# Real time realization concepts of large adaptive filters

# Real Time Realization Concepts of large Adaptive Filters

G.P.M. Egelmeers

# Real Time Realization Concepts of Large Adaptive Filters

At the cover:
**Real**-ization concept of **Adaptive Filter** by Gabi Bertram.

# Real Time Realization Concepts of Large Adaptive Filters

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. J.H. van Lint, voor een commisie aangewezen door het College van Dekanen in het openbaar te verdedigen op maandag 20 november 1995 om 16.00 uur

door

**Gerardus Paul Maria Egelmeers**

geboren te Veldhoven

Dit proefschrift is goedgekeurd
door de promotoren:

prof.dr.ir. W.M.G. van Bokhoven
en
prof.dr.ir. P.P.J. van den Bosch

Copromotor: dr.ir. P.C.W. Sommen



CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Egelmeers, Gerardus Paul Maria

Real time realization concepts of large adaptive filters /
Gerardus Paul Maria Egelmeers. - Eindhoven : Technische
Universiteit Eindhoven
Proefschrift Technische Universiteit Eindhoven. - Met lit.
opg. - Met samenvatting in het Nederlands.
ISBN 90-386-0456-4
Trefw.: adaptieve filters / akoestische echo-compensator.

*to my parents*

# Summary

The real time application of large adaptive filters (thousands of coefficients) with a small processing delay (a few milli-seconds) and good convergence behaviour, as needed in, for example, the acoustic echo canceller, is not possible with the currently available adaptive filtering algorithms. Realization in time domain by transversal filter structures yields a computational complexity that makes real time implementation on, for example, a Digital Signal Processor (DSP), for a lot of applications impossible. Besides that, often convergence behaviour, when coloured input signals like speech are used, is not good enough for practical applications.

By using block processing and Fourier transforms, transversal filters (convolutions) can be performed efficiently in frequency domain. Applying these techniques in adaptive filtering leads to the Block Frequency Domain Adaptive Filter (BFDAF). The convergence behaviour of this algorithm for strongly correlated (coloured) input signals is improved by frequency domain normalization of the adaptation procedure for the coefficients.

A side effect of these block processing techniques is a processing delay equal to the block length. When processing delay, and thus block length, are bound to a certain (small) maximum, computational complexity becomes quite large. Partitioning of the convolution in smaller parts yields a much smaller computational complexity in Partitioned BFDAF (PBFDAF). This partitioning however implies that the length of the coefficient update part is also reduced, which means that the length of the normalization vector gets smaller. This can lead to bad convergence behaviour for highly correlated input signals.

By using different partition factors, block lengths and Fourier transform lengths in the update and the convolution part of the adaptive filter, a Decoupled PBFDAF (DPBFDAF) algorithm is obtained. When a certain maximum processing delay is allowed, this Decoupled algorithm has a much smaller computational complexity, and a larger normalization resolution in frequency domain, than the algorithm using only one partition factor. The

larger resolution of normalization can improve the convergence behaviour of the adaptive filter considerably for highly correlated input signals.

In applications where large adaptive filters with a small processing delay are needed, computational complexity of the Decoupled algorithm can be reduced even further by using a non-uniform partitioning of the filter in the convolution part. The resulting Non-Uniform Partitioned BFDAF (NUPBFDAF) algorithm has practically the same convergence properties as the DPBFDAF algorithm with a much smaller computational complexity.

By simulations and by implementation of an acoustic echo canceller on a single DSP, the adaptive filtering algorithms are compared. The results for acoustic echo cancellation, where a non stationairy and highly correlated input signal (speech) and a time variable echo-path are strong handicaps for the adaptive filtering algorithms, made it necessary to adjust the algorithms. Particularly normalization and stepsize control needed further improvements.

The DPBFDAF, and even more, the NUPBFDAF make it possible to use adaptive filters in real-time applications where both a large filter length and a small processing delay are required.

# Samenvatting

De "real time" toepassing van grote adaptieve filters (enkele duizenden coëfficiënten) met een kleine berekeningsvertraging (enkele milli-seconden) en goede convergentie eigenschappen, zoals bijvoorbeeld nodig is bij de akoestische echo compensator, leidt bij gebruik van de reeds langer bestaande algoritmen tot niet bruikbare oplossingen. De directe realisering in het tijddomein door middel van transversale filterstructuren levert een berekeningscomplexiteit op die voor veel toepassingen "real time" implementatie op bijvoorbeeld een Digitale Signaal Processor uitsluit. Bovendien zijn de convergentie eigenschappen bij gekleurde ingangssignalen, zoal bijvoorbeeld spraak, vaak onvoldoende voor praktische toepassingen.

Door gebruik te maken van blok berekeningsmethoden en Fourier transformaties kunnen transversale filters (convoluties) efficiënt in het frequentie domein berekend worden. Toepassing in adaptieve filters levert het "Block Frequency Domain Adaptive Filter" op. De convergentie eigenschappen van dit algoritme voor sterk gecorreleerde (gekleurde) ingangssignalen kunnen worden verbeterd door de adaptatie van de coëffiënten in het frequentie domein te normaliseren.

Door gebruik te maken van deze blok berekeningsmethoden ontstaat echter een berekeningsvertraging die overeenkomt met deze blok lengte. Indien de berekeningsvertraging, en dus de blok lengte, gebonden is aan een bepaald maximum kan de berekeningscomplexiteit toch nog vrij hoog oplopen. Het opdelen (partitioneren) van de convolutie in kleinere delen levert een veel lagere berekeningscomplexiteit op. Aangezien ook het "adaptatie" gedeelte wordt opgedeeld wordt de lengte van de normalisatie vector sterk gereduceerd wat kan leiden tot een verslechtering van de convergentie eigenschappen voor sterk gekleurde ingangssignalen.

Door gebruik te maken van verschillende partitie factoren, blok lengtes en Fourier transformatie lengtes in het "adaptatie" en het "convolutie" gedeelte van het adaptieve filter wordt een "Decoupled" algoritme verkregen. Bij een gelijke vertraging als bij het met één factor partitioneren,

heeft dit "Decoupled" algoritme een veel lagere berekeningscomplexiteit en een veel grotere normalisatie resolutie in het frequentie domein. Door deze laatste eigenschap kunnen de convergentie eigenschappen van het adaptieve filter voor sterk gekleurde ingangssignalen verbeteren.

Voor toepassingen van grote adaptieve filters met kleine toegestane maximale vertraging en grote filter lengte kan de berekeningscomplexiteit nog verder worden gereduceerd door gebruik te maken van een niet-uniforme partitie van het "convolutie" gedeelte. Het resulterende algoritme heeft vrijwel gelijke convergentie eigenschappen als het "Decoupled" algoritme bij een veel lagere berekeningscomplexiteit.

Door middel van simulaties en de implementatie van een akoestische echo compensator op één Signaal Processor zijn de eerder genoemde algoritmen getest. De resultaten bij de akoestische echo compensator, waar een niet stationair en sterk gecorreleerd ingangssignaal (spraak) en een tijdvariant echo-pad het adaptatie proces sterk bemoeilijken, maakten verdere aanpassingen aan de algoritmen noodzakelijk. Met name bij het regelen van de adaptatie-constante en de normalisatie blijken nog verbeteringen mogelijk.

Het "Decoupled" algoritme en, nog sterker, het niet-uniform gepartitioneerde algoritme maken het mogelijk om adaptieve filters te gebruiken in "real time" toepassingen waar zowel een grote filter lengte als een kleine berekeningsvertraging worden gevraagd.

# Contents

# Chapter 1

# Introduction

*After the introduction of the term Adaptive Filter some applications of these adaptive filters in the area of signal estimation, signal correction and signal prediction are mentioned. Some basic adaptive filter theory is needed for further development in the remaining chapters, where real-time realization concepts of large adaptive filters will be discussed.*

## 1.1 Linear Adaptive Filters

The object filter is used to describe a device performing the basic information processing operation: using noisy data to extract information about a quantity of interest. A filter is called linear when its output is a linear function of its inputs. By using certain parameters, such as correlation, variance and mean of the input data, we can minimize the mean square value of the error signal, defined as the average squared difference between some desired filter output and the actual filter output. When both the input signal and the quantity of interest are stationary the Wiener filter achieves this goal [58]. In the first instance, the Wiener filter theory was developed for continuous-time systems, later on it was applied to discrete-time systems. In this thesis, we will only consider discrete-time filters.

The use of Wiener filters assumes the availabilty of information about the correlations in the data to be processed. When not all of the information that is needed is known completely, the resulting filter (if it is still possible to design) may be non optimal. A solution to this problem is the use of adaptive filters.

An adaptive filter uses a recursive algorithm, which makes it possible for the filter to perform in an environment where knowledge of correlations in

the input signals is not available, or these correlations are varying slowly. In average the algorithm converges to the Wiener solution in some statistical sense. In the next section some applications of adaptive filters are discussed.

## 1.2 Applications of Adaptive Filters

### 1.2.1 Signal Estimation

#### System Identification for Signal Estimation

In the general signal estimation problem a known signal $x[k]$ is applied to an unknown system, possibly corrupted with noise, as depicted in figure 1.1. Our goal is to develop a model for this system in the form of a transversal

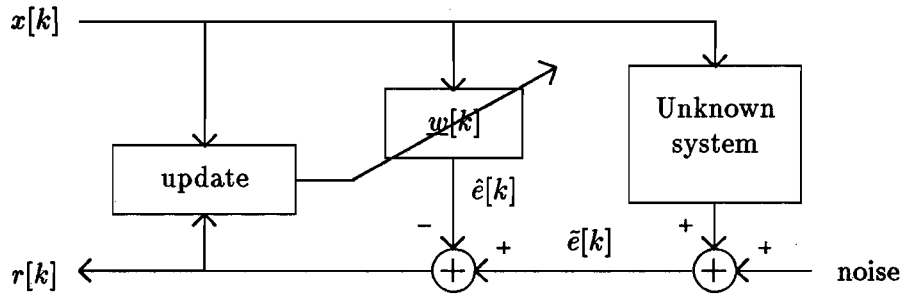



Figure 1.1: General signal estimator.

filter, consisting of a tapped delay-line and a corresponding set of adjustable coefficients $\underline{w}[k]$, with the length $N$ vector $\underline{w}[k]$ defined as

$$\underline{w}^N[k] = \left( \begin{array}{ccc} w_{N-1}[k] & \cdots & w_0[k] \end{array} \right)^t, \tag{1.1}$$

like in figure 1.2. The update algorithm of figure 1.1 has to adjust the coefficients of $\underline{w}[k]$ in such a way that the estimation error $r[k]$, defined as the difference between the measurable (corrupted) output $\tilde{e}[k]$ of the unknown system and the filter output $\hat{e}[k]$, is minimized in some statistical sense.

#### Acoustic Echo Cancellation

An application of signal estimation can be found in echo cancellation. In figure 1.3 a general teleconferencing scheme is given [28]. We see that two acoustic echos are generated ($e_1[k]$ and $e_2[k]$) caused by the fact that the microphones and loudspeakers are coupled by an acoustic path. By using

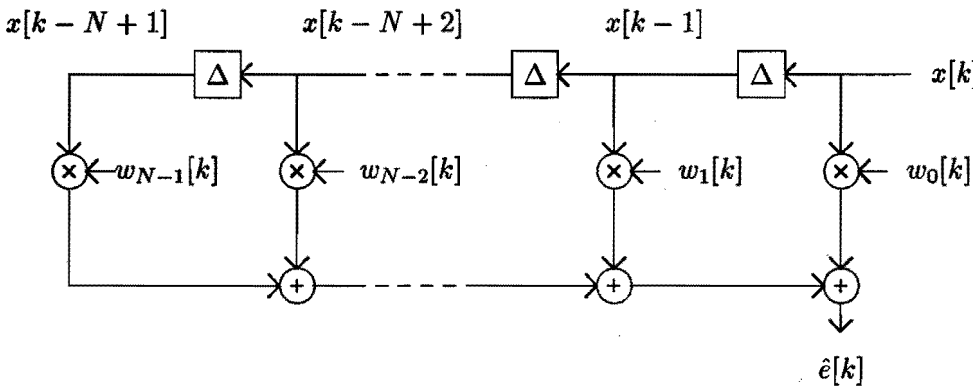



Figure 1.2: Transversal filter.

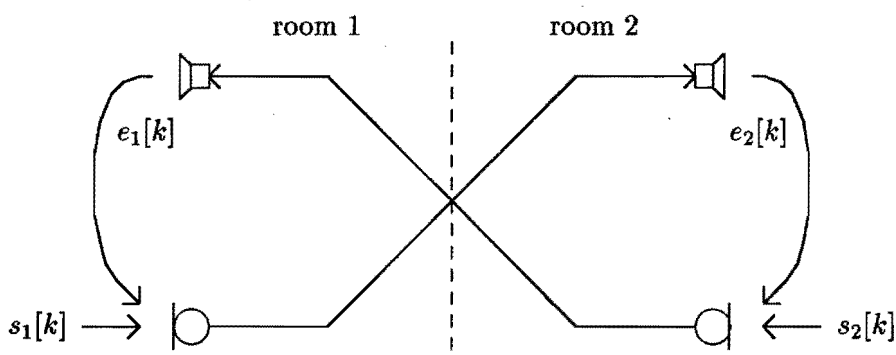



Figure 1.3: Teleconferencing scheme.

two Acoustic Echo Cancellers (AECs) these echos can be compensated. One of the AECs is isolated in figure 1.4. The goal of the AEC is to make

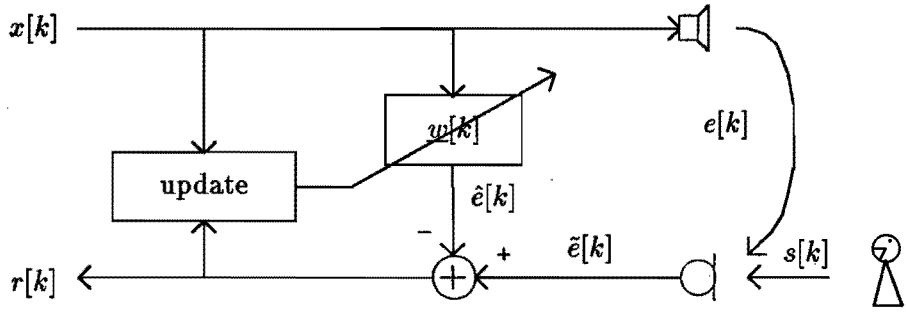



Figure 1.4: Acoustic echo canceller.

an estimate $r[k]$ of $s[k]$. This is achieved by making an estimate $\hat{e}[k]$ of $e[k]$ and subtract it from $\tilde{e}[k]$. The main problems here are, besides the physical length of the acoustic echo path, the non-stationarities in the input signal $x[k]$ and the time variant character of the echo path. When double talk is detected, meaning that both $s[k]$ and $x[k]$ contain speech, updating of the echo canceller coefficients is inhibited.

### Data Echo Cancellation

Another type of echo canceller is the Data Echo Canceller. When only two wires are available for bi-directional communication (for example in telephony [47]), we can connect a four- to a two-wire transmission using hybrids as shown in figure 1.3. Because the hybrids are not perfect, two

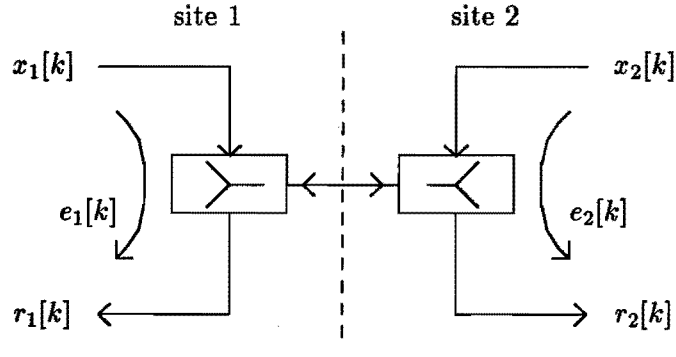



Figure 1.5: Two wire communication.

echos are generated ($e_1[k]$ and $e_2[k]$). By using two Data Echo Cancellers

these echos can be compensated. One of those Data Echo Cancellers is depicted in figure 1.6. The adaptive filter has to produce an estimate $\hat{e}[k]$

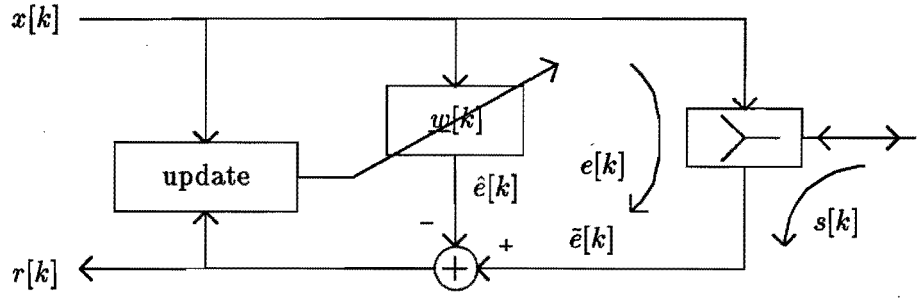



Figure 1.6: Data echo canceller.

of the echo $e[k]$ generated by the hybrid.

### Noise Cancellation

A third application of the signal estimator is the adaptive noise canceller [39]. In figure 1.7, an unknown system with impulse response $\underline{h}[k]$ colours the measurable noise $x[k]$. This results in the coloured noise signal $e[k]$, that corrupts a desired signal $s[k]$. An example is that of a car driver using a telephone, where $x[k]$ is produced by the car engine (picked up by a microphone near the engine) and $s[k]$ is the speech of the car driver. The (hands-free) telephone microphone picks up both the speech of the driver and the distorted engine noise, resulting in $\tilde{e}[k]$. The task of our adaptive

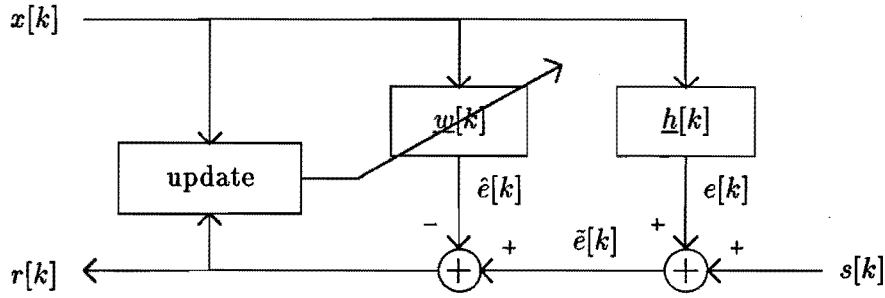



Figure 1.7: Noise canceller.

filter is to produce an estimate $\hat{e}[k]$ of the signal $e[k]$, which subsequently can be subtracted from the measurable signal $\tilde{e}[k]$, resulting in an estimate $r[k]$ of $s[k]$.

### Adaptive Beamforming

An adaptive beamformer [27] processes signals from a number of independent sensors (e.g. microphones or antennas) by an array of adaptive filters whose outputs are summed together (see figure 1.8). We try to adjust the adaptive filters in such a way that they generate a maximum output for a desired signal from a possibly unknown direction, and place nulls in the direction of the interference sources.

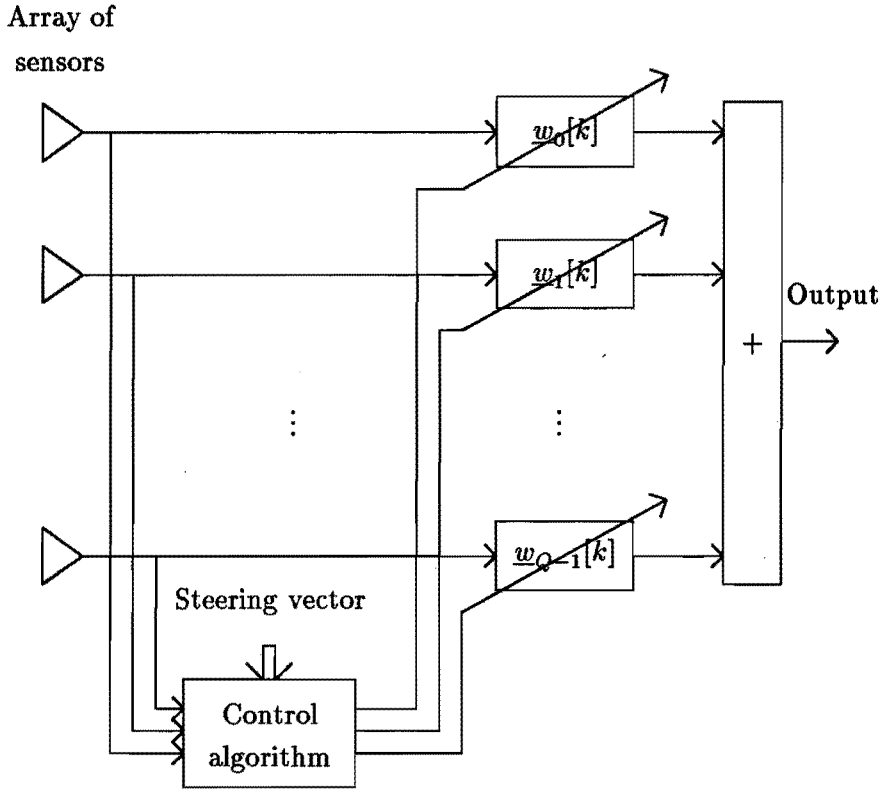



Figure 1.8: Adaptive beamformer.

## 1.2.2 Signal Correction

### Equalization

An example where signal correction is needed is the adaptive equalizer. A signal $e[k]$ is to be received through a channel with unknown impulse response $\underline{h}[k]$, corrupted by an additive noise signal $n[k]$. The resulting signal

$x[k]$ has to be corrected by an adaptive filter in such a way that the channel distortion is removed and the desired signal $e[k]$ can be estimated. In the

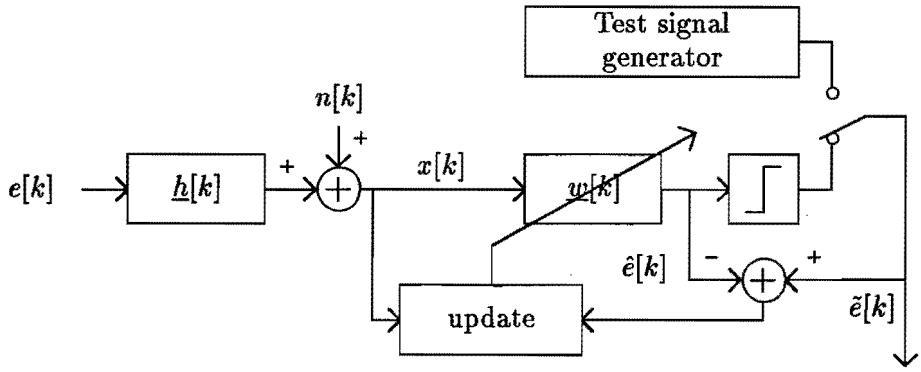



Figure 1.9: Adaptive equalizer.

Decision Directed Equalizer [27] from figure 1.9, the signal $e[k]$ is assumed to be discrete in amplitude (often binary, allowing only two different values for $e[k]$). The adaptive filter generates an estimate $\hat{e}[k]$ of $e[k]$, which is corrected by a decision element to the nearest discrete value allowed for $e[k]$. The adaptive filter error is estimated by taking the difference between the corrected estimate $\tilde{e}[k]$ of $e[k]$, that is the output from the decision element, and the output $\hat{e}[k]$ of the adaptive filter. In order for this system to work properly, the adaptive filter must be close to its optimal solution, which requires the use of an initial training sequence (generated by the test signal generator).[1]

### 1.2.3 Signal Prediction

#### Signal Prediction in General

A signal $x[k]$ consists of a predictable part $e[k]$, that can be determined from information of the past signal values, and an unpredictable part. An adaptive filter is used to produce an estimate $\hat{e}[k]$ of that predictable part. The residual signal $r[k] = x[k] - \hat{e}[k]$ then is an estimate of the unpredictable part of $x[k]$.

[1]On both sides of the communication channel, a test signal generator produces the same sequence of samples. In figure 1.9 only the test signal generator on the receiver side is depicted.

### Adaptive Line Enhancer

An example of a signal predictor is the Adaptive Line Enhancer (ALE) [56, 40], given in figure 1.10. An ALE can be used to detect a narrow-band

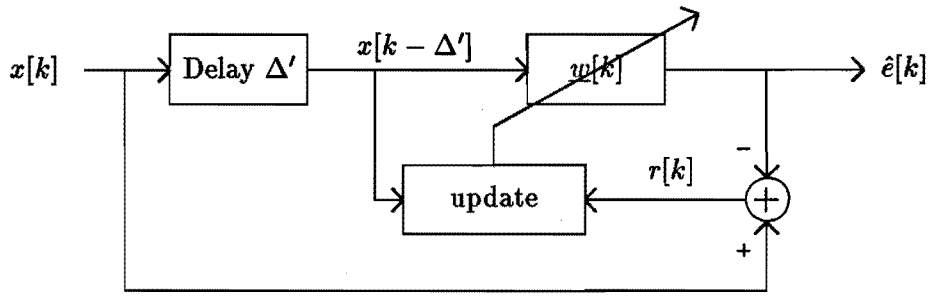



Figure 1.10: Adaptive line enhancer.

signal $e[k]$ (that is highly correlated), for example a sine-wave, embedded in wide-band noise $n[k]$ ($x[k] = n[k] + e[k]$). Assuming that there is no correlation between $n[k]$ and $n[k - \Delta']$, the adaptive filter produces an estimate $\hat{e}[k]$ of $e[k]$.

### Autoregressive Spectrum Analysis

Signal prediction can also be used in autoregressive spectrum analysis [27]. In figure 1.11 we see a signal $x[k]$, of which we assume that it is the output of a linear filter that is excited by a white-noise process. Further we assume that the filter has a transfer function that consists of poles only (Auto-Regressive (AR) model). The inverse transfer function thus is all-zero, and a transversal adaptive filter of sufficient order can be used. The adaptive filter is adjusted in such a way that $e[k]$ becomes a white noise signal.

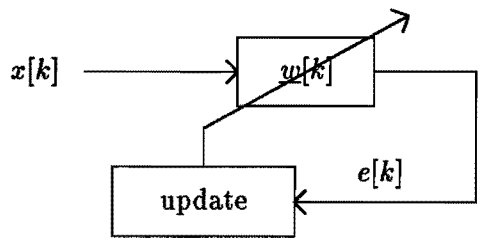



Figure 1.11: Autoregressive spectrum analysis.

## 1.3 Assumptions and Definitions

### 1.3.1 General Definitions

Before introducing general adaptive filter theory some assumptions and definitions have to be made. All in- and output signals are real and discrete in time, $x[k]$ denoting the signal $x$ at time instance $k$, which equals the continuous time instance $k\mathrm{T}$. T is the intersample distance (sample time), where $1/\mathrm{T} = f_\mathrm{S}$ symbolizes the sample frequency. The delay operation is named $\Delta$, delaying over one sample interval.

Signals (like $x[k]$) are described by lower case characters, while upper case characters represent constants (e.g. $A$). Underlining is used for vectors, lower case for time domain ($\underline{x}[k]$), and upper case for frequency domain ($\underline{X}[k]$). Matrices are denoted by bold face calligraphic or upper case characters (like $\boldsymbol{\mathcal{X}}[k]$ or $\mathbf{I}$).

Dimensions of vectors and matrices, when necessary, are put in superscript like $\underline{x}^N[k]$ for a length $N$ vector, or $\boldsymbol{\mathcal{X}}^{B,Q}[k]$ for a $B \times Q$ matrix. For a square matrix the second dimension may be omitted. Subscripts are used for further distinction, for example, $w_i$ denotes the $i$'th coefficient of $\underline{w}$.

An operation on a single scalar, vector or matrix is denoted by putting the scalar, vector or matrix between brackets and appending the operation:

- $(\underline{x}^N[k])^t$, $(\boldsymbol{\mathcal{X}}^{N,B}[k])^t$: transpose.
- $(x[k])^*$, $(\underline{x}^N[k])^*$, $(\boldsymbol{\mathcal{X}}^{N,B}[k])^*$: elementwise complex conjugate.
- $(\underline{x}^N[k])^h$, $(\boldsymbol{\mathcal{X}}^{N,B}[k])^h$: hermitian transpose (complex conjugate of transpose).
- $(\underline{x}^N[k])_a$, $(\boldsymbol{\mathcal{X}}^{N,B}[k])_{a,b}$: $a$'th, $(a,b)$'th element (starting with the 0'th element).
- $(x[k])^a$: $x[k]$ to the power $a$.

Another group of operations is denoted be putting the operand between curly brackets and putting the operator in front:

- $\mathcal{E}\{x[k]\}$: the mathematical expectation.[2]
- $\mathrm{diag}\{\boldsymbol{\mathcal{X}}^N\}$: a vector containing the diagonal of the $N \times N$ matrix $\boldsymbol{\mathcal{X}}^N$.

[2] $\mathcal{E}\{x[k]\}$ is the average of an ensemble of signals $x[k]$, which equals the average over time in the stationary case.

- $\mathrm{diag}\{\underline{x}^N\}$: the $N \times N$ matrix containing zeroes outside its main diagonal, with $\mathrm{diag}\{\mathrm{diag}\{\underline{x}^N\}\} = \underline{x}^N$.
- $\max\{N, B\}$, $\min\{N, B\}$: the maximum and minimum of $N$ and $B$.
- $\Re\{X\}, \Im\{X\}$: real and imaginary part of $X$, thus $X = \Re\{X\} + \jmath \cdot \Im\{X\}$, where $\jmath$ denotes the imaginary unit.
- $\lceil N/Q \rceil$: largest integer not larger than $N/Q$.
- $\lfloor N/Q \rfloor$: smallest integer not smaller than $N/Q$.
- $\gcd\{N, Q\}$: greatest common divisor of $N$ and $Q$.

For real valued signals $x[k]$, with $\mathcal{E}\{x[k]\} = 0$, the variance of $x[k]$ is denoted by $\sigma_x^2$, with

$$\sigma_x^2 = \mathcal{E}\{(x[k])^2\}, \tag{1.2}$$

while the autocorrelation coefficients $\rho_i$ are defined by

$$\rho_i = \mathcal{E}\{x[k] \cdot x[k-i]\}. \tag{1.3}$$

The $N \times N$ (auto-)correlation matrix of such a signal $x[k]$ is defined by

$$\mathcal{R}_x^N = \mathcal{E}\{\underline{x}^N[k] \cdot (\underline{x}^N[k])^t\}. \tag{1.4}$$

As we will be using non-stationary signals later on in this thesis, we will append a time index to the averages as soon as this is necessary. For the moment we assume that all signals involved are stationary.

In the glossary (page 189) an extended description about notation, abbreviations and the symbols used in the figures is given.

### 1.3.2 Assumptions for the Adaptive Filters

In figure 1.12 a general adaptive filter is depicted. We assume that:

- Both $x[k]$ and $s[k]$ have zero mean, so $\mathcal{E}\{x[k]\} = 0$ and $\mathcal{E}\{s[k]\} = 0$.
- There is no correlation between $x[k]$ and $s[k]$ (within the length of the adaptive filter), implying that $\mathcal{E}\{\underline{x}^N[k]s[k]\} = \underline{0}^N$.
- The unknown system $\underline{h}[k]$ can be modelled by a FIR filter of length $N$.[3]

[3] When this is not the case, we assume that the "rest" of $\underline{h}[k]$ (that cannot be modelled) is part of the desired signal $s[k]$.

- The filter vector $\underline{w}^N[k]$ is statistically independent of the input signal $x[k]$.[4]
- All input signals are stationary, and the unknown system is time invariant (in the first instance). Later we will use the developed adaptive filters under non-stationary conditions.

## 1.4 Adaptive Filter Theory

### 1.4.1 Introduction

For the purpose of further development we use the general signal estimation scheme of figure 1.1, of which a modified form is given in figure 1.12.

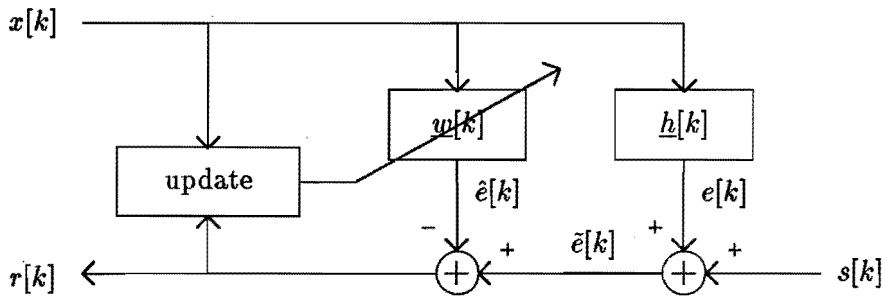



Figure 1.12: General adaptive filter.

In the previous section the assumption was made that the unknown system function, that can be represented by the vector $\underline{h}[k]$, can be modelled exactly with a Finite Impulse Response (FIR) filter. The task of the adaptive filter is to produce an estimate $\hat{e}[k]$ of the unknown signal $e[k]$, resulting from passing the signal $x[k]$ through the system $\underline{h}[k]$. For the moment, we assume that the unknown system is fixed (with impulse response $\underline{h}$).

The above implies that two processes take place in an adaptive filter:

1. The "update" process, which involves the adjustment of the tap weights of the filter according to some algorithm.
2. The "filter" process, which involves multiplying the tap inputs by the corresponding tap weights and generating an estimate of the desired

[4]This is not the case in practice, but by choosing the adaptation constant sufficiently small, this is an acceptable assumption.

response. Besides that an estimation error is generated by comparing the estimate $\hat{e}[k]$ to the corrupted response $\tilde{e}[k]$. This estimation error is coupled back ("coupling") to the update part to actuate the adaptive process.

Before proceeding with adaptive filtering, a short introduction to Wiener (fixed) filter theory is given.

### 1.4.2 Wiener Filter Theory

#### Optimum Filtering

Consider the linear transversal (fixed) filter of figure 1.13. We assume real

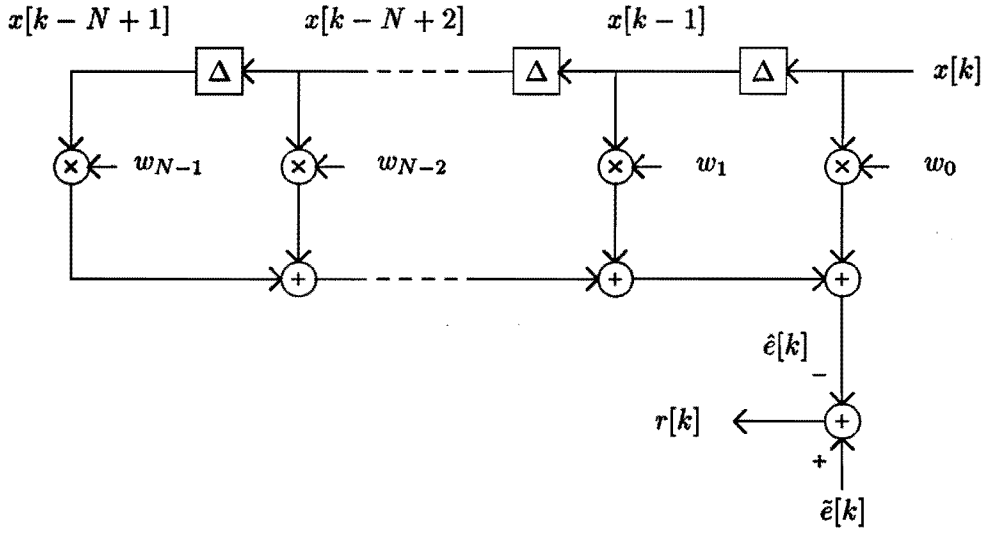



Figure 1.13: Transversal filter.

valued stationary inputs, with zero mean. The filter output can then be written by the convolution sum

$$\begin{aligned} \hat{e}[k] &= \sum_{i=0}^{N-1} w_i \cdot x[k-i] \\ &= (\underline{w}^N)^t \underline{x}^N[k] \end{aligned} \tag{1.5}$$

where the filter vector $\underline{w}^N$ is given by

$$(\underline{w}^N)^t = \begin{pmatrix} w_{N-1} & \cdots & w_0 \end{pmatrix} \tag{1.6}$$

and the input signal vector $\underline{x}^N[k]$ by

$$\underline{x}^N[k] = \begin{pmatrix} x[k-N+1] & \cdots & x[k] \end{pmatrix}^t. \tag{1.7}$$

The goal is to estimate the filter coefficients $w_0$ till $w_{N-1}$ in such a way that the difference $r[k]$ between the desired response $\tilde{e}[k]$ and the actual filter response $\hat{e}[k]$ is minimized in some statistical sense, with

$$\begin{aligned} r[k] &= \tilde{e}[k] - \hat{e}[k] \\ &= \tilde{e}[k] - (\underline{w}^N)^t \underline{x}^N[k]. \end{aligned} \tag{1.8}$$

In Wiener filter theory the minimum Mean Square Error (MSE) criterion is used for optimization, with the MSE defined as

$$\Upsilon_{\mathbf{w}} = \mathcal{E}\{(r[k])^2\}. \tag{1.9}$$

Minimizing $\Upsilon_{\mathbf{w}}$ with respect to $\underline{w}$ yields the optimum linear filter in the minimum mean square sense.

### Mean Square Error (MSE)

The MSE can be expressed with equations (1.8) and (1.9) as follows

$$\begin{aligned} \Upsilon_{\mathbf{w}} &= \mathcal{E}\{(\tilde{e}[k])^2\} - (\underline{w}^N)^t \mathcal{E}\{\underline{x}^N[k]\tilde{e}[k]\} - \mathcal{E}\{\tilde{e}[k](\underline{x}^N[k])^t\}\underline{w}^N \\ &\quad + (\underline{w}^N)^t \mathcal{E}\{\underline{x}^N[k](\underline{x}^N[k])^t\}\underline{w}^N. \end{aligned} \tag{1.10}$$

We assume that $\tilde{e}[k]$ has zero mean and that $\underline{x}^N[k]$ and $\tilde{e}[k]$ are jointly stationary and denote the variance of $\tilde{e}[k]$ as

$$\sigma_{\tilde{e}}^2 = \mathcal{E}\{(\tilde{e}[k])^2\}. \tag{1.11}$$

With equations (1.4), (1.11) and defining the crosscorrelation between $\underline{x}^N[k]$ and $\tilde{e}[k]$ by

$$\underline{p}_{x,\tilde{e}}^N = \mathcal{E}\{\underline{x}^N[k]\tilde{e}[k]\} \tag{1.12}$$

we can write equation (1.10) as

$$\Upsilon_{\mathbf{w}} = \sigma_{\tilde{e}}^2 - (\underline{p}_{x,\tilde{e}}^N)^t \underline{w}^N - (\underline{w}^N)^t \underline{p}_{x,\tilde{e}}^N + (\underline{w}^N)^t \mathcal{R}_x^N \underline{w}^N. \tag{1.13}$$

$\Upsilon_{\mathbf{w}}$, as in the above equation (1.13), is precisely a quadratic function of the adaptive weight vector $\underline{w}^N$. As $\mathcal{R}_x^N$ is positive definitive, the dependence of the mean square error $\Upsilon_{\mathbf{w}}$ on the unknown weights may be visualized in the form of a multidimensional paraboloid with a uniquely defined bottom

or minimum point $\Upsilon_{\min}$. The weights corresponding to this minimum point define the optimum Wiener solution vector $\underline{w}^N_{\text{opt}}$. In this point the gradient $\underline{\nabla}^N$ of $\Upsilon_{\mathbf{w}}$, defined as

$$\underline{\nabla}^N = \frac{\delta \Upsilon_{\mathbf{w}}}{\delta \underline{w}^N} \tag{1.14}$$

equals a vector containing all zeroes. This is depicted for a single adaptive weight in figure 1.14.

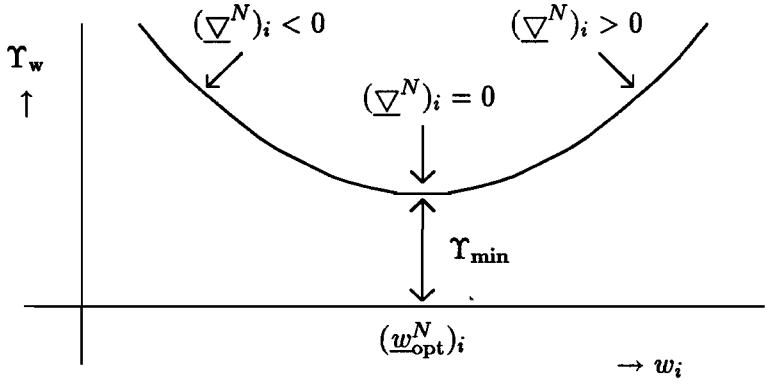



Figure 1.14: The MSE as function of a single adaptive weight $w_i$.

### Optimum Tap Weight Vector

To determine the optimum tap weight vector $\underline{w}^N_{\text{opt}}$ we have to calculate the gradient vector $\underline{\nabla}^N$. From equation (1.14) and (1.13) it follows that

$$\underline{\nabla}^N = -2\underline{p}^N_{x,\tilde{e}} + 2\mathcal{R}^N_x \underline{w}^N. \tag{1.15}$$

For the optimum tap weigth vector, where $\underline{\nabla}^N$ equals the null vector, this implies

$$\underline{w}^N_{\text{opt}} = (\mathcal{R}^N_x)^{-1} \underline{p}^N_{x,\tilde{e}}. \tag{1.16}$$

From equations (1.13) and (1.16) it follows that the minimum MSE, reached for $\underline{w}^N = \underline{w}^N_{\text{opt}}$, equals

$$\Upsilon_{\min} = \sigma^2_{\tilde{e}} - (\underline{p}^N_{x,\tilde{e}})^t \underline{w}^N_{\text{opt}}. \tag{1.17}$$

### 1.4.3 Application of Wiener to Adaptive Filters

#### Structure and Introduction

Now we will apply the Wiener filter theory to adaptive filtering. As stated before, the structure of the adaptive filter consists of two basic parts: a transversal filter with adjustable tap weights $w_0[k]$ till $w_{N-1}[k]$ and a mechanism for updating these weights. The update part has to solve the "normal equation" of the adaptive filter (equation (1.16)). To avoid the computational difficulties of doing this explicitly we can use iteration by the method of steepest descent [33].

#### Steepest Descent

Succesive corrections to the tap weight vector in the direction of the negative of the gradient vector (in the direction of the steepest descent of the error performance surface) should intuitively lead to the optimum tap weigth vector [57, 26]. This can be achieved by using a steepest descent update, from which the update rule looks like

$$\underline{w}^N[k+1] = \underline{w}^N[k] - \alpha \underline{\nabla}^N[k], \tag{1.18}$$

where $\alpha$ is the step size (or adaptation) parameter ($\alpha > 0$).

If it were possible to make exact measurements of the gradient vector at each iteration and if the step size parameter $\alpha$ is suitably chosen, then the tap weight vector would indeed converge to the optimum Wiener solution using this steepest descent method [57, 26]. In practice however, these exact measurements of the gradient vector are not possible, so an estimate of the gradient vector has to be used. A well known and simple algorithm that uses such an estimate is the Least Mean Square (LMS) algorithm [54, 55]. This LMS algorithm will be the starting point for the sequel of this thesis.

## 1.5 Properties of Adaptive Filters

From now on the focus will be on efficient implementations of adaptive algorithms when the filter length $N$ is very large (say $N \gg 500$). We will keep in mind that we want to realize such an adaptive filter on a Digital Signal Processing system, and if possible, with the use of only a single Digital Signal Processor (DSP). To do so we have to look at some properties of the diverse algorithms mentioned in this thesis, in order to make a good comparison between them.

- Processing delay:
  - Algorithm processing delay ($\Delta$): delay caused by the structure of the algorithm (thus only depending on the sample rate and the algorithm, and not on the specific hardware used).
  - Computation processing delay ($D$): delay caused by the delay of the hardware that is used.
- Computational complexity: number of
  - Multiplications.
  - Additions and Subtractions.
  - Loads and Stores.
  - Other (Divisions, Square roots, ...).

  DSPs, in general, have single cycle multiplications, and most of them are capable of performing additions, loads and stores in parallel to those multiplications. In this thesis we will use as a measure for complexity the number of real floating point multiplications per sample ($\Psi$). The operations that are not single cycle in general, like divisions, square roots, ..., will be performed by algorithms using only single cycle instructions (see chapter 6). Although this number of real floating point multiplications per sample is not an exact measure for the number of instructions (or cycles) needed for a DSP implementation, it does give a good indication for the case of adaptive filtering, as the number of "other" operations is proportional to the number of multiplications in the algorithms used in this thesis.
- Memory occupation ($\Theta$): least number of memory locations that a DSP implementation of an adaptive filtering algorithm needs for its data.
- Convergence and Tracking: we distinguish between
  1. Performance under time-invariant conditions (time-invariant unknown system impulse response and stationary input signals), where the final misadjustment $\Upsilon$ and convergence speed are parameters of interest. Note that in this thesis only qualitative comparisons between diverse algorithms are made. The noise in the adaptive coefficients (caused by a too large step-size parameter $\alpha$), causes a misadjustment $\Upsilon_1$.

2. Performance when the unknown system impulse response is time-variant (like an acoustic echo) ("tracking" properties). Here a too small step-size parameter $\alpha$ can make the adaptation process too slow to follow the unknown system impulse response, implying a misadjustment $\Upsilon_2$, while a too large block size (for block processing algorithms) implies an extra misadjustment $\Upsilon_3$.
3. Performance when the input signals are non-stationary. (like speech).

- Numerical behaviour: inaccuracies are produced caused by the finite precision when operating on DSPs. We distinguish between numerical stability and numerical accuracy. The former is an inherent characteristic of an adaptive filtering algorithm, while the latter is determined by the number of bits used in the representation of data in the adaptive filter.

In appendix A properties (particularly computional complexity and computation delay) of some important elements (Fast Fourier Transform, Elementwise multiplications, Power estimation) are explained in detail.

Every algorithm will be illustrated by an example. For this example we keep the acoustic echo canceller in mind, assuming that we have a sample frequency of 8 kHz (telephony standard) and a filter of 4000 coefficients (cancelling an echo of 500 milli-seconds)[5] that has to be implemented on one Texas Instruments TMS320C30 signal processor. We will try to find an algorithm with a total processing delay $D_{\max}$ smaller than 0.5 milli-second.[6]

## 1.6 Further Outline

This thesis is organized as follows: After the introduction of the Least Mean Square (LMS) algorithm a short derivation of the well known Block Normalized Least Mean Square (BNLMS) algorithm is given. From literature [57] it is known that convergence properties of this algorithm depend on the correlation in the input signal. The Recursive Least Squares (RLS) method uses past information of the input signal $x[k]$ to remove this dependency. To decrease complexity the amount of past information needed can be reduced, which is shown by using a geometrical approach. The results

[5]Most room impulse responses can be modelled accurately by an impulse response of this length.

[6]The development of world-wide standards for delays in telecommunication equipment tend towards a maximum allowable delay of 1 milli-second for the whole system.

are the Block Orthogonal Projection (BOP) algorithm, and a generalization of the BOP algorithm, the Decoupled BOP (DBOP) algorithm. In these (D)BOP and RLS algorithms the influence of the correlation in the input signal on the convergence properties of the adaptive filter is reduced by using an estimate of the inverse autocorrelation matrix. The result of this decorrelation is that, in general, convergence properties of the adaptive filter become better.

The two main operations of the adaptive algorithms in this thesis are:

1. Calculation of the output signal $\hat{e}[k]$ of the adaptive filter. This is done by convolving the adaptive weight vector $\underline{w}^N[k]$ with the input signal vector $\underline{x}^N[k]$.

2. Calculation of an estimate of the gradient vector $\underline{\nabla}^N[k]$. For this the crosscorrelation between the signal vector $\underline{x}^N[k]$ and the residual signal $r[k]$ has to be calculated.

From literature [34] it is known that, for large filters, the convolution and correlation operations can be calculated efficiently by implementation on block basis and evaluation in frequency domain. However, the main disadvantage of this technique is the resulting processing delay.

In chapter 3 a mathematical description is given for an efficient implementation of both a convolution and a correlation operation in frequency domain by using block processing techniques. From this an efficiently implemented BNLMS algorithm follows in a straightforward way. To implement the BOP and RLS algorithms in an efficient way, the same approach is followed. The extra step here, in comparison with BNLMS, is that an approximation has to be made in order to implement the decorrelation by the inverse autocorrelation matrix efficiently in frequency domain using (fast) Fourier transforms. The result of this approach is the Block Frequency Domain Adaptive Filter (BFDAF) [8]. In this approach the influence of the input signal correlation is reduced by normalization of each separate Fourier transform output by its variance.

The drawback of using the BFDAF is that is not possible to satisfy both the following contradictive requirements:

1. The block length has to be as small as possible for a small processing delay.

2. The block length has to be as large as possible to obtain an efficient implementation.

These contradictive requirements can be circumvented by applying partition techniques to the filter and the update part. The results are the Partitioned BFDAF (PBFDAF) and the Decoupled Partitioned BFDAF (DPBFDAF). The former uses one partition factor to partition both filter and update part, while the latter uses two different partition factors. With the DPBFDAF approach we are able to choose an implementation that realizes simultaneously a given minimal allowable processing delay and a small computational complexity.

Complexity however can still be too large for practical situations, such as large Acoustic Echo Cancellers (AECs) at high frequency (8-16 kHz.). Further reduction in complexity can be achieved by using a non-uniform partition technique in the filter part of the adaptive filter. This leads to the Non-Uniform Partitioned BFDAF (NUPBFDAF). With this new approach complexity can be reduced even further than in the DPBFDAF case.

In chapter 6 methods for normalization in DPBFDAF and NUPBFDAF are introduced. A decoupling of the update part dimensions and the normalization vector length improves the flexibility of the algorithms.

The properties of the given algorithms are compared to each other, together with the results of simulations in chapter 7. The DPBFDAF algorithm has been realized in a practical Acoustic Echo Canceller on a single Digital Signal Processor (DSP) as described in chapter 8. Finally some conclusions are given in chapter 9.

# Chapter 2

# Time Domain Adaptive Filtering

*The best known algorithm for adaptive filtering in time domain is the Least Mean Square (LMS) algorithm. Due to its simple structure and low computational complexity compared to other time domain algorithms, it has become very popular. Its convergence behaviour however, depends strongly on the input signal variance and (auto-)correlation.*

*By normalization of the update equation of LMS we obtain the Normalized LMS (NLMS) algorithm. The NLMS convergence behaviour no longer depends on the input signal variance. Both the LMS and NLMS algorithm make one update every sample interval. The Block (N)LMS (B(N)LMS) algorithm performs only one update every B sample intervals. The convergence properties of this generalization of the (N)LMS algorithm still depend on the input signal (auto-)correlation. Since many physical processes of interest, such as speech and special codes, are highly correlated, we would like to have adaptive algorithms that are less dependent on these characteristics of the input signal.*

*In time domain the Recursive Least Squares (RLS) algorithm decorrelates the input signal with the inverse of the $N \times N$ autocorrelation matrix of the input signal. The RLS algorithm minimizes the dependency of the convergence behaviour on the input signal correlation, but even fast implementations imply a huge computational complexity.*

*The (Block) Orthogonal Projection ((B)OP) algorithm decorrelates the input signal with its $B \times B$ autocorrelation matrix, where $1 \leq B \leq N$. This method, that is a generalization of both BNLMS (for $B = 1$) and (a block version of) RLS (for $B = N$), decreases the dependency of the convergence*

*behaviour on the input signal correlation, but also increases computational complexity, compared to BNLMS. The coupling of the block length B to the autocorrelation matrix dimension in BOP can be removed. The resulting Decoupled BOP (DBOP) is a generalization of the BOP algorithm.*

## 2.1 Least Mean Square (LMS)

### 2.1.1 LMS Algorithm

From chapter 1 we know that the application of Wiener filter theory to adaptive filtering requires the use of an estimate $\hat{\underline{\nabla}}^N[k]$ of the gradient vector $\underline{\nabla}^N[k]$ (in equation (1.15)), where

$$\underline{\nabla}^N[k] = -2\underline{p}^N_{x,\tilde{e}}[k] + 2\mathcal{R}^N_x[k]\underline{w}^N[k] \tag{2.1}$$

with

$$\underline{w}^N[k] = \begin{pmatrix} w_{N-1}[k] & \cdots & w_0[k] \end{pmatrix}^t. \tag{2.2}$$

To obtain such an estimate, the Least Mean Square (LMS) algorithm uses instantaneous estimates $\hat{\mathcal{R}}^N_x[k]$ and $\hat{\underline{p}}^N_{x,\tilde{e}}[k]$ of the (auto-)correlation matrix $\mathcal{R}^N_x[k]$ and cross-correlation vector $\underline{p}^N_{x,\tilde{e}}[k]$ as given by

$$\begin{aligned} \hat{\mathcal{R}}^N_x[k] &= \underline{x}^N[k](\underline{x}^N[k])^t && (2.3)\\ \hat{\underline{p}}^N_{x,\tilde{e}}[k] &= \underline{x}^N[k]\tilde{e}[k], && (2.4) \end{aligned}$$

where

$$\underline{x}^N[k] = \begin{pmatrix} x[k-N+1] & \cdots & x[k] \end{pmatrix}^t. \tag{2.5}$$

This implies for the instantaneous estimate of the gradient vector

$$\begin{aligned} \hat{\underline{\nabla}}^N[k] &= -2\underline{x}^N[k](\tilde{e}[k] - (\underline{x}^N[k])^t\underline{w}^N[k]) \\ &= -2\underline{x}^N[k]r[k]. && (2.6) \end{aligned}$$

Using equation (2.6), equation (1.18) and figure 1.12, we get the next pair of relations for the LMS algorithm [54, 55, 57]

$$\begin{aligned} r[k] &= \tilde{e}[k] - (\underline{w}^N[k])^t\underline{x}^N[k] && (2.7)\\ \underline{w}^N[k+1] &= \underline{w}^N[k] + 2\alpha\underline{x}^N[k]r[k]. && (2.8) \end{aligned}$$

The above equations are depicted in figures 2.1 and 2.2. A one sample delay (delaying over T seconds, with $1/\mathrm{T} = f_s$) is depicted by boxes labelled "$\Delta$".

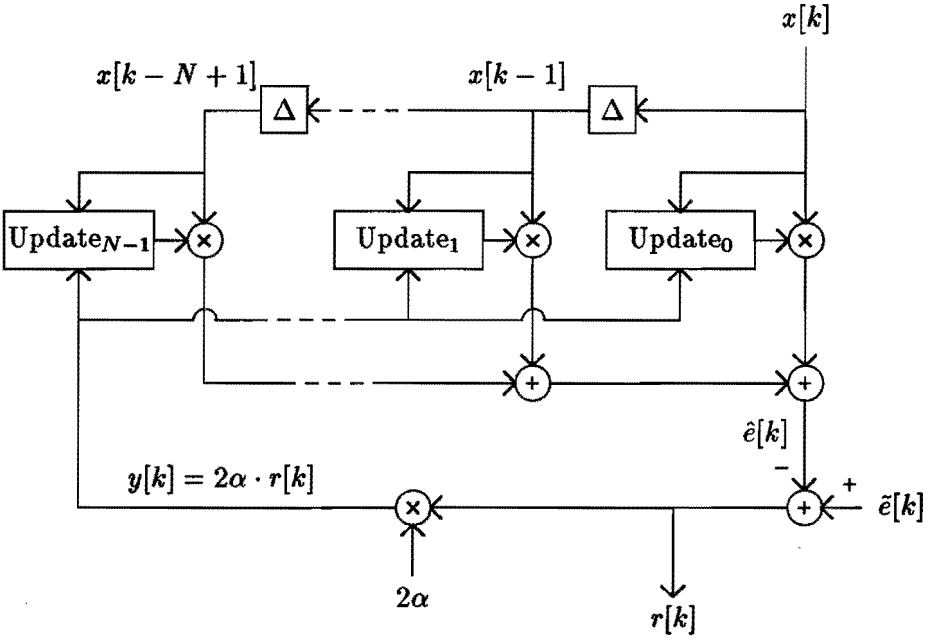



Figure 2.1: Least Mean Square (LMS) algorithm.

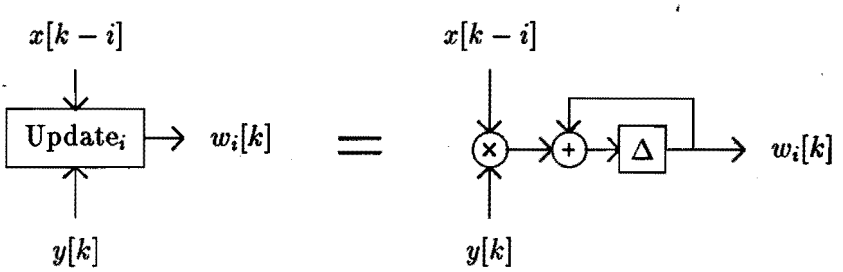



Figure 2.2: Update block of LMS.

### 2.1.2 Properties of LMS

The following features of the Least Mean Square algorithm can be derived:

- The algorithm itself has no inherent delay (for all time indices $k$, $r[k]$ depends only on input signals with a time index that equals at most $k$). This means for the algorithm processing delay $\Delta_{\text{LMS}}$ that

$$\Delta_{\text{LMS}} = 0. \tag{2.9}$$

- Computation processing delay (for only one multiplication, see appendix A)

$$D_{\text{LMS}} < 1. \tag{2.10}$$

- Computational complexity

$$\Psi_{\text{LMS}} = 2 \cdot N. \tag{2.11}$$

- Memory occupation:
  - Input delayline (length $N$)
  - Weight vector $\underline{w}^N$ (length $N$)

$$\Theta_{\text{LMS}} = 2 \cdot N. \tag{2.12}$$

### 2.1.3 Convergence and Tracking

From literature [54, 55, 57, 46] it is known that the convergence behaviour of the LMS algorithm depends strongly on the input signal variance and autocorrelation. The final misadjustment and the speed of convergence can degrade rather heavily when highly correlated ("coloured") input signals are used (such as speech). If we look at the LMS update equation and define the difference channel $\underline{d}^N[k]$ as the difference between the actual impulse reponse $\underline{h}^N[k]$ to estimate, and the estimated impulse response $\underline{w}^N[k]$, (thus $\underline{d}^N[k] = \underline{h}^N[k] - \underline{w}^N[k]$) we get from equation (2.8) and the assumptions in section 1.3 that

$$\begin{aligned} \mathcal{E}\{\underline{d}^N[k+1]\} &= \mathcal{E}\{(I^N - 2\alpha\underline{x}^N[k](\underline{x}^N[k])^t)\,\underline{d}^N[k] - 2\alpha\underline{x}^N[k]s[k]\} \\ &\approx (I^N - 2\alpha\mathcal{R}_x^N[k])\mathcal{E}\{\underline{d}^N[k]\} \end{aligned} \tag{2.13}$$

The presence of the input signal autocorrelation matrix $\mathcal{R}_x^N[k]$ in the above equation is the main cause of degradation in convergence properties. The

distribution of the eigenvalues of that matrix defines the speed of convergence and the final misadjustment [54, 55, 57, 46].

