

With CD-ROM



SUBBAND ADAPTIVE FILTERING

THEORY AND IMPLEMENTATION



KONG-AIK LEE | WOON-SENG GAN | SEN M. KUO

WILEY

Subband Adaptive Filtering

Subband Adaptive Filtering

Theory and Implementation

Kong-Aik Lee

Institute for Infocomm Research, Singapore

Woon-Seng Gan

Nanyang Technological University, Singapore

Sen M. Kuo

Northern Illinois University, USA



A John Wiley and Sons, Ltd., Publication

Disclaimer: This eBook does not include ancillary media that was packaged with the printed version of the book.

This edition first published 2009
© 2009, John Wiley & Sons, Ltd.

Registered office

John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

MATLAB® MATLAB and any associated trademarks used in this book are the registered trademarks of The MathWorks, Inc.

For MATLAB® product information, please contact:

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7001
E-mail: info@mathworks.com
Web: www.mathworks.com

LabVIEW™ and National Instruments™ are trademarks and trade names of National Instruments Corporation.

Interactive LabVIEW-based applications are available to support selected concepts and exercises from the textbook. The software is available on the companion website for this book: www3.ntu.edu.sg/home/ewsgan/saf_book.html or directly from the National Instruments website (go to <http://www.ni.com/info> and enter the code *safext*).

Library of Congress Cataloging-in-Publication Data

Lee, Kong-Aik.

Subband adaptive filtering : theory and implementation / by Kong-Aik Lee,
Woon-Seng Gan, Sen M. Kuo.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-470-51694-2 (cloth)

1. Adaptive filters. 2. Adaptive signal processing. 3. Radio frequency modulation, Narrow-band. I. Gan, Woon-Seng. II. Kuo, Sen M. (Sen-Maw) III. Title.

TK7872.F5L435 2009
621.3815'324--dc22

2009015205

A catalogue record for this book is available from the British Library.
ISBN: 978-0-470-51694-2

Typeset in 10/12pt Times by Laserwords Private Limited, Chennai, India.
Printed and bound in Great Britain by CPI Antony Rowe, Chippenham, Wiltshire

Contents

About the authors	xi
Preface	xiii
Acknowledgments	xv
List of symbols	xvii
List of abbreviations	xix
1 Introduction to adaptive filters	1
1.1 Adaptive filtering	1
1.2 Adaptive transversal filters	2
1.3 Performance surfaces	4
1.4 Adaptive algorithms	6
1.5 Spectral dynamic range and misadjustment	13
1.6 Applications of adaptive filters	15
1.6.1 Adaptive system identification	15
1.6.2 Adaptive prediction	23
1.6.3 Adaptive inverse modeling	25
1.6.4 Adaptive array processing	28
1.6.5 Summary of adaptive filtering applications	31
1.7 Transform-domain and subband adaptive filters	31
1.7.1 Transform-domain adaptive filters	31
1.7.2 Subband adaptive filters	38
1.8 Summary	39
References	39
2 Subband decomposition and multirate systems	41
2.1 Multirate systems	41
2.2 Filter banks	44
2.2.1 Input–output relation	46

2.2.2	Perfect reconstruction filter banks	47
2.2.3	Polyphase representation	48
2.3	Paraunitary filter banks	54
2.4	Block transforms	55
2.4.1	Filter bank as a block transform	55
2.5	Cosine-modulated filter banks	59
2.5.1	Design example	63
2.6	DFT filter banks	65
2.6.1	Design example	66
2.7	A note on cosine modulation	67
2.8	Summary	68
	References	69
3	Second-order characterization of multirate filter banks	73
3.1	Correlation-domain formulation	73
3.1.1	Critical decimation	77
3.2	Cross spectrum	79
3.2.1	Subband spectrum	82
3.3	Orthogonality at zero lag	85
3.3.1	Paraunitary condition	86
3.4	Case study: Subband orthogonality of cosine-modulated filter banks	89
3.4.1	Correlation-domain analysis	89
3.4.2	MATLAB simulations	92
3.5	Summary	96
	References	97
4	Subband adaptive filters	99
4.1	Subband adaptive filtering	99
4.1.1	Computational reduction	100
4.1.2	Spectral dynamic range	101
4.2	Subband adaptive filter structures	104
4.2.1	Open-loop structures	104
4.2.2	Closed-loop structures	104
4.3	Aliasing, band-edge effects and solutions	106
4.3.1	Aliasing and band-edge effects	107
4.3.2	Adaptive cross filters	108
4.3.3	Multiband-structured SAF	110
4.3.4	Closed-loop delayless structures	113
4.4	Delayless subband adaptive filters	114
4.4.1	Closed-loop configuration	114
4.4.2	Open-loop configuration	115
4.4.3	Weight transformation	116
4.4.4	Computational requirements	123
4.5	MATLAB examples	124
4.5.1	Aliasing and band-edge effects	125

4.5.2	Delayless alias-free SAFs	126
4.6	Summary	128
	References	129
5	Critically sampled and oversampled subband structures	133
5.1	Variants of critically sampled subband adaptive filters	133
5.1.1	SAF with the affine projection algorithm	134
5.1.2	SAF with variable step sizes	136
5.1.3	SAF with selective coefficient update	137
5.2	Oversampled and nonuniform subband adaptive filters	138
5.2.1	Oversampled subband adaptive filtering	138
5.2.2	Nonuniform subband adaptive filtering	140
5.3	Filter bank design	141
5.3.1	Generalized DFT filter banks	141
5.3.2	Single-sideband modulation filter banks	142
5.3.3	Filter design criteria for DFT filter banks	144
5.3.4	Quadrature mirror filter banks	149
5.3.5	Pseudo-quadrature mirror filter banks	153
5.3.6	Conjugate quadrature filter banks	155
5.4	Case study: Proportionate subband adaptive filtering	156
5.4.1	Multiband structure with proportionate adaptation	156
5.4.2	MATLAB simulations	157
5.5	Summary	161
	References	163
6	Multiband-structured subband adaptive filters	167
6.1	Multiband structure	167
6.1.1	Polyphase implementation	170
6.2	Multiband adaptation	173
6.2.1	Principle of minimal disturbance	173
6.2.2	Constrained subband updates	173
6.2.3	Computational complexity	175
6.3	Underdetermined least-squares solutions	177
6.3.1	NLMS equivalent	178
6.3.2	Projection interpretation	179
6.4	Stochastic interpretations	179
6.4.1	Stochastic approximation to Newton's method	179
6.4.2	Weighted MSE criterion	181
6.4.3	Decorrelating properties	186
6.5	Filter bank design issues	187
6.5.1	The diagonal assumption	187
6.5.2	Power complementary filter bank	187
6.5.3	The number of subbands	188
6.6	Delayless MSAF	189
6.6.1	Open-loop configuration	189

6.6.2	Closed-loop configuration	191
6.7	MATLAB examples	192
6.7.1	Convergence of the MSAF algorithm	193
6.7.2	Subband and time-domain constraints	195
6.8	Summary	198
	References	199
7	Stability and performance analysis	203
7.1	Algorithm, data model and assumptions	203
7.1.1	The MSAF algorithm	203
7.1.2	Linear data model	204
7.1.3	Paraunitary filter banks	206
7.2	Multiband MSE function	209
7.2.1	MSE functions	209
7.2.2	Excess MSE	210
7.3	Mean analysis	211
7.3.1	Projection interpretation	211
7.3.2	Mean behavior	213
7.4	Mean-square analysis	214
7.4.1	Energy conservation relation	214
7.4.2	Variance relation	216
7.4.3	Stability of the MSAF algorithm	216
7.4.4	Steady-state excess MSE	217
7.5	MATLAB examples	219
7.5.1	Mean of the projection matrix	219
7.5.2	Stability bounds	220
7.5.3	Steady-state excess MSE	222
7.6	Summary	223
	References	224
8	New research directions	227
8.1	Recent research on filter bank design	227
8.2	New SAF structures and algorithms	228
8.2.1	In-band aliasing cancellation	228
8.2.2	Adaptive algorithms for the SAF	230
8.2.3	Variable tap lengths for the SAF	230
8.3	Theoretical analysis	232
8.4	Applications of the SAF	232
8.5	Further research on a multiband-structured SAF	233
8.6	Concluding remarks	234
	References	235
Appendix A	Programming in MATLAB	241
A.1	MATLAB fundamentals	241
A.1.1	Starting MATLAB	241
A.1.2	Constructing and manipulating matrices	244
A.1.3	The colon operator	244

A.1.4	Data types	248
A.1.5	Working with strings	248
A.1.6	Cell arrays and structures	249
A.1.7	MATLAB scripting with M-files	251
A.1.8	Plotting in MATLAB	252
A.1.9	Other useful commands and tips	255
A.2	Signal processing toolbox	258
A.2.1	Quick fact about the signal processing toolbox	258
A.2.2	Signal processing tool	262
A.2.3	Window design and analysis tool	267
A.3	Filter design toolbox	268
A.3.1	Quick fact about the filter design toolbox	268
A.3.2	Filter design and analysis tool	269
A.3.3	MATLAB functions for adaptive filtering	270
A.3.4	A case study: adaptive noise cancellation	272
Appendix B	Using MATLAB for adaptive filtering and subband adaptive filtering	279
B.1	Digital signal processing	279
B.1.1	Discrete-time signals and systems	279
B.1.2	Signal representations in MATLAB	280
B.2	Filtering and adaptive filtering in MATLAB	282
B.2.1	FIR filtering	282
B.2.2	The LMS adaptive algorithm	284
B.2.3	Anatomy of the LMS code in MATLAB	285
B.3	Multirate and subband adaptive filtering	292
B.3.1	Implementation of multirate filter banks	292
B.3.2	Implementation of a subband adaptive filter	297
Appendix C	Summary of MATLAB scripts, functions, examples and demos	301
Appendix D	Complexity analysis of adaptive algorithms	307
Index		317

About the authors

Kong-Aik Lee received his B.Eng (1st Class Hons) degree from Universiti Teknologi Malaysia in 1999, and his Ph.D. degree from Nanyang Technological University, Singapore, in 2006. He is currently a Research Fellow with the Institute for Infocomm Research (I²R), Agency for Science, Technology and Research (A*STAR), Singapore. He has been actively involved in the research on subband adaptive filtering techniques for the past few years. He invented the Multiband-structured Subband Adaptive Filter (MSAF), a very fast converging and computationally efficient subband adaptive filtering algorithm. His current research has primarily focused on improved classifier design for speaker and language recognition.

Woon-Seng Gan received his B.Eng (1st Class Hons) and PhD degrees, both in Electrical and Electronic Engineering from the University of Strathclyde, UK in 1989 and 1993 respectively. He joined the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, as a Lecturer and Senior Lecturer in 1993 and 1998 respectively. In 1999, he was promoted to Associate Professor. He is currently the Deputy Director of the Center for Signal Processing at Nanyang Technological University. His research interests include adaptive signal processing, psycho-acoustical signal processing, audio processing, and real-time embedded systems.

He has published more than 170 international refereed journals and conference papers, and has been awarded four Singapore and US patents. He has previously co-authored two technical books on *Digital Signal Processors: Architectures, Implementations, and Applications* (Prentice Hall, 2005) and *Embedded Signal Processing with the Micro Signal Architecture* (Wiley-IEEE, 2007).

Dr. Gan has also won the Institute of Engineers Singapore (IES) Prestigious Engineering Achievement Award in 2001 for his work on Audio Beam System. He is currently serving as an Associate Editor for the EURASIP Journal on Audio, Speech and Music Processing, and EURASIP Research Letters in Signal Processing. He is also a Senior Member of IEEE and serves as a committee member in the IEEE Signal Processing Society Education Technical Committee.

Sen M. Kuo received the B.S. degree from National Taiwan Normal University, Taipei, Taiwan, in 1976 and the M.S. and Ph.D. degrees from the University of New Mexico, Albuquerque, NM in 1983 and 1985, respectively.

He is a Professor and served as the department chair from 2002 to 2008 in the Department of Electrical Engineering, Northern Illinois University, DeKalb, IL. He was with Texas Instruments, Houston, TX in 1993, and with Chung-Ang University, Seoul, Korea in 2008. He is the leading author of four books: *Active Noise Control Systems* (Wiley, 1996), *Real-Time Digital Signal Processing* (Wiley, 2001, 2006), and *Digital Signal Processors* (Prentice Hall, 2005), and a co-author of *Embedded Signal Processing with the Micro Signal Architecture* (Wiley 2007). He holds seven US patents, and has published over 200 technical papers. His research focuses on active noise and vibration control, real-time DSP applications, adaptive echo and noise cancellation, digital audio and communication applications, and biomedical signal processing.

Prof. Kuo received the IEEE first-place transactions (Consumer Electronics) paper award in 1993, and the faculty-of-year award in 2001 for accomplishments in research and scholarly areas. He served as an associate editor for IEEE Transactions on Audio, Speech and Language Processing, and serves as a member of the editorial boards for EURASIP Research Letters in Signal Processing and Journal of Electrical and Computer Engineering.

Preface

Subband adaptive filtering is rapidly becoming one of the most effective techniques for reducing computational complexity and improving the convergence rate of algorithms in adaptive signal processing applications. Additional features of subband adaptive filters also make this technique suitable for many real-life applications.

This book covers the fundamental theory and analysis of commonly used subband adaptive filter structures and algorithms with concise derivations. The book is further enhanced with ready-to-run software written in MATLAB for researchers, engineers and students. These MATLAB codes are introduced in many chapters in order to clarify important concepts of subband adaptive algorithms, create figures and tables that will promote the understanding of theory, implement some important applications and provide a further extension of research works in this area. These programs along with associated data files and useful information are included in the companion CD.

This book serves as a state-of-the-art and comprehensive text/reference for researchers, practicing engineers and students who are developing and researching in advanced signal processing fields involving subband adaptive filters. The book consists of eight chapters, outlined as follows:

- **Chapter 1** covers fundamentals of adaptive filtering and introduces four basic applications. The general problems and difficulties encountered in adaptive filtering are presented and the motivations for using subband adaptive filters are explained.
- **Chapter 2** reviews basic concepts, design techniques and various formulations of multirate filter banks. In particular, we introduce a special class of paraunitary filter banks that can perfectly reconstruct the fullband signal after subband decomposition. The underlying affinity between filter bank and block transform is also addressed. Finally, techniques for designing cosine-modulated and discrete Fourier transform (DFT) filter banks that will be used extensively in the book are discussed.
- **Chapter 3** introduces important concepts of subband orthogonality and analyzes the correlation between subband signals. The second-order characteristics of multirate filter banks are discussed using a correlation-domain formulation. With this analysis technique, the effect of filtering random signals using a filter bank can be conveniently described in terms of the system effect on the autocorrelation function of the input signal.

- **Chapter 4** introduces fundamental principles of using subband adaptive filtering. The major problems and some recent improvements are also addressed. The delayless SAF structures, which eliminate the signal path delay introduced by the filter banks, are also presented in this chapter. Different delayless SAF structures and the corresponding weight transformation techniques are discussed.
- **Chapter 5** analyzes and compares various subband adaptive filter structures and techniques used to update tap weights in individual subbands. The main features of critically sampled and oversampled subband adaptive filter structures are also discussed. Each structure has its own filter bank design issues to be considered, which will be demonstrated using MATLAB examples.
- **Chapter 6** presents a class of multiband-structured subband adaptive filters that uses a set of normalized subband signals in order to adapt the fullband tap weights of a single adaptive filter. The fundamental idea and convergence behavior of the multiband weight-control mechanism is described. Filter bank design issues and delayless implementation for the multiband-structured subband adaptive filters are also discussed in this chapter.
- **Chapter 7** focuses on the stability and performance analysis of subband adaptive filters. Convergence analysis of subband adaptive filters in the mean and mean-square senses are presented. These derivations resulted in the stability bounds for step size and the steady-state MSE. The theoretical analysis provides a set of working rules for designing SAFs in practical applications.
- **Chapter 8** is the last chapter and introduces recent research works in the areas of subband adaptive filtering. In particular, new subband adaptive filter structures and algorithms, and the latest development in theoretical analysis and applications of subband adaptive filtering are discussed. This chapter also summarizes some research directions in multiband-structured subband adaptive filters.

As in many other technical reference books, we have attempted to cover all state-of-the-art techniques in subband adaptive filtering. However, there will still be some inevitable omissions due to the page limitation and publication deadline. A companion website (www3.ntu.edu.sg/home/ewsgan/saf_book.html) for this book has been created and will be continuously updated in order to provide references to the most recent progress in the area of subband adaptive filtering. Promising research topics and new applications will also be posted on this website so that researchers and students can become aware of challenging areas. We hope that this book will serve as a useful guide for what has already been done and as an inspiration for what will follow in the emerging area of subband adaptive filtering.

Acknowledgments

We are very grateful to many individuals for their assistance with writing this book. In particular, we would like to thank Kevin Kuo for proofreading the book, Dennis R. Morgan and Stephen Weiss for their patience in reviewing two early chapters of the book, as well as their helpful suggestions that have guided the presentation layout of the book. We would also like to thank Paulo S. R. Diniz and Behrouz Farhang-Boroujeny for valuable comments on an earlier version of the book.

Several individuals at John Wiley & Sons, Ltd provided great help in making this book a reality. We wish to thank Simone Taylor (Editor for Engineering Technology), Georgia Pinteau (Commissioning Editor), Liz Benson (Content Editor), Emily Bone (Associate Commissioning Editor) and Nicky Skinner (Project Editor) for their help in seeing this book through to the final stage.

This book is dedicated to many of our past and present students who have taken our digital signal processing and adaptive signal processing courses and have written MS theses and PhD dissertations and completed projects under our guidance at both Nanyang Technological University (NTU), Singapore, and Northern Illinois University (NIU), USA. Both institutions have provided us with a stimulating environment for research and teaching, and we appreciate the strong encouragement and support we have received. We would also like to mention the Institute for Infocomm Research (I²R), Singapore, for their support and encouragement in writing this book. Finally, we are greatly indebted to our parents and families for their understanding, patience and encouragement throughout this period.

Kong-Aik Lee, Woon-Seng Gan and Sen M. Kuo
November 2008

List of symbols

\mathbf{A}^{-1}	Inverse of matrix \mathbf{A}
\mathbf{A}^T	Transposition of matrix \mathbf{A}
\mathbf{A}^H	Hermitian transposition of matrix \mathbf{A}
$\kappa(\mathbf{A})$	Condition number of matrix \mathbf{A}
$\mathbf{I}_{N \times N}$	$N \times N$ identity matrix
$\Lambda = \text{diag}(\cdot)$	Diagonal matrix
$\tilde{\mathbf{E}}(z)$	Paraconjugate of $\mathbf{E}(z)$
$\ \cdot\ $	Euclidean norm
$ \cdot $	Absolute value
$\lfloor \cdot \rfloor$	Integer part of its argument
$(\cdot)^*$	Complex conjugation
$E\{\cdot\}$	Statistical expectation
$*$	Convolution
$\delta(n)$	Unit sample sequence
$O(\cdot)$	Order of complexity
\approx	Approximately equal to
\forall	For all
∞	Infinity
∇	Gradient
\equiv	Defined as
j	$\sqrt{-1}$
$\downarrow D$	D -fold decimation
$\uparrow I$	I -fold interpolation
$W_D \equiv \sqrt[D]{e^{-j2\pi}}$	D th root of unity
z^{-1}	Unit delay
$z^{-\Delta}$	Δ samples delay
$\gamma_{uu}(l)$	Autocorrelation function of $u(n)$
$\Gamma_{uu}(e^{j\omega})$	Power spectrum of $u(n)$
N	Number of subbands
M	Length of fullband adaptive filter
M_S	Length of subband adaptive filter
L	Length of analysis and synthesis filters

P	Project order of affine projection algorithm
$p(n), P(z)$	Prototype filter
$h_i(n), H_i(z)$	The i th analysis filter
$f_i(n), F_i(z)$	The i th synthesis filter
\mathbf{H}	Analysis filter bank ($L \times N$ matrix)
\mathbf{F}	Synthesis filter bank ($L \times N$ matrix)
$x_i(n), X_i(z)$	The i th subband signal
$x_{i,D}(k), X_{i,D}(z)$	The i th subband signal after decimation
$\gamma_{ip}(l)$	Cross-correlation function between the i th and p th subbands
$\Gamma_{ip}(e^{j\omega})$	Cross-power spectrum between the i th and p th subbands
$J = E\{e^2(n)\}$	MSE function
J_M	Multiband MSE function
J_{LS}	Weighted least-square error
J_{SB}	Subband MSE function
$J(\infty)$	Steady-state MSE
J_{ex}	Steady-state excess MSE
$u(n)$	Input signal
$d(n)$	Desired response
$e(n)$	Error signal
μ	Step-size parameter
α	Regularization parameter
\mathbf{R}	Input autocorrelation matrix
\mathbf{p}	Cross-correlation vector
\mathbf{w}_o	Optimum weight vector
$\mathbf{b}, B(z)$	Unknown system
$\mathbf{w}(n), W(z)$	Adaptive weight vector (fullband)
$\mathbf{w}_i(k), W_i(z)$	Adaptive filter in the i th subband

List of abbreviations

ANC	Active noise control
AEC	Acoustic echo cancellation
AP	Affine projection
AR	Autoregressive
DCT	Discrete cosine transform
DFT	Discrete Fourier transform
e.g.	exempli gratia: for example
FFT	Fast Fourier transform
FIR	Finite impulse response
IDFT	Inverse DFT
i.e.	id est: that is
IFFT	Inverse FFT
IIR	Infinite impulse response
LMS	Least-mean-square
MSAF	Multiband-structured subband adaptive filter
MSE	Mean-square error
NLMS	Normalized LMS
PRA	Partial rank algorithm
PQMF	Pseudo-QMF
QMF	Quadrature mirror filter
SAF	Subband adaptive filter
RLS	Recursive least-squares

1

Introduction to adaptive filters

This chapter introduces the fundamental principles of adaptive filtering and commonly used adaptive filter structures and algorithms. Basic concepts of applying adaptive filters in practical applications are also highlighted. The problems and difficulties encountered in time-domain adaptive filters are addressed. Subband adaptive filtering is introduced to solve these problems. Some adaptive filters are implemented using MATLAB for different applications. These ready-to-run programs can be modified to speed up research and development in adaptive signal processing.

1.1 Adaptive filtering

A filter is designed and used to extract or enhance the desired information contained in a signal. An adaptive filter is a filter with an associated adaptive algorithm for updating filter coefficients so that the filter can be operated in an unknown and changing environment. The adaptive algorithm determines filter characteristics by adjusting filter coefficients (or tap weights) according to the signal conditions and performance criteria (or quality assessment). A typical performance criterion is based on an error signal, which is the difference between the filter output signal and a given reference (or desired) signal.

As shown in Figure 1.1, an adaptive filter is a digital filter with coefficients that are determined and updated by an adaptive algorithm. Therefore, the adaptive algorithm behaves like a human operator that has the ability to adapt in a changing environment. For example, a human operator can avoid a collision by examining the visual information (input signal) based on his/her past experience (desired or reference signal) and by using visual guidance (performance feedback signal) to direct the vehicle to a safe position (output signal).

Adaptive filtering finds practical applications in many diverse fields such as communications, radar, sonar, control, navigation, seismology, biomedical engineering and even in financial engineering [1–7]. In Section 1.6, we will introduce some typical applications using adaptive filtering.

This chapter also briefly introduces subband adaptive filters that are more computationally efficient for applications that need a longer filter length, and are more effective if

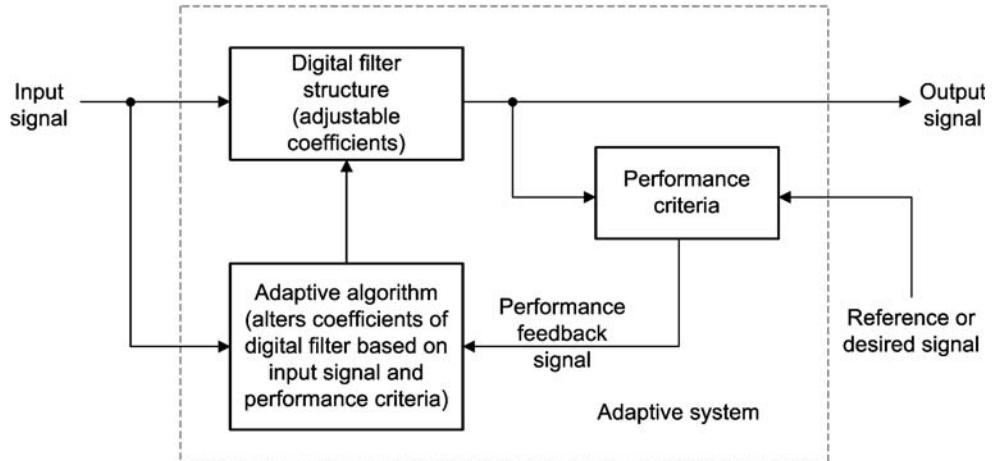


Figure 1.1 Basic functional blocks of an adaptive filter

the input signal is highly correlated. For example, high-order adaptive filters are required to cancel the acoustic echo in hands-free telephones [2]. The high-order filter together with a highly correlated input signal degrades the performances of most time-domain adaptive filters. Adaptive algorithms that are effective in dealing with ill-conditioning problems are available; however, such algorithms are usually computationally demanding, thereby limiting their use in many real-world applications.

In the following sections we introduce fundamental adaptive filtering concepts with an emphasis on widely used adaptive filter structures and algorithms. Several important properties on convergence rate and characteristics of input signals are also summarized. These problems motivated the development of subband adaptive filtering techniques in the forthcoming chapters. In-depth discussion on general adaptive signal processing can be found in many reference textbooks [1–3, 5–9].

1.2 Adaptive transversal filters

An adaptive filter is a self-designing and time-varying system that uses a recursive algorithm continuously to adjust its tap weights for operation in an unknown environment [6]. Figure 1.2 shows a typical structure of the adaptive filter, which consists of two basic functional blocks: (i) a digital filter to perform the desired filtering and (ii) an adaptive algorithm to adjust the tap weights of the filter. The digital filter computes the output $y(n)$ in response to the input signal $u(n)$, and generates an error signal $e(n)$ by comparing $y(n)$ with the desired response $d(n)$, which is also called the reference signal, as shown in Figure 1.1. The performance feedback signal $e(n)$ (also called the error signal) is used by the adaptive algorithm to adjust the tap weights of the digital filter.

