint
\[ x^{2N}[kN] \rightarrow y^{2N}[kN] \]
\[ y^{2N}[kN] = \frac{1}{2N} (X^{2N} \oplus G^{2N}), \]
where
\[ (G^{2N})_i = \begin{cases} 
2N \cdot \frac{1}{2} & \text{for } i = 0 \\
0 & \text{for } 0 < i < 2N, i \text{ even} \\
1 + \frac{-2N \cdot j}{2N \cdot \tan(\frac{\pi}{2N})} & \text{for } 0 < i < 2N, i \text{ odd.}
\end{cases} \]

High-slope window
\[ x^{2N}[kN] \rightarrow y^{2N}[kN] \]
\[ y^{2N}[kN] = \frac{1}{2N} (X^{2N}[kN] \oplus G^{2N}), \]
where
\[ (G^{2N})_i = \begin{cases} 
2N \cdot a & \text{for } i = 0 \\
0 & \text{for } 0 < i < 2N, i \text{ even} \\
-\frac{2N \cdot a}{2} \left( m^{-(\frac{1}{2})} - m^{-(\frac{2N-1}{2})} \right) j & \text{for } 0 \leq i < 2N, i \text{ odd.}
\end{cases} \]

Sinusoid window
\[ x^{2N}[kN] \rightarrow y^{2N}[kN] \]
\[ y^{2N}[kN] = \frac{1}{2N} (X^{2N}[kN] \oplus G^{2N}), \]
where
\[ (G^{2N})_i = \begin{cases} 
2N \cdot \frac{1}{2} & \text{for } i = 0 \\
2N \cdot \frac{1}{4} & \text{for } i = 1 \\
2N \cdot \frac{1}{4} j & \text{for } i = 2N - 1 \\
0 & \text{elsewhere.}
\end{cases} \]
Appendix A

Computational complexity of several parts

A.1 (I)FFTs

The FFT and the IFFT can be calculated in several ways. A specific implementation with a low number of additions/subtractions and multiplications is the split-radix (I)FFT. The following table shows the number of additions/subtractions and multiplications needed in the split-radix FFT as implemented in APES. Furthermore we assume that for an IFFT the computational complexity is the same.

<table>
<thead>
<tr>
<th>$2N$</th>
<th>adds/subs $A$</th>
<th>muls $M$</th>
<th>$\Psi_{FFT}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>950</td>
<td>328</td>
<td>1278</td>
</tr>
<tr>
<td>256</td>
<td>2240</td>
<td>822</td>
<td>3062</td>
</tr>
<tr>
<td>512</td>
<td>5158</td>
<td>1984</td>
<td>7142</td>
</tr>
<tr>
<td>1024</td>
<td>11680</td>
<td>4646</td>
<td>16326</td>
</tr>
<tr>
<td>2048</td>
<td>26086</td>
<td>10656</td>
<td>36742</td>
</tr>
</tbody>
</table>

Table A.1: Computational complexity for split-radix FFT

We note that this implementation of the split-radix FFT uses 3 multiplications and 3 additions for a complex multiplication with a fixed complex value. Hence additions and multiplications can be traded.

---

1 Acoustic Performance and Enhancement System (APES) is a library of signal processing routines used at Philips Research.
A.2 Elementwise multiplication

As we use only real valued time-vectors, the FFT transformed complex vector is symmetrical. Hence the 0th and the \(N\)th element of a length \(2N\) vector are real, and for all \(0 < i < N\) the vector contains complex conjugates. As the result after elementwise multiplication is also symmetrical, we only need to calculate (and store) half of the elements. The number of additions/subtractions and multiplications are given in Table 5.1.