When the unknown impulse response $\underline{h}^N[k]$ is time varying (meaning that its is non-stationary), the tracking of the LMS algorithm becomes important. Increasing $\alpha$ has to contradictive effects. It decreases the misadjustment $\Upsilon_2$ caused by the time varying effect of $\underline{h}^N[k]$, while it increases the misadjustment $\Upsilon_1$ caused by the noise in the coefficient vector $\underline{w}^N[k]$. There will be an optimum for $\alpha$ (leading in average to the smallest misadjustment $\Upsilon_1 + \Upsilon_2$), depending on the amount of non-stationarity in $\underline{h}^N[k]$.

## 2.2 Normalized LMS (NLMS)

### 2.2.1 NLMS Algorithm

As the residual signal $r[k] = \tilde{e}[k] - (\underline{x}^N[k])^t \underline{w}[k]$ contains the input signal vector, convergence of the LMS algorithm depends on the variance $\sigma_x^2[k] = \mathcal{E}\{(x[k])^2\}$.[1] This effect can be cancelled by normalizing the adaptation constant $\alpha$ by an estimate $\hat{\sigma}_x^2[k]$ of this variance. This results in the Normalized LMS (NLMS) algorithm

$$r[k] = \hat{e}[k] - (\underline{w}^N[k])^h \underline{x}^N[k] \tag{2.15}$$

$$\underline{w}^N[k+1] = \underline{w}^N[k] + \frac{2\alpha}{\hat{\sigma}_x^2[k]} \underline{x}^N[k] r[k] \tag{2.16}$$

Equations (2.15) and (2.16) are depicted in figures 2.3 and 2.2. In chapter 6 procedures to estimate the variance are discussed. An efficient way to calculate the inverse input signal variance is compared to direct variance estimation.

### 2.2.2 Properties of NLMS

Most of the properties of NLMS are equal to the corresponding LMS properties:

[1] To be more precise, convergence depends on

$$\frac{1}{N} \sum_{i=0}^{N-1} \sigma_x^2[k] = \frac{1}{N} \mathcal{E}\{(\underline{x}^N[k])^t \underline{x}^N[k]\}. \tag{2.14}$$

In the estimation procedures discussed in chapter 6 this is inherently taken into account by using the "past" of $x[k]$ to obtain an estimate of $\sigma_x^2[k]$.

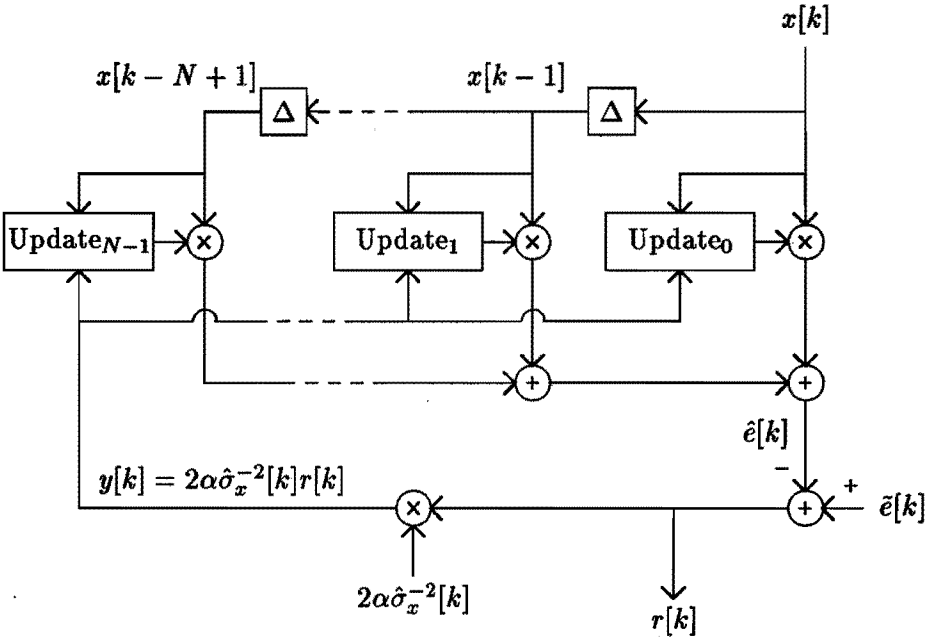



Figure 2.3: Normalized LMS (NLMS) algorithm.

- Algorithm processing delay

$$\Delta_{\text{NLMS}} = 0. \tag{2.17}$$

- Computation processing delay (for only one multiplication, see appendix A)

$$D_{\text{NLMS}} < 1. \tag{2.18}$$

- Computational complexity, where $\Psi_\sigma$ is the computational complexity of the variance estimation procedure (see chapter 6)

$$\begin{aligned} \Psi_{\text{NLMS}} &= 2 \cdot N + \Psi_\sigma \\ &\approx 2 \cdot N. \end{aligned} \tag{2.19}$$

- Memory occupation:
  - Input delayline (length $N$)
  - Weight vector $\underline{w}^N$ (length $N$)

$$\Theta_{\text{NLMS}} = 2 \cdot N. \tag{2.20}$$

### 2.2.3 Convergence and Tracking

The normalization has cancelled the dependence of the convergence behaviour on the input signal variance (if one assumes that a perfect variance estimator is used). This however only means that the adaptation constant is normalized, and has no influence on the relative distribution of the eigenvalues of the autocorrelation matrix. This implies that the dependency of convergence behaviour on the eigenvalue distribution in the input signal is equal to that in LMS.

## 2.3 Block NLMS (BNLMS)

### 2.3.1 BNLMS Algorithm

The Block NLMS (BNLMS) algorithm combines more or less $B$ updates of the NLMS algorithm. This implies that only once every $B$ input samples, thus once every $B \cdot T$ seconds, an update takes place, with the block length $B \geq 1$. To describe this mathematically, we will first introduce the $N \times B$ input signal matrix $\mathcal{X}^{N,B}[\kappa B]$. The block index $\kappa$ is an integer, so $\kappa B$ denotes a time instance that is an integer multiple of $B$. This matrix $\mathcal{X}^{N,B}[\kappa B]$ contains the $B$ most recent input signal vectors

$$\mathcal{X}^{N,B}[\kappa B] = \left( \underline{x}^N[\kappa B - B + 1] \quad \cdots \quad \underline{x}^N[\kappa B - 1] \quad \underline{x}^N[\kappa B] \right) \tag{2.21}$$

and the adaptive weight vector at time instance $\kappa B$ equals

$$\underline{w}^N[\kappa B] = \left( w_{N-1}[\kappa B] \quad \cdots \quad w_1[\kappa B] \quad w_0[\kappa B] \right)^t. \tag{2.22}$$

The filter part of the algorithm yields the vector $\underline{\hat{e}}^B[\kappa B]$

$$\underline{\hat{e}}^B[\kappa B] = \left( \hat{e}[\kappa B - B + 1] \quad \cdots \quad \hat{e}[\kappa B] \right)^t \tag{2.23}$$

by combining $B$ filter operations into one vector

$$\begin{aligned} \underline{\hat{e}}^B[\kappa B] &= \left( (\underline{x}^N[\kappa B - B + 1])^t \underline{w}^N[\kappa B] \quad \cdots \quad (\underline{x}^N[\kappa B])^t \underline{w}^N[\kappa B] \right)^t \\ &= (\mathcal{X}^{N,B}[\kappa B])^t \underline{w}^N[\kappa B]. \end{aligned} \tag{2.24}$$

For the update part we define the residual signal vector $\underline{r}^B[\kappa B]$ containing the $B$ most recent residual signal samples as

$$\begin{aligned} \underline{r}^B[\kappa B] &= \left( r[\kappa B - B + 1] \quad \cdots \quad r[\kappa B] \right)^t \\ &= \underline{\tilde{e}}[\kappa B] - \underline{\hat{e}}[\kappa B]. \end{aligned} \tag{2.25}$$

By taking again the instantanous value of the gradient vector [8] and normalizing, like in the NLMS case, we get for the BNLMS update equation

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + \frac{2\alpha}{\hat{\sigma}_x^2[\kappa B]}\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B] \tag{2.26}$$

where $\hat{\sigma}_x^2[\kappa B]$ is an estimate of the "block" variance $\sigma_x^2[\kappa B]$, defined by

$$\sigma_x^2[\kappa B] = \frac{1}{B \cdot N}\sum_{i=0}^{B-1}\mathcal{E}\{(\underline{x}^N[\kappa B - i])^t\underline{x}^N[\kappa B - i]\} \tag{2.27}$$

Estimation procedures for $\hat{\sigma}_x^2[\kappa B]$ can be found in chapter 6. The BNLMS algorithm is depicted in figures 2.4 and 2.5. The update of all weight coefficients is performed once every $B$ samples. The Serial to Parallel converters in figure 2.4 denote the collection of $B$ samples at rate $1/T = f_S$ into one vector at rate $f_S/B$, while the Parallel to Serial converter denotes the inverse operation. The boxes "$\Delta'$" depict one sample delays, each delaying over $B \cdot T$ seconds (caused by the reduced sample rate in the Update blocks). In the glossary at page 195 the symbols used in the figures are described.

### 2.3.2 Properties of BNLMS

Block processing has some consequences for the features of the Block NLMS algorithm:

- The algorithm itself contains a delay (see figure 2.4). This means for the algorithm processing delay $\Delta_{\text{BNLMS}}$ that

$$\Delta_{\text{BNLMS}} = B - 1. \tag{2.28}$$

- Computation processing delay for $B$ multiplications and additions (see appendix A)

$$D_{\text{BNLMS}} < 1. \tag{2.29}$$

- Computational complexity, where $\Psi_\sigma\{B\}$ is the complexity of the normalization procedure (that must be carried out once every $B$ samples, see chapter 6)

$$\Psi_{\text{BNLMS}} = 2 \cdot N + \frac{\Psi_\sigma\{B\}}{B}. \tag{2.30}$$

- Memory occupation:
  - Input delayline (length $N + B$) and output values (length $B$)
  - Weight vector $\underline{w}^N$ (length $N$)

$$\Theta_{\text{BNLMS}} \approx 2 \cdot N + 2 \cdot B. \tag{2.31}$$

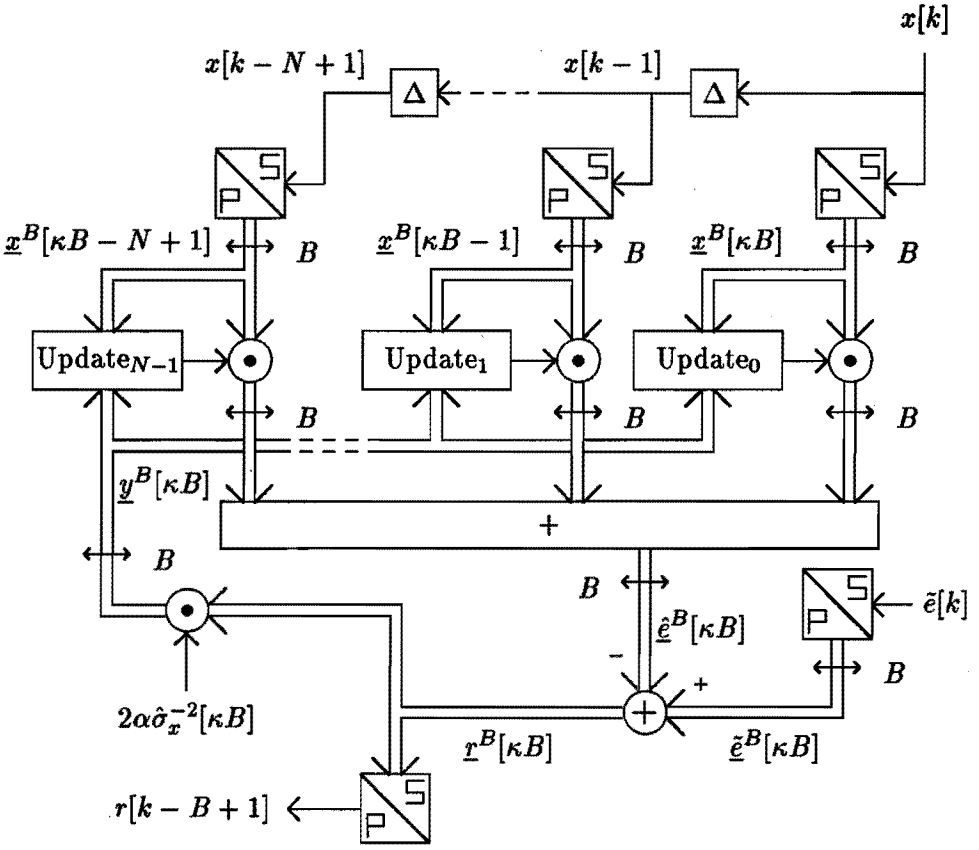



Figure 2.4: Block NLMS (BNLMS) algorithm.

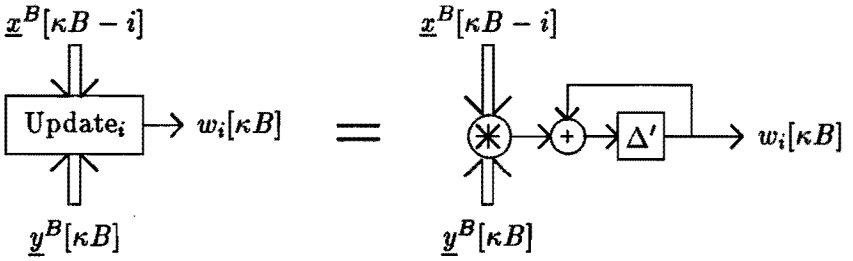



Figure 2.5: Update of BNLMS.

### 2.3.3 Convergence and Tracking

The difference between NLMS and BNLMS is that the latter combines more or less $B$ updates of the former without adapting the filter vector. This means that "older" filter vector values are used to produce an estimate of the unknown response. This implies that a $B$ times smaller maximum value for the adaptation constant $\alpha$ can be chosen (to ensure stability) (see [24]). A second disadvantage is the delay occuring in the weight vector adaptation, which degrades the quality of tracking of the algorithm. In average, information of the unknown response takes $B/2$ samples extra to be processed into the adaptive filter vector update (compared to (N)LMS), implying an extra misadjustment $\Upsilon_3$ of the adaptive filter (see also $\Upsilon_1$ and $\Upsilon_2$ in subsection 2.1.3). The average amount of change in the unknown impulse response during $B/2$ sample intervals defines the amount of extra misadjustment $\Upsilon_3$.

## 2.4 Recursive Least Squares (RLS)

### 2.4.1 RLS Algorithm

The convergence behaviour of the algorithms introduced so far can be degraded heavily by the input signal eigenvalue distribution. To minimize the influence of the input signal correlation, the update equation can be decorrelated using an estimate $\hat{\mathcal{R}}_x^N[k]$ of the $N \times N$ autocorrelation matrix $\mathcal{R}_x^N[k]$ of the input signal. This yields the LMS-Newton algorithm [57], given by

$$\underline{w}^N[k+1] = \underline{w}^N[k] + 2\alpha(\hat{\mathcal{R}}_x^N[k])^{-1}\underline{x}^N[k]r[k] \tag{2.32}$$

with

$$\mathcal{R}_x^N[k] = \mathcal{E}\{\underline{x}^N[k](\underline{x}^N[k])^t\}. \tag{2.33}$$

Implementation of the above equations directly in time domain requires a matrix inversion every sample, implying in the order of $N^3$ multiplications per sample. By using a weighting factor (or forgetting factor) in the mean squared error cost function and using an efficient inverse matrix update algorithm (with help of the matrix inversion lemma), exploiting the fact that the data shifts by only one sample, we obtain the Recursive Least Squares (RLS) algorithm. It still requires in the order of $N^2$ multiplications per sample. In literature other faster calculation methods have been introduced, like FTF [6, 42] (Fast Transversal Filter) and FAEST [5] (Fast

Aposteriori Error Sequential Technique). Stabilized versions of these "fast" methods still require at least $8N$ multiplications per sample.

### 2.4.2 Block RLS

Like in the NLMS case, we can construct a block computation version of the LMS-Newton algorithm by combination of $B$ updates. This Block RLS (BRLS) algorithm is given by[2]

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + 2\alpha(\hat{\mathcal{R}}_x^N[\kappa B])^{-1}\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B] \tag{2.34}$$

where $\hat{\mathcal{R}}_x^N[\kappa B]$ is an estimate of

$$\mathcal{R}_x^N[\kappa B] = \mathcal{E}\{\mathcal{X}^{N,B}[\kappa B](\mathcal{X}^{N,B}[\kappa B])^t\} \tag{2.35}$$

with the input signal matrix defined in equation (2.21), the residual signal in (2.25), and the filtering operation as in (2.24). Working with block based equations induces an algorithm delay of $B-1$ samples as in the BNLMS case, and will have influence on the tracking properties (see subsection 2.3.3).

### 2.4.3 Properties of RLS

As there are a lot of different adaptive filtering algorithms based on the RLS method (or an equivalent approach) we cannot speak about *the* properties. We can however mention some global measures.

- Algorithm processing delay (in most algorithms)

$$\Delta_{\text{RLS}} = 0. \tag{2.36}$$

- The computation processing delay equals the LMS computation processing delay as the filter part of RLS is just a convolution like in the LMS algorithm.

$$D_{\text{RLS}} < 1. \tag{2.37}$$

- Computational complexity (efficient implementation)

$$\Psi_{\text{RLS}} \geq 8 \cdot N. \tag{2.38}$$

[2]Note that this is in fact not a block version of RLS, but of LMS-Newton. We could however also derive faster calculation methods for this block-processing algorithm. In the rest of this thesis we will refer to this block version of LMS-Newton by BRLS.

- Memory occupation: a few (5 to 10) length $N$ vectors have to be stored in efficient implementations

$$5 \cdot N < \Theta_{\text{RLS}} < 10 \cdot N. \tag{2.39}$$

### 2.4.4 Convergence and Tracking

From literature it is known that in a stationary environment with a highly correlated input signal $x[k]$ the convergence behaviour of RLS is superiour to that of (B)(N)LMS [27] when the update parameters are optimized. Convergence behaviour is made independent of the input signal (auto-)correlation. In non-stationary environments however, LMS might sometimes even perform better caused by measurement and algorithm noise [22, 3, 27].

When the relevant input signal correlation vector length $B_\rho$, with $\rho_i[k] = \mathcal{E}\{x[k] \cdot x[k-i]\} \approx 0$ for $i > B_\rho$, is much smaller than the length $N$ of the adaptive filter, an unnessarily large amount of calculations is needed for the inversion of the $N \times N$ autocorrelation matrix. Reduction of this dimension is the subject of the following section.

## 2.5 Block Orthogonal Projection (BOP)

### 2.5.1 BOP Algorithm

In the RLS algorithm and its fast derivates the dimension of the input signal autocorrelation matrix used to decorrelate the update, equals the filter length $N$. As the input signal correlation length in general does not have anything in common with the length of the filter, we can try to reduce computational complexity by reducing the dimension of the inverse autocorrelation matrix. In this section a technique is given that can decorrelate an input signal of an adaptive filter with $N$ weights, by using a $B \times B$ autocorrelation matrix ($B \geq 1$).

In [35] a $B$–step Orthogonal Projection (OP) method is introduced. With this OP method a projection is made on an $B$ dimensional hyperplane. In general there are two variants to implement this idea

- The "sliding" procedure that uses in every iteration only one new input signal sample. This is the B–step Orthogonal Projection method (OP) as described in [35].

- The "block" approach that uses $B$ new samples and performs only one update of the adaptive weight vector every $B$ samples. This is the Block Orthogonal Projection (BOP) algorithm [25, 44].

The BOP algorithm is explained in some more detail, as it will be the basis for the introduction of the frequency domain algorithms in the following chapters.

In order to derive the BOP algorithm we will use a geometrical interpretation [35, 29]. We define the difference vector $\underline{d}^N[\kappa B]$ as the difference of the adaptive weight vector $\underline{w}^N[\kappa B]$ and the unknown system $\underline{h}^N[\kappa B]$, thus $\underline{d}^N[\kappa B] = \underline{h}^N[\kappa B] - \underline{w}^N[\kappa B]$. The geometrical procedure is that we first make a projection of $\underline{d}^N[\kappa B]$ on a $B$ dimensional hyper-plane, spanned by $B$ vectors $\underline{x}^N[\kappa B - B + 1]$ till $\underline{x}^N[\kappa B]$ in an $N$ dimensional hyper-space. This projection is made by interpreting the difference vector $\underline{d}^N[\kappa B]$ as a sum of two components, one perpendicular ($\underline{d}_\perp^N[\kappa B]$) and the other parallel ($\underline{d}_\parallel^N[\kappa B]$) to the hyper-plane, as depicted in figure 2.6.

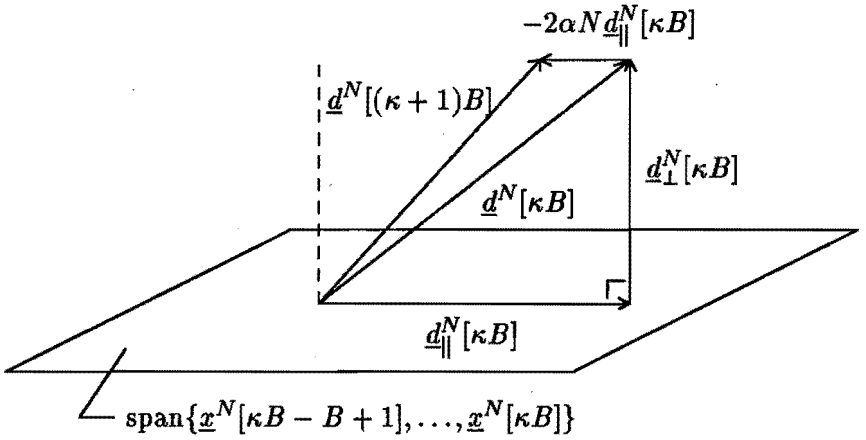



Figure 2.6: Geometric interpretation of BOP algorithm.

Mathematically this geometric interpretation described as

$$\underline{d}^N[\kappa B] = \underline{d}_\parallel^N[\kappa B] + \underline{d}_\perp^N[\kappa B] \tag{2.40}$$

where the perpendicular component $\underline{d}_\perp^N[\kappa B]$ of $\underline{d}^N[\kappa B]$ is pependicular to all $\underline{x}^N[\kappa B - i]$ for $0 \leq i < B$

$$(\underline{d}_\perp^N[\kappa B])^t \cdot \underline{x}^N[\kappa B - i] = 0 \tag{2.41}$$

and the parallel component $\underline{d}_{\|}^{N}[\kappa B]$ is lying in the hyper-plane

$$\begin{aligned} \underline{d}_{\|}^{N}[\kappa B] &= \sum_{i=0}^{B-1} a_i[\kappa B]\underline{x}^N[\kappa B - i] \\ &= \mathcal{X}^{N,B}[\kappa B]\underline{a}^B[\kappa B] \end{aligned} \tag{2.42}$$

with the $B$ dimensional vector $\underline{a}^B[\kappa B]$ defined as

$$\underline{a}^B[\kappa B] = \begin{pmatrix} a_{B-1}[\kappa B] & \cdots & a_0[\kappa B] \end{pmatrix}^t. \tag{2.43}$$

To obtain a simple derivation of the algorithm, we first assume $s[k] = 0$. Later on we will use the same algorithm for the case $s[k] \neq 0$.[3] Now the residual signal vector $\underline{r}^B[\kappa B]$ can be written as

$$\underline{r}^B[\kappa B] = (\mathcal{X}^{N,B}[\kappa B])^t \underline{d}^N[\kappa B]. \tag{2.44}$$

We have to find a procedure to calculate the coefficients of the vector $\underline{a}^B[\kappa B]$. This can be done by using the fact that $\underline{d}_{\perp}^{N}[\kappa B]$ is perpendicular to all vectors $\underline{x}^N[\kappa B - i]$ for $0 \leq i < B - 1$, so

$$\begin{aligned} \underline{r}^B[\kappa B] &= (\mathcal{X}^{N,B}[\kappa B])^t \underline{d}^N[\kappa B] \\ &= (\mathcal{X}^{N,B}[\kappa B])^t \underline{d}_{\|}^{N}[\kappa B] \\ &= (\mathcal{X}^{N,B}[\kappa B])^t \mathcal{X}^{N,B}[\kappa B]\underline{a}^B[\kappa B]. \end{aligned} \tag{2.45}$$

The solution of these equations is given by

$$\begin{aligned} \underline{a}^B[\kappa B] &= ((\mathcal{X}^{N,B}[\kappa B])^t \mathcal{X}^{N,B}[\kappa B])^{-1} \underline{r}^B[\kappa B] \\ &= \frac{1}{N}(\hat{\mathcal{R}}_x^B[\kappa B])^{-1} \underline{r}^B[\kappa B] \end{aligned} \tag{2.46}$$

where

$$\hat{\mathcal{R}}_x^B[\kappa B] = \frac{1}{N}(\mathcal{X}^{N,B}[\kappa B])^t \mathcal{X}^{N,B}[\kappa B] \tag{2.47}$$

is an estimate of the $B \times B$ autocorrelation matrix $\mathcal{R}_x^B[\kappa B]$.[4]

[3] In practice, this means that when $s[k]$ is (highly) correlated, adaption of the coefficients has to be frozen (in the previously introduced algorithms, the same problem is encountered). In chapter 8, where the implementation of an acoustic echo canceller is discussed, this problem occurs ("double talk").

[4] The matrix $(\mathcal{X}^{N,B}[\kappa B])^t \mathcal{X}^{N,B}[\kappa B]$ can only be inverted if it has full rank, which requires (at least) that $B \leq N$. By using an approximation of $\hat{\mathcal{R}}_x^B[\kappa B]$ this problem is avoided.

The update procedure "rotates" the vector $\underline{d}^N[\kappa B]$ in such a way that it becomes more orthogonal to the previous data, and reduces its length, by using the following update equation

$$\begin{aligned}\underline{d}^N[(\kappa+1)B] &= \underline{d}^N[\kappa B] - 2\alpha N \cdot \underline{d}^N_{||}[\kappa B] \\ &= \underline{d}^N[\kappa B] - 2\alpha \mathcal{X}^{N,B}[\kappa B](\hat{\mathcal{R}}^B_x[\kappa B])^{-1}\underline{r}^B[\kappa B]. \quad (2.48)\end{aligned}$$

This equation leads, with the definition of the difference vector, to the BOP update equation

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + 2\alpha \mathcal{X}^{N,B}[\kappa B](\hat{\mathcal{R}}^B_x[\kappa B])^{-1}\underline{r}^B[\kappa B]. \quad (2.49)$$

In figures 2.7 and 2.8 the whole BOP algorithm is depicted. Comparing this algorithm to the BNLMS and NLMS algorithms shows that the BOP algorithm is an extension of the BNLMS algorithm. The input signal is decorrelated with a $B \times B$ estimate $\hat{\mathcal{R}}^B_x[\kappa B]$ of the autocorrelation matrix $\mathcal{R}^B_x[\kappa B]$.

Implementation of the BOP equation directly in time domain requires a matrix inversion and multiplication every block, implying in the order of $N + B^2$ multiplications per sample. In literature, like in the RLS case, fast calculation methods have been introduced, based on a sliding version of the BOP algorithm. The FNTF (Fast Newton Transversal Filter) [32, 38] still requires at least $2N + 6B$ multiplications per sample.

### 2.5.2 Decoupled BOP

In the BOP algorithm the inverse autocorrelation matrix dimension is decoupled from the filter length $N$, but coupled to the block length $B$. By splitting a BOP algorithm with matrix order $Q$, into $Q/B$ parts this coupling can be removed.[5] Such a BOP algorithm with block length $Q$ and block index $\kappa'$ can be written as

$$\underline{w}^N[(\kappa'+1)Q] = \underline{w}^N[\kappa' Q] + 2\alpha \mathcal{X}^{N,Q}[\kappa' Q](\hat{\mathcal{R}}^Q_x[\kappa' Q])^{-1}\underline{r}^Q[\kappa' B]. \quad (2.50)$$

Now we try to reverse the order of the input signal matrix $\mathcal{X}^{N,Q}[\kappa' Q]$ and the (inverse) autocorrelation matrix estimate $(\hat{\mathcal{R}}^Q_x[\kappa' Q])^{-1}$. The autocorrelation matrix estimate can be written as

$$\hat{\mathcal{R}}^Q_x[\kappa' Q] = \frac{1}{N}(\mathcal{X}^{N,Q}[\kappa' Q])^t \mathcal{X}^{N,Q}[\kappa' Q]$$

[5] We assume for the moment that $N$ in an integer multiple of $Q$ and $Q$ is an integer multiple of $B$. When $N$ is not an integer multiple of $Q$, the adaptive filter vector can be extended with $\lceil N/Q \rceil \cdot Q - N$ coefficients that are kept zero.

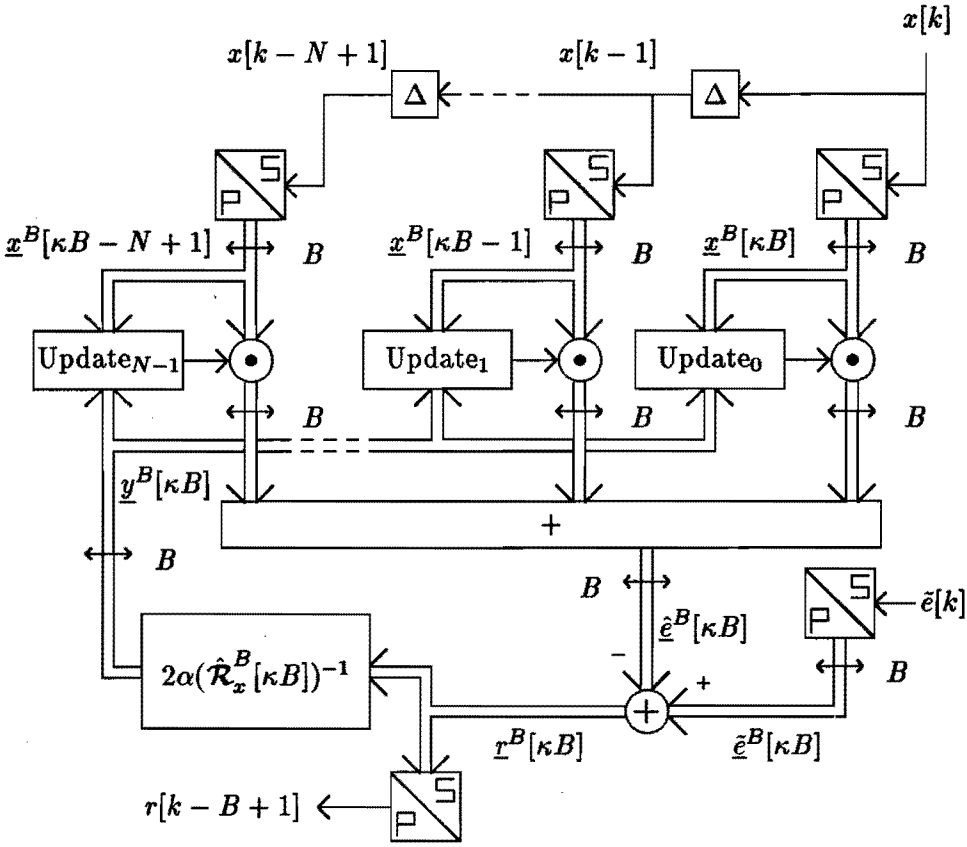



Figure 2.7: Block Orthogonal Projection (BOP) algorithm.

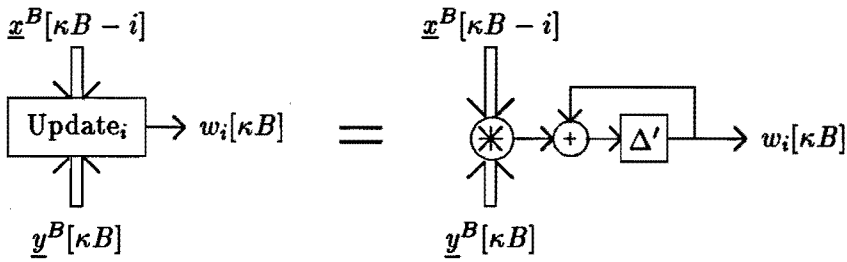



Figure 2.8: Update of BOP.

$$= \frac{1}{N}\sum_{i=0}^{\frac{N}{Q}-1}(\mathcal{X}^{Q,Q}[(\kappa'-i)Q])^t\mathcal{X}^{Q,Q}[(\kappa'-i)Q] \quad (2.51)$$

For large $Q$ the above can be approximated for $0 \le j < N/Q$ by[6]

$$\hat{\mathcal{R}}_x^Q[\kappa' Q] \approx \frac{1}{Q}(\mathcal{X}^{Q,Q}[(\kappa'-j)Q])^t\mathcal{X}^{Q,Q}[(\kappa'-j)Q] \quad (2.57)$$

implying that for $0 \le j < N/Q$

$$\mathcal{X}^{Q,Q}[(\kappa'-j)Q](\hat{\mathcal{R}}^Q[\kappa' Q])^{-1} \approx (\hat{\mathcal{R}}_x^Q[\kappa' Q])^{-1}\mathcal{X}^{Q,Q}[(\kappa'-j)Q]. \quad (2.58)$$

By using the above relation (2.58), we obtain

$$\begin{aligned}\mathcal{X}^{N,Q}[\kappa' Q](\hat{\mathcal{R}}_x^Q[\kappa' Q])^{-1} &= \begin{pmatrix} \mathcal{X}^{Q,Q}[(\kappa'-\frac{N}{Q}+1)Q]\cdot(\hat{\mathcal{R}}_x^Q[\kappa' Q])^{-1} \\ \vdots \\ \mathcal{X}^{Q,Q}[\kappa' Q]\cdot(\hat{\mathcal{R}}_x^Q[\kappa' Q])^{-1} \end{pmatrix} \\ &= \begin{pmatrix} (\hat{\mathcal{R}}_x^Q[\kappa' Q])^{-1}\cdot\mathcal{X}^{Q,Q}[(\kappa'-\frac{N}{Q}+1)Q] \\ \vdots \\ (\hat{\mathcal{R}}_x^Q[\kappa' Q])^{-1}\cdot\mathcal{X}^{Q,Q}[\kappa' Q] \end{pmatrix}\end{aligned}$$

[6] For a stationary signal $x[k]$, we note that for all $0 \le j < N/Q$

$$\mathcal{E}\{(\mathcal{X}^{Q,Q}[\kappa' Q])^t\mathcal{X}^{Q,Q}[\kappa' Q]\} = \mathcal{E}\{(\mathcal{X}^{Q,Q}[(\kappa'-j)Q])^t\mathcal{X}^{Q,Q}[(\kappa'-j)Q]\} \quad (2.52)$$

which implies that for all $0 \le j < N/Q$

$$\mathcal{E}\left\{\frac{1}{N}\sum_{i=0}^{\frac{N}{Q}-1}(\mathcal{X}^{Q,Q}[(\kappa'-i)Q])^t\mathcal{X}^{Q,Q}[(\kappa'-i)Q]\right\} \quad (2.53)$$

$$= \frac{1}{Q}\mathcal{E}\left\{(\mathcal{X}^{Q,Q}[(\kappa'-j)Q])^t\mathcal{X}^{Q,Q}[(\kappa'-j)Q]\right\}. \quad (2.54)$$

For large $Q$, the expectation approximates its momentaneous estimate

$$\mathcal{E}\left\{(\mathcal{X}^{Q,Q}[\kappa' Q])^t\mathcal{X}^{Q,Q}[\kappa' Q]\right\} \approx (\mathcal{X}^{Q,Q}[\kappa' Q])^t\mathcal{X}^{Q,Q}[\kappa' Q]. \quad (2.55)$$

This all implies that for $0 \le j < \frac{N}{Q}$

$$\frac{1}{N}\sum_{i=0}^{\frac{N}{Q}-1}(\mathcal{X}^{Q,Q}[(\kappa'-i)Q])^t\mathcal{X}^{Q,Q}[(\kappa'-i)Q] \approx \frac{1}{Q}(\mathcal{X}^{Q,Q}[(\kappa'-j)Q])^t\mathcal{X}^{Q,Q}[(\kappa'-j)Q]. \quad (2.56)$$

$$= (\hat{\mathcal{R}}^N_{x,Q}[\kappa' Q])^{-1} \mathcal{X}^{N,Q}[\kappa' Q] \tag{2.59}$$

with

$$(\hat{\mathcal{R}}^N_{x,Q}[\kappa' Q])^{-1} = \begin{pmatrix} (\hat{\mathcal{R}}^Q_x[\kappa' Q])^{-1} & \mathbf{0}^Q & \cdots & \mathbf{0}^Q \\ \mathbf{0}^Q & \ddots & & \vdots \\ \vdots & & \ddots & \mathbf{0}^Q \\ \mathbf{0}^Q & \cdots & \mathbf{0}^Q & (\hat{\mathcal{R}}^Q_x[\kappa' Q])^{-1} \end{pmatrix}. \tag{2.60}$$

Incorporating this in equation (2.50) we get

$$\underline{w}^N[(\kappa'+1)Q] = \underline{w}^N[\kappa' Q] + 2\alpha(\hat{\mathcal{R}}^N_{x,Q}[\kappa' Q])^{-1} \mathcal{X}^{N,Q}[\kappa' Q] \underline{r}^Q[(\kappa' Q]. \tag{2.61}$$

By defining

$$\underline{r}^Q[\kappa' Q] = \begin{pmatrix} \underline{r}^B[((\kappa' \frac{Q}{B} - \frac{Q}{B} + 1) \cdot B] \\ \vdots \\ \underline{r}^B[(\kappa' \frac{Q}{B} - 1) \cdot B] \\ \underline{r}^B[\kappa' \frac{Q}{B} \cdot B] \end{pmatrix} \tag{2.62}$$

equation (2.61) can be split into $Q/B$ parts as follows

$$\begin{aligned} \underline{w}^N[(\kappa'+1)Q] &= \underline{w}^N[\kappa' Q] + 2\alpha(\hat{\mathcal{R}}^N_{x,Q}[\kappa' Q])^{-1} \\ &\quad \cdot \sum_{i=0}^{\frac{Q}{B}-1} \mathcal{X}^{N,Q}[\kappa' Q] \cdot \begin{pmatrix} \underline{0}^{Q-B-iB} \\ \underline{r}^B[(\kappa' \frac{Q}{B} - i)B] \\ \underline{0}^{iB} \end{pmatrix}. \end{aligned} \tag{2.63}$$

Note that for all $0 \leq i < N/Q$

$$\begin{aligned} &(\mathcal{X}^{N,Q}[\kappa' Q] \cdot \begin{pmatrix} \underline{0}^{Q-B-iB} \\ \underline{r}^B[(\kappa' \frac{Q}{B} - i)B] \\ \underline{0}^{iB} \end{pmatrix} \\ &= (\mathcal{X}^{N,B}[(\kappa' \frac{Q}{B} - i)B] \cdot \underline{r}^B[(\kappa' \frac{Q}{B} - i)B] \end{aligned} \tag{2.64}$$

By using the above relation and performing an inverse "block computation transformation" on equation (2.63), we get the DBOP update formula

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + 2\alpha(\hat{\mathcal{R}}^N_{x,Q}[\kappa B])^{-1} \mathcal{X}^{N,B}[\kappa B] \underline{r}^B[\kappa B]. \tag{2.65}$$

In the first instance, the above equation seems very unattractive because of the large matrix multiplication involved. In chapter 4 we will introduce

an efficient implementation. If we compare equation (2.65) with the BRLS update equation (2.34), we see that the $N \times N$ autocorrelation matrix estimate $\hat{\mathcal{R}}_x^N[\kappa B]$ is appoximated by $\hat{\mathcal{R}}_{x,Q}^N[\kappa B]$. Assuming that BRLS achieves "perfect" decorrelation, a matrix $(\hat{\mathcal{R}}_{x,Q}^N[\kappa B])^{-1} \cdot \hat{\mathcal{R}}_x^N[\kappa B]$ remains. The eigenvalue distribution of this matrix defines the convergence behaviour of (D)BOP.

### 2.5.3 Properties of BOP

Block processing has some consequences for the features of the BOP algorithm:

- Extra algorithm processing delay of $B - 1$ samples

$$\Delta_{\text{BOP}} = B - 1. \tag{2.66}$$

  Fast versions, based on a sliding algorithm, have no algorithm delay.

- The computation processing delay for small $B$ equals the BNLMS computation processing delay

$$D_{\text{BOP}} < 1. \tag{2.67}$$

  In the FNTF the computation processing delay equals the LMS computation processing delay, like in the RLS (FTF) case.

- Computational complexity (efficient implementation)

$$\Psi_{\text{BOP}} \geq 2 \cdot N + 6 \cdot B. \tag{2.68}$$

- Memory occupation: a few (5 to 10) length $N$ vectors have to be stored in efficient implementations.

$$5 \cdot N < \Theta_{\text{BOP}} < 10 \cdot N. \tag{2.69}$$

### 2.5.4 Convergence and Tracking

The convergence properties of fast versions (based on sliding OP) will be inbetween those of NLMS and RLS, depending on the matrix dimension $B$, for BOP, or $Q$, for DBOP. In general, the choice of this matrix dimension will give good decorrelation properties if the input signal can be decorrelated by an autocorrelation matrix of order sufficiently smaller than

$B$ (or $Q$) [45, 12, 32, 38]. If the input signal is not white, there will always remain correlation between two adjacent blocks caused by the matrix $(\hat{\mathcal{R}}^N_{x,Q}[\kappa B])^{-1} \cdot \mathcal{R}^N_x[\kappa B]$. Increasing $Q$ will decrease this inter-block correlation. If this matrix dimension is sufficiently larger than the relevant length of the autocorrelation vector, the total inter block correlation (that remains) is much smaller than the amount of correlation within the blocks, that is removed by the (D)BOP algorithm.

If the latter is the case, then the (D)BOP algorithm behaves approximately as a BNLMS algorithm with white noise as input signal. Increasing the matrix dimension towards $N$ will then increase noise and computational complexity without improving convergence and tracking properties.

Increasing the matrix dimensions means that we need more sample intervals to make an equally accurate estimate of the current (inverse) autocorrelation matrix. When the input signal is stationary, this is no problem. If we are dealing with a highly correlated and non-stationary signal (such as speech) the decorrelation of the input signal can become worse for larger matrix dimensions. As less decorrelation, in most cases, reduces speed of convergence, tracking properties of the algorithm might even degrade when the matrix dimension is increased.

## 2.6 Example of Time Domain Algorithms

In our example we want to realize a length $N = 4000$ adaptive filter with a delay of 0.5 milli-seconds, at a sample rate of 8000 Hz. For (N)LMS and B(N)LMS this means that we have to perform $2 \cdot N = 8000$ multiplications per sample. With a sample rate of 8000 Hz, this means $64 \cdot 10^6$ multiplications per second. This makes realization on one (or even a few) DSPs with the current state of technique impossible. Besides that, both algorithms suffer from very bad convergence behaviour for coloroured input signals like speech. Fast implementations of RLS and BOP do require an even larger amount of multiplications per second (from slightly more than $64 \cdot 10^6$ for FNTF to al least $256 \cdot 10^6$ for FTF). In figure 2.9 the complexity, delay and memory of the diverse time domain algorithms are depicted as a function of the block length $B$. For sliding algorithms (such as LMS, FTF and FNTF) where parameters do not depend on the block length, processing delay is always 1 (see appendix A) and is not depicted.

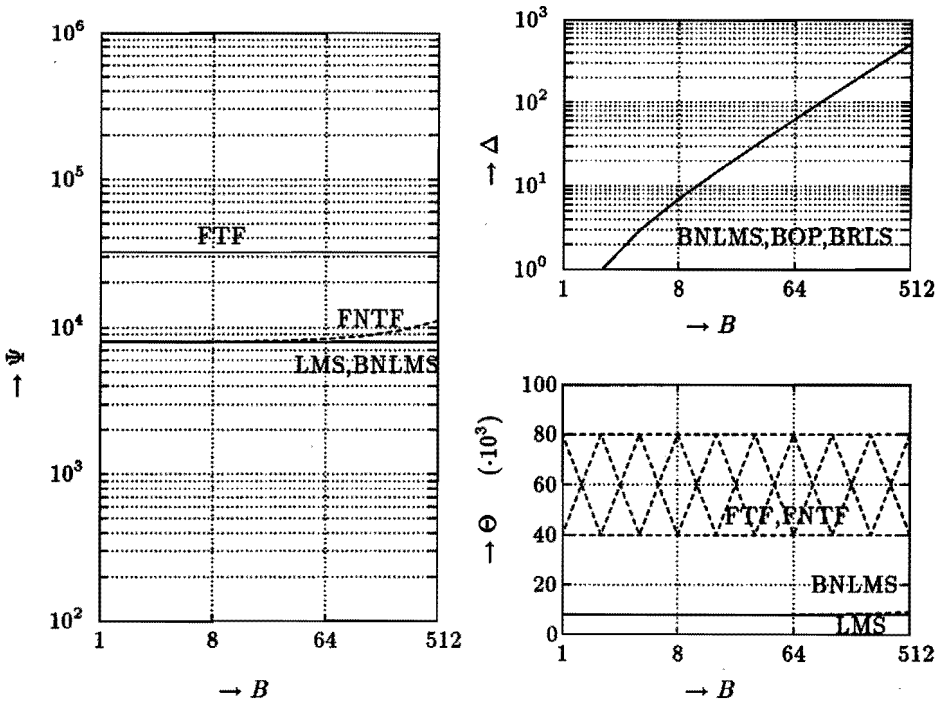



Figure 2.9: Properties of time domain algorithms.

## 2.7 Conclusions

For small adaptive filters, the RLS algorithm (or reduced complexity versions of RLS) might be a good solution. The huge computational complexity of all time domain algorithms makes implementation of large adaptive filters on one (or even a few) DSPs impossible. Besides that, the simplest algorithms suffer from very bad convergence behaviour for highly coloured input signals. We therefore have to look at ways to reduce computational complexity in the (D)BOP (or (B)RLS) algorithms. This can be done with the help of fast implementations of Discrete Fourier Transforms, as will be shown in the next chapters.

# Chapter 3

# Block Frequency Domain Adaptive Filtering

*The two main operations to implement an adaptive algorithm are a (linear) convolution, to perform the filtering of the input signal with the adaptive weights, and a (linear) correlation, to calculate an estimate of the gradient that is needed for the update of the adaptive weights. For large filter lengths $N$ these operations can be carried out efficiently in frequency domain by using Fast Fourier Transforms (FFTs) for the transformation between time- and frequency-domain [34]. Overlap-save is a well known technique to convolve an infinite length input sequence (e.g. the input signal samples $x[k]$) with a finite length impulse response (e.g. the $N$ adaptive weights $w_i$).*

*An efficient implementation of the overlap-save method, is used to realize the BNLMS algorithm in frequency domain, resulting in the Frequency domain BNLMS (FBNLMS) algorithm. Like in time domain, we also like to improve convergence behaviour in frequency domain. The BRLS and the BOP algorithm of the previous chapter can be implemented in frequency domain by making some small changes in the FBNLMS algorithm and using some approximations. The result is a generalized version of the Block Frequency Domain Adaptive Filter (BFDAF).*

## 3.1 Overlap Save Method

### 3.1.1 Introduction

In this section we will review the overlap-save method for fixed filters, therefore we assume that the weight vector $\underline{w}^N$ is fixed for the moment. As

described before, using a block processing approach implies that each step a block of $B$ output samples is calculated ($B \geq 1$). These output samples are collected in the length $B$ vector $\underline{\hat{e}}^B[\kappa B]$,[1] described by

$$\underline{\hat{e}}^B[\kappa B] = (\mathcal{X}^{N,B}[\kappa B])^t \underline{w}^N \tag{3.1}$$

where

$$\begin{aligned} \mathcal{X}^{N,B}[\kappa B] &= \left( \underline{x}^N[\kappa B - B + 1] \ \cdots \ \underline{x}^N[\kappa B] \right) && (3.2) \\ \underline{x}^N[\kappa B] &= \left( x[\kappa B - N + 1] \ \cdots \ x[\kappa B] \right)^t && (3.3) \\ \underline{w}^N &= \left( w_{N-1} \ \cdots w_0 \right)^t . && (3.4) \end{aligned}$$

With the overlap-save method it is possible to implement the above given linear convolution efficiently in frequency domain. It uses a length $M$ segment of the input signal $x[k]$ to obtain a partial convolution of that input signal and a length $N$ weight vector $\underline{w}^N$. With $M \geq N - 1 + B$ the procedure generates each step $B$ new output samples in a vector $\underline{\hat{e}}^B[\kappa B]$.[2] This method can be implemented with Discrete Fourier Transforms (DFTs) (or a fast version of DFTs: Fast Fourier Transforms (FFTs)).

### 3.1.2 Diagonalization with DFTs

The main problem of the calculations in equation (3.1) is that a matrix and a vector have to be multiplied. With help of DFTs this matrix can be transformed to a diagonal matrix (diagonalization) [9]. The DFT of a length $M$ vector $\underline{x}^M[\kappa B]$ is defined for all $l$, $0 \leq l < M$ as

$$\underline{X}^M[\kappa B] = \mathcal{F}^M \cdot \underline{x}^M[\kappa B] \tag{3.6}$$

with the l'th element $(\underline{X}^M[\kappa B])_l$ of $\underline{X}^M[\kappa B]$ given by

$$(\underline{X}^M[\kappa B])_l = \sum_{i=0}^{M-1} x[\kappa B - M + 1 + i] \cdot e^{-j2\pi \frac{i \cdot l}{M}} . \tag{3.7}$$

[1]The block index $\kappa$ is an integer, thus $\kappa B$ denotes a time instance that is a multiple of $B$.

[2]Note that the matrix $\mathcal{X}^{N,B}[\kappa B]$ consists of the samples $x[\kappa B - N - B + 2]$ till $x[\kappa B]$. It can thus be constructed using a length $N + B - 1$ segment of the input signal. We assume the use of a length $M$ segment in order to choose a DFT length later on that is a power of two (for efficient computation purposes) by taking

$$M = 2^{\lceil \log_2(N+B-1) \rceil} . \tag{3.5}$$

The length $M$ input signal vector $\underline{x}^M[\kappa B]$ contains the last $M$ input signal samples

$$\underline{x}^M[\kappa B] = \begin{pmatrix} x[\kappa B - M + 1] \\ \vdots \\ x[\kappa B] \end{pmatrix}. \tag{3.8}$$

The elements of the $M \times M$ Fourier matrix $\mathcal{F}^M$ are defined for all $0 \leq a < M$ and $0 \leq b < M$ as

$$(\mathcal{F}^M)_{a,b} = e^{-\jmath 2\pi \frac{a \cdot b}{M}}. \tag{3.9}$$

The inverse transformation (the Inverse DFT (IDFT)) then is defined as

$$\begin{aligned} \underline{x}^M[\kappa B] &= (\mathcal{F}^M)^{-1} \cdot \underline{X}^M[\kappa B] \\ &= \frac{1}{M}(\mathcal{F}^M)^* \cdot \underline{X}^M[\kappa B] \end{aligned} \tag{3.10}$$

with

$$\begin{aligned} (\underline{x}^M[\kappa B])_l &= x[\kappa B - M + 1 + l] \\ &= \frac{1}{M} \sum_{i=0}^{M-1} (\underline{X}[\kappa B])_i \cdot e^{+\jmath 2\pi \frac{l \cdot i}{M}}. \end{aligned} \tag{3.11}$$

Three properties of these Fourier-matrices and transforms are of special interest for our purpose:

1. Orthogonality:

$$\begin{aligned} (\mathcal{F}^M)^{-1} \cdot \mathcal{F}^M &= \frac{1}{M}(\mathcal{F}^M)^* \cdot \mathcal{F}^M \\ &= \mathbf{I}^M. \end{aligned} \tag{3.12}$$

2. Shift invariancy: for all integers $b$, with $0 \leq b < M$ and all integers $d$:

$$\sum_{a=0}^{M-1} c_a \cdot e^{-\jmath 2\pi \frac{a \cdot b}{M}} = \sum_{a=0}^{M-1} c_{\lfloor \frac{a-d}{M} \rfloor M} \cdot e^{-\jmath 2\pi \frac{(a-d) \cdot b}{M}} \tag{3.13}$$

3. For all integers $a$, with $0 < a < M$:[3]

$$\sum_{b=0}^{M-1} e^{-\jmath 2\pi \cdot \frac{b \cdot a}{M}} = 0. \tag{3.14}$$

For $a = 0$ the above sum equals $M$.

[3]This property follows directly from equation 3.12 (by substituting equation (3.9)).