The digital filter shown in Figure 1.2 can be realized using many different structures. The commonly used transversal or finite impulse response (FIR) filter is shown in

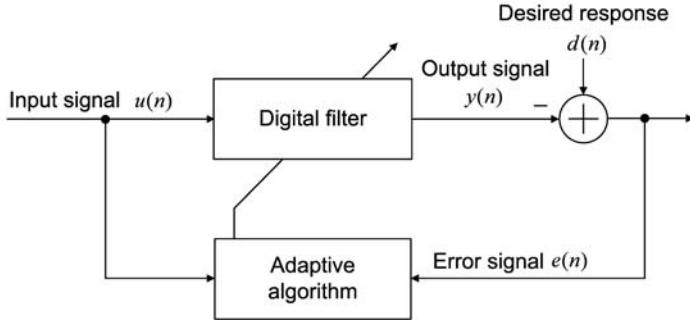


Figure 1.2 Typical structure of the adaptive filter using input and error signals to update its tap weights

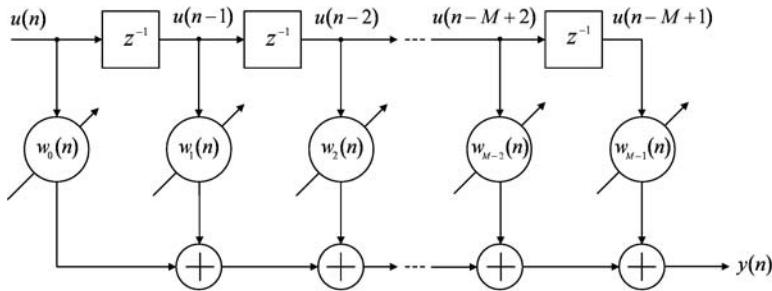


Figure 1.3 An M -tap adaptive transversal filter

Figure 1.3. The adjustable tap weights, $w_m(n)$, $m = 0, 1, \dots, M - 1$, indicated by circles with arrows through them, are the filter tap weights at time instance n and M is the filter length. These time-varying tap weights form an $M \times 1$ weight vector expressed as

$$\mathbf{w}(n) \equiv [w_0(n), w_1(n), \dots, w_{M-1}(n)]^T, \quad (1.1)$$

where the superscript T denotes the transpose operation of the matrix. Similarly, the input signal samples, $u(n - m)$, $m = 0, 1, \dots, M - 1$, form an $M \times 1$ input vector

$$\mathbf{u}(n) \equiv [u(n), u(n - 1), \dots, u(n - M + 1)]^T. \quad (1.2)$$

With these vectors, the output signal $y(n)$ of the adaptive FIR filter can be computed as the inner product of $\mathbf{w}(n)$ and $\mathbf{u}(n)$, expressed as

$$y(n) = \sum_{m=0}^{M-1} w_m(n)u(n - m) = \mathbf{w}^T(n)\mathbf{u}(n). \quad (1.3)$$

1.3 Performance surfaces

The error signal $e(n)$ shown in Figure 1.2 is the difference between the desired response $d(n)$ and the filter response $y(n)$, expressed as

$$e(n) = d(n) - \mathbf{w}^T(n)\mathbf{u}(n). \quad (1.4)$$

The weight vector $\mathbf{w}(n)$ is updated iteratively such that the error signal $e(n)$ is minimized. A commonly used performance criterion (or cost function) is the minimization of the mean-square error (MSE), which is defined as the expectation of the squared error as

$$J \equiv E\{e^2(n)\}. \quad (1.5)$$

For a given weight vector $\mathbf{w} = [w_0, w_1, \dots, w_{M-1}]^T$ with stationary input signal $u(n)$ and desired response $d(n)$, the MSE can be calculated from Equations (1.4) and (1.5) as

$$J = E\{d^2(n)\} - 2\mathbf{p}^T\mathbf{w} + \mathbf{w}^T\mathbf{R}\mathbf{w}, \quad (1.6)$$

where $\mathbf{R} \equiv E\{\mathbf{u}(n)\mathbf{u}^T(n)\}$ is the input autocorrelation matrix and $\mathbf{p} \equiv E\{d(n)\mathbf{u}(n)\}$ is the cross-correlation vector between the desired response and the input vector. The time index n has been dropped in Equation (1.6) from the vector $\mathbf{w}(n)$ because the MSE is treated as a stationary function.

Equation (1.6) shows that the MSE is a quadratic function of the tap weights $\{w_0, w_1, \dots, w_{M-1}\}$ since they appear in first and second degrees only. A typical performance (or error) surface for a two-tap transversal filter is shown in Figure 1.4. The corresponding MATLAB script for computing this performance surface is given in Example 1.1. For $M > 2$, the error surface is a hyperboloid. The quadratic performance surface guarantees that it has a single global minimum MSE corresponding to the

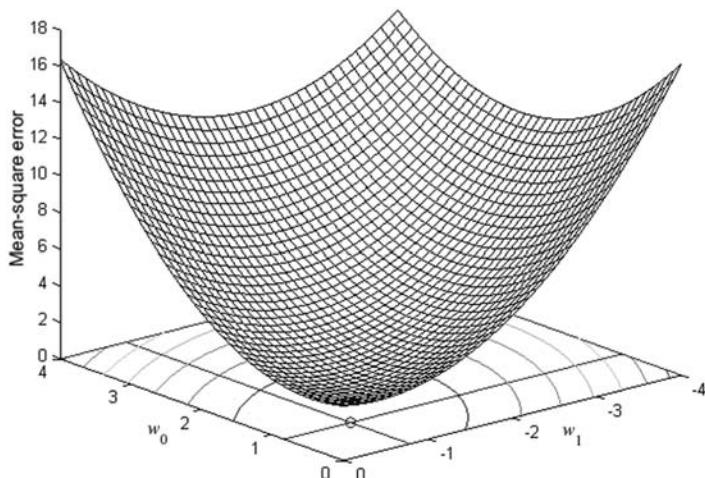


Figure 1.4 A typical performance surface for a two-tap adaptive FIR filter

optimum vector \mathbf{w}_o . The optimum solution can be obtained by taking the first derivative of Equation (1.6) with respect to \mathbf{w} and setting the derivative to zero. This results in the Wiener–Hopf equation

$$\mathbf{R}\mathbf{w}_o = \mathbf{p}. \quad (1.7)$$

Assuming that \mathbf{R} has an inverse matrix, the optimum weight vector is

$$\mathbf{w}_o = \mathbf{R}^{-1}\mathbf{p}. \quad (1.8)$$

Substituting Equation (1.8) into (1.6), the minimum MSE corresponding to the optimum weight vector can be obtained as

$$J_{\min} = E \{d^2(n)\} - \mathbf{p}^T \mathbf{w}_o. \quad (1.9)$$

Example 1.1 (a complete listing is given in the M-file `Example_1P1.m`)

Consider a two-tap case for the FIR filter, as shown in Figure 1.3. Assuming that the input signal and desired response are stationary with the second-order statistics

$$\mathbf{R} = \begin{bmatrix} 1.1 & 0.5 \\ 0.5 & 1.1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} 0.5272 \\ -0.4458 \end{bmatrix} \text{ and } E\{d^2(n)\} = 3.$$

The performance surface (depicted in Figure 1.4) is defined by Equation (1.6) and can be constructed by computing the MSE at different values of w_0 and w_1 , as demonstrated by the following partial listing:

```
R = [1.1 0.5; 0.5 1.1]; % Input autocorrelation matrix
p = [0.5272; -0.4458]; % Cross-correlation vector
dn_var = 3; % Variance of desired response d(n)

w0 = 0:0.1:4; % Range of first tap weight values
w1 = -4:0.1:0; % Range of second tap weight values
J = zeros(length(w0),length(w1));
% Clear MSE values at all (w0, w1)

for m = 1:length(w0)
    for n = 1:length(w1)
        w = [w0(m) w1(n)]';
        J(m,n) = dn_var-2*p'*w + w'*R*w;
        % Compute MSE values at all (w0, w1)
    end
    % points, store in J matrix
end

figure; meshc(w0,w1,J'); % Plot combination of mesh and contour
view([-130 22]); % Set the angle to view 3-D plot
hold on; w_opt = inv(R)*p;
plot(w_opt(1),w_opt(2), 'o');
```

```

plot(w0,w_opt(2)*ones(length(w0),1));
plot(w_opt(1)*ones(length(w1),1),w1);
xlabel('w_0'); ylabel('w_1');
plot(w0,-4*ones(length(w0),1)); plot(4*ones(length(w1),1),w1);

```

1.4 Adaptive algorithms

An adaptive algorithm is a set of recursive equations used to adjust the weight vector $\mathbf{w}(n)$ automatically to minimize the error signal $e(n)$ such that the weight vector converges iteratively to the optimum solution \mathbf{w}_o that corresponds to the bottom of the performance surface, i.e. the minimum MSE J_{\min} . The least-mean-square (LMS) algorithm is the most widely used among various adaptive algorithms because of its simplicity and robustness. The LMS algorithm based on the steepest-descent method using the negative gradient of the instantaneous squared error, i.e. $J \approx e^2(n)$, was devised by Widrow and Stearns [6] to study the pattern-recognition machine. The LMS algorithm updates the weight vector as follows:

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \mu \mathbf{u}(n) e(n), \quad (1.10)$$

where μ is the step size (or convergence factor) that determines the stability and the convergence rate of the algorithm. The function M-files of implementing the LMS algorithm are `LMSinit.m` and `LMSadapt.m`, and the script M-file that calls these MATLAB functions for demonstration of the LMS algorithm is `LMSdemo.m`. Partial listing of these files is shown below. The complete function M-files of the adaptive LMS algorithm and other adaptive algorithms discussed in this chapter are available in the companion CD.

The LMS algorithm

There are two MATLAB functions used to implement the adaptive FIR filter with the LMS algorithm. The first function `LMSinit` performs the initialization of the LMS algorithm, where the adaptive weight vector w_0 is normally clear to a zero vector at the start of simulation. The step size `mu` and the leaky factor `leak` can be determined by the user at run time (see the complete `LMSdemo.m` for details). The second function `LMSadapt` performs the actual computation of the LMS algorithm given in Equation (1.10), where u_n and d_n are the input and desired signal vectors, respectively. s is the initial state of the LMS algorithm, which is obtained from the output argument of the first function. The output arguments of the second function consist of the adaptive filter output sequence y_n , the error sequence e_n and the final filter state s .

```

S = LMSinit(zeros(M,1),mu); % Initialization
S.unknownsys = b;

```

```
[yn,en,S] = LMSadapt(un,dn,S); % Perform LMS algorithm
...
function S = LMSinit(w0,mu,leak)
% Assign structure fields
S.coeffs      = w0(:);           % Weight (column) vector of filter
S.step        = mu;              % Step size of LMS algorithm
S.leakage     = leak;             % Leaky factor for leaky LMS
S.iter        = 0;                % Iteration count
S.AdaptStart  = length(w0);     % Running effect of adaptive
                                % filter, minimum M

function [yn,en,S] = LMSadapt(un,dn,S)
M = length(S.coeffs);           % Length of FIR filter
mu = S.step;                   % Step size of LMS algorithm
leak = S.leakage;               % Leaky factor
AdaptStart = S.AdaptStart;
w = S.coeffs;                  % Weight vector of FIR filter
u = zeros(M,1);                 % Input signal vector
%
ITER = length(un);             % Length of input sequence
yn = zeros(1,ITER);             % Initialize output vector to zero
en = zeros(1,ITER);             % Initialize error vector to zero
%
for n = 1:ITER
    u = [un(n); u(1:end-1)];   % Input signal vector contains
                                % [u(n),u(n-1),...,u(n-M+1)]'
    yn(n) = w'*u;              % Output signal
    en(n) = dn(n) - yn(n);     % Estimation error
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
                                % System error norm (normalized)
    end
    if n >= AdaptStart
        w = (1-mu*leak)*w + (mu*en(n))*u; % Tap-weight adaptation
        S.iter = S.iter + 1;
    end
end
end
```

As shown in Equation (1.10), the LMS algorithm uses an iterative approach to adapt the tap weights to the optimum Wiener–Hopf solution given in Equation (1.8). To guarantee the stability of the algorithm, the step size is chosen in the range

$$0 < \mu < \frac{2}{\lambda_{\max}}, \quad (1.11)$$

where λ_{\max} is the largest eigenvalue of the input autocorrelation matrix \mathbf{R} . However, the eigenvalues of \mathbf{R} are usually not known in practice so the sum of the eigenvalues

(or the trace of \mathbf{R}) is used to replace λ_{\max} . Therefore, the step size is in the range of $0 < \mu < 2/\text{trace}(\mathbf{R})$. Since $\text{trace}(\mathbf{R}) = MP_u$ is related to the average power P_u of the input signal $u(n)$, a commonly used step size bound is obtained as

$$0 < \mu < \frac{2}{MP_u}. \quad (1.12)$$

It has been shown that the stability of the algorithm requires a more stringent condition on the upper bound of μ when convergence of the weight variance is imposed [3]. For Gaussian signals, convergence of the MSE requires $0 < \mu < 2/3MP_u$.

The upper bound on μ provides an important guide in the selection of a suitable step size for the LMS algorithm. As shown in (1.12), a smaller step size μ is used to prevent instability for a larger filter length M . Also, the step size is inversely proportional to the input signal power P_u . Therefore, a stronger signal must use a smaller step size, while a weaker signal can use a larger step size. This relationship can be incorporated into the LMS algorithm by normalizing the step size with respect to the input signal power. This normalization of step size (or input signal) leads to a useful variant of the LMS algorithm known as the normalized LMS (NLMS) algorithm.

The NLMS algorithm [3] includes an additional normalization term $\mathbf{u}^T(n)\mathbf{u}(n)$, as shown in the following equation:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \frac{\mathbf{u}(n)}{\mathbf{u}^T(n)\mathbf{u}(n)} e(n), \quad (1.13)$$

where the step size is now bounded in the range $0 < \mu < 2$. It makes the convergence rate independent of signal power by normalizing the input vector $\mathbf{u}(n)$ with the energy $\mathbf{u}^T(n)\mathbf{u}(n) = \sum_{m=0}^{M-1} u^2(n-m)$ of the input signal in the adaptive filter. There is no significant difference between the convergence performance of the LMS and NLMS algorithms for stationary signals when the step size μ of the LMS algorithm is properly chosen. The advantage of the NLMS algorithm only becomes apparent for nonstationary signals like speech, where significantly faster convergence can be achieved for the same level of MSE in the steady state after the algorithm has converged. The function M-files for the NLMS algorithm are `NLMSadapt.m` and `NLMSadapt.m`. The script M-file that calls these MATLAB functions for demonstration is `NLMSdemo.m`. Partial listing of these files is shown below.

The NLMS algorithm

Most parameters of the NLMS algorithm are the same as the LMS algorithm, except that the step size `mu` is now bounded between 0 and 2. The normalization term, `u'*u + alpha`, makes the convergence rate independent of signal power. An additional constant `alpha` is used to avoid division by zero in normalization operations.

```
...
S = NLMSinit(zeros(M,1),mu); % Initialization
S.unknownsys = b;
```

```
[yn,en,S] = NLMSadapt(un,dn,S); % Perform NLMS algorithm
...
function [yn,en,S] = NLMSadapt(un,dn,S)
M = length(S.coeffs); % Length of FIR filter
mu = S.step; % Step size (between 0 and 2)
leak = S.leakage; % Leaky factor
alpha = S.alpha; % A small constant
AdaptStart = S.AdaptStart;
w = S.coeffs; % Weight vector of FIR filter
u = zeros(M,1); % Input signal vector
%
ITER = length(un); % Length of input sequence
yn = zeros(1,ITER); % Initialize output vector to zero
en = zeros(1,ITER); % Initialize error vector to zero
...
for n = 1:ITER
    u = [un(n); u(1:end-1)]; % Input signal vector
    yn(n) = w'*u; % Output signal
    en(n) = dn(n) - yn(n); % Estimation error
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
    end % System error norm (normalized)
    if n >= AdaptStart
        w = (1-mu*leak)*w + (mu*en(n)/(u'*u + alpha))*u;
        % Tap-weight adaptation
        S.iter = S.iter + 1;
    end
end
end
```

Assuming that all the signals and tap weights are real valued, the FIR filter requires M multiplications to produce the output $y(n)$ and the update equation (1.10) requires $(M + 1)$ multiplications. Therefore, a total of $(2M + 1)$ multiplications per iteration are required for the adaptive FIR filter with the LMS algorithm. On the other hand, the NLMS algorithm requires additional $(M + 1)$ multiplications for the normalization term, giving a total of $(3M + 2)$ multiplications per iteration. Since M is generally large for most practical applications, the computational complexity of the LMS and NLMS algorithms is proportional to M , denoted as $O(M)$. Note that the normalization term $\mathbf{u}^T(n)\mathbf{u}(n)$ in Equation (1.13) can be approximated from the average power P_u , which can be recursively estimated at time n by a simple running average as

$$P_u(n) = (1 - \beta)P_u(n - 1) + \beta u^2(n), \quad (1.14)$$

where $0 < \beta \ll 1$ is the smoothing (or forgetting) parameter.

The affine projection (AP) algorithm [3] is a generalized version of the NLMS algorithm. Instead of minimizing the current error signal given in Equation (1.4), the AP algorithm increases the convergence rate of the NLMS algorithm by using a set of P constraints $d(n - k) = \mathbf{w}^T(n + 1)\mathbf{u}(n - k)$ for $k = 0, 1, \dots, P - 1$. Note that P is less

than the length M used in the adaptive FIR filter and results in an underdetermined case. Therefore, a pseudo-inversion of the input signal matrix is used to derive the AP algorithm as follows:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \mathbf{A}^T(n) [\mathbf{A}(n)\mathbf{A}^T(n)]^{-1} \mathbf{e}(n), \quad (1.15)$$

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n), \quad (1.16)$$

where the input signal matrix $\mathbf{A}^T(n) = [\mathbf{u}(n), \mathbf{u}(n-1), \dots, \mathbf{u}(n-P+1)]$ consists of P columns of input vectors of length M , $\mathbf{e}(n) = [e(n), e(n-1), \dots, e(n-P+1)]^T$ is the error signal vector and $\mathbf{d}(n) = [d(n), d(n-1), \dots, d(n-P+1)]^T$ is the desired signal vector. The computation of error signals given in Equation (1.16) is based on P constraints (or order) of the AP algorithm. More constraints (i.e. larger P) results in faster convergence but at the cost of higher complexity. When P reduces to one (i.e. a single column), the updating equation in (1.15) becomes a special case of the NLMS algorithm as shown in Equation (1.13). The computational complexity of the AP algorithm is $O(P^2M)$. The function M-files of the AP algorithm are `APinit.m` and `APadapt.m`, and the script M-file to demonstrate the AP algorithm is `APdemo.m`. The partial listing of these files is shown below.

The AP algorithm

The AP algorithm is a generalized version of the NLMS algorithm with P constraints and M taps of adaptive filter. In the MATLAB program, $P = 10$ and $M = 256$ are used in simulation. The normalization term, $\text{A}^*\text{inv}(\text{A}'*\text{A} + \text{alpha})$, is the pseudo-inversion of the data matrix A . The identity matrix alpha consists of a small constant of 10^{-4} along its diagonal elements to avoid division by zero in the matrix-inversion operations.

```

...
P = 10; M = 256;
S = APinit(zeros(M,1),mu,P); % Initialization
S.unknownsys = b;
[yn,en,S] = APadapt(un,dn,S); % Perform AP algorithm
...

function S = APinit(w0,mu,P)
% Assign structure fields
S.coeffs      = w0(:);          % Weight (column) vector of filter
S.step        = mu;             % Step size
S.iter        = 0;              % Iteration count
S.alpha       = eye(P,P)*1e-4;% A small constant
S.AdaptStart = length(w0)+P-1;% Running effect of adaptive
                               % filter and projection matrix
S.order       = P;              % Projection order

function [yn,en,S] = APadapt(un,dn,S)

```

```

M = length(S.coeffs); % Length of FIR filter
mu = S.step; % Step size (between 0 and 2)
P = S.order; % Projection order
alpha = S.alpha; % Small constant
AdaptStart = S.AdaptStart;
w = S.coeffs; % Weight vector of FIR filter
u = zeros(M,1); % Tapped-input vector
A = zeros(M,P); % Projection matrix
d = zeros(P,1); % Desired response vector
%
ITER = length(un); % Length of input sequence
yn = zeros(1,ITER); % Initialize output vector to zero
en = zeros(1,ITER); % Initialize error vector to zero
...
for n = 1:ITER
    u = [un(n); u(1:end-1)]; % Input signal vector contains
    % [u(n),u(n-1),...,u(n-M+1)]'
    A = [u A(:,1:end-1)]; % Data matrix
    d = [dn(n); d(1:end-1)]; % Desired response vector contains
    % [d(n),d(n-1),...,d(n-P+1)]'
    yn(n) = u'*w; % Output signal
    e = d - A'*w; % Error estimation
    en(n) = e(1); % Take the first element of e
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
        % System error norm (normalized)
    end
    if n >= AdaptStart
        w = w + mu*A*inv(A'*A + alpha)*e;
        % Tap-weight adaptation
    end
end

```

Unlike the NLMS algorithm that is derived from the minimization of the expectation of squared error, the recursive least-square (RLS) [3] algorithm is derived from the minimization of the sum of weighted least-square errors as

$$J_{LS}(n) = \sum_{i=1}^n \lambda^{n-i} e^2(i), \quad (1.17)$$

where λ is the forgetting factor and has a value less than and close to 1. The forgetting factor weights the current error heavier than the past error values to support filter operation in nonstationary environments. Therefore, in the least-square method, the weight vector $w(n)$ is optimized based on the observation starting from the first iteration ($i = 1$) to the current time ($i = n$). The least-square approach can also be expressed in the form similar to the Wiener–Hopf equation defined in Equation (1.7), where the autocorrelation

matrix and cross-correlation vector are expressed as $\mathbf{R} \approx \mathbf{R}(n) = \sum_{i=1}^n \lambda^{n-i} \mathbf{u}(i)\mathbf{u}^T(i)$ and $\mathbf{p} \approx \mathbf{p}(n) = \sum_{i=1}^n \lambda^{n-i} d(i)\mathbf{u}(i)$, respectively.

Following the derivation given in Reference [3] using the matrix inversion lemma, the RLS algorithm can be written as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{g}(n)e(n), \quad (1.18)$$

where the updating gain vector is defined as

$$\mathbf{g}(n) = \frac{\mathbf{r}(n)}{1 + \mathbf{u}^T(n)\mathbf{r}(n)} \quad (1.19)$$

and

$$\mathbf{r}(n) = \lambda^{-1} \mathbf{P}(n-1) \mathbf{u}(n). \quad (1.20)$$

The inverse correlation matrix of input data, $\mathbf{P}(n) \equiv \mathbf{R}^{-1}(n)$, can be computed recursively as

$$\mathbf{P}(n) = \lambda^{-1} \mathbf{P}(n-1) - \mathbf{g}(n) \mathbf{r}^T(n). \quad (1.21)$$

Note that the AP algorithm approaches the RLS algorithm when P increases to M and $\lambda = 1$. Since both the NLMS and RLS algorithms converge to the same optimum weight vector, there is a strong link between them. Montazeri and Duhamel [10] linked the set of adaptive algorithms including the NLMS, AP and RLS algorithms.

The computational complexity of the RLS algorithm is in the order of $O(M^2)$. There are several efficient versions of the RLS algorithm, such as the fast transversal filter with the reduced complexity to $O(M)$ [3]. Compared to the NLMS and AP algorithms, the RLS algorithm is more expensive to implement. The AP algorithm has a complexity that lies between the NLMS (for $P = 1$) and RLS (for $P = M$) algorithms. The function M-files of the RLS algorithm are `RLSinit.m` and `RLSadapt.m`, and the script M-file to demonstrate the RLS algorithm is `RLSdemo.m`. Partial listing of these files is shown below.

The RLS algorithm

The RLS algorithm whitens the input signal using the inverse of the input autocorrelation matrix. Its convergence speed is faster than the LMS, NLMS and AP algorithms under colored input signals. This MATLAB simulation uses the forgetting factor `lamda = 1` and the initial inverse input correlation matrix `P0 = eye(length(w0))*0.01`. The RLS algorithm updates the filter coefficients according to Equations (1.18) to (1.21).

```
...
S = RLSinit(zeros(M,1),lamda); % Initialization
```

```

S.unknownsys = b;
[yn,en,S] = RLSadapt(un,dn,S); % Perform RLS algorithm
...

function S = RLSinit(w0,lambda)
% Assign structure fields
S.coeffs      = w0(:);          % Weight (column) vector of filter
S.lamda       = lambda;         % Exponential weighting factor
S.iter        = 0;              % Iteration count
S.P0          = eye(length(w0))*0.01; % Initialize inverse input
                                % autocorrelation matrix
S.AdaptStart  = length(w0);    % Running effect

function [yn,en,S] = APadapt(un,dn,S)
M = length(S.coeffs);           % Length of FIR filter
lambda = S.lamda;              % Exponential factor
invlambda = 1/lambda;
AdaptStart = S.AdaptStart;
w = S.coeffs;                  % Weight vector of FIR filter
P = S.P0;                      % Inverse correlation matrix
u = zeros(M,1);                % Input signal vector
...
for n = 1:ITER
    u = [un(n); u(1:end-1)];   % Input signal vector contains
                                % [u(n),u(n-1),...,u(n-M+1)]';
    yn(n) = w'*u;             % Output of adaptive filter
    en(n)=dn(n)-yn(n);       % Estimated error
    if n >= AdaptStart
        r = invlambda*P*u;    % See equ. (1.20)
        g = r/(1+u'*r);       % See equ. (1.19)
        w = w + g.*en(n);     % Updated tap weights for the
                                % next cycle, (1.18)
        P = invlambda*(P-g*u'*P); % Inverse correlation matrix
                                % update, see(1.20) and (1.21)
    end
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;
                                % System error norm (normalized)
    end
end

```

1.5 Spectral dynamic range and misadjustment

The convergence behavior of the LMS algorithm is related to the eigenvalue spread of the autocorrelation matrix \mathbf{R} that is determined by the characteristics of the input signal $u(n)$. The eigenvalue spread is measured by the condition number defined as $\kappa(\mathbf{R}) = \lambda_{\max}/\lambda_{\min}$, where λ_{\max} and λ_{\min} are the maximum and minimum eigenvalues,

respectively. Furthermore, the condition number depends on the spectral distribution of the input signal [1, p. 97] and is bounded by the dynamic range of the spectrum $\Gamma_{uu}(e^{j\omega})$ as follows:

$$\kappa(\mathbf{R}) \leq \frac{\max_{\omega} \Gamma_{uu}(e^{j\omega})}{\min_{\omega} \Gamma_{uu}(e^{j\omega})}, \quad (1.22)$$

where $\max_{\omega} \Gamma_{uu}(e^{j\omega})$ and $\min_{\omega} \Gamma_{uu}(e^{j\omega})$ are the maximum and minimum of the input spectrum, respectively. The ideal condition $\kappa(\mathbf{R}) = 1$ occurs for white input signals. The convergence speed of the LMS algorithm decreases as the ratio of the spectral dynamic range increases. When the input signal is highly correlated with a large spectral dynamic range, the autocorrelation matrix \mathbf{R} is ill-conditioned due to the large eigenvalue spread and the LMS algorithm suffers from slow convergence. The deficiency of LMS and NLMS algorithms has been analyzed in the literature [1, 3, 5, 6, 9]. Subband adaptive filtering will be introduced in Section 1.7.2 to improve the convergence rate when the input signal is highly correlated.