<table>
<thead>
<tr>
<th>(2N)</th>
<th>adds/subs (A)</th>
<th>muls (M)</th>
<th>(\Psi_5{2N})</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>254</td>
<td>128</td>
<td>382</td>
</tr>
<tr>
<td>256</td>
<td>510</td>
<td>256</td>
<td>766</td>
</tr>
<tr>
<td>512</td>
<td>1022</td>
<td>512</td>
<td>1534</td>
</tr>
<tr>
<td>1024</td>
<td>2046</td>
<td>1024</td>
<td>3070</td>
</tr>
<tr>
<td>2048</td>
<td>4094</td>
<td>2048</td>
<td>6142</td>
</tr>
</tbody>
</table>

Table A.2: Computational complexity for the complex elementwise multiplication

We have implemented 1 complex multiplication by 4 real multiplications and 2 real additions.

A.3 PBFDAF power normalization

For estimating the inverse power vector, we first calculate the input power in every frequency-bin of the Fourier transformed input signal. Furthermore, we use an exponential averaging network for smoothing the power estimation in every frequency-bin. Finally, we also have to perform the power normalization in the PBFDAF algorithm, where the inverse power has to be multiplied by a complex valued vector. The complete implementation of the PBFDAF power normalization in APES needs a numbers of additions/subtractions, multiplications and divisions as indicated in Table A.1.

<table>
<thead>
<tr>
<th>(2N)</th>
<th>adds/subs (A)</th>
<th>muls (M)</th>
<th>divs (D)</th>
<th>(\Psi_5{2N})</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>325</td>
<td>388</td>
<td>66</td>
<td>779</td>
</tr>
<tr>
<td>256</td>
<td>645</td>
<td>772</td>
<td>130</td>
<td>1547</td>
</tr>
<tr>
<td>512</td>
<td>1285</td>
<td>1540</td>
<td>258</td>
<td>3083</td>
</tr>
<tr>
<td>1024</td>
<td>2565</td>
<td>3076</td>
<td>514</td>
<td>6155</td>
</tr>
<tr>
<td>2048</td>
<td>5125</td>
<td>6148</td>
<td>1026</td>
<td>12299</td>
</tr>
</tbody>
</table>

Table A.3: Computational complexity for the PBFDAF power normalization
Appendix B

PBFDAF memory requirements

The overall memory requirements of the PBFDAF algorithm can be divided into several parts. These parts are listed in Table B.1. We note that these memory requirements are only valid for the Half Overlap PBFDAF, with the Overlap-Save as efficient convolution method.

<table>
<thead>
<tr>
<th>Vector name</th>
<th>Number</th>
<th>Real/Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input delay-line</td>
<td>$K \cdot 2N$</td>
<td>complex</td>
</tr>
<tr>
<td>Adaptive weights</td>
<td>$K \cdot 2N$</td>
<td>complex</td>
</tr>
<tr>
<td>Far-end signal vector</td>
<td>$N$</td>
<td>real</td>
</tr>
<tr>
<td>Near-end signal vector</td>
<td>$N$</td>
<td>real</td>
</tr>
<tr>
<td>Residual vector</td>
<td>$N$</td>
<td>real</td>
</tr>
<tr>
<td>Half Overlap delay</td>
<td>$N$</td>
<td>real</td>
</tr>
<tr>
<td>Power-normalization vector</td>
<td>$N$</td>
<td>real</td>
</tr>
<tr>
<td>Intermediate-result vector</td>
<td>$2N$</td>
<td>complex</td>
</tr>
</tbody>
</table>

Table B.1: Different parts of the overall PBFDAF memory requirements

As we use real-valued time-domain vectors, all frequency-domain vectors are Hermitian. When exploiting the symmetry of the Hermitian vector, a $2N$ length complex vector can be stored with only $2N$ real values. The total memory occupation $\Theta$ can now be expressed as:

$$\Theta_{PBFDAF} = N K \cdot \frac{4K + 7}{K},$$  \hspace{1cm} (B.1)

where $N$ is the block-length and $K$ denotes the number of partitions in the PBFDAF algorithm.
Appendix C

DENS parameters

The DENS needs the acoustical parameter $\alpha_r$ (related to the reverberation time $T_{60}$) and the parameter $Y_{comp-dir}$ (related to the Clarity Index $C$).