With these properties circulant matrices can be diagonalized [9]. A circulant $M \times M$ matrix $\breve{\mathcal{C}}^M$ is defined for all $0 \leq a < M$ and $0 \leq b < M$ by

$$(\breve{\mathcal{C}}^M)_{a,b} = c_{\lfloor \frac{a-b}{M} \rfloor M} \tag{3.15}$$

$$\mathcal{C}^M = \begin{pmatrix} c_0 & c_{M-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & \cdots & c_3 & c_2 \\ \vdots & \vdots & & \vdots & \vdots \\ c_{M-1} & c_{M-2} & \cdots & c_1 & c_0 \end{pmatrix}. \tag{3.16}$$

We will now show that pre- and post-multiplication of this circulant matrix $\breve{\mathcal{C}}^M$ by the Fourier matrix $\mathcal{F}^M$ and its inverse $(\mathcal{F}^M)^{-1}$ yields a diagonal matrix $\mathcal{F}^M \cdot \breve{\mathcal{C}}^M \cdot (\mathcal{F}^M)^{-1}$. Knowing that

$$(\mathcal{F}^M \cdot \mathcal{C}^M)_{a,b} = \sum_{i=0}^{M-1} c_{\lfloor \frac{i-b}{M} \rfloor M} \cdot e^{-\jmath 2\pi \frac{a \cdot i}{M}}, \tag{3.17}$$

applying the shift invariance properties and replacing $i - b$ by $g$ gives

$$\begin{aligned} & (\mathcal{F}^M \cdot \mathcal{C}^M \cdot (\mathcal{F}^M)^{-1})_{a,d} \\ = & \sum_{b=0}^{M-1} (\mathcal{F}^M \cdot \mathcal{C}^M)_{a,b} \cdot ((\mathcal{F}^M)^{-1})_{b,d} \\ = & \frac{1}{M} \sum_{b=0}^{M-1} \sum_{i=0}^{M-1} c_{\lfloor \frac{i-b}{M} \rfloor M} \cdot e^{-\jmath 2\pi \frac{a \cdot i}{M}} \cdot e^{+\jmath 2\pi \frac{b \cdot d}{M}} \\ = & \frac{1}{M} \sum_{g=0}^{M-1} c_g \cdot e^{-\jmath 2\pi \frac{g \cdot a}{M}} \cdot \sum_{b=0}^{M-1} e^{+\jmath 2\pi \cdot \frac{b \cdot (d-a)}{M}}. \end{aligned} \tag{3.18}$$

The third property of circulant matrices says that the sum on the right hand side of the above equation equals zero if $d \neq a$, which implies that all non-diagonal elements of the matrix $\mathcal{F}^M \cdot \mathcal{C}^M \cdot (\mathcal{F}^M)^{-1}$ equal zero, thus

$$\mathcal{F}^M \cdot \mathcal{C}^M \cdot (\mathcal{F}^M)^{-1} = \mathrm{diag}\{\underline{C}^M\} \tag{3.19}$$

where $\underline{C}^M$ is the Fourier transform of the first column of $\mathcal{C}^M$

$$\underline{C}^M = \sum_{g=0}^{M-1} c_g \cdot e^{-\jmath 2\pi \frac{g \cdot a}{M}}$$

$$= \mathcal{F}^M \cdot \begin{pmatrix} c_0 \\ \vdots \\ c_{M-1} \end{pmatrix}. \tag{3.20}$$

### 3.1.3 Block Frequency Domain Convolution

We can use the diagonalization technique if we can transform the matrix $\mathcal{X}^{N,B}[\kappa B]$ to a circulant matrix. Starting with equation (3.1) we get

$$\begin{aligned} \underline{\hat{e}}^B[\kappa B] &= (\mathcal{X}^{N,B}[\kappa B])^t \cdot \underline{w}^N \\ &= (\mathcal{X}^{N,B}[\kappa B])^t \cdot \mathbf{J}^N \cdot \mathbf{J}^N \cdot \underline{w}^N \end{aligned} \tag{3.21}$$

with the $N \times N$ mirror matrix $\mathbf{J}^N$ defined as

$$\mathbf{J}^N = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ \vdots & & 1 & 0 \\ 0 & 1 & & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}. \tag{3.22}$$

The matrix $(\mathcal{X}^{N,B}[\kappa B])^t \cdot \mathbf{J}^N$, given by

$$\begin{aligned} &(\mathcal{X}^{N,B}[\kappa B])^t \cdot \mathbf{J}^N \\ &= \begin{pmatrix} x[\kappa B - B + 1] & \cdots & x[\kappa B - B - N + 2] \\ \vdots & & \vdots \\ x[\kappa B] & \cdots & x[\kappa B - N + 1] \end{pmatrix}, \end{aligned} \tag{3.23}$$

can now easily by extended to a circulant matrix, and used in the convolution of equation (3.21). If we define a circulant extension $\breve{\mathcal{X}}^M[\kappa B]$ of $(\mathcal{X}^{N,B}[\kappa B])^t \cdot J^N$ as

$$\begin{aligned} \breve{\mathcal{X}}^M[\kappa B] &= \begin{pmatrix} \uparrow & \nearrow \\ (\mathcal{X}^{N,B}[\kappa B])^t \cdot \mathbf{J}^N & \rightarrow \end{pmatrix} \\ &= \begin{pmatrix} x[\kappa B - M + 1] & x[\kappa B] & \cdots & x[\kappa B - M + 2] \\ \vdots & & & \vdots \\ \vdots & & & x[\kappa B] \\ x[\kappa B] & \cdots & \cdots & x[\kappa B - M + 1] \end{pmatrix} \end{aligned} \tag{3.24}$$

and an extension $\breve{\underline{w}}^M$ of $\mathbf{J}^N \cdot \underline{w}^N$ by

$$\begin{aligned} \breve{\underline{w}}^M &= \begin{pmatrix} \mathbf{J}^N \cdot \underline{w}^N \\ \underline{0}^{M-N} \end{pmatrix} \\ &= \begin{pmatrix} w_0 & \cdots & w_{N-1} & (\underline{0}^{M-N})^t \end{pmatrix}^t \end{aligned} \tag{3.25}$$

then the result $\underline{\hat{e}}^B[\kappa B]$ of the convolution is given by

$$\underline{\hat{e}}^B[\kappa B] = \begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix} \cdot \check{\mathcal{X}}^M[\kappa B] \cdot \underline{\check{w}}^M \tag{3.26}$$

Now the DFTs can be applied to equation (3.26)

$$\begin{aligned} \underline{\hat{e}}^B[\kappa B] &= \begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix} (\mathcal{F}^M)^{-1} \mathcal{F}^M \check{\mathcal{X}}^M[\kappa B] (\mathcal{F}^M)^{-1} \mathcal{F}^M \underline{\check{w}}^M \\ &= \begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix} (\mathcal{F}^M)^{-1} \mathrm{diag}\{\underline{X}^M[\kappa B]\} \cdot \underline{W}^M \\ &= \begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix} (\mathcal{F}^M)^{-1} (\underline{X}^M[\kappa B] \otimes \underline{W}^M) \end{aligned} \tag{3.27}$$

with $\otimes$ the elementwise multiplication. As $\check{\mathcal{X}}^M[\kappa B]$ is a circulant matrix, $\mathcal{F}^M \check{\mathcal{X}}^M[\kappa B](\mathcal{F}^M)^{-1} = \mathrm{diag}\{\underline{X}^M[\kappa B]\}$ is a diagonal matrix, with

$$\underline{X}^M[\kappa B] = \mathcal{F}^M \cdot \begin{pmatrix} x[\kappa B - M + 1] \\ \vdots \\ x[\kappa B] \end{pmatrix}. \tag{3.28}$$

The transformation of the filter vector $\underline{w}^N$ is given by

$$\begin{aligned} \underline{W}^M &= \mathcal{F}^M \underline{\check{w}}^M \\ &= \mathcal{F}^M \cdot \begin{pmatrix} w_0 \\ \vdots \\ w_{N-1} \\ \underline{0}^{M-N} \end{pmatrix}. \end{aligned} \tag{3.29}$$

The whole procedure for fixed filters is depicted in figure 3.1. Each iteration $B$ new consecutive input signal samples $x[k]$ are collected ("S/P"). These collected samples are put into overlapping length $M$ vectors $\underline{x}^M[\kappa B]$ ("Overlap"), that have an overlap of $M-B$ samples with the input vector of the previous block. The vector $\underline{x}^M[\kappa B]$ is transformed to frequency domain ("DFT$_M$"), resulting in $\underline{X}^M[\kappa B]$. Furthermore the length $N$ weight vector $\underline{w}^N$ is first mirrored ("$\mathbf{J}^N$") and then augmented with zeroes ("$\underline{0}^{M-N}$") to a vector of length $M$. The resulting length $M$ vector is transformed to frequency domain. The (cyclic) convolution is carried out in frequency domain by an elementwise multiplication ("$\otimes$") of the two vectors $\underline{X}^M[\kappa B]$ and $\underline{W}^M$. Finally the result $\underline{X}^M[\kappa B] \otimes \underline{W}^M$ is transformed back to time domain by an inverse Fourier transform ("IDFT$_M$"). Only the last $B$ out of $M$ samples from this cyclic convolution represent the desired linear convolution result. Thus $M - B$ samples have to be discarded, resulting in

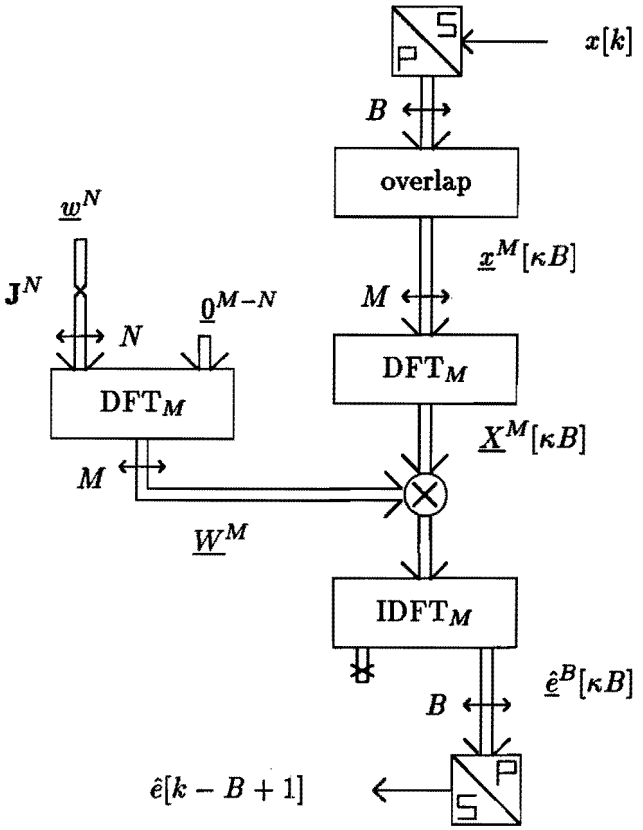



Figure 3.1: Block Frequency Domain Convolution.

the length $B$ vector $\underline{\hat{e}}^B[\kappa B]$. The original sample rate is obtained by desegmenting ("P/S") this vector into samples $\hat{e}[\kappa B - B + 1]$ to $\hat{e}[\kappa B]$.

The above described Block Frequency Domain Convolution (BFDC) takes one DFT and one IDFT for fixed filters. The DFT for the weight vector is superfluous when the weight vector $\underline{w}^N$ is constant. When the overlap-save technique is applied in adaptive filtering this DFT however is necessary, which means that 3 transforms are needed then.

### 3.1.4 Properties of BFDC

With the help of appendix A the next features of the Block Frequency Domain Convolution can be derived:

- Algorithm processing delay

$$\Delta_{\text{BFDC}} = B - 1. \tag{3.30}$$

- Computation processing delay for FFT, IFFT and elementwise multiplication

$$D_{\text{BFDC}} = 2 \cdot D_{\text{FFT}}\{M\} + D_{\otimes}\{M\}. \tag{3.31}$$

- Computational complexity: computation of FFT, IFFT and elementwise multiplication

$$\Psi_{\text{BFDC}} = \frac{2 \cdot \Psi_{\text{FFT}}\{M\} + \Psi_{\otimes}\{M\}}{B}. \tag{3.32}$$

- Memory occupation:
  - Input delayline (length $M$) and output samples (length $B$)
  - FFT input/output (length $M$)
  - Twiddle factors (elements of $\mathcal{F}^M$) (number $< M$)
  - Weight vector $\underline{W}^M$ (length $M$)

$$\Theta_{\text{BFDC}} \approx 4 \cdot M + B. \tag{3.33}$$

### 3.1.5 Example of BFDC

Our example illustrates the relation between delay and complexity. In table 3.1 and figure 3.2 the BFDC complexity, delay and memory are given as a function of the block length $B$ for a filter length $N = 4000$. The sudden step in $\Theta_{\text{BFDC}}$, $D_{\text{BFDC}}$ and $\Psi_{\text{BFDC}}$ is caused by the doubling of the FFT length

$M$. As our example requires a filter length of (at least) 4000 coefficients and the FFT length is a power of two (see appendix A), we use $M = 4096$ for $B \leq 64$ and $M = 8192$ for $B \geq 128$.

| $B$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $M$ | 4096 | 4096 | 4096 | 4096 | 4096 | 4096 | 4096 | 8192 | 8192 | 8192 |
| $\Delta$ | 0 | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |
| $D$ | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 176 | 176 | 176 |
| $\Psi$ $(\cdot 10^3)$ | 81.9 | 41.0 | 20.5 | 10.2 | 5.12 | 2.56 | 1.28 | 1.41 | 0.704 | 0.352 |
| $\Theta$ $(\cdot 10^3)$ | 16.4 | 16.4 | 16.4 | 16.4 | 16.4 | 16.4 | 16.4 | 32.9 | 33.0 | 33.3 |

Table 3.1: Delay, complexity and memory of BFDC.

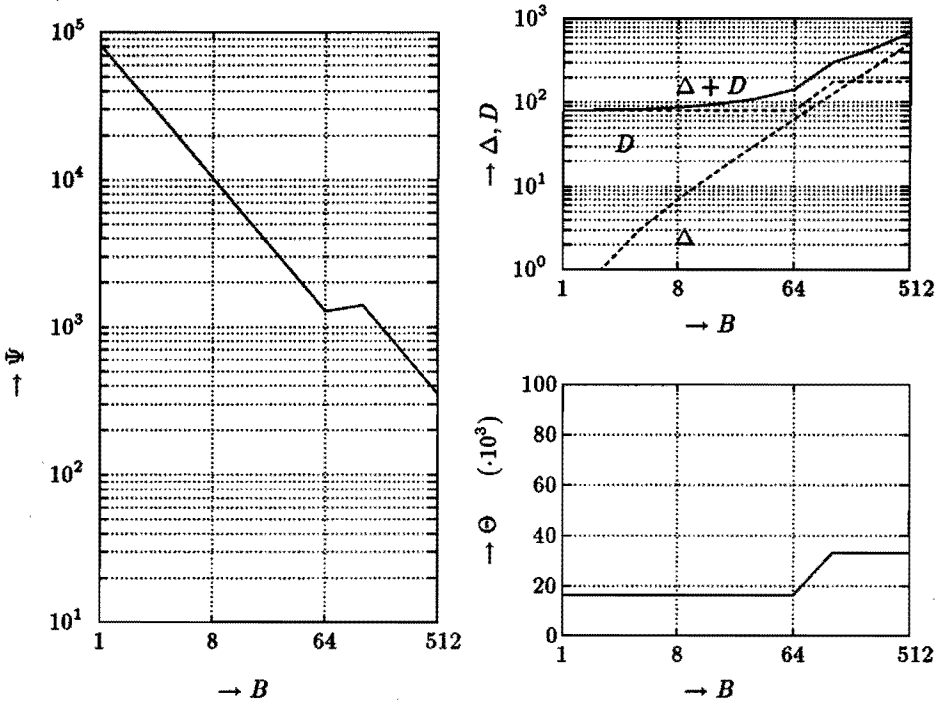



Figure 3.2: Properties of BFDC.

Because of the large FFTs needed (even for small $B$), the implementation always results in large computation processing delay. From the previous chapter we know that a time domain convolution requires 4000 multi-

plications per sample. For a block length $B > 20$, the complexity $\Psi_{\mathrm{BFDC}}$ is smaller than the time domain convolution complexity.

## 3.2 Frequency Domain BNLMS

### 3.2.1 FBNLMS Algorithm

The Frequency domain BNLMS (FBNLMS) is an efficient implementation for the BNLMS algorithm of chapter 2. The BNLMS algorithm consists of a linear convolution, to perform the filtering of the input signal with the adaptive weights, and a linear correlation, to calculate the gradient estimation that is needed for the update of the adaptive weights. The convolution is implemented as a BFDC of the previous section, while now, of course, we have to perform an extra FFT once every $B$ samples in order to get the Fourier transform of the weight vector.

The BNLMS update equation (2.26) that has to be transformed to frequency domain, contains the correlation $\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B]$. First we have to generate a circulant matrix by extending the matrix $\mathcal{X}^{N,B}[\kappa B]$, like in equation (3.24). The correlation then can be expressed as

$$\begin{aligned} &\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B] \\ &= \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} \begin{pmatrix} \leftarrow & \mathbf{J}^N \cdot \mathcal{X}^{N,B}[\kappa B] \\ \swarrow & \downarrow \end{pmatrix} \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}. \end{aligned} \tag{3.34}$$

Note that the above circulant input signal matrix is the exact transpose of the matrix $\check{\mathcal{X}}^M[\kappa B]$ in equation (3.24), thus

$$\begin{aligned} &\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B] \\ &= \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\mathcal{F}^M)^{-1}\mathcal{F}^M(\check{\mathcal{X}}^M[\kappa B])^t(\mathcal{F}^M)^{-1}\mathcal{F}^M \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\mathcal{F}^M)^{-1}(\mathcal{F}^M\check{\mathcal{X}}^M[\kappa B](\mathcal{F}^M)^{-1})^h \cdot \underline{R}^M[\kappa B] \\ &= \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\mathcal{F}^M)^{-1}((\underline{X}^M[\kappa B])^* \otimes \underline{R}^M[\kappa B]) \end{aligned} \tag{3.35}$$

where $\underline{X}^M[\kappa B]$ is defined in equation (3.6) and

$$\underline{R}^M[\kappa B] = \mathcal{F}^M \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}. \tag{3.36}$$

The correlation in the BNLMS update equation (2.26) can be substituted with help of equation (3.35), which results in the FBNLMS update

equation

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + \frac{2\alpha}{\sigma_x^2[\kappa B]} \left( \mathbf{J}^N \quad \mathbf{0}^{N,M-N} \right) \cdot(\mathcal{F}^M)^{-1}((\underline{X}^M[\kappa B])^* \otimes \underline{R}^M[\kappa B]). \tag{3.37}$$

The FBNLMS algorithm is depicted in figure 3.3. The residual signal

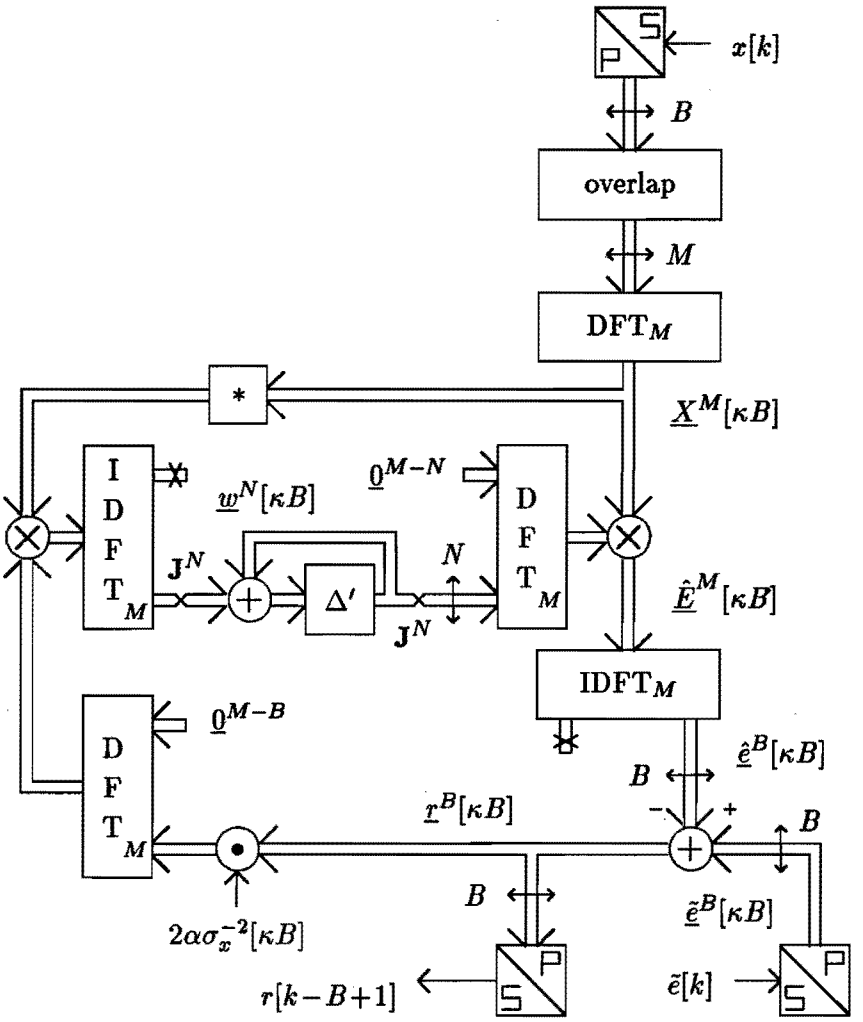



Figure 3.3: Frequency domain BNLMS.

$r[k]$ is obtained by putting the signal $\tilde{e}[k]$ in a length $B$ vector $\underline{\tilde{e}}^B[\kappa B]$ ("S/P"). The length $B$ residual signal vector is then given by $\underline{r}^B[\kappa B] =$

$\underline{\tilde{e}}^B[\kappa B] - \underline{\hat{e}}^B[\kappa B]$. The return to the original sample rate is carried out by desegmenting the residual signal vector ("P/S"). This results in the delayed residual signal $r[k - B + 1]$.

For the update part first the residual vector $\underline{r}^B[\kappa B]$ is multiplied by $2\alpha/\sigma_x^2[\kappa B]$ before it is augmented with zeroes ("$\underline{0}^{M-B}$") to a length $M$ vector,[4] and this vector is transformed to frequency domain ("DFT$_M$"). Note that the main difference between the correlation and convolution operation is reflected in frequency domain in the fact that correlation needs a complex conjugate operation ("*") of the input signal vector $\underline{X}^M[\kappa B]$, and the convolution does not.

Like in the BNLMS algorithm of equation (2.26), the normalization is carried out in such a way that the the number of multiplications is minimized. For this each element of $\underline{r}^B[\kappa B]$ is first multiplied by the adaptation constant $2\alpha\sigma_x^{-2}[\kappa B]$. The two mirror operations $\mathbf{J}^N$ in figure 3.3, one before and one after the update of the adaptive weight vector, can easily be combined and left out. In order to keep the separate implementations of the convolution and correlation operations visible, this has not been done.

### 3.2.2 Properties of FBNLMS

With help of appendix A the next features of the Frequency domain BNLMS algorithm can be derived:

- Algorithm processing delay

$$\Delta_{\text{FBNLMS}} = B - 1. \tag{3.38}$$

- Computation processing delay for FFT, IFFT and elementwise multiplication

$$D_{\text{FBNLMS}} = 2 \cdot D_{\text{FFT}}\{M\} + D_{\otimes}\{M\}. \tag{3.39}$$

- Computational complexity: computation of FFTs, elementwise multiplications and normalization

$$\Psi_{\text{FBNLMS}} = \frac{5 \cdot \Psi_{\text{FFT}}\{M\} + 2 \cdot \Psi_{\otimes}\{M\} + \Psi_{\sigma}\{B\}}{B}. \tag{3.40}$$

- Memory occupation:

[4] To minimize the number of multiplications (when $B < N$), this multiplication is performed there (requiring $B$ multiplications), instead of after the IDFT (which would require $N$ multiplications).

- Input delayline (length $M$) and output samples (length $B$)
- FFT input/output (length $M$)
- Twiddle factors (elements of $\mathcal{F}^M$) (number $< M$)
- Weight vector $\underline{W}^M$ (length $M$)
- Normalization (length $B$)

$$\Theta_{\text{FBNLMS}} < 4 \cdot M + 2 \cdot B. \tag{3.41}$$

### 3.2.3 Convergence and Tracking

As the FBNLMS algorithm is an exact transformation of the BNLMS algorithm, the dynamic behaviour of both the adaptive filtering algorithms is identical (see subsection 2.3.3). The FBNLMS update equation (3.37) can be written in frequency domain using equation (3.29) as

$$\begin{aligned} \underline{W}^M[(\kappa+1)B] &= \underline{W}^M[\kappa B] + \frac{2\alpha}{\sigma_x^2[\kappa B]} \mathcal{F}^M \cdot \mathbf{V}_N^M \cdot (\mathcal{F}^M)^{-1} \cdot \\ &\quad ((\underline{X}^M[\kappa B])^* \otimes \underline{R}^M[\kappa B]) \end{aligned} \tag{3.42}$$

with the window matrix $\mathbf{V}_N^M$ given by

$$\begin{aligned} \mathbf{V}_N^M &= \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{J}^N \\ \mathbf{0}^{M-N,N} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \\ \mathbf{0}^{M-N,N} & \mathbf{0}^{M-N} \end{pmatrix}. \end{aligned} \tag{3.43}$$

If we leave out this windowing matrix $\mathbf{V}_N^M$ [31, 46], a DFT and an IDFT become superfluous, and can be removed, leading to the following Unconstrained FBNLMS update equation that uses only three (I)DFTs (instead of five)

$$\underline{W}^M[(\kappa+1)B] = \underline{W}^M[\kappa B] + \frac{2\alpha}{\sigma_x^2[\kappa B]}((\underline{X}^M[\kappa B])^* \otimes \underline{R}^M[\kappa B]). \tag{3.44}$$

Because the circular behaviour of the frequency domain convolution is no longer cancelled, there is a coupling between the part of the coefficient update that we desire (the first $N$ elements of $(\underline{X}^M[\kappa B])^* \otimes \underline{R}^M[\kappa B]$) and the part we would like to skip (the last $M-N$ elements of $(\underline{X}^M[\kappa B])^* \otimes \underline{R}^M[\kappa B]$). As long as the length of the unknown system $\underline{h}$ is smaller than $N$ [31] the only effect is that $M-N$ more weights are fluctuating around zero,

which increases the final misadjustment by a factor $(M - N)/N$ without any effect on the rate of convergence [46].

When the unknown system impulse reponse is longer than $N$, the unconstrained algorithm no longer converges to the Wiener solution [31], which can even lead to divergence of the algorithm.

## 3.3 BFDAF Algorithm

### 3.3.1 Filter Part

Using an equivalent approach as in subsection 3.2.1 , we will describe here an efficient way to perform decorrelation by the inverse input signal autocorrelation matrix, in frequency domain, resulting in the Block Frequency Domain Adaptive Filter (BFDAF). As a basis for transformation to the frequency domain, one out of two time domain algorithms, the BOP and BRLS algorithm, will be used, depending on the choice of the block length $B$ and the filter length $N$. As the filter part of both algorithms is equivalent to the filter part of the BNLMS algorithm, also the filter parts of FBNLMS and BFDAF are equal (see equations (3.1), (3.21), (3.27) and figure 3.1).

The update equation (2.49) of the BOP algorithm and the update equation (2.34) of the BRLS algorithm can also be implemented efficiently in frequency domain. When the block length $B$ is larger than the filter length $N$, BOP is used. On the other hand, if the filter length $N$ is larger than the block length $B$, BRLS is used as basis. Both approaches lead to the same algorithm, with the use of some approximations. In chapter 6 we will show that if the block length $B$ is larger than the filter length $N$, the BOP algorithm is indeed the best basis (leading to the smallest approximation error). On the other hand, when $B$ is smaller than $N$, the BRLS is the best choice.

### 3.3.2 BOP as Basis

The BOP update equation (2.49) is rewritten as follows

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + \mathcal{X}^{N,B}[\kappa B]\underline{y}^B[\kappa B] \tag{3.45}$$

with

$$\underline{y}^B[\kappa B] = 2\alpha(\hat{\mathcal{R}}_x^B[\kappa B])^{-1}\underline{r}^B[\kappa B]. \tag{3.46}$$

The implementation of $\mathcal{X}^{N,B}[\kappa B]\underline{y}^B[\kappa B]$ is equivalent to equation (3.35) with the only difference that $\underline{R}^M[\kappa B]$ must be replaced by the vector

$\underline{Y}^M[\kappa B]$. The frequency domain transform $\underline{Y}^M[\kappa B]$ of $\underline{y}^B[\kappa B]$ is defined as

$$\begin{aligned}\underline{Y}^M[\kappa B] &= \mathcal{F}^M \begin{pmatrix} \underline{0}^{M-B} \\ \underline{y}^B[\kappa B] \end{pmatrix} \\ &= \mathcal{F}^M \begin{pmatrix} \underline{0}^{M-B} \\ 2\alpha(\hat{\mathcal{R}}_x^B[\kappa B])^{-1}\underline{r}^B[\kappa B] \end{pmatrix}. \end{aligned} \tag{3.47}$$

For large block lengths $B$ this is a very unattractive equation to compute in time domain, as the matrix inversion $(\hat{\mathcal{R}}_x^B[\kappa B])^{-1}$ needs in the order of $B^3$ multiplications. With the use of some approximations however, we can transform the inverse autocorrelation matrix $(\hat{\mathcal{R}}_x^B[\kappa B])^{-1}$ to a circulant $M \times M$ matrix $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ (whose inverse is also circulant) [9]). This procedure is described in chapter 6 and results in

$$\underline{Y}^M[\kappa B] \approx 2\alpha\mathcal{F}^M(\check{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix} \tag{3.48}$$

with

$$\check{\mathcal{R}}_x^M[\kappa B] = \frac{1}{M}\mathcal{E}\{\check{\mathcal{X}}^M[\kappa B](\check{\mathcal{X}}^M[\kappa B])^t\}. \tag{3.49}$$

This circulant matrix $\check{\mathcal{R}}_x^M[\kappa B]$ can be diagonalized with the Fourier matrix $\mathcal{F}^M$ as follows

$$\begin{aligned}\mathrm{diag}\{\underline{P}_x^M[\kappa B]\} &= \mathcal{F}^M\check{\mathcal{R}}^M[\kappa B](\mathcal{F}^M)^{-1} \\ &= \frac{1}{M}\mathcal{E}\{\mathcal{F}^M\check{\mathcal{X}}^M[\kappa B](\check{\mathcal{X}}^M[\kappa B])^t(\mathcal{F}^M)^{-1}\}\end{aligned} \tag{3.50}$$

with the input signal power vector (see also chapter 6)

$$\begin{aligned}\underline{P}_x^M[\kappa B] &= \frac{1}{M}\mathcal{E}\{(\underline{X}^M[\kappa B])^* \otimes \underline{X}^M[\kappa B]\} \\ &= \frac{1}{M}\mathcal{E}\{|\underline{X}^M[\kappa B]|^2\}.\end{aligned} \tag{3.51}$$

Combining equations (3.45) to (3.50) and using an estimate $(\hat{\underline{P}}_x^M[\kappa B])^{-1}$ of $(\underline{P}_x^M[\kappa B])^{-1}$ results in the following BFDAF update formula

$$\begin{aligned}\underline{w}^N[(\kappa+1)B] &= \underline{w}^N[\kappa B] + 2\alpha \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\mathcal{F}^M)^{-1} \\ &\quad \cdot((\underline{X}^M[\kappa B])^* \otimes (\hat{\underline{P}}_x^M[\kappa B])^{-1} \otimes \underline{R}^M[\kappa B]).\end{aligned} \tag{3.52}$$

$(\underline{P}_x^M[\kappa B])^{-1}$ denotes elementwise inverse of $\underline{P}_x^M[\kappa B]$, as

$$(\underline{P}_x^M[\kappa B])^{-1} = \mathrm{diag}\{(\mathrm{diag}\{\underline{P}_x^M[\kappa B]\})^{-1}\} \tag{3.53}$$

implying that for $0 \leq i < M$

$$((\underline{P}_x^M[\kappa B])^{-1})_i = ((\underline{P}_x^M[\kappa B])_i)^{-1}. \tag{3.54}$$

In chapter 6 several methods for calculating an estimate $(\hat{\underline{P}}_x^M[\kappa B])^{-1}$ of the inverse power vector $(\underline{P}_x^M[\kappa B])^{-1}$ directly from the input signal frequency domain transform $\underline{X}^M[\kappa B]$ are given. The result of the above described efficient implementation of (an approximation of) the BOP algorithm in frequency domain is called the Block Frequency Domain Adaptive Filter (BFDAF) [8], from which the implementation is depicted in figure 3.4.

### 3.3.3 BRLS as Basis

The same result as in the previous section can be obtained by using the BRLS update equation (2.34) as basis, given by

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + 2\alpha(\hat{\mathcal{R}}_x^N[\kappa B])^{-1}\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B]. \tag{3.55}$$

To obtain an efficient realization, also here approximations to circulant matrices will be used. In chapter 6 it is shown that the above can be approximated by

$$\begin{aligned}\underline{w}^N[(\kappa+1)B] &= \underline{w}^N[\kappa B] + 2\alpha \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{R}}^M[\kappa B])^{-1} \\ &\quad \cdot(\check{\mathcal{X}}^M[\kappa B])^t \cdot \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}.\end{aligned} \tag{3.56}$$

Since the order in a product of circulant matrices may be interchanged, this equation is equal to the one obtained with BOP as basis, so the above can be transformed to the BFDAF update equation (3.52).

### 3.3.4 Properties of BFDAF

With appendix A the next features of the Block Frequency Domain Adaptive Filter can be derived:

- Algorithm processing delay

$$\Delta_{\mathrm{BFDAF}} = B - 1. \tag{3.57}$$

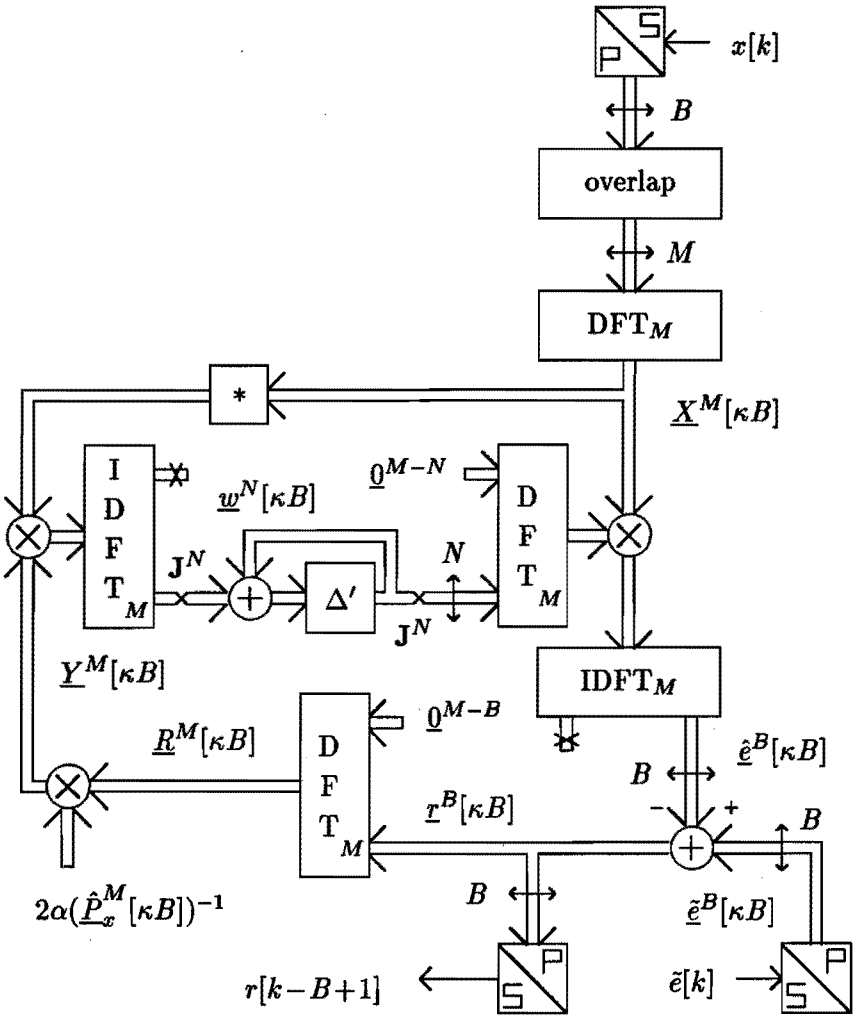



Figure 3.4: BFDAF algorithm.

- Computation processing delay for FFT, IFFT and elementwise multiplication

$$D_{\text{BFDAF}} = 2 \cdot D_{\text{FFT}}\{M\} + D_{\otimes}\{M\}. \tag{3.58}$$

- Computational complexity: computation of FFTs, elementwise multiplications and decorrelation (see appendix A)

$$\Psi_{\text{BFDAF}} = \frac{5 \cdot \Psi_{\text{FFT}}\{M\} + 2 \cdot \Psi_{\otimes}\{M\} + \Psi_{P}\{M\}}{B}. \tag{3.59}$$

- Memory occupation:
  - Input delayline (length $M$) and output samples (length $B$)
  - FFT input/output (length $M$)
  - Twiddle factors (elements of $\mathcal{F}^M$) (number $< M$)
  - Weight vector $\underline{W}^M$ (length $M$)
  - Decorrelation (length $M/2$)

$$\Theta_{\text{BFDAF}} \approx \frac{9}{2} \cdot M + B. \tag{3.60}$$

### 3.3.5 Convergence and Tracking

In a stationairy environment, the normalization makes the algorithm behave as an FBNLMS (and thus as a BNLMS) algorithm with a white noise input signal, provided that the approximation error is sufficiently small. This means that the convergence behaviour of the RLS algorithm can be obtained by BFDAF. As we strive towards algorithms with low processing delay for large adaptive filters, thus assuming that $N \gg B$, the approximation error can be neglected (see chapter 6, subsections 2.4.4 and 2.5.4 and [46]). If $N \gg B$, then the BFDAF algorithm is an (approximated) transformation of BRLS, implying that the difference between RLS and BFDAF merely consists of the consequences of block-based computation.

Compared to RLS, tracking will suffer from the same problems as BNLMS compared to LMS. These problems, caused by the block processing technique, are described in more detail in subsection 2.3.3. Like in the FBNLMS case of subsection 3.2.3, also here an unconstrained version can be obtained by skipping the window (and two Fourier transforms). This Unconstrained BFDAF [31] has the same problems as the Unconstrained FBNLMS.

## 3.4 Example of BFDAF and FBNLMS

Our example illustrates the relation between delay and complexity. In table 3.2 and figure 3.5 the FBNLMS complexity, delay and memory are given, while in table 3.3 and figure 3.6 the BFDAF properties are shown as a function of the block length $B$ for a filter length $N = 4000$. Also here the sudden step in $\Theta$, $\Delta$ and $\Psi$ is caused by the doubling of the FFT length $M$ from 4096 (for $B \leq 64$) to 8192 (for $B \geq 128$). Compared to the time

| $B$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $M$ | 4096 | 4096 | 4096 | 4096 | 4096 | 4096 | 4096 | 8192 | 8192 | 8192 |
| $\Delta$ | 0 | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |
| $D$ | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 176 | 176 | 176 |
| $\Psi$ $(\cdot 10^3)$ | 201 | 100 | 50.2 | 25.1 | 12.5 | 6.27 | 3.14 | 3.46 | 1.73 | 0.865 |
| $\Theta$ $(\cdot 10^3)$ | 16.4 | 16.4 | 16.4 | 16.4 | 16.4 | 16.5 | 16.5 | 33.0 | 33.3 | 33.8 |

Table 3.2: Delay, complexity and memory of FBNLMS.

| $B$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $M$ | 4096 | 4096 | 4096 | 4096 | 4096 | 4096 | 4096 | 8192 | 8192 | 8192 |
| $\Delta$ | 0 | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |
| $D$ | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 176 | 176 | 176 |
| $\Psi$ $(\cdot 10^3)$ | 217 | 109 | 54.2 | 27.1 | 13.6 | 6.78 | 3.39 | 3.71 | 1.86 | 0.930 |
| $\Theta$ $(\cdot 10^3)$ | 18.4 | 18.4 | 18.4 | 18.4 | 18.4 | 18.5 | 18.5 | 37.0 | 37.1 | 37.4 |

Table 3.3: Delay, complexity and memory of BFDAF.

domain adaptive algorithms of the previous chapter, that required at least 8000 multiplications per sample, the BFDAF has a smaller computational complexity for a block length $B > 27$.

## 3.5 Conclusions

For large $N$ and $B$ it is known from literature [46, 30] that the BFDAF has good convergence properties for relative low computational complexity. However, when the processing delay is limited, low complexity can not be reached any more with the BFDAF approach. Besides that, the large FFTs imply a large computation processing delay, so if a small processing delay is required, BFDAF is not the right solution. These are the main

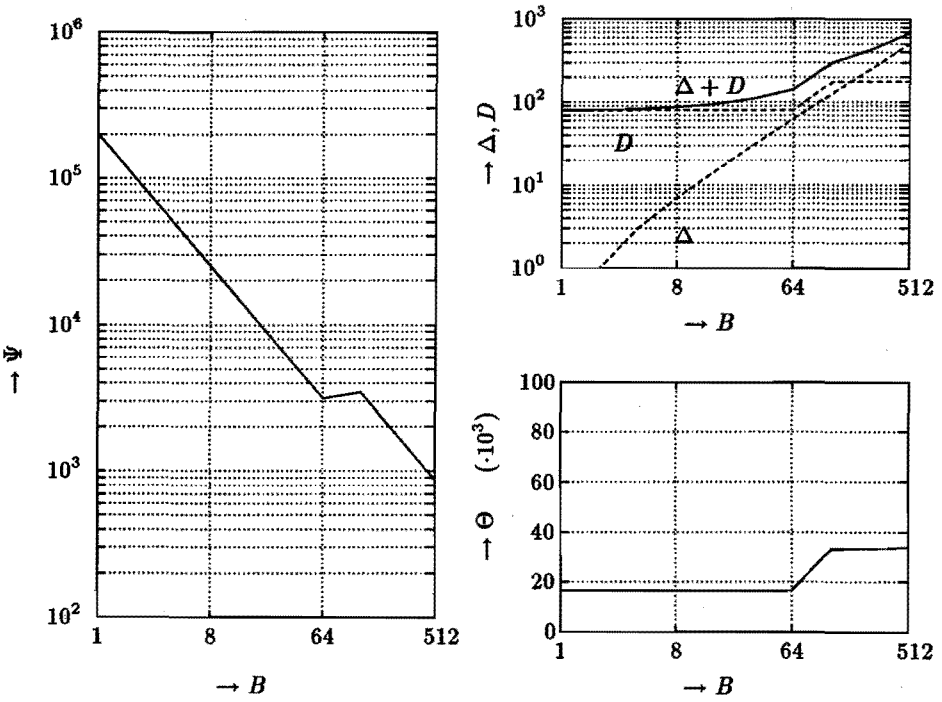



Figure 3.5: Properties of FBNLMS.

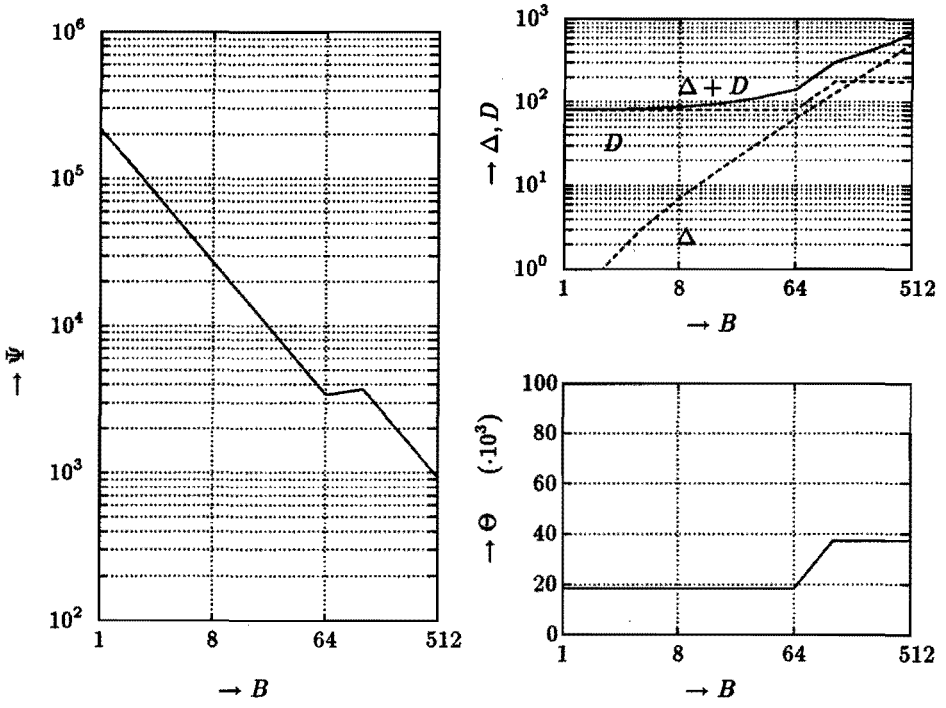



Figure 3.6: Properties of BFDAF.

motivations to derive partitioned structures, that will be treated in the following chapter.

# Chapter 4

# (Decoupled) Partitioned BFDAF

*For the realization of large adaptive filters the Block Frequency Domain Adaptive Filter (BFDAF), as described in the previous chapter, combines a relative low implementation complexity with good performance. On the other hand however the resulting processing delay of this algorithm is very large. For many practical situations (e.g. an acoustic echo canceller) this is unacceptable. On one hand a large filter length $N$ is needed, while, on the other hand, only a very small processing delay can be tolerated. In the BFDAF algorithm the processing delay can only be reduced by using a small block length $B$, but this increases the computational complexity in such a way that the approach becomes very inefficient. Besides that, the computation processing delay can not be reduced to an acceptable level, as the FFTs involved always have a length not smaller than the filter length $N$.*

*By partitioning of the filter vector into smaller vectors, a small block length can be used, without a huge penalty in the computational complexity. This results in the Partitioned BFDAF (PBFDAF). Using smaller vectors and block lengths however also results in smaller Fourier transforms in the update part of the algorithm, thus implying a smaller decorrelation vector.*

*By using a different partitioning in filter and update part of the algorithm, we can make an independent choice for both the length of the decorrelation vector and the block length in the filter part of the algorithm. This results in the Decoupled PBFDAF (DPBFDAF). By choosing larger Fourier transforms in the update part the computational complexity compared to PBFDAF can also be reduced.*

## 4.1 Partitioning of Convolution

### 4.1.1 Partitioning in Time Domain

The block processing approach produces a length $B$ vector $\underline{\hat{e}}^B[\kappa B]$ of output samples, that consists of $B$ convolutions of $N$ input signal samples with a filter impulse response of length $N$. This filter can be partitioned into smaller subfilters of length $Q$. Defining that $g_F = \lceil N/Q \rceil$,[1] this partitioning in time domain can be described as follows

$$\begin{aligned} \underline{\hat{e}}^B[\kappa B] &= (\mathcal{X}^{N,B}[\kappa B])^t \cdot \underline{w}^N \\ &= (\mathcal{X}^{g_F \cdot Q,B}[\kappa B])^t \cdot \underline{w}^{g_F \cdot Q} \\ &= \sum_{i=0}^{g_F-1} (\mathcal{X}^{Q,B}[\kappa B - iQ])^t \cdot \underline{w}_i^Q \end{aligned} \tag{4.2}$$

in which the partitioning of the input-signal matrix is defined as

$$\mathcal{X}^{g_F \cdot Q,B}[\kappa B] = \begin{pmatrix} \mathcal{X}^{Q,B}[\kappa B - (g_F - 1)Q] \\ \vdots \\ \mathcal{X}^{Q,B}[\kappa B] \end{pmatrix} \tag{4.3}$$

$$\mathcal{X}^{Q,B}[\kappa B] = \begin{pmatrix} \underline{x}^Q[\kappa B - B + 1] & \cdots & \underline{x}^Q[\kappa B] \end{pmatrix} \tag{4.4}$$

$$\underline{x}^Q[\kappa B] = \begin{pmatrix} x[\kappa B - Q + 1] & \cdots & x[\kappa B] \end{pmatrix}^t \tag{4.5}$$

The (fixed) filter coefficients are partitioned in an equivalent way, resulting in

$$\underline{w}^{g_F \cdot Q} = \begin{pmatrix} \underline{w}_{g_F-1}^Q \\ \vdots \\ \underline{w}_0^Q \end{pmatrix} \tag{4.6}$$

with for $0 \leq i < g_F$

$$\underline{w}_i^Q = \begin{pmatrix} w_{iQ+Q-1} & \cdots & w_{iQ} \end{pmatrix}^t. \tag{4.7}$$

This partitioning of the filter vector is depicted in figure 4.1.

[1] When $N/Q$ is not an integer, the filter vector $\underline{w}^N$ is extended to a length $\lceil N/Q \rceil \cdot Q = g_F \cdot Q$ vector by appending $Q \cdot \lceil N/Q \rceil - N$ coefficients equal to zero

$$\begin{aligned} \underline{w}^{g_F \cdot Q} &= \begin{pmatrix} \underline{0}^{g_F \cdot Q - N} \\ \underline{w}^N \end{pmatrix} \\ &= \begin{pmatrix} w_{g_F \cdot Q - 1} & \cdots & w_0 \end{pmatrix}^t. \end{aligned} \tag{4.1}$$

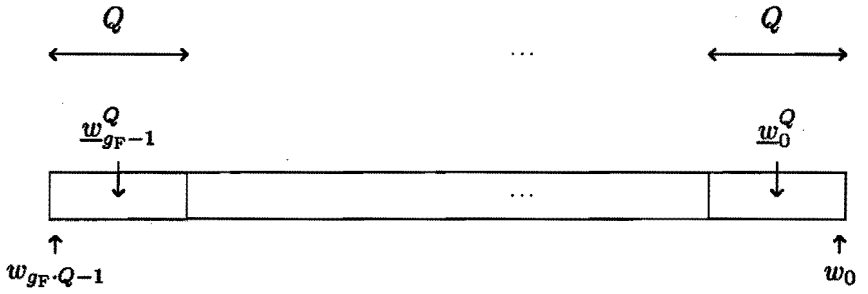



Figure 4.1: Partitioning of filter vector.

### 4.1.2 Partitioned BFDC

The $g_F$ convolutions of equation (4.2) can now be computed separately by using an overlap-save method in frequency domain, as described in subsection 3.1.3. By choosing a DFT length $M \geq Q + B - 1$ the resulting Partitioned Block Frequency Domain Convolution (PBFDC) can be described as

$$\underline{\hat{E}}^M[\kappa B] = \sum_{i=0}^{g_F-1} \underline{X}_i^M[\kappa B] \otimes \underline{W}_i^M \tag{4.8}$$

$$\underline{\hat{e}}^B[\kappa B] = \begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix} \cdot (\mathcal{F}^M)^{-1} \cdot \underline{\hat{E}}^M[\kappa B] \tag{4.9}$$

where for $0 \leq i < g_F$

$$\underline{X}_i^M[\kappa B] = \mathcal{F}^M \cdot \underline{x}^M[\kappa B - iQ] \tag{4.10}$$

$$\underline{W}_i^M = \mathcal{F}^M \begin{pmatrix} \mathbf{J}^Q \cdot \underline{w}_i^Q \\ \underline{0}^{M-Q} \end{pmatrix} \tag{4.11}$$

If we assume that $Q/B = q$ is an integer, a simple delay line in frequency domain can be used for the vectors $\underline{X}_i^M[\kappa B]$ with $i > 0$,[2] since then for $0 < i < g_F$

$$\underline{X}_i^M[\kappa B] = \underline{X}_{i-1}^M[(\kappa - q)B]. \tag{4.13}$$

The realization of equation (4.9) for integer $Q/B$ is depicted in figure 4.2. In this figure each box "$q\Delta$'"represents $q$ one sample delay elements in

[2]When $Q/B$ is an integer, implying that the greatest common divisor of $Q$ and $B$, $\gcd\{Q, B\}$, equals $B$, only one FFT is needed. Otherwise we still can use a delayline, but we need $B/\gcd\{Q, B\}$ FFTs (per $B$ samples), by using (for $B/\gcd\{Q, B\} \leq i < g_F$)

$$\underline{X}_i^M[\kappa B] = \underline{X}_{i-B/\gcd\{Q,B\}}^M[(\kappa - Q/\gcd\{Q, B\})B]. \tag{4.12}$$

In the examples considered in this thesis, $Q/B$ is an integer.

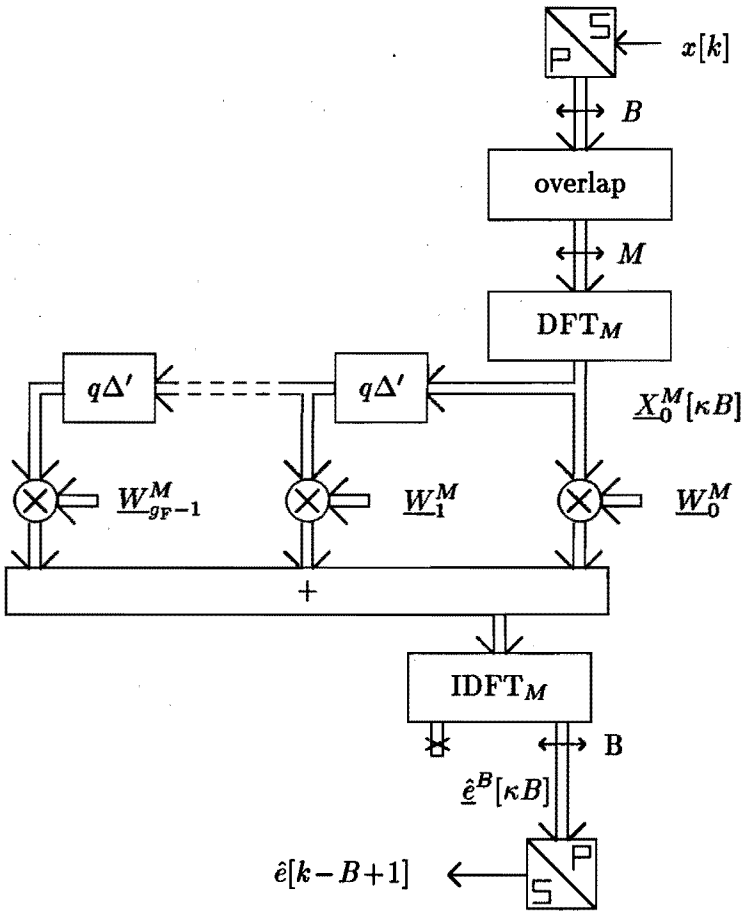



Figure 4.2: Partitioned BFDC.

series, each delaying over $B \cdot \mathrm{T}$ seconds. The effect of this partitioning is that a part of the convolution is carried out in frequency domain, and a part in time domain. The factor $Q$ can be chosen to minimize computational complexity, keeping in mind that increasing $Q$ will increase $M$. A larger FFT length implies a larger computation processing delay.

### 4.1.3 Properties of PBFDC

With help of appendix A the next features of the Partitioned BFDC can be derived:

- Algorithm processing delay

$$\Delta_{\mathrm{PBFDC}} = B - 1. \tag{4.14}$$

- Computation processing delay for FFT, IFFT and elementwise multiplication

$$D_{\mathrm{PBFDC}} = 2 \cdot D_{\mathrm{FFT}}\{M\} + D_{\otimes}\{M\}. \tag{4.15}$$

- Computational complexity: computation of FFT, IFFT and elementwise multiplications

$$\Psi_{\mathrm{PBFDC}} = \frac{2 \cdot \Psi_{\mathrm{FFT}}\{M\} + g_{\mathrm{F}} \cdot \Psi_{\otimes}\{M\}}{B}. \tag{4.16}$$

- Memory occupation:
  - Input delayline (length $M$), $g_{\mathrm{F}} - 1$ times $q$ vector delays (width $M$) and output samples (length $B$)
  - FFT input/output (length $M$)
  - Twiddle factors (elements of $\mathcal{F}^M$) (number $< M$)
  - Weight vectors $\underline{W}_i^M$ ($g_{\mathrm{F}}$ times length $M$)

$$\Theta_{\mathrm{PBFDC}} \approx (3 + (g_{\mathrm{F}} - 1) \cdot (q + 1)) \cdot M + B. \tag{4.17}$$

If $M$ is much smaller than $N$,[3] this can be approximated by

$$\begin{aligned} \Theta_{\mathrm{PBFDC}} &\approx (2 + \frac{N}{B} + \frac{N}{Q} - \frac{Q}{B}) \cdot (B + Q) + B \\ &\approx (2 + q + \frac{1}{q}) \cdot N. \end{aligned} \tag{4.18}$$

[3]Normally, $M$ indeed is much smaller than $N$, as we try to realize large filters with small block lengths (and FFT lenghts).

### 4.1.4 Example of PBFDC

Our example illustrates the relation between delay and complexity. In table 4.1 and figure 4.3 the PBFDC complexity, delay and memory are given as a function of the block length $B$ for a filter length $N = 4000$.[4] The partition length $Q$ is chosen to minimize the computational complexity.

| $B$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $Q$ | 1 | 30 | 60 | 56 | 112 | 96 | 192 | 384 | 256 | 512 |
| $q$ | 1 | 15 | 15 | 7 | 7 | 3 | 3 | 3 | 1 | 1 |
| $M$ | 1 | 32 | 64 | 64 | 128 | 128 | 256 | 512 | 512 | 1024 |
| $g_F$ | 4000 | 134 | 67 | 72 | 36 | 42 | 21 | 11 | 16 | 8 |
| $\Delta$ | 0 | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |
| $D$ | 1 | 1 | 1 | 1 | 2 | 2 | 4 | 8 | 8 | 18 |
| $\Psi$ | 4000 | 4222 | 2209 | 1183 | 636 | 366 | 207 | 136 | 88 | 60 |
| $\Theta$ $(\cdot 10^3)$ | 8.0 | 68.7 | 68.7 | 37.0 | 37.1 | 21.8 | 22.1 | 23.7 | 17.7 | 18.9 |

Table 4.1: Delay and complexity and memory of PBFDC.

Comparing these results to a time domain convolution, requiring 4000 multplications per sample, we see that for a block length $B > 2$, the complexity $\Psi_{\text{PBFDC}}$ is smaller than 4000, which makes this PBFDC an atractive way of implementing convolution and correlation in adaptive filtering algorithms. We observe that the amount of memory $\Theta_{\text{PBFDC}}$ depends strongly on the block length $B$ in this case. This is caused by the fact that we chose to minimize the computational complexity, which implies that for decreasing $B$, the factor $q = Q/B$ increases. Equation (4.18) shows that a mimimum for $\Theta_{\text{PBFDC}}$ is obtained when $q = 1$.

## 4.2 Partitioning of the Update Part

### 4.2.1 Starting Points in Time Domain

The PBFDC can be used in adaptive filtering to perform the convolution. For the update part of an adaptive filtering algorithm we would also like to reduce computational complexity. Therefore we partition the update part into smaller parts, as was done with the filter part in the previous section. For the update part, we will use a block length $A$, a partition length $Z$, an FFT length $L$ and a block index $l$.[5] In the BFDAF update

[4] For $B = 1$, the TDC is the "optimal" PFDC. Of course, no FFT is carried out then!

[5] As the block index is an integer, the time instance $lA$ is a multiple of $A$.

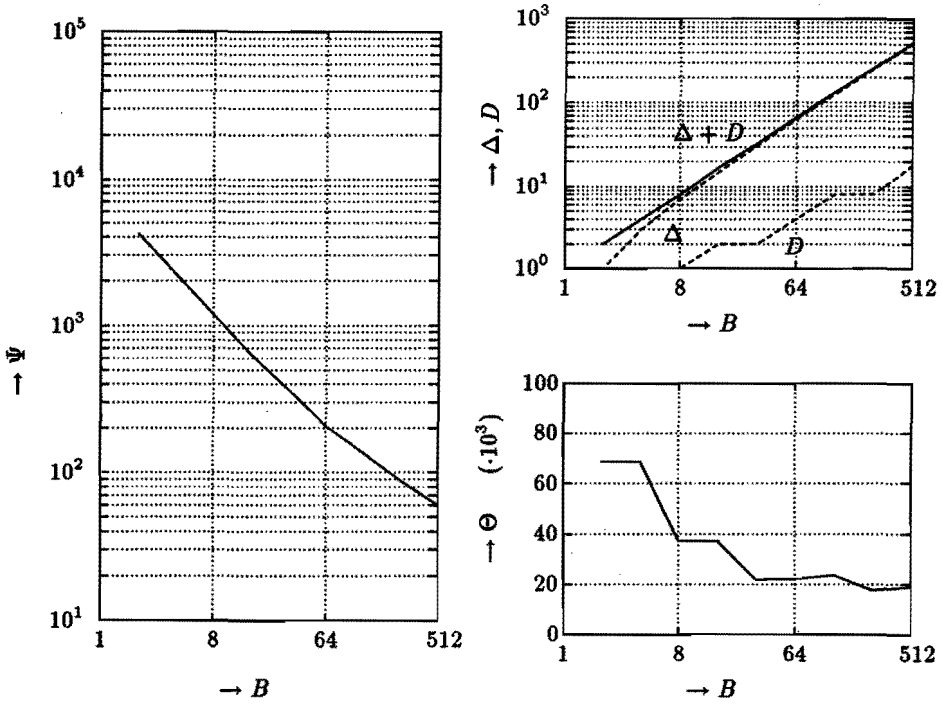



Figure 4.3: Properties of PBFDC.

part one out of two time domain algorithms, BOP and BRLS, was chosen as basis, depending on the choice of the block length $B$ and the filter length $N$. Also here two starting points can be used, the BOP algorithm and the DBOP algorithm. When the block length $A$ is larger than $Z$, BOP is used (compare to $B$ larger than $N$ in the BFDAF case). On the other hand, if the partition length $Z$ is larger than the block length $A$, DBOP is used as basis (compare to $N$ larger than $B$ in the BFDAF case). Like in the BFDAF case, also here the result of both approaches will be the same.