The theoretical convergence curve of the LMS algorithm decays according to the factors $r_m = 1 - \mu\lambda_m$, where λ_m is the m th eigenvalue of \mathbf{R} [6]. Therefore, a time constant that defines the convergence rate of the MSE is obtained as

$$\tau_m = \frac{1}{2\mu\lambda_m}, \quad m = 0, 1, \dots, M-1. \quad (1.23)$$

Consequently, the slowest convergence is determined by the smallest eigenvalue λ_{\min} . Even though a large step size can result in faster convergence, it must be limited by the upper bound given in Equation (1.11), which is inversely proportional to the maximum eigenvalue λ_{\max} .

In practice, the weight vector $\mathbf{w}(n)$ deviates from the optimum weight vector \mathbf{w}_o in the steady state due to the use of the instantaneous squared error by the LMS algorithm. Therefore, the MSE in the steady state after the algorithm has converged is greater than the minimum MSE, J_{\min} . The difference between the steady-state MSE $J(\infty)$ and J_{\min} is defined as the excess MSE expressed as $J_{\text{ex}} = J(\infty) - J_{\min}$. In addition, the misadjustment is defined as

$$\mathcal{M} = \frac{J_{\text{ex}}}{J_{\min}} = \frac{\mu}{2} MP_u. \quad (1.24)$$

This equation shows that the misadjustment is proportional to the step size, thus resulting in a tradeoff between the misadjustment and the convergence rate given in Equation (1.23). A small step size results in a slow convergence, but has the advantage of smaller misadjustment, and vice versa. If all the eigenvalues are equal, Equations (1.23) and (1.24) are related by the following simple equation:

$$\mathcal{M} = \frac{M}{4 \times \tau_m} = \frac{M}{\text{settling time}}, \quad (1.25)$$

where the settling time is defined as four times the convergence rate τ_m . Therefore, a long filter requires a long settling time in order to achieve a small value of misadjustment,

which is normally expressed as a percentage and has a typical value of 10% for most applications.

1.6 Applications of adaptive filters

Adaptive filters are used in many applications because of their ability to operate in unknown and changing environments. Adaptive filtering applications can be classified in four categories: adaptive system identification, adaptive inverse modeling, adaptive prediction and adaptive array processing.

In this section, we introduce several applications of adaptive filtering. These applications are implemented in MATLAB code for computer simulation. MATLAB examples include acoustic echo cancellation, adaptive interference cancellation, adaptive line enhancer, active noise control, adaptive channel equalizer, acoustic feedback reduction for hearing aids and adaptive array processing. These applications provide a useful platform to evaluate the performance of different adaptive algorithms. Several plots include an ensemble or time-averaged square error, and the norm of weight error, echo return loss, spectrogram of error signal and magnitude-squared coherence, etc., are used to evaluate the performance of these algorithms. The complete MATLAB programs for these adaptive filtering applications are available in the companion CD.

1.6.1 Adaptive system identification

A structure of adaptive system identification (or modeling) is shown in Figure 1.5, where the adaptive filter is placed in parallel with an unknown system (or plant) to be identified. The adaptive filter provides a linear model that is the best approximation of the unknown system. The excitation signal $u(n)$ serves as the input to both the unknown system and the adaptive filter, while $\eta(n)$ represents the disturbance (or plant noise) occurring within the unknown system. The objective of the adaptive filter is to model the unknown system such that the adaptive filter output $y(n)$ closely matches the unknown system output $d(n)$. This can be achieved by minimizing the error signal $e(n)$, which is the difference between the physical response $d(n)$ and the model response $y(n)$. If the excitation signal

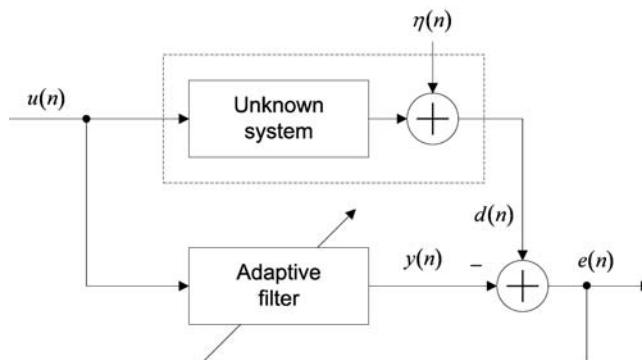


Figure 1.5 Block diagram of adaptive system identification

$u(n)$ is rich in frequency content, such as a white noise, and the plant noise $\eta(n)$ is small and uncorrelated with $u(n)$, the adaptive filter identifies the unknown system from the input–output viewpoint. The script M-file for the adaptive system identification is `SYSID.m` and its partial listing is shown below.

Adaptive system identification

The application of adaptive system identification is illustrated in the MATLAB program, where the plant to be modeled is a 32-tap lowpass FIR filter with a cutoff frequency of 0.3 radians per sample. The adaptive weight vector w_0 containing 32 coefficients and the white Gaussian noise are used as input u_n to both the plant and the adaptive filter. The desired signal d_n is the plant's output. The following listing uses only the LMS algorithm to update the adaptive filter. Other algorithms can also be used in order to compare their convergence performance with the LMS algorithm.

```
% SYSID.m      System Identification as Shown in Fig.1.5
iter = 1024;
un = 0.1*randn(1,iter);      % Input signal u(n)
b = fir1(31,0.3)';          % Unknown FIR system
dn = filter(b,1,un);         % Desired signal d(n)
w0 = zeros(32,1);            % Initialize filter coefficients to 0

% LMS algorithm

mulms = 0.5;                % Step size
...
Slms = LMSinit(w0,mulms);    % Initialization
Slms.unknownsys = b;
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                           % Perform LMS algorithm
```

The adaptive system identification has been widely applied in echo cancellation [2], active noise control [4], digital control [7], geophysics [6] and communications [5]. In fact, many problems can be formulated from the adaptive system identification point of view. A typical example is the acoustic echo cancellation (AEC), which finds many applications in hands-free telephones, audio (or video) conferencing systems, hearing aids, voice-control systems and much more. This book uses the adaptive identification of unknown systems with a long impulse response in later chapters to illustrate the performance of subband adaptive filters.

1.6.1.1 Acoustic echo cancellation

The acoustic echo in hands-free telephones arises when the microphone picks up the sound radiated by the loudspeaker and its reflections in a room [2, 11]. Figure 1.6 shows a general AEC configuration for hands-free telephony systems. Speech originating from the far-end talker is amplified and reproduced by a loudspeaker. The output of the loudspeaker

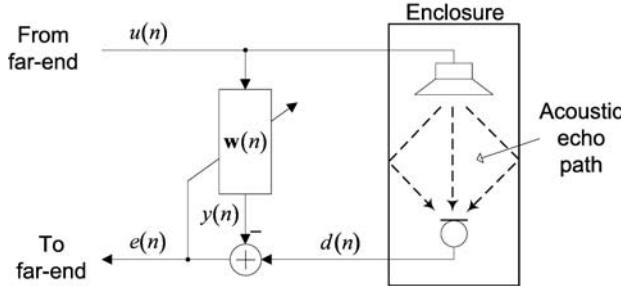


Figure 1.6 Block diagram of an acoustic echo canceller

will be picked up by the microphone via the acoustic echo path in the room. This signal is transmitted back to the far-end user where it is perceived as acoustic echo. The far-end user is annoyed by listening to his/her own speech delayed by the round-trip traveling time of the system.

The acoustic echo can be canceled electronically using an adaptive filter, as illustrated in Figure 1.6. The adaptive filter models the acoustic echo path between the loudspeaker and microphone. The estimated transfer function is used to filter the far-end speech signal $u(n)$ to generate an echo replica $y(n)$, which is then used to cancel the acoustic echo components in the microphone signal $d(n)$, resulting in the echo-free signal $e(n)$. The script M-file of the acoustic echo cancellation application is `AEC.m` and its partial listing is shown below.

Acoustic echo cancellation

Acoustic echo cancellation is an example of the adaptive system identification. The far-end speech signal `un` is used as input to both the room transfer function `ho` and the 1024-tap adaptive filter `w0`. No near-end speech signal is used in this simulation; instead, a room noise `vn` is added to the output of the room transfer function `out` to form the desired signal `dn`. An extension to this simulation adds near-end speech and investigates methods to handle double-talk conditions. In the partial listing, only the LMS algorithm is shown to update the adaptive filter. Other algorithms can also be used to compare their convergence performance and the system error norm with the LMS algorithm. Echo return loss enhancement is also computed in this script M-file.

```
% AEC.m      Application Program to Test Acoustics Echo Cancellation
%           as Shown in Fig.1.6
% Input far-end signal
[far_end,fsf,nbitf] = wavread('far.wav');
                           % Speech signal from far end
far_end = far_end(1:end);
...
un = far_end';               % Input signal
```

```
% Input near-end signal (room noise)
vn = 0.01.*randn(1,length(far_end)); % Additive room noise
% (variance ~ 0.01^2)
near_end = vn'; % No near-end speech
% (Single-talk)

...
% Load impulse response of room transfer function (300 ms)
load('h1.dat'); ho = h1(1:1024); % Truncate impulse response
% to 1024 samples
out = filter(ho,1,far_end'); % Filtering of far-end signal
% with room impulse response
dn = out+near_end';

% Perform adaptive filtering using LMS algorithm
mulms = 0.01; % Step size mu
...
Slms = LMSinit(w0,mulms); % Initialization
Slms.unknownsys = ho;
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
% Perform LMS algorithm
```

The following three factors make the AEC a very difficult task:

- (i) The acoustic echo path usually has long impulse response. The characteristics of the acoustic path depend on the physical conditions of the room. For a typical office, the duration of the impulse response (i.e. the reverberation time) is about 300 ms. At 8 kHz sampling frequency, an adaptive FIR filter of 2400 taps is required to model the acoustic echo path. As discussed in Section 1.4, a long adaptive FIR filter incurs a heavy computational load and results in slower convergence because of small step sizes, as indicated in (1.12).
- (ii) The characteristics of the acoustic echo path are time varying and may change rapidly due to the movement of people, doors opening or closing, temperature changes in the room, etc. Therefore, an adaptive algorithm has to converge quickly to track any changes by continuously updating the adaptive filter.
- (iii) The excitation speech signal is highly correlated and nonstationary. For a short speech segment, the spectral density may differ by more than 40 dB at different frequencies. This also results in slow convergence, as shown in Equation (1.22).

Many techniques have been proposed to overcome these problems [2, 3, 8, 11]. In this book, we will focus on using subband adaptive filters to tackle these problems.

1.6.1.2 Active noise control

A single-channel active noise control (ANC) for suppressing acoustic noise in an air duct is shown in Figure 1.7 [4]. The input signal $u(n)$ is acquired by a reference microphone placed near the noise source. The primary noise travels to the error microphone via

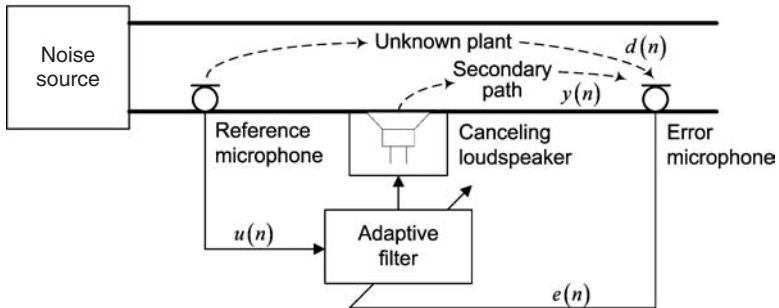


Figure 1.7 An example of single-channel active noise control viewed as an adaptive system identification problem

an unknown plant. At the same time, the canceling loudspeaker generates a secondary canceling noise that propagates to the error microphone via the secondary path. This secondary noise has the same amplitude with opposite phase to cancel the primary noise based on the principle of acoustic superposition. The residual noise picked up by the error microphone is used as the error signal $e(n)$ to update the adaptive filter. The ANC system operates in the electrical domain, while the unknown plant and secondary path are located in the electroacoustic domain. ANC is a challenging application since there are many considerations such as secondary path modeling and compensation, fast changing environment and noise characteristics, interactions between acoustic and electrical domains, placement of microphones and loudspeakers, and undesired acoustic feedback from the canceling loudspeaker to the reference microphone. The script M-file of the ANC application is `ANC.m` and its partial listing is shown below.

Active noise control

The application of active noise control is illustrated in MATLAB, where the engine wave file `engine_2.wav` is used as the source noise. The primary path is modeled as a rational transfer function with denominator `pri_den` and numerator `pri_num`. Similarly, the secondary path is modeled by denominator `sec_den` and numerator `sec_num`. A primary difference between the active noise control and other adaptive noise cancellation techniques is that the active noise control system does not generate an error signal electrically; instead, the error signal `en` is measured by the error microphone. Due to the presence of the secondary path, the filtered-x LMS (FXLMS) algorithm is used to adapt the coefficients of the adaptive filter. This FXLMS algorithm is implemented by function M-files `FXLMSinit.m` and `FXLMSadapt.m`.

```
% ANC.m          Application Program to Test Active Noise Control
%           as Shown in Fig.1.7
[noise,fs] = wavread('engine_2.wav'); % Engine noise
```

```

pri_num = load('p_z.asc'); % Primary path, numerator
pri_den = load('p_p.asc'); % Primary path, denominator
sec_num = load('s_z.asc'); % Secondary path, numerator
sec_den = load('s_p.asc'); % Secondary path, denominator
ir_sec = impz(sec_num, sec_den); % Actual secondary path
est_sec = ir_sec(1:150)'; % Estimated secondary path
dn = filter(pri_num, pri_den, noise)'; % Desired signal
un = noise'; % Input signal
M = 32; % Filter length
w0 = zeros(M,1); % Initialize coeffs to 0

% Perform adaptive filtering using FXLMS algorithm

mulms = 0.05; % Step size
...
Sfxlms = FXLMSinit(w0,mulms,est_sec,sec_num,sec_den); % Initialization
[yfms,enlms,Sfxlms] = FXLMSadapt(un,dn,Sfxlms); % Perform FXLMS algorithm

```

1.6.1.3 Adaptive noise cancellation

Adaptive noise (or interference) cancellation, illustrated in Figure 1.8, can be classified as an adaptive system identification problem [6]. The system response $d(n)$ consists of the desired signal $s(n)$ corrupted by noise $v'(n)$, i.e. $d(n) = s(n) + v'(n)$. Since there is spectra overlap between the desired signal and noise, conventional fixed filters may not be an effective solution in attenuating this noise. As shown in Figure 1.8, the input signal $u(n)$ consists of a noise $v(n)$ that is correlated to $v'(n)$, but must be uncorrelated with $s(n)$. The adaptive filter produces an output signal $y(n)$ that estimates $v'(n)$. Thus the error signal approximates the signal $s(n)$. If the signal $s(n)$ is considered as the disturbance $\eta(n)$ shown in Figure 1.5 and the noise path is treated as the unknown system, adaptive noise cancellation can be considered as a system identification problem.

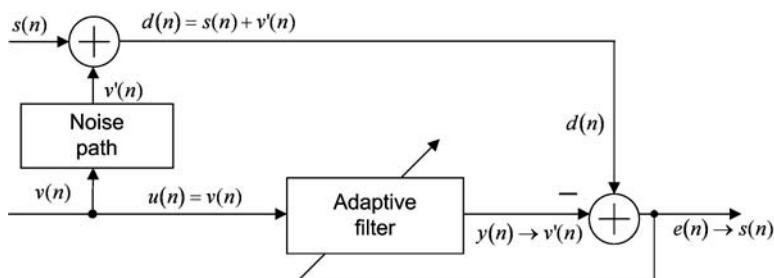


Figure 1.8 Block diagram of the adaptive noise canceller

When the signal $s(n)$ is absent during silent periods, the error signal approaches zero and an accurate system identification of the noise path can be obtained. The script M-file of the adaptive interference cancellation system is `AIC.m` and its partial listing is shown below.

Adaptive interference canceller

In MATLAB implementation of the adaptive interference canceller, the speech signal `timit.wav` is corrupted with white Gaussian noise. The input signal `un` is the source noise and the signal `dn` is the corrupted speech signal. The error signal `enlms` consists of a cleaner version of the speech signal after the output `ynlms` of the adaptive LMS filter has converged to the broadband noise.

```
% AIC.m      Application Program to Test Adaptive Noise
%           (Interference) Canceller as Shown in Fig.1.8
[y,fs] = wavread('timit.wav');          % Extract normalized wave file
noise=0.1*randn(size(y));              % Noisy signal
b = fir1(16,0.25);                   % Filter to simulate the noise
                                      % path
filt_noise = filter(b,1,noisy);       % Filtered version of noise
                                      % signal
dn = (y+filt_noise)';                % Speech corrupted with
                                      % filtered noise
un = noise';                         % Input signal to adaptive
                                      % filter
M = 32;                             % Filter length
w0 = zeros(M,1);                     % Initialize coefs to 0

% Perform adaptive filtering using LMS algorithm

mulms = 0.05;                        % Step size
...
Slms = LMSinit(w0,mulms);            % Initialization
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                      % Perform LMS algorithm
```

1.6.1.4 Acoustic feedback cancellation in hearing aids

A hearing aid modifies and amplifies incoming sound to make it audible for people with hearing loss [2]. Hearing aids are normally small in size to fit into the ear canal or worn behind the ear. Due to the close proximity between the sound emitting transducer (or receiver) and the microphone, acoustic feedback can occur in hearing aids or other hearing devices such as Bluetooth headsets. Figure 1.9 shows the model of hearing aids and the application of the adaptive filter to remove the acoustic feedback (leakage) from the receiver to the microphone. The forward path is the processing unit of the hearing aids and its output $u(n)$ is fed into the adaptive filter and the receiver. The reference signal $d(n)$ picked up by the microphone is the summation of incoming sound and the

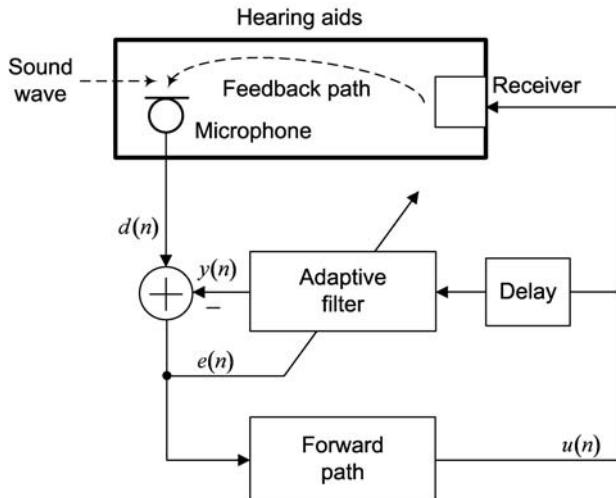


Figure 1.9 Block diagram of the hearing aids with acoustic feedback cancellation

feedback signal. The error signal $e(n)$ is derived by subtracting the adaptive filter output from the reference signal.

Due to the high correlation between the reference signal $d(n)$ and the input signal $u(n)$, the adaptive filter cannot model the feedback path accurately. A delay is used in the cancellation path to decorrelate the reference signal and the input signal. This acoustic feedback cancellation is also classified as the adaptive system identification problem that uses the adaptive filter to model the feedback path. The script M-file of the hearing aids application is `HA.m` and its partial listing is shown below.

Acoustic feedback reduction in hearing aids

In MATLAB implementation of acoustic feedback reduction, a speech signal `timit.wav` at the receiver is corrupted with white Gaussian noise `noise` to form the desired signal `dn`. The feedback path loaded from the data file `fb.dat` is used to simulate the feedback of the speech signal from the receiver to the microphone. A delay unit is used as the feedforward path `ff` for the simulation. The input signal `un` consists of the received speech signal when no acoustic feedback exists.

```
% HA.m          Application Program to Test Acoustic Feedback
% Reduction for Hearing Aids as Shown in Fig.1.9
load fb.dat;           % Load the feedback path
norm_fb = norm(fb);   % Norm-2 of feedback path
delay = 10;             % Delay in feedforward path
ff= [zeros(1,delay+1) 4]; % Simple feedforward path
[un,fs] = wavread('timit.wav'); % Extract normalized wave file
noise=0.01*randn(size(un)); % Noisy signal
```

```

dn = (un+noise)';
num = conv(ff,fb);
den = [1; -num(2:end)];
y = filter(num,den,dn);
M = 64;
w0 = zeros(M,1);

% Perform adaptive filtering using LMS algorithm

mulms = 0.005; % Step size
...
Slms = HALMSinit(w0,mulms,fb,ff,delay);
% Initialization
[ylms,enlms,fblms,Slms] = HALMSadapt(un,Slms);
% Perform LMS algorithm

```

1.6.2 Adaptive prediction

Adaptive prediction is illustrated in Figure 1.10, where the desired signal $d(n)$ is assumed to have predictable signal components [3]. This signal is delayed by Δ samples to form the input signal $u(n) = d(n - \Delta)$ for the adaptive filter to minimize the error signal $e(n)$. The adaptive filter predicts the current value of the desired signal based on the past samples. A major application of adaptive prediction is the waveform coding of speech, such as the adaptive differential pulse code modulation [1]. In this application, the adaptive filter exploits the correlation between adjacent samples of speech such that the prediction error is much smaller than the original signal on average. This prediction error is then quantized using fewer bits for transmission, resulting in a lower bit rate. Other applications of adaptive prediction include spectrum estimation and signal whitening.

Adaptive prediction is also used to separate the signal from noise based on differences between the signal and noise correlation. Depending on whether $y(n)$ or $e(n)$ is the desired output, the adaptive predictor enhances the narrowband signal corrupted by broadband noise or removes the narrowband interference from the corrupted broadband signal. With reference to Figure 1.10, the input of the adaptive line enhancement consists

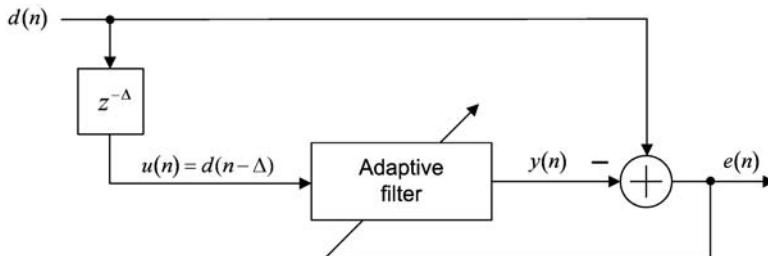


Figure 1.10 Block diagram of the adaptive predictor

of desired narrowband (or tonal) components $t(n)$ corrupted by broadband noise $v(n)$, i.e. $d(n) = t(n) + v(n)$. The delay Δ must be sufficiently large to decorrelate the broadband noise components, but the narrowband components still correlate in $d(n)$ and $u(n)$. The adaptive filter compensates for the delay (or phase shift) in order to cancel the narrowband components in $d(n)$. Therefore, the filter output $y(n)$ estimates the narrowband components $t(n)$ only, while the error signal $e(n)$ approximates the wideband noise $v(n)$. This structure enhances the narrowband signal at the adaptive filter output $y(n)$ and is called the adaptive line enhancer. Usually, $\Delta = 1$ is adequate for white noise and $\Delta > 1$ is used for color noise. The script M-file of the adaptive line enhancer is `ALE.m` and allows the user to select different modes of operations.

Adaptive line enhancer

The adaptive line enhancer can be used either to enhance the tonal signal or enhance the broadband (such as speech) signal corrupted by tonal noise. As shown in Figure 1.10, the input signal u_n to the adaptive filter is the delayed version of the mixed input of desired signal and noise. The delay unit is used to decorrelate the broadband noise component so that the adaptive filter output y_{lms} converges to the narrowband component of the mixed signal. The desired signal d_n is the delayless version of the mixed signal and is subtracted from the output signal to obtain the error signal e_{lms} , which consists of the enhanced broadband signal such as speech. Therefore, the adaptive line enhancer can be programmed to extract the narrowband or broadband component depending on the given application.

```
% ALE.m          Application Program to Test Adaptive Line Enhancer
%
%           as Shown in Fig.1.10
select = input('Select (1) separate noise from tone, (2) separate
speech from tone:','s');
if select == '1'
    f = 400; fs = 4000;           % Tonal signal at 400 Hz and
                                  % sampling at 4000 Hz
    iter = 8000; n = 0:1:iter-1;% Length of signal and time index
    sn = sin(2*pi*f*n/fs);     % Generate sinewave
    noise=randn(size(sn));      % Generate random noise
    dn = sn+0.2*noise;          % Mixing sinewave with white noise
    delay = 1;                  % Delay by 1
    un = [zeros(1,delay) dn(1,1:length(dn)-delay)]; % Generate delayed version of d(n)
else
    [y,fs] = wavread('timit.wav');% Extract normalized wave file
    f = 1000;                      % Tone frequency 1000 Hz
    iter = length(y);              % Length of signal
    n = 0:1:iter-1;                % Time index
    sn = sin(2*pi*f*n/fs);        % Generate sinewave
    dn = sn+y';                   % Mixing sinewave with wave file
    delay = 5;                     % Delay by 5
    un = [zeros(1,delay) dn(1,1:length(dn)-delay)];
```

```

        % Generate delayed version of d(n)
end
M = 64;                                % Filter length
w0 = zeros(M,1);                         % Initialize filter coeffs to 0

% Perform adaptive filtering using LMS algorithm

mulms = 0.01;                            % Step size
...
Slms = LMSinit(w0,mulms);                % Initialization
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                         % Perform LMS algorithm

```

One practical application is to enhance the tonal signal in the dual-tone multi-frequency [12] signals found in touchtone phones. Alternatively, this structure can be used as an adaptive notch filter where the error signal $e(n)$ is the system output. One possible application of the adaptive notch filter is to remove any tonal noise (e.g. power hum) from the wideband signal, such as a biomedical measurement signal.

1.6.3 Adaptive inverse modeling

Adaptive filters can be applied for adaptive inverse modeling (or channel equalization) applications [7], where an unknown system is cascaded with an adaptive filter, as illustrated in Figure 1.11. The desired signal $d(n)$ is derived by delaying L samples of the input signal $s(n)$ using a delay unit z^{-L} to compensate for the overall propagation delay through the unknown system and the adaptive filter. The proper delay allows the adaptive filter to converge to a causal filter that is the inverse of the unknown system. Thus the adaptive filter equalizes the unknown system to recover the delayed version of signal $s(n - L)$ at the output $y(n)$. The combined transfer function of the unknown system and the adaptive filter approaches z^{-L} . In some applications, such as the adaptive channel equalizer in data modems, the desired signal is derived from the predefined data sequence during the training stage.