The parameter $\alpha_r$ is related to the reverberation time by the following formula [18]:

$$\alpha_r = 10^{-\frac{6}{T_{60}}}, \quad (C.1)$$

where $B$ is the DENS block-length, $F_s$ is the sampling-frequency and $T_{60}$ is the reverberation time. We note that the DENS uses the parameter $\alpha'_r = \sqrt{\alpha_r}$, which is based on magnitudes instead of powers.

For calculating $Y_{comp-dir}$ we only need to estimate the diffuse energy in the late reflections, indicated with $E_{late}$ and the non-diffuse energy of the early reflections, indicated with $E_{early}$.

In the report of Janse [18], $Y_{comp}$ is defined as follows:

$$Y_{comp} = 1 - R_{y,diff} = 1 - \frac{\sum_{i=N-B}^{N-1} h[i]^2}{\sum_{i=0}^{N-1} h[i]^2} = \frac{\sum_{i=0}^{N-B} h[i]^2}{\sum_{i=0}^{N-1} h[i]^2}, \quad (C.2)$$

where $N$ is the adaptive-filter length, $B$ is the DENS block-length ($B < N$) and $h[i]$ are the coefficients of the impulse response modelled by the adaptive filter. Alternatively $R_{y,diff}$ can be estimated as [18]:

$$R_{y,diff} = \frac{\sum_{i=N-B}^{N-1} h[i]^2}{\sum_{i=0}^{N-1} h[i]^2} = \frac{\alpha_r^{N-1}}{\alpha_r^{N-B}} = \frac{\alpha_r^{N-1} (1 - \alpha_r)}{1 - \alpha_r^{N-B}}, \quad (C.3)$$

\footnote{We assume that the direct sound field between the loudspeaker and microphone is also part of the early reflections.}
in which $\alpha_r$ is defined as given in Equation C.1.

In the report of Janse [18] the formulas C.2 and C.3 are based upon powers. However the DENS is based upon magnitudes instead of powers. Therefore we indicate all parameters based on magnitudes by a prime (e.g. $\alpha'_r$ or $\gamma'_\text{comp}$).

The parameter $\gamma_\text{comp}$ is equal to the parameter $\gamma'_\text{comp-dir}$ in case there is no non-diffuse part in the Energy Decay Curve. Usually this is not a valid assumption [18] and we use $\gamma'_\text{comp-dir}$ instead of $\gamma_\text{comp}$, because $\gamma'_\text{comp-dir}$ also takes the non-diffuse part into account.

In order to calculate the parameter $\gamma'_\text{comp-dir}$ we use the Clarity Index [12], which is defined as:

$$ C = 10 \log_{10} \left( \frac{E_{\text{total}}}{E_{\text{late}}} \right) = 10 \log_{10} \left( \frac{E_{\text{early}} + E_{\text{late}}}{E_{\text{late}}} \right) = 10 \log_{10} \left( 1 + \frac{E_{\text{early}}}{E_{\text{late}}} \right). \quad (C.4) $$

The relation between $R'_y,\text{diff}$ (only a diffuse field) and $R'_y$ (both diffuse and non-diffuse field) is given by [11]:

$$ R'_y = R'_y,\text{diff} \cdot 10^{-\frac{C}{10}}. \quad (C.5) $$

Now we can deduce the formula for the calculation of $\gamma'_\text{comp-dir}$:

$$ \gamma'_\text{comp-dir} = 1 - \left( 10^{-\frac{1}{2} \log_{10}(1 + \frac{E_{\text{early}}}{E_{\text{late}}})} \cdot R_y,\text{diff} \right) \left( \frac{10^{-\frac{1}{2} \log_{10}(1 + \frac{E_{\text{early}}}{E_{\text{late}}})} \cdot \alpha_r \frac{N}{1 - \alpha_r} \left( 1 - \alpha'_r \right)}{1 - \alpha_r \frac{N}{1 - \alpha_r}} \right). \quad (C.6) $$

As can be seen in the formula, $\gamma'_\text{comp-dir}$ can be calculated with the parameters $E_{\text{early}}$, $E_{\text{late}}$ and $\alpha'_r$. 