### 4.2.2 BOP as Basis

First a BOP update equation (2.49) with block length $A$, is partitioned in $g_{\mathrm{U}} = \lceil N/Z \rceil$ parts[6]

$$\begin{pmatrix} \underline{w}^Z_{g_{\mathrm{U}}-1}[(l+1)A] \\ \vdots \\ \underline{w}^Z_0[(l+1)A] \end{pmatrix} = \begin{pmatrix} \underline{w}^Z_{g_{\mathrm{U}}-1}[lA] \\ \vdots \\ \underline{w}^Z_0[lA] \end{pmatrix} + \begin{pmatrix} \mathcal{X}^{Z,A}[lA-(g_{\mathrm{U}}-1)\cdot Z] \\ \vdots \\ \mathcal{X}^{Z,A}[lA] \end{pmatrix} \cdot \underline{y}^A[lA] \tag{4.19}$$

and each of these parts is implemented efficiently in frequency domain. The partitioned BOP update equation parts, that can be extracted from equation (4.19), are given for $0 \leq i < g_{\mathrm{U}}$ by

$$\underline{w}^Z_i[(l+1)A] = \underline{w}^Z_i[lA] + \mathcal{X}^{Z,A}[lA-iZ] \cdot \underline{y}^A[lA] \tag{4.20}$$

with

$$\underline{y}^A[lA] = 2\alpha(\hat{\mathcal{R}}^A_x[lA])^{-1}\underline{r}^A[lA] \tag{4.21}$$

$$\mathcal{X}^{Z,A}[lA] = \begin{pmatrix} \underline{x}^Z[lA-A+1] & \cdots \underline{x}^Z[lA] \end{pmatrix} \tag{4.22}$$

$$\underline{x}^Z[lA] = \begin{pmatrix} x[lA-Z+1] & \cdots & x[lA] \end{pmatrix}^t \tag{4.23}$$

and for $0 \leq i < g_{\mathrm{U}}$

$$\underline{w}^Z_i[lA] = \begin{pmatrix} w_{iZ+Z-1} & \cdots & w_{iZ} \end{pmatrix}^t. \tag{4.24}$$

[6]When $N/Z$ is not an integer, the filter vector $\underline{w}^N$ is extended to a length $\lceil N/Z \rceil \cdot Z = g_{\mathrm{U}} \cdot Z$ vector by appending $Z \cdot \lceil N/Z \rceil - N$ coefficients that are kept zero, as was done in the filter part. Note that $g_{\mathrm{U}} \cdot Z$ is not necessarily equal to $g_{\mathrm{F}} \cdot Q$. As long as they are both not smaller than $N$, this is no problem (as we will see later on).

Choosing the FFT length $L \geq Z + A - 1$, an equivalent procedure as used in chapter 3 (with BOP as basis) can be used to approximate equations (4.20) and (4.21). The efficient frequency domain normalization is given by

$$\underline{Y}^L[lA] = 2\alpha(\hat{\underline{P}}_x^L[lA])^{-1} \otimes \underline{R}^L[lA] \tag{4.25}$$

while the update equations (4.20) are transformed for $0 \leq i < g_{\mathrm{U}}$ to

$$\begin{aligned} \underline{w}_i^Z[(l+1)A] &= \underline{w}_i^Z[lA] + \begin{pmatrix} \mathbf{J}^Z & \mathbf{0}^{Z,L-Z} \end{pmatrix} \cdot (\mathcal{F}^M)^{-1} \\ &\quad \cdot((\underline{X}_i^L[lA])^* \otimes \underline{Y}^L[lA]) \end{aligned} \tag{4.26}$$

with for $0 \leq i < g_{\mathrm{U}}$

$$\underline{X}_i^L = \mathcal{F}^L \cdot \underline{x}^L[lA - iZ] \tag{4.27}$$

and

$$\underline{R}^L = \mathcal{F}^L \begin{pmatrix} \underline{0}^{L-A} \\ \underline{r}^A[lA] \end{pmatrix}. \tag{4.28}$$

### 4.2.3 DBOP as Basis

If $Z$ is larger than $A$, then the DBOP update equation (2.65) with autocorrelation matrix dimension $Z$, is partitioned in $g_{\mathrm{U}} = \lceil N/Z \rceil$ parts[7]

$$\begin{aligned} \begin{pmatrix} \underline{w}_{g_{\mathrm{U}}-1}^Z[(l+1)A] \\ \vdots \\ \underline{w}_0^Z[(l+1)A] \end{pmatrix} &= \begin{pmatrix} \underline{w}_{g_{\mathrm{U}}-1}^Z[lA] \\ \vdots \\ \underline{w}_0^Z[lA] \end{pmatrix} \\ &+ 2\alpha \begin{pmatrix} (\hat{\mathcal{R}}_x^Z[lA])^{-1} \cdot \mathcal{X}^{Z,A}[lA - (g_{\mathrm{U}}-1)\cdot Z] \\ \vdots \\ (\hat{\mathcal{R}}_x^Z[lA])^{-1} \cdot \mathcal{X}^{Z,A}[lA] \end{pmatrix} \cdot \underline{r}^A[lA], \end{aligned} \tag{4.29}$$

and each of these parts is implemented efficiently in frequency domain. The partitioned BOP update equation parts, that can be extracted from equation (4.29), are given for $0 \leq i < g_{\mathrm{U}}$ by

$$\underline{w}_i^Z[(l+1)A] = \underline{w}_i^Z[lA] + 2\alpha(\hat{\mathcal{R}}_x^Z[lA])^{-1} \cdot \mathcal{X}_i^{Z,A}[lA] \cdot \underline{r}^A[lA]. \tag{4.30}$$

An equivalent procedure as used in chapter 3 (with BRLS as basis)[8] can be used to approximate this equation. Like in chapter 3 also here the result

[7] When $N/Z$ is not an integer, $\underline{w}^N[k]$ is extended like in the case where BOP is used as basis.

[8] The partition length $Z$ equals the dimension of the inverse autocorrelation matrix in partitioned DBOP, while in BRLS the filter length $N$ equals the dimension of the inverse autocorrelation matrix. By replacing $Z$ by $N$, $A$ by $B$ and $lA$ by $kB$ both algorithms are equal.

is equal to the result of the approach with BOP as basis (see previous subsection).

### 4.2.4 FD Implementation of Update Part

For an efficient implementation, the input signal FFTs can be combined like in the filter part. If we assume that $Z/A = z$ is an integer, a simple delay line in frequency domain can be used for the vectors $\underline{X}_i^L[lA]$ defined in equation (4.27), with $i > 0$,[9] since then for $0 < i < g_U$

$$\underline{X}_i^L[lA] = \underline{X}_{i-1}^L[(l-z)A]. \tag{4.32}$$

To keep the number of FFTs as small as possible, in most cases $Z/A$ is an integer, which yields the partitioned update part depicted in figure 4.4. In this figure each box "$\Delta''$"represents a one sample delay delaying over $A \cdot T$ seconds ("$z\Delta''$" depicts $z$ of those in series). In the following sections two methods are described to couple the partitioned update part to a partitioned filter part.

## 4.3 Partitioned BFDAF

### 4.3.1 PBFDAF Algorithm

The easiest way to couple the partitioned filter and update part is by choosing the partition parameters equal. This can be achieved by choosing $Z = Q$, $A = B$, $L = M$ and thus $g_F = g_U$. As we try to obtain efficient realizations of low delay adaptive filters, we will assume that $Q/B = q$ is an integer, thus requiring only one input signal FFT.[10] If the conjugation operator is moved, then the update part of the algorithm can use the delayline of the filter part, as then the vectors $\underline{X}_0^L[lA]$ to $\underline{X}_{N/Z-1}^L[lA]$ of figure 4.4 are equivalent to the vectors used in the delay line of figure 4.2.

The result is the Partitioned Block Frequency Domain Adaptive Filter (PBFDAF) [13, 46] (a generalization of the Multi Step Size (MSS) Frequency Domain Adaptive Filter in [49]). The result is depicted in figures 4.5 and 4.6.

[9]When $Z/A$ is an integer, implying that the greatest common divisor of $Z$ and $A$, $\gcd\{Z, A\}$, equals $A$, only one FFT is needed. Otherwise we still can use a delayline, but we need $A/\gcd\{Z, A\}$ FFTs (per $A$ samples), by using (for $A/\gcd\{Z, A\} \leq i < g_U$)

$$\underline{X}_i^L[lA] = \underline{X}_{i-A/\gcd\{Z,A\}}^L[(l - Z/\gcd\{Z, A\})A]. \tag{4.31}$$

In the examples considered in this thesis $Z = A$, so $Z/A$ is an integer.

[10]This also means that DBOP is always the basis!

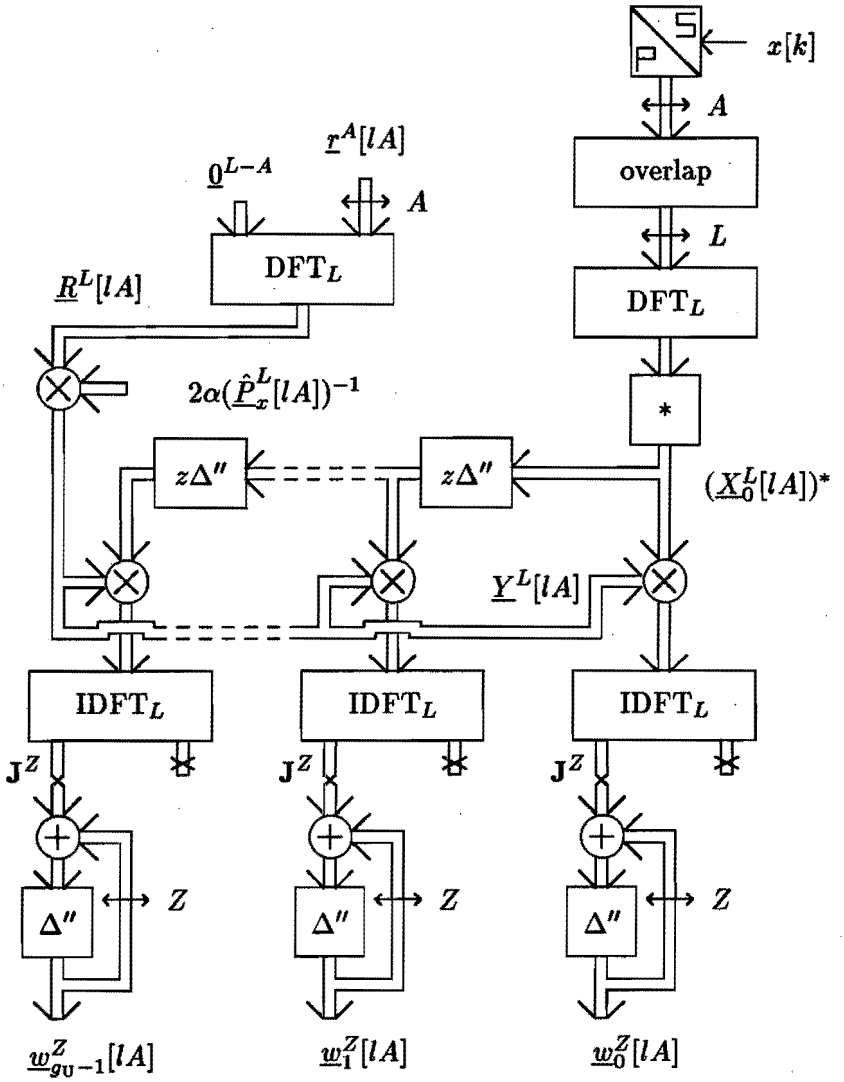



Figure 4.4: Partitioned update part.

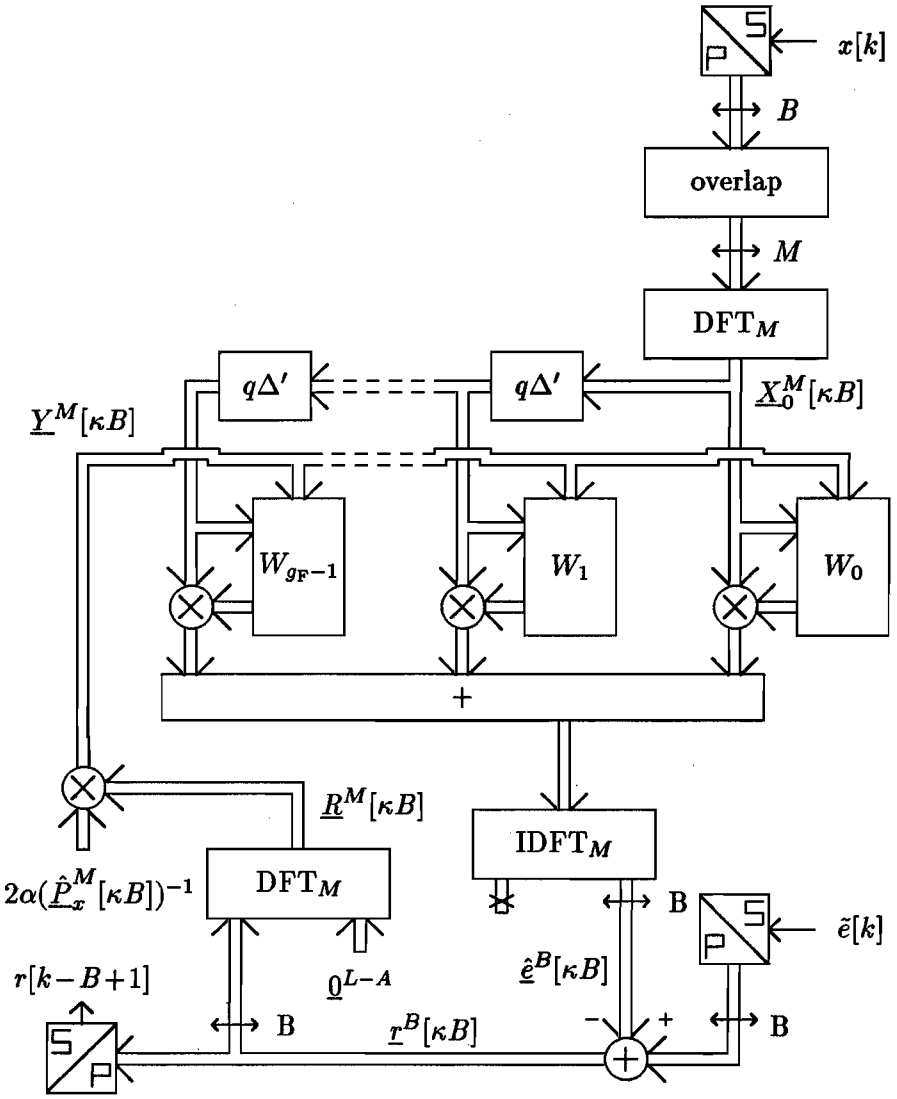



Figure 4.5: Partitioned BFDAF.

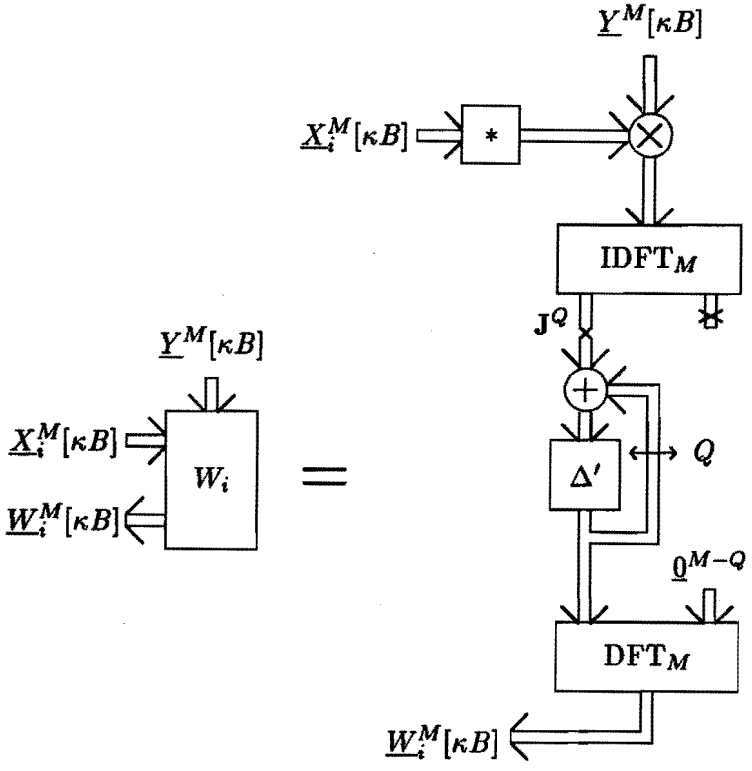



Figure 4.6: Update block of PBFDAF.

### 4.3.2 Properties of PBFDAF

With help of the properties of the PBFDC and BFDAF the following properties for the PBFDAF can be derived:

- Algorithm processing delay

$$\Delta_{\text{PBFDAF}} = B - 1. \tag{4.33}$$

- Computation processing delay for FFT, IFFT and elementwise multiplication

$$D_{\text{PBFDAF}} = 2 \cdot D_{\text{FFT}}\{M\} + D_{\otimes}\{M\}. \tag{4.34}$$

- Computational complexity: computation of FFTs, elementwise multiplications and decorrelation

$$\Psi_{\text{PBFDAF}} = \frac{(3 + 2g_{\text{F}}) \cdot \Psi_{\text{FFT}}\{M\} + 2g_{\text{F}} \cdot \Psi_{\otimes}\{M\} + \Psi_{P}\{M\}}{B}. \tag{4.35}$$

- Memory occupation:
  - Input delayline (length $M$), $g_{\text{F}} - 1$ times $q$ vector delays (width $M$) and output samples (length $B$)
  - FFT input/output (length $M$)
  - Twiddle factors (elements of $\mathcal{F}^M$) (number $< M$)
  - Weight vectors $\underline{W}_i^M$ ($g_{\text{F}}$ times length $M$)
  - Decorrelation (length $M/2$)

$$\Theta_{\text{PBFDAF}} \approx (\frac{7}{2} + (g_{\text{F}} - 1) \cdot (q + 1)) \cdot M + B. \tag{4.36}$$

If $M$ is much smaller than $N$,[11] this can be approximated by

$$\Theta_{\text{PBFDAF}} \approx (2 + q + \frac{1}{q}) \cdot N. \tag{4.37}$$

[11] See the remarks in connection with equation (4.18).

### 4.3.3 Convergence and Tracking

The PBFDAF algorithm behaves like a (D)BOP algorithm with autocorrelation matrix dimension $\max\{Q, B\}$. When the input signal can be decorrelated with its inverse autocorrelation matrix of order $\max\{Q, B\}$, the PBFDAF algorithm behaves (approximately) as a BFDAF algorithm with an equal block length (see subsection 3.3.5). If $Q$ is in the order of $B$ (say $B < 3Q < 9B$) then the approximation error can influence the decorrelation properties, depending on the input signal autocorrelation vector.

Like in the FBNLMS and BFDAF case (subsections 3.2.3 and 3.3.5), one could also try to use an unconstrained approach here, which means that we can skip $2g_F$ FFTs. As the filter parts link up to one another, the circular nature of the frequency domain convolution, no longer cancelled by a time domain windowing operation, will cause a coupling of the coefficients from one subfilter to the next (the length of the unknown system is not smaller than $Q$). The adaptive filter algorithm then no longer converges to the Wiener solution [31]. This causes large degradation in convergence behaviour or even divergence of the adaptive filter coefficients.

### 4.3.4 Example of PBFDAF

Our example illustrates the relation between delay and complexity. In table 4.2 and figure 4.7 the PBFDAF complexity, delay and memory are given as a function of the block length $B$ for a filter length $N = 4000$.[12] The partition length $Q$ is chosen in such a way that the complexity is minimized. Complexity of PBFDAF is much reduced compared to BFDAF

| $B$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $Q$ | 1 | 2 | 12 | 24 | 48 | 96 | 192 | 384 | 768 | 512 |
| $q$ | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 |
| $M$ | 1 | 4 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 1024 |
| $g_F$ | 4000 | 2000 | 334 | 167 | 84 | 42 | 21 | 11 | 6 | 8 |
| $\Delta$ | 0 | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |
| $D$ | 1 | 1 | 1 | 1 | 1 | 2 | 4 | 8 | 18 | 18 |
| $\Psi$ | 8005 | 12010 | 8382 | 5470 | 3434 | 2086 | 1253 | 792 | 532 | 338 |
| $\Theta$ $(\cdot 10^3)$ | 8.0 | 16.0 | 21.4 | 21.5 | 21.7 | 21.9 | 22.2 | 23.9 | 27.4 | 19.5 |

Table 4.2: Delay, complexity and memory of PBFDAF.

(and certainly compared to the time domain algorithms). For very small

[12] For $B = 1$, the NLMS algorithm is the "optimal" PBFDAF. Of course, no FFT is carried out then!

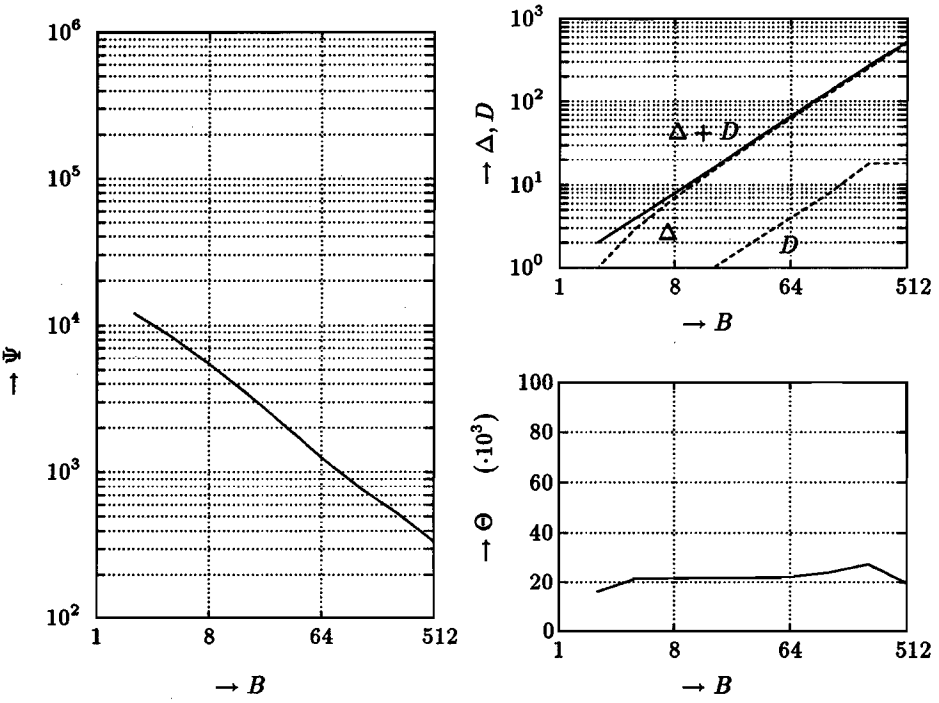



Figure 4.7: Properties of PBFDAF.

delays, like the 0.5 milli-seconds of our example, we still need a huge complexity (approximately equal to the complexity needed in the (simplest) time domain case (LMS)).

## 4.4 Decoupled PBFDAF

### 4.4.1 DPBFDAF Algorithm

When choosing the partition lengths of both the filter and update part different we are able to realize an efficient overall implementation with on one hand a given minimal allowable processing delay (by choosing $B$ small) and on the other hand a large decorrelation dimension (by choosing large $Z$ and $A$). The result is the Decoupled Partitioned Block Frequency Domain Adaptive Filter (DPBFDAF) [16]. The filter part of DPBFDAF is depicted in figure 4.8.

To create one algorithm from the filter part of figure 4.8 and the update part of figure 4.4 we have to define two interfaces between this filter and the update part: one to connect the adaptive weight vectors and the other one to connect the residual signal vectors.

For the interface that connects the adaptive weight vectors it is noted that it is not useful to compute the weights more often than they are needed. For this reason we assume $A \geq B$. For simplicity reasons we will take $A$ as an integer multiple of $B$, with $\gamma = A/B = \lceil A/B \rceil$. Connection is established by first merging the separate vectors of the update part into one filter vector, and, after that, partitioning this filter vector again into parts of length $Q$. These vectors can be transformed to frequency domain. This is depicted in figure 4.9 and can be expressed as follows

$$\begin{pmatrix} \underline{w}^{Q}_{g_F-1}[\kappa B] \\ \vdots \\ \underline{w}^{Q}_{0}[\kappa B] \end{pmatrix} = \begin{pmatrix} \underline{w}^{Z}_{g_U-1}[\lfloor \frac{\kappa}{\gamma} \rfloor A] \\ \vdots \\ \underline{w}^{Z}_{0}[\lfloor \frac{\kappa}{\gamma} \rfloor A] \end{pmatrix}. \tag{4.38}$$

The "Hold" boxes in figure 4.9 perform the upsampling operation from the lower update part sample rate $(1/(A \cdot T))$ to the higher filter part sample rate $(1/(B \cdot T))$, by defining its output vector as its most recent input vector (which means that $\gamma$ copies of each input vector are put out before a new input vector is offered to the "Hold" box). Note that the frequency domain transformation takes place before the hold operation. This way the FFTs are perfomed at the lowest possible sample rate. In equation (4.38)

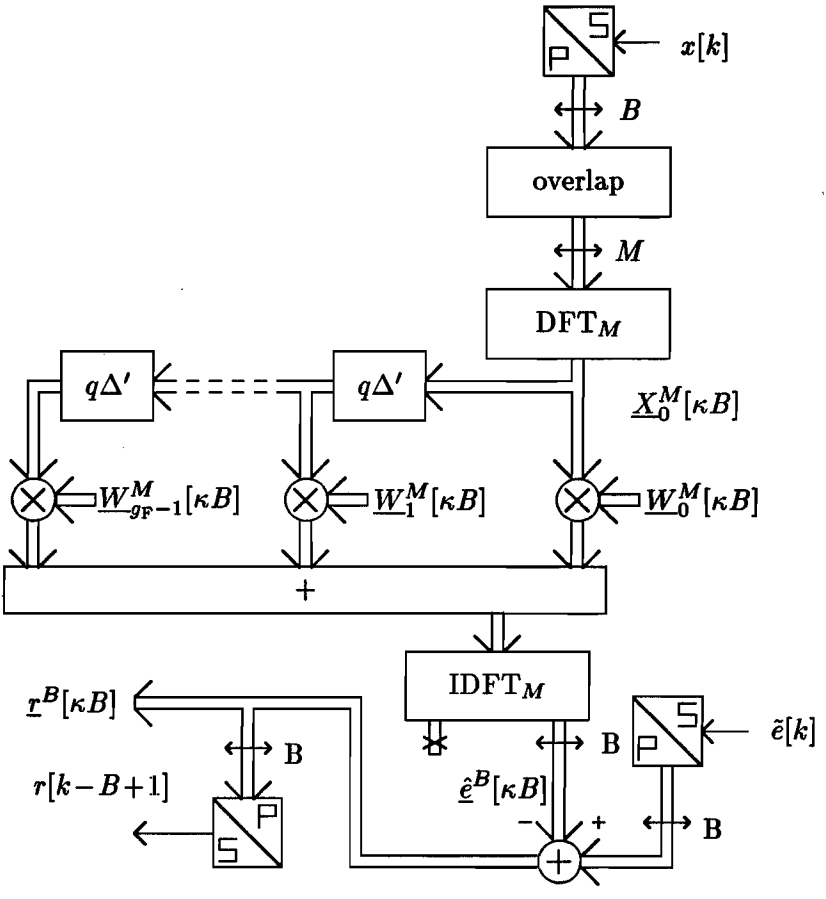



Figure 4.8: Filter part of DPBFDAF.

the FFTs are not incorporated to keep the "Compose and Split" operation clearly visible.[13]

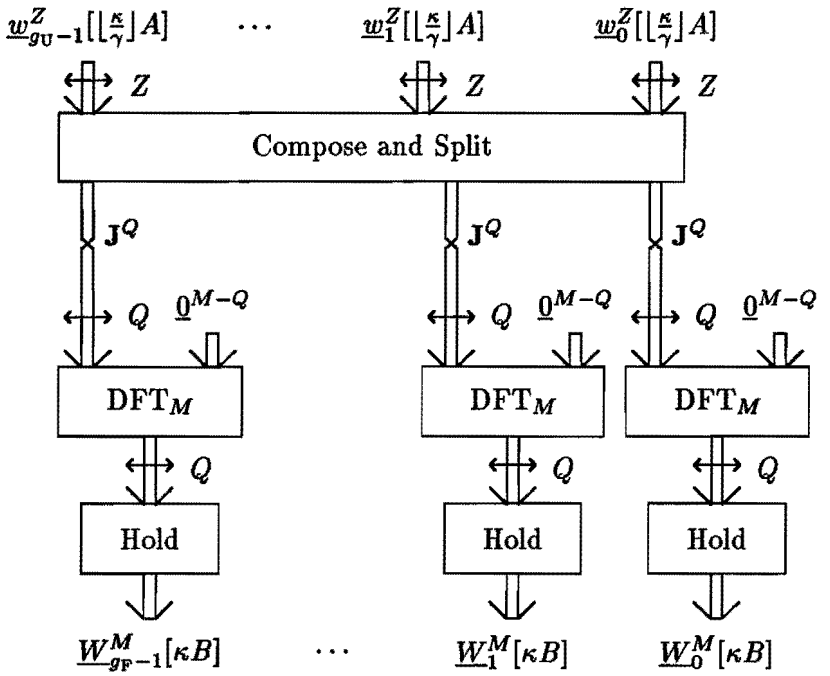



Figure 4.9: Interface for adaptive weight coefficients.

For the second interface, that connects the residual signal vector, we note that the filter part generates vectors $\underline{r}^B[\kappa B]$. These vectors contain $B$

[13] If $g_U Z \neq g_F Q$, and/or $N$ is smaller than $g_U Z$ or $g_F Q$, the filter and update part can still be connected (see also footnotes at page 76 and 82). We therefore have to divide the "Compose and Split" operation in "Compose" and "Split" operations. The first one composes $\underline{w}^N[\lfloor\kappa/\gamma\rfloor A]$ by

$$\underline{w}^N[\lfloor\kappa/\gamma\rfloor A] = \begin{pmatrix} \mathbf{0}^{N, g_U Z - N} & \mathbf{I}^N \end{pmatrix} \cdot \begin{pmatrix} \underline{w}^Z_{g_U-1}[\lfloor\frac{\kappa}{\gamma}\rfloor A] \\ \vdots \\ \underline{w}^Z_0[\lfloor\frac{\kappa}{\gamma}\rfloor A] \end{pmatrix} \tag{4.39}$$

while the second one splits an extension by zeros of $\underline{w}^N[\lfloor\kappa/\gamma\rfloor A]$ by

$$\begin{pmatrix} \underline{w}^Q_{g_F-1}[\lfloor\kappa/\gamma\rfloor A] \\ \vdots \\ \underline{w}^Q_0[\lfloor\kappa/\gamma\rfloor A] \end{pmatrix} = \begin{pmatrix} \underline{0}^{g_F Q - N} \\ \underline{w}^N[\lfloor\kappa/\gamma\rfloor A] \end{pmatrix}. \tag{4.40}$$

In figure 4.10 equations (4.39) and (4.40) are depicted.

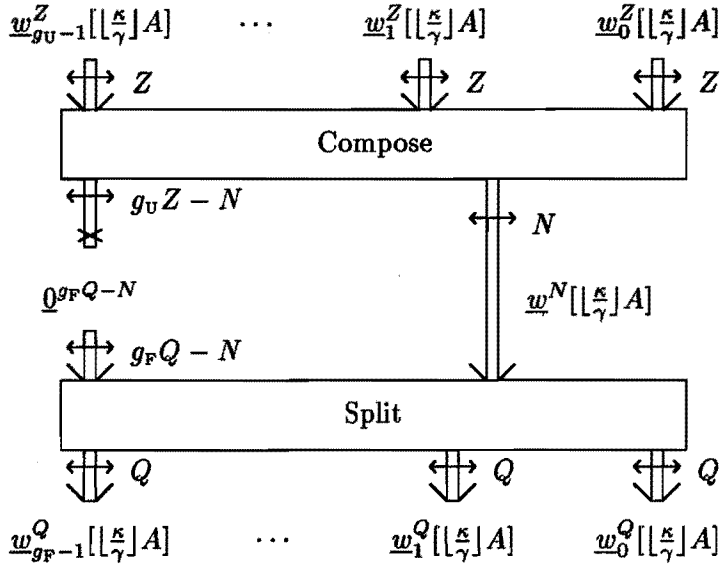



Figure 4.10: Compose and Split in detail.

samples at a rate $\frac{1}{B \cdot T}$. Since $A/B = \gamma$ is assumed to be an integer, we can put $\gamma$ of these residual signal vecors in a delay line, upto a length $A$ vector. Sample rate reduction with a factor $\gamma = A/B$ finally yields $\underline{r}^A[lA]$

$$\underline{r}^A[lA] = \begin{pmatrix} \underline{r}^B[(l\gamma - \gamma + 1) \cdot B] \\ \vdots \\ \underline{r}^B[l\gamma \cdot B] \end{pmatrix}. \tag{4.41}$$

This is depicted in figure 4.11.

### 4.4.2 Properties of DPBFDAF

With help of the properties of the PBFDC and PBFDAF the following properties for the DPBFDAF can be derived:

- Algorithm processing delay

$$\Delta_{\text{DPBFDAF}} = B - 1. \tag{4.42}$$

- Computation processing delay for FFT, IFFT and elementwise multiplication

$$D_{\text{DPBFDAF}} = 2 \cdot D_{\text{FFT}}\{M\} + D_{\otimes}\{M\}. \tag{4.43}$$

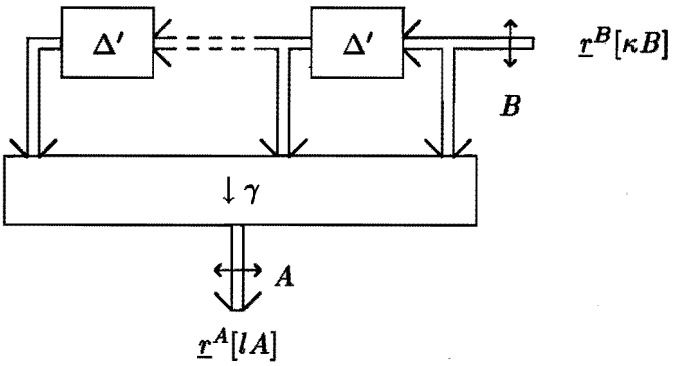



Figure 4.11: Interface for residual signal.

- Computational complexity: computation of FFTs, elementwise multiplications and decorrelation

$$\Psi_{\text{DPBFDAF}} = \frac{2 \cdot \Psi_{\text{FFT}}\{M\} + g_{\text{F}} \cdot \Psi_{\otimes}\{M\}}{B} \tag{4.44}$$
$$+ \frac{g_{\text{F}} \cdot \Psi_{\text{FFT}}\{M\} + (2 + g_{\text{U}}) \cdot \Psi_{\text{FFT}}\{L\} + g_{\text{U}} \cdot \Psi_{\otimes}\{L\} + \Psi_{P}\{L\}}{A}.$$

- Memory occupation:
  - Input delaylines (length $M$ and $L$), $g_{\text{F}} - 1$ times $q$ vector delays (width $M$), $g_{\text{U}} - 1$ times $z$ vector delays (width $L$) and output samples (length $B + A$)
  - FFT input/output (length $M$ and $L$)
  - Twiddle factors (elements of $\mathcal{F}^M$ and $\mathcal{F}^L$) (number $< M + L$)
  - Weight vectors $\underline{W}_i^M$ and $\underline{w}^N$ ($g_{\text{F}}$ times length $M$ and length $N$)
  - Decorrelation (length $L/2$)

$$\begin{aligned}\Theta_{\text{DPBFDAF}} \approx\ & (3 + (g_{\text{F}} - 1) \cdot (q + 1)) \cdot M + B + A + N \\ & + (\frac{7}{2} + (g_{\text{U}} - 1) \cdot z) \cdot L.\end{aligned} \tag{4.45}$$

If $M$ is much smaller than $N$,[14] and $Z = A$,[15] this can be approximated by

$$\Theta_{\text{DPBFDAF}} \approx (5 + q + \frac{1}{q}) \cdot N + 6 \cdot A. \tag{4.46}$$

[14] See the remarks in connection with equation (4.18).
[15] For every case considered in this thesis, this is true.

### 4.4.3 Convergence and Tracking

Now the value of the update part parameters $A$ and $Z$ can be chosen large enough to ensure that the input signal can be decorrelated with an autocorrelation matrix of order $A$ (see chapter 6), and that the inter block correlation can be neglected (see subsection 2.5.4).

In order to reduce computational complexity, we would like to choose the block length $A$ in the order of $N$. The use of DPBFDAF implies that the power vector is updated only once every $A$ samples, and that the update also takes place only once every $A$ samples. The latter means that tracking properties are much worsened (see 2.3.3), and the former means that we may not be able to follow non-stationarities in the input signal. Especially in cases where speech signals are used, like in the Acoustic Echo Canceller, this means that normalization is of no use anymore (see chapter 8). This all induces a maximum on the block length $A$, which implies a certain minimal computational complexity.

In the Decoupled algorithm thus still another coupling exists: the dimension and rate of the normalization vector are coupled to the update part parameters (FFT length and block length). In chapter 6 a method to decouple the dimension of the normalization vector from the update part parameters is discussed. This makes it possible to enlarge $A$ without influencing the normalization properties.

### 4.4.4 Example of DPBFDAF

Our example illustrates the relation between delay and complexity. In table 4.3 and figure 4.12 the DPBFDAF complexity, delay and memory are given as a function of the block length $B$ for a filter length $N = 4000$.[16] The partition length $Q$ is chosen in such a way that the complexity is minimized. For all DPBFDAF examples, the update part parameters are equal, with $A = Z = 512$,[17] $L = 1024$ and $g_U = 8$. Complexity of DPBFDAF is much reduced compared to PBFDAF (and certainly compared to BFDAF and the time domain algorithms). For very small delays, like the 0.5 milliseconds of our example, we still need a huge complexity. In the example, for $B = 4$, the implementation requires 2414 multiplications per sample, equal to 19.3 million multiplications per second.

We observe that the amount of memory $\Theta_{\text{DPBFDAF}}$ depends strongly on

[16] For $B = 1$, the DPBFDAF uses a TDC as "optimal" PFDC. Of course, no FFT is carried out then!

[17] This implies that for $B = Q = 512$, PBFDAF and DPBFDAF are equal.

| $B$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $Q$ | 1 | 30 | 60 | 56 | 112 | 96 | 192 | 384 | 768 | 512 |
| $q$ | 1 | 15 | 15 | 7 | 7 | 3 | 3 | 3 | 3 | 1 |
| $M$ | 1 | 32 | 64 | 64 | 128 | 128 | 256 | 512 | 1024 | 1024 |
| $g_F$ | 4000 | 134 | 67 | 72 | 36 | 42 | 21 | 11 | 6 | 8 |
| $\Delta$ | 0 | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |
| $D$ | 1 | 1 | 1 | 1 | 2 | 2 | 4 | 8 | 18 | 18 |
| $\Psi$ | 4180 | 4420 | 2414 | 1391 | 852 | 588 | 440 | 382 | 368 | 338 |
| $\Theta$ $(\cdot 10^3)$ | 19.4 | 84.0 | 84.1 | 52.4 | 52.5 | 37.2 | 37.4 | 39.0 | 42.2 | 19.5 |

Table 4.3: Delay, complexity and memory of DPBFDAF.

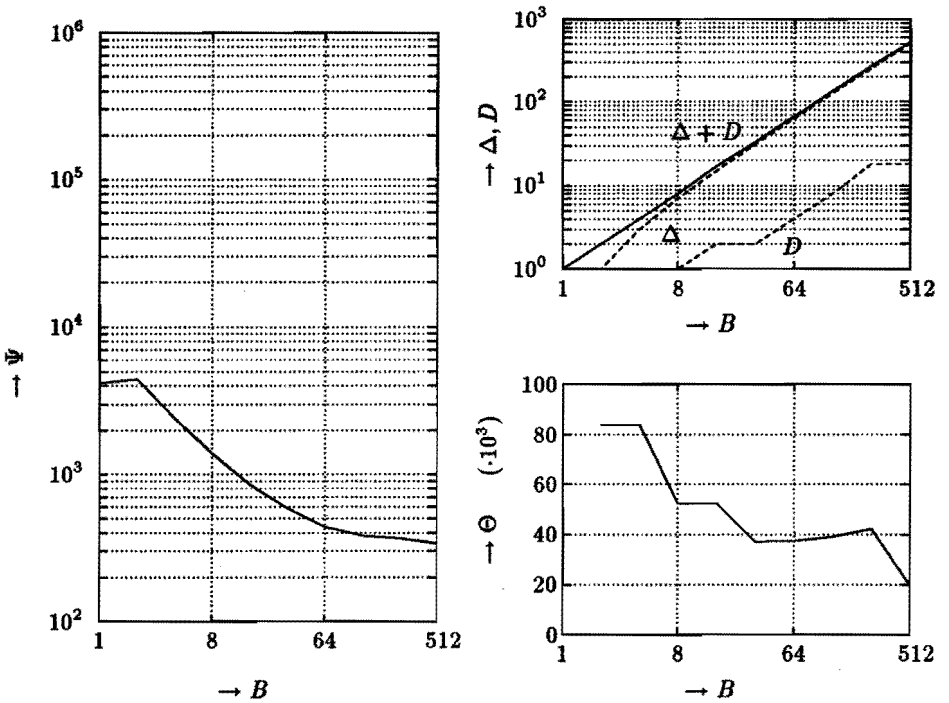



Figure 4.12: Properties of DPBFDAF.

the block length $B$ in this case. This is caused by the fact that we chose to minimize the computational complexity, which implies that for decreasing $B$, the factor $q = Q/B$ increases. At cost of a slightly larger computational complexity, $\Theta_{\text{DPBFDAF}}$ can be reduced considerably. If we look at the example ($B = 4$), we can choose $Q = 12$, then $\Psi_{\text{DPBFDAF}} = 2711$ (an increase by a factor 1.12) and $\Theta_{\text{DPBFDAF}} = 36.8 \cdot 10^3$ (a decrease by a factor 2.29).

## 4.5 Conclusions

When a small processing delay is needed, the PBFDAF requires a lower computational complexity than the BFDAF at cost of a smaller decorrelation dimension. To gain the same decorrelation dimension as the BFDAF with a small processing delay and a computational complexity that is even lower than with the PBFDAF, the DPBFDAF is a good solution. However, even the DPBFDAF often has a computational complexity that is too high for real-time implementation purposes, especially when non-stationarity of the input signal is the main problem. In the next section we will introduce a method for further complexity reduction by non-uniform partitioning of the convolution.

# Chapter 5

# Non-Uniform Partitioned BFDAF

*For the realization of large adaptive filters the Block Frequency Domain Adaptive Filter (BFDAF), as described in chapter 3, combines a relative low implementation complexity with good convergence behaviour. On the other hand however the resulting processing delay of this algorithm is very large caused by the length of the FFTs involved.*

*By partitioning of the filter vector into smaller vectors a small block length can be used without a huge penalty in the computational complexity. This results in the Partitioned BFDAF (PBFDAF). By using a different partition length in the filter and in the update part of the algorithm, we can make an independent choice for both the block length in the filter part and the decorrelation vector length in the update part. The result is the Decoupled PBFDAF (DPBFDAF), with an even lower computational complexity than PBFDAF.*

*However, the computational complexity of the filter part still is quite large in DPBFDAF, and in fact determines the overall complexity. In some applications (such as the Acoustic Echo Canceller) the resulting computational complexity of the whole algorithm still is very large. Therefore a new algorithm for frequency domain convolution is introduced, that uses a non-uniform partitioning of the filter (into sub-filters). This Non-Uniform Partitioned BFDC (NUPBFDC) can be used in adaptive filtering, resulting in the Non-Uniform Partitioned BFDAF (NUPBFDAF).*

## 5.1 Non-Uniform Partitioning of Convolution

### 5.1.1 Information in Delayline

To reduce computational complexity, a filter can be partitioned into several sub-filters of equal length $Q$ that are individually transformed to frequency domain, as was shown in the previous chapter. This uniform partitioning is described by the following equation (compare to the block-based equation (4.2))

$$\begin{aligned} \hat{e}[k] &= \sum_{i=0}^{g_F-1} (\underline{x}^Q[k-iQ])^t \cdot \underline{w}_i^Q \\ &= \sum_{i=0}^{g_F-1} \hat{e}_i[k] \end{aligned} \tag{5.1}$$

with $g_F = \lceil N/Q \rceil$.[1] The partitioning is given in equations (4.4) and (4.7) and figure 4.1. The above (uniform) Partitioned Time Domain Convolution (Partitioned TDC) is depicted in figure 5.1. The boxes "$Q\Delta$" denote a delay over Q samples, while the boxes "$\underline{w}_i^Q$" are used for the filtering (convolution) operation. In figure 5.1 we see that the $i$'th sub-filter (with

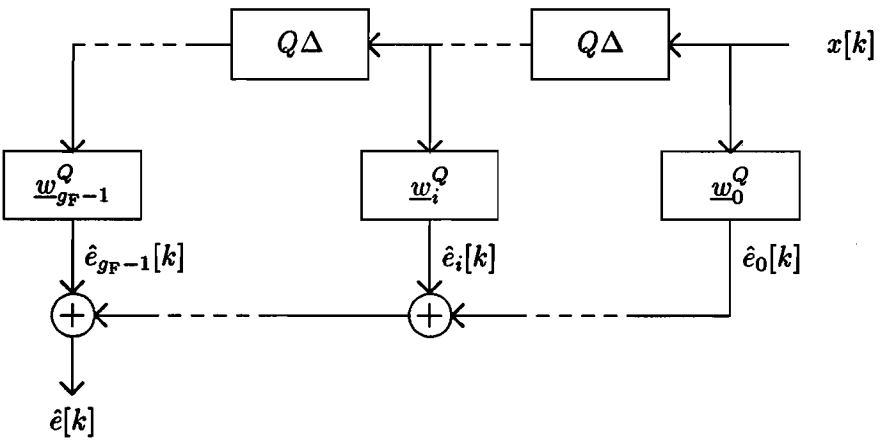



Figure 5.1: (Uniform) Partitioned Time Domain Convolution.

filter vector $\underline{w}_i^Q$) needs an input signal that is delayed over $iQ$ samples.

[1] If $g_F \cdot Q > N$, then $\underline{w}^N$ is extended to $\underline{w}^{Q \cdot \lceil N/Q \rceil}$ by appending $Q \cdot \lceil N/Q \rceil - N$ coefficients (that equal zero).

This means that the calculation of $\hat{e}_i[k]$ can be started $iQ$ samples earlier, as shown in figure 5.2.

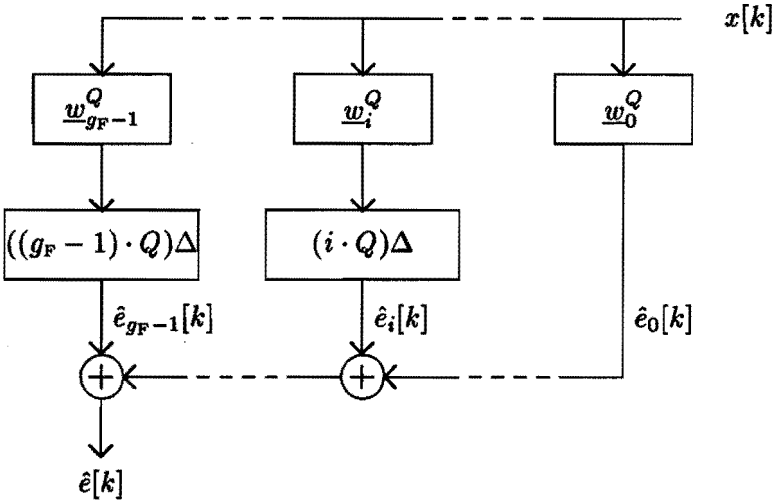



Figure 5.2: Moving the delayline in (Uniform) Partitioned TDC.

The resulting delays can be used to compensate for the algorithm processing delay induced by a block processing approach. For the $i$'th sub-filter we can increase the block length by $iQ$ compared to the 0'th subfilter block length, without extra algorithm processing delay. Realizing sub-filters with a different block length implies that we cannot use a delay-line in frequency domain for the input signal vectors (as in the previous chapter). This means that we can as well choose a non-uniform partitioning of the filter (using several different partition lengths) to minimize computational complexity [17].

### 5.1.2 Non-Uniform TD Partitioning

Here we introduce a new approach to partition the filter into sub-filters of not necessarily equal length. The goal is to reduce computational complexity by using a larger filter length where a larger block length is possible. In figure 5.3 this non-uniform partitioning of the filter vector is depicted. The application of this non-uniform partitioning to a convolution is depicted in figure 5.4 (compare to figure 5.1), and can be described by

$$\hat{e}[k] = \sum_{j=0}^{G-1} (\underline{x}^{S_{j+1}-S_j}[k-S_j])^t \cdot \underline{w}_j^{S_{j+1}-S_j}$$

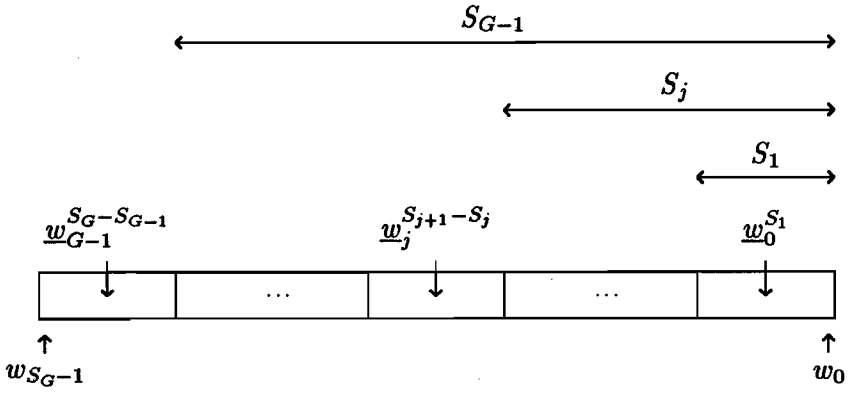



Figure 5.3: Non-uniform partitioning of coefficient vector.

$$= \sum_{j=0}^{G-1} \hat{e}_j[k], \tag{5.2}$$

where $S_0 = 0$ and $S_G \geq N$.[2]

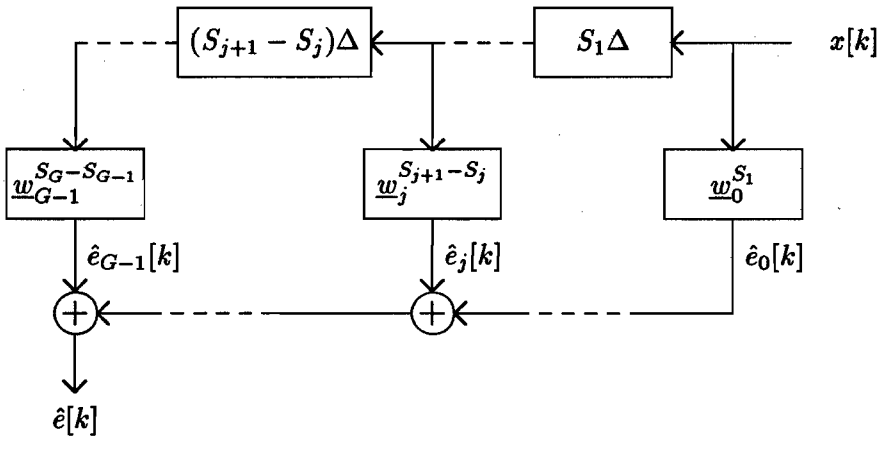



Figure 5.4: Non-uniform partitioned TDC.

In the uniform partitioned case all sub-filters where realized with the same block and partition length, which implies that every sub-filter has the same computation (and algorithm) processing delay. This does, in general, not apply to the case of non-uniform partitioning, so we have

[2]If $S_G > N$ then, like in the uniform partitioned case, $\underline{w}^N$ is extended to $\underline{w}^{S_G}$ by appending $S_G - N$ coefficients (that equal zero).

to take both the computation and algorithm processing delay into account while developing the non-uniform partitioned algorithm. Assuming that the maximum allowable processing delay (sum of algorithm and computation delay) is $D_{\max}$ samples, the $j$'th sub-filter has $D_{\max} + S_j$ sample intervals available for its algorithm and computation processing delay, as shown in figure 5.5. If we assume that the computation processing delay for the $j$'th

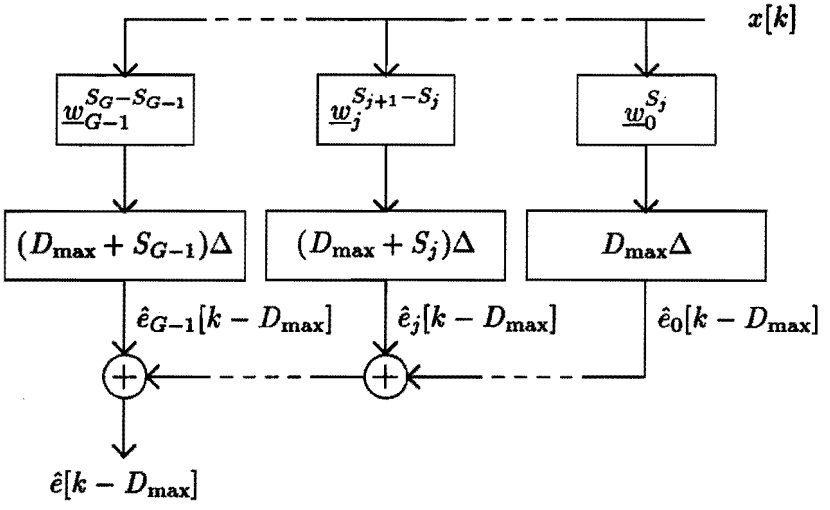



Figure 5.5: Non-uniform partitioned TDC with maximum delay.

sub-filter equals $D_j$, we have $D_{\max} + S_j - D_j$ delays left for the algorithm processing delay.[3]

To minimize computational complexity, every $\hat{e}_j[k]$ of equation (5.2) will be calculated with a PBFDC, which implies the use of $G$ different partition lengths $Q_j$ and block lengths $B_j$ (with for $1 \leq j < G : B_j > B_{j-1}$). The computation of $\hat{e}_j[k]$ with such a PBFDC induces an algorithm processing delay of $B_j - 1$ samples. If we define $\tau_j$ as

$$\tau_j = D_{\max} + S_j - B_j + 1, \tag{5.3}$$

and subtract the algorithm delay $B_j - 1$ from $D_{\max} + S_j - D_j$, we see that $\tau_j - D_j$ delays are left over[4] as depicted in figure 5.6.

[3]While estimating the computation processing delay $D_j$ for each sub-filter, one has to realize that an implementation on a single DSP implies that we cannot compute the sub-filters in parallel. If we assume that the order of the computation of the sub-filters is such that the smaller indexed filters are calculated as late as possible, the $D_j$ can be estimated recursively, by starting with $D_0$, as only the parameters of sub-filters with an index smaller than $j$ will influence the value of $D_j$.

[4]From $\tau_j$ we need $D_j$ delays to compensate for the computation processing delay.

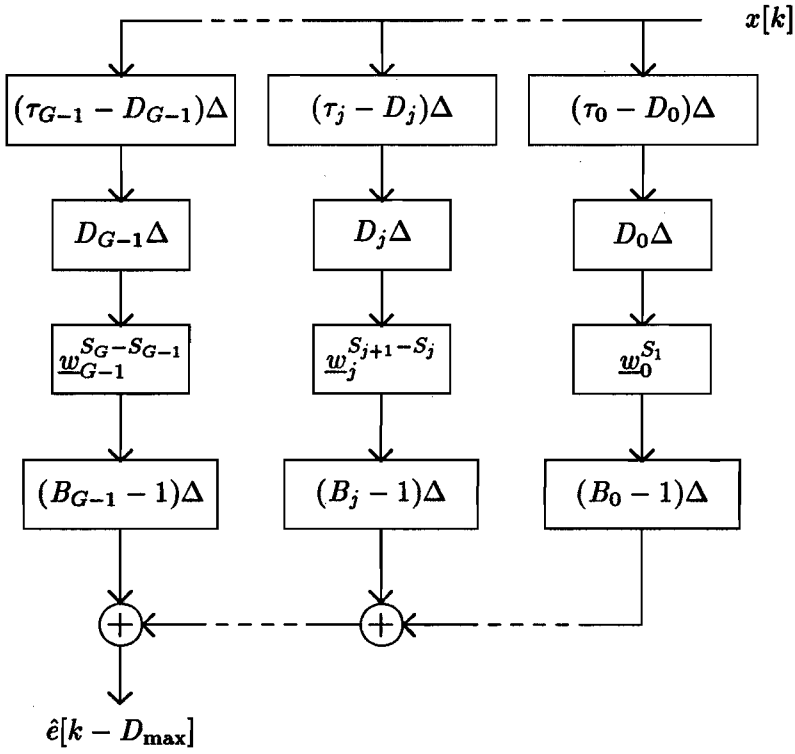



Figure 5.6: Non-uniform partitioned TDC, prepared for FD transformation.