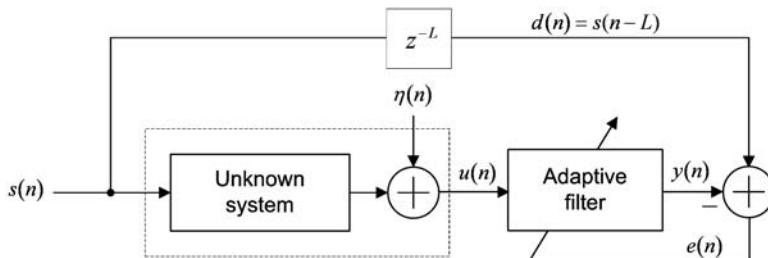


Figure 1.11 Inverse channel modeling using an adaptive filter

Similar to the system modeling shown in Figure 1.5, if $s(n)$ has a flat spectrum and the plant noise is small and uncorrelated with $s(n)$, the adaptive filter can adapt to an accurate inverse model of the unknown system. It is interesting to note that the adaptive line enhancer (or adaptive predictor) described in Figure 1.10 can be considered as inverse modeling by treating the delay $z^{-\Delta}$ as the unknown channel, and the adaptive filter converges to $z^{-L+\Delta}$. If $L = 0$ (as in the case of an adaptive predictor), the adaptive filter converges to a noncausal system of z^Δ . Therefore, the adaptive filter predicts the input at Δ samples ahead. The script M-file of the adaptive inverse modeling is `INVSYSID.m` and its partial listing is shown below.

Inverse system identification

The application of adaptive inverse system identification is illustrated in the MATLAB program `INVSYSID`. Here, a binary random signal is generated and transmitted to a channel modeled by a three-tap FIR filter `h_1`. The parameter `w` controls the amount of amplitude distortion produced by the channel. The output of the channel `out` becomes the input `un` to the adaptive filter. The length of the adaptive filter `eq_length` is 11 and the coefficients of adaptive filter are all initialized to zero. The desired signal `dn` is the delayed version of the binary signal. By selecting the delay to match the midpoint of the combined length of channel and adaptive filter impulse responses, the adaptive LMS filter is able to approximate the inverse of the channel response.

```
% INVSYSID.m    Inverse System Identification as Shown in Fig. 1.11

...
s = 2*(rand(1,iter) > 0.5) - 1;      % Generate binary input signal

% Impulse response of channel 1
W = [2.9 3.1 3.3 3.5];                % Parameters affect the
                                         % eigenvalue spread (evs)
h_1 = 0.5*(1+cos((2*pi/W(1))*(1:3) - 2)));
                                         % Channel 1 impulse response (evs = 6.0782)
% Select channel 1

h = h_1;
ch_delay = fix(length(h)/2);           % Channel delay = 2
eq_length = 11;                        % Length of adaptive equalizer
eq_delay = fix(eq_length/2);            % Delay of channel equalizer
total_delay = ch_delay + eq_delay;     % Total delay
out = filter(h,1,s);                  % Channel output signal
un = out + sqrt(0.001).*randn(1,iter); % Input to adaptive filter
dn = [zeros(1,total_delay) s(1,1:length(un)-total_delay)];
                                         % Desired signal
w0 = zeros(eq_length,1);               % Initialize filter coeffs to 0

% LMS algorithm
```

```

mulms = 0.075; % Step size
...
Slms = LMSinit(w0,mulms); % Initialization
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
% Perform LMS algorithm

```

The above description of an adaptive channel equalizer assumes that the additive noise $\eta(n)$ shown in Figure 1.11 can be ignored. When significant noise is present, the adaptive filter cannot converge to the exact inverse of the channel as effort is also needed to minimize the additive noise. In addition, due to the channel inversion function, the adaptive filter may amplify the noise at frequency bands where the unknown system has small magnitude responses. Therefore, special considerations are needed to prevent excessive equalization at the expense of noise enhancement.

Inverse modeling is widely used in adaptive control, deconvolution and channel equalization. For example, the adaptive equalizer is used to compensate the distortion caused by transmission over telephone and radio channels [5]. In adaptive channel equalization, $s(n)$ is the original data at the transmitter that is transmitted to the receiver through the unknown communication channel. At the receiver, the received data $u(n)$ is distorted because each data symbol transmitted over a time-dispersive channel extends beyond the time interval used to represent that symbol, thus causing an overlay of received symbols. Since most channels are time varying and unknown in advance, the adaptive filter is required to converge to the inverse of the unknown and time-varying channel so that $y(n)$ approximates the delayed version of $s(n)$.

After a short training period using pseudo-random numbers known to both the transmitter and receiver in modems, the transmitter begins to transmit the modulated data signal $s(n)$. To track the possible changes in the channel during data transmission, the adaptive equalizer must be adapted continuously while receiving the data sequence. However, the desired signal is no longer available to the adaptive equalizer in the receiver located at the other end of the channel. This problem can be solved by treating the output of the decision device at the receiver as correct and using it as the desired signal $d(n)$ to compute the error signal, as shown in Figure 1.12. This technique is called the decision-feedback equalizer, which works well when decision errors occur infrequently. The script M-file of the adaptive channel equalizer is `CH_EQ.m` and its partial listing is shown below.

Adaptive channel equalization

The channel equalization is commonly used to illustrate the principle behind inverse system identification. As illustrated in Figure 1.12, there are two modes of channel equalization: training and decision-directed modes. The main difference between these modes lies in the generation of the desired signal. In the training mode, the desired signal is obtained from the delayed version of the binary random signal, as described in the previous inverse system identification example. In the

decision-directed mode, the desired signal is derived from the decision device at the receiver end. Therefore, a different MATLAB function `LMSadapt_dec` is used for the decision-directed mode.

```
% CH_EQ.m          Channel Equalizer as Shown in Fig.1.12.

...
% Select channel 1

h = h_1;
ch_delay = fix(length(h)/2);           % Channel delay, 2
eq_length = 11;                      % Length of adaptive equalizer
eq_delay = fix(eq_length/2);          % Delay of channel equalizer
total_delay = ch_delay + eq_delay;
out = filter(h,1,s);                  % Channel output signal
un = out(1:iter_train) + sqrt(0.001).*randn(1,iter_train);
                                       % Input to adaptive filter
un_dec = out(iter_train+1:iter_total) +
sqrt(0.001).*randn(1,iter_dec);      % Input to adaptive filter
                                       % in decision directed mode
dn = [zeros(1,total_delay) s(1,1:iter_train-total_delay)];
                                       % Desired signal in
                                       % training mode
w0 = zeros(eq_length,1);              % Initialize filter coeffs to 0

% Adaptive filtering using LMS algorithm

mulms = 0.075;                      % Step size mu
...
Slms = LMSinit(w0,mulms);           % Training mode
[ylms,enlms,Slms] = LMSadapt(un,dn,Slms);
                                       % Initialization
Slms_dec = LMSinit(Slms.coeffs,mulms/10); % Decision-directed mode
[ylms_dec,enlms_dec,Slms_dec] = LMSadapt_dec(un_dec,Slms_dec);
                                       % Perform LMS algorithm
```

1.6.4 Adaptive array processing

So far, we have introduced adaptive filtering applications that combine weighted input signal samples at different time instants to produce an output signal that is related to the desired signal. This type of filtering is classified as temporal filtering. In this section, we discuss an alternate adaptive filtering application that acquires input signals at different positions to form a spatial filtering approach. The term adaptive array processing is commonly used to describe multiple adaptive filters that perform spatial filtering. An example of adaptive array processing [5] is shown in Figure 1.13, where the array of multiple sensors (spaced l units apart) with steering delays are used to pick up signals and interference arriving from different directions. The main objective of adaptive array

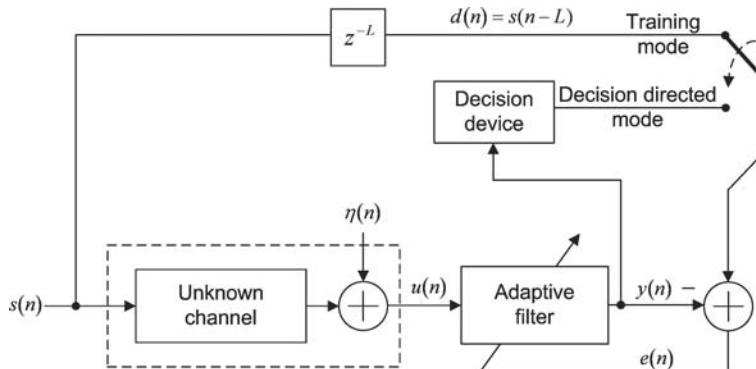


Figure 1.12 Two modes (training and decision directed) of operation in adaptive channel equalization

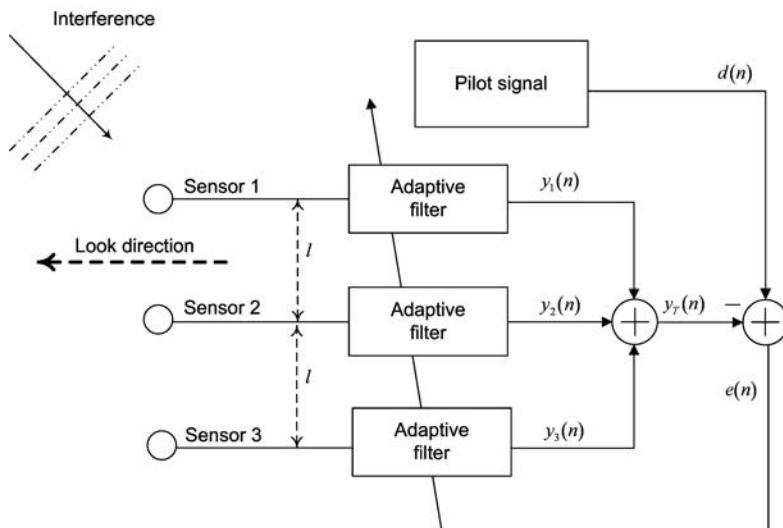


Figure 1.13 An example of adaptive array processing with three sensors

processing is to form a beam that produces a peak array gain in the desired look direction and adaptively to cancel interference outside the desired beam.

Each array element is connected to an adaptive filter that produces an output that is subsequently combined with the outputs of other adaptive filters to form an array pattern in the desired look direction. In order to pick up the desired signal in the look direction, a pilot signal that consists of the desired spectral and directional characteristics is used as the desired signal $d(n)$. The error signal $e(n)$ is used to adjust the adaptive filters to form a beam pattern in the look direction, and at the same time eliminate interference coming from other directions.

The sensor array in the adaptive array processing system is analogous to the tapped-delay line of the adaptive FIR filter shown in Figure 1.3. At any particular instant of

Table 1.1 Summary of adaptive filtering applications

Application types	Desired (or reference) signal, $d(n)$	Input signal, $u(n)$	Error signal, $e(n)$	Applications
System identification	Output from the unknown plant/system, may be corrupted by noise.	Excitation signal or processed version of excitation signal.	Difference between the unknown system output and the adaptive filter output. The adaptive filter converges to the approximation of the unknown system.	<ul style="list-style-type: none"> • Acoustic/network echo cancellation. • Active noise control. • Adaptive noise cancellation. • Feedback cancellation.
Inverse system identification	A delayed or processed version of the reference signal.	Output from the unknown plant/system, may be corrupted by noise.	Difference between the desired signal and the adaptive filter output. The adaptive filter converges to an inverse of the unknown system.	<ul style="list-style-type: none"> • Channel equalization for communication systems and disc drive.
Prediction	Excitation signal consists of both wideband signal (noise) and narrowband noise (signal).	Delayed version of the excitation signal.	Error signal consists of the wideband signal (noise), while the output of the adaptive filter consists of narrowband noise (signal). The adaptive filter converges to a narrowband bandpass filter.	<ul style="list-style-type: none"> • Adaptive line enhancer. • Predictive coding. • Spectral analysis.
Array processing	Pilot signal or some reference waveform in the look direction.	An array of input signals that detect signal from different directions.	Error signal is the difference between the combined adaptive filters' outputs and the pilot signal. Adaptive filter converges to a beam pattern in the look direction.	<ul style="list-style-type: none"> • Adaptive array processing for hearing aids. • Antenna adaptive array.

time, spatial signals captured in the sensor array are individually filtered by its respective adaptive filter and combined to capture the target signal with the maximum signal-to-noise ratio. Similarly, the adaptive filter adaptively combines the input samples to form an output signal that approximates the desired (target) signal.

1.6.5 Summary of adaptive filtering applications

A summary of adaptive filtering applications is listed in Table 1.1 to highlight the main differences among different applications. It is noted that the desired (or reference) signal is not always available for a given application. The reference signal $d(n)$ may be derived from the input signal $u(n)$. For example, in the adaptive channel equalization operating in the decision-directed mode (Figure 1.12), the desired signal is derived from a nonlinear decision device. Alternatively, the error signal $e(n)$ may be derived without the reference signal. For example, a constant modulus adaptive algorithm [5] is applied to a constant-envelope signal in communications. Based on a priori information that the amplitude of signal is constant, we can compare the adaptive filter output to this constant value and derive an error signal without the desired signal. Another example is the active noise control shown in Figure 1.7, where the noise cancellation (acoustic superposition) is performed in the acoustic domain. Therefore, a microphone is used to sense the residual noise as the error signal $e(n)$ for filter adaptation.

1.7 Transform-domain and subband adaptive filters

As explained in Sections 1.2 and 1.3, the most widely used adaptive structure is the transversal (or FIR) filter shown in Figure 1.3. This structure operates in the time domain and can be implemented in either the sample or block processing mode [13]. However, the time-domain LMS-type adaptive algorithms associated with the FIR filter suffer from high computational cost if applications (such as acoustic echo cancellation) demand a high-order filter and slow convergence if the input signal is highly correlated.

This section describes the transform-domain and subband adaptive filters [1, 3] that have advantages of low computational cost and fast convergence. The differences between subband and transform-domain adaptive filters are highlighted. Some insights into the relation between these structures are briefly introduced. This section also provides a brief introduction to subband adaptive filtering, which is suitable for adaptive filtering applications that need a long filter length, such as acoustics echo cancellation.

1.7.1 Transform-domain adaptive filters

This subsection highlights two classes of transform-domain adaptive filters including frequency-domain adaptive filters and self-orthogonalizing adaptive filters.

1.7.1.1 Frequency-domain adaptive filters

The frequency-domain adaptive filter transforms the desired signal $d(n)$ and the input signal $u(n)$ using the FFT and performs frequency-domain filtering and adaptation based on these transformed signals. As discussed in Sections 1.2 and 1.3, the time-domain adaptive filter performs linear convolution (filtering) and linear

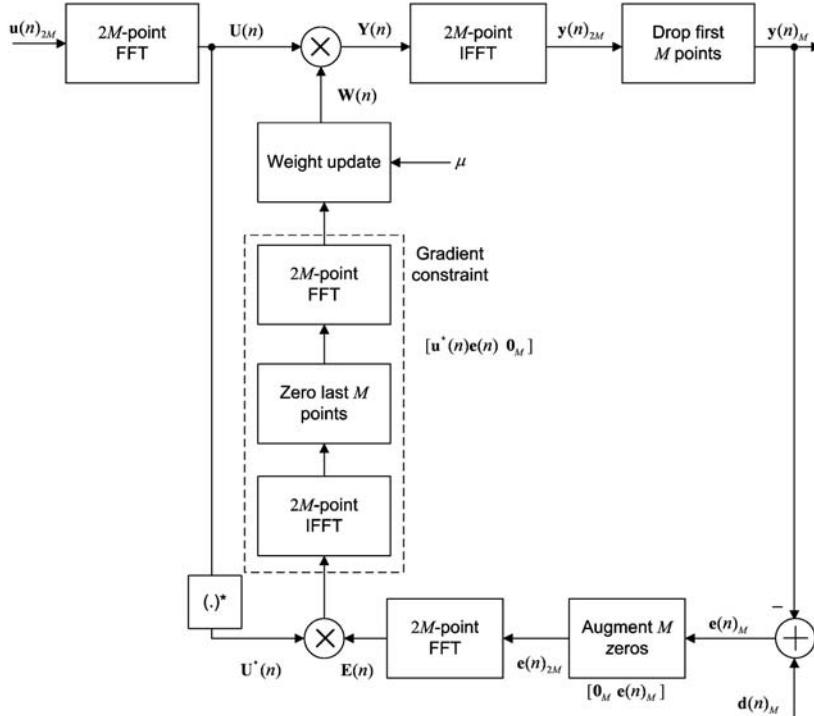


Figure 1.14 Block diagram of the frequency-domain fast LMS algorithms

correlation (weight updating) iteratively based on the arrival of new data samples. The frequency-domain adaptive filter shown in Figure 1.14 performs frequency transforms and processing over a block of input and desired signals, thus achieving substantial computational savings by using an FFT, especially for applications that use high-order filters. Unfortunately, the frequency-domain operations result in circular convolution and circular correlation. Thus a more complicated overlap-add or overlap-save method [13] is required to overcome the error introduced in these circular operations.

Figure 1.14 shows the block diagram of the frequency-domain adaptive filter using the fast LMS algorithm [3, 4]. This algorithm employs the overlap-save method with 50% overlap and requires five 2M-point FFT/IFFT operations.

In order to achieve the correct linear convolution and correlation results in frequency-domain filtering, extended data vectors are required. This type of processing is known as block processing, with the following definitions:

Input vector:

$$\mathbf{u}(n)_{2M} = [u(n - 2M + 1), \dots, u(n - M), u(n - M + 1), \dots, u(n - 1), u(n)]^T$$

FFT of input vector:

$$\mathbf{U}(n) = \text{FFT}[\mathbf{u}(n)] = [U_{2M-1}(n), \dots, U_M(n), U_{M-1}(n), \dots, U_1(n), U_0(n)]^T$$

Error vector:

$$\mathbf{e}(n)_M = [e(n), e(n-1), \dots, e(n-M+1)]^T$$

FFT of error vector:

$$\mathbf{E}(n) = \text{FFT}[\mathbf{0}_M \mathbf{e}(n)_M] = [E_0(n), E_1(n), \dots, E_{2M-1}(n)]^T$$

Note that the subscripts $2M$ and M denotes the length of the vectors $\mathbf{u}(n)_{2M}$ and $\mathbf{e}(n)_M$, respectively. The input vector $\mathbf{u}(n)_{2M}$ concatenates the previous M data samples with the present M data samples, and a $2M$ -point FFT operates on this input vector to form the frequency-domain vector $\mathbf{U}(n)$. Note that $\mathbf{U}(n)$ is a $2M$ -point complex vector.

The output signal vector from the adaptive filter can be computed by taking the element-by-element multiplication between the input vector and the $2M$ -point weight vector as

$$\mathbf{Y}(n) = \mathbf{W}(n) \cdot \mathbf{U}(n), \quad (1.26)$$

where the operator ‘.’ is the corresponding element-by-element multiplication for the frequency index $k = 0, 1, \dots, 2M - 1$ and $\mathbf{W}(n) = [W_0(n), W_1(n), \dots, W_{2M-1}(n)]^T$ is the complex-valued weight vector at time n . In order to obtain the correct result using circular convolution, the frequency-domain output vector, $\mathbf{Y}(n) = [Y_0(n), Y_1(n), \dots, Y_{2M-1}(n)]^T$, is transformed back to the time domain using the inverse FFT (IFFT) and the first M points of the $2M$ -point IFFT outputs are discarded to obtain the output vector $\mathbf{y}(n)_M = [y(n), y(n-1), \dots, y(n-M+1)]^T$. The output vector is subtracted from the desired vector $\mathbf{d}(n) = [d(n), d(n-1), \dots, d(n-M+1)]^T$ to produce the error signal vector $\mathbf{e}(n)_M$. The error vector $\mathbf{e}(n)$ is then augmented with M zeros and transformed to the frequency-domain vector $\mathbf{E}(n)$ using the FFT, as shown in Figure 1.14. The adaptation of the filter coefficients can be expressed as

$$W_k(n+M) = W_k(n) + \mu U_k^*(n) E_k(n), \quad k = 0, 1, \dots, 2M - 1, \quad (1.27)$$

where $U_k^*(n)$ is the complex conjugate of the frequency-domain signal $U_k(n)$. This weight-updating equation is known as the block complex LMS algorithm [3]. Note that the weight vector is not updated sample by sample as the time-domain LMS algorithm; it is updated once for a block of M samples. Also, the weight variables are complex valued due to the FFT.

As shown in the gradient constraint box in Figure 1.14, the weight-updating terms $[\mu \mathbf{U}^*(n) \cdot \mathbf{E}(n)]$ are inverse transformed, and the last M points are set to zeros before taking the $2M$ -point FFT for weight updating. An alternate algorithm, called the unconstrained frequency-domain adaptive filter [14], removes the gradient constraint block, thus producing a simpler implementation that involves only three transform operations. Unfortunately, this simplified algorithm no longer produces a linear correlation between the transformed error and input vectors, thus resulting in poorer performance compared to the frequency-domain adaptive filters with gradient constraints. Nevertheless, frequency-domain adaptive filters with fast convolution and correlation schemes are able to achieve lower computational cost as compared to high-order time-domain adaptive filters. The function M-files of the frequency-domain adaptive filter algorithm are

`FDAInit.m` and `FDAFadapt.m`, and the script M-file that calls these MATLAB functions is `FDAFDemo.m`. Partial listing of these files is shown below.

The frequency-domain adaptive filter (FDAF) algorithm

The function `FDAFinit` performs the initialization of the FDAF algorithm, where the weight vector w_0 is normally set to a zero vector at the start of the simulation. The function `FDAFadapt` performs the actual computation of the FDAF algorithm given in Equation (1.27). Note that both constrained and unconstrained versions of the FDAF algorithm are implemented in the program, and users have the option to select one. The step sizes `mu` and `mu_unconst` are used in the FDAF algorithm with gradient constraint and gradient unconstraint, respectively. Note that the function `FDAFadapt` calls another MATLAB function `fdaf` to perform a frequency-domain adaptive filter algorithm, which is modified from John Forte, BWV Technologies Inc. Version v1.0.

```

M = 256;
mu = 0.5; % Step size for constrained Fdaf
mu_unconst = 0.001; % Step size for unconstrained Fdaf
...
S = FDAFinit(zeros(M,1),mu,mu_unconst,iter); % Initialization
S.unknownsys = b;
[en,S] = FDAFadapt(un,dn,S); % Perform Fdaf algorithm
...

function S = FDAFinit(w0,mu,mu_unconst,iter)
...
S.coeffs = w0; % Coefficients of FIR filter
S.length = M; % Length of adaptive filter
S.step = mu; % Step size of constrained Fdaf
S.step_unconst = mu_unconst; % Step size of unconstrained Fdaf
S.iter = 0; % Iteration count
S.AdaptStart = length(w0); % Running effect of adaptive filter
S.weight = WEIGHT; % FFT of zero-padded weight vector
S.palpha = 0.5; % Constant to update power in
% each frequency bin
S.ref = un_blocks; % 2M-sample block of input signal

function [en,S] = FDAFadapt(un,dn,S)
...
for n = 1:ITER
    u_new = [u_new(2:end); un(n)]; % Start input signal blocks
    dn_block = [dn_block(2:end); dn(n)]; % Start desired signal block
    if mod(n,M)==0 % If iteration == block length
        ...
    end
end

```

```

un_blocks = [u_old; u_new];
u_old = u_new;
b = S.unknownsys;
if n >= AdaptStart % Frequency-domain adaptive filtering
    [error,WEIGHT,power,misalign] =
fdaf(M,mu,mu_unconst,power_alpha,WEIGHT,dn_block,un_blocks,power,b
,select);
    en = [en; error];% Update error block
    eml = [eml; misalign];
        % Update misalignment at each block
    end
end
end

```

1.7.1.2 Self-orthogonalizing adaptive filters

As explained in Section 1.5, the eigenvalue spread plays an important role in determining the convergence speed of the time-domain adaptive filters. When the input signal is highly correlated, the convergence of the LMS algorithm is very slow. Many methods were developed to solve this problem. One method is to use the RLS algorithm (see Section 1.4), which extracts past information to decorrelate the present input signal [3]. This approach suffers from high computational cost, poor robustness, low numerical stability and slow tracking ability in nonstationary environments. The other method is to decorrelate the input signal using a transformation such as the discrete Fourier transform (DFT) or discrete cosine transform (DCT) [1]. This method is known as the self-orthogonalizing adaptive filter. Compared with the RLS approach, the self-orthogonalizing adaptive filter saves computational cost and is independent of the characteristics of the input signal.

Figure 1.15 shows the self-orthogonalizing adaptive filter. It has a similar structure to the M -tap adaptive transversal filter shown in Figure 1.3, but with preprocessing (transform and normalization) on the input signal using the DFT or DCT. The transform is carried out for every new input signal sample. The transformed signals $U_k(n)$ are normalized by the square root of its respective power in the transform domain as

$$U'_k(n) = \frac{U_k(n)}{\sqrt{P_k(n) + \delta}}, \quad k = 0, 1, \dots, M - 1, \quad (1.28)$$

where δ is a small constant to prevent division by zero and $P_k(n)$ is the power that can be estimated recursively as

$$P_k(n) = (1 - \beta)P_k(n - 1) + \beta U_k^2(n). \quad (1.29)$$

Note that the power $P_k(n)$ is updated iteratively for every new sample arriving at the tapped-delay line and β is a forgetting factor that is usually chosen as $1/M$. The resulting transform-domain signals with similar power are processed by the adaptive linear combiner whose tap weights are updated by the LMS algorithm given in Equation (1.10).

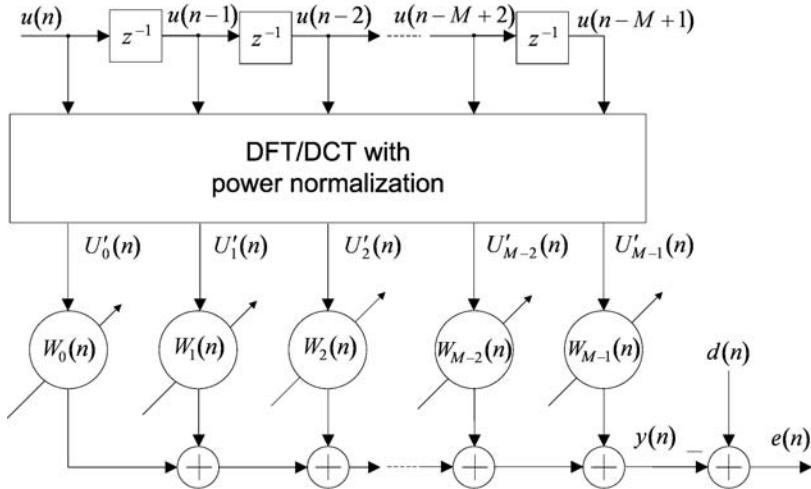


Figure 1.15 Block diagram of the self-orthogonalizing algorithm

Therefore, the extra computational cost over the conventional LMS algorithm is the additional transform power estimator given in Equation (1.29) and the normalization described in Equation (1.28).