The PBFDC partitions for all $j$, with $0 \leq j < G$, the $j$'th subfilter of length $S_{j+1} - S_j$ into $g_j$ smaller subfilters of length $Q_j$, implying that for all $j$, with $0 \leq j < G$: $S_{j+1} - S_j = g_j \cdot Q_j$.[5] The non-uniform partitioning of the coefficient vector is depicted in figure 5.7, and the non-uniform partioned convolution is described as

$$\begin{aligned} \hat{e}[k] &= \sum_{j=0}^{G-1} \sum_{i=0}^{g_j-1} (\underline{x}^{Q_j}[k - S_j - i \cdot Q_j])^t \cdot \underline{w}_{j,i}^{Q_j} \\ &= \sum_{j=0}^{G-1} \hat{e}_j[k] \end{aligned} \tag{5.4}$$

with for $0 \leq j < G$

$$\hat{e}_j[k] = \sum_{i=0}^{g_j-1} (\underline{x}^{Q_j}[k - S_j - i \cdot Q_j])^t \cdot \underline{w}_{j,i}^{Q_j} \tag{5.5}$$

$$S_j = \sum_{a=0}^{j-1} g_a \cdot Q_a \tag{5.6}$$

$$\underline{x}^{Q_j}[k] = \begin{pmatrix} x[k - Q_j + 1] & \cdots & x[k] \end{pmatrix}^t \tag{5.7}$$

and for $0 \leq j < G$ with $0 \leq i < g_j$

$$\underline{w}_{j,i}^{Q_j} = \begin{pmatrix} w_{S_j - i \cdot Q_j + Q_j - 1} & \cdots & w_{S_j - i \cdot Q_j} \end{pmatrix}^t . \tag{5.8}$$

To obtain a causal realization, for all $j$, with $0 \leq j < G$, the number of delays $\tau_j - D_j$ that is left over must be non-negative. This means that the next set of conditions must be fulfilled for $0 \leq j < G$

$$\begin{aligned} 0 &\leq \tau_j - D_j \\ \Leftrightarrow B_j &\leq D_{\max} - D_j + 1 + S_j \\ \Leftrightarrow B_j &\leq D_{\max} - D_j + 1 + \sum_{a=0}^{j-1} g_a \cdot Q_a . \end{aligned} \tag{5.9}$$

The $j$'th block length $B_j$ is determined by the partition lengths of the sub-filters with a smaller index.[6] $B_0$ is bound to $D_{\max} - D_0 + 1$. The other block lengths can be chosen (much) larger to minimize computational complexity.

[5] It is assumed that for all $0 \leq j < G$, the $S_j$ where chosen such that $(S_{j+1} - S_j)/Q_j$ is an integer.

[6] This means that besides the computation processing delays $D_j$, also the block lengths $B_j$, the partition lengths $Q_j$ and the number of sub-filters $g_j$ can be chosen recursively.

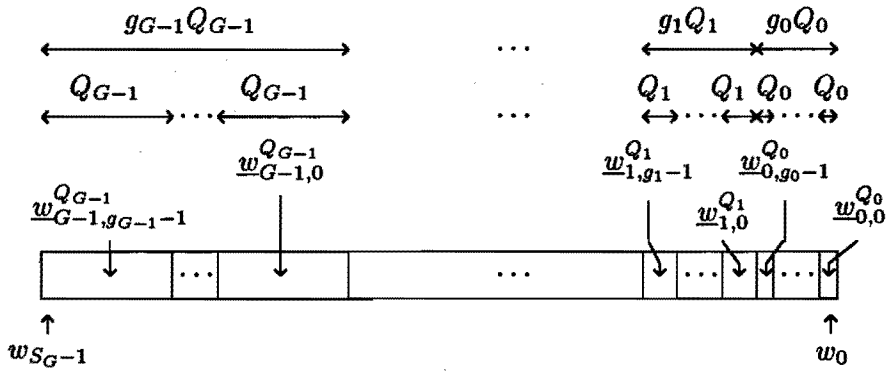



Figure 5.7: Non-uniform partitioning of coefficient vector.

### 5.1.3 Block-based TDC

To prepare a transformation to frequency domain, equations 5.6 are written on block basis. For all $j$, with $0 \leq j < G$, the last $B_j$ samples of $\hat{e}_j[k]$ are collected in one block $\underline{\hat{e}}_j^{B_j}[\kappa_j B_j]$, whose computation is assumed to finish at time instance $\kappa_j B_j$ ($\kappa_j$ denotes the block index of the $j$'th subfilter). To obtain a processing delay of $D_{\max}$ samples, $\hat{e}_j[\kappa_j B_j - D_{\max}]$ has to be the first element of $\underline{\hat{e}}_j^{B_j}[\kappa_j B_j]$, implying that for $0 \leq j < G$

$$\begin{aligned}
\underline{\hat{e}}_j^{B_j}[\kappa_j B_j] &= \begin{pmatrix} \hat{e}_j[\kappa_j B_j - D_{\max}] \\ \vdots \\ \hat{e}_j[\kappa_j B_j - D_{\max} + B_j - 1] \end{pmatrix} \\
&= \sum_{i=0}^{g_j-1} \begin{pmatrix} (\underline{x}^{Q_j}[\kappa_j B_j - \tau_j - i \cdot Q_j - B_j + 1])^t \\ \vdots \\ (\underline{x}^{Q_j}[\kappa_j B_j - \tau_j - i \cdot Q_j])^t \end{pmatrix} \cdot \underline{w}_{j,i}^{Q_j} \\
&= \sum_{i=0}^{g_j-1} (\mathcal{X}^{Q_j,B_j}[\kappa_j B_j - \tau_j - i \cdot Q_j])^t \cdot \underline{w}_{j,i}^{Q_j} \qquad (5.10)
\end{aligned}$$

where

$$\mathcal{X}^{Q_j,B_j}[\kappa_j B_j] = \begin{pmatrix} \underline{x}^{Q_j}[\kappa_j B_j - B_j + 1] & \cdots & \underline{x}^{Q_j}[\kappa_j B_j] \end{pmatrix} \qquad (5.11)$$

Note that if there is only one partition factor ($G = 1$), then $D_{\max} = D_0 + B_0 - 1$, which implies that $\tau_0 = D_0$. Comparing equations (5.10) to the PBFDC equations (4.2), we see that incorporating the computation processing delay causes a shift in the input signal matrix of $D_0$ samples in

equation (5.10) (therefore we can start computation $D_0$ samples earlier). This computation delay has to be incorporated in the equations because the results of all block equations (when $G > 1$), having a different computation delay, must be added in the end.

### 5.1.4 Non-Uniform Partitioned BFDC

Now we can transform all block-equations to frequency domain with the help of Discrete Fourier Transforms (DFT's) (or more efficiently Fast Fourier Transforms (FFT's)) using an overlap save method like in the PBFDC case of the previous chapter, with the Fourier Transform lengths $M_j \geq B_j + Q_j - 1$. For every different partition factor one FFT and one IFFT is needed. For $0 \leq j < G$ this is denoted as

$$\begin{aligned} \underline{\hat{e}}_j^{B_j}[\kappa_j B_j] &= \left( \mathbf{0}^{B_j, M_j - B_j} \quad \mathbf{I}^{B_j} \right) \cdot (\mathcal{F}^{M_j})^{-1} \\ &\quad \cdot \sum_{i=0}^{g_j - 1} (\underline{X}_{j,i}^{M_j}[\kappa_j B_j] \otimes \underline{W}_{j,i}^{M_j}) \end{aligned} \tag{5.12}$$

with for $0 \leq j < G$ and $0 \leq i < g_j$

$$\underline{W}_{j,i}^{M_j} = \mathcal{F}^{M_j} \cdot \begin{pmatrix} \mathbf{J}^{Q_j} \cdot \underline{w}_{j,i}^{Q_j} \\ \underline{0}^{M_j - Q_j} \end{pmatrix} \tag{5.13}$$

$$\underline{X}_{j,i}^{M_j}[\kappa_j B_j] = \mathcal{F}^{M_j} \cdot \underline{x}^{M_j}[\kappa_j B_j - \tau_j - i \cdot Q_j] \tag{5.14}$$

If we assume that $Q_j / B_j = q_j$ is integer (if $g_j > 1$),[7] then also here delay-lines can be used for equation (5.15),[8] as for $0 \leq j < G$ and $1 \leq i < g_j$

$$\underline{X}_{j,i}^{M_j}[\kappa_j B_j] = \underline{X}_{j,i-1}^{M_j}[(\kappa_j - \frac{Q_j}{B_j}) \cdot B_j]. \tag{5.15}$$

The algorithm of equations (5.12), (5.14) and (5.15) is depicted in figure 5.8. The input signal to this Subfil$_j$-block is $\underline{x}^{M_j}[\kappa_j B_j - \tau_j]$. The output signal is $\underline{\hat{e}}_j^{B_j}[\kappa_j B_j]$. The filter vectors $\underline{W}_{j,0}^{M_j}$ till $\underline{W}_{j,g_j-1}^{M_j}$ are fixed vectors. The boxes "$q_j \Delta$" represent $q_j$ one-sample vector delays, where a one-sample delay here means a delay over $B_j \cdot T$ seconds. The whole filter operation (convolution) according to equation (5.4) is depicted in figures

[7] If $g_j = 1$, then the $j$'th PBFDC in fact is a BFDC, so no delayline is needed.

[8] When $Q_j / B_j$ is integer, implying that the greatest common divisor of $Q_j$ and $B_j$, $\gcd\{Q_j, B_j\}$, equals $B_j$, only one FFT is needed. Otherwise we still can use a delayline, but we need $B_j / \gcd\{Q_j, B_j\}$ FFTs (per $B_j$ samples) (see equation (4.12)).

5.9 and 5.8. The boxes "$\tau_j\Delta$" are $\tau_j$ one-sample delays, each one-sample delay delaying over $T$ seconds. The parameter $G$, and all parameters $Q_j$, $B_j$, $g_j$ and $M_j$ can be chosen within the conditions of inequalities (5.9) to minimize computational complexity.

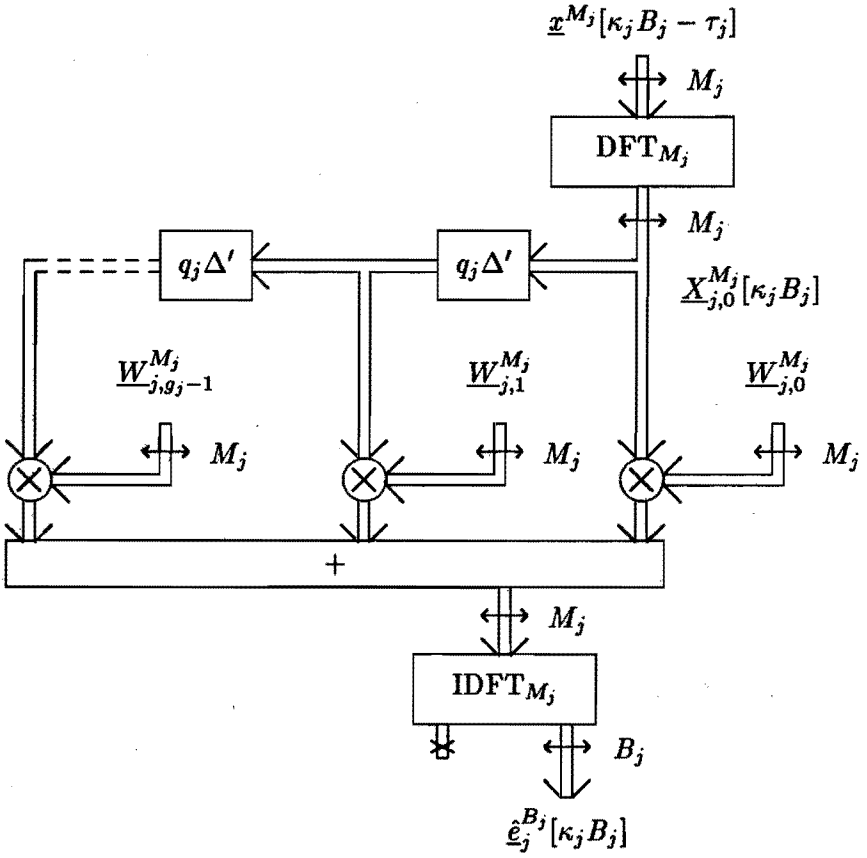



Figure 5.8: Subfilter $\mathrm{Subfil}_j$.

### 5.1.5 Properties of NUPBFDC

The next features of the Non-Uniform PBFDC are derived with help of the PBFDC properties and appendix A:

- Algorithm and computation processing delay

$$\Delta_{\mathrm{NUPBFDC}} + D_{\mathrm{NUPBFDC}} \leq D_{\mathrm{max}}. \tag{5.16}$$

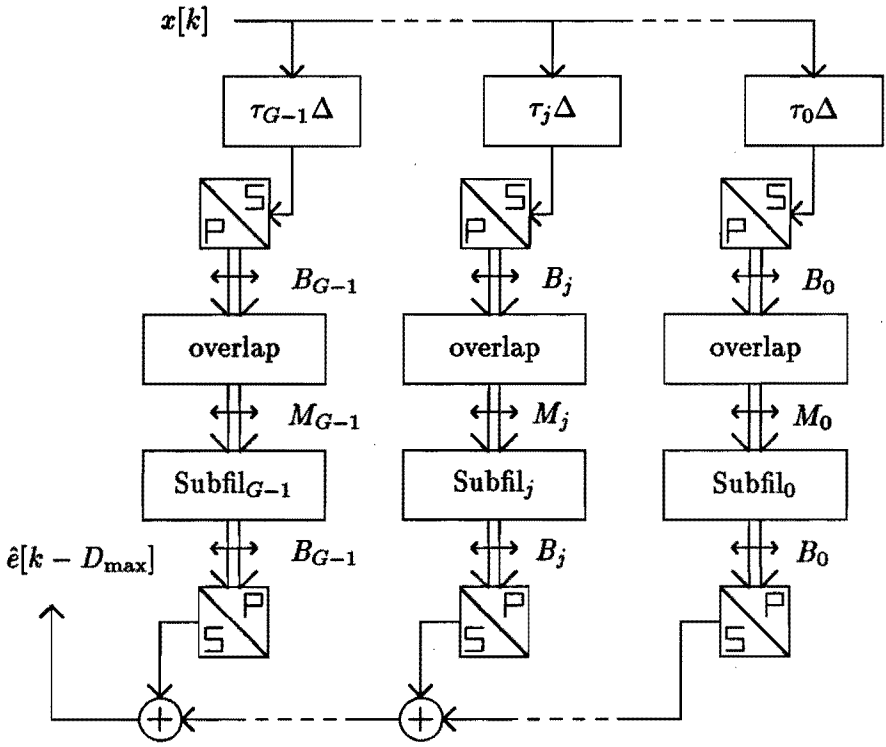



Figure 5.9: Non-Uniform PBFDC.

- Computational complexity: FFTs, IFFTs, elementwise multplications:

$$\Psi_{\text{NUPBFDC}} = \sum_{j=0}^{G-1} \Psi_{\text{NUPBFDC},j} \tag{5.17}$$

$$\Psi_{\text{NUPBFDC},j} = \frac{2 \cdot \Psi_{\text{FFT}}\{M_j\} + g_j \cdot \Psi_{\otimes}\{M_j\}}{B_j}. \tag{5.18}$$

- Memory occupation (compare to PBFDC)

$$\Theta_{\text{NUPBFDC}} \approx \sum_{j=0}^{G-1}((3 + (g_j - 1) \cdot (q_j + 1)) \cdot M_j + B_j). \tag{5.19}$$

If for all $j$, $0 \leq j < G$, $M_j$ is much smaller than $N$,[9] this can be approximated by

$$\Theta_{\text{NUPBFDC}} \approx \sum_{j=0}^{G-1}((2 + q_j + \frac{1}{q_j}) \cdot g_j Q_j). \tag{5.20}$$

### 5.1.6 Example of NUPBFDC

The example illustrates the relation between delay and complexity. In table 5.1 and figure 5.10 the NUPBFDC complexity, delay and memory are given as a function of the block length $B_0$ (determining the algorithm processing delay) for a filter length $N = 4000$. The number of different partition factors and for all $j$, $0 \leq j < G$, the parameters $B_j$, $Q_j$, $g_j$ and $M_j$ (except for $B_0$) are chosen to minimize the complexity $\Psi_{\text{NUPBFDC}}$.

| $B_0$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\Delta + D$ | 1 | 2 | 4 | 8 | 16 | 32 | 65 | 131 | 263 | 529 |
| $G$ | 6 | 5 | 5 | 4 | 3 | 3 | 2 | 2 | 2 | 1 |
| $\Psi$ | 142 | 141 | 138 | 132 | 128 | 116 | 103 | 92 | 84 | 60 |
| $\Theta$ ($\cdot 10^3$) | 22.0 | 22.0 | 22.0 | 21.9 | 19.6 | 21.8 | 19.3 | 19.6 | 22.8 | 18.9 |

Table 5.1: Delay, complexity and memory of NUPBFDC.

For the NUPBFDC the maximum allowable processing delay of 0.5 milli-seconds results in $D_{\max} = 0.5 \cdot 10^{-3} \cdot f_s = 4$. In tables 5.2 and 5.3

[9] In the cases considered in this thesis, even the largest FFT length $M_{G-1}$ is at most $N/4$, see also the remarks in connection with equation (4.18).

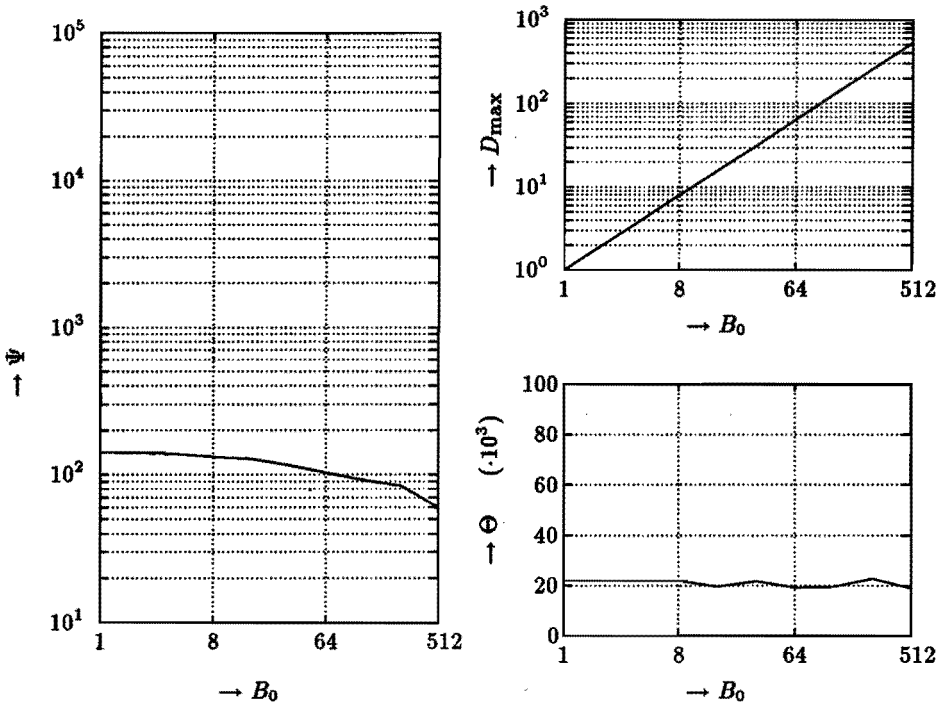



Figure 5.10: Properties of NUPBFDC.

the complexity $\Psi_{\text{NUPBFDC}}$ is given as a function of the number of different partition factors $G$ in this case. For all $j$, $0 \leq j < G$, the parameters $B_j$, $Q_j$, $g_j$ and $M_j$ are chosen to minimize the complexity $\Psi_{\text{NUPBFDC}}$. From

| Algorithm | PBFDC | NUPBFDC | | | | |
|---|---|---|---|---|---|---|
| # part. fac. $G$ | 1 | 2 | | 3 | | |
| Subset index $j$ | 0 | 0 | 1 | 0 | 1 | 2 |
| Block length $B_j$ | 4 | 4 | 128 | 4 | 32 | 256 |
| Part. length $Q_j$ | 60 | 12 | 384 | 4 | 32 | 256 |
| FFT length $M_j$ | 64 | 16 | 512 | 8 | 64 | 512 |
| # subfilters $g_j$ | 67 | 11 | 11 | 7 | 8 | 15 |
| $\Psi_{\text{NUPBFDC}}$ | 2209 | 228 | | 154 | | |
| $\frac{\Psi_{\text{PBFDC}}}{\Psi_{\text{NUPBFDC}}}$ | 1.00 | 9.67 | | 14.3 | | |

Table 5.2: Complexity using 1 to 3 partition factors.

| Algorithm | NUPBFDC | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| # part. fac. $G$ | 4 | | | | 5 | | | | |
| Subset index $j$ | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 4 |
| Block length $B_j$ | 4 | 16 | 64 | 512 | 4 | 8 | 32 | 128 | 512 |
| Part. length $Q_j$ | 4 | 16 | 64 | 512 | 4 | 8 | 32 | 128 | 512 |
| FFT length $M_j$ | 8 | 32 | 128 | 1024 | 8 | 16 | 64 | 256 | 1024 |
| # subfilters $g_j$ | 3 | 4 | 8 | 7 | 1 | 3 | 4 | 3 | 7 |
| $\Psi_{\text{NUPBFDC}}$ | 140 | | | | 138 | | | | |
| $\frac{\Psi_{\text{PBFDC}}}{\Psi_{\text{NUPBFDC}}}$ | 15.7 | | | | 16.0 | | | | |

Table 5.3: Complexity using 4 or 5 partition factors.

table 5.3 it can be seen that the number of real multiplications per sample can be reduced by a factor 16 compared to a uniform partitioned frequency domain convolution for this example. A further increase in the number of different sub-filter lengths $G$ does not increase $\Psi_{\text{PBFDC}}/\Psi_{\text{NUPBFDC}}$ any more.

The maximum of $\Psi_{\text{PBFDC}}/\Psi_{\text{NUPBFDC}}$ depends strongly on the value of $D_{\max}$ and $N$. In figure 5.11 this dependency is depicted for different $N$ while $D_{\max}$ is varied between 1 and $N$. This figure shows that for $D_{\max} < N/8$ the introduced non-uniform partitioning concept is very usefull. Finally from figure 5.12, where the computational complexity $\Psi_{\text{NUPBFDC}}$ is depicted as function of $D_{\max}$ and $N$, it follows that an increase of the number of

filter coefficients $N$ by a factor 4 cost approximately 40 real multiplies per sample, independent of $N$ and $D_{\max}$.

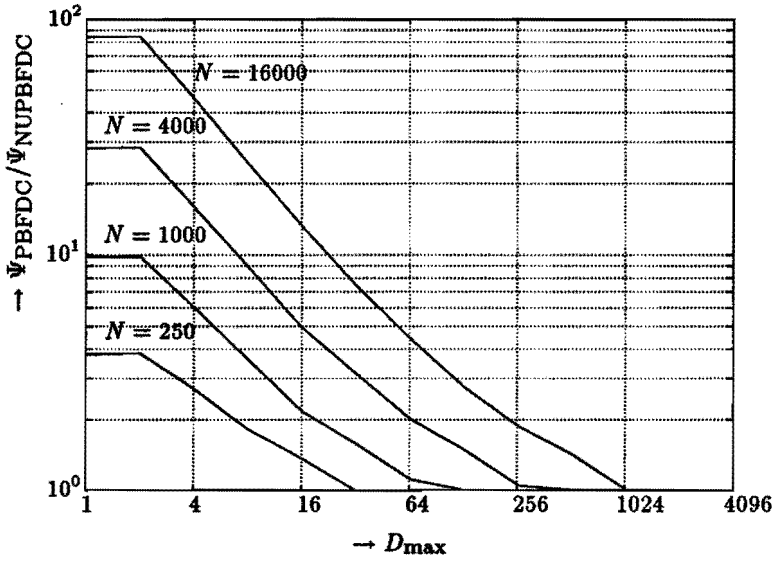



Figure 5.11: Complexity gain of NUPBFDC over PBFDC.

## 5.2 Non-Uniform Partitioned BFDAF

### 5.2.1 Filter part for NUPBFDAF

The filter part of the Non-Uniform PBFDAF (NUPBFDAF) adaptive filter is constructed according to the NUPBFDC method introduced in the previous section. However, the coefficients of the filter are not fixed but have to be adapted. If we assume that all block lengths $B_j$ with $0 < j < G$ are integer multiples of $B_0$, then the residual signal is constructed as follows

$$\underline{r}^{B_0}[\kappa_0 B_0] = \underline{\tilde{e}}^{B_0}[\kappa_0 B_0] - \underline{\hat{e}}^{B_0}[\kappa_0 B_0] \tag{5.21}$$

$$\underline{\tilde{e}}^{B_0}[\kappa_0 B_0] = \begin{pmatrix} \tilde{e}[\kappa_0 B_0 - D_{\max}] \\ \cdots \\ \tilde{e}[\kappa_0 B_0 - D_{\max} + B_0 - 1] \end{pmatrix} \tag{5.22}$$

$$\underline{\hat{e}}^{B_0}[\kappa_0 B_0] = \begin{pmatrix} \hat{e}[\kappa_0 B_0 - D_{\max}] \\ \cdots \\ \hat{e}[\kappa_0 B_0 - D_{\max} + B_0 - 1] \end{pmatrix} \tag{5.23}$$

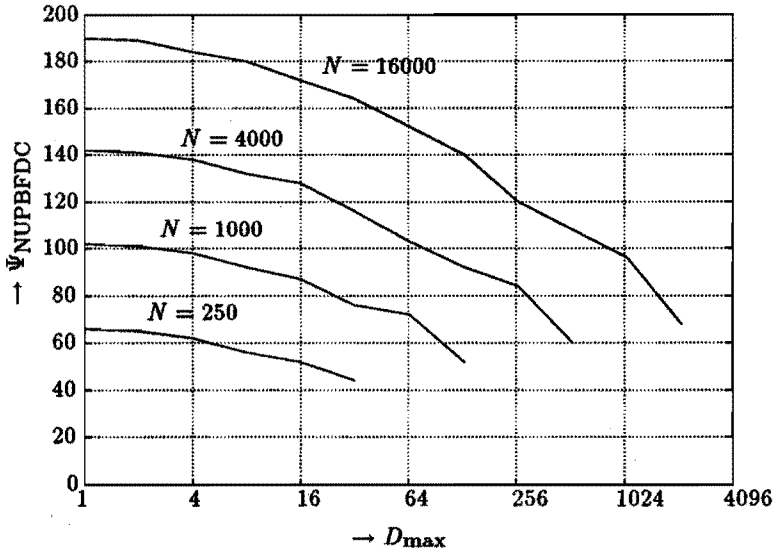



Figure 5.12: Complexity of NUPBFDC.

In figure 5.13 the above equations are combined with the NUPBFDC. The subfilters can be found in figure 5.8. The parallel to parallel converters (denoted by "P/P") are needed for conversion from one sample rate and block length to another. The change in block length is performed by splitting each block of length $B_j$ (that arrives every $B_j \cdot \mathrm{T}$ seconds) into $B_j/B_0$ blocks of length $B_0$. The time between two blocks is decreased to $B_0 \cdot \mathrm{T}$ seconds, so all blocks of length $B_0$ can be transmitted before the new block of length $B_j$ arrives. Mathemathically this combination to blocks of length $B_0$ can be described as

$$\underline{\hat{e}}^{B_0}[\kappa_0 B_0] = \sum_{j=0}^{G-1} \underline{\hat{e}}_j^{B_0}[\kappa_0 B_0] \tag{5.24}$$

where for $0 \leq j < G$

$$\begin{aligned} \underline{\hat{e}}_j^{B_0}[\kappa_0 B_0] &= \left( \mathbf{0}^{B_0,\Omega_j[\kappa_0 B_0]} \quad \mathbf{I}^{B_0} \quad \mathbf{0}^{B_0,B_j-B_0-\Omega_j[\kappa_0 B_0]} \right) \\ &\quad \cdot \underline{\hat{e}}_j^{B_j}[\lfloor \frac{\kappa_0 B_0}{B_j} \rfloor B_j] \end{aligned} \tag{5.25}$$

with the position of the window defined by $\Omega_j[\kappa_0 B_0]$

$$\Omega_j[\kappa_0 B_0] = \kappa_0 B_0 - \lfloor \frac{\kappa_0 B_0}{B_j} \rfloor \cdot B_j \tag{5.26}$$

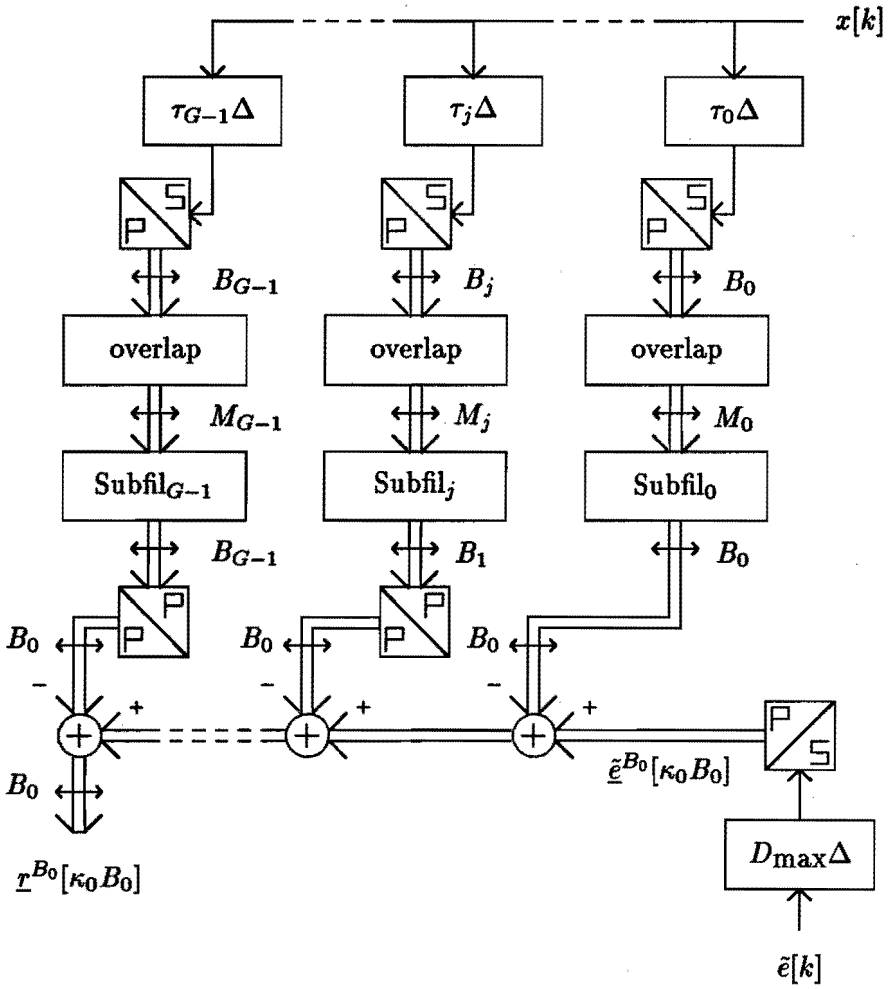



Figure 5.13: Filter part for NUPBFDAF.

### 5.2.2 Update part for NUPBFDAF

The update rate for the adaptive coefficients is chosen in such a way that all filter part rates are integer multiples of the update rate, which implies that the update part block length $A$ is an integer multiple of the largest filter part block length $B_{G-1}$. A modified version of the DPBFDAF update part of section 4.2 is used for the NUPBFDAF.

As the processing delay of the filter part equals $D_{\max}$ sample intervals, the residual signal will be delayed. This delay is taken into account in the update equations by a delay $\tau_{\text{up}}$. The update equation of a (delayed) BOP with block length $A$ is partitioned in $g_{\text{U}} = \lceil N/Z \rceil$ parts.[10] For $0 \leq i < g_{\text{U}}$ this can be described by

$$\underline{w}_i^Z[(l+1)A] = \underline{w}_i^Z[lA] + \mathcal{X}^{A,Z}[lA - iZ - \tau_{\text{up}}] \cdot \underline{y}^A[lA] \tag{5.27}$$

with

$$\underline{y}^A[lA] = 2\alpha(\hat{\mathcal{R}}_x^A[lA])^{-1} \cdot \underline{r}^A[lA]. \tag{5.28}$$

The residual signal is constructed as follows

$$\begin{aligned} \underline{r}^A[lA] &= \begin{pmatrix} \underline{r}^{B_0}[lA - A + B_0] \\ \cdots \\ \underline{r}^{B_0}[lA - B_0] \\ \underline{r}^{B_0}[lA] \end{pmatrix} \\ &= \begin{pmatrix} r[lA - A + B_0 - D_{\max}] \\ \cdots \\ r[lA + B_0 - 1 - D_{\max}] \end{pmatrix}. \end{aligned} \tag{5.29}$$

From equations (5.29) and (5.27) follows the correct value for $\tau_{\text{up}}$

$$\tau_{\text{up}} = D_{\max} - B_0 + 1. \tag{5.30}$$

Equation (5.27) can be implemented efficiently with DFTs like in the DPBFDAF case. For $0 \leq i < g_{\text{U}}$ this can be described by

$$\underline{Y}^L[lA] = 2\alpha(\hat{\underline{P}}_x^L[lA])^{-1} \otimes \underline{R}^L[lA] \tag{5.31}$$

$$\begin{aligned} \underline{w}_i^Z[(l+1)A] &= \underline{w}_i^Z[lA] + \begin{pmatrix} \mathbf{J}^Z & \mathbf{0}^{Z,L-Z} \end{pmatrix} \cdot (\mathcal{F}^L)^{-1} \\ &\quad \cdot ((\underline{X}_i^L[lA])^* \otimes \underline{Y}^L[lA]) \end{aligned} \tag{5.32}$$

$$\underline{R}^L[lA] = \mathcal{F}^L \cdot \begin{pmatrix} \underline{0}^{L-A} \\ \underline{r}^A[lA] \end{pmatrix} \tag{5.33}$$

[10] Also here the coefficient vector $\underline{w}^N$ is extended if $\lceil N/Z \rceil \cdot Z > N$ by $\lceil N/Z \rceil \cdot Z - N$ coefficients that are kept zero (as in DPBFDAF).

with

$$\underline{X}_i^L[lA] = \mathcal{F}^L \cdot \underline{x}^L[lA - iZ - \tau_{\text{up}}]. \tag{5.34}$$

The new Non-Uniform Partitioned Block Frequency Domain Adaptive Filter (NUPBFDAF) can now be obtained by combining the results of the two previous sections. The filter vectors are coupled by the Fourier-transforms and the "Hold"-boxes, as in the previous chapter with the DPBFDAF. In figure 5.14 the resulting update part with coupling of the adaptive coefficients is depicted.

### 5.2.3 Properties of NUPBFDAF

With help of the properties of the NUPBFDC and DPBFDAF the next properties for the NUPBFDAF can be derived:

- Algorithm and computation processing delay

$$D_{\text{NUPBFDAF}} + \Delta_{\text{NUPBFDAF}} \leq D_{\max}. \tag{5.35}$$

- Computational complexity: if the largest filter part FFT, block length and partition length equal the corresponding update part parameters, one FFT can be omitted. The resulting sum of FFTs, IFFTs and elementwise multplications equals

$$\begin{aligned}\Psi_{\text{NUPBFDAF}} &= \frac{(\Xi + g_{\text{U}}) \cdot \Psi_{\text{FFT}}\{L\} + g_{\text{U}} \cdot \Psi_{\otimes}\{L\} + \Psi_P\{L\}}{A} \\ &\quad + \sum_{j=0}^{G-1} (\Psi_{\text{NUPBFDC},j} + \frac{g_j \cdot \Psi_{\text{FFT}}\{M_j\}}{A}) \end{aligned} \tag{5.36}$$

$$\Psi_{\text{NUPBFDC},j} = \frac{2 \cdot \Psi_{\text{FFT}}\{M_j\} + g_j \cdot \Psi_{\otimes}\{M_j\}}{B_j} \tag{5.37}$$

where

$$\Xi = \begin{cases} 1 & \text{if } L = M_{G-1},\ A = B_{G-1} \text{ and } Z = Q_{G-1} \\ 2 & \text{otherwise} \end{cases}. \tag{5.38}$$

- Memory occupation (compare to DPBFDAF)

$$\begin{aligned}\Theta_{\text{NUPBFDAF}} &\approx \sum_{j=0}^{G-1} ((3 + (g_j - 1) \cdot (q_j + 1)) \cdot M_j + B_j) \\ &\quad + (\frac{7}{2} + (g_{\text{U}} - 1) \cdot z) \cdot L + A + N. \end{aligned} \tag{5.39}$$

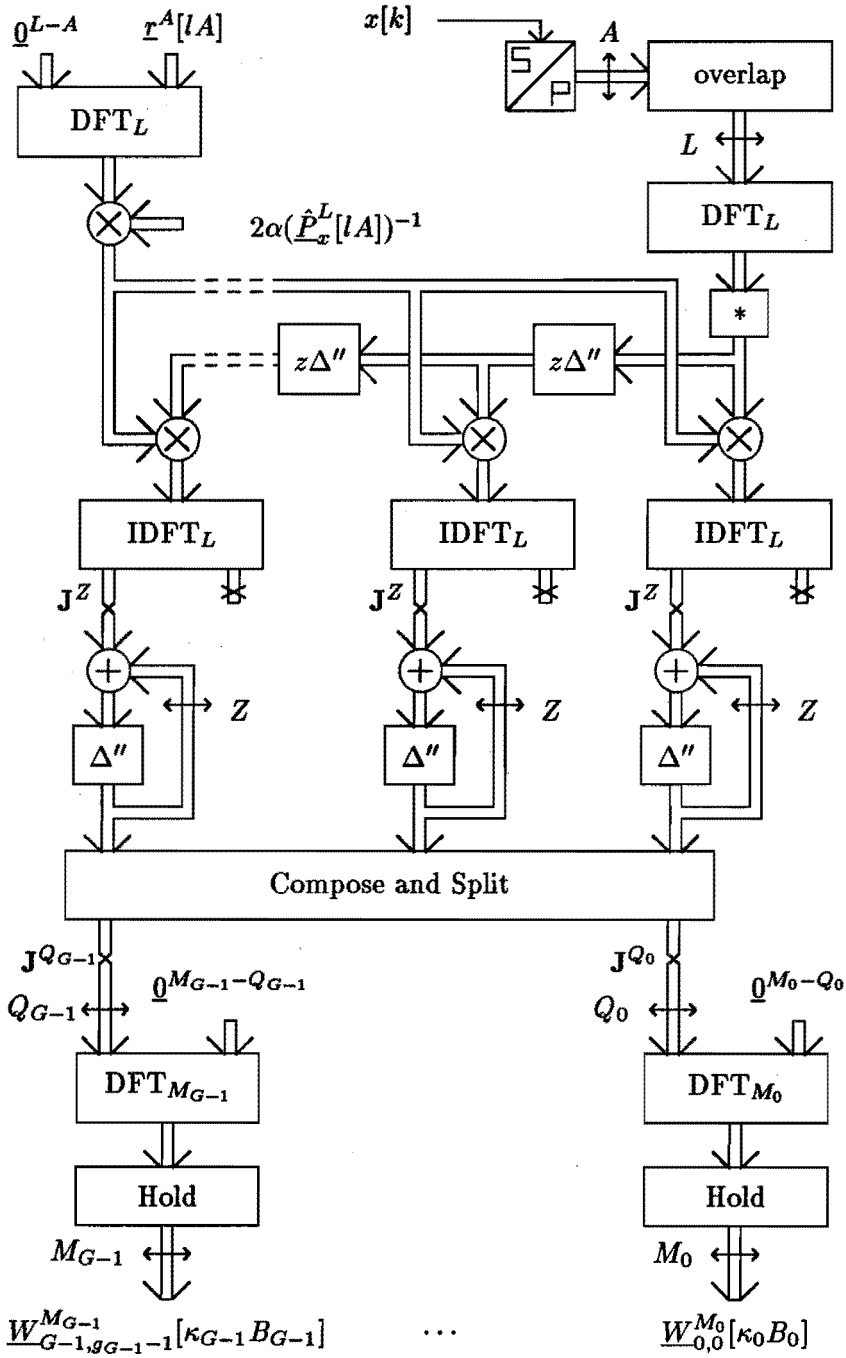



Figure 5.14: Update part of NUPBFDAF.

If for all $j$, $0 \leq j < G$, $M_j$ is much smaller than $N$,[11] and $A = Z$,[12] this can be approximated by

$$\Theta_{\text{NUPBFDAF}} \approx 2 \cdot N + 6 \cdot A + \sum_{j=0}^{G-1} (2 + q_j + \frac{1}{q_j}) \cdot g_j Q_j. \qquad (5.40)$$

### 5.2.4 Convergence and Tracking

If we compare DPBFDAF with NUPBFDAF the update part is equal. The larger block lengths in the filter part of NUPBFDAF imply the use of an older adaptive filter vector $\underline{w}^N$. As the update part block length $A$ is an integer multiple of the largest filter part block length (and of all other filter part block lengths), and the filter and update part are synchronized, no update of that adaptive filter vector takes place during calculation of the filter part blocks. This means that the older adaptive filter vector $\underline{w}^N$ in NUPBFDAF equals the one used in DPBFDAF. This implies that convergence and tracking properties of NUPBFDAF exactly equal those of DPBFDAF, when the update parameters are identical and $B = B_0$.

### 5.2.5 Example of NUPBFDAF

Our example illustrates the relation between delay and complexity. In table 5.4 and figure 5.15 the NUPBFDAF complexity, delay and memory are given as a function of the block length $B_0$ (determining the algorithm processing delay) for a filter length $N = 4000$. The number of different partition factors and for $0 \leq j < G$, the parameters $B_j$, $Q_j$, $g_j$ and $M_j$ (except for $B_0$) are chosen to minimize the complexity $\Psi_{\text{NUPBFDAF}}$. For all filters the update part FFT length $L$ equals 1024 (the update block length $A$ (and the partition length $Z$) equal the largest integer multiple of all $B_j$ smaller than or equal to 512, which is 512 in all cases considered here).

Like in the NUPBFDC also here the maximum allowable processing delay of 0.5 milli-seconds results in $D_{\max} = 0.5 \cdot 10^{-3} \cdot f_S = 4$. In tables 5.5 and 5.6 the complexity $\Psi_{\text{NUPBFDAF}}$ is given as a function of the number of different partition factors $G$ in this case. For $0 \leq j < G$, the parameters $B_j$, $Q_j$, $g_j$ and $M_j$ are chosen to minimize the complexity $\Psi_{\text{NUPBFDAF}}$. Also here for all filters the update part FFT length $L$ equals 1024 (the update block length $A$ (and the partition length $Z$) equal the largest integer multiple of all $B_j$ smaller than or equal to 512).

[11] In the cases considered in this thesis, even the largest FFT length $M_{G-1}$ is at most $N/4$, see also the remarks in connection with equation (4.18).

[12] For every case considered in this thesis, this is true.

| $B_0$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\Delta + D$ | 1 | 2 | 4 | 8 | 16 | 32 | 65 | 131 | 263 | 529 |
| $G$ | 6 | 5 | 5 | 4 | 3 | 3 | 2 | 2 | 2 | 1 |
| $\Psi$ | 416 | 415 | 411 | 406 | 401 | 389 | 376 | 366 | 364 | 338 |
| $\Theta$ $(\cdot 10^3)$ | 36.8 | 36.8 | 36.8 | 36.8 | 34.4 | 36.6 | 34.1 | 34.4 | 37.6 | 19.5 |

Table 5.4: Delay, complexity and memory of NUPBFDAF.

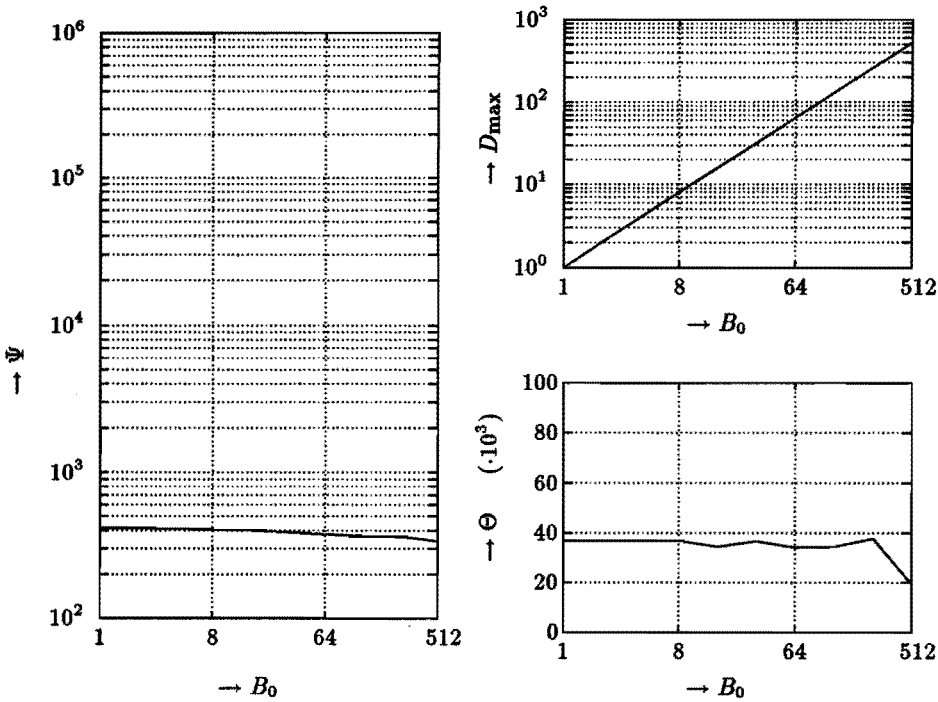



Figure 5.15: Properties of NUPBFDAF.

| Algorithm | DPBFDAF | NUPBFDAF | | | | |
|---|---|---|---|---|---|---|
| # part. fac. $G$ | 1 | 2 | | 3 | | |
| Subset index $j$ | 0 | 0 | 1 | 0 | 1 | 2 |
| Block length $B_j$ | 4 | 4 | 128 | 4 | 64 | 512 |
| Part. length $Q_j$ | 60 | 12 | 384 | 12 | 64 | 512 |
| FFT length $M_j$ | 64 | 16 | 512 | 16 | 128 | 1024 |
| # subfilters $g_j$ | 67 | 11 | 11 | 6 | 8 | 7 |
| $\Psi_{\text{NUPBFDAF}}$ | 2414 | 475 | | 431 | | |
| $\frac{\Psi_{\text{DPBFDAF}}}{\Psi_{\text{NUPBFDAF}}}$ | 1.00 | 5.08 | | 5.60 | | |

Table 5.5: Complexity using 1 to 3 partition factors.

| Algorithm | NUPBFDAF | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| # part. fac. $G$ | 4 | | | | 5 | | | | |
| Subset index $j$ | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 4 |
| Block length $B_j$ | 4 | 16 | 64 | 512 | 4 | 8 | 32 | 128 | 512 |
| Subfil. length $Q_j$ | 4 | 16 | 64 | 512 | 4 | 8 | 32 | 128 | 512 |
| FFT length $M_j$ | 8 | 32 | 128 | 1024 | 8 | 16 | 64 | 256 | 1024 |
| # subfilters $g_j$ | 3 | 4 | 8 | 7 | 1 | 3 | 4 | 3 | 7 |
| $\Psi_{\text{NUPBFDAF}}$ | 413 | | | | 411 | | | | |
| $\frac{\Psi_{\text{DPBFDAF}}}{\Psi_{\text{NUPBFDAF}}}$ | 5.85 | | | | 5.87 | | | | |

Table 5.6: Complexity using 4 or 5 partition factors.

Tables 5.5 and 5.6 show that the average number of real multiplications per sample can be reduced by a factor $\Psi_{\text{DPBFDAF}}/\Psi_{\text{NUPBFDAF}} = 5.87$ for this example. A further increase in the number of different sub-filter lengths $G$ does not increase $\Psi_{\text{DPBFDAF}}/\Psi_{\text{NUPBFDAF}}$. The maximum of $\Psi_{\text{DPBFDAF}}/\Psi_{\text{NUPBFDAF}}$ depends strongly on the value of $D_{\max}$ and $N$. In figure 5.16 this dependency is depicted for different $N$ while $D_{\max}$ is varied between 1 and $N/8$. The update part block length $A$ and partition length $Z$ equal the largest integer multiple of all $B_j$ smaller than or equal to $N/8$. The figure 5.16 shows that for $D_{\max} < N/8$ the introduced non-uniform partitioning concept is usefull.

Figure 5.17, where the computational complexity $\Psi_{\text{NUPBFDAF}}$ is depicted as function of $D_{\max}$ and $N$, shows that an increase of the number of filter coefficients $N$ by a factor 4 cost approximately 100 real multiplies per sample independent of $N$ and $D_{\max}$. For small $D_{\max}$ the next rule of thumb for the number of real multiplications can be deduced from figure 5.17

$$\Psi_{\text{NUPBFDAF}} \approx 50 \cdot \log_2(N) - 190. \tag{5.41}$$

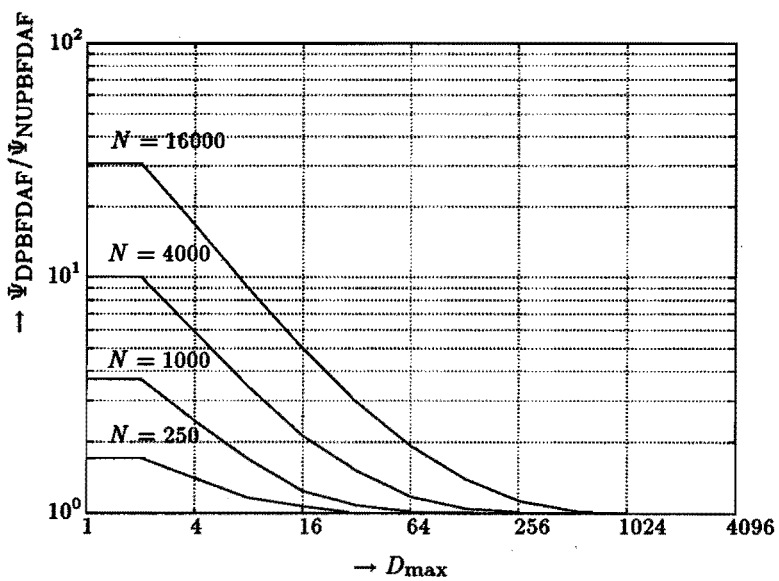



Figure 5.16: Complexity gain of NUPBFDAF over DPBFDAF.

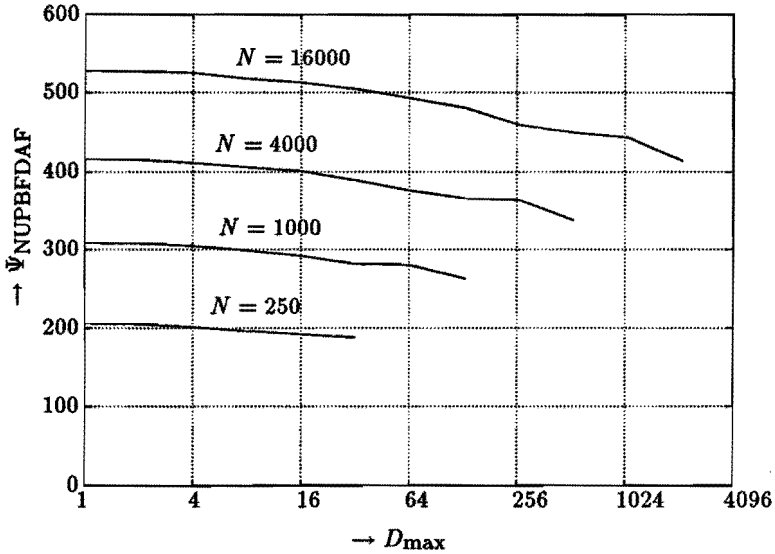



Figure 5.17: Complexity of NUPBFDAF.

## 5.3 Conclusions

The introduced new method for fast real time convolution in frequency domain by using a non-uniform partitioning reduces the required number of real multiplications per sample compared to an approach using uniform partitioning.

For the case of adaptive filtering the computational complexity can be reduced enormously by using NUPBFDAF, compared to implementations using (D)(P)BFDAF. Complexity becomes almost independent of the maximum allowable delay. As a rule of thumb the number of real multiplications per sample equals approximately $50 \cdot \log_2(N) - 190$, with $N$ the number of filter coefficients.

# Chapter 6

# Normalization

*Convergence characteristics of adaptive filters depend on the input signal variance and (auto-)correlation. To remove these dependencies several methods are used. Normalization in time domain removes the dependency on the input signal variance with a division of the adaptation constant $\alpha$ by this input signal variance. This input variance therefore must be estimated. The division needed in this normalization can be omitted by making a direct estimate of the inverse of the input signal variance. Full decorrelation in time domain can be obtained by multiplying the gradient vector by the inverse autocorrelation matrix of the input signal. Efficient algorithms to perform such an operation still need a lot of computational power.*

*In frequency domain normalization of the update by the input power vector reduces the dependency of convergence characteristics on the input signal autocorrelation. Direct estimation of the inverse power vector reduces computational complexity as divisions can be avoided. This estimate of the power vector however does not always match the actual input signal power.*

*In the previous chapters it was assumed that the normalization always takes place by multiplying the frequency domain transform of the residual signal by the inverse power spectrum. In the partitioned approach this normalization can take place in three ways: we can normalize the residual signal, the input signal or both of them.*

*At cost of some small extra computational complexity, the power vector dimension and rate can be chosen independently of the update part dimensions and rate. This yields a more flexible algorithm, which can be of use in cases where the non-stationarity of the input signal is the main limiting factor.*

## 6.1 NLMS

By normalizing the update of the LMS algorithm the dependency of its convergence behaviour on the input signal variance is removed. This normalization procedure requires per sample an estimation of the input signal variance and a division (see equation (2.16)). As $\mathcal{E}\{x[k]\} = 0$, the variance (power) $\sigma_x^2[k] = \mathcal{E}\{(x[k])^2\}$ of the input signal $x[k]$ can be estimated for $0 < \beta < 1$ by

$$\hat{\sigma}_x^2[k] = (1-\beta)\hat{\sigma}_x^2[k-1] + \beta(x[k])^2. \tag{6.1}$$

The parameter $\beta$ controls the dynamic behaviour of the above equation. Decreasing $\beta$ implies that the past of the input signal becomes more important in the estimate of the variance, which implies on one hand that the estimate is smoothened, and, on the other hand, that (fast) variations in the input signal variance cannot be followed by the estimator.

To avoid the use of divisions, a direct estimate for the inverse variance can be used. With help of equation (6.1), we get for $0 < \beta < 1$

$$\begin{aligned} \hat{\sigma}_x^{-2}[k] &= \frac{1}{(1-\beta)\hat{\sigma}_x^2[k-1] + \beta(x[k])^2} \\ &= \frac{\frac{1}{1-\beta}\hat{\sigma}_x^{-2}[k-1]}{1 + \frac{\beta}{1-\beta}\hat{\sigma}_x^{-2}[k-1](x[k])^2} \end{aligned} \tag{6.2}$$

By using $1+\xi \approx 1/(1-\xi)$ for small $\xi$, the above is approximated for $\beta \ll 1$ and $\frac{\beta}{1-\beta} \cdot \hat{\sigma}_x^{-2}[k-1](x[k])^2 \ll 1$ by

$$\hat{\sigma}_x^{-2}[k] \approx (1+\beta) \cdot \hat{\sigma}_x^{-2}[k-1] \cdot (1 - \frac{\beta}{1-\beta} \cdot \hat{\sigma}_x^{-2}[k-1](x[k])^2). \tag{6.3}$$

By approximating $\beta(1+\beta)/(1-\beta)$ by $\beta$, we finally get

$$\hat{\sigma}_x^{-2}[k] \approx (1+\beta) \cdot \hat{\sigma}_x^{-2}[k-1] - \beta \cdot (\hat{\sigma}_x^{-2}[k-1] \cdot x[k])^2. \tag{6.4}$$

To ensure stability in the above direct inverse variance estimate, we have to avoid that $\beta \cdot (\hat{\sigma}_x^{-2}[k-1] \cdot x[k])^2$ is larger than than $(1+\beta) \cdot \hat{\sigma}_x^{-2}[k-1]$. The actual inverse power estimation algorithm therefore is given with a certain (small) positive threshold $\epsilon$ and $0 < \beta \ll 1$ by

$$\varsigma^{-2}[k] = (1+\beta) \cdot \hat{\sigma}_x^{-2}[k-1] - \beta \cdot (\hat{\sigma}_x^{-2}[k-1] \cdot x[k])^2 \tag{6.5}$$

$$\hat{\sigma}_x^{-2}[k] = \begin{cases} \epsilon & \text{for } \varsigma^{-2}[k] < \epsilon \\ \varsigma^{-2}[k] & \text{otherwise} \end{cases} . \tag{6.6}$$

The above equation requires 4 multiplications, which implies for the total extra computational complexity for normalization[1]

$$\Psi_\sigma = 5. \tag{6.7}$$

## 6.2 BNLMS

In the BNLMS case we can use the same estimate for the variance as in the NLMS case. A block processing procedure can also be used, by accumulating $B$ variance updates (of equation (6.1)). Therefore we approximate $(1-\beta)^B$ by $(1-B\beta)$ and $\beta(1-\beta)^i$ by $\beta$ for $0 < i < B$. This gives the following block procedure for direct variance estimation, with $0 < B \cdot \beta < 1$

$$\hat{\sigma}_x^2[\kappa B] = (1 - B \cdot \beta)\hat{\sigma}_x^2[(\kappa - 1)B] + \beta \cdot (\underline{x}^B[\kappa B])^t \underline{x}^B[\kappa B]. \tag{6.8}$$

Like in the previous section, a direct estimate for the inverse variance can be obtained using a certain (small) positive threshold $\epsilon$ for $0 < B \cdot \beta \ll 1$ by

$$\begin{aligned} \varsigma^{-2}[\kappa B] &= (1 + B \cdot \beta) \cdot \hat{\sigma}_x^{-2}[(\kappa - 1)B] \\ &\quad -\beta \cdot (\hat{\sigma}_x^{-2}[(\kappa - 1)B])^2 \cdot (\underline{x}^B[\kappa B])^t \underline{x}^B[\kappa B] \end{aligned} \tag{6.9}$$

$$\hat{\sigma}_x^{-2}[k] = \begin{cases} \epsilon & \text{for } \varsigma^{-2}[\kappa B] < \epsilon \\ \varsigma^{-2}[\kappa B] & \text{otherwise} \end{cases} . \tag{6.10}$$

The above estimation algorithm requires $B + 4$ multiplications, which implies for the total extra computational complexity for normalization[2]

$$\Psi_\sigma\{B\} = 2B + 4. \tag{6.11}$$

## 6.3 Frequency Domain Normalization

### 6.3.1 BOP as Basis

#### Circulant Matrices

The BNLMS algorithm can be performed efficiently in frequency domain with FFTs. To improve convergency behaviour we want to incorporate the

[1] We multiply the residual signal $r[k]$ by $2\alpha\hat{\sigma}_x^{-2}[k]$, taking 1 multiply (as $2\alpha$ can be incorporated in the variance estimation algorithm).

[2] When the block length $B$ is smaller than the filter length $N$, we multiply the residual signal vector $\underline{r}^B[\kappa B]$ of the BNLMS update equation by the inverse variance estimate. This takes $B$ multiplications (as also here $2\alpha$ can be incorporated in the variance estimation algorithm). If $B > N$, we multiply $\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B]$ by the inverse variance estimate, this implies that in fact $\Psi_\sigma\{B\} = B + \min\{N, B\} + 4$.

BOP or BRLS decorrelation properties in this frequency domain algorithm. We start with the BOP update equation, given by[3]

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + 2\alpha \mathcal{X}^{N,B}[\kappa B](\hat{\mathcal{R}}_x^B[\kappa B])^{-1}\underline{r}^B[\kappa B]. \tag{6.13}$$

To perform the above BOP update equation efficiently in frequency domain, the inverse autocorrelation matrix $(\hat{\mathcal{R}}_x^B[\kappa B])^{-1}$ must be approximated by a circulant matrix $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ and the input signal matrix $\mathcal{X}^{N,B}[\kappa B]$ must be transformed to a circulant matrix $(\check{\mathcal{X}}^M[\kappa B])^t$, like in the FBNLMS case of chapter 3. This results in a (time domain) update equation with circulant matrices

$$\begin{aligned}\underline{w}^N[(\kappa+1)B] &= \underline{w}^N[\kappa B] + 2\alpha \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} \\ &\quad \cdot(\check{\mathcal{X}}^M[\kappa B])^t(\check{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}. \end{aligned} \tag{6.14}$$

Algorithms (6.13) and (6.14) will behave identically when there is a matrix $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ such that

$$\begin{aligned} &\mathcal{X}^{N,B}[\kappa B](\hat{\mathcal{R}}_x^B[\kappa B])^{-1}\underline{r}^B[\kappa B] \\ &= \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}^M[\kappa B])^t(\check{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}. \end{aligned} \tag{6.15}$$

As the above condition (6.15) must be be valid for all $\underline{r}^B[kB]$, we can eleminate the residual signal on both sides. By replacing the input signal matrix by its circulant version, and multiplying by $\mathbf{J}^N$, we get

$$\begin{aligned} &\begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}^M[\kappa B])^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} (\hat{\mathcal{R}}_x^B[\kappa B])^{-1} \\ &= \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}^M[\kappa B])^t(\check{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix}. \end{aligned} \tag{6.16}$$

Knowing that condition (6.16) must be fulfilled for all circulant matrices $(\check{\mathcal{X}}^M[\kappa B])^t$ the input signal matrix $(\check{\mathcal{X}}^M[\kappa B])^t$ can be discarded, as will be

[3]In partitioned structures, the partitioned BOP formula is given for $0 \leq i < g_U$ by

$$\underline{w}_i^Z[(l+1)A] = \underline{w}_i^Z[lA] + 2\alpha \mathcal{X}^{Z,A}[lA - iZ](\hat{\mathcal{R}}_x^A[lA])^{-1}\underline{r}^A[lA]. \tag{6.12}$$

The only differences are the change in parameters and that we have $g_U$ equations instead of one, meaning that this section also holds for partitioned structures, by replacing the parameters.

shown hereafter, leading to

$$\begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} (\hat{\mathcal{R}}_x^B[\kappa B])^{-1} = (\check{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix}. \tag{6.17}$$

**Elemination of the Input Signal**

It is clear that (6.17) is a sufficient condition, we will show that it is also necessary. Suppose that there is a matrix $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ that fulfills condition (6.16) for all circulant matrices $(\check{\mathcal{X}}^M[\kappa B])^t$,[4] thus also for the circulant matrices $(\check{\mathcal{X}}_i^M)^t$, with

$$(\check{\mathcal{X}}_i^M)^t = \begin{pmatrix} \mathbf{0}^{M-i,i} & \mathbf{I}^{M-i} \\ \mathbf{I}^i & \mathbf{0}^{i,M-i} \end{pmatrix}. \tag{6.18}$$

By substituting these matrices in condition (6.16) we get for all $0 \leq i < M$

$$\begin{aligned} &\begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} (\hat{\mathcal{R}}_x^B[\kappa B])^{-1} \\ &= \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t (\check{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix}. \end{aligned} \tag{6.19}$$

To continue, we first define a matrix $\mathbf{U}_i^{M,N}$, with

$$\mathbf{U}_i^{M,N} = \check{\mathcal{X}}_i^M \cdot \begin{pmatrix} 1 & (\underline{0}^{N-1})^t \\ \underline{0}^{M-1} & \mathbf{0}^{M-1,N-1} \end{pmatrix}. \tag{6.20}$$

The following property holds for this matrix

$$\begin{aligned} \mathbf{U}_i^{M,N} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t &= \check{\mathcal{X}}_i^M \begin{pmatrix} 1 & (\underline{0}^{M-1})^t \\ \underline{0}^{M-1} & \mathbf{0}^{M-1} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t \\ &= \mathrm{diag}\{\begin{pmatrix} \underline{0}^i \\ 1 \\ \underline{0}^{M-1-i} \end{pmatrix}\} \end{aligned} \tag{6.21}$$

which implies that

$$\begin{aligned} \sum_{i=0}^{M-1} \mathbf{U}_i^{M,N} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t &= \sum_{i=0}^{M-1} \mathrm{diag}\{\begin{pmatrix} \underline{0}^i \\ 1 \\ \underline{0}^{M-1-i} \end{pmatrix}\} \\ &= \mathbf{I}^M. \end{aligned} \tag{6.22}$$

[4] The transpose of a circulant matrix is also circulant.

By a left-hand side multiplication of condition (6.19) by the matrix $\mathbf{U}_i^{M,N}$ and an accumulation, we get

$$\left(\sum_{i=0}^{M-1} \mathbf{U}_i^{M-N} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t\right) \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} (\hat{\mathcal{R}}_x^B[\kappa B])^{-1} \quad (6.23)$$
$$= \left(\sum_{i=0}^{M-1} \mathbf{U}_i^{M-N} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t\right) (\check{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix}.$$

With help of property (6.22), we obtain condition (6.17), implying that it is indeed necessary.

### Approximation

By multiplying both sides of condition (6.17) by the autocorrelation matrices, we get

$$\check{\mathcal{R}}_x^M[\kappa B] \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} = \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} \hat{\mathcal{R}}_x^B[\kappa B]. \quad (6.24)$$

The matrix on the right-hand side of condition (6.24) is part of a circulant matrix $\check{\mathcal{R}}_x^M[\kappa B]$. If there were such a circulant matrix, then the rows of the matrix $\begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix}^t \hat{\mathcal{R}}_x^B[\kappa B]$ on the right-hand side have to be shifted versions of one another, which is clearly not the case. We therefore have to make an approximation in order to obtain a circulant matrix. We will do this by averaging all elements in $\begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix}^t \hat{\mathcal{R}}_x^B[\kappa B]$ that should have been equal for the existance of a circulant matrix $\check{\mathcal{R}}_x^M[\kappa B]$ to fulfill condition (6.24).

As $\hat{\mathcal{R}}_x^B[\kappa B]$ is symmetric, also $\check{\mathcal{R}}_x^M[\kappa B]$ is supposed to be symmetric. For a symmetric circulant matrix $\check{\mathcal{R}}_x^M[\kappa B]$ we know that for all $0 \leq a < M$ and $0 \leq b < M$ there are $\check{\rho}_{|a-b|}[\kappa B]$, such that

$$(\check{\mathcal{R}}_x^M[\kappa B])_{a,b} = \check{\rho}_{|a-b|}[\kappa B] \quad (6.25)$$

where for all $0 < a < M$, $\check{\rho}_a[\kappa B] = \check{\rho}_{M-a}[\kappa B]$, so (by omitting the time

indices on the right-hand side)

$$\check{\mathcal{R}}_x^M[\kappa B] = \begin{pmatrix} \check{\rho}_0 & \check{\rho}_1 & \cdots & \check{\rho}_{\frac{M}{2}} & \cdots & \check{\rho}_1 \\ \check{\rho}_1 & \ddots & \ddots & & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \check{\rho}_{\frac{M}{2}} \\ \check{\rho}_{\frac{M}{2}} & & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & & \ddots & \ddots & \check{\rho}_1 \\ \check{\rho}_1 & \cdots & \check{\rho}_{\frac{M}{2}} & \cdots & \check{\rho}_1 & \check{\rho}_0 \end{pmatrix}. \tag{6.26}$$

The $B$ right-most columns of this matrix $\check{\mathcal{R}}_x^M[\kappa B]$ have to approximate $\begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix}^t \hat{\mathcal{R}}_x^B[\kappa B]$, so we suggest to average all elements in this matrix $\begin{pmatrix} \mathbf{0}^{B,M-B} & \mathbf{I}^B \end{pmatrix}^t \hat{\mathcal{R}}_x^B[\kappa B]$ that are supposed to be equal. With $(\hat{\mathcal{R}}_x^B[\kappa B])_{a,b} = \hat{\rho}_{|a-b|}[\kappa B]$, this implies for $B \geq \frac{M}{2}$ that[5]

$$\check{\rho}_i[\kappa B] = \begin{cases} \frac{B-i}{B}\hat{\rho}_i[\kappa B] & \text{for } 0 \leq i < M-B \\ \frac{B-i}{B}\hat{\rho}_i[\kappa B] + \frac{i-M+B}{B}\hat{\rho}_{M-i}[\kappa B] & \text{for } M-B \leq i < \frac{M}{2} \\ \frac{2B-M}{B}\hat{\rho}_{\frac{M}{2}} & \text{for } i = \frac{M}{2} \end{cases} \tag{6.27}$$

The main problem that remains is how to compute (the frequency domain transform of) the vector $\underline{\check{\rho}}^M[\kappa B]$ efficiently, with

$$\underline{\check{\rho}}^M[\kappa B] = \begin{pmatrix} \check{\rho}_0[\kappa B] & \cdots & \check{\rho}_{M-1}[\kappa B] \end{pmatrix} \tag{6.28}$$

and for $M/2 < i < M$, $\check{\rho}_i[\kappa B] = \check{\rho}_{M-i}[\kappa B]$. If we look at the power vector $\underline{P}_x^M[\kappa B]$ (see appendix C), defined as

$$\underline{P}_x^M[\kappa B] = \mathcal{E}\{\underline{X}^M[\kappa B] \otimes (\underline{X}^M[\kappa B])^*\} \tag{6.29}$$

and take the inverse DFT of an estimate (see appendix C, equation (C.8), we see a strong resemblance with equation (6.27), because

$$\left((\mathcal{F}^M)^{-1}\underline{\hat{P}}_x^M[\kappa B]\right)_i = (1 - \frac{i}{M}) \cdot \hat{\rho}_i[\kappa B] + \frac{i}{M}\hat{\rho}_{M-i}[\kappa B] \tag{6.30}$$

[5]For a reasonable approximation, the matrix $\mathbf{0}^{B,M-B}$ containing zeroes must be small compared to the autocorrelation matrix $\hat{\mathcal{R}}_x^B[\kappa B]$ (or the influence of $\hat{\mathcal{R}}_x^B[\kappa B]$ must be neglectible, but then we do not have any decorrelation at all). We therefore assume that $B \geq \frac{M}{2}$.

When $B \approx M$, which is the case when $B \gg N$, then $(B-i)/B \approx (M-i)/M$ (for $i < M/2$), then equation (6.27) is approximated by

$$\breve{\rho}_i[\kappa B] \approx \begin{cases} \frac{M-i}{M}\hat{\rho}_i[\kappa B] & \text{for } 0 \leq i < M-B \\ \frac{M-i}{M}\hat{\rho}_i[\kappa B] + \frac{i}{M}\hat{\rho}_{M-i}[\kappa B] & \text{for } M-B \leq i < \frac{M}{2} \\ \hat{\rho}_{\frac{M}{2}}[\kappa B] & \text{for } i = \frac{M}{2} \end{cases} \tag{6.31}$$

When $\hat{\rho}_i[\kappa B] \approx 0$ for $i > B$, then equation (6.31) and (6.30) are equal, which means that we may use $\breve{\mathcal{R}}_x^M[\kappa B]$, with (see appendix C, equation (C.11))

$$\breve{\mathcal{R}}_x^M[\kappa B] = \frac{1}{M}\mathcal{E}\{\breve{\mathcal{X}}^M[\kappa B] \cdot (\breve{\mathcal{X}}^M[\kappa B])^t\}. \tag{6.32}$$

### 6.3.2 Approximation Error

The quality of our approximation can be given as a function of the input signal correlation matrices. Suppose there is a not-necessarily circulant matrix $\tilde{\mathcal{R}}_x^M[\kappa B]$, fulfilling condition (6.17), so

$$(\tilde{\mathcal{R}}_x^M[\kappa B])^{-1} \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} = \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} (\hat{\mathcal{R}}_x^B[\kappa B])^{-1}. \tag{6.33}$$

Using $(\tilde{\mathcal{R}}_x^M[\kappa B])^{-1}$ instead of $(\breve{\mathcal{R}}_x^M[\kappa B])^{-1}$ would make the frequency domain algorithm behave exactly as BOP. We can construct a deviation matrix $\tilde{\mathcal{I}}^M[\kappa B]$, with

$$(\tilde{\mathcal{R}}_x^M[\kappa B])^{-1} = (\breve{\mathcal{R}}_x^M[\kappa B])^{-1}\tilde{\mathcal{I}}^M[\kappa B] \tag{6.34}$$

by choosing

$$\tilde{\mathcal{I}}^M[\kappa B] = \breve{\mathcal{R}}_x^M[\kappa B] \begin{pmatrix} \mathbf{I}^{M-B} & \mathbf{0}^{M-B,B} \\ \mathbf{0}^{B,M-B} & (\hat{\mathcal{R}}_x^B[\kappa B])^{-1} \end{pmatrix}. \tag{6.35}$$

According to equation (6.34) frequency domain algorithms achieve the same decorrelation as BOP when we use $(\breve{\mathcal{R}}_x^M[\kappa B])^{-1}\tilde{\mathcal{I}}^M[\kappa B]$ instead of $(\tilde{\mathcal{R}}_x^M[\kappa B])^{-1}$. This means that by using $(\breve{\mathcal{R}}_x^M[\kappa B])^{-1}$, a correlation within the block remains described by the matrix $\tilde{\mathcal{I}}^M[\kappa B]$, implying a deviation in convergence behaviour defined by the eigenvalue distribution of $\tilde{\mathcal{I}}^M[\kappa B]$.

### 6.3.3 BRLS and DBOP as Basis

#### Circulant Matrices

When $B > N$, the BOP algorithm can be performed efficiently in frequency domain with FFTs. If $N > B$ (in non-partitioned structures), the BRLS algorithm will be a better basis for transformation.[6] The BRLS update equation is given by

$$\underline{w}^N[(\kappa+1)B] = \underline{w}^N[\kappa B] + 2\alpha(\hat{\mathcal{R}}_x^N[\kappa B])^{-1}\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B]. \tag{6.37}$$

To perform the above BRLS update equation efficiently in frequency domain, the inverse autocorrelation matrix $(\hat{\mathcal{R}}_x^N[\kappa B])^{-1}$ must be approximated by a circulant matrix $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ and the input signal matrix $\mathcal{X}^{N,B}[\kappa B]$ must be transformed to a circulant matrix $(\check{\mathcal{X}}^M[\kappa B])^t$, like in previous section where BOP was used. This results in a (time domain) update equation with circulant matrices

$$\begin{aligned}\underline{w}^N[(\kappa+1)B] &= \underline{w}^N[\kappa B] + 2\alpha \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} \\ &\quad \cdot(\check{\mathcal{R}}_x^M[\kappa B])^{-1}(\check{\mathcal{X}}^M[\kappa B])^t \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}. \end{aligned} \tag{6.38}$$

Algorithms (6.37) and (6.38) will behave identically when there is a matrix $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ such that

$$\begin{aligned}&(\hat{\mathcal{R}}_x^N[\kappa B])^{-1}\mathcal{X}^{N,B}[\kappa B]\underline{r}^B[\kappa B] \\ &= \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix}(\check{\mathcal{R}}_x^M[\kappa B])^{-1}(\check{\mathcal{X}}^M[\kappa B])^t \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}. \end{aligned} \tag{6.39}$$

As the above condition (6.39) must be be valid for all $\underline{r}^B[kB]$, we can eleminate the residual signal on both sides. By replacing the input signal

[6]For partitioned structures, BOP is used as basis when $A > Z$. When this is not the case, the partitioned DBOP is chosen, with its update equation for $0 \leq i < g_U$ given by

$$\underline{w}_i^Z[(l+1)A] = \underline{w}_i^Z[lA] + 2\alpha(\hat{\mathcal{R}}_x^Z[lA])^{-1}\mathcal{X}^{Z,A}[lA - iZ]\underline{r}^A[lA]. \tag{6.36}$$

The only differences compared to the update equation of BRLS are the change in parameters and that we have $g_U$ equations instead of one, meaning that this section also holds for partitioned structures, by replacing the parameters.

matrix by its circulant version, and multiplying by $\mathbf{J}^N$, we get[7]

$$(\hat{\mathcal{R}}_x^N[\kappa B])^{-1} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}^M[\kappa B])^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} \tag{6.40}$$
$$= \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{R}}_x^M[\kappa B])^{-1} (\check{\mathcal{X}}^M[\kappa B])^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix}.$$

Knowing that condition (6.40) must be fulfilled for all circulant matrices $(\check{\mathcal{X}}^M[\kappa B])^t$ the input signal matrix $(\check{\mathcal{X}}^M[\kappa B])^t$ can be discarded, as will be shown hereafter, leading to

$$(\hat{\mathcal{R}}_x^N[\kappa B])^{-1} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} = \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{R}}_x^M[\kappa B])^{-1}. \tag{6.41}$$

### Elemination of the Input Signal

It is clear that (6.41) is a sufficient condition, we will show that it is also necessary. Suppose that there is a matrix $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ that fulfills condition (6.40) for all circulant matrices $\check{\mathcal{X}}^M[\kappa B]$, thus also for the circulant matrices $\check{\mathcal{X}}_i^M$, with

$$\check{\mathcal{X}}_i^M = \begin{pmatrix} \mathbf{0}^{M-i,i} & \mathbf{I}^{M-i} \\ \mathbf{I}^i & \mathbf{0}^{i,M-i} \end{pmatrix}. \tag{6.42}$$

By substituting these matrices in condition (6.40) we get for all $0 \leq i < M$

$$(\hat{\mathcal{R}}_x^N[\kappa B])^{-1} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{X}}_i^M)^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} \tag{6.43}$$
$$= \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\check{\mathcal{R}}_x^M[\kappa B])^{-1} (\check{\mathcal{X}}_i^M)^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix}.$$

To continue, we first define a matrix $\mathbf{U}_i^{B,M}$, with here

$$\mathbf{U}_i^{B,M} = \cdot \begin{pmatrix} (\underline{0}^{M-B})^t & 1 & (\underline{0}^{B-1})^t \\ \mathbf{0}^{B-1,M-B} & \underline{0}^{B-1} & \mathbf{0}^{B-1} \end{pmatrix} \check{\mathcal{X}}_i^M. \tag{6.44}$$

[7]Because $\hat{\mathcal{R}}_x^N[\kappa B]$ is symmetric and toeplitz, $\hat{\mathcal{R}}_x^N[\kappa B] = \mathbf{J}^N \hat{\mathcal{R}}_x^N[\kappa B]\mathbf{J}^N$. This means that also $(\hat{\mathcal{R}}_x^N[\kappa B])^{-1} = \mathbf{J}^N(\hat{\mathcal{R}}_x^N[\kappa B])^{-1}\mathbf{J}^N$. It is not necessary for the sequel of this subsection to eliminate $\mathbf{J}^N$ in condition (6.39), but it keeps the resemblance with the previous section.

The following property holds for this matrix

$$
\begin{aligned}
(\breve{\mathcal{X}}_i^M)^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} \mathbf{U}_i^{B,M} &= (\breve{\mathcal{X}}_i^M)^t \begin{pmatrix} 1 & (\underline{0}^{M-1})^t \\ \underline{0}^{M-1} & \mathbf{0}^{M-1} \end{pmatrix} \breve{\mathcal{X}}_i^M \\
&= \mathrm{diag}\{ \begin{pmatrix} \underline{0}^i \\ 1 \\ \underline{0}^{M-1-i} \end{pmatrix} \} \qquad (6.45)
\end{aligned}
$$

which implies that

$$
\begin{aligned}
\sum_{i=0}^{M-1} (\breve{\mathcal{X}}_i^M)^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} \mathbf{U}_i^{B,M} &= \sum_{i=0}^{M-1} \mathrm{diag}\{ \begin{pmatrix} \underline{0}^i \\ 1 \\ \underline{0}^{M-1-i} \end{pmatrix} \} \\
&= \mathbf{I}^M. \qquad (6.46)
\end{aligned}
$$

By a right-hand side multiplication of condition (6.43) by the matrix $\mathbf{U}_i^{B,M}$ and an accumulation, we get

$$
\begin{aligned}
&(\hat{\mathcal{R}}_x^N[\kappa B])^{-1} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} \left( \sum_{i=0}^{M-1} (\breve{\mathcal{X}}_i^M)^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} \mathbf{U}_i^{B,M} \right) \qquad (6.47) \\
&= \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\breve{\mathcal{R}}_x^M[\kappa B])^{-1} \left( \sum_{i=0}^{M-1} (\breve{\mathcal{X}}_i^M)^t \begin{pmatrix} \mathbf{0}^{M-B,B} \\ \mathbf{I}^B \end{pmatrix} \mathbf{U}_i^{B,M} \right).
\end{aligned}
$$

With help of property (6.46), we obtain condition (6.41), implying that it is indeed necessary.

### Approximation

By multiplying both sides of condition (6.41) by the autocorrelation matrices, we get

$$
\begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} \breve{\mathcal{R}}_x^M[\kappa B] = \hat{\mathcal{R}}_x^N[\kappa B] \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix}. \qquad (6.48)
$$

The matrix on the right-hand side of condition (6.48) is part of a circulant matrix $\breve{\mathcal{R}}_x^M[\kappa B]$. If there were such a circulant matrix, then the rows of the matrix $\hat{\mathcal{R}}_x^N[\kappa B] \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix}$ on the right-hand side have to be shifted versions of one another, which is clearly not the case. We therefore have to make an approximation in order to obtain a circulant matrix. We will do this by averaging all elements in $\hat{\mathcal{R}}_x^N[\kappa B] \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix}$ that should

have been equal for the existance of a circulant matrix $\check{\mathcal{R}}_x^M[\kappa B]$ to fulfill condition (6.48).

As $\hat{\mathcal{R}}_x^N[\kappa B]$ is symmetric, also $\check{\mathcal{R}}_x^M[\kappa B]$ is supposed to be symmetric. For a symmetric circulant matrix $\check{\mathcal{R}}_x^M[\kappa B]$ we know that for all $0 \leq a < M$ and $0 \leq b < M$ there are $\check{\rho}_{|a-b|}[\kappa B]$, such that

$$(\check{\mathcal{R}}_x^M[\kappa B])_{a,b} = \check{\rho}_{|a-b|}[\kappa B] \tag{6.49}$$

where for all $0 < a < M$, $\check{\rho}_a[\kappa B] = \check{\rho}_{M-a}[\kappa B]$, so (by omitting the time indices on the right-hand side)

$$\check{\mathcal{R}}_x^M[\kappa B] = \begin{pmatrix} \check{\rho}_0 & \check{\rho}_1 & \cdots & \check{\rho}_{\frac{M}{2}} & \cdots & \check{\rho}_1 \\ \check{\rho}_1 & \ddots & \ddots & & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \check{\rho}_{\frac{M}{2}} \\ \check{\rho}_{\frac{M}{2}} & & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & & \ddots & \ddots & \check{\rho}_1 \\ \check{\rho}_1 & \cdots & \check{\rho}_{\frac{M}{2}} & \cdots & \check{\rho}_1 & \check{\rho}_0 \end{pmatrix}. \tag{6.50}$$

The $N$ upper-most rows of this matrix $\check{\mathcal{R}}_x^M[\kappa B]$ have to approximate the matrix $\hat{\mathcal{R}}_x^N[\kappa B]\begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix}$, so we suggest to average all elements in this matrix $\hat{\mathcal{R}}_x^N[\kappa B]\begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix}$ that are supposed to be equal. With $(\hat{\mathcal{R}}_x^N[\kappa B])_{a,b} = \hat{\rho}_{|a-b|}[\kappa B]$, this implies for $N \geq \frac{M}{2}$ that[8]

$$\check{\rho}_i[\kappa B] = \begin{cases} \frac{N-i}{N}\hat{\rho}_i[\kappa B] & \text{for } 0 \leq i < M-N \\ \frac{N-i}{N}\hat{\rho}_i[\kappa B] + \frac{i-M+N}{N}\hat{\rho}_{M-i}[\kappa B] & \text{for } M-N \leq i < \frac{M}{2} \\ \frac{2N-M}{N}\hat{\rho}_{\frac{M}{2}} & \text{for } i = \frac{M}{2} \end{cases} \tag{6.51}$$

Like in the BOP case (with $B$ and $N$ interchanged), if $N \approx M$, which is the case when $N \gg B$, then $(N-i)/N \approx (M-i)/M$ (for $i < M/2$), and equation (6.27) is approximated by

$$\check{\rho}_i[\kappa B] \approx \begin{cases} \frac{M-i}{M}\hat{\rho}_i[\kappa B] & \text{for } 0 \leq i < M-N \\ \frac{M-i}{M}\hat{\rho}_i[\kappa B] + \frac{i}{M}\hat{\rho}_{M-i}[\kappa B] & \text{for } M-N \leq i < \frac{M}{2} \\ \hat{\rho}_{\frac{M}{2}}[\kappa B] & \text{for } i = \frac{M}{2} \end{cases} \tag{6.52}$$

[8] For a reasonable approximation, the matrix $\mathbf{0}^{N,M-N}$ containing zeroes must be small compared to the autocorrelation matrix $\hat{\mathcal{R}}_x^N[\kappa B]$ (or the influence of $\hat{\mathcal{R}}_x^N[\kappa B]$ must be neglectible, but then we do not have any decorrelation at all). We therefore assume that $N \geq \frac{M}{2}$.

When $\hat{\rho}_i \approx 0$ for $i > N$, then equation (6.52) and (6.30) are equal, which means that we may use $\check{\mathcal{R}}_x^M[\kappa B]$ also here, with (see appendix C, equation (C.11))

$$\check{\mathcal{R}}_x^M[\kappa B] = \frac{1}{M}\mathcal{E}\{\check{\mathcal{X}}^M[\kappa B] \cdot (\check{\mathcal{X}}^M[\kappa B])^t\}. \tag{6.53}$$

### 6.3.4 Approximation Error

The quality of our approximation can be given as a function of the input signal correlation matrices. Suppose there is a not-necessarily circulant matrix $\tilde{\mathcal{R}}_x^M[\kappa B]$, fulfilling condition (6.41), so

$$(\hat{\mathcal{R}}_x^N[\kappa B])^{-1} \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} = \begin{pmatrix} \mathbf{I}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\tilde{\mathcal{R}}_x^M[\kappa B])^{-1}. \tag{6.54}$$

Using $(\tilde{\mathcal{R}}_x^M[\kappa B])^{-1}$ instead of $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ would make the frequency domain algorithm behave exactly as BRLS. We can construct a deviation matrix $\tilde{\mathcal{I}}^M[\kappa B]$, with

$$(\tilde{\mathcal{R}}_x^M[\kappa B])^{-1} = \tilde{\mathcal{I}}^M[\kappa B](\check{\mathcal{R}}_x^M[\kappa B])^{-1} \tag{6.55}$$

by choosing

$$\tilde{\mathcal{I}}^M[\kappa B] = \begin{pmatrix} (\hat{\mathcal{R}}_x^N[\kappa B])^{-1} & \mathbf{0}^{N,M-N} \\ \mathbf{0}^{M-N,N} & \mathbf{I}^{M-N} \end{pmatrix} \check{\mathcal{R}}_x^M[\kappa B] \tag{6.56}$$

According to equation (6.55) frequency domain algorithms achieve the same decorrelation as BRLS when we use $\tilde{\mathcal{I}}^M[\kappa B](\check{\mathcal{R}}_x^M[\kappa B])^{-1}$ instead of $(\tilde{\mathcal{R}}_x^M[\kappa B])^{-1}$. This means that by using $(\check{\mathcal{R}}_x^M[\kappa B])^{-1}$, a correlation within the block remains described by the matrix $\tilde{\mathcal{I}}^M[\kappa B]$, implying a deviation in convergence behaviour defined by the eigenvalue distribution of $\tilde{\mathcal{I}}^M[\kappa B]$.

### 6.3.5 Transformation to Frequency Domain

The circulant matrix $\check{\mathcal{R}}_x^M[\kappa B]$ can be diagonalized to the input signal power (see appendix C)

$$\mathrm{diag}\{\underline{P}_x^M[\kappa B]\} = \mathcal{F}^M \check{\mathcal{R}}_x^M[\kappa B](\mathcal{F}^M)^{-1} \tag{6.57}$$

with

$$\underline{P}_x^M[\kappa B] = \frac{1}{M}\mathcal{E}\{\underline{X}^M[\kappa B] \otimes (\underline{X}^M[\kappa B])^*\}. \tag{6.58}$$

Using an estimate $\hat{\underline{P}}_x^M[\kappa B]$ of $\underline{P}_x^M[\kappa B]$, equation (6.14) (and equation (6.38)) can be computed in frequency domain as follows

$$\begin{aligned}\underline{w}^N[(\kappa+1)B] &= \underline{w}^N[\kappa B] + 2\alpha \begin{pmatrix} \mathbf{J}^N & \mathbf{0}^{N,M-N} \end{pmatrix} (\mathcal{F}^M)^{-1} \\ &\quad \cdot((\underline{X}^M[\kappa B])^* \otimes (\hat{\underline{P}}_x^M[\kappa B])^{-1} \otimes \underline{R}^M[\kappa B]) \end{aligned} \quad (6.59)$$

with

$$\underline{R}^M[\kappa B] = \begin{pmatrix} \underline{0}^{M-B} \\ \underline{r}^B[\kappa B] \end{pmatrix}. \quad (6.60)$$

The inverse power vector $(\hat{\underline{P}}^M[\kappa B])^{-1}$ is the elementwise inverse of the estimate $\hat{\underline{P}}^M[\kappa B]$ of the power vector.

### 6.3.6 Power Spectrum Estimation

The input power vector as defined in equation (6.58) can be estimated elementwise (like the variance in the NLMS and BNLMS case) for $0 < M\beta < 1$ by

$$(\hat{\underline{P}}^M[\kappa B])_i = (1 - M\beta)(\hat{\underline{P}}^M[(\kappa-1)B])_i + \beta|(\underline{X}^M[\kappa B])_i|^2. \quad (6.61)$$

The main disadvantage of this equation is that the inverse of it is needed in the update equation, which implies the use of divisions. This can be avoided by a direct estimation of the inverse power vector, like in the NLMS and BNLMS case, using a certain (small) positive threshold $\epsilon$ for $0 < M \cdot \beta \ll 1$ by

$$\begin{aligned}\mathcal{P}_i^{-1}[\kappa B] &= (1 + M\beta)(\hat{\underline{P}}_x^M[(\kappa-1)B])_i^{-1} \\ &\quad -\beta(\hat{\underline{P}}_x^M[(\kappa-1)B])_i^{-2}|(\underline{X}^M[\kappa B])_i|^2 \end{aligned} \quad (6.62)$$

$$(\hat{\underline{P}}_x^M[\kappa B])_i^{-1} = \begin{cases} \epsilon & \text{for } \mathcal{P}_i^{-1}[\kappa B] < \epsilon \\ \mathcal{P}_i^{-1}[\kappa B] & \text{otherwise} \end{cases} \quad (6.63)$$

In appendix A the impact on computational complexity and memory occupation of the above estimation method is calculated.

## 6.4 Partitioning in the Update Part

### 6.4.1 Place of Normalization

The power vector estimate does not always match the actual input signal power. For example, if we look at figure 4.4, we see that delayed versions

of the input signal vector are used. The normalization however does not take this into account. It was assumed that the correlation and variance are equal for these delayed input signals. When we have speech-like input signals, this is certainly not a valid assumption.

A solution can be to normalize the input signal instead of the residual signal. The normalization of the older input signal vectors is then carried out by an older estimate of the input signal power. In figure 6.1 the place to normalize the residual signal is labelled $\aleph_1$. The normalization of the input signal takes place at $\aleph_2$.

The residual signal however, does contain elements from the delayline, so a better solution for the normalization problem is to normalize partly at place $\aleph_2$ (for the residual signal) and partly at $\aleph_1$ (for the input signal). This implies a normalization by the square root of the power vector, like

$$\begin{aligned} \underline{Y}^L[lA] &= 2\alpha(\hat{\underline{P}}^L_{x,1}[lA])^{-\frac{1}{2}} \otimes \underline{R}^L[lA] && (6.64) \\ \underline{w}^Z_i[(l+1)A] &= \underline{w}^Z_i[lA] + \begin{pmatrix} \mathbf{J}^Z & \mathbf{0}^{Z,L-Z} \end{pmatrix} \cdot (\mathcal{F}^M)^{-1} \\ &\quad \cdot((\underline{X}^L_i[lA])^* \otimes (\hat{\underline{P}}^L_{x,2}[(l-i)A])^{-\frac{1}{2}} \otimes \underline{Y}^L[lA]). && (6.65) \end{aligned}$$

Also here a direct estimate can be used to avoid the use of square roots and divisions. With a certain (small) positive threshold $\epsilon$, the inverse square root is estimated for $0 \leq i < L$ and $0 < L \cdot \beta \ll 1$ by

$$\begin{aligned} \mathcal{P}_i^{-\frac{1}{2}}[lA] &= (1 + L\beta)(\hat{\underline{P}}^L_x[(l-1)A])_i^{-\frac{1}{2}} \\ &\quad -\beta((\hat{\underline{P}}^L_x[(l-1)A])_i^{-\frac{1}{2}})^3 |(\underline{X}^L[lA])_i|^2 && (6.66) \\ (\hat{\underline{P}}^L_x[lA])_i^{-\frac{1}{2}} &= \begin{cases} \epsilon & \text{for } \mathcal{P}_i^{-\frac{1}{2}}[lA] < \epsilon \\ \mathcal{P}_i^{-\frac{1}{2}}[lA] & \text{otherwise} \end{cases} && (6.67) \end{aligned}$$

### 6.4.2 Reduced Dimension Normalization

#### Large Dimensions

In the previous subsection it was already concluded that in applications where speech-like input signals are used, normalization can be a problem. In chapter 8 the acoustic echo canceller implementation shows that a large update part block length, partition length and Fourier transform can have a negative influence on the decorrelation properties. This is caused by the fact that on one hand the sample frequency in the update part is too low compared to the non-stationary aspects of the input signal (speech). On the other hand also the large Fourier transforms involve a high resolution

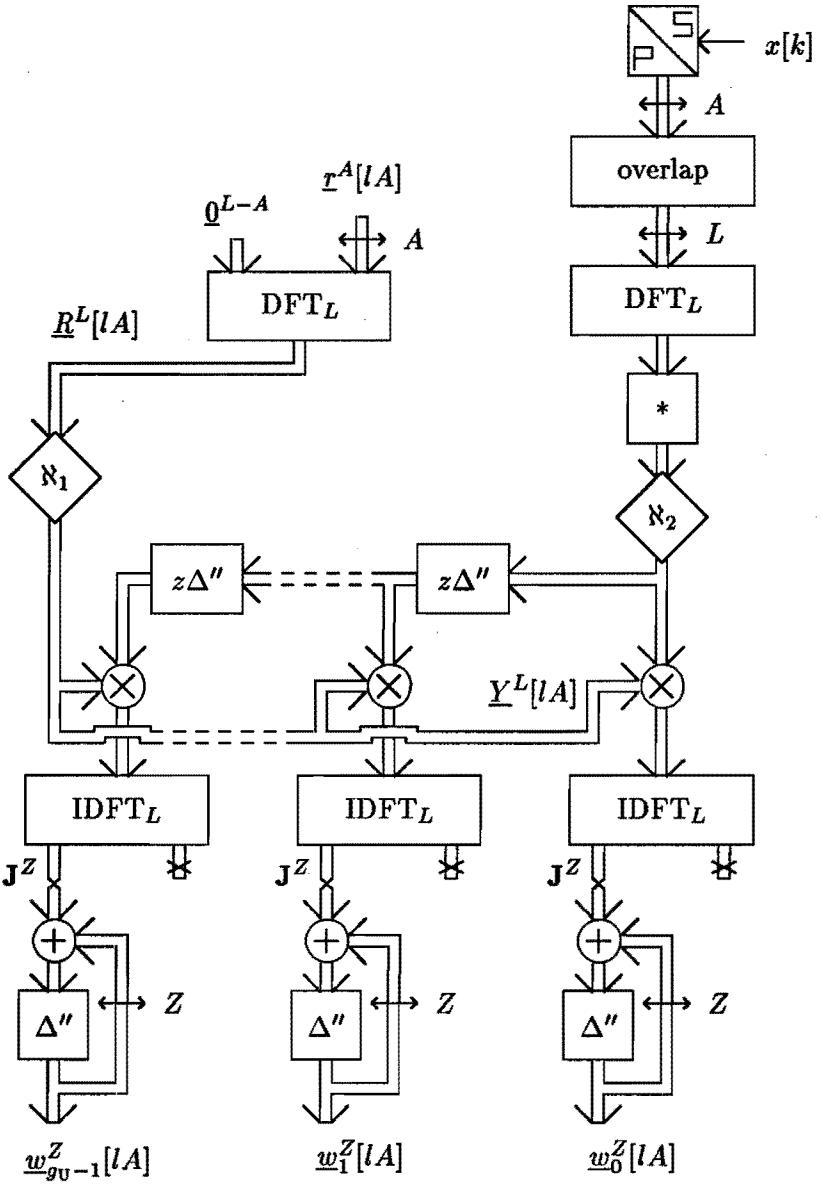



Figure 6.1: Places to normalize.

in frequency domain, which implies that the variance per frequency domain bin (vector element) fluctuates enormously. Both problems can be solved by taking smaller parameters in the update part, but this would increase the computational complexity to an often unacceptable level. We can improve the power vector estimation by increasing the rate of the power vector update, which would need some extra (large) FFTs of the input signal. Another solution would be to decouple update part parameters from the length and rate of the power spectrum estimation part.

### Parameter Decoupling

We start with a BOP update equation, with block length $A_p$ (and block index $\lambda$), given by

$$\underline{w}^N[(\lambda+1)A_p] = \underline{w}^N[\lambda A_p] + 2\alpha \mathcal{X}^{N,A_p}[\lambda A_p](\hat{\mathcal{R}}_x^{A_p}[\lambda A_p])^{-1}\underline{r}^{A_p}[\lambda A_p], \quad (6.68)$$

and partition it into $N$ parts, so for $0 \leq j < N$

$$w_j[(\lambda+1)A_p] = w_j[\lambda A_p] + 2\alpha(\underline{x}^{A_p}[\lambda A_p - j])^t(\hat{\mathcal{R}}_x^{A_p}[\lambda A_p])^{-1}\underline{r}^{A_p}[\lambda A_p] \quad (6.69)$$

By choosing a Fourier transform length $L_p$, this can be written as

$$w_j[(\lambda+1)A_p] = w_j[\lambda A_p] + (\underline{x}^{A_p}[\lambda A_p - j])^t\underline{y}^{A_p}[\lambda A_p] \quad (6.70)$$

with

$$\underline{y}^{A_p}[\lambda A_p] = \begin{pmatrix} \mathbf{0}^{A_p,L_p-A_p} & \mathbf{I}^{A_p} \end{pmatrix} (\mathcal{F}^{L_p})^{-1}\underline{Y}^{L_p}[\lambda A_p] \quad (6.71)$$

$$\underline{Y}^{L_p}[\lambda A_p] = 2\alpha\mathcal{F}^{L_p}\begin{pmatrix} \underline{0}^{L_p-A_p,A_p} \\ (\hat{\mathcal{R}}_x^{A_p}[\lambda A_p])^{-1}\underline{r}^{A_p}[\lambda A_p] \end{pmatrix}. \quad (6.72)$$

Like in the previous chapters, $\underline{Y}^{L_p}[\lambda A_p]$ can be approximated by

$$\underline{Y}^{L_p}[\lambda A_p] = 2\alpha(\hat{\underline{P}}_x^{L_p}[\lambda A_p])^{-1} \otimes \underline{R}^{L_p}[\lambda A_p] \quad (6.73)$$

where

$$\underline{R}^{L_p}[\lambda A_p] = \mathcal{F}^{L_p}\begin{pmatrix} \mathbf{0}^{L_p-A_p,A_p} \\ \underline{r}^{A_p}[\lambda A_p] \end{pmatrix}. \quad (6.74)$$

By assembling groups of $Z$ coefficients out of the update equations (6.70), with, like in the (D)PBFDAF case, $g_U = \lceil \frac{N}{Z} \rceil$, we obtain for $0 \leq i < g_U$

$$\underline{w}_i^Z[(\lambda+1)A_p] = \underline{w}_i^Z[\lambda A_p] + \mathcal{X}^{Z,A_p}[\lambda A_p - iZ]\underline{y}^{A_p}[\lambda A_p] \quad (6.75)$$

with

$$\underline{w}_i^Z[\lambda A_p] = \left( \begin{array}{ccc} w_{iZ+Z-1}[\lambda A_p] & \cdots & w_{iZ}[\lambda A_p] \end{array} \right)^t. \tag{6.76}$$

Choosing a block length $A$, such that $\frac{A}{A_p}$ is integer, and block index $l$, we can accumulate $A/A_p$ update equations (6.75), as follows for $0 \leq i < g_U$

$$\begin{aligned} \underline{w}_i^Z[(l+1)A] &= \underline{w}_i^Z[lA] + \sum_{j=0}^{A/A_p-1} (\mathcal{X}^{Z,A_p}[(l\frac{A}{A_p} - j)A_p - iZ] \\ &\quad \cdot \underline{y}^{A_p}[(l\frac{A}{A_p} - j)A_p]) \qquad (6.77) \\ &= \underline{w}_i^Z[lA] + \mathcal{X}^{Z,A}[lA - iZ]\underline{y}^A[lA] \qquad (6.78) \end{aligned}$$

with

$$\underline{y}^A[lA] = \left( \begin{array}{c} \underline{y}^{A_p}[lA - A + A_p] \\ \vdots \\ \underline{y}^{A_p}[lA - A_p] \\ \underline{y}^{A_p}[lA] \end{array} \right). \tag{6.79}$$

Equation (6.78) resembles the update equation of chapter 4, and by chosing $L \geq A + Z - 1$, we obtain

$$\begin{aligned} \underline{w}_i^Z[(l+1)A] &= \underline{w}_i^Z[lA] + \left( \begin{array}{cc} \mathbf{J}^Z & \mathbf{0}^{Z,L-Z} \end{array} \right) \cdot (\mathcal{F}^L)^{-1} \\ &\quad \cdot ((\underline{X}_i^L[lA])^* \otimes \underline{Y}^L[lA]) \qquad (6.80) \\ \underline{Y}^L[lA] &= \mathcal{F}^L \left( \begin{array}{c} \underline{0}^{L-A} \\ \underline{y}^A[lA] \end{array} \right) \qquad (6.81) \end{aligned}$$

The part of the partitioned update part where the vector $\underline{Y}^L[lA]$ is calculated with reduced dimension and increased rate of the power vector is depicted in figure 6.2. The one sample delays $\Delta'''$ delay over $A_p \cdot \mathrm{T}$ seconds here. The rest of the update part is equivalent to figure 4.4.

The method for reducing power dimension requires three extra (I)FFTs. Two of them are depicted in figure 6.2, the third is needed for the estimation of the input signal power vector of length $L_p$ (we cannot use the already available vector of length $L$ here). Knowing that also the power estimation and elementwise multiplication take place on a different rate and with a different size we obtain for the extra computational complexity

$$\Psi_{\mathrm{EXTRA}} = \frac{3 \cdot \Psi_{\mathrm{FFT}}\{L_p\} + \Psi_P\{L_p\}}{A_p} - \frac{\Psi_P\{L\}}{A} \tag{6.82}$$

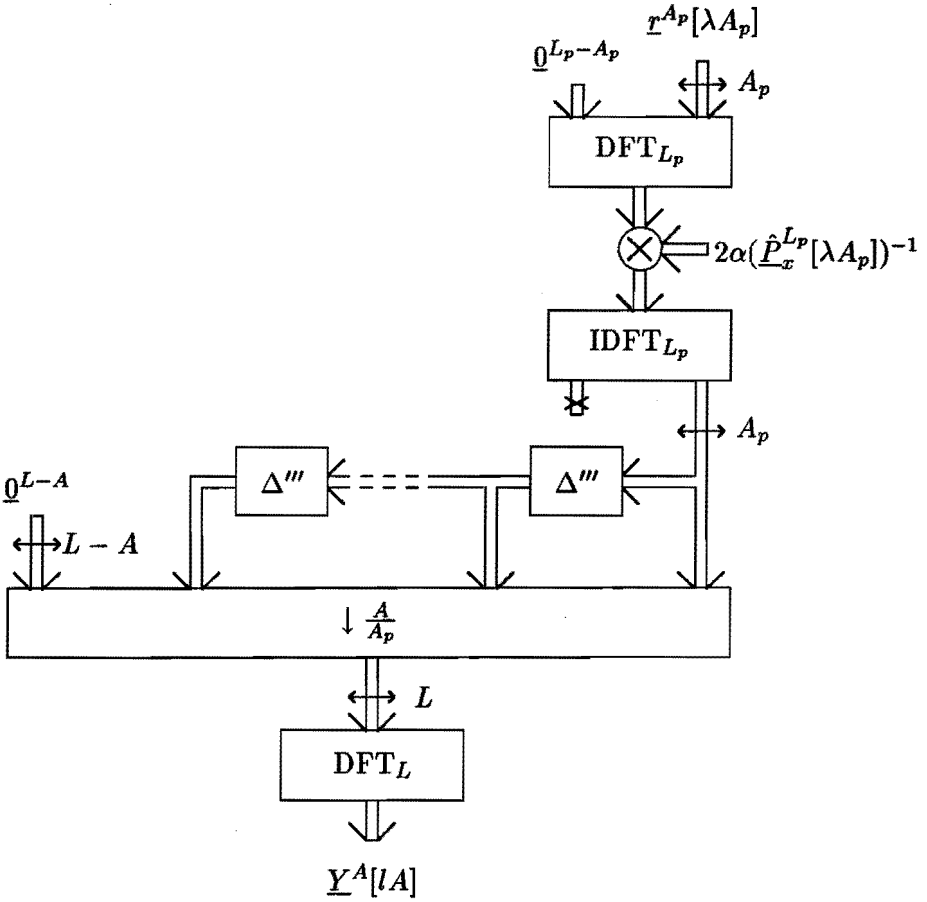



Figure 6.2: Decoupling power vector.

## 6.5 Conclusions

By making a direct estimation of the inverse power (vector), divisions can be avoided in adaptive filtering, which induces a large reduction of the computational complexity. Time domain decorrelation by an inverse autocorrelation matrix can be computed in frequency domain, making some approximations, by an elementwise vector multiplication. The quality of the approximations made in frequency domain, compared to time domain adaptive filtering, depend on the algorithm parameters. The approximation error in (D)PBFDAF can be reduced by increasing the partition length.

At the cost of a small extra computational complexity, the power vector dimension and rate can be chosen independent of the update part dimensions and rate. This yields a more flexible algorithm, which can be of use in cases where the non-stationarity of the input signal causes problems in the normalization (and thus in convergence behaviour).

# Chapter 7

# Comparison

*The algorithms introduced in the previous chapters all have different properties. First we will look at the relation between the computational complexity of the algorithms and the algorithm processing delay. After that the implementation dependent computation processing delay is taken into account.*

*Simulation results using an acoustic echo cancellation structure with a simulated room are used to compare the convergence and tracking properties of the diverse algorithms.*

## 7.1 Complexity, Delay and Memory

### 7.1.1 Fixed Filters

As the total processing delay depends on the specific hardware used, it is impossible to make an implementation independent comparison. Therefore we will first look at the relation between the algorithm processing delay $\Delta$ and the complexity $\Psi$. In figure 7.1 the dependency of this complexity on the algorithm delay is depicted for different algorithms to compute a convolution (fixed filter) of length 4000 (the number of coefficients). The "almost" logaritmic scale for $\Delta$ is chosen because for all block processing algorithms $\Delta = B - 1$, and the examples used to plot the figure use $B = 2^i$ for $0 \leq i \leq 9$. This implies that the vertical lines correspond to the examples used in chapters 2 till 5.

In order to take the computation processing delay into account, we choose the TMS320C30 processor as implementation hardware. With help of appendix A and assuming a sample frequency of 8 kHz, we can plot the number of real multiplications per sample for a TDC, BFDC, PBFDC and NUPBFDC as a function of the maximum allowable delay in figure 7.2. At 8

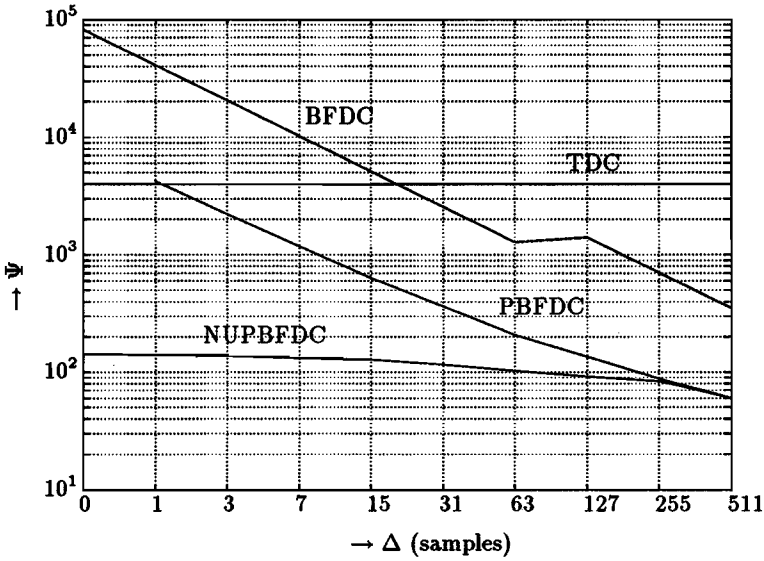



Figure 7.1: Complexity of TDC, BFDC, PBFDC and NUPBFDC as function of algorithm processing delay.

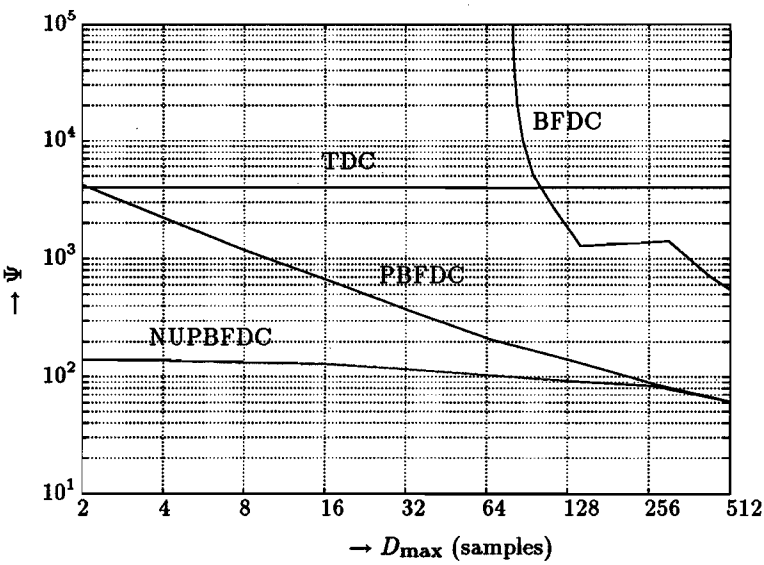



Figure 7.2: Complexity of TDC, BFDC, PBFDC and NUPBFDC as function of total processing delay.

kHz a single TMS320C30 processor executes 2083 instructions per sample. Besides $\Psi$ multiplications, the algorithms also need additions, load, etc., also taking in the order of $\Psi$ instructions. This means that we need a few times $\Psi$ instuctions to implement an algorithm with a computational complexity of $\Psi$. This means that a few signal processors are already needed to implement a TDC in real time. The figure also shows the huge computation processing delay of BFDC compared to PBFDC and certainly NUPBFDC for an equal complexity $\Psi$. The "inregularity" that can be observed in the BFDC curves is caused by the jump from FFT length 4096 (for $B \leq 64$) to 8192 (for $B \geq 128$), because $B + N$ must be smaller that the FFT length.

The NUPBFDC is in fact a generalization of PBFDC, which on its term is a generalization of BFDC. This implies that for increasing maximum allowable processing delay $D_{\max}$ all complexity curves of the frequency domain algorithms will converge to the same curve. The first signs of this proces occur on the right-hand side of the figures.

### 7.1.2 Adaptive Filters

Having compared the convolutions, we will now look at adaptive filters Also here the total processing delay depends on the specific hardware used, so first the relation between the algorithm processing delay $\Delta$ and the complexity $\Psi$ is studied. In figure 7.3 the dependency of this complexity on the algorithm delay is depicted for a filter length $N = 4000$. For DPBFDAF and NUPBFDAF we use an update part block length and partition length of 512 (implying a Fourier transformation length of 1024), like in the examples in chapters 4 and 5.

Taking the computation processing delay into account, as in the previous subsection, we can plot complexity for the adaptive filtering algorithms as a function of the maximum allowable delay in figure 7.4. The computational complexity of the adaptive filter algorithms is clearly proportional to the complexity of the convolutions used. For the "irregularities" in the BFDAF curves, the same explanation as in the fixed filter case for BFDC is valid. We see that for a maximum allowable processing delay of 512 samples, NUPBFDAF, DPBFDAF and PBFDAF have equal complexity. This is caused by the facted that the most efficient realization of both NUPBFDAF and DPBFDAF in this case is a PBFDAF. For smaller maximum allowable processing delays, we observe a huge decrease in complexity from BFDAF, RLS (here FTF) and BNLMS towards PBFDAF, DPBFDAF and eventually NUPBFDAF.

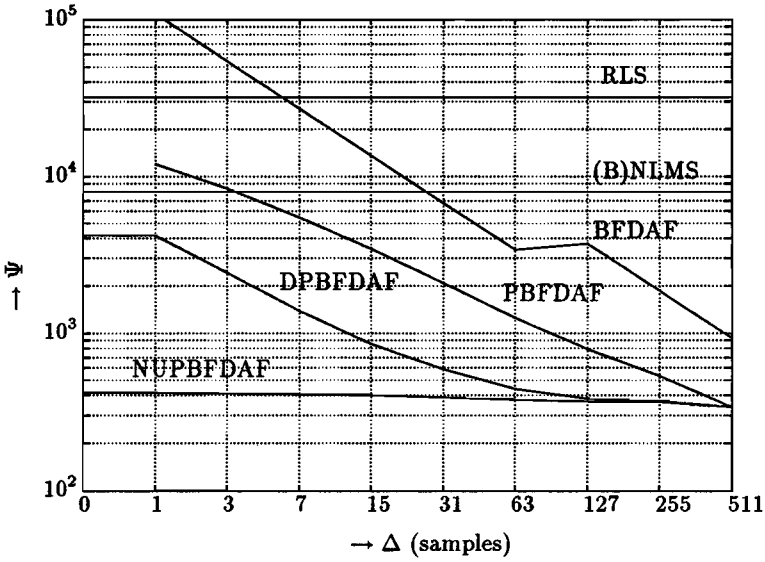



Figure 7.3: Complexity of AF's as function of algorithm processing delay.

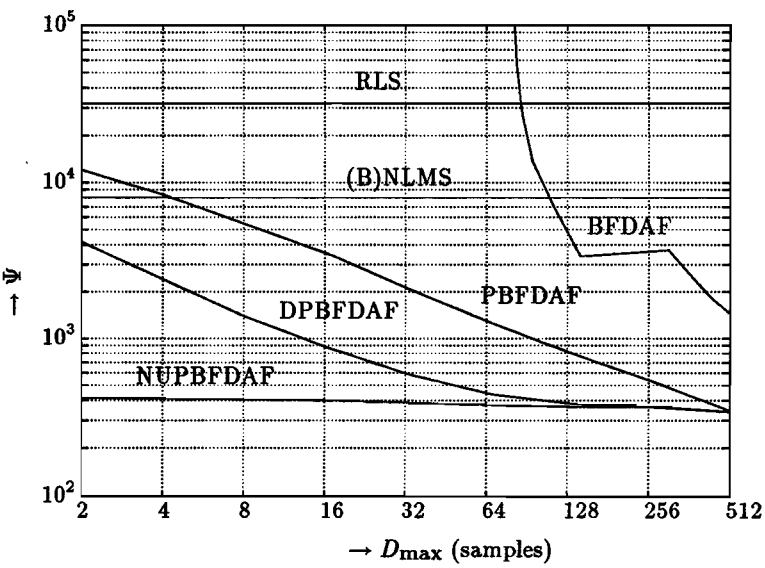



Figure 7.4: Complexity of AF's as function of the total processing delay.