The normalization defined in Equation (1.28) can be absorbed into the LMS algorithm to become the normalized LMS algorithm given in Equation (1.13). It is also noted that when DFT is used to transform the input signal, the DFT outputs are complex variables and the complex LMS algorithm must be used to update the weight vector as

$$W_k(n+1) = W_k(n) + \mu_k U_k^*(n) e(n), \quad k = 0, 1, \dots, M-1, \quad (1.30)$$

where $\mu_k = \mu / \sqrt{P_k(n) + \delta}$ is the normalized step size. In this case, the output of the algorithm is still real valued if the desired signal is real valued because the complex error signal $e(n)$ has a negligible imaginary part. The function M-files of the self-orthogonalizing adaptive filter algorithm are `SOAFinit.m` and `SOAFadapt.m`, and the script M-file `SOAFdemo.m` calls these MATLAB functions. Partial listing of these M-files is shown below.

The self-orthogonalizing adaptive filter algorithm

The self-orthogonalizing adaptive filter requires a transformed input vector. In this MATLAB example, we use a DCT-transform matrix `s.t` to transform the input data vector `u`. The transformed input `u` is applied to a set of parallel coefficients `w` and updates similar to the normalized LMS algorithm. The normalized step size

at each channel is based on the individual signal power `power_vec` at that frequency bin.

```

...
S = SOAFinit(zeros(M,1),mu,iter); % Initialization
S.unknownsys = b;
[yn,en,S] = SOAFadapt(un,dn,S); % Perform algorithm

function S = SOAFinit(w0,mu,leak)
    % Assign structure fields
    S.coeffs = w0(:); % Coefficients of FIR filter
    S.step = mu; % Step size
    S.leakage = leak; % Leaky factor
    S.iter = 0; % Iteration count
    S.AdaptStart = length(w0); % Running effects
    M = length(w0); % Length of filter
    S.beta = 1/M; % Forgetting factor
%
for j=1:M, % DCT-transform matrix
    S.T(1,j)=1/sqrt(M);
    for i=2:M,
        S.T(i,j)=sqrt(2/M)*cos(pi*(i-1)*(2*j-1)/2/M);
    end
end

function [yn,en,S] = SOAFadapt(un,dn,S)
...
for n = 1:ITER
    u = [un(n); u(1:end-1)]; % Input signal vector contains
                                % [u(n),u(n-1),...,u(n-M+1)]'
    U = T*u; % Transformed input vector
    yn(n) = w'*U; % Output signal
    power_vec= (1-S.beta)*power_vec+S.beta*(U.*U);
                                % Estimated power
    inv_sqrt_power = 1./sqrt(power_vec+(0.00001.*ones(M,1)));
    en(n) = dn(n) - yn(n); % Estimation error
    if ComputeEML == 1;
        eml(n) = norm(T*b-w)/norm_Tb;
                                % System error norm (normalized)
    end
    if n >= AdaptStart
        w = w + (mu*en(n)*inv_sqrt_power).*U;
                                % Tap-weight adaptation
        S.iter = S.iter + 1;
    end
end

```

The following section briefly introduces a different structure known as the subband adaptive filter, which closely resembles the self-orthogonalizing filter.

1.7.2 Subband adaptive filters

As stated in Section 1.6.1, the acoustic echo canceller in a typical room requires a high-order adaptive FIR filter to cover long echo tails. This is a disadvantage of using FIR filters for adaptive filtering because of the high computational load and slower convergence. Subband filtering structures have been proposed to overcome these problems [1, 3].

Figure 1.16 shows the block diagram of a general subband adaptive filter (SAF) that partitions the fullband input and desired signals into N subbands and decimates the subband signals from the original sampling rate f_s to the lower rate of f_s/N . The long M -tap filter (shown in Figure 1.3) is now replaced by N shorter M_S -tap FIR filters (where $M_S < M$) operating in parallel at a lower rate. The analysis filter bank consists of N parallel bandpass filters that are designed to partition the fullband signal into N overlapped frequency bands. The details of designing the analysis filter bank will be discussed in Chapters 2 and 5.

Also shown in Figure 1.16, a synthesis filter bank is used to combine the N subband output signals, $y_{i,D}(n)$, $i = 0, 1, \dots, N - 1$, into a fullband signal $y(n)$. The synthesis filter bank consists of a bank of interpolators that upsample the subband signals before filtering and adds these subband signals. Again, the details of designing synthesis filter banks will be discussed in Chapters 2 and 5. A synthesis filter bank is only required when fullband signals, such as the output signal $y(n)$ or error signal $e(n)$, are required. A detailed description of subband adaptive filters and their important properties will be given in Chapters 4, 5, 6 and 7.

A problem with subband adaptive filters is the introduction of substantial delay in the signal path caused by analysis and synthesis filter banks. Several modifications to the subband structures, known as delayless subband adaptive filters, are introduced in Chapter 4. A new multiband-structured SAF that has several useful properties as compared to conventional subband adaptive filters will be discussed in Chapter 6. Chapter 7 deals with the stability and performance analysis of SAF. Chapter 8 reports new research directions and recent works in the area of subband adaptive filtering.

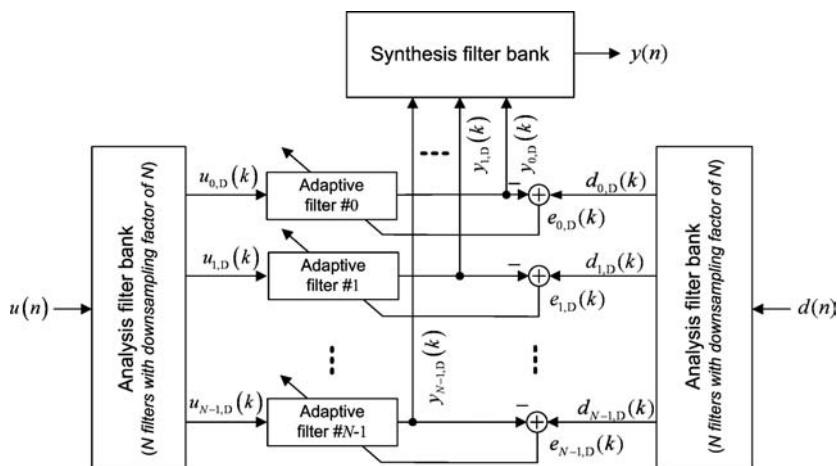


Figure 1.16 Block diagram of the subband adaptive filter

There are some similarities and differences between the SAF and the self-orthogonalizing adaptive filter shown in Figure 1.15. Both structures are designed to improve convergence over the FIR filter with the LMS algorithm by partitioning the fullband input signal $u(n)$ into a bank of parallel bandpass filters. The differences include: (i) the SAF involves the decimation operation to reduce the sampling rate for adaptive filtering, while there is no decimation in the self-orthogonalizing adaptive filter, (ii) N subband error signals are produced to update respective subband adaptive filters, but only a single error signal is used to update all the tap weights in the self-orthogonalizing adaptive filter, (iii) if the DFT is used in the self-orthogonalizing adaptive filter, the tap weights are complex numbers, while the SAF has real-valued tap weights, which leads to a simpler computational architecture, and (iv) the subband adaptive filter needs the synthesis filter bank to reconstruct the fullband output and error signals, which is not required in the self-orthogonalizing adaptive filter. These attributes of the subband adaptive filters and the comparison with other adaptive filters will be further explored in subsequent chapters.

1.8 Summary

This chapter introduced fundamental concepts of adaptive filtering. The basic block diagrams of filter structures, adaptive algorithms and applications were briefly discussed. Adaptive algorithms discussed were limited to the class that is based on minimizing the mean-square error, e.g. the LMS-type algorithms. A brief mention of another class of algorithms that is based on the deterministic framework of finding the least-square error over a period of past error samples was also given. In addition, important features of the transform-domain and subband adaptive filters were presented to illustrate how they differ from each other and the time-domain approaches. A more detailed description and analysis of these time-domain and transform-domain adaptive algorithms can be found in References [1] and [3].

References

- [1] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.
- [2] E. Hänsler and G. Schmidt, *Acoustic Echo and Noise Control: A Practical Approach*, New York: John Wiley & Sons, Inc., 2004.
- [3] S. Haykin, *Adaptive Filter Theory*, 4th edition, Upper Saddle River, New Jersey: Prentice Hall, 2002.
- [4] S. M. Kuo and D. R. Morgan, *Active Noise Control Systems: Algorithms and DSP Implementations*, New York: John Wiley & Sons, Inc., 1996.
- [5] J. R. Treichler, C. R. Johnson and M. G. Larimore, *Theory and Design of Adaptive Filters*, Upper Saddle River, New Jersey: Prentice Hall, 2001.
- [6] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Upper Saddle River, New Jersey: Prentice Hall, 1985.
- [7] B. Widrow and E. Walach, *Adaptive Inverse Control*, Upper Saddle River, New Jersey: Prentice Hall, 1996.

- [8] S. Haykin and B. Widrow, *Least-Mean-Square Adaptive Filters*, New York: John Wiley & Sons, Inc., 2003.
- [9] A. H. Sayed, *Fundamentals of Adaptive Filtering*, New York: John Wiley & Sons, Inc., 2003.
- [10] M. Montazeri and P. Duhamel, ‘A set of algorithms linking NLMS and block RLS algorithms’, *IEEE Trans. Signal Processing*, **43**(2), February 1995, 444–453.
- [11] S. M. Kuo, B. H. Lee and W. Tian, *Real-Time Digital Signal Processing: Implementations and Application*, 2nd edition, Chichester, West Sussex: John Wiley & Sons, Ltd, 2006.
- [12] W. S. Gan and S. M. Kuo, *Embedded Signal Processing with the Micro Signal Architecture*, Hoboken, New Jersey: John Wiley & Sons, Inc., 2007.
- [13] S. M. Kuo and W. S. Gan, *Digital Signal Processors: Algorithms, Implementations, and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 2005.
- [14] D. Mansour and A. H. Gray, ‘Unconstrained frequency-domain adaptive filters’, *IEEE Trans. Acoust. Speech Signal Processing*, **30**(3), October 1982, 726–734.

2

Subband decomposition and multirate systems

In some signal processing applications, such as subband adaptive filtering [1–11] and subband coding [12–16], it is advantageous to partition the input signal into a set of subband signals prior to specific processing. Decomposition of a fullband signal into multiple subbands facilitates the manipulation of information contained in each subband. Because each subband signal occupies only a small portion of the original frequency band, it can be critically decimated in order to preserve the total number of samples to be processed and maintain the effective sampling rate. Therefore, subband and multirate techniques [12, 13, 15, 17–22] are always employed together in designing computationally efficient and effective signal processing systems.

This chapter reviews fundamental concepts, introduces design techniques and defines various formulations for multirate filter banks. In Section 2.1, the basic theory of multirate signal processing is presented. These concepts are used in Section 2.2 for the analysis of multirate filter banks. Section 2.3 introduces paraunitary filter banks, a special class of filter banks that are able to reconstruct perfectly the fullband signal after subband decomposition. Section 2.4 addresses the underlying affinity between filter banks and block transforms. Finally, design techniques for cosine-modulated filter banks and discrete Fourier transform filter banks (which will be used extensively in subsequent chapters) are discussed in Sections 2.5 and 2.6, respectively.

2.1 Multirate systems

Digital signal processing systems that use more than one sampling rate are referred to as multirate systems [12, 13, 15, 19, 21]. The decimator and interpolator are two basic sampling rate conversion (or alteration) devices employed in a multirate system to obtain different sampling rates used at different stages of the system.

A decimator, as depicted in Figure 2.1(a), retains only those samples of $x(n)$ that occur at time instants equal to multiples of D . The decimator output can be expressed as

$$x_D(k) = x(kD), \quad (2.1)$$

where D is the decimation factor. The variable n is used for the time index of the original sequence $x(n)$ and k is used for the decimated sequences $x_D(k)$. The sampling rate of the decimated sequence is D times lower than that of the input sequence $x(n)$. The input–output relation for a D -fold decimator can be written in the z -domain as [15, 19, 21]

$$X_D(z) = \frac{1}{D} \sum_{l=0}^{D-1} X(z^{1/D} W_D^l), \quad (2.2)$$

where $W_D \equiv e^{-j2\pi/D}$ is the D th root of unity and $j \equiv \sqrt{-1}$.

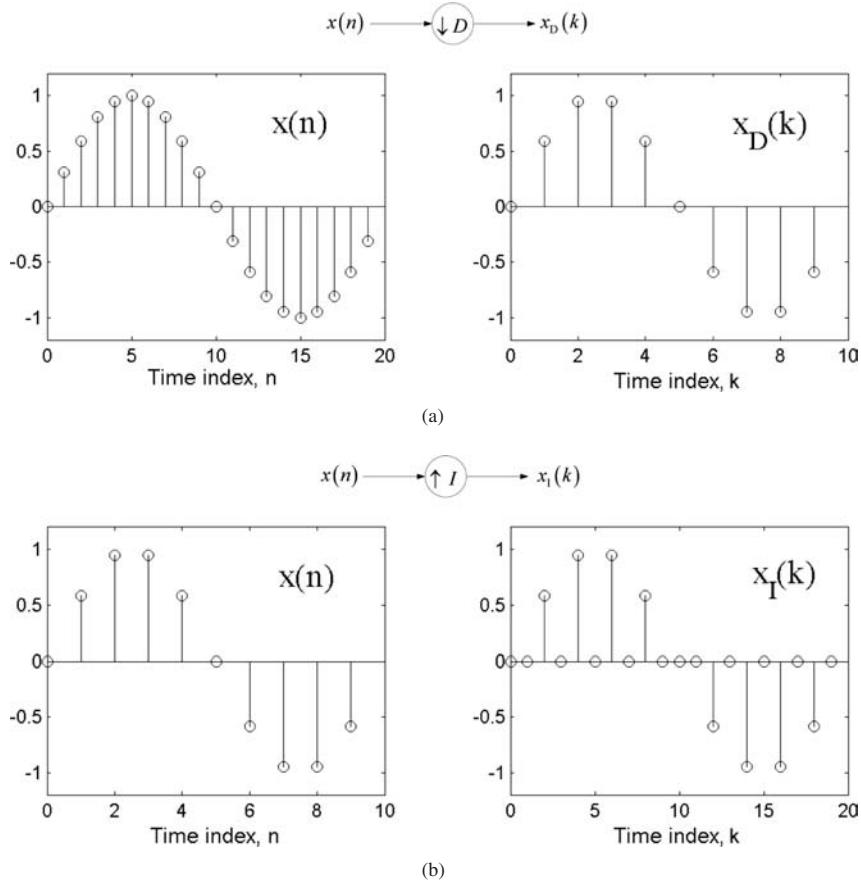


Figure 2.1 Sampling rate conversion devices: (a) decimation by a factor D and (b) interpolation by a factor I

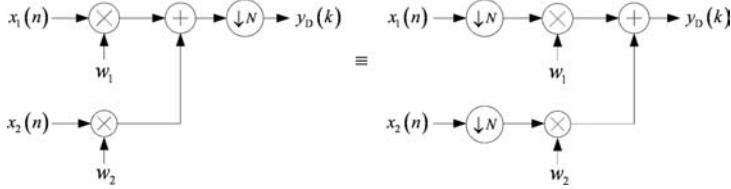


Figure 2.2 Linearity of the decimator. Any memoryless operation commutes with the decimator

On the other hand, the interpolator depicted in Figure 2.1(b) increases the sampling rate of input signal $x(n)$ by inserting $(I - 1)$ zero samples between each adjacent pair of input samples according to the relation

$$x_I(k) = \begin{cases} x(k/I), & k = 0, \pm I, \pm 2I, \dots \\ 0, & \text{otherwise} \end{cases} \quad (2.3)$$

where I is the interpolation factor and k is the time index of the interpolated sequence $x_I(k)$. The input and output of the I -fold interpolator are related in the z -domain as [15, 19, 21]

$$X_I(z) = X(z^I). \quad (2.4)$$

The decimation and interpolation operations are demonstrated by Example 2.1 and Example 2.2, respectively. Complete M-files are given as `Example_2P1.m` and `Example_2P2.m` in the companion CD.

Decimation and interpolation are linear but shift-variant operations [13, 15, 19, 21]. The linearity of the decimators and interpolators ensures that any memoryless operation, such as addition or multiplication by a constant, is commutable with a decimator (or an interpolator), as illustrated in Figure 2.2. On the other hand, the cascade of a filter (i.e. a dynamic system with memory) and a sampling rate conversion device is not commutative due to the shift-variant property. Hence, it makes a great difference whether filtering occurs before or after the sampling rate conversion.

Example 2.1 (a complete listing is given in M-file `Example_2P1.m`)

Consider a 50 Hz sinusoidal signal $x(n)$ with a sampling rate of 1000 Hz. The sinewave before and after decimation (with a decimation factor of $D = 2$) are shown in Figure 2.1(a). Notice that the sampling period doubled due to the decimation operation. The decimator can be implemented using the following MATLAB script:

```
fs = 1000; % Sampling rate
freq = 50; % Sinewave frequency
N = 100; % Data length
```

```

n = 0:N-1; % Time index
x = sin(2*pi*(freq/fs)*n); % Input sequence
D = 2; % Decimation factor
x_D = x(1:D:end); % Generate the decimated sequence

```

Example 2.2 (a complete listing is given in M-file `Example_2P2.m`)

Consider a 50 Hz sinusoidal signal with a 500 Hz sampling rate instead of 1000 Hz as in Example 2.1. An interpolator can be used to increase the sampling rate by inserting zero samples as shown in the following MATLAB script. Figure 2.1(b) shows the input and output of the interpolator with an interpolation factor of $I = 2$.

```

fs = 500; % Sampling rate
freq = 50; % Sinewave frequency
N = 100; % Data length
n = 0:N-1; % Time index
x = sin(2*pi*(freq/fs)*n); % Input sequence
I = 2; % Interpolation factor
x_I = zeros(1, I*length(x)); % Initialize the buffer
x_I(1:I:end) = x; % Generate the interpolated sequence

```

2.2 Filter banks

A filter bank is a set of bandpass filters with a common input for the analysis filter bank or a summed output for the synthesis filter bank [12, 13, 15, 20, 21]. Figure 2.3 shows an N -channel (or N -band) filter bank using z -domain notation, where $H_i(z)$ and $F_i(z)$ are the analysis and synthesis filters, respectively, and the variable $i = 0, 1, \dots, N - 1$ is used as the subband index. The analysis filter bank partitions the incoming signal $X(z)$ into N subband signals $X_i(z)$, each occupying a portion of the original frequency band. The synthesis filter bank reconstructs the output signal $Y(z)$ from N subband signals $Y_i(z)$ to approximate the input signal.

A filter bank is called a uniform filter bank if the center frequencies of bandpass filters are uniformly spaced and all filters have equal bandwidth. By decomposing a fullband signal using an N -channel uniform filter bank, each subband signal $X_i(z)$ contains only $1/N$ of the original spectral band. Since the bandwidth of the subband signals $X_i(z)$ is $1/N$ of the original signal $X(z)$, these subband signals can be decimated to $1/N$ of the original sampling rate while preserving the original information. A filter bank is called a critically (or maximally) decimated filter bank [12, 13, 15, 20, 21] if the decimation factor is equal to the number of subbands, i.e. $D = N$. Critical decimation preserves the effective sampling rate with N decimated subband signals, $X_{i,D}(z)$, each with $1/N$ of the original sampling rate, so that the total number of subband samples is identical to that of the fullband signal $X(z)$. In the synthesis section, the decimated subband signals

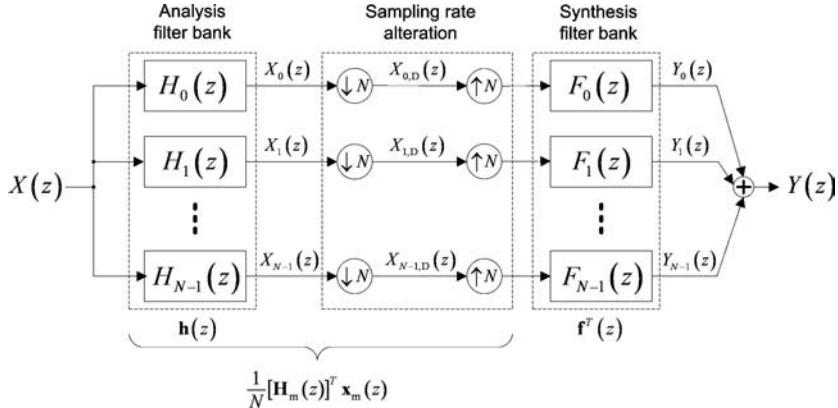


Figure 2.3 An N -channel critically decimated filter bank. The combined structure of the analysis–synthesis system is often referred to as a quadrature-mirror filter bank

$X_{i,D}(z)$ are interpolated by the same factor before being combined by the synthesis filter bank. Therefore, the original sampling rate is restored in the reconstructed fullband signal $Y(z)$. The MATLAB script given in Example 2.3 demonstrates the analysis, sampling rate conversion and synthesis operations of a critically decimated filter bank.

Example 2.3 (a complete listing is given in M-file `Example_2P3.m`)

We design a cosine-modulated filter bank (which will be introduced in Section 2.5) in this example. The coefficients of the analysis and synthesis filters are stored in the matrices \mathbf{H} and \mathbf{F} , respectively. The analysis filter bank \mathbf{H} and subsequent decimation operation is performed using the `upfirdn` function. The decimated subband signals are interpolated and combined through the synthesis filter bank \mathbf{F} . The `opt_filter` function returns a prototype filter, which is used to construct the analysis and synthesis filter banks via the `make_bank` function.

```

N = 4; % Number of subbands
K = 8; % Overlapping factor
L = 2*K*N; % Length of analysis filters

[hopt,passedge] = opt_filter(L-1,N); % Create lowpass
% prototype filter
[H,F] = make_bank(hopt,N); % Generate filter banks
H = sqrt(N)*H'; F = sqrt(N)*F'; % Scaling

xn = sin(2*pi*0.01*(0:1000)); % Input sinusoidal signal
xD = upfirdn(xn,H,1,N); % Analysis and decimation
yn = sum(upfirdn(xD,F,N),2); % Interpolation and synthesis

```

In practice, realizable filters have transition bands and finite stopband attenuation. Therefore, critical decimation results in aliasing. However, the aliasing components can

be cancelled by properly designed synthesis filters, resulting in an aliasing-free filter bank. The combined structure of analysis and synthesis filter banks with aliasing-free properties is generally referred to as a quadrature-mirror filter (QMF) bank [15, 21, 24–26].

2.2.1 Input–output relation

Consider the N -channel filter bank shown in Figure 2.3. The subband signals can be expressed as $X_i(z) = H_i(z)X(z)$. Using Equation (2.2), the decimated subband signals can be written as

$$X_{i,D}(z) = \frac{1}{N} \sum_{l=0}^{N-1} H_i(z^{1/N} W_N^l) X(z^{1/N} W_N^l), \quad i = 0, 1, \dots, N-1. \quad (2.5)$$

These decimated subband signals $X_{i,D}(z)$ are interpolated by the same factor N before being combined by the synthesis filter bank to form the fullband output $Y(z)$. According to Equations (2.4) and (2.5), the output of the synthesis filter $F_i(z)$ is given by

$$\begin{aligned} Y_i(z) &= F_i(z) \left[\frac{1}{N} \sum_{l=0}^{N-1} H_i(z W_N^l) X(z W_N^l) \right] \\ &= \frac{F_i(z)}{N} \left[H_i(z)X(z) + \sum_{l=1}^{N-1} H_i(z W_N^l) X(z W_N^l) \right] \\ &= \frac{F_i(z)}{N} \left[X_i(z) + \sum_{l=1}^{N-1} X_i(z W_N^l) \right]. \end{aligned} \quad (2.6)$$

Clearly, $Y_i(z)$ consists of the desired component $X_i(z) = H_i(z)X(z)$ and its frequency shifted versions, $X_i(z W_N^l) = H_i(z W_N^l) X(z W_N^l)$ for $l = 1, 2, \dots, N-1$. The term $X_i(z W_N^l)$ is called the l th aliasing component, which comes from the sampling rate conversion (i.e. the decimation and interpolation operations as illustrated in Figure 2.3) of subband signals $X_i(z)$. It can be noted from Equation (2.6) that the terms, $X(z W_N^l)$, $l = 0, 1, \dots, N-1$, are common to all the synthesis filter outputs $Y_0(z), Y_1(z), \dots, Y_{N-1}(z)$. By grouping these common terms together, the input–output relation of the N -channel QMF bank shown in Figure 2.3 can be written as

$$\begin{aligned} Y(z) &= \sum_{i=0}^{N-1} Y_i(z) \\ &= \sum_{i=0}^{N-1} \left[\frac{1}{N} \sum_{l=0}^{N-1} F_i(z) H_i(z W_N^l) X(z W_N^l) \right] \\ &= \sum_{l=0}^{N-1} \left[\frac{1}{N} \sum_{i=0}^{N-1} F_i(z) H_i(z W_N^l) \right] X(z W_N^l) \\ &= \sum_{l=0}^{N-1} A_l(z) X(z W_N^l), \end{aligned} \quad (2.7)$$

where

$$A_l(z) \equiv \frac{1}{N} \sum_{i=0}^{N-1} F_i(z) H_i(z W_N^l). \quad (2.8)$$

The transfer function $A_l(z)$ can be considered as the gain of the l th aliasing component $X(z W_N^l)$ at the output $Y(z)$. Equation (2.7) signifies that the QMF bank is a time-varying system [15, p. 226].

The input–output relation given in Equation (2.7) can also be written in a more compact vector form as

$$Y(z) = \frac{1}{N} \mathbf{f}^T(z) \mathbf{H}_m^T(z) \mathbf{x}_m(z), \quad (2.9)$$

where

$$\mathbf{f}^T(z) \equiv [F_0(z), F_1(z), \dots, F_{N-1}(z)], \quad (2.10)$$

$$\mathbf{H}_m(z) \equiv [\mathbf{h}(z), \mathbf{h}(z W_N), \dots, \mathbf{h}(z W_N^{N-1})]^T, \quad (2.11)$$

$$\mathbf{x}_m(z) \equiv [X(z), X(z W_N), \dots, X(z W_N^{N-1})]^T \quad (2.12)$$

and

$$\mathbf{h}(z) \equiv [H_0(z), H_1(z), \dots, H_{N-1}(z)]^T. \quad (2.13)$$

In the above equations, $\mathbf{h}(z)$ denotes the analysis filter bank, $\mathbf{f}^T(z)$ denotes the synthesis filter bank, $\mathbf{H}_m(z)$ is the modulation matrix of the analysis filter bank and $\mathbf{x}_m(z)$ is the modulation vector of the input signal $X(z)$. Notice that the analysis filter bank is a one-input, N -output system with the transfer vector $\mathbf{h}(z)$, whereas the synthesis filter bank is an N -input, one-output system with the transfer vector $\mathbf{f}^T(z)$. The number of elements in a column represents the number of outputs, whereas the number of elements in a row represents the number of inputs.

2.2.2 Perfect reconstruction filter banks

Equation (2.7) shows that the output $Y(z)$ of a QMF bank is a weighted sum of its input $X(z)$ and the uniformly shifted versions $X(z W_N^l)$. With the proper design of analysis and synthesis filters such that $A_l(z) = 0$ for $l = 1, 2, \dots, N - 1$, the common aliasing components from all the N subbands will cancel each other out. Therefore, the QMF bank can be referred to as aliasing-free and the filter bank becomes a linear time-invariant system [15, p. 228] with its input–output relation given by

$$\begin{aligned} Y(z) &= \left[\frac{1}{N} \sum_{i=0}^{N-1} F_i(z) H_i(z) \right] X(z) \\ &= T(z) X(z), \end{aligned} \quad (2.14)$$

where

$$T(z) = A_0(z) = \frac{1}{N} \sum_{i=0}^{N-1} F_i(z) H_i(z) \quad (2.15)$$

is known as the distortion transfer function of the filter bank. $T(z)$ is a measure of any possible magnitude and phase distortion of the filter bank output. If $T(z)$ has constant magnitude and linear phase at all frequencies (i.e. $T(z) = cz^{-\Delta}$), we achieve the so-called perfect-reconstruction (i.e. magnitude and phase preservation in addition to alias-free) filter bank, where the filter output is a time-delayed version of the scaled input (i.e. $Y(z) = cX(z)z^{-\Delta}$).

As pointed out earlier, a QMF bank is alias-free if the aliasing components from all the subbands cancel each other out when the interpolated subband signals $Y_i(z)$ are combined to form the fullband output $Y(z)$. Nevertheless, it should be noted that aliasing does exist in the decimated subband signals $X_{i,D}(z)$ given in Equation (2.5), even if the QMF bank is alias-free. Aliasing can be eliminated if the analysis filters are ideal filters (with a zero-transition band and infinite stopband attenuation). This ideal filter cannot be realized for real-world applications. However, it would be practical to assume that the stopband attenuation of the filters is sufficiently high so that few aliasing components $H_i(z)W_N^l X(z)W_N^l$ that overlap with $H_i(z)X(z)$ are significant.

2.2.3 Polyphase representation

An analysis filter bank can be represented by an $N \times 1$ vector $\mathbf{h}(z)$ as shown in Section 2.2.1. The set of N analysis filters $H_i(z)$ in $\mathbf{h}(z)$ can be alternatively expressed by an N -band polyphase decomposition in the form

$$H_i(z) = \sum_{r=0}^{N-1} E_{i,r}(z^N) z^{-r}, \quad i = 0, 1, \dots, N-1, \quad (2.16)$$

where $E_{i,r}(z)$ is the r th polyphase component of the i th analysis filter $H_i(z)$ expressed as

$$E_{i,r}(z) \equiv \sum_{n=0}^{K-1} h_i(nN + r) z^{-n}. \quad (2.17)$$

In the above equations, the analysis filter $H_i(z)$ is assumed to be causal with the impulse response $h_i(n)$ of length $L = KN$, where K denotes the length of the polyphase components. An equivalent matrix representation of the set of N equations in (2.16) is given by

$$\mathbf{h}(z) = \mathbf{E}(z^N) \mathbf{e}(z), \quad (2.18)$$

where the $N \times N$ matrix

$$\mathbf{E}(z) \equiv \begin{bmatrix} E_{0,0}(z) & E_{0,1}(z) & \cdots & E_{0,N-1}(z) \\ E_{1,0}(z) & E_{1,1}(z) & \cdots & E_{1,N-1}(z) \\ \vdots & \vdots & \ddots & \vdots \\ E_{N-1,0}(z) & E_{N-1,1}(z) & \cdots & E_{N-1,N-1}(z) \end{bmatrix} \quad (2.19)$$

is called the type-I polyphase component matrix for the analysis filter bank $\mathbf{h}(z)$ and

$$\mathbf{e}(z) \equiv [1, z^{-1}, \dots, z^{-N+1}]^T \quad (2.20)$$

represents the delay chain as depicted in Figure 2.4. Notice that the polyphase terms defined in Equations (2.17) and (2.19) are used in their interpolated forms in Equations (2.16) and (2.18) to represent the analysis filter $H_i(z)$ and the filter bank $\mathbf{h}(z)$, respectively. The idea is illustrated in Figure 2.4, where input samples are delivered to each of the components $E_{i,r}(z^N)$ through a tapped-delay line.

Likewise, the N synthesis filters $\mathbf{f}^T(z)$ can also be represented in polyphase form as

$$F_i(z) = \sum_{r=0}^{N-1} R_{r,i}(z^N) z^{-(N-1-r)}, \quad i = 0, 1, \dots, N-1, \quad (2.21)$$

where $R_{r,i}(z)$ are the type-II polyphase components of the filter $F_i(z)$. For a given transfer function $H(z) = \sum_{r=0}^{N-1} E_r(z^N) z^{-r}$, represented by N polyphase components $E_r(z)$, its type-II polyphase components are permutations of $E_r(z)$ in the form of $R_r(z) = E_{N-1-r}(z)$. In matrix notation, the set of equations expressed in Equation (2.21) can be written as

$$\mathbf{f}^T(z) = z^{-N+1} \tilde{\mathbf{e}}(z) \mathbf{R}(z^N), \quad (2.22)$$

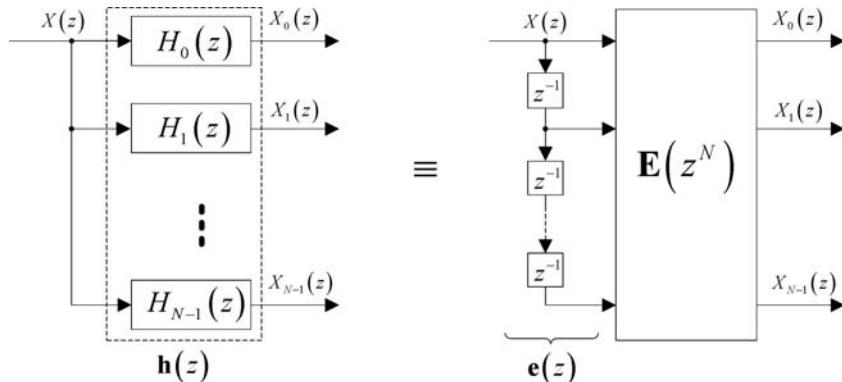


Figure 2.4 Type-I polyphase representation of the analysis filter bank

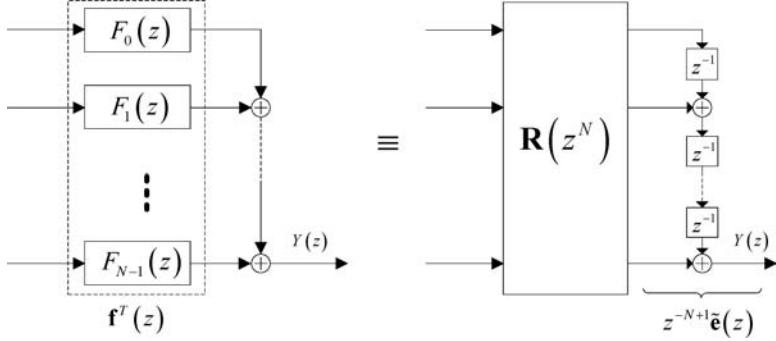


Figure 2.5 Type-II polyphase representation of the synthesis filter bank

where

$$\mathbf{R}(z) \equiv \begin{bmatrix} R_{0,0}(z) & R_{0,1}(z) & \cdots & R_{0,N-1}(z) \\ R_{1,0}(z) & R_{1,1}(z) & \cdots & R_{1,N-1}(z) \\ \vdots & \vdots & \ddots & \vdots \\ R_{N-1,0}(z) & R_{N-1,1}(z) & \cdots & R_{N-1,N-1}(z) \end{bmatrix} \quad (2.23)$$

is referred to as the type-II polyphase component matrix for the synthesis filter bank $\mathbf{f}(z)$ and

$$\begin{aligned} z^{-N+1}\tilde{\mathbf{e}}(z) &= z^{-N+1}[1, z^1, \dots, z^{N-1}] \\ &= [z^{-N+1}, z^{-N+2}, \dots, 1] \end{aligned} \quad (2.24)$$

is the delay chain pertaining to the type-II polyphase representation, as depicted in Figure 2.5. In the above equations, $\tilde{\mathbf{e}}(z)$ is the paraconjugate of $\mathbf{e}(z)$ (which will be introduced in Section 2.3). It should be noted that each row of the matrix $\mathbf{E}(z)$ corresponds to an analysis filter, whereas each column of the matrix $\mathbf{R}(z)$ defines the polyphase components of a synthesis filter.

Figure 2.6(a) shows the polyphase realization of an analysis filter followed by a decimator. The decimator can be moved into the branches by exploiting the linearity property (i.e. the addition and decimation operations are commutative, as explained in Section 2.1). Furthermore, by means of the noble identities [13, 15], the decimators can be moved across the polyphase subfilters, leading to the equivalent structure in Figure 2.6(b). The cascade of an interpolator and a synthesis filter can also be implemented in polyphase form, as illustrated in Figure 2.7(a). Likewise, the equivalent structure depicted in Figure 2.7(b) can be obtained by exploiting the noble identities and the linearity property.

Substituting the polyphase representations of Figures 2.4 and 2.5 into Figure 2.3, the equivalent N -channel QMF bank can be represented in polyphase form as depicted in Figure 2.8(a). Next, by exploiting the linearity of the decimator and interpolator together with the noble identities (as outlined in Figures 2.6 and 2.7 for the analysis and synthesis filters, respectively), we derive a simplified structure as illustrated in Figure 2.8(b). The

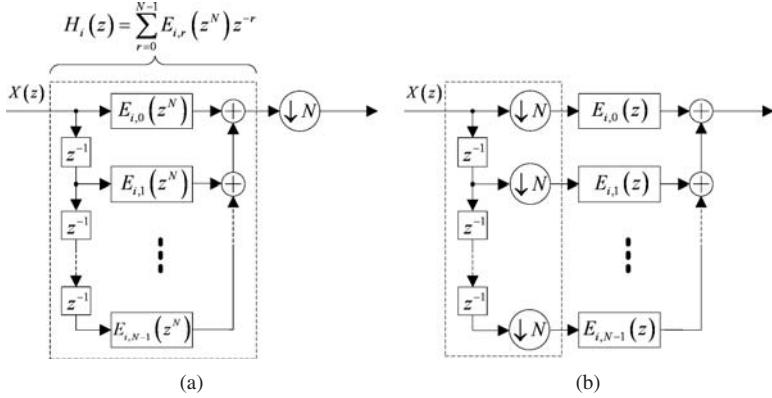


Figure 2.6 An analysis filter $H_i(z)$ with a decimator: (a) type-I polyphase realization of the analysis filter followed by the decimator and (b) decimators moved into the branches before the polyphase subfilters

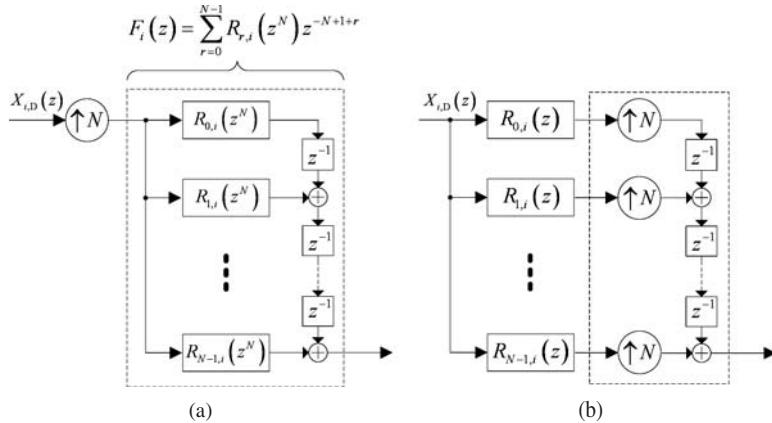


Figure 2.7 A synthesis filter $F_i(z)$ with an interpolator: (a) type-II polyphase realization of the synthesis filter preceded by the interpolator and (b) interpolators moved into the branches after the polyphase subfilters

set of delays and decimators on the left side of Figure 2.8(b) can be regarded as the polyphase representation of a serial-to-parallel converter, which gathers a block of N samples from a given signal $x(n)$ as follows:

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

On the other hand, the set of interpolators and delays on the right side of Figure 2.8(b) is the multirate representation of a parallel-to-serial converter, which appends blocks of N samples to the output. Thus, for the case where $\mathbf{R}(z)\mathbf{E}(z) = \mathbf{I}$, we obtain a trivial perfect-reconstruction filter bank, as depicted in Figure 2.9. The filter bank processes a

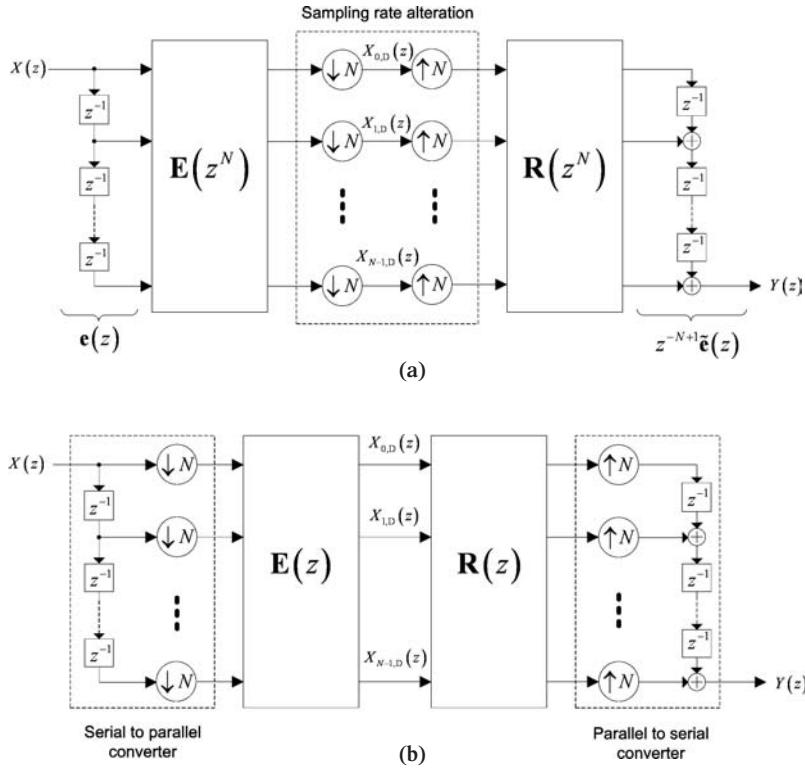


Figure 2.8 N -channel maximally decimated filter bank: (a) polyphase representation and (b) rearrangement using noble identities

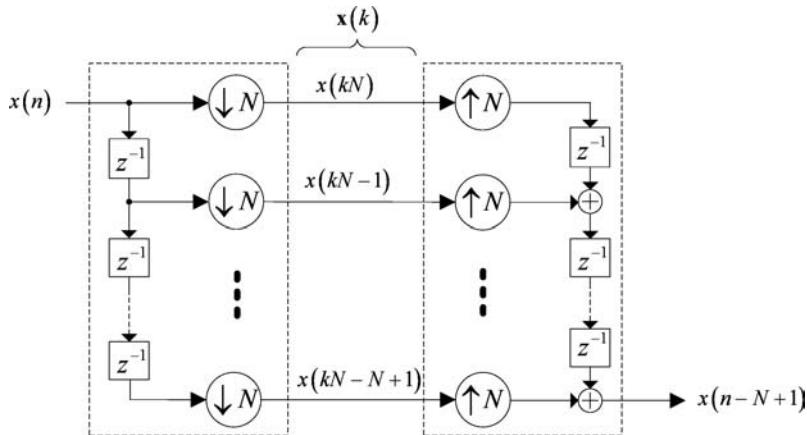


Figure 2.9 A trivial perfect-reconstruction filter bank

block of input samples in $\mathbf{x}(k)$ and produces a delayed version of the output, $y(n) = x(n - N + 1)$. The concept of polyphase realization is demonstrated in Example 2.4.

Example 2.4 In this example, we design a cosine-modulated filter bank using the `opt_filter` and `make_bank` functions (the procedure will be detailed in Section 2.5). The coefficients of the resulting analysis and synthesis filters are stored in the matrices `H` and `F`, respectively. The polyphase implementations of the analysis and synthesis filters (in the matrices `E` and `R`, respectively) are shown in the following script. A complete listing of the M-file is given as `Example_2P4.m` in the companion CD.

```
[hopt,passedge] = opt_filter(L-1,N); % Lowpass prototype filter
[H,F] = make_bank(hopt,N); % Generate filter banks
H = sqrt(N)*H'; F = sqrt(N)*F'; % Scaling

xn = sin(2*pi*0.01*(0:1000)); % Input sinusoidal signal
ITER = length(xn);
x = zeros(N,1); % Input tapped-delay line
X = zeros(N,L/N); % Polyphase buffers (analysis)
Y = zeros(L/N,N); % Polyphase buffers (synthesis)
yn = zeros(fix(ITER/N)*N,1);

for i = 1:N
    % Type-I polyphase decomposition. Polyphase components of the
    % ith filter is represented by rows in the matrix
    E(:,:,i) = reshape(H(:,:,i),N,L/N);
    % Type-II polyphase decomposition. Type-I polyphase
    % decomposition followed by a permutation
    R(:,:,i) = flipud(reshape(F(:,:,i),N,L/N));
end

for n = 1:ITER
    x = [xn(n); x(1:end-1)];
    if mod(n,N) == 0
        %--Analysis section-----
        X = [x, X(:,1:end-1)];
        for i = 1:N
            xD(1,i) = sum(sum(E(:,:,i).*X));
        end
        %--Synthesis section-----
        Y = [xD; Y(1:end-1,:)];
        y = 0;
        for i = 1:N
            y = y + R(:,:,i)*Y(:,i);
        end
        yn(n:-1:n-N+1) = y;
    end
end
```

2.3 Paraunitary filter banks

In Figure 2.8, the analysis and synthesis filters of the N -channel critically decimated filter bank are expressed in terms of the polyphase component matrices $\mathbf{E}(z)$ and $\mathbf{R}(z)$. A perfect-reconstruction system can be obtained by first imposing the paraunitary property on the analysis filter bank $\mathbf{E}(z)$ such that

$$\tilde{\mathbf{E}}(z)\mathbf{E}(z) = \mathbf{I}, \quad (2.25)$$

where \mathbf{I} is the identity matrix and $\tilde{\mathbf{E}}(z) = \mathbf{E}^{*T}(z^{-1})$ is the paraconjugate of the matrix $\mathbf{E}(z)$. To obtain the paraconjugate of $\mathbf{E}(z)$, we first conjugate the coefficients and replace z with z^{-1} for each element of $\mathbf{E}(z)$, i.e. $\tilde{E}_{i,r}(z) = E_{i,r}^*(z^{-1})$. Transposing the resulting matrix gives $\tilde{\mathbf{E}}(z) = [E_{r,i}^*(z^{-1})]$ for $i, r = 0, 1, \dots, N - 1$. The synthesis filters can be obtained as the time reversal of the analysis filters in the form

$$\mathbf{R}(z) = z^{-K+1}\tilde{\mathbf{E}}(z). \quad (2.26)$$

In the above equations, $K - 1$ is the order of the polyphase component matrices. Again, we assume that the analysis and synthesis filters are FIR filters of length $L = KN$.

The paraunitary condition defined in Equation (2.25) implies that $\mathbf{E}^{-1}(z) = \tilde{\mathbf{E}}(z)$. The polyphase components of the synthesis filter bank can be obtained according to Equation (2.26) as $R_{r,i}(z) = z^{-K+1}\tilde{E}_{i,r}(z)$. Since the order of each polyphase component is $K - 1$, the delay z^{-K+1} ensures that the resulting $\mathbf{R}(z)$ is causal. The transfer vector of the synthesis filter bank $\mathbf{f}^T(z)$ can then be obtained from $\mathbf{R}(z)$ by substituting Equation (2.26) into Equation (2.22) in the form

$$\begin{aligned} \mathbf{f}^T(z) &= z^{-N+1}\tilde{\mathbf{e}}(z)[z^{(-K+1)N}\tilde{\mathbf{E}}(z^N)] \\ &= z^{-KN+1}\tilde{\mathbf{e}}(z)\tilde{\mathbf{E}}(z^N) \\ &= z^{-L+1}\tilde{\mathbf{h}}(z), \end{aligned} \quad (2.27)$$

where $\tilde{\mathbf{h}}(z) = \tilde{\mathbf{e}}(z)\tilde{\mathbf{E}}(z^N)$ is the paraconjugate of $\mathbf{h}(z) = \mathbf{E}(z^N)\mathbf{e}(z)$. From Equations (2.10) and (2.27), the transfer functions of the synthesis filters are given by

$$F_i(z) = z^{-L+1}\tilde{H}_i(z), \quad i = 0, 1, \dots, N - 1, \quad (2.28)$$

or, equivalently, expressed in the time domain as

$$f_i(n) = h_i(L - 1 - n), \quad i = 0, 1, \dots, N - 1, \quad (2.29)$$

where the filter coefficients are assumed to be real valued; i.e. the impulse response of the synthesis filter is equal to the time-reversed impulse response of the corresponding analysis filter.

Equations (2.25) and (2.26) define a sufficient condition for perfect reconstruction [15, 21]. By imposing the paraunitary condition given in Equation (2.25) on $\mathbf{E}(z)$ and then setting $\mathbf{R}(z) = z^{-K+1}\tilde{\mathbf{E}}(z)$, the product of matrices, $\mathbf{R}(z)\mathbf{E}(z)$, reduces to z^{-K+1} . Therefore, the cascade of $\mathbf{E}(z)$ and $\mathbf{R}(z)$ (as shown in Figure 2.8(b)) results in a net

delay of $K - 1$ at the decimated rate. Considering the delay inflicted by the trivial perfect-reconstruction system (which consists of the delay chains and sampling rate conversion devices, as illustrated in Figure 2.9), the QMF bank introduces an overall delay of $\Delta = N - 1 + (K - 1)N = KN - 1 = L - 1$ sampling periods on the input signal. To this end, the aliasing has been cancelled and the distortion transfer function $T(z)$ defined in Equation (2.15) has been forced to a pure delay, $z^{-\Delta}$.

2.4 Block transforms

A block transform, such as the discrete Fourier transform (DFT) or the discrete cosine transform (DCT), can be regarded as a critically decimated paraunitary filter bank with a zero-order polyphase component matrix, i.e. $\mathbf{E}(z) = \mathbf{A}^H$, where \mathbf{A} denotes a unitary (or orthogonal) transformation matrix and H is a Hermitian transposition [13, 19, 27–29]. Notice that any unitary matrix is (trivially) paraunitary [15, p. 289]. The basis functions of the transform (i.e. the columns of the matrix \mathbf{A}) are the impulse responses of bandpass filters with length L equal to the block size N . Alternatively, we may interpret a filter bank as a block transform with memory [13], where the length L of bandpass filters is larger than the block size N , i.e. $L > N$.

The main differences between a block transform and a filter bank are the degree of subband separation (or decorrelation) that can be achieved and the computational complexity. In some applications, we may need the subband signals (or the transformed parameters) to be uncorrelated. Strong subband separation can generally be achieved by using filter banks with high-order analysis filters. Thus, increasing the filter length reduces the correlation between subband signals with the penalty of higher computational complexity.

Block transforms and filter banks are widely employed in adaptive filtering. Examples include transform-domain adaptive filters (TDAF) [27, 28, 30–34], subband adaptive filters (SAF) [2, 4, 5, 11, 29, 35, 36] and filter-bank adaptive filters (FBAF) [37]. From the underlying affinity between block transforms and filter banks, we readily see that these adaptive filters are closely related to each other. A brief description and comparison of these adaptive filters is summarized in Table 2.1. This table also includes a brief description of the multiband-structured SAF (MSAF) [1, 6, 7, 10, 38–41], which will be discussed in Chapter 6.

2.4.1 Filter bank as a block transform

An analysis filter bank can be treated as a block transform with memory, whereas the synthesis filter bank can be regarded as the corresponding inverse transform [13, 42]. The idea is illustrated in Figure 2.10, where the analysis and synthesis filter banks are, respectively, represented by $L \times N$ matrices $\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{N-1}]$ and $\mathbf{F} = [\mathbf{f}_0, \mathbf{f}_1, \dots, \mathbf{f}_{N-1}]$. The columns of the matrices, $\mathbf{h}_i = [h_i(0), h_i(1), \dots, h_i(L-1)]^T$ and $\mathbf{f}_i = [f_i(0), f_i(1), \dots, f_i(L-1)]^T$, hold the coefficients of the analysis and synthesis filters, respectively. Recall that for a paraunitary filter bank, the synthesis filters are the time-reversed versions of the corresponding analysis filters.

Table 2.1 Subband and transform-domain adaptive filters (part 1 of 2)

Adaptive filters	Adaptive tap weights	Features	Objectives	Remarks
Transform domain adaptive filter (TDAF)	An $N \times N$ orthogonal matrix \mathbf{A} is applied to the input signal vector. The orthogonal matrix \mathbf{A} is used to diagonalize the input autocorrelation matrix $R = E\{\mathbf{u}(n)\mathbf{u}^T(n)\}$ of the input signal $\mathbf{u}(n)$. The transformation is effective if [27, 28] $\mathbf{A}^T \mathbf{R} \mathbf{A} \approx \text{a diagonal matrix.}$	Here, it is assumed that the transformation matrix \mathbf{A} is real-valued and the columns of the matrix represent the basis functions of the transform [42, pp 11]. Sample based delayless structure. No inverse transform is necessary to obtain the output signal. Convergence improvement is achieved by diagonalizing (i.e., rotating the correlation matrix to the principal axes) and normalizing (i.e., scaling) the correlation matrix as close as possible to the identity matrix.	<ul style="list-style-type: none"> To improve the convergence rate of gradient-based algorithms when the input signal is colored. There is an increment in the computational load due to the transformation. 	<ul style="list-style-type: none"> The transform size N depends on the desired length M of the time-domain adaptive tap-weight vector. For acoustic echo cancellation (AEC) application, a large transform size is required. The optimal orthogonal transform is the Karhunen-Lo��ve transform (KLT). However, the KLT is data dependent and thus it is not applicable when no <i>a priori</i> information of the input is available. Optimal convergence can only be achieved with the KLT. It has been concluded in [33] that there is no optimal or near optimal transform except KLT.
Filter bank adaptive filter (FBAF)	Recursively updated in transform domain	An $N \times L$ transformation matrix (i.e. a filter bank) is applied on the input signal. The length L of the basis functions of the transform is larger than the transform size N . Similar structure to that of the TDAF. No inverse transform is necessary to obtain the output signal.	<ul style="list-style-type: none"> An $N \times L$ transformation matrix (i.e. a filter bank) is applied on the input signal. The length L of the basis functions of the transform is larger than the transform size N. Can be seen as an extension to the TDAF, where a set of N sparse filters are applied on the transformed outputs instead of one adaptive tap weight for each of the transformed variables in the TDAF. 	<ul style="list-style-type: none"> For TDAF, the transform size N has to be increased for a higher-order adaptive filter. FBAF offers an alternative, where additional adaptive weights are introduced by extending the single weight to a sparse filter. The FBAF approximates a higher-order system with a smaller number of free parameters. Therefore, it can not represent all higher-order systems [37].