## 7.2 Convergence and Tracking

In this section simulation results of convergence behaviour of the frequency domain algorithms are presented. The acoustic echo canceller is kept in mind as example, so all algorithms are tested using a simulated room impulse response as "echo path", where we assume a $5 \times 6 \times 3.5$ room whose impulse response is sampled at 8 kHz, and truncated at 992 samples.[1] In figure 7.5 this impulse response is depicted.[2]

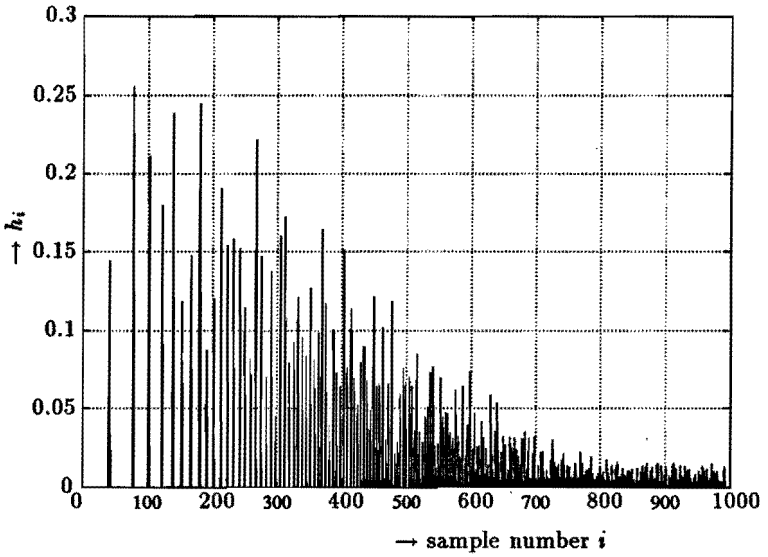



Figure 7.5: Truncated room impulse response $\underline{h}$.

We perform three groups of test, with three different input signals.

- Tests A: A white noise signal filtered by a Moving Average (MA) filter of order 1 is used as input signal, $x_1[k] = 0.711 \cdot (n[k] + 0.99 \cdot n[k-1])$, where $n[k]$ is white noise with $\mathcal{E}\{n[k]\} = 0$ and $\mathcal{E}\{(n[k])^2\} = 1$. The

[1] Choosing $N = 992$ makes it possible for a PBFDAF with $B = 4$, $Q = 124$ and $g_F = 4$ to have $g_F \cdot Q \geq N$.

[2] Note that all room impulse response components are positive. This is caused by using the following assumption for the construction of the simulated room impulse response. Reflections against walls are seen as a longitudinal waves reaching a "closed end", so a positive pulse reflects as a positive pulse [53, 4]. In practice, not all components will be positive, caused by non ideal reflections and non ideal responses of the loudspeaker and microphone.

Power Spectral Density (PSD) $S_{x_1,\omega}$ of $x_1[k]$ is depicted in figure 7.6 (see appendix C).

- Tests B: The PSD $S_{x_2,\omega}$ of the input signal $x_2[k]$ of the second group of experiments is depicted in figure 7.7. This signal is chosen to show that complicated signals (with a large relevant autocorrelation length) require a large decorrelation dimension.

- Tests C: A white noise signal filtered by a Moving Average (MA) filter of order 16 is used as input signal, $x_3[k] = 0.577(-n[k]+n[k-8]+n[k-16])$, where $n[k]$ is white noise with $\mathcal{E}\{n[k]\} = 0$ and $\mathcal{E}\{(n[k])^2\} = 1$. The PSD $S_{x_3,\omega}$ of $x_3[k]$ is depicted in figure 7.8. Choosing the signal this way makes decorrelation with a matrix of dimension 8 or smaller useless, as there is no correlation within a block of that length.

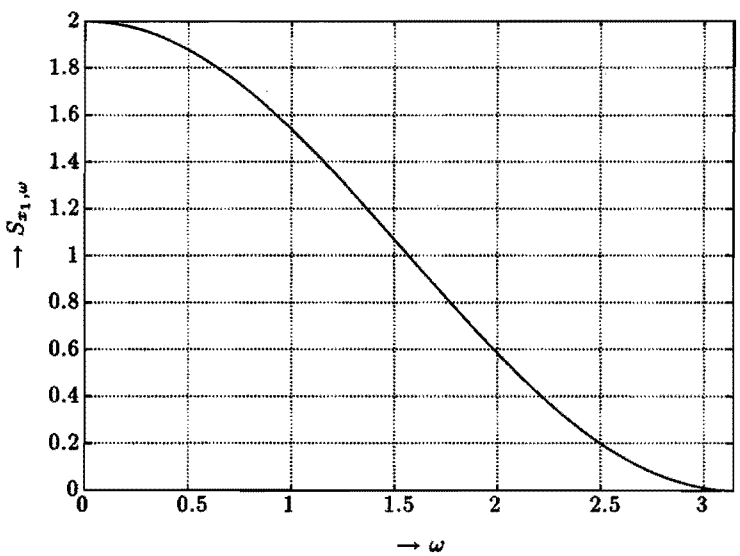



Figure 7.6: PSD $S_{x_1,\omega}$ of $x_1[k]$.

In tables 7.1, 7.2 and 7.3 the diverse testing parameters are given, as well as the complexity and the number of memory locations needed according to the formulas of the previous chapters. Note that increasing the quotient of $Q$ and $B$ also increases the amount of memory needed. This is caused by the input signal delay line (and could be reduced at cost of some extra FFTs). The results of the diverse tests are depicted in figures 7.9, 7.10 and

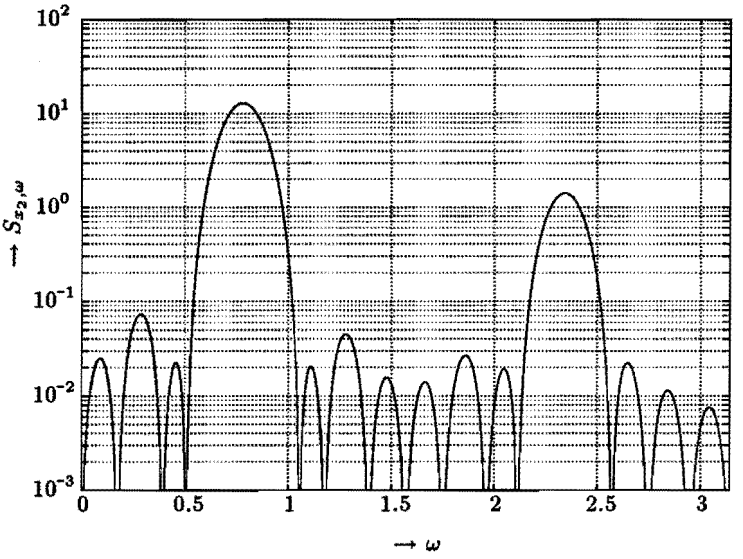



Figure 7.7: PSD $S_{x_2,\omega}$ of $x_2[k]$.

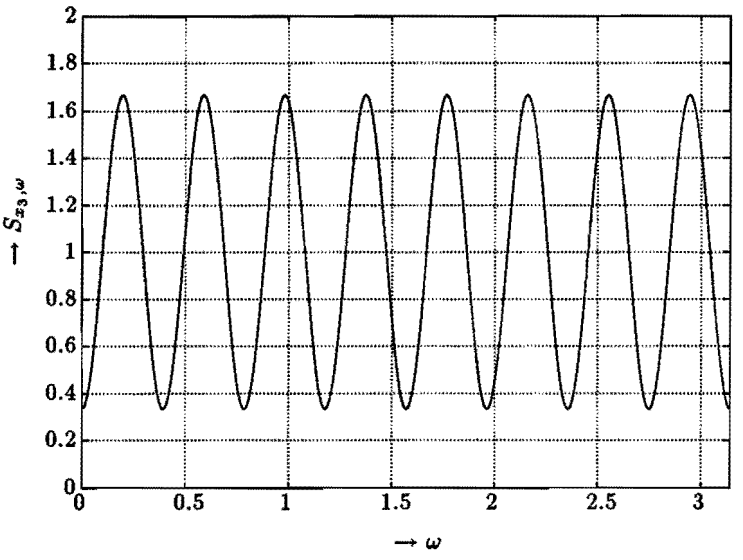



Figure 7.8: PSD $S_{x_3,\omega}$ of $x_3[k]$.

7.11. As we are using stationary input signals, and chose the input signal in such a way that $\mathcal{E}\{(e[k])^2\} = 1$, we can look at the squared residual signal $(r[k])^2$ to investigate convergence behaviour. In order to keep the convergence behaviour visible, some averaging of $(r[k])^2$ is done to get an estimate of $\mathcal{E}\{(r[k])^2\}$. In all tests, the adaptive weights are initially put to zero.

| Test | Algorithm | Parameters | | | | | | $\Psi$ | $\Theta$ |
|---|---|---|---|---|---|---|---|---|---|
| | | Filter | | | | Update | | | |
| | | $B$ | $Q$ | $M$ | $g_F$ | $A = Z$ | $L$ | | $\cdot 10^3$ |
| A_ F | FBNLMS | 4 | | 1024 | | | | 11013 | 5.12 |
| A_ B | BFDAF | 4 | | 1024 | | | | 9991 | 5.12 |
| A_ P1 | PBFDAF | 4 | 4 | 8 | 248 | | | 2244 | 3.99 |
| A_ P2 | PBFDAF | 4 | 28 | 32 | 36 | | | 2424 | 9.30 |
| A_ P3 | PBFDAF | 4 | 124 | 128 | 8 | | | 3596 | 33.0 |
| A_ D | DPBFDAF | 4 | 28 | 32 | 36 | 504 | 1024 | 670 | 13.9 |

Table 7.1: Test parameters of test 1, $N = 992$ and $\alpha = 2 \cdot 10^{-4}$

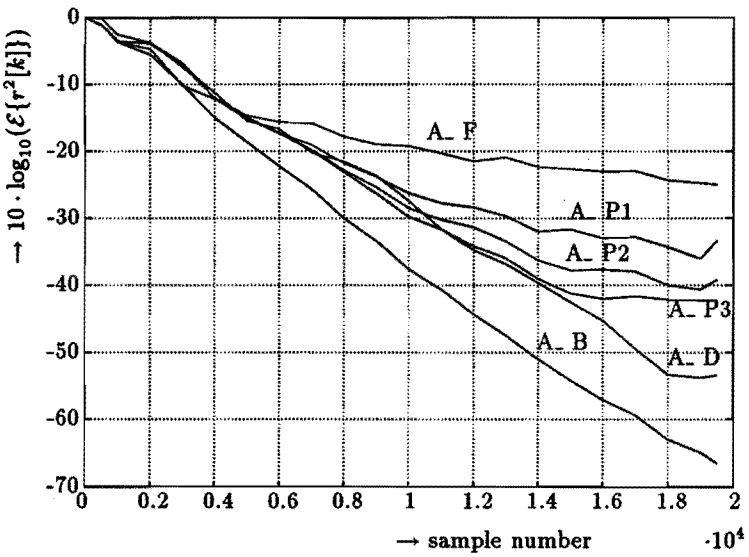



Figure 7.9: Convergence behaviour for test 1.

In the first group of tests, we see that the PBFDAF with a small partition length $Q$ already improves convergence behaviour significantly compared to FBNLMS. Increasing $Q$ will decrease the inter-block correlation

| Test | Algorithm | Parameters | | | | | | $\Psi$ | $\Theta$ |
|---|---|---|---|---|---|---|---|---|---|
| | | Filter | | | | Update | | | |
| | | $B$ | $Q$ | $M$ | $g_F$ | $A=Z$ | $L$ | | $\cdot 10^3$ |
| B_ F | FBNLMS | 4 | | 1024 | | | | 11013 | 5.12 |
| B_ B | BFDAF | 4 | | 1024 | | | | 9991 | 5.12 |
| B_ P2 | PBFDAF | 4 | 28 | 32 | 36 | | | 2424 | 9.30 |
| B_ P3 | PBFDAF | 4 | 124 | 128 | 8 | | | 3596 | 33.0 |
| B_ D | DPBFDAF | 4 | 28 | 32 | 36 | 504 | 1024 | 670 | 13.9 |

Table 7.2: Test parameters of test 2, $N = 992$ and $\alpha = 2 \cdot 10^{-4}$

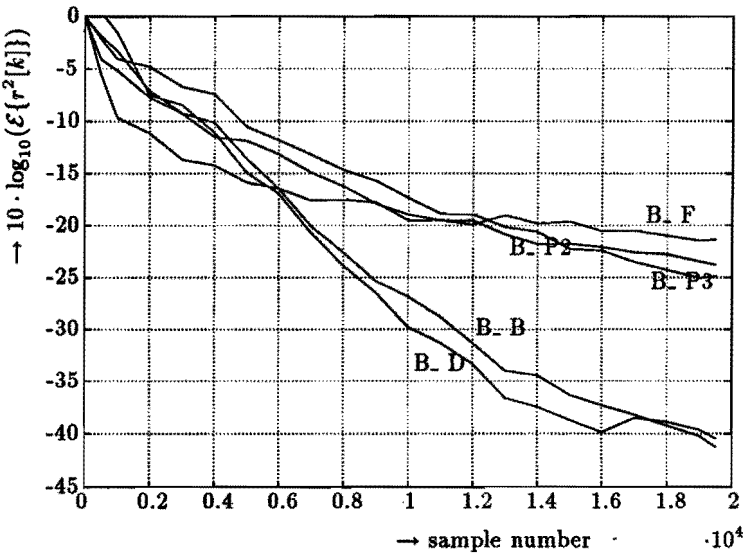



Figure 7.10: Convergence behaviour for test 2.

| Test | Algorithm | Parameters | | | | | | $\Psi$ | $\Theta$ |
|---|---|---|---|---|---|---|---|---|---|
| | | Filter | | | | Update | | | |
| | | $B$ | $Q$ | $M$ | $g_F$ | $A=Z$ | $L$ | | $\cdot 10^3$ |
| C_ F | FBNLMS | 4 | | 1024 | | | | 11013 | 5.12 |
| C_ B | BFDAF | 4 | | 1024 | | | | 9991 | 5.12 |
| C_ P1 | PBFDAF | 4 | 4 | 8 | 248 | | | 2244 | 3.99 |
| C_ P2 | PBFDAF | 4 | 28 | 32 | 36 | | | 2424 | 9.30 |
| C_ P3 | PBFDAF | 4 | 124 | 128 | 8 | | | 3596 | 33.0 |
| C_ D | DPBFDAF | 4 | 28 | 32 | 36 | 252 | 512 | 699 | 12.6 |

Table 7.3: Test parameters of test 3, $N = 992$ and $\alpha = 2 \cdot 10^{-4}$

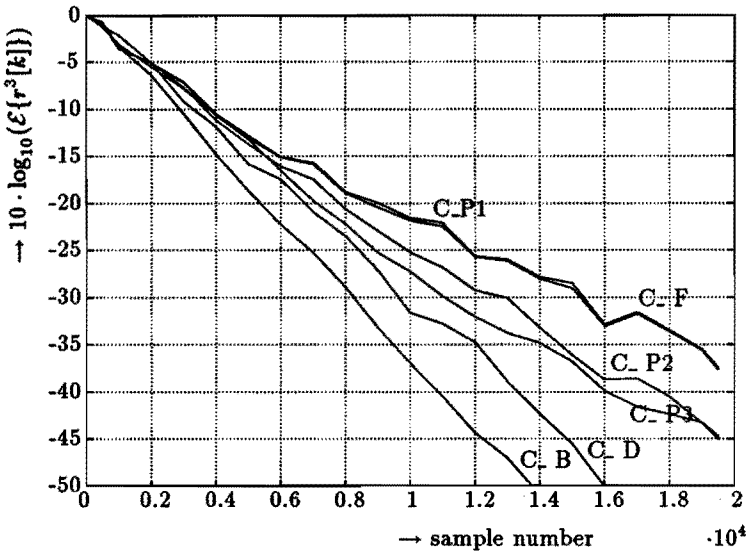



Figure 7.11: Convergence behaviour for test 3.

proportional to the number of blocks, which is clearly shown in the figure. Of course BFDAF performs the best, while also DPBFDAF shows good performance (under these stationairy input signal conditions). In both DPBFDAF and BFDAF the approximation error (from the "perfect" time domain algorithms to the frequency domain algorithms) seems neclectibly small. This is caused by the small length of the relevant part of the input signal autocorrelation function.

In the second group of tests a more "sophisticated" signal causes more influence by the approximation error, as can be seen in figure 7.10. We need a much larger partition length $Q$ (determining the decorrelation dimension in PBFDAF when $Q > B$) to obtain equal improvement in convergence behaviour as in the first group of tests. A larger relevant autocorrelation length for the input signal $x_2[k]$ compared to $x_1[k]$ keeps the inter block correlation (that is not removed) large compared to the correlation within the block (that is removed).

The third group of experiments shows that decorrelation (or its frequency domain approximation) by a matrix that is too small to contain any of the relevant input signal correlation does not improve convergence behaviour compared to FBNLMS (there is no correlation within the blocks to be removed). Larger partition lengths of course start to reduce the inter

block correlation (by moving it within the blocks, where it is removed) so convergence behaviour is improved by increasing the partition length.

## 7.3 Conclusions

The simulations in this section confirm the assumptions made about the convergence behaviour in the previous chapters. Comparing complexity of the diverse algorithms, we observe that the frequency domain algorithms DPBFDAF and NUPBFDAF can reduce computational complexity enormously compared to BFDAF and the time domain algorithms while maintaining convergence properties.

# Chapter 8

# Acoustic Echo Cancellation

*To investigate the real-time performance of the DPBFDAF algorithm, an acoustic echo canceller based on this algorithm is implemented on a single DSP, the TMS320C30. Real-time echo cancellation requires some extensions to the algorithm. Especially stepsize control in combination with normalization needs attention. Furthermore, when "double talk" appears the adaptation of the coefficients must be frozen, otherwise large misadjustments (or even divergence) of the weights yields a very bad performance.*

*A flexible setup of the DPBFDAF for acoustic echo cancellation makes it possible to choose the sample frequency, the number of coefficients and the processing delay independent of one another (only limited by the total complexity).*

## 8.1 Introduction to AEC

In [36] a one DSP implementation of an Acoustic Echo Canceller (AEC) based on an unconstrained PBFDAF approach with 2048 coefficients at a rate of 16kHz (thus cancelling an echo of at most 128 milli-seconds) with a processing delay of 32 milli-seconds is introduced. The use of PBFDAF in a one chip implementation, results in a large processing delay (32 milli-seconds) caused by complexity constraints. A second disadvantage of the above mentioned implementation is the use of the unconstrained version of PBFDAF to decrease computational complexity as only three (Inverse) Fast Fourier Transforms are needed then [36]. Using a constrained frequency domain algorithm in partitioned structures leads to much better convergence behaviour compared to unconstrained structures (see subsection 4.3.3)

The reduction in complexity of DPBFDAF compared to PBFDAF makes

it possible to implement a real-time AEC with a smaller processing delay and a better convergence behaviour [18, 4]. A further improvement would be obtained using NUPBFDAF, which means that the PBFDC used to implement the filter part of DPBFDAF has to be replaced by NUPBFDC, which has not been completed yet. Real-time echo cancellation according to the scheme of figure 8.1, requires some enhancements to the adaptive algorithms. A variable stepsize algorithm (in combination with the normalization part) is needed for performance optimization. When "double talk" appears, meaning that both the input signal $x[k]$ and the local speech signal $s[k]$ contain speech, the adaptation of the coefficients must be frozen, otherwise large misadjustments (or even divergence) of the weights leads to a bad performance [4, 28].

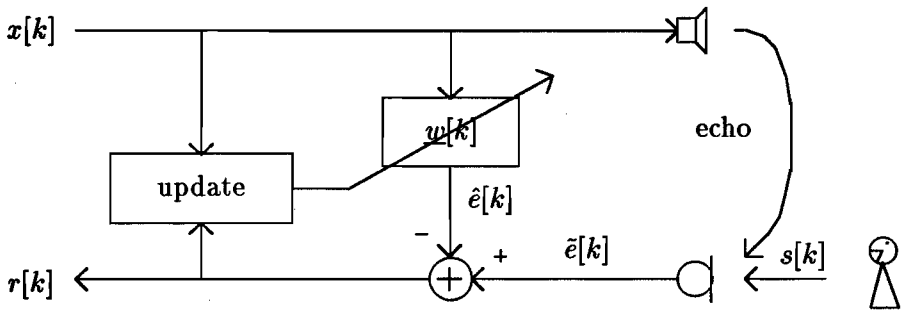



Figure 8.1: Acoustic Echo Canceller scheme.

## 8.2 Implementation

### 8.2.1 Hardware

The AEC implementation is realized on a single DSP, the Texas Imstruments TMS320C30. It has a capacity of 33.3 MFLOPS (Million FLoating point Operations Per Second) and is mounted on a Loughborough Sound Images (LSI) prototyping board, inserted in a PC-slot. In figure 8.2 the hardware configuration is depicted.

The input signal $x[k]$ is taken from PC memory, and fed to a speaker through a Digital to Analog (D/A) converter, Low-Pass Filter (LPF) and amplifier, controlled by the DSP. In the same room the microphone is situated, whose output is directed towards an Analog to Digital (A/D) converter via an amplifier and a low-pass filter, also controlled by the DSP. The residual signal is returned to the PC memory (from which it can eventually

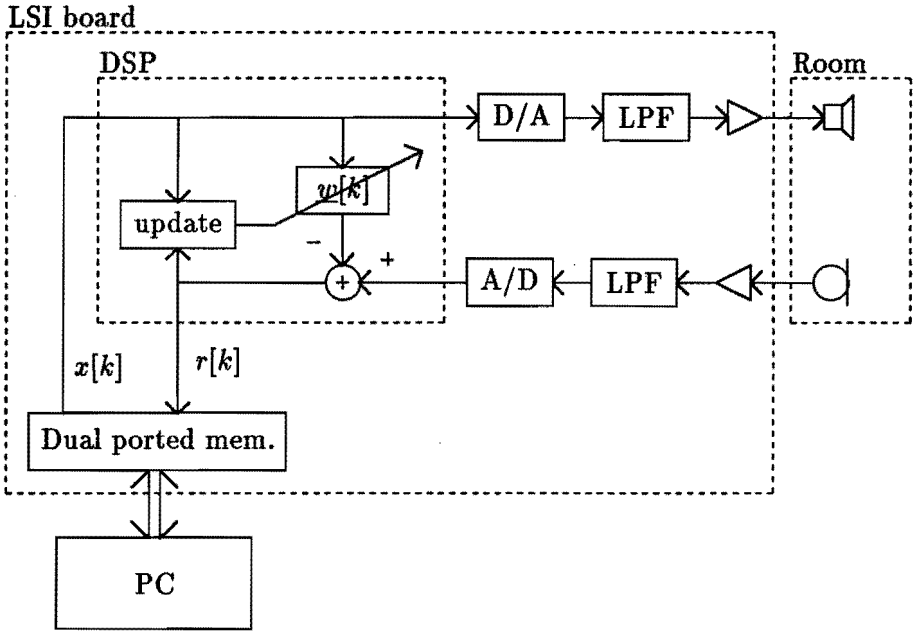



Figure 8.2: Hardware organisation.

be sent towards the loudspeaker). Communication between PC and DSP takes place via the Dual Ported memory, that can be accessed by the PC without halting the DSP.

### 8.2.2 Software

The PC controls the AEC software, running on the DSP. This AEC implementation consists of several modules:

- **Main part:** Initialization of the AEC.
- **Interrupt Service Routine (ISR):** Because of the very small block length in the filter part and large block length in the update part of the adaptive filter, (many) very small time frames to compute the update part are available. Therefore a complex interrupt scheme is needed to control the spread of the calculation over several time-frames. Interrupts are also needed for the A/D en D/A converters.
- **Filter part:** Calling (I)FFTs and performing elementwise multiplications. To obtain an efficient elementwise multiplication routine, all data must reside in the DSP's on chip internal memory. Because of

the limited amount of internal memory, the DMA (Direct Memory Access) has to run in parallel to transport data from the external (on board) memory to the internal memory (and back).

- **Update and coupling part:** Calling (I)FFTs and performing elementwise multiplications.

- **Stepsize control and normalization:** The stepsize influences the misadjustment and the speed of convergence. Initially, we prefer a fast convergence, implying a large stepsize parameter. When the residual signal decays, the stepsize parameter is decreased, to obtain a smaller misadjustment. When "double talk" occurs, the large residual signal implies a huge coefficient misadjustment or even instabillity. Adaption of the coefficients must be inhibited during "double talk". This implies that we need a "double talk" detector [4].

- **(I)FFT:** The (Real) (I)FFT routines available for the TMS320C30 processor were not efficient enough, so new routines have been developed based on [50], that are about 30% faster than previously available routines [4]. Note that a further improvement in speed can be obtained using the techniques described in appendix B.

### 8.2.3 Example System Settings

In principle the variables of the DPBFDAF algorithm can be chosen freely, although certain bounds exist to make the implementation less complicated. In table 8.1 these bounds are given, together with two possible variable sets as example. In the first instance sample frequencies of 8 and 16 kHz where chosen for examples 1 and 2, since these are telephony standards. However, during the implementation it became clear that these desired frequencies had to be reduced, because the total computational load did not fit within the DSP's capacity.

In table 8.2 the computational load of the two examples is given. The load of each seperate part is defined as a percentage of the DSP's total available number of instructions. The overhead parts contain some data-rearranging and initialization of loops. The table shows that most of the load is consumed in the filter part (caused by the small block length chosen there). This means that a significant reduction in computational complexity can be obtained by using NUPBFDAF instead of DPBFDAF.

| Variable | | | Maximum | Example 1 | Example 2 |
|---|---|---|---|---|---|
| Filter length | | $N$ | 8192 | 2016 | 2560 |
| Filter Part | Block length | $B$ | 256 | 8 | 64 |
| | Part. length | $Q$ | 256 | 56 | 192 |
| | FFT size | $M$ | 256 | 64 | 256 |
| Update Part | Block length | $A$ | 512 | 504 | 512 |
| | Part. length | $Z$ | 1024 | 504 | 512 |
| | FFT size | $L$ | 1024 | 1024 | 1024 |
| Sample frequency | | $f_s$ | | 7 kHz | 13 kHz |
| Echo path length | | | | 288 ms | 192 ms |
| Processing delay | | | | 1.6 ms | 6.5 ms |

Table 8.1: Maxima and examples for variable sets.

| Part | Rate | Operation | # per part | Ex.1 Load | Ex.2 Load |
|---|---|---|---|---|---|
| Filter part | $\frac{1}{B \cdot T}$ | $\text{FFT}_M$ | 1 | 5.0% | 6.3% |
| | | $\text{IFFT}_M$ | 1 | 5.9% | 7.4% |
| | | $\otimes$ | $N/Q$ | 58.3% | 16.7% |
| | | Overhead | 1 | 1.3% | 0.8% |
| Update part | $\frac{1}{A \cdot T}$ | $\text{FFT}_L$ | 2 | 4.6% | 8.3% |
| | | $\text{IFFT}_L$ | $N/Z$ | 10.3% | 20.9% |
| | | $\otimes$ | $N/Z$ | 1.5% | 2.7% |
| | | Overhead | 1 | 4.0% | 3.1% |
| Coupling | $\frac{1}{A \cdot T}$ | $\text{FFT}_M$ | $N/Q$ | 3.3% | 10.3% |
| | | Overhead | 1 | 0.2% | 0.9% |
| Stepsize control and Normalization | $\frac{1}{A \cdot T}$ | | 1 | 1.2% | 2.3% |
| Interrupt Service | $\frac{1}{T}$ | | 1 | 2.8% | 5.0% |
| Total | | | | 98.4% | 84.7% |

Table 8.2: Computational load of algorithm.

## 8.3 Tests

### 8.3.1 Test Conditions

The test room has dimensions $4.90 \times 3.75 \times 3.20\ \mathrm{m}^3$. The distance between the loudspeaker and microphone is 0.40 m. As we can see in figure 8.2, the AEC does not only have to cancel the actual echo path, but also the (non ideal) impulse responses of the D/A and A/D converters, the LPF's, the amplifiers, the loudspeaker and the microphone. As input signal we first use white noise, to show that the AEC indeed works. After that, the sentence "entering the forest without moving the grass" is taken as input signal and is depicted in figure 8.3. The non-stationarity of the input signal implies that we can not look at the residual signal only to investigate the performance, but that we have to normalize it to obtain the Echo Return Loss Enhancement, defined by $-10 \cdot \log_{10}(\mathcal{E}\{r^2[k]\}/\mathcal{E}\{\tilde{e}^2[k]\})$.

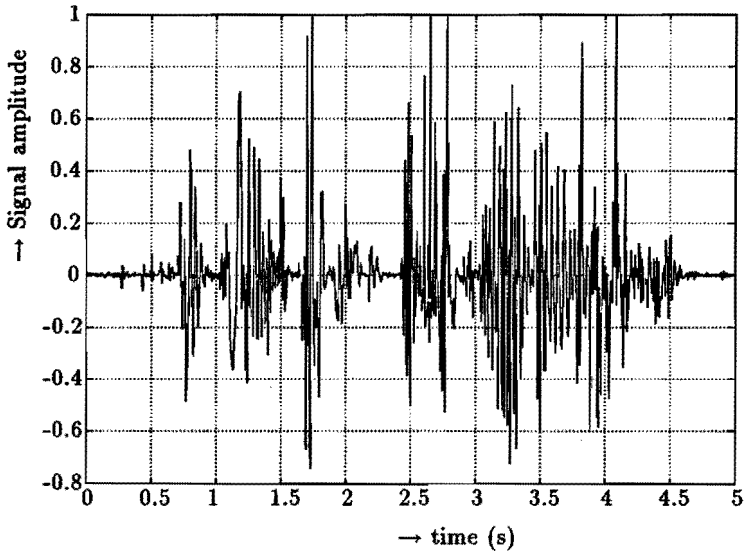



Figure 8.3: Speech signal: "entering the forest without moving the grass".

### 8.3.2 Test Results

In the first test, both examples have a white noise input signal, with the adaptive filter weights initialized to zero. The results are given in figure 8.4.

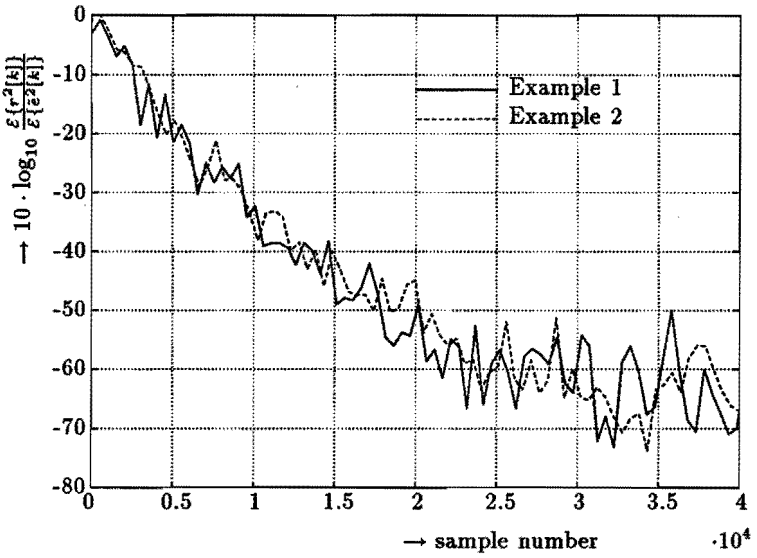



Figure 8.4: Convergence behaviour, $x[k]$ white noise.

The second experiments involves a sudden change in echo path, making it explicitely time variant (of course it never will be invariant using a real room as in this AEC). This is done by placing a hand between microphone and loudspeaker. In figure 8.5 the effect of this sudden change in the echo path impulse response is shown. Note that the sudden change does not take place at the same time in the examples. With a white noise signal, the AEC works very well, but it is not cancelling an *acoustic* echo caused by speech.

The third experiment involves speech. The sentence "entering the forest without moving the grass" is applied to the AEC (using the parameters of example 2), and the result is depicted in figure 8.6. Before the sentence was applied, the AEC was fully adjusted (to an approximately time invariant echo path). We see that when the input signal amplitude is high, good suppression of the echo is obtained (compare with figure 8.3). If we try to do an experiment with a jump on this input signal, it takes upto one minute, before the echo canceller reconverges to its steady state. This is caused by the fact that the normalization module is not capable of following the non-stationarities of the input speech signal. The power vector update rate is much to small compared to the length of a segment of a speech signal in which we may assume that the speech signal is stationary.

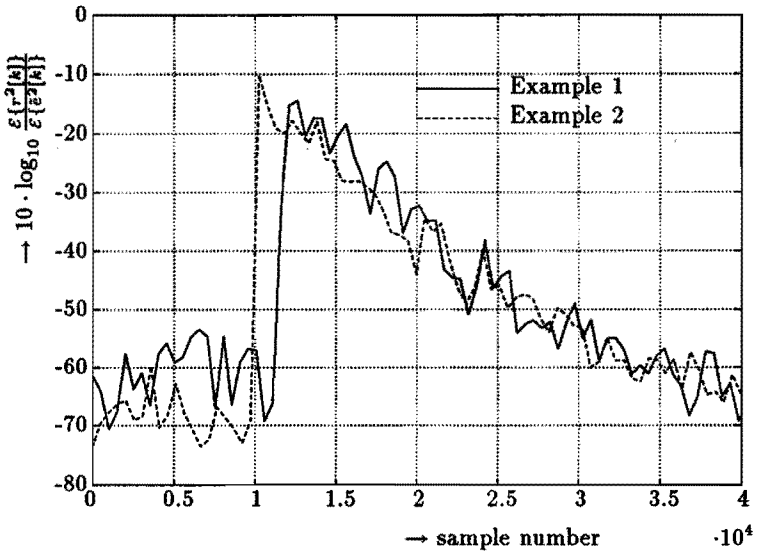



Figure 8.5: Convergence behaviour with jump, $x[k]$ white noise.

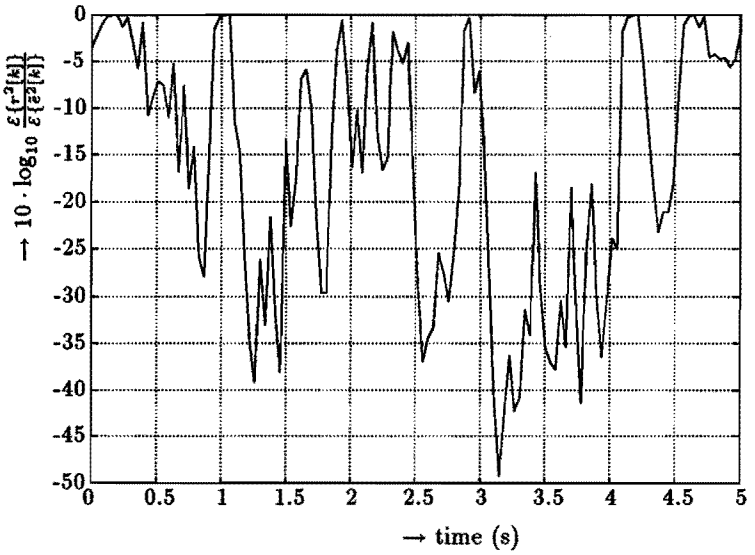



Figure 8.6: Convergence behaviour, $x[k]$ speech at 13 kHz.

## 8.4 Conclusions

A real-time implementation of a low delay acoustic echo-canceller can be realized on one Digital Signal Processor (the TMS320C30) using the Decoupled PBFDAF (DPBFDAF) algorithm. This implementation can be used to do further investigations on step-size and power estimation algorithms. Using a large FFT length in the update part to reduce computational complexity, implies that the variations in the input signal correlation can not be followed, especially when speech signals are used. This means in most cases that the convergence speed is reduced, so the quality of tracking variations in the echo-path is diminished. In chapter 6 methods of improving the normalization are discussed.

By using the Non Uniform PBFDAF algorithm the complexity in the filter part of the algorithm could be reduced even further, leading to implementations with a smaller processing delay. A second advantage is that more computation power is available for the update part, so smaller FFT lengths can be used there. This results in better normalization of non-stationary signals, implying an improvement on tracking variations in the echo-path.

# Chapter 9

# Conclusions

The huge complexity of the time domain algorithms currently makes implementation of large adaptive filters on one (or even a few) DSPs impossible. Besides that, the simplest algorithms, such as (Block) (Normalized) Least Means square ((B)(N)LMS), suffer from bad convergence behaviour when using highly coloured input signals. Time domain decorrelation with the inverse of an estimate of the input signal autocorrelation matrix improves convergence behaviour (in most cases), but also increases complexity.

For a large filter length it is known from literature [46, 30] that the Block Frequency Domain Adaptive Filter (BFDAF) has good convergence properties for relative low complexity, by performing the filtering operations and the decorrelation in frequency domain. By making a direct estimation of the inverse power vector, divisions can be avoided in frequency domain adaptive filtering. The degradation of convergence behaviour caused by the approximations made in frequency domain compared to time domain adaptive filtering, depend on the algorithm parameters and the input signal (auto-)correlation. By making the right choice in parameters, the effects of these approximations can be neglected.

However, when the processing delay is limited, low complexity can not be reached with the BFDAF approach. Besides that, the large FFTs imply a large computation processing delay, so if a small processing delay is required, BFDAF is not the right solution. Partitioning of the filter into smaller parts reduces the length of the FFTs needed. When a small processing delay is needed, the Partitioned BFDAF (PBFDAF) requires a lower computational complexity than the BFDAF at the cost of a lower decorrelation dimension. To gain the same decorrelation dimension as the BFDAF with a small processing delay and a computational complexity that is even

lower than with the PBFDAF, the Decoupled PBFDAF (DPBFDAF) is a good solution.

A real-time implementation of a low delay acoustic echo-canceller can be realized on one Digital Signal Processor (the TMS320C30) using this DPBFDAF algorithm. Using a large FFT length in the update part to reduce computational complexity, implies that the variation in the input signal (auto-)correlation can not be followed, especially when speech signals are used. This means in most cases that the convergence speed is reduced, so the quality of tracking variations in the echo-path becomes worse.

At cost of a small extra computational complexity, the power vector dimension and rate can be chosen independently of the update part dimensions and rate. This yields a more flexible algorithm, which can be of use in cases where the non-stationarity of the input signal is the main limiting factor.

However, even the DPBFDAF often has a computational complexity that is too high for real-time implementation purposes, especially when non-stationarity of the input signal is the main problem.

The introduced new method for fast real time convolution in frequency domain by using a non-uniform partitioning (NUPBFDC) reduces the required number of real multiplications compared to an approach using uniform partitioning.

For the case of adaptive filtering using the NUPBFDAF reduces computational complexity enormously, compared to implementations using the (D)(P)BFDAF. Complexity becomes almost independent on the maximum allowable delay. As a rule of thumb the number of real multiplications per sample equals approximately $50 \cdot \log_2(N) - 190$, with $N$ the number of filter coefficients.

By using the Non Uniform PBFDAF algorithm instead of the DPBFDAF algorithm in the AEC, the complexity of the filter part of the algorithm can be reduced, leading to implementations with a smaller processing delay. A second advantage is that more computation power is available for the update part, so smaller FFT lengths can be used there, which makes it possible to improve tracking of non-stationarities in the input signal correlation and the echo path impulse response.

# Appendix A

# Properties of Diverse Parts

*FFTs, elementwise multiplications and power vector estimation form the main part of frequency domain adaptive filters. Summation of their computational complexity, computation delay and memory occupation leads to the properties of the frequency domain adaptive filters as discussed in this thesis.*

## A.1 Transversal Filters

### A.1.1 Computation Processing Delay

Transversal filters consist of a delay line whose outputs are multiplied by filter coefficients. The multiplication results are summed up to form the output sample. As only one element of the delayline is new each sample, computation of all multiplications and additions but one, can be carried out during the previous sample interval. This implies that we only have to perform one multiplication and addition before a new ouput sample can be produced. This means that the computation delay of (B)(N)LMS algorithms is approximately zero, and certainly smaller than one ($D < 1$), provided that the computational complexity is not too high for real-time implementation.

## A.2 (I)FFTs

### A.2.1 Radix-2 FFTs

On most DSPs (depending on their architecture) the radix-2 Decimation In Time (DIT) algorithm seems a good choice [37]. In appendix B a further

improvement of this DIT FFT algorithm for partially overlapping FFTs is introduced, using a block sliding approach. We will not take this into account for our complexity measurement in this thesis. Here we assume the use of a radix-2 DIT algorithm operating on real input data. The calculation of the computational complexity is adapted to the implementation on a DSP. We will assume that an IFFT can be realized with the same number of real floating point multiplications (using a Decimation In Frequency (DIF) algorithm for real ouput data).

### A.2.2 Computational Complexity

A general radix-2 DIT FFT decomposition is given in appendix B, equation (B.4). The total number of real multiplications $\Psi_{\text{FFT}}\{M\}$ needed to compute a length $M$ FFT thus equals

$$\Psi_{\text{FFT}}\{M\} = 2 \cdot \Psi_{\text{FFT}}\{M/2\} + \Psi_{\text{STAGE}}\{M\} \tag{A.1}$$

where the computational complexity of combining two length $M/2$ FFTs to one length $M$ FFT equals $\Psi_{\text{STAGE}}\{M\}$. As for $M \leq 4$ an FFT contains only additions, we get for all $M \leq 4$

$$\Psi_{\text{FFT}}\{M\} = 0. \tag{A.2}$$

The complexity involved in combining two FFTs is caused by an elementwise multiplication where both vectors have special symmetry properties (see appendix B). One can deduce that the next number of multiplications suffices to compute this elementwise multiplication

$$\Psi_{\text{STAGE}}\{M\} = 4 \cdot (\frac{M}{4} - 1). \tag{A.3}$$

Combination of equations (A.3) and (A.1) gives

$$\Psi_{\text{FFT}}\{M\} = M \cdot \log_2 \frac{M}{8} + 4. \tag{A.4}$$

### A.2.3 Computation Processing Delay

The computation processing delay of the FFT depends on the specific hardware that is used. In the frequency domain adaptive filters presented in this thesis, we need to know the computation delay of two FFTs and one elementwise multiplication. The summation of these can be found in the next section A.3.

### A.2.4 Memory Occupation

Radix-2 DIT FFTs for real input data can be performed "in place", which means that we need $M$ memory locations for a length $M$ FFT. Besides that we need to store the "twiddle factors", the elements of the Fourier matrix. The number of different real and imaginary parts of these "twiddle factors" different from $\pm 1$, equals $M/4 - 1$. Efficient referencing in DSP implementations requires the reservation of more memory. To be on the save side, lets say at most $M$ memory locations.

## A.3 Elementwise Multiplication

### A.3.1 Computational Complexity

Both the complex vectors, whose elements are multiplied elementwise, are Fourier transforms of real valued vectors, and thus have symmetry properties. This means that the 0'th and $M/2$'th of a length $M$ vector are real, and that for all $0 < j < M/2$ the $M - j$'th element is the complex conjugate of the $j$'th element. As for all complex numbers $c_a$ and $c_b$ the property $(c_a)^* \cdot (c_b)^* = (c_a \cdot c_b)^*$ holds, the result of the elementwise multiplication also has the symmetry property. This implies that only half of the frequency components have to be calculated (and stored).

A complex multiplication can be realized with 4 multiplications and 2 additions or 3 multiplications and 5 additions. We will assume the use of the first option here, as it takes the least number of operations (although of course the number of real multiplications per sample is larger). This all implies for the elementwise multiplication of two length $M$ vectors that

$$\begin{aligned} \Psi_{\otimes}\{M\} &= 4 \cdot (\frac{M}{2} - 1) + 2 \\ &= 2 \cdot M - 2. \end{aligned} \tag{A.5}$$

### A.3.2 Computation Processing Delay

The computation processing delay for the computation of a real FFT, an IFFT and one elementwise vector multiplication defines the processing delay of the frequency domain adaptive filtering algorithms. In partitioned structures we can see the delayline in frequency domain as a transversal filter (the name time-frequency domain filtering is often used in literature). Like in the time domain transversal filters, also here the older delay-line elements are processed during the previous interval.

As the computation processing delay is hardware dependent, we cannot give a general expression for it. Using a sample frequency of 8000 Hertz, and implementing the FFTs on the TMS320C30 signal processor yields the delays of table A.1.

| $M$ | $\leq 64$ | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|---|---|
| $2 \cdot D_{\mathrm{FFT}}\{M\}$ $+D_{\otimes}\{M\}$ | 1 | 2 | 4 | 8 | 18 | 38 | 80 | 176 |

Table A.1: Processing delay for FFTs and multiplications.

## A.4 Power Vector Estimation

### A.4.1 Computational Complexity

In chapter 6, equation (6.63) the next method for direct inverse power estimation was mentioned

$$\begin{aligned} \mathcal{P}_i^{-1}[\kappa B] &= (1+M\beta)(\hat{\underline{P}}_x^M[(\kappa-1)B])_i^{-1} - \\ &\quad \beta(\hat{\underline{P}}_x^M[(\kappa-1)B])_i^{-2}|(\underline{X}^M[\kappa B])_i|^2 \end{aligned} \tag{A.6}$$

As $|(\underline{X}^M[\kappa B])_i|^2 = (\Re\{(\underline{X}^M[\kappa B])_i^2\} - (\Im\{(\underline{X}^M[\kappa B])_i^2\}$, and $\underline{X}^M[\kappa B]$ is the DFT of a real valued vector, we need $2(M/2-1)+2 = M$ multiplications to calculate all $|(\underline{X}^M[\kappa B])_i|^2$. The result are $M/2+1$ different real values, so we need $4 \cdot (M/2+1)$ multiplications for the rest of the inverse power estimation equation. This inverse power has to be multiplied by a complex valued vector, which requires $2+(M/2-1)\cdot 2$ multiplications. Summing up the number of multiplications gives the total complexity $\Psi_P\{M\}$

$$\Psi_P\{M\} = 4M + 4. \tag{A.7}$$

### A.4.2 Memory Occupation

The storage of all $(\hat{\underline{P}}_x^M[(\kappa-1)B])_i^{-1}$, the storage of the intermediate result of all $|(\underline{X}^M[\kappa B])_i|^2$ and the two constants requires a total of $2\cdot(M/2+1)+2 = M+4$ memory locations.

# Appendix B

# Recursive Computation of Block FFTs

*In many applications, such as block frequency domain adaptive filtering, Fast Fourier Transforms (FFTs) are used. Because of reducing complexity in other parts, FFTs take an increasing amount of the computational complexity in such applications. These FFTs often have an input sequence that is partially zero, or, in the inverse case, yield output sequences that are only partially used. These properties can be used to reduce the computational burden of the (I)FFTs [41, 52]. Furthermore often FFTs are used that have an input sequence that partially overlaps with the input sequence of a previously computed FFT. These "block" FFTs can be computed recursively with help of the partially zero input sequence FFTs ("windowed" FFTs) [20]. We introduce a new concept that is an extension to the sliding approach (where the overlap equals the FFT length minus one) [51, 1]. By defining the block length as the FFT length minus the overlap, we perform a rotation of the input sequence over this block length and use a "windowed" FFT with window length equal to this block length to compute "block" FFTs.*

*This new method for recursive computation of "block" FFTs decreases computational complexity compared to direct computation methods. In applications where these FFTs are the major computational burden, for example adaptive filtering algorithms based on Block Frequency Domain computation (see chapters 3, 4 and 5) this leads to a huge decrease in overall complexity.*

## B.1 Introduction

The implementation of a Fourier Transform, even a Fast one, takes a lot of computations. In many applications, such as block frequency domain adaptive filtering, Fast Fourier Transforms (FFTs) are used. Because of reducing complexity in other parts, the FFTs take an increasing amount of the computational complexity in such applications (see chapters 3, 4 and 5). These FFTs often have an input sequence that is partially zero, or, in the inverse case, yield output sequences that are partially used. This can be used to reduce the computational burden of the (I)FFTs [41, 52].

A third group of FFTs where the computational complexity can be reduced, are the "block" FFTs. "Block" FFTs are defined as FFTs whose input overlaps partially with the input of a previously computed FFT. The FFTs with partially zero input sequence ("windowed" FFTs) can be used in recursive computation of "block" FFTs. Therefore first a short derivation of the "windowed" real FFTs and their complexity is given. After that the new generalized recursive implementation is explained.

As a measure of computational complexity, in this appendix the sum of the number of real additions and multiplications is used. This measure is chosen because the windowing does not have the same effect on the number of multiplications as on the number of additions. As stated in the introduction (chapter 1), multiplications and additions in general take the same processing time (or can be performed simultaneously) on Digital Signal Processors (DSPs). For a fair comparison between the recursive algorithms introduced here and the algorithms that do not exploit the a prioiri knowledge, a sum of multiplications and additions seems a better choice than just looking at the number of multiplications.

## B.2 "Windowed" Real FFTs

### B.2.1 General Windowing

A "windowed" real FFT is defined as a length $M$ real FFT from which the last $M - B$ input samples are zero, thus

$$\begin{aligned} \underline{X}^M &= \mathcal{F}^M \underline{x}^M \\ &= \mathcal{F}^M \begin{pmatrix} \underline{y}^B \\ \underline{0}^{M-B} \end{pmatrix} \end{aligned} \tag{B.1}$$

with

$$\underline{x}^M = \begin{pmatrix} x_0 & \cdots & x_{M-1} \end{pmatrix}^t \tag{B.2}$$

$$\underline{y}^B = \left( y_0 \ \cdots \ y_{B-1} \right)^t. \tag{B.3}$$

The $M \times M$ Fourier matrix is denoted as $\mathcal{F}^M$, with its $(a,b)$'th element $(\mathcal{F}^M)_{a,b}$ defined as $(\mathcal{F}^M)_{a,b} = e^{-\jmath 2\pi \frac{ab}{M}}$. Both the FFT length $M$ and the block length $B$ are assumed to be powers of 2. This assumption makes it possible to calculate the FFT efficiently by using a recursive scheme. In [41, 52] it is shown that by using Decimation-In-Time (DIT) the windowing effect with real data can be exploited. On most DSPs (depending on their architecture) the radix-2 algorithm seems a good choice [37]. In the next section an example of such a DIT radix-2 FFT is given. The calculation of the computational complexity of such an algorithm operating on real data is adapted to the implementation on a DSP.

### B.2.2 Radix-2 DIT Real FFT

A general radix-2 length $M$ DIT FFT decomposition ($M$ is a power of 2) of the FFT of equation (B.1) is defined for all $0 \leq l < M$ as

$$\begin{aligned} X_l &= \sum_{k=0}^{M-1} x_k e^{-\jmath 2\pi \frac{kl}{M}} \\ &= \sum_{k=0}^{\frac{M}{2}-1} x_{2k} e^{-\jmath 2\pi \frac{kl}{M/2}} + e^{-\jmath 2\pi \frac{l}{M}} \sum_{k=0}^{\frac{M}{2}-1} x_{2k+1} e^{-\jmath 2\pi \frac{kl}{M/2}} \end{aligned} \tag{B.4}$$

with $X_l = (\underline{X}^M)_l$. We see in equation (B.4) that the length $M$ FFT is splitted into two length $M/2$ FFTs. For the number of real floating point multiplications and additions $\Psi_{\text{WIN}}\{M,B\}$ of a "windowed" length $M$ FFT with $B$ non-zero input samples this implies that for $M > 4$ and $B > 2$

$$\Psi_{\text{WIN}}\{M,B\} = 2 \cdot \Psi_{\text{WIN}}\{\frac{M}{2}, \frac{B}{2}\} + \Psi_{\text{S1}}\{M\}. \tag{B.5}$$

Note that in the above recursive scheme every FFT has the same relative number of non-zero input data (the windowing properties are kept). In equation (B.5) we see that $\Psi_{\text{S1}}\{M\}$ operations are needed for the combining of two length $M/2$ FFTs to one length $M$ FFT. This combining consists of two parts: the elementwise multiplication of the result of one of the length $M/2$ FFTs by the elements of the Fourier matrix $\mathcal{F}^M$ and the addition of this elementwise multiplication result and the result of the other length $M/2$ FFT. As both length $M/2$ FFTs are transforms of real vectors, only $(M/4) - 1$ complex and 2 real values are relevant. Knowing that one

complex multiplication takes 4 real multiplications and 2 real additions, we can derive the next formula for the number of real floating point addditions and multiplications $\Psi_{S1}\{M\}$ to combine two length $M/2$ FFTs

$$\begin{aligned}\Psi_{S1}\{M\} &= 4\cdot(\frac{M}{4}-1)+2\cdot(\frac{M}{4}-1)+4\cdot(\frac{M}{4}-1)+2\\ &= \frac{5}{2}M-8. \end{aligned} \tag{B.6}$$

If the number of non zero input samples $B$ equals the FFT length $M$ (no windowing), then the last FFT to be computed in the recursive scheme is the length 4 FFT. The twiddle factors (elements of the Fourier matrix) appearing in that stage are all $\pm 1$ or $\pm j$. This implies that only additions remain for the computation of such a length 4 FFT. The remaining number of additions $\Psi_{WIN}\{4,4\}$ then equals

$$\Psi_{WIN}\{4,4\} = 6. \tag{B.7}$$

In the case where part of the input data is zero ($B$ not equal to $M$) we proceed with our recursive scheme until the block length equals 2. This implies that in the last stage the FFTs have length $2\cdot M/B$ with only two non-zero inputs. Equation (B.1) can then be simplified for all $0 \le l < 2M/B$ to

$$\begin{aligned}X_l &= \sum_{k=0}^{\frac{2M}{B}-1} x_k e^{-j2\pi\frac{kl}{2M/B}}\\ &= x_0 + x_1 e^{-j2\pi\frac{l}{2M/B}}. \end{aligned} \tag{B.8}$$

The number of different non-trivial (not equal to $\pm 1$ or $\pm j$) real or imaginary parts of the twiddle factors (Fourier matrix elements) is $M/(2B)-1$, so $M/(2B)-1$ real multiplications are needed. As $x_0$ is real, only the real parts have to be added, which needs $M/B$ adds. For the total number of operations $\Psi_{WIN}\{2M/B,2\}$ needed for $B < M$, the above implies that

$$\Psi_{WIN}\{\frac{2M}{B},2\} = \frac{3M}{2B}-1. \tag{B.9}$$

By using the recursive scheme, we can now compute the overall complexity. Again we have two cases, $B = M$ and $B < M$. In the case of $B = M$ (no windowing), the resulting number of operations equals

$$\Psi_{WIN}\{M,M\} = \frac{M}{4}\Psi_{WIN}\{4,4\} + \sum_{b=1}^{\log_2(\frac{M}{4})} 2^{b-1}\Psi_{S1}\{\frac{M}{2^{b-1}}\}$$

$$= \frac{7M}{2} + \frac{5M}{2}\log_2(\frac{M}{4}) - 8. \tag{B.10}$$

In the "windowed" case, where $B < M$ the resulting number of operations is

$$\begin{aligned}\Psi_{\mathrm{WIN}}\{M,B\} &= \frac{B}{2}\Psi_{\mathrm{WIN}}\{\frac{2M}{B},2\} + \sum_{b=1}^{\log_2(\frac{B}{2})} 2^{b-1}\Psi_{\mathrm{S1}}\{\frac{M}{2^{b-1}}\} \\ &= \frac{3M}{4} + \frac{7B}{2} + \frac{5M}{2}\log_2(\frac{B}{2}) - 8. \end{aligned} \tag{B.11}$$

# B.3 Recursive Computation of "Block" FFTs

## B.3.1 Definition

In [51, 1] "sliding" FFTs are introduced. With the help of the results for "windowed" FFTs we can extend that concept of "sliding" FFTs to a method for recursive computation of "block" FFTs. We assume that on a sampled time signal $x[k]$ each $B$ samples a length $M$ FFT of its last $M$ samples has to be performed, then

$$\mathcal{F}^M \underline{x}^M[(k+1)B] = \mathcal{F}^M \mathbf{D}_B^M \left( \underline{x}^M[kB] + \underline{y}_B^M[kB] \right) \tag{B.12}$$

with for all $M$, $B$ and $k$

$$\underline{x}^M[kB] = \left( x[kB-M+1] \quad \cdots x[kB] \right)^t \tag{B.13}$$

$$\underline{y}_B^M[kB] = \begin{pmatrix} \underline{x}^B[(k+1)B] - \underline{x}^B[kB-M+B] \\ \underline{0}^{M-B} \end{pmatrix}. \tag{B.14}$$

The rotation matrix $\mathbf{D}_B^M$ rotates the vector on its right hand $B$ places upward. This can be accomplished by defining this $B$ place upward rotation matrix $\mathbf{D}_B^M$ as

$$\mathbf{D}_B^M = \begin{pmatrix} \mathbf{0}^{M-B,B} & \mathbf{I}^{M-B} \\ \mathbf{I}^B & \mathbf{0}^{B,M-B} \end{pmatrix}. \tag{B.15}$$

Note that the rotation matrices are recursively related by, for $B > 1$

$$\begin{aligned}\mathbf{D}_B^M &= \mathbf{D}_1^M \mathbf{D}_{B-1}^M \\ &= (\mathbf{D}_1^M)^B. \end{aligned} \tag{B.16}$$

To obtain an efficient recursive implementation, equation (B.12) is written as

$$\begin{aligned}&\mathcal{F}^M \mathbf{D}_B^M (\underline{x}^M[kB] + \underline{y}_B^M[kB]) = \\ &\quad \mathcal{F}^M \mathbf{D}_B^M (\mathcal{F}^M)^{-1} (\mathcal{F}^M \underline{x}^M[kB] + \mathcal{F}^M \underline{y}^M[kB]). \end{aligned} \tag{B.17}$$

The above equation requires the following operations

1. A time-domain rotation

$$\mathcal{F}^M \mathbf{D}_B^M (\mathcal{F}^M)^{-1}. \tag{B.18}$$

2. The result of the previously calculated FFT

$$\mathcal{F}^M \underline{x}^M[kB]. \tag{B.19}$$

3. A "windowed" FFT

$$\mathcal{F}^M \underline{y}_B^M[kB] = \mathcal{F}^M \begin{pmatrix} \underline{x}^B[(k+1)B] - \underline{x}^B[kB - M + B] \\ \underline{0}^{M-B} \end{pmatrix}. \tag{B.20}$$

The "windowed" FFT is described in the previous section, the previous FFT is available and the time domain rotation is introduced in the next section.