Table 2.1(a) Subband and transform-domain adaptive filters (part 2 of 2)

Adaptive filters	Tap weight adaptation	Features	Objectives	Remarks
Conventional subband adaptive filter	Recursively updated in subband domain	<ul style="list-style-type: none"> An $N \times L$ transformation matrix (i.e., a filter bank, as explained in Section 2.4) is employed, where the length L of the basis functions is larger than the transform size N. The input signal and desired response are partitioned into N subbands. Each subband has its own subfilter and adaptation loop, where subband estimation errors are locally evaluated for tap-weight adaptation. A synthesis filter bank (i.e., an inverse transform) is required to obtain the fullband error signal. Computational complexity is greatly reduced by decimating both the order and adaptation rate of the adaptive subfilters. 	<ul style="list-style-type: none"> To improve the convergence rate of gradient-based algorithms when the input signal is colored. Computational complexity is greatly reduced with shorter subfilters that operate at a lower rate. 	<ul style="list-style-type: none"> End-to-end delay of the adaptive filter is increased due to the analysis and synthesis filter banks. In a critically-sampled scheme, there is degradation in convergence performance due to aliasing [4]. In an over-sampled scheme, band-edges of the subband spectrum introduce small eigenvalues, which limits the convergence rate [9].
Multiband-structured subband adaptive filter (MSAF)	Recursively updated in time domain	<ul style="list-style-type: none"> An $N \times L$ transformation matrix is employed, where the length L of the basis functions is larger than the transform size N. The input signal and desired response are partitioned into N subbands. These subband signals are used to adapt a fullband tap-weight vector. The fullband filter operates on the bandlimited input signals at the original sampling rate, while its tap weights are iteratively updated at the decimated rate. A synthesis filter bank (i.e., an inverse transform) is required to obtain the fullband error signal. 	<ul style="list-style-type: none"> To improve the convergence rate of the NLMS algorithm with a minimum amount of additional computations [6, 39, 51]. 	<ul style="list-style-type: none"> End-to-end delay of the adaptive filter is increased due to analysis and synthesis filter banks. The MSAF does not suffer from the aliasing and band-edge effects [6, 39, 51].

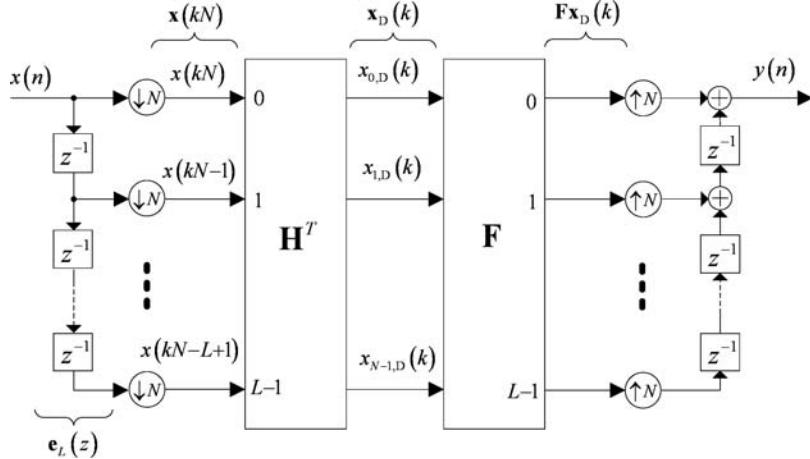


Figure 2.10 Filter bank as a block transform with memory

In Figure 2.10, the delay chain $\mathbf{e}_L(z) = [1, z^{-1}, \dots, z^{-L+1}]$ consists of $L - 1$ delay elements. For every N input samples, the input vector

$$\mathbf{x}(kN) \equiv [x(kN), x(kN - 1), \dots, x(kN - N + 1), x(kN - N), \dots, x(kN - L + 1)]^T \quad (2.30)$$

is packed with N new samples and $L - N$ old samples. The input vector of length L is then mapped into the transformed vector $\mathbf{x}_D(k) = \mathbf{H}^T \mathbf{x}(kN)$ of length N . Notice that the elements of the vector $\mathbf{x}_D(k) \equiv [x_{0,D}(k), x_{1,D}(k), \dots, x_{N-1,D}(k)]^T$ are essentially the decimated subband signals.

The input vector $\mathbf{x}(kN)$ has a higher dimension than the transformed vector $\mathbf{x}_D(k)$. Furthermore, there is an overlap of $L - N$ samples between the current block $\mathbf{x}(kN)$ and the next block $\mathbf{x}[(k + 1)N]$, which indicates the memory of the transform. A similar overlapping operation is applied in the synthesis section. As shown in Figure 2.10, consecutive blocks of the inverse transform $\mathbf{F}\mathbf{x}_D(k)$ have to be overlapped and added to synthesize the fullband output through the delay chain on the right side of the figure. The idea of implementing a filter bank as a block transform with memory is demonstrated in Example 2.5.

Example 2.5 (a complete listing is given in M-file `Example_2P5.m`)

In this example, we use a similar cosine-modulated filter bank as in Example 2.4. The coefficients of the resulting analysis and synthesis filters are stored in the matrices \mathbf{H} and \mathbf{F} , respectively. The direct and inverse transform operations are shown in the following script.

```

[hopt,passedge] = opt_filter(L-1,N);      % Lowpass prototype filter
[H,F] = make_bank(hopt,N);                % Generate filter banks
H = sqrt(N)*H';   F = sqrt(N)*F';        % Scaling

xn = sin(2*pi*0.01*(0:1000));           % Input sinusoidal signal
ITER = length(xn);
x = zeros(length(H),1);                 % Input tapped-delay line
b = zeros(length(F),1);                 % Output tapped-delay line
yn = zeros(ITER,1);

for n = 1:ITER
    x = [xn(n); x(1:end-1)];
    if mod(n,N) ==0
        xD = H.*x;                      % Analysis filtering
        b = F*xD + b;                   % Synthesis filtering
    end
    yn(n) = real(b(1));
    b = [b(2:end); 0];
end

```

2.5 Cosine-modulated filter banks

The theory and design of cosine-modulated filter banks has been studied extensively in the literature [13, 15, 22, 24, 26, 43–46]. In an N -channel cosine-modulated filter bank, the analysis and synthesis filters are cosine-modulated versions of a prototype lowpass filter $P(z)$ with a cutoff frequency of $\pi/2N$, as depicted in Figure 2.11(a). The cosine-modulated analysis filters can be obtained as

$$H_i(z) = \alpha_i P \left[z W_{2N}^{(i+0.5)} \right] + \alpha_i^* P \left[z W_{2N}^{-(i+0.5)} \right], \quad i = 0, 1, \dots, N-1, \quad (2.31)$$

where $W_{2N} = e^{-j\pi/N}$ is the $2N$ th root of unity, α_i is a unit-magnitude constant given as

$$\alpha_i = \exp \left\{ j \left[\theta_i - \frac{\pi}{N}(i+0.5) \left(\frac{L-1}{2} \right) \right] \right\}, \quad \theta_i = (-1)^i \frac{\pi}{4}, \quad (2.32)$$

and L is the length of the prototype filter. The synthesis filters are obtained by time-reversing the analysis filters in the form

$$F_i(z) = z^{-L+1} H_i(z^{-1}). \quad (2.33)$$

The design of complete filter banks thus reduces to the design of the prototype filter.

Perfect (or approximately perfect) reconstruction of the overall analysis–synthesis system can be achieved by optimizing the prototype filter design to satisfy a set of pre-determined constraints. For example, the flatness constraint [13, 15, 24, 26, 43, 45–47]

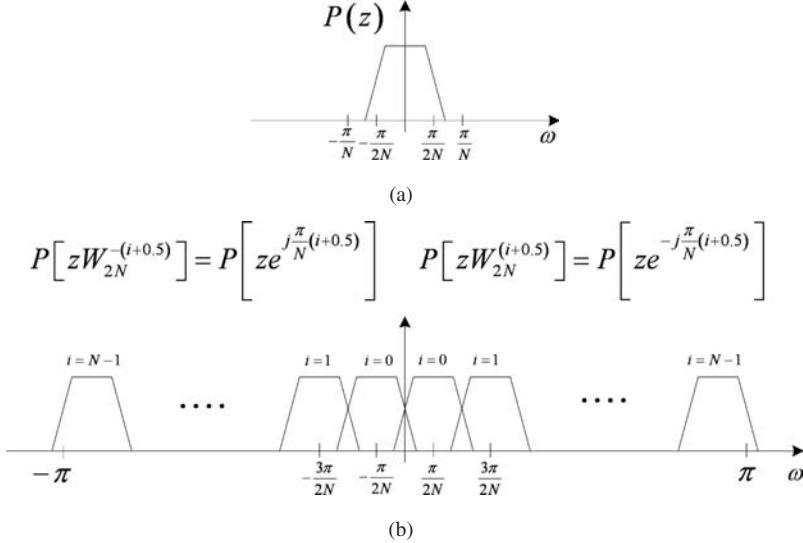


Figure 2.11 Cosine modulation: (a) frequency response of a typical prototype lowpass filter and (b) frequency responses of the cosine-modulated analysis filters

and the $2N$ th band constraint [44, 48] lead to approximately perfect reconstruction. For perfect reconstruction [13, 42], each of the N polyphase components of $P(z)$ should be a spectral factor of a halfband filter (see Table 2.2). Different constraints and optimization procedures have been proposed, resulting in two main classes of cosine-modulated filter banks, namely (i) pseudo-QMF cosine-modulated filter banks and (ii) paraunitary cosine-modulated filter banks, as summarized in Table 2.2.

The prototype filter $P(z)$ is generally restricted to a linear-phase filter with a real-valued symmetric impulse response $p(n)$. Even though the prototype filter has a linear phase, cosine modulation does not result in linear-phase analysis and synthesis filters. The proof is given in Section 2.7. Nevertheless, the time-reversing property of Equation (2.33) ensures that the cascade of the analysis and synthesis filters $F_i(z)H_i(z)$ has a linear phase. Thus the overall distortion transfer function $T(z)$ has a linear phase as shown below:

$$\begin{aligned}
 T(e^{j\omega}) &= \frac{1}{N} \sum_{i=0}^{N-1} F_i(e^{j\omega}) H_i(e^{j\omega}) \\
 &= \frac{1}{N} \sum_{i=0}^{N-1} [e^{-j\omega(L-1)} \tilde{H}_i(e^{j\omega})] H_i(e^{j\omega}) \\
 &= \frac{1}{N} \sum_{i=0}^{N-1} e^{-j\omega(L-1)} H_i^*(e^{j\omega}) H_i(e^{j\omega}) \\
 &= \frac{e^{-j\omega(L-1)}}{N} \sum_{i=0}^{N-1} |H_i(e^{j\omega})|^2.
 \end{aligned} \tag{2.34}$$

Table 2.2 Two main classes of cosine-modulated filter banks

	Pseudo-QMF cosine-modulated filter banks	Paraunitary cosine-modulated filter banks
Aliasing cancellation	Adjacent aliasing component is cancelled structurally by selecting appropriate phase factor in the modulating function. Aliasing from distant bands is assumed to be sufficiently attenuated. That is, the stopband attenuation of a given analysis filter in all non-adjacent bands is sufficiently high by imposing the following condition on the prototype filter:	The polyphase component matrix $\mathbf{E}(z)$ of the filter bank is paraunitary by forcing each of the N polyphase components of the prototype filter to be a spectral factor of a halfband filter. The paraunitary property, together with the time-reversal relationship between the analysis and synthesis filters, ensures perfect signal reconstruction, i.e., phase preserving, amplitude preserving, and alias-free.
Phase distortion		Phase distortion is avoided with the time-reversal relationship between the analysis and synthesis filters.
Amplitude distortion		Amplitude distortion is eliminated by having magnitude response of the distortion transfer function, $ T(e^{j\omega}) $, acceptably flat.

(continued overleaf)

Table 2.2 (continued)

	Pseudo-QMF cosine-modulated filter banks	Paraunitary cosine-modulated filter banks
Constraints	<p>(a) <i>Flatness constraint:</i>. The prototype filter $P(z)$ satisfies the following condition as much as possible</p> $ P(e^{j\omega}) ^2 + P(e^{j(\omega-\pi/N)}) ^2 \cong 1,$ <p style="text-align: center;">for $0 < \omega < \pi/N$.</p> <p>(b) <i>2Nth band constraint:</i>. The prototype filter $P(z)$ is a spectral factor of a $2N$th band filter $Q(z)$, where</p> $Q(z) = P(z)\tilde{P}(z),$ <p style="text-align: center;">and</p> $Q_r(z) + Q_k(-z) = 1.$ <p style="text-align: center;">Let</p> $q_r(l) = p_r(l) * p_r(-l)$ <p style="text-align: center;">and</p> $\sum_{l=0}^{2N-1} Q(zW^l) = 1.$	<p><i>Orthogonality constraint:</i>. Each of the N polyphase components $P_r(z)$ of the prototype filter $P(z) = \sum_{r=0}^{N-1} P_r(z^N)z^{-r}$ is a spectral factor of a halfband filter $Q_r(z)$, where</p> $Q_r(z) = P_r(z)\tilde{P}_r(z),$ <p style="text-align: center;">and</p> $Q_r(z) + Q_k(-z) = 1.$ <p style="text-align: center;">Let</p> $q_r(l) = p_r(l) * p_r(-l)$ <p style="text-align: center;">be the autocorrelation sequence of the rth polyphase component $p_r(l) = p(Nl+r)$ of the prototype filter $p(n)$. For $p_r(l)$ a spectral factor of a halfband filter, its autocorrelation sequence satisfies the following condition</p> $q_r(2l) = \frac{1}{2}\delta(l).$ <p>Let $q(n)$ be the non-causal impulse response of the $2N$th band filter $Q(z)$, the $2N$th band constraint can be equivalently written in the time domain as</p> $q(n) = p(n) * p(-n),$ <p style="text-align: center;">and</p> $q(2Nn) = \frac{1}{2N}\delta(n).$

It can be noted from Equation (2.31) that a cosine-modulated analysis filter $H_i(z)$ consists of two complex-valued filters, $P[zW_{2N}^{(i+0.5)}]$ and $P[zW_{2N}^{-(i+0.5)}]$, as illustrated in Figure 2.11(b), which are generated from $P(z)$ by complex modulation. The impulse response of the right-shifted version, $P[zW_{2N}^{(i+0.5)}]$, is the complex conjugate of that pertaining to the left-shifted version, $P[zW_{2N}^{-(i+0.5)}]$. Combining this conjugate pair of impulse responses gives the analysis filter $H_i(z)$ and the corresponding synthesis filter $F_i(z)$ real-valued impulse responses as follows:

$$\begin{aligned} h_i(n) &= 2p(n) \cos \left[\frac{\pi}{N}(i + 0.5) \left(n - \frac{L-1}{2} \right) + \theta_i \right], \\ f_i(n) &= 2p(n) \cos \left[\frac{\pi}{N}(i + 0.5) \left(n - \frac{L-1}{2} \right) - \theta_i \right]. \end{aligned} \quad (2.35)$$

In conclusion, the practical virtues of cosine-modulated filter banks are summarized as follows:

- (i) The design of the filter bank can be accomplished by designing a single prototype lowpass filter and cosine-modulating the prototype filter to generate the analysis and synthesis filters according to Equation (2.35). Perfect or approximately perfect reconstruction of the overall filter bank (analysis–synthesis system) can be achieved by designing the single prototype filter to satisfy the predefined constraints listed in Table 2.2.
- (ii) A cosine-modulated filter bank consists of analysis and synthesis filters with real-valued coefficients. In a real-time subband signal processing system, it would be expedient to avoid complex arithmetic involving real, imaginary and cross-product terms.
- (iii) Fast transform-based implementation is available due to the modulated nature of bandpass filters [13, 15, 42, 49, 50]. For example, the extended lapped transform filter bank can be computed efficiently with computational complexity that is similar to those of standard transforms [13, 42].
- (iv) A cosine-modulation of a lowpass filter with high stopband attenuation results in a filter bank that is able to generate subband signals that are nearly orthogonal at zero lag [51].

The first three advantages of the cosine-modulated filter banks are well known. The orthogonality property will be analyzed further in Section 3.4.

2.5.1 Design example

Consider the design of a 16-channel cosine-modulated filter bank. The Parks–McClellan algorithm is used to design a prototype filter of length $L = 256$. The stopband edge

frequency ω_s of the prototype lowpass filter is fixed at $\pi/16$, whereas its passband edge ω_p is iteratively adjusted to minimize the following objective function:

$$\phi = \max_{\omega} \left[|P(e^{j\omega})|^2 + |P(e^{j(\omega-\pi/N)})|^2 - 1 \right], \quad 0 < \omega < \pi/N, \quad (2.36)$$

such that the flatness constraint (see Table 2.1) is satisfied [47]. The magnitude responses of all the cosine-modulated filters are depicted in Figure 2.12(a). The transition bands of nonadjacent filters do not overlap since the stopband edge of the prototype filter is fixed at $\omega_s = \pi/16$. Hence, aliasing from distant bands is sufficiently attenuated. The distortion transfer function $T(e^{j\omega})$ of the overall analysis–synthesis system is plotted in Figure 2.12(b). Clearly, the frequency response is almost constant, which allows a nearly perfect reconstruction of the original input signal. The design procedure is detailed in Example 2.6.

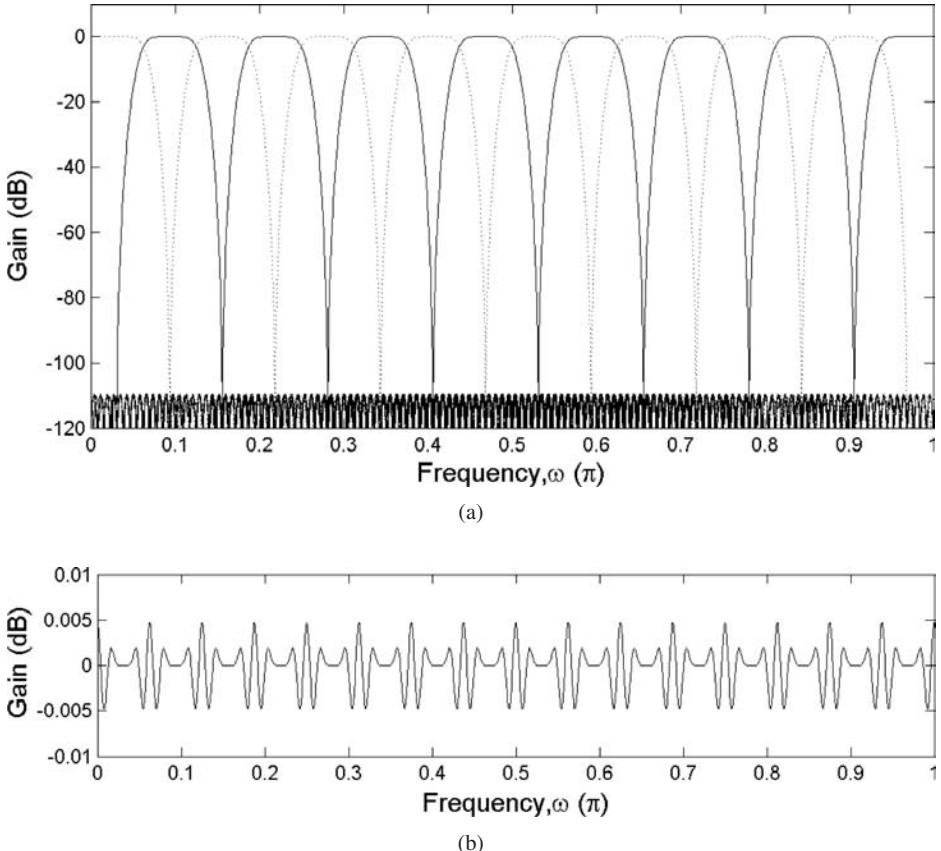


Figure 2.12 A 16-channel pseudo-QMF cosine-modulated filter bank: (a) magnitude responses of all the cosine-modulated filters and (b) distortion transfer function $T(e^{j\omega})$ for the analysis–synthesis system

Example 2.6 (a complete listing is given in M-file `Example_2P6.m`)

The following script generates a 16-channel cosine-modulated filter bank, plots the frequency response (with the resolution specified by `DFTpoint`) and computes the distortion transfer function T_z using Equation (2.15). The `opt_filter` function returns a prototype lowpass filter by optimizing Equation (2.36). The analysis and synthesis filters are then generated according to Equation (2.35). The results are shown in Figure 2.12.

```

N = 16;                                % Number of subbands
K = 8;                                   % Overlapping factor
L = 2*K*N;                             % Length of the analysis filters

[hopt,passedge] = opt_filter(L-1,N);      % Create lowpass
                                           % prototype filter
[H,F] = make_bank(hopt,N);              % Generate filter banks
H = H.';                                 % Analysis section
F = F.';                                 % Synthesis section

%---Plot frequency response-----
DFTpoint = 4096;
[Hz,w] = FreqResp(H,DFTpoint); Hz = Hz.';
figure; subplot(3,1,[1 2]), hold on;
for k = 1:N
    if mod(k,2)==0
        plot(w/pi,20*log10(abs(Hz(k,:))+eps));
    else
        plot(w/pi,20*log10(abs(Hz(k,:))+eps),':');
    end
end
axis([0 1 -120 10]); box on; ylabel('Gain (dB)');
%-----


%---Distortion function-----
Tz = sum(abs(Hz).^2);
subplot(3,1,3); plot(w/pi,10*log10(Tz));
xlabel('Frequency,\omega (\pi)'); ylabel('Gain (dB)');
axis([0 1 -0.01 0.01]);
%-----
```

2.6 DFT filter banks

Similar to the cosine-modulated filter banks, a DFT filter bank [12, 21] is also derived from a prototype filter via modulation. Specifically, the analysis filters of an N -channel DFT filter bank are obtained via complex modulation in the following form:

$$H_i(z) = P \left(z e^{-j2\pi i/N} \right), \quad i = 0, 1, \dots, N-1, \quad (2.37)$$

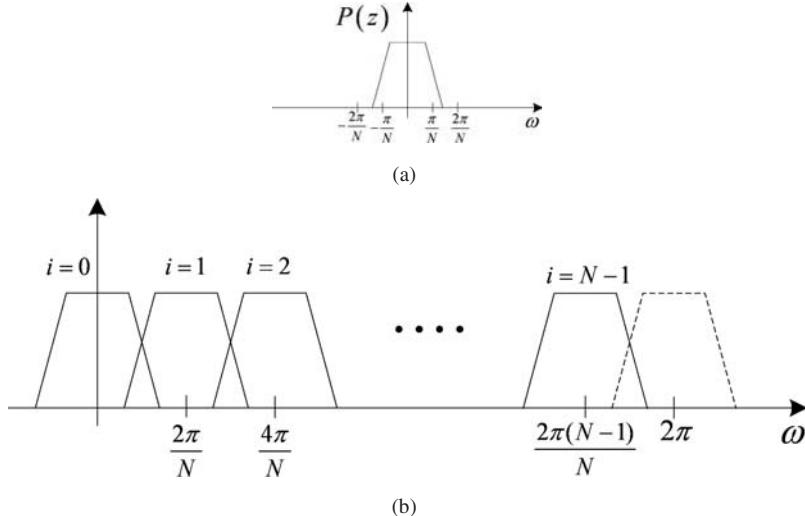


Figure 2.13 Complex modulation: (a) frequency response of a prototype lowpass filter and (b) frequency responses of the complex-modulated filters

where $P(z)$ is the real-valued prototype lowpass filter with a cutoff frequency of π/N . As shown in Figure 2.13, the complex-modulated filters $H_i(z)$ are obtained by shifting the lowpass filter $P(z)$ to the right by multiples of $2\pi/N$. This frequency shifting results in a set of filters that equally divide the normalized frequency from 0 to 2π into N uniform portions with a distance of $2\pi/N$ between adjacent filters.

In the time domain, complex modulation is equivalent to multiplying the impulse response $p(n)$ of the prototype filter with an exponential sequence $e^{j(2\pi i/N)n}$ in the following form:

$$h_i(n) = p(n)e^{j(2\pi i/N)n}, \quad i = 0, 1, \dots, N - 1. \quad (2.38)$$

As a result, the impulse responses of the modulated filters are generally complex valued. If N is an even number, we will obtain a real-valued highpass filter at $i = N/2$, where $h_{N/2}(n) = p(n)(-1)^n$. Furthermore, the impulse response coefficients of $H_i(z)$ and $H_{N-i}(z)$, for $i = 1, 2, \dots, N/2 - 1$, are complex conjugates of each other. Hence, for real-valued signals, only the first $N/2 + 1$ subbands need to be processed.

2.6.1 Design example

We consider the design of an eight-channel DFT filter bank in Example 2.7. The prototype filter has a cutoff frequency of $\pi/8$ and sufficiently high stopband attenuation at frequencies $\omega > \pi/4$. The magnitude responses of all the complex-modulated filters are depicted in Figure 2.14. Notice that the transition bands of nonadjacent filters do not overlap. This allows the aliasing components from distant bands to be sufficiently attenuated. The same prototype filter is usually used to derive the synthesis filters, resulting in two identical sets of analysis and synthesis filters. To cancel the aliasing components between adjacent bands, the modified DFT filter bank [12] is required.