### B.3.2 Rotation

The time domain rotation can be computed efficiently in frequency domain. As the time-domain matrix $\mathbf{D}_1^M$ is a circulant matrix, the result of a pre- and post-multiplication by the $M \times M$ Fourier matrix and its inverse will be a diagonal matrix [9]. The diagonal of that matrix equals the Fourier transform of the first column of the time domain matrix. Thus

$$\mathcal{F}^M \mathbf{D}_1^M (\mathcal{F}^M)^{-1} = \mathrm{diag}\{\underline{v}^M\} \tag{B.21}$$

with

$$\underline{v}^M = \mathcal{F}^M \begin{pmatrix} \underline{0}^{M-1} \\ 1 \end{pmatrix}. \tag{B.22}$$

where the $\kappa$'th element of vector $\underline{v}^M$ is defined as

$$(\underline{v}^M)_\kappa = e^{+\jmath \frac{2\pi\kappa}{M}}. \tag{B.23}$$

As $\mathcal{F}^M \mathbf{D}_B^M (\mathcal{F}^M)^{-1} = (\mathcal{F}^M \mathbf{D}_1^M (\mathcal{F}^M)^{-1})^B$, the $\kappa$'th element of the main diagonal of $\mathcal{F}^M \mathbf{D}_B^M (\mathcal{F}^M)^{-1}$ equals $((\underline{v}^M)_\kappa)^B$. For the diagonal matrix $\mathcal{F}^M \mathbf{D}_B^M (\mathcal{F}^M)^{-1}$ then the next theorem can be deduced

$$\mathcal{F}^M \mathbf{D}_B^M (\mathcal{F}^M)^{-1} = \mathrm{diag}\{\begin{pmatrix} \underline{v}^{\frac{M}{B}} \\ \vdots \\ \underline{v}^{\frac{M}{B}} \end{pmatrix}\}. \tag{B.24}$$

Using the above equations, the rotation reduces to an elementwise multiplication. Using the special properties of the Fourier vector, we can deduce that the number of real operations needed to multiply $\underline{v}^Q$ elementwise with a length $Q$ complex vector with arbitrary elements is 0 for $Q < 8$, and equals $4Q - 24$ multiplications and $2Q - 8$ additions for $Q \geq 8$, thus a total $6Q - 32$ real operations.

Knowing that the rotation contains $B$ length $M/B$ Fourier vectors and that both the left and right operand of the elementwise multiplication are Fourier transforms of real valued vectors (so the second part is complex conjugate of the first part) the total rotation requires $\Psi_{\text{ROT}}\{M, B\}$ real operations

$$\Psi_{\text{ROT}}\{M, B\} = \begin{cases} 0 & \text{for } M/B < 8 \\ 3M - 16B & \text{for } M/B \geq 8 \end{cases} . \tag{B.25}$$

### B.3.3 Total Complexity of Recursive "Block" FFTs

The total complexity for the recursive implementation of the "block" FFTs is obtained by summation of equation (B.25), (B.10) and (B.11)

$$\begin{aligned} \Psi_{\text{REC}}\{M, B\} &= \Psi_{\text{WIN}}\{M, B\} + \Psi_{\text{ROT}}\{M, B\} \qquad \text{(B.26)} \\ &= \begin{cases} \frac{M}{2} + \frac{5M}{2}\log_2(\frac{M}{2}) - 8 & \text{for } B = M \\ \frac{3M}{4} + \frac{7B}{2} + \frac{5M}{2}\log_2(\frac{B}{2}) - 8 & \text{for } \frac{M}{4} \leq B < M \\ \frac{15M}{4} - \frac{25B}{2} + \frac{5M}{2}\log_2(\frac{B}{2}) - 8 & \text{for } B \leq \frac{M}{8} \end{cases} . \end{aligned}$$

## B.4 "Windowed" Real Output IFFTs

### B.4.1 General Windowing

Windowing means here that only part of the output has to be computed. In [11] the problem of an FFT with real input data and partial use of the output nodes is adressed, but here we have real output data, so that method cannot be used. A general real output IFFT is defined for all $0 \leq l < M$ by

$$x_l = \sum_{k=0}^{M-1} X_k \cdot e^{+j \cdot 2\pi \cdot \frac{k \cdot l}{M}} \tag{B.27}$$

with $X_k = (\underline{X}^M \mathcal{F}^M)_k$ and $x_l = (\underline{x}^M)_l$.

As the problem is the inverse of the "windowed" real input FFT, here we should use a Decimation-In-Frequency (DIF) algorithm (DIF means in this case also that the time domain vector is decimated, but this is the output vector here (and not the input vector, as in the FFT case)).

### B.4.2 Radix-2 DIF Real Output (I)FFT

A general radix-2 DIF decomposition is defined for all $0 \leq l < M/2$ as

$$\begin{aligned}
x_{2l} &= \sum_{k=0}^{M-1} X_k \cdot e^{-j\cdot 2\pi\cdot\frac{k\cdot(2l)}{M}} \\
&= \sum_{k=0}^{\frac{M}{2}-1} (X_k + X_{\frac{M}{2}+k}) \cdot e^{-j\cdot 2\pi\cdot\frac{k\cdot l}{M/2}} \\
&= \sum_{k=0}^{\frac{M}{2}-1} (X_k + X^*_{\frac{M}{2}-k}) \cdot e^{-j\cdot 2\pi\cdot\frac{k\cdot l}{M/2}} \qquad \text{(B.28)} \\
x_{2l+1} &= \sum_{k=0}^{M-1} X_k \cdot e^{-j\cdot 2\pi\cdot\frac{k\cdot(2l+1)}{M}} \\
&= \sum_{k=0}^{\frac{M}{2}-1} (X_k \cdot e^{-j\cdot 2\pi\cdot kM} + X_{\frac{M}{2}+k} \cdot e^{-j\cdot 2\pi\cdot\frac{\frac{M}{2}+k}{M}}) \cdot e^{-j\cdot 2\pi\cdot\frac{k\cdot l}{M/2}} \\
&= \sum_{k=0}^{\frac{M}{2}-1} (X_k - X^*_{\frac{M}{2}-k}) \cdot e^{-j\cdot 2\pi\cdot\frac{k}{M}} \cdot e^{-j\cdot 2\pi\cdot\frac{k\cdot l}{M/2}}. \qquad \text{(B.29)}
\end{aligned}$$

The total complexity of a length $M$ IFFT of which the first $B$ output values are needed thus is $\Psi_{\text{IFFT}}\{M, B\}$, where both $B$ and $M$ are powers of 2

$$\Psi_{\text{IFFT}}\{M, B\} = 2 \cdot \Psi_{\text{IFFT}}\{\frac{M}{2}, \frac{B}{2}\} + \Psi_{\text{S2}}\{M\}. \qquad \text{(B.30)}$$

The combining equations take $\Psi_{\text{S2}}\{M\}$ operations. For equation (B.28) we need to add $X_k + X^*_{\frac{M}{2}-k}$ for $0 \leq k \leq M/4$ (symmetry poperties are kept), which requires $M/2$ real adds. For equation (B.29) we need to add $X_k - X^*_{\frac{M}{2}-k}$, whose imaginary part equals to the imaginary part of $X_k + X^*_{\frac{M}{2}-k}$, thus only $M/4$ additions are needed. The multiplication requires $6\cdot(M/4-1)$ operations so the total number of operations, needed to combine two length $M/2$ IFFTs equals

$$\Psi_{\text{S2}}\{M\} = \frac{9M}{4} - 6 \qquad \text{(B.31)}$$

If the number $B$ of output samples to be computed, equals the IFFT length $M$, or half of it ($M/2$), then we proceed with our recursive scheme until we

reach IFFTs of length 4, with

$$x_0 = \Re\{X_0\} + \Re\{X_2\} + 2 \cdot \Re\{X_1\} \tag{B.32}$$
$$x_1 = \Re\{X_0\} - \Re\{X_2\} + 2 \cdot \Im\{X_1\} \tag{B.33}$$
$$x_2 = \Re\{X_0\} + \Re\{X_2\} - 2 \cdot \Re\{X_1\} \tag{B.34}$$
$$x_3 = \Re\{X_0\} - \Re\{X_2\} - 2 \cdot \Im\{X_1\}. \tag{B.35}$$

This implies

$$\Psi_{\text{IFFT}}\{4,4\} = 8 \tag{B.36}$$
$$\Psi_{\text{IFFT}}\{4,2\} = 6 \tag{B.37}$$
$$\tag{B.38}$$

If $M \geq 4B$ then we proceed till the stage in which 2 values per length $M$ IFFT must be computed, implying that we have length $2M/B$ IFFTs, with

$$x_0 = \Re\{X_0\} + \Re\{X_{\frac{M}{B}}\} + 2 \cdot \sum_{k=1}^{\frac{M}{B}-1} \Re\{X_k\} \tag{B.39}$$

$$x_1 = \Re\{X_0\} - \Re\{X_{\frac{M}{B}}\} + 2 \cdot \sum_{k=1}^{\frac{M}{B}-1} \Re\{X_k \cdot e^{-j \cdot 2\pi \cdot \frac{kB}{2M}}\}. \tag{B.40}$$

The above equations cost in total $\Psi_{\text{IFFT}}\{2M/B, 2\}$ operations

$$\Psi_{\text{IFFT}}\{\frac{2M}{B}, 2\} = \frac{3M}{B} + 3 \tag{B.41}$$

By using the recursive scheme, we can now compute the overall complexity. We have three cases, $M \geq 4B$, $M = 2B$ and $M = B$

$$\begin{aligned}\Psi_{\text{IFFT}}\{M, M\} &= \Psi_{\text{IFFT}}\{4,4\} \cdot \frac{M}{4} + \sum_{b=1}^{\log_2(\frac{M}{4})} 2^{b-1} \cdot \Psi_{S2}\{\frac{M}{2^{b-1}}\} \\ &= \frac{9M}{4} \cdot \log_2(\frac{M}{4}) + \frac{M}{2} + 6 \end{aligned} \tag{B.42}$$

$$\begin{aligned}\Psi_{\text{IFFT}}\{M, \frac{M}{2}\} &= \Psi_{\text{IFFT}}\{4,2\} \cdot \frac{M}{4} + \sum_{b=1}^{\log_2(\frac{M}{4})} 2^{b-1} \cdot \Psi_{S2}\{\frac{M}{2^{b-1}}\} \\ &= \frac{9M}{4} \cdot \log_2(\frac{M}{4}) + 6 \end{aligned} \tag{B.43}$$

$$\begin{aligned}\Psi_{\text{IFFT}}\{M,B\} &= \Psi_{\text{IFFT}}\{\frac{2M}{B},2\}\cdot\frac{B}{2}++\sum_{b=1}^{\log_2(\frac{B}{2})}2^{b-1}\cdot\Psi_{\text{S2}}\{\frac{M}{2^{b-1}}\}\\ &= \frac{9M}{4}\cdot\log_2(\frac{B}{2})+\frac{3M}{2}-\frac{3B}{2}+6 \qquad \text{(B.44)}\end{aligned}$$

## B.5 Results

### B.5.1 Gain Compared to Traditional FFTs

In figure B.1 the computational complexity of the the "windowed" FFT is given as a function of the block length B for three different FFT lengths (the solid lines). We observe a decreasing complexity as the window length decreases. For a block length $B = M$ (the right-edge of each curve), we obtain the complexity of FFTs that do not exploit the windowing property. At cost of a small extra complexity (the rotation) we can use the "windowed" FFTs in the recursively computed "block" FFTs (the dashed lines), for an overlap of $M - B$. We see that for an increasing overlap (decreasing $B$) complexity is reduced enormously compared to the traditional computed FFTs (for $B = M$ we get the complexity of the traditional approach).

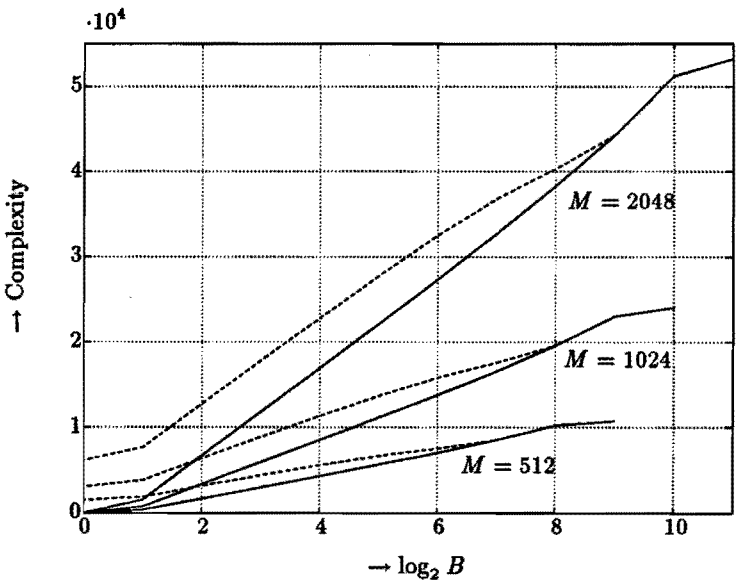



Figure B.1: Complexity as function of $B$ and $M$.

### B.5.2 Application in Adaptive Filter: PBFDAF

In adaptive filtering the use of "windowed" and recursive "block" FFTs yields a huge reduction of the number of operations needed. Take, for example, the Partitioned Block Frequency Domain Adaptive Filter (PBFDAF) of chapter 4 used as acoustic echo-canceller. In [10] the suggestion of using "windowed" FFTs in Frequency Domain Adaptive Filters is made. However an incorrect assumption about the resulting complexity is made, a 50% zero input vector does not result into a 50% saving in complexity.

We assume to have signals sampled at 8kHz, and an echo to be cancelled of 250 milli-seconds (inducing a filter of at least 2000 coefficients). Further we assume that the maximum allowable processing delay is 1 milli-second. With traditional FFTs, the lowest attainable number of operations is then reached for a filter length of 2004 coefficients partitioned into 167 subfilters of length 12. This costs approximately 14000 operations per sample. Using "windowed" and recursive "block" FFTs, the smallest complexity is obtained for a filterlength of 2040 coefficients partitioned into 34 subfilters of length 60. The adaptive filter then needs approximately 8500 operations per sample. In both cases the block length equals 4. We see that the use of these FFTs reduces the number of operations needed with 40%.

## B.6 Conclusions

The introduced new method for recursive computation of "block" Fast Fourier Transforms decreases computational complexity compared to the straight approach. In applications where these FFTs are the major computational burden, this can lead to a huge reduction in overall complexity.

# Appendix C

# Power Spectral Density

## C.1 Definition of Power Spectral Density

Consider a signal $x[k]$, that represents a single realization of a wide-sense stationary stochastic process, with $\mathcal{E}\{x[k]\} = 0$. The discrete-time Fourier transform of such a signal at time $k$ is given for $\omega \in [-\pi, \pi]$ by

$$X_{M,\omega}[k] = \sum_{m=0}^{M-1} x[k-m] \cdot e^{-\jmath\omega m}. \tag{C.1}$$

The statistical expectation of the squared magnitude of $X_M[k,\omega]$ can be expressed as follows

$$\begin{aligned}
\mathcal{E}\{|X_{M,\omega}[k]|^2\} &= \mathcal{E}\{X_{M,\omega}[k] \cdot (X_M[k,\omega])^*\} \\
&= \mathcal{E}\left\{\sum_{n=0}^{M-1}\sum_{m=0}^{M-1} x[k-n] \cdot (x[k-m])^* \cdot e^{-\jmath\omega(n-m)}\right\} \\
&= \sum_{n=0}^{M-1}\sum_{m=0}^{M-1} \mathcal{E}\{x[k-n] \cdot (x[k-m])^*\} \cdot e^{-\jmath\omega(n-m)} \\
&= \sum_{n=0}^{M-1}\sum_{m=0}^{M-1} \rho_{n-m} \cdot e^{-\jmath\omega(n-m)}. \qquad \text{(C.2)}
\end{aligned}$$

By letting $a = m - n$, we rewrite the above equation as follows

$$\mathcal{E}\{|X_{M,\omega}[k]|^2\} = M \cdot \sum_{a=-M+1}^{M-1} (1 - \frac{|a|}{M}) \cdot \rho_a \cdot e^{-\jmath\omega a} \tag{C.3}$$

which can be interpreted as the discrete-time Fourier transform of a windowed autocorrelation function. Now the Power Spectral Density (PSD) of $x[k]$ can be defined as

$$\begin{aligned} S_{x,\omega} &= \lim_{M\to\infty} \frac{1}{M}\mathcal{E}\{|X_{M,\omega}[k]|^2\} \\ &= \sum_{a=-infty}^{\infty} \rho_a \cdot e^{-\jmath\omega a}. \end{aligned} \tag{C.4}$$

## C.2 DFTs and Non-Stationarities

By using DFTs of length $M$ instead of Fourier transforms, we can obtain discrete functions defined for $0 \leq m < L$ by

$$(\underline{X}^L[k])_m = \sum_{n=0}^{M-1} x[k-n] \cdot e^{-\jmath 2\pi \frac{m \cdot n}{M}} \tag{C.5}$$

$$\begin{aligned} (\underline{P}_x^M)_m &= \frac{1}{M}\mathcal{E}\{(\underline{X}^M[k])_m \cdot ((\underline{X}^M[k])^*)_m\} \\ &= \sum_{a=-M+1}^{M-1} (1 - \frac{|a|}{M}) \cdot \rho_a \cdot e^{-\jmath 2\pi \frac{a \cdot m}{M}} \\ &= \sum_{b=0}^{M-1} \left((1 - \frac{b}{M}) \cdot \rho_b + \frac{b}{M}\rho_{b-M}\right) \cdot e^{-\jmath 2\pi \frac{b \cdot m}{M}} \end{aligned} \tag{C.6}$$

$$(\underline{S}_x^M)_m = \sum_{a=0}^{M-1} \rho_a \cdot e^{-\jmath 2\pi \frac{a \cdot m}{M}}. \tag{C.7}$$

In the case of non-stationary processes, the theory of the previous section is not valid anymore. We assume that the time scale in which non-stationarities occur in the input signals used in this thesis, is much larger than the DFT-length $M$. By appending a time index to the power spectrum ($\underline{P}_x^M$), we then get

$$(\underline{P}_x^M[k])_m = \sum_{b=0}^{M-1} \left((1 - \frac{b}{M}) \cdot \rho_b[k] + \frac{b}{M}\rho_{b-M}[k]\right) \cdot e^{-\jmath 2\pi \frac{b \cdot m}{M}} \tag{C.8}$$

where for $0 \leq b < M$ the autocorrelation $\rho_b[k] = \mathcal{E}\{x[k] \cdot (x[k-b])^*\}$. In practice, in real-time systems, only a single realization is available, so we cannot calculate the ensemble average. When using $\underline{P}_x^M[k]$, we will

therefore have to approximate the ensemble average $\mathcal{E}$ by a time average.[1] In this thesis we will often refer to $\underline{P}_x^M[k]$ as the power vector (that is not equal to the Power Spectral Density).

## C.3 Circulant Matrices

We can also find a description of $\underline{P}_x^M[k]$ in time domain by inverse transformation of $\underline{X}^M[k] \otimes (\underline{X}^M[k])^*$, with for $0 \leq m < M$

$$(\underline{X}^M[k] \otimes (\underline{X}^M[k])^*)_m = (\underline{X}^M[k])_m \cdot ((\underline{X}^M[k])^*)_m. \tag{C.9}$$

With help of the circulant matrices in chapter 3, where it is shown that

$$\mathrm{diag}\{\underline{X}^M[k]\} = \mathcal{F}^M \check{\mathcal{X}}^M[k](\mathcal{F}^M)^{-1} \tag{C.10}$$

we obtain

$$\mathrm{diag}\{\underline{P}_x^M[k]\} = \mathcal{F}^M \mathcal{E}\{\check{\mathcal{X}}^M[k] \cdot (\check{\mathcal{X}}^M[k])^t\}\mathcal{F}^{-1} \tag{C.11}$$

This gives the opertunity to interpret $\mathcal{E}\{\check{\mathcal{X}}^M[k] \cdot (\check{\mathcal{X}}^M[k])^t\}$ as the circulant approximation of the autocorrelation matrix, and $(\mathcal{F}^M)^{-1}\underline{P}_x^M[k]$ as the circulant approximation of the autocorrelation vector.

[1] In a wide-sense staionary process, these are equal.

# Glossary

## Abbreviations

| | |
|---|---|
| A/D | Analog to Digital (converter). |
| AEC | Acoustic Echo Canceller. |
| AF | Adaptive Filter. |
| ALE | Adaptive Line Enhancer. |
| AR | Auto Regressive. |
| ARMA | Auto Regressive Moving Average. |
| BFDAF | Block Frequency Domain AF. |
| BFDC | Block Frequency Domain Convolution. |
| BLMS | Block Least Mean Square. |
| BNLMS | Block Normalised Least Mean Square. |
| BOP | Block Orthogonal Projection. |
| BTDC | Block Time Domain Convolution. |
| BRLS | Block Recursive Least Squares. |
| D/A | Digital to Analog (converter). |
| DBOP | Decoupled Block Orthogonal Projection. |
| DDE | Decision Directed Equalizer. |
| DPBFDAF | Decoupled PBFDAF. |
| BOP | Block Orthogonal Projection. |
| DFT | Discrete Fourier Transform. |
| DIF | Decimation In Frequency. |
| DIT | Decimation In Time. |
| DSP | Digital Signal Processor. |
| ERLE | Echo Return Loss Enhancement. |
| FAEST | Fast Aposteriori Error Sequential Technique. |
| FBNLMS | Frequency domain BNLMS. |
| FDAF | Frequency Domain Adaptive Filter. |
| FFT | Fast Fourier Transform. |
| FIR | Finite Impulse Response. |

| | |
|---|---|
| FNTF | Fast Newton Transversal Filter. |
| FTF | Fast Transversal Filter. |
| IDFT | Inverse Discrete Fourier Transform. |
| IFFT | Inverse Fast Fourier Transform. |
| IIR | Infinite Impulse Response. |
| LMS | Least Mean Square. |
| LPF | Low-Pass Filter. |
| LSI | Loughborough Sound Images. |
| MA | Moving Average. |
| MFLOPS | Million FLoating point Operations Per Second. |
| MSE | Mean Squared Error. |
| MSS | Multi Step Size. |
| NLMS | Normalised Least Mean Square. |
| NUPBFDAF | Non Uniform PBFDAF |
| NUPBFDC | Non Uniform PBFDC |
| OP | Orthogonal Projection. |
| PBFDAF | Partitioned BFDAF. |
| PBFDC | Partitioned BFDC. |
| PSD | Power Spectral Density |
| P/S | Parallel to Serial Converter. |
| RLS | Recursive Least Squares. |
| S/P | Serial to Parallel converter. |
| TDC | Time Domain Convolution. |

# Notation

| | |
|---|---|
| $x$ | Signals (Lower case character). |
| $A$ | Constant (Upper case character). |
| $\underline{x}$ | Time domain vector (Lower case underlined). |
| $\underline{X}$ | Frequency domain vector (Upper case underlined). |
| $\underline{x}^N$ | Vector of length N. |
| $\boldsymbol{\mathcal{X}}, \mathbf{I}$ | Matrices (Bold upper case or caligraphic). |
| $\boldsymbol{\mathcal{X}}^{B,Q}$ | $B \times Q$ matrix $\boldsymbol{\mathcal{X}}$. |
| $\mathbf{I}^M$ | For a square matrix the second dimension is omitted: $M \times M$ matrix $\mathbf{I}$. |
| $\underline{w}_i$ | A subscript $i$ denotes the $i$'th version. |
| $(\underline{w})_a$ | The $a$'th element of $\underline{w}$. |
| $(\boldsymbol{\mathcal{X}})_{a,b}$ | The $a$'th element of the $b$'th column of $\boldsymbol{\mathcal{X}}$. |
| $[k]$ | Denotes the time index. |
| $(\underline{x})^t, (\boldsymbol{\mathcal{X}})^t$ | Transpose of $\underline{x}$, $\boldsymbol{\mathcal{X}}$. |
| $(x)^*, (\underline{x})^*, (\boldsymbol{\mathcal{X}})^*$ | Complex conjugate of $x$, $\underline{x}$, $\boldsymbol{\mathcal{X}}$. |
| $(\underline{x})^h, (\boldsymbol{\mathcal{X}})^h$ | Complex conjugate transpose of $\underline{x}$, $\boldsymbol{\mathcal{X}}$. |
| $\otimes$ | Elementwise multiplication of two vectors. |
| $\mathrm{diag}\{\underline{x}^N\}$ | $N \times N$ matrix with on its main diagonal $\underline{x}^N$. |
| $\max\{B, N\}$ | Maximum of $B$ and $N$. |
| $\min\{B, N\}$ | Minimum of $B$ and $N$. |
| $\mathcal{E}\{x[k]\}$ | Ensemble average of $x[k]$. |
| $\lfloor N/Q \rfloor$ | The smallest integer, not smaller than $N/Q$. |
| $\lceil N/Q \rceil$ | The largest integer, not larger than $N/Q$. |
| $\gcd\{N, Q\}$ | Greatest common divisor of $N$ and $Q$. |
| $\Re\{X\}$ | Real part of $X$. |
| $\Im\{X\}$ | Imaginary part of $X$. |
| $\mathrm{span}\{\underline{x}_{B-1}, \ldots, \underline{x}_0\}$ | $B$-dimensional hyperplane spanned by $\underline{x}_{B-1}$ till $\underline{x}_0$. |

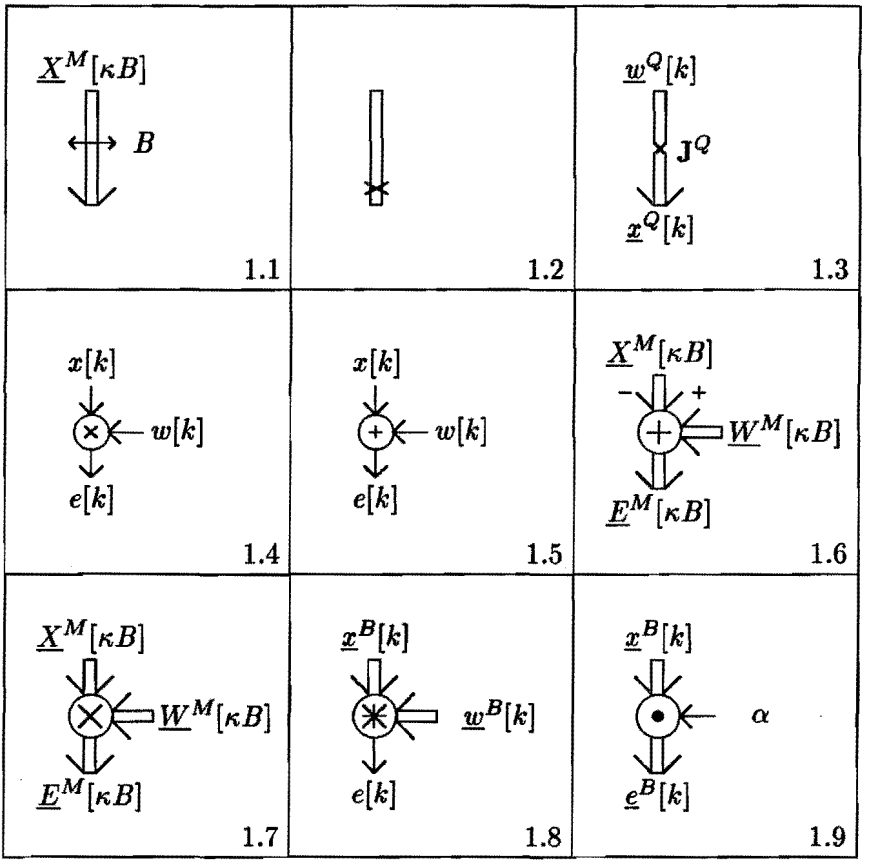
# List of Used Symbols

| | |
|---|---|
| $\underline{a}$ | Projection parameters in BOP. |
| $A$ | Update part block length. |
| $A_p$ | Normalization block length. |
| $\alpha$ | Adaptation constant. |
| $B$ | Filter part block length. |
| $B_\rho$ | Relevant input signal autocorrelation length. |
| $\beta$ | Forgetting factor in power estimation. |
| $\mathbf{C}$ | Circulant matrix. |
| $\underline{d}[k]$ | Difference of filter and echo-path impulse response. |
| $\underline{d}_{\parallel}[k]$ | Part of $\underline{d}[k]$ parallel to input signal hyperplane. |
| $\underline{d}_{\perp}[k]$ | Part of $\underline{d}[k]$ perpendicular to input signal hyperplane. |
| $D$ | Computation processing delay. |
| $D_{\max}$ | Maximum allowable processing delay. |
| $\mathbf{D}_B^M$ | Rotation matrix. |
| $\Delta$ | Algorithm processing delay. |
| $\Delta$ | Delay element in figures. |
| $e[k]$ | Desired signal (echo in AEC). |
| $\hat{e}[k]$ | Estimate of desired signal. |
| $\tilde{e}[k]$ | Desired signal corrupted with $s[k]$ . |
| $\epsilon$ | Threshold in inverse variance and power estimate. |
| $f_{\mathrm{S}}$ | Sample rate. |
| $\mathcal{F}$ | Fourier transform matrix. |
| $g_{\mathrm{F}}$ | Number of filter-part sub-filters. |
| $g_i$ | Number of sub-filters in $i$'th sub-set. |
| $g_{\mathrm{U}}$ | Number of update-part sub-filters. |
| $G$ | Number of sub-sets in NUPBFDC. |
| $\gamma$ | $A/B$, sample rate quotient in D- and NU-PBFDAF. |
| $\underline{\nabla}$ | Gradient of MSE-surface. |
| $\underline{h}[k]$ | Impulse response of echo-path. |
| $I$ | Identity matrix. |
| $\tilde{\mathcal{I}}[k]$ | Approximation error matrix. |
| $\jmath$ | Imaginary unit. |
| $J$ | Mirror matrix. |
| $k$ | Time index. |
| $\kappa$ | Block time index. |
| $l$ | Block time index. |
| $L$ | Update part FFT length. |
| $L_p$ | Normalization FFT length. |

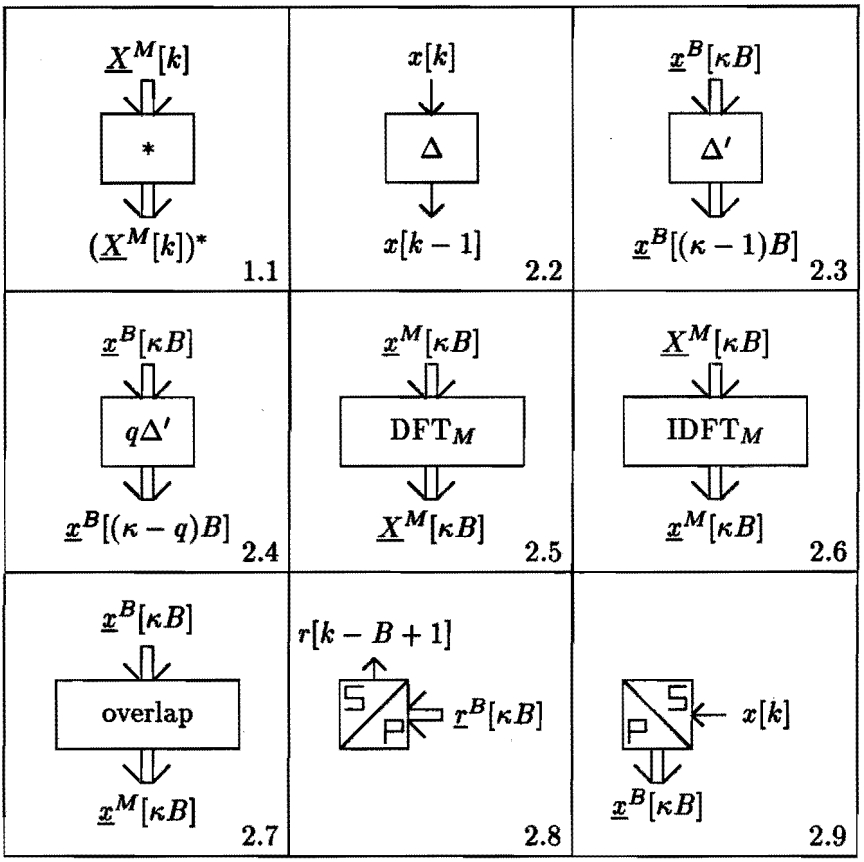
| | |
|---|---|
| $\lambda$ | Normalization part block index. |
| $M$ | Filter part FFT length. |
| $\Upsilon$ | Mean Squared Error. |
| $\Upsilon_{\min}$ | Minimum Mean Squared Error. |
| $\Theta$ | Memory occupation. |
| $N$ | Number of filter coefficients. |
| $\aleph$ | Place of Normalization. |
| $\underline{0}$,$\mathbf{0}$ | All zero vector, matrix. |
| $\Omega$ | Window positioning length. |
| $\underline{p}_{x,\tilde{e}}[k]$ | Cross-correlation vector of $x$ and $\tilde{e}$. |
| $\underline{P}_x[k]$ | Normalization (power) vector. |
| $\hat{\underline{P}}_x[k]$ | Estimate of $\underline{P}_x[k]$ |
| $\mathcal{P}[k]$ | Intermediate variable in estimation of $\hat{\underline{P}}_x^{-1}[k]$. |
| $\Psi$ | Complexity mearure (real multiplications per sample). |
| $q$ | $Q/B$, number of delays in series of delays. |
| $Q$ | Filter part partition length. |
| $r[k]$ | Residual signal. |
| $\mathcal{R}_x[k]$ | Auto-correlation matrix of $x[k]$. |
| $\check{\mathcal{R}}_x[k]$ | Circulant approximation of $\hat{\mathcal{R}}_x[k]$. |
| $\tilde{\mathcal{R}}_x[k]$ | Exact FD implementation of $\hat{\mathcal{R}}[k]$. |
| $\hat{\mathcal{R}}_x[k]$ | Approximation of autocorrelation matrix. |
| $\hat{\mathcal{R}}_{x,Q}^{N}[k]$ | DBOP decorrelation matrix. |
| $\rho_i[k]$ | Autocorrelation, $\rho_a[k] = \mathcal{E}\{x[k] \cdot (x[k-a])^*\}$. |
| $\check{\rho}_i[k]$ | $(\check{\mathcal{R}}[k])_{a-b} = \check{\rho}_{\lvert a-b \rvert}[k]$. |
| $\rho_i[k]$ | Approximation of $\rho_i[k]$ |
| $s[k]$ | Near end signal. |
| $S$ | Sum of subfilter lengths. |
| $S_{x,\omega}$ | Power Spectral Density of $x[k]$ |
| $\sigma_x^2[k]$ | Variance of $x[k]$. |
| $\varsigma[k]$ | Intermediate variable in estimation of $\sigma_x^{-2}[k]$. |
| $T$ | Sample interval. |
| $\tau$ | Extra delays in NUPBFDC. |
| $\mathbf{U}_i^{M,N}, \mathbf{U}_i^{B,N}$ | Windowing matrices. |
| $\mathbf{V}_N^M$ | Windowing matrix. |
| $\omega$ | Frequency domain variable (between $-\pi$ and $\pi$). |
| $\underline{w}[k]$ | Adaptive filter vector. |
| $\underline{\check{w}}[k]$ | AF vector extended with zeroes. |
| $x[k]$ | Input signal (Far end signal). |

| | |
|---|---|
| $X_{M,\omega}[k]$ | Discrete-time Fourier transform of $x[k]$ |
| $\mathcal{X}[k]$ | Input signal matrix. |
| $\check{\mathcal{X}}[k]$ | Circulant extension of $\mathcal{X}[k]$. |
| $\check{\mathcal{X}}_i$ | Example of $\check{\mathcal{X}}[k]$. |
| $\xi$ | Some small number, $\xi \ll 1$. |
| $\Xi$ | $\Xi + g_U$ length $L$ FFTs are needed in NUPBFDAF. |
| $\underline{y}[k]$ | Normalized residual signal vector. |
| $z$ | $Z/A$, number of delays in series of delays. |
| $Z$ | Update part partition length. |
| $Z_p$ | Normalization partition length. |

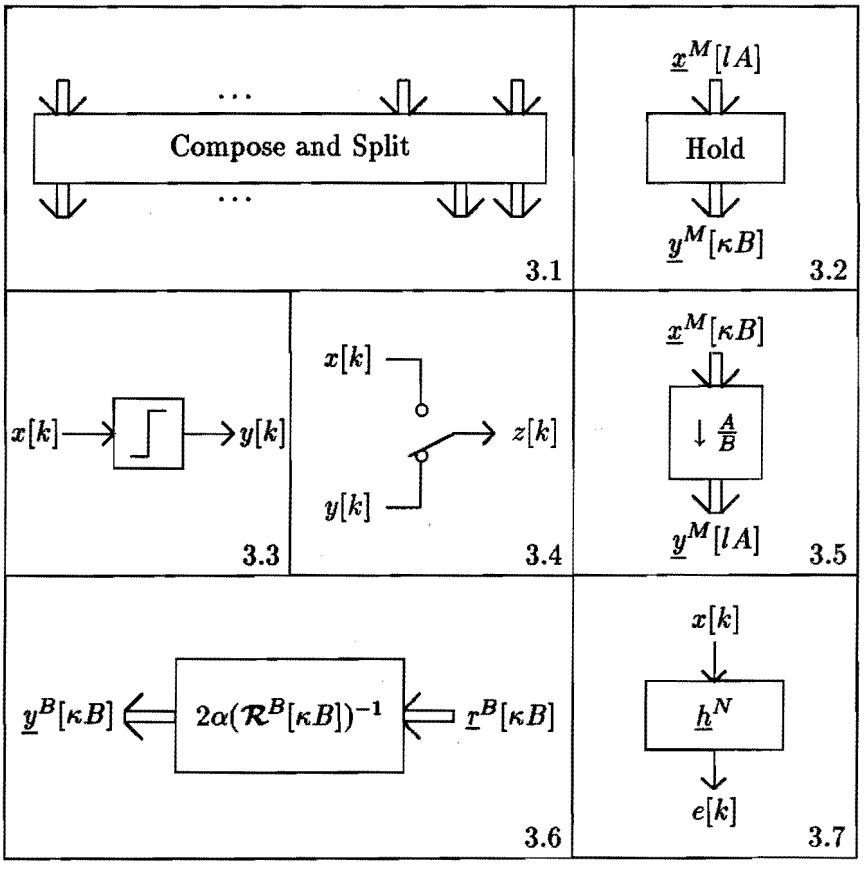
# Figures




1.1: Vector with length $B$.
1.2: Discontinued vector (discarding of samples).
1.3: Mirror operation $\underline{x}^Q[k] = \mathbf{J}^Q \cdot \underline{w}^Q[k]$.
1.4: Scalar multiplication $e[k] = x[k] \cdot w[k]$.
1.5: Scalar addition $e[k] = x[k] + w[k]$.
1.6: Elementwise addition $\underline{E}^M[\kappa B] = \underline{W}^M[\kappa B] - \underline{X}^M[\kappa B]$.
1.7: Elementwise multiplication $\underline{E}^M[\kappa B] = \underline{W}^M[\kappa B] \otimes \underline{X}^M[\kappa B]$.
1.8: Convolution $e[k] = (\underline{x}^B[k])^t \cdot \underline{w}^B[k]$.
1.9: Multiplication by scalar $\underline{e}^B[k] = \alpha \cdot \underline{x}^B[k]$.




2.1: Complex conjugation.
2.2: One sample (scalar) delay.
2.3: One sample vector delay.
2.4: Series of $q$ one sample vector delays.
2.5: DFT $\underline{X}^M[\kappa B] = \mathcal{F}^M \cdot \underline{x}^M[\kappa B]$.
2.6: IDFT $\underline{x}^M[\kappa B] = (\mathcal{F}^M)^{-1} \cdot \underline{X}^M[\kappa B]$.
2.7: Overlap $\underline{x}^M[\kappa B] = \left( \cdots (\underline{x}^B[\kappa B])^t \right)^t$.
2.8: P/S converter $\underline{r}^B[\kappa B] = \left( r[\kappa B - B + 1] \quad \cdots \quad r[\kappa B] \right)^t$.
2.9: S/P converter $\underline{x}^B[\kappa B] = \left( x[\kappa B - B + 1] \quad \cdots \quad x[\kappa B] \right)^t$.




3.1: Compose and Split operation.
3.2: Hold operation $\underline{y}^M[\kappa B] = \underline{x}^M[\lfloor \kappa B/A \rfloor A]$.
3.3: Decision element.
3.4: Switch.
3.5: Down sampler $\underline{y}^M[lA] = \underline{x}^M[(l \cdot A/B)B]$.
3.6: Matrix multplication $\underline{y}^B[\kappa B] = 2\alpha(\mathcal{R}^B[\kappa B])^{-1}\underline{r}^B[\kappa B]$.
3.7: Filter operation $e[k] = \begin{pmatrix} x[k-N+1] & \cdots & x[k] \end{pmatrix} \cdot \underline{h}^N$.

# Bibliography


[1] Aravena J.L., "Recursive moving window DFT algorithm," *IEEE Transactions on Computers*, vol. 39, no. 1, Jan. 1990, pp. 146-148.

[2] Asharif M.R. e.a., "Frequency Domain Noise Canceller: Frequency Bin Adaptive Filtering," in *Proc. ICASSP* (Tokyo, Japan), Sep. 1986, pp. 2219-2222.

[3] Bershad N. and Macchi O., "Comparison of RLS and LMS algorithms for tracking a chirped signal," in *Proc. ICASSP* (Glasgow, Scotland, UK), May 1989, pp. 896-899.

[4] Boer J.H. de, *Real-time DSP Implementation of an Acoustic Echo Canceller based on Adaptive Filtering* Eindhoven (The Netherlands): Master's thesis Eindhoven University of Technology, Faculty of Electrical Engineering, Mar. 1995.

[5] Carayannis G., Manolakis D. and Kalouptsidis N., "A fast sequential algorithm for least squares filtering and prediction," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 31, no. 6, Dec. 1983, pp. 1394-1402.

[6] Cioffi J. and Kailath T., "Fast recursive-least-squares transversal filters for adaptive filtering," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, Apr. 1984, pp. 304-337.

[7] Clark G.A., "Block Implementation of Adaptive Digital Filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 29, no. 3, June 1981, pp. 744-752.

[8] Clark G., Mitra S.K. and Parker S.R., "A unified approach to Time– and Frequency–Domain Realization of FIR Adaptive Digital Filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 31, no. 5, Oct. 1983, pp. 1073-1083.

[9] Davis P.J., *Circulant Matrices.* New York (USA): Wiley, 1979.

[10] Deisher M.E. and Spanias A.S., "Practical Considerations in the Implemenatation of a Frequency-Domain Adaptive Noise Canceller," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 41, no. 2, Feb. 1994, pp. 164-168.

[11] Demidov D.M., "An Algorithm for a Truncated Fast Fourier Transform and Its Implementation," *Computers and microprocessors in radio systems*, 1989, pp. 140-141.

[12] Egelmeers G.P.M., *Interpretations of an Orthogonal Projection (OP) algorithm for Adaptive Digital Signal Processing* Eindhoven (The Netherlands): Master's thesis Eindhoven University of Technology, Faculty of Electrical Engineering, Aug. 1991.

[13] Egelmeers G.P.M. and Sommen P.C.W., "Relation between reduced dimension time and frequency domain adaptive algorithm," in *Signal Processing VI, Proc. EUSIPCO 1992* (Brussel, Belgium), Aug. 1992, pp. 1065-1068.

[14] Egelmeers G.P.M., "Decoupling of partition factors in Partitioned Block FDAF," in *Proc. ProRisc/IEEE Workshop on Circuits, Systems and Signal Processing* (Houthalen, Belgium), Mar. 1993, pp. 203-208.

[15] Egelmeers G.P.M., "Adaptive Filtersystems: Complexity-Performance Optimalisation," *Tijdschrift NERG*, vol. 58, no. 2, 1993, pp. 59-64.

[16] Egelmeers G.P.M., "Decoupling of partition factors in Partitioned Block FDAF", in *Proc. ECCTD 1993* (Davos, Switserland), Aug. 1993, pp. 323-328.

[17] Egelmeers G.P.M. and Sommen P.C.W., "A new method for efficient convolution in frequency domain by non-uniform partitioning," in *Proc. EUSIPCO 1994* (Edinburgh, Scotland, UK), Sep. 1994, pp. 1030-1033.

[18] Egelmeers G.P.M. e.a., "Real-time implementation of low delay acoustic echo-canceller on one TMS320C30 DSP," in *Proc. ICSPAT 1994* (Dallas, Texas, USA), Oct. 1994, pp. 13-18.

[19] Egelmeers G.P.M. and Sommen P.C.W., "Acoustic echo cancellation based on Non-Uniform Partitioned BFDAF," in *Proc. ProRISC/IEEE*

*Symp. on Cicuits, Systems and Signal Processing 1995* (Mierlo, The Netherlands), Mar. 1995, pp. 67-74.

[20] Egelmeers G.P.M. and Sommen P.C.W., "Recursive computation of block FFTs," in *Proc. ProRISC/IEEE Symp. on Cicuits, Systems and Signal Processing 1995* (Mierlo, The Netherlands), Mar. 1995, pp. 59-66.

[21] Egelmeers G.P.M. and Sommen P.C.W., "A New Method for Efficient Convolution in Frequency Domain by Non-Uniform Partitioning for Adaptive Filtering," to appear in *IEEE Transactions on Signal Processing.*

[22] Eleftheriou E. and Falconer D.D., "Tracking properties and steady state performance of RLS adaptive filter algorithms," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, 1986, pp. 1097-1110.

[23] Ferrara E.R., "Fast implementation of LMS adaptive filter," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, Aug. 1980, pp. 474-475.

[24] Feuer A., "Performance Analysis of the Block Least Mean Square Algorithm," *IEEE Transactions on Circuits And Systems*, vol. 32, no. 9, Sep. 1985, pp. 960-963.

[25] Furukawa T. e.a., "The Orthogonal Projection Algorithm for Block Adaptive Signal Processing," in *Proc. ICASSP* (Glasgow, Scotland, UK), May 1989, pp. 1059-1062.

[26] Haykin S., *Adaptive Filter Theory.* Englewood Cliffs, NJ (USA): Prentice-Hall, 1986, ISBN 0-13-004052-5 025.

[27] Haykin S., *Adaptive Filter Theory.* Englewood Cliffs, NJ (USA): Prentice-Hall, 1991, second edition, ISBN 0-13-005513-1.

[28] Horna O.A., "Echo control in teleconferencing," in *Proc. Globecom* (San Diego, USA), 1983, pp. 16.2.1-16.2.7.

[29] Kurosawa K. and Furusawa T., "A Geometric Interpretation of Adaptive Algorithms," in *Proc. Globecom Conf.* (Tokyo, Japan), Nov. 1987, pp. 49.7.1-49.7.5.

[30] Lee J.C. and Un C.K., "Performance Analysis of Frequency-Domain Block LMS Adaptive Digital Filters," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 2, Feb. 1989, pp. 173-189.

[31] Mansour D. and Gray Jr. A.H., "Unconstrained Frequency Domain Adaptive Filter," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 30, no. 5, Oct. 1982, pp. 726-734.

[32] Moustakides G.V. and Theodoridis S., "Fast Newton Transversal Filters-A New Class of Adaptive Estimation Algorithms," *IEEE Transactions on Signal Processing*, vol. 39, no. 10, Oct. 1991, pp. 2184-2193.

[33] Murray W. ed., *Numerical Methods for Unconstrained Optimization.* New York (USA): Academic Press, 1972.

[34] Oppenheim A.V. and Shafer R.W., *Digital Signal Processing.* Englewood Cliffs, NJ (USA): Prentice-Hall, 1975.

[35] Ozeki K. and Umeda T., "An adaptive Filtering Algorithm Using an Orthogonal Projection to an Affine Subspace and its properties," *Electronics and Communications in Japan*, vol. 67-A, no. 5, 1984, pp. 19-27.

[36] Páez Borallo J.M. and Garcia Otero M., "On the implementation of a partitioned block frequency domain adaptive filter (PBFDAF) for long acoustic echo cancellation," *Signal Processing*, vol. 27, no. 3, June 1992, pp. 301-315.

[37] Papamichalis P.E., "FFT implementation on the TMS320C30," in *Proc. ICASSP* (New York, USA), Apr. 1988, pp. 1399-1402.

[38] Pétillon T., Gilloire A. and Theodoridis S., "Complexity Reduction in Fast RLS Transversal Adaptive Filters with Application to Acoustic Echo Cancellation," in *Proc. ICASSP 1992* (San Franscisco, CA, USA), Mar. 1992, pp. IV-37-IV-40.

[39] Sambur M.R., "Adaptive Noise Cancellation for Speech Signals," *IEEE Transactions on Acoustics, Speech and SIgnal Processing*, vol. 26, no. 5, 1978, pp. 419-423.

[40] Schroeder M.R., "Linear Prediction, entropy and signal analysis," *IEEE ASSP Magazine*, vol. 1, no. 3, 1984, pp. 3-11.

[41] Skinner D.P., "Pruning the Decimation In-Time FFT Algorithm," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 24, Apr. 1976, pp. 193-194.

[42] Slock D.T.M. and Kailath T., "Numerically stable fast transversal filters for recursive least squares adaptive filtering," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 39, no. 1, Jan. 1991, pp. 92-114.

[43] Sommen P.C.W., "Partitioned Frequency Domain Adaptive Filters," in *Proc. Asilomar conf. on Signals and Systems* (Pacific Grove, CA, USA), 1989, pp. 676-681.

[44] Sommen P.C.W., "On the Orthogonal Projection Method for Adaptive Filters," in *Proc. Intern. Symp. Mathematical Theory of Networks and Systems*, vol. III, 1989, pp. 283-290.

[45] Sommen P.C.W. and Valburg C.J. van, "Efficient Realization of Adaptive Filter using an Orthogonal Projection Method," in *Proc. ICASSP* (Glasgow, Scotland, UK), May 1989.

[46] Sommen P.C.W., *Adaptive Filtering Methods.* Eindhoven (The Netherlands): Thesis Eindhoven University of Technology, June 1992, ISBN 90-9005143-0.

[47] Sondhi M.M. and Berkley D.A., "Silencing echoes in the telephone network," in Proc. IEEE, vol. 68, pp. 948-963.

[48] Soo J.S. and Pang K.K., "A new structure for Block FIR Adaptive Digital Filters," in *Proc. IEEE Conf.* (Sydney, Australia), Sep. 1987, pp. 364-367.

[49] Soo J.S. and Phang K.K., "A multistep size (MSS) Frequency Domain Adaptive Filter," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 39, no. 1, Jan. 1991, pp. 115-121.

[50] Sorensen H.V., Jones D.L., Heidemann M.T. and Burrus C.S., "Real-valued Fast Fourier Transform Algorithms," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, 1987, pp. 849-863.

[51] Springer T., "Sliding FFT computes frequency spectra in real time," *EDN*, Sep. 1988, pp. 161-170.

[52] Sreenivas T.V. and Rao P.V.S., "FFT Algorithm for Both Input and Output Pruning," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 27, no. 3, June 1979, pp. 191-292.

[53] Vogel P., *Application of Wave Filed Synthesis in Room Acoustics.* Delft (The Netherlands): Thesis Delft University of Technology, 1993.

[54] Widrow B. and Hoff Jr. M.E., "Adaptive switching circuits," in *IRE WESCON Conv.Rec.*, part 4, 1960, pp. 96-104.

[55] Widrow B., "Adaptive Filters," in Kalman R.E. and DeClaris N. eds., *Aspects of Network and System Theory.* New York (USA): Holt, Rinehart and Winston, 1970.

[56] Widrow B. e.a., "Adaptive noise cancelling: principles and applications," in *Proc. IEEE*, vol. 63, pp. 1692-1716.

[57] Widrow B. and Stearns S.D., *Adaptive Signal Processing.* Englewood Cliffs, NJ (USA): Prentice-Hall, 1985, ISBN 0 013 004029 01

[58] Wiener N. and Hopf E., "On a class of singular integral equations," in *Proc. Prussian Acad. Math-Phys. Ser.*, 1931, p. 696.

# Acknowledgement

I would like to thank all people that contributed in some way to this thesis. First, I am grateful that my first promotor Wim van Bokhoven gave me the opportunity to carry out this research in his group and to write this thesis. I thank Piet Sommen, my coach and copromotor, for the stimulating discussions and the useful critisism. I would like to thank my second promotor, prof. P. van den Bosch, prof. O. Herrmann, prof. E. Deprettere and Daniel Schobben for their useful comments on the first concept of this thesis. Further I thank Jacob de Boer and Mugur Alexiu for programming the DSP for the real-time realization of the Acoustic Echo Canceller, Heini Withagen for being a pleasant roommate and all other collegues in the group EEB for creating a nice and stimulating research environment.

I am also grateful towards my parents who always stimulated me to do more than just enough and gave me the opportunity to get my masters degree. Last I would like to thank William for reviewing the statements to this thesis and Gabi, who besides her indirect support at home also made her direct contribution to this thesis by designing the cover, showing her view of adaptive filters.

# Curriculum Vitae

Gerardus Paul Maria Egelmeers (Gerard) was born on May 1, 1969 in Veldhoven, the Netherlands.

He graduated at the Anton van Duinkerken College in Veldhoven on June 1987. He received the Ingenieur (Ir.) degree in electrical engineering with honours from the Eindhoven University of Technolgy on August 1991. His master report was titled "Interpretations of an Orthogonal Projection (OP) algorithm for Adaptive Digital Signal Processing".

From September 1, 1991 till September 1, 1995 he has been working as a researcher (AIO-4) in the area of (adaptive) Digital Signal Processing (DSP) at the Electronic Circuit Design group (EEB), Faculty of Electrical Engineering (E) at the Eindhoven University of Technolgy (TUE).

# STELLINGEN

Behorende bij het proefschrift

## Real Time Realization Concepts of Large Adaptive Filters

*door G.P.M. Egelmeers*

1. Door het ontkoppelen van parameters in algoritmen voor adaptieve filters kunnen belangrijke eigenschappen zoals berekeningsvertraging, berekeningscomplexiteit en convergentiegedrag veel beter op de bedoelde toepassing worden afgestemd.
   (Bron: [1])

2. De berekeningscomplexiteit van convoluties kan sterk gereduceerd worden door gebruik te maken van niet uniforme partitie technieken in het frequentie domein, zonder dat dit leidt tot een (grote) berekeningsvertraging.
   (Bron: [1])

3. De concessies die aan gestelde randvoorwaarden gedaan worden om de berekeningscomplexiteit van een akoestische echo compensator zodanig laag te krijgen dat deze op één signaalprocessor kan worden gerealiseerd, zijn als gevolg van de in dit proefschrift geïntroduceerde technieken niet langer nodig.
   (Bron: [2, 1])

4. De aanpassing van de coefficienten in een transversaal filter zoals bijvoorbeeld toegepast in akoestische echo compensatoren, levert als gevolg van de lengte van dit transversaal filter een dusdanig grote hoeveelheid ruis op, dat moet worden gezocht naar een andere manier om de coefficienten aan te passen.

5. De uiterlijke eenvoud van de formules die het Least Mean Square (LMS) algoritme beschrijven houdt niet in dat zijn berekenings complexiteit laag is, of dat het gedrag van dit algoritme op eenvoudige wijze te analyseren is.
   (Bron: [3, 4])

6. De berekeningscomplexiteit van gedeeltelijk overlappende FFTs (snelle Fourier transformaties) kan worden gereduceerd door gebruik te maken van deze overlap.
(Bron: [1], appendix B)

7. Het feit dat de besturen van studentenverenigingen vaak voor een deel bestaan uit mensen die bestuurslid geworden zijn om hun Curriculum Vitae een mooier aanzien te geven, bevordert niet bepaald de kwaliteit van deze besturen.

8. De diversiteit in te varen nummers bij het onderdeel roeien op de olympische spelen en de wereldkampioenschappen dient (nog verder) beperkt te worden (tot b.v. 1x, 2-, 4x en 8+). Dit bevordert de overzichtelijkheid en aantrekkelijkheid voor niet roeiers en voorkomt dat roeiers elkaars concurrentie gaan ontlopen door in andere nummers uit te komen.

9. Een roeier vergroot zijn kans om voor de nationale selectie te worden uitgenodigd door zich in de omgeving van Amsterdam te vestigen.

10. De eerste twee jaren van alle universitaire studies dienen te worden samengevoegd tot een beperkt aantal verschillende twee jarige "onderbouw" studies.

11. De gewichtslimiet die verbonden is aan het uitkomen in de lichte categorie bij roeien nodigt zwaardere mensen uit tot het nemen van extreme maatregelen om gewicht te verliezen, wat in absolute zin tot prestatieverlies leidt en vaak ook nog de gezondheid bedreigt. Dit is in tegenspraak met de grondbeginselen van sport.

12. Het begeleiden van vrouwelijke sporters vereist een andere aanpak dan de begeleiding van mannelijke sporters. Sociale aspecten spelen bij de eerstgenoemden vaak een veel grotere rol dan bij de mannen, waar prestatiedrang meestal de overhand heeft.

13. De aanleg van wegen en kabelbanen in het hooggebergte bedreigt behalve het milieu ook het plezier van de bergwandelaar.

14. Het uitvaardigen van een rookverbod in openbare ruimtes (met name de liften en toiletten) blijkt vrijwel geen effect te hebben als daar geen sanctie tegenover staat. De (gevulde) asbakken en het permanente rookgordijn in diverse ruimtes in de TUE (b.v. hal E-Hoog) doen niet vermoeden dat het rookverbod zoals dat staat aangegeven bij de hoofdingang serieus genomen wordt.

15. Het vervullen van de dienstplicht in de laatste lichting heeft als voordeel dat de betrokkenen zichzelf waarschijnlijk niet zullen vervelen. Het opruimen van de te sluiten kazernes zal voldoende bezigheden verschaffen.


[1] Egelmeers G.P.M., *Real Time Realization Concepts of Large Adaptive Filters.* Eindhoven (Netherland): Proefschrift Technische Universiteit Eindhoven, Nov. 1995, ISBN 90-386-0456-4.

[2] Páez Borallo J.M. and Garcia Otero M., "On the implementation of a partitioned block frequency domain adaptive filter (PBFDAF) for long acoustic echo cancellation," *Signal Processing*, vol. 27, no. 3, Juni 1992, pp. 301-315.

[3] Haykin S., *Adaptive Filter Theory.* Englewood Cliffs, NJ (USA): Prentice-Hall, 1986, ISBN 0-13-004052-5 025.

[4] Widrow B. and Hoff Jr. M.E., "Adaptive switching circuits," in *IRE WESCON Conv.Rec.*, part 4, 1960, pp. 96-104.