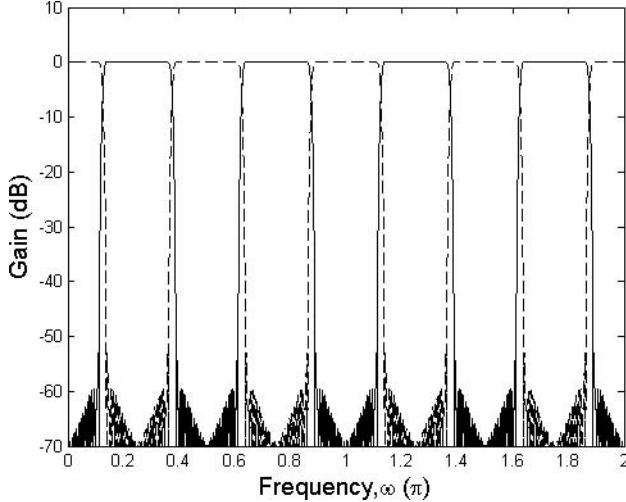


Figure 2.14 Magnitude responses of all the analysis filters for an eight-channel DFT filter bank

Example 2.7 (a complete listing of the M-file is given in `Example_2P7.m`)

The following script generates an eight-channel DFT filter bank and plots the frequency response. The prototype filter of length $L = 256$ is designed using the MATLAB function `fir1`. The `make_bank_DFT` function is then used to generate analysis and synthesis filters according to Equation (2.38). The magnitude responses of the DFT filter bank are depicted in Figure 2.14.

```
N = 8; % Number of subbands
K = 16; % Overlapping factor
L = 2*K*N; % Length of the analysis filters

%---DFT filter bank-----
hopt = fir1(L-1,1/N); % Design a prototype filter
[H,G] = make_bank_DFT(hopt,N);
%-----
```

2.7 A note on cosine modulation

In an N -channel cosine-modulated filter bank, the analysis filters are cosine-modulated versions of a prototype lowpass filter $P(z)$ with a cutoff frequency of $\pi/2N$. The prototype filter $P(z)$ is generally restricted to be a linear-phase filter with a symmetric impulse response [13, 15, 22, 47, 48] such that

$$P(e^{j\omega}) = e^{-j\omega(L-1)/2} P_R(\omega), \quad (2.39)$$

where $P_R(\omega)$ is the real-valued amplitude response of the prototype filter. Evaluating Equation (2.31) on the unit circle and using Equation (2.39) in the expression, the frequency response of the cosine-modulated filter $H_i(z)$ can be formulated in terms of the zero-phase response $P_R(\omega)$ in the following form:

$$H_i(e^{j\omega}) = e^{-j\omega(L-1/2)} [e^{j\theta_i} U_i(\omega) + e^{-j\theta_i} V_i(\omega)], \quad \theta_i = (-1)^i \frac{\pi}{4}, \quad (2.40)$$

where

$$U_i(\omega) = P_R \left[\omega - \frac{\pi}{N}(i + 0.5) \right] \text{ and } V_i(\omega) = P_R \left[\omega + \frac{\pi}{N}(i + 0.5) \right] \quad (2.41)$$

are the right- and left-shifted versions of the zero-phase response, respectively.

For a causal real-valued FIR filter $H(z)$ of length L , it has a linear phase with a symmetric or antisymmetric impulse response if

$$H(e^{j\omega}) = e^{-j\omega(L-1/2)} (e^{\pm j\pi/2} H_R(\omega)), \quad (2.42)$$

where $H_R(\omega)$ is the real-valued zero-phase response of $H(z)$. Comparing Equation (2.39) with Equation (2.42), it is obvious that the cosine-modulated filter $H_i(e^{j\omega})$ cannot have a linear phase due to the alignment factor $\theta_i = \pm\pi/4$, which is not equal to $\pm\pi/2$.

2.8 Summary

This chapter reviewed the theory and design of multirate filter banks, which are the fundamental components of subband adaptive filters. In Section 2.1, the fundamental concepts of multirate signal processing were introduced with emphasis on the properties of decimators and interpolators. The input–output relations of these sampling rate conversion devices were discussed both in the time and transform domains. The basic structure of multirate filter banks was reviewed in Section 2.2. In particular, the input–output relation of multirate filter banks was formulated and the distortion transfer function was defined. Polyphase representation, which is a convenient and effective way of representing the multirate filter banks, was also discussed.

In Section 2.3, the concept of paraunitary was presented. We showed that perfect reconstruction is achievable by imposing the paraunitary condition on the polyphase component matrix of the filter bank. The paraunitary condition can be exploited to simplify the convergence analysis of subband adaptive filters, which will be introduced in Chapter 7. In Section 2.4, the relationships between filter banks and block transforms are briefly described. In particular, we showed that a filter bank can be treated as a block transform with memory. This interpretation provides a convenient way of formulating subband adaptive filtering algorithms, which will be presented in Chapter 6.

Sections 2.5 and 2.6 introduced the analysis and design of modulated filter banks. Cosine and complex modulations are attractive techniques for the design and implementation of uniform filter banks with a large number of subbands. Furthermore, fast transform-based implementation is possible due to the modulated nature of the filters. In addition, cosine-modulated filter banks possess an attractive partial decorrelation feature, which will be examined closely in Chapter 3.

References

- [1] M. de Courville and P. Duhamel, ‘Adaptive filtering in subbands using a weighted criterion’, *IEEE Trans. Signal Processing*, **46**(9), September 1998, 2359–2371.
- [2] P. L. de León and D. M. Etter, ‘Experimental results with increased bandwidth analysis filters in oversampled, subband acoustic echo cancellers’, *IEEE Signal Processing Lett.*, **2**(1), January 1995, 1–3.
- [3] B. Farhang-Boroujeny and Z. Wang, ‘Adaptive filtering in subbands: design issues and experimental results for acoustic echo cancellation’, *Signal Processing*, **61**(3), September 1997, 213–223.
- [4] A. Gilloire and M. Vetterli, ‘Adaptive filtering in subbands with critical sampling: analysis, experiments, and application to acoustic echo cancellation’, *IEEE Trans. Signal Processing*, **40**(8), August 1992, 1862–1875.
- [5] W. Kellermann, ‘Analysis and design of multirate systems for cancellation of acoustical echoes’, in *Proc. ICASSP*, 1988, pp. 2570–2573.
- [6] K. A. Lee and W. S. Gan, ‘Improving convergence of the NLMS algorithm using constrained subband updates’, *IEEE Signal Processing Lett.*, **11**(9), September 2004, 736–739.
- [7] K. A. Lee and W. S. Gan, ‘Inherent decorrelating and least perturbation properties of the normalized subband adaptive filter’, *IEEE Trans. Signal Processing*, **54**(11), November 2006, 4475–4480.
- [8] R. Merched, P. S. R. Diniz and M. R. Petraglia, ‘A new delayless subband adaptive filter structure’, *IEEE Trans. Signal Processing*, **47**(6), June 1999, 1580–1591.
- [9] D. R. Morgan, ‘Slow asymptotic convergence of LMS acoustic echo cancellers’, *IEEE Trans. Speech Audio Processing*, **3**(2), March 1995, 126–136.
- [10] S. S. Pradhan and V. U. Reddy, ‘A new approach to subband adaptive filtering’, *IEEE Trans. Signal Processing*, **47**(3), March 1999, 655–664.
- [11] J. J. Shynk, ‘Frequency domain and multirate adaptive filtering’, *IEEE Signal Processing Mag.*, **9**, January 1992, 14–37.
- [12] N. J. Fliege, *Multirate Digital Signal Processing*, New York: John Wiley & Sons, Inc., 1994.
- [13] H. S. Malvar, ‘Extended lapped transforms: properties, applications, and fast algorithms’, *IEEE Trans. Signal Processing*, **40**(11), November 1992, 2703–2714.
- [14] M. J. T. Smith and S. L. Eddins, ‘Analysis/synthesis techniques for subband image coding’, *IEEE Trans. Acoust., Speech, Signal Processing*, **38**(8), August 1990, 1446–1456.
- [15] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Englewood Cliffs, New Jersey: Prentice Hall, 1993.
- [16] M. Vetterli and J. Kovacević, *Wavelets and Subband Coding*, Englewood Cliffs, New Jersey: Prentice Hall, 1995.
- [17] M. Abo-Zahhad, ‘Current state and future directions of multirate filter banks and their applications’, *Digital Signal Processing*, **13**(3), July 2003, 295–518.
- [18] J. Arrowood, T. Randolph and M. J. T. Smith, ‘Filter bank design’, in *Digital Signal Processing Handbook* (eds V. K. Madisetti and D. B. Williams), Boca Raton, Florida: CRC Press, 1999.
- [19] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Englewood Cliffs, New Jersey: Prentice Hall, 1983.
- [20] E. A. B. da Silva and P. S. R. Diniz, ‘Time-varying filters’, in *Encyclopedia of Electrical and Electronics Engineering* (ed. J. G. Webster), New York: John Wiley & Sons, Inc., 1999.

- [21] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, New York: McGraw-Hill, 2001.
- [22] T. Saramäki and R. Bregović, ‘Multirate systems and filter banks’, in *Multirate Systems: Design and Applications* (ed. G. Jovanovic-Dolecek), Hershey, Pennsylvania: Idea Group, 2002.
- [23] J. G. Proakis and M. G. Manolakis, *Digital Signal Processing – Principles, Algorithms, and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 1996.
- [24] P. L. Chu, ‘Quadrature mirror filter design for an arbitrary number of equal bandwidth channels’, *IEEE Trans. Acoust., Speech, Signal Processing*, **ASSP-33**(1), February 1985, 203–218.
- [25] J. D. Johnston, ‘A filter family designed for use in quadrature mirror filter banks’, in *Proc. ICASSP*, 1980, pp. 291–294.
- [26] J. H. Rothweiler, ‘Polyphase quadrature filters – a new subband coding technique’, in *Proc. ICASSP*, 1983, pp. 1280–1283.
- [27] B. Farhang-Boroujeny and S. Gazor, ‘Performance analysis of transform domain normalized LMS algorithm’, in *Proc. ICASSP*, 1991, pp. 2133–2136.
- [28] B. Farhang-Boroujeny and S. Gazor, ‘Selection of orthonormal transforms for improving the performance of the transform domain normalised LMS algorithm’, *IEE Proc., Part F*, **139**(5), October 1992, 327–335.
- [29] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.
- [30] F. Beaufays, ‘Orthogonalizing adaptive algorithms: RLS, DFT/LMS, and DCT/LMS’, in *Adaptive Inverse Control* (eds B. Widrow and E. Walach), Englewood Cliffs, New Jersey: Prentice Hall, 1995.
- [31] F. Beaufays, *Two-Layer Linear Structures for Fast Adaptive Filtering*, PhD dissertation, Stanford University, 1995.
- [32] W. K. Jenkins and D. F. Marshall, ‘Transform domain adaptive filtering’, in *Digital Signal Processing Handbook* (eds V. K. Madisetti and D. B. Williams), Boca Raton, Florida: CRC Press, 1999.
- [33] D. F. Marshall and W. K. Jenkins, ‘The use of orthogonal transforms for improving performance of adaptive filters’, *IEEE Trans. Circuits Syst.*, **36**(4), April 1989, 474–484.
- [34] S. S. Narayan, A. M. Peterson and M. J. Narasimha, ‘Transform domain LMS algorithm’, *IEEE Trans. Acoust., Speech, Signal Processing*, **31**(3), June 1983, 609–615.
- [35] A. Gilloire, ‘Experiments with subband acoustic echo cancellers for teleconferencing’, in *Proc. ICASSP*, 1987, pp. 2142–2144.
- [36] A. Gilloire and M. Vetterli, ‘Adaptive filtering in subbands’, in *Proc. ICASSP*, 1988, pp. 1572–1575.
- [37] B. E. Usevitch and M. T. Orchard, ‘Adaptive filtering using filter banks’, *IEEE Trans. Circuits Syst. II*, **43**(3), March 1996, 255–265.
- [38] M. de Courville and P. Duhamel, ‘Adaptive filtering in subbands using a weighted criterion’, in *Proc. ICASSP*, 1995, vol. 2, pp. 985–988.
- [39] K. A. Lee and W. S. Gan, ‘Adaptive filtering using constrained subband updates’, in *Proc. ISCAS*, 2005, pp. 2275–2278.
- [40] K. Mayyas and T. Aboulnasr, ‘A fast weighted subband adaptive algorithm’, in *Proc. ICASSP*, 1999, vol. 3, pp. 1249–1252.
- [41] K. Mayyas and T. Aboulnasr, ‘A fast exact weighted subband adaptive algorithm and its application to mono and stereo acoustic echo cancellation’, *J. Franklin Inst.*, **342**(3), May 2005, 235–253.

- [42] H. S. Malvar, *Signal Processing with Lapped Transform*, Norwood, Massachusetts: Artech House, 1992.
- [43] R. V. Cox, ‘The design of uniformly and nonuniformly spaced pseudoquadrature mirror filters’, *IEEE Trans. Acoust., Speech, Signal Processing*, **ASSP-34**(5), October 1986, 1090–1096.
- [44] Y. P. Lin and P. P. Vaidyanathan, ‘A Kaiser window approach for the design of prototype filters of cosine-modulated filterbanks’, *IEEE Signal Processing Lett.*, **5**(6), June 1998, 132–134.
- [45] J. Masson and Z. Picel, ‘Flexible design of computationally efficient nearly perfect QMF filter banks’, in *Proc. ICASSP*, 1985, pp. 541–544.
- [46] H. J. Nussbaumer and M. Vetterli, ‘Computationally efficient QMF filter banks’, in *Proc. ICASSP*, 1984, pp. 11.3.1–11.3.4.
- [47] C. D. Creusere and S. K. Mitra, ‘A simple method for designing high-quality prototype filters for M -band pseudo-QMF banks’, *IEEE Trans. Signal Processing*, **43**(4), April 1995, 1005–1007.
- [48] T. Q. Nguyen, ‘Near-perfect-reconstruction pseudo-QMF banks’, *IEEE Trans. Signal Processing*, **42**(2), March 1994, 65–76.
- [49] R. D. Koilpillai and P. P. Vaidyanathan, ‘Cosine-modulated FIR filter banks satisfying perfect reconstruction’, *IEEE Trans. Signal Processing*, **40**(4), April 1992, 770–783.
- [50] H. S. Malvar, ‘Lapped transforms for efficient transform/subband coding’, *IEEE Trans. Signal Processing*, **38**(6), June 1990, 969–978.
- [51] K. A. Lee and W. S. Gan, ‘On the subband orthogonality of cosine-modulated filter banks’, *IEEE Trans. Circuits Syst. II*, **53**(8), August 2006, 677–681.

3

Second-order characterization of multirate filter banks

For a filter bank to be lossless (i.e. stable, causal and paraunitary), spectral gaps between subbands are not allowed. Thus spectral overlap between adjacent filters is unavoidable in a lossless filter bank [1]. As discussed in Chapter 2, a realizable filter has a transition band and finite stopband attenuation [2–4]. Therefore, a realizable and lossless filter bank is not able to decompose its input signal into completely nonoverlapping spectral bands that are mutually exclusive in their spectral contents, at least when prior knowledge of the input signal is not available. This results in a considerable degree of correlation among subband signals due to the spectral overlaps of the analysis filters.

This chapter analyzes the correlation between subband signals. A correlation-domain formulation for multirate filter banks is used to investigate various conditions for subband orthogonality [5], i.e. whether the subband signals are partially or completely uncorrelated for an arbitrary input signal. A case study is presented at the end of the chapter to demonstrate various techniques for correlation-domain analysis.

In the following sections, a random signal is used as the input signal; thus the resulting subband signals are random in nature. A brief introduction of the basic concepts and notations of stochastic processes can be found in References [3], [4] and [6] to [10]. A more detailed treatment of these subjects can be found in References [11] and [12].

3.1 Correlation-domain formulation

Consider the multirate filter bank as shown in Figure 3.1(a), where only two channels are shown for simplicity. Let the input signal $u(n)$ be a zero-mean random signal and $u_i(n)$ and $u_p(n)$ two arbitrary outputs of the N -channel filter bank. Assuming that $u(n)$ is wide-sense stationary, its autocorrelation function depends only on the time difference (or lag) l between samples, which can be defined in the following form:

$$\gamma_{uu}(l) \equiv E\{u(n)u(n-l)\}, \quad (3.1)$$

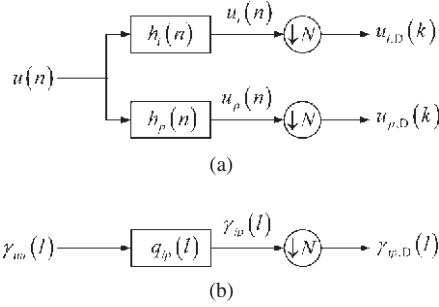


Figure 3.1 Correlation-domain formulation for multirate filter banks (adapted from Reference [5]): (a) the i th and p th channels of an N -band critically decimated analysis filter bank and (b) the effect of partitioning a random signal described in terms of the effect of the system $q_{ip}(l)$ on the autocorrelation function $\gamma_{uu}(l)$

where $E\{\cdot\}$ denotes the expectation operator. By virtue of the linear time-invariant property of the analysis filters, the output subband signals are also wide-sense stationary (see Reference [3] for details) and are related to the input signal by the convolution sum, expressed as

$$u_i(n) = \sum_{r=0}^{\infty} h_i(r)u(n-r) = h_i(n) * u(n), \text{ for } i = 0, 1, \dots, N-1, \quad (3.2)$$

where $*$ denotes the linear convolution operator. Notice that the lower limit 0 on the convolution sum reflects the causality of the analysis filters $h_i(n)$. We also assume that all filter coefficients and signals are real valued. The real-valued formulation is not restrictive, as most systems employ real-valued filter banks in order to avoid complex arithmetic. Nevertheless, the correlation-domain formulation can be easily generalized to include complex-valued filter banks and signals.

The cross-correlation function between two arbitrary subband signals, $u_i(n)$ and $u_p(n)$, in response to the random input signal $u(n)$ is defined as

$$\gamma_{ip}(n, n-l) \equiv E\{u_i(n)u_p(n-l)\}. \quad (3.3)$$

Using Equation (3.2) in the right-hand side of Equation (3.3) and some algebraic manipulation, we obtain

$$\begin{aligned} \gamma_{ip}(n, n-l) &= E \left\{ \sum_{r=0}^{\infty} h_i(r)u(n-r) \sum_{m=0}^{\infty} h_p(m)u(n-l-m) \right\} \\ &= \sum_{r=0}^{\infty} h_i(r) \sum_{m=0}^{\infty} h_p(m) E\{u(n-r)u(n-l-m)\} \\ &= \sum_{r=0}^{\infty} h_i(r) \sum_{m=0}^{\infty} h_p(m) E\{u(n)u[n-(l+m-r)]\}. \end{aligned} \quad (3.4)$$

Assuming that the input signal is wide-sense stationary, from Equation (3.1) the above equation can be written as

$$\begin{aligned}\gamma_{ip}(n, n - l) &= \sum_{r=0}^{\infty} h_i(r) \sum_{m=0}^{\infty} h_p(m) \gamma_{uu}(l + m - r) \\ &= \sum_{r=0}^{\infty} h_i(r) \sum_{m=0}^{\infty} h_p(m) \gamma_{uu}[l - (r - m)].\end{aligned}\quad (3.5)$$

For $s = r - m$, Equation (3.5) can be further simplified to

$$\begin{aligned}\gamma_{ip}(n, n - l) &= \sum_{r=0}^{\infty} h_i(r) \sum_{s=-\infty}^{\infty} h_p(r - s) \gamma_{uu}(l - s) \\ &= \sum_{s=-\infty}^{\infty} \gamma_{uu}(l - s) \sum_{r=0}^{\infty} h_i(r) h_p(r - s).\end{aligned}\quad (3.6)$$

Notice that the summation over s is now from $-\infty$ to ∞ and we have swapped the order of the two summations. It is clear that the subband signals $u_i(n)$ and $u_p(n)$ are jointly wide-sense stationary when $u(n)$ is wide-sense stationary, i.e. their cross-correlation depends only on the time lag l . It can be easily seen that the cross-correlation function (3.6) can also be expressed in terms of the input autocorrelation function $\gamma_{uu}(l)$ in the following form:

$$\gamma_{ip}(l) = \sum_{s=-\infty}^{\infty} q_{ip}(s) \gamma_{uu}(l - s) = q_{ip}(l) * \gamma_{uu}(l), \quad (3.7)$$

where

$$\begin{aligned}q_{ip}(l) &\equiv \sum_{r=0}^{\infty} h_i(r) h_p(r - l) \\ &= \sum_{r=0}^{\infty} h_i(r) h_p[-(l - r)] \\ &= h_i(l) * h_p(-l)\end{aligned}\quad (3.8)$$

is the deterministic cross-correlation sequence between the impulse responses of the analysis filters $h_i(n)$ and $h_p(n)$. It should be emphasized that $q_{ip}(l)$ is the cross-correlation of two finite-energy sequences and should not be confused with the correlation functions of the infinite-energy random processes $\gamma_{uu}(l)$ and $\gamma_{ip}(l)$. The differences between these correlation sequences can be seen in Figure 3.2. In Figure 3.2(a), the finite-duration nature of $q_{ip}(l)$ is obvious as compared to $\gamma_{uu}(l)$ and $\gamma_{ip}(l)$ in Figures 3.2(b) and (c), respectively. The MATLAB script given in Example 3.1 computes and plots these correlation sequences for a filter bank with a colored noise as input. Notice that the autocorrelation and cross-correlation sequences of random processes are absolutely summable even

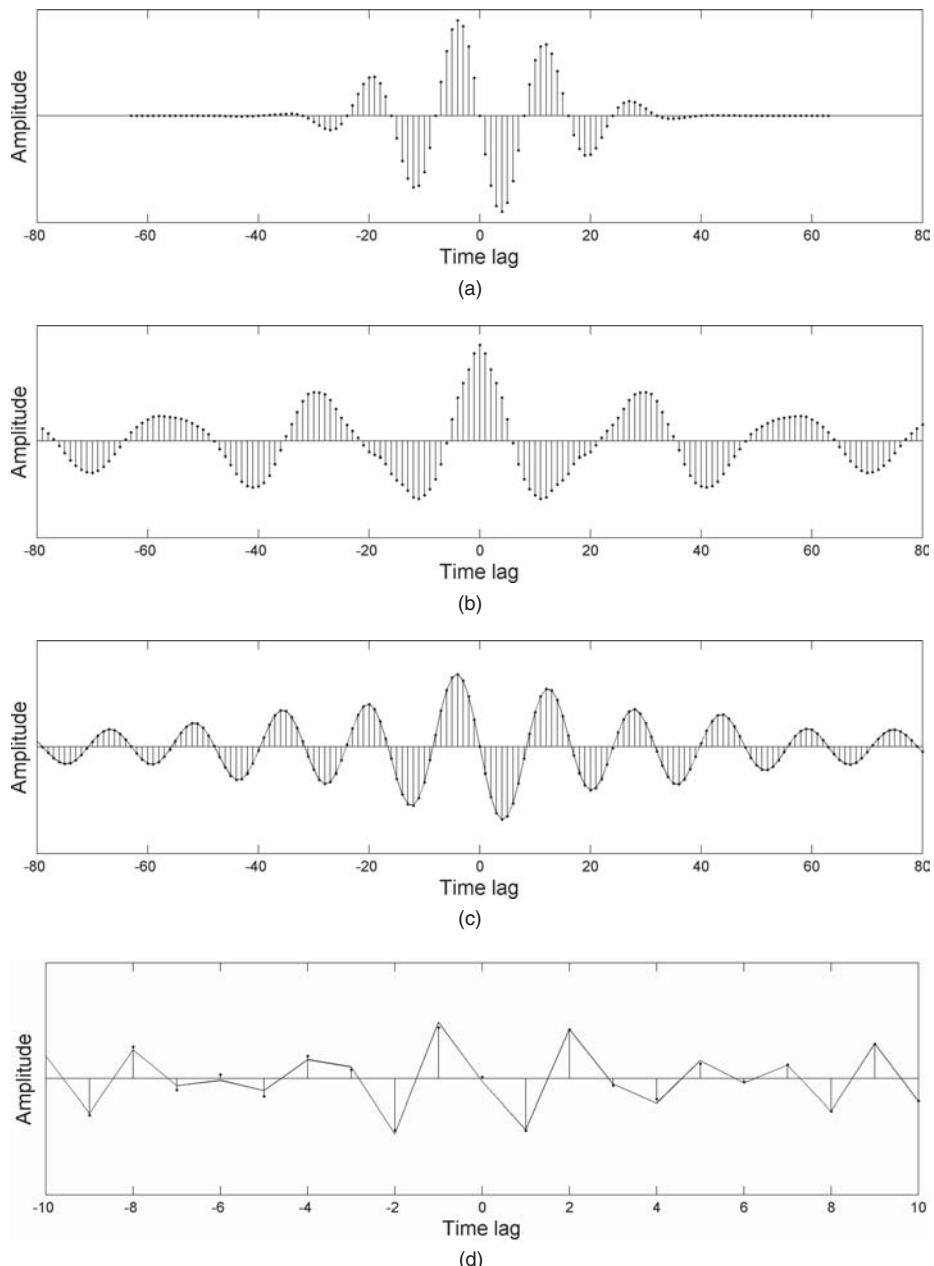


Figure 3.2 Deterministic and stochastic correlation sequences: (a) a deterministic correlation sequence $q_{ip}(l)$ representing the system, (b) an autocorrelation function of the input random process $\gamma_{uu}(l)$, (c) a cross-correlation function $\gamma_{ip}(l)$ between the i th and p th channels and (d) a cross-correlation function $\gamma_{ip,D}(l)$ between the subband random processes after critical decimation

though the random process itself is not; frequency-domain analysis via a Fourier transform is possible, as demonstrated in Section 3.2.

3.1.1 Critical decimation

The correlation-domain formulation can be easily extended to include the critical decimation operation. Let the cross-correlation function between the decimated subband signals $u_{i,D}(k)$ and $u_{p,D}(k)$ be defined as

$$\gamma_{ip,D}(k, k-l) \equiv E\{u_{i,D}(k)u_{p,D}(k-l)\}. \quad (3.9)$$

The $N : 1$ decimator retains only those samples of $u_i(n)$ and $u_p(n)$ that occur at instants of time equal to multiples of N (as explained in Section 2.1). Hence, the decimated subband signals can be written as

$$u_{i,D}(k) = u_i(kN) \text{ and } u_{p,D}(k-l) = u_p[(k-l)N]. \quad (3.10)$$

Substituting Equation (3.10) into Equation (3.9) and using the fact that $u_i(n)$ and $u_p(n)$ are jointly wide-sense stationary so that their cross-correlation function depends only on the time difference lN , we obtain

$$\gamma_{ip,D}(l) = E\{u_i(kN)u_p(kN - lN)\} = \gamma_{ip}(lN). \quad (3.11)$$

Equation (3.11) signifies that the cross-correlation function $\gamma_{ip,D}(l)$ of the decimated subband signals is the N -fold decimated version of $\gamma_{ip}(l)$, as can be seen from Figures 3.2(c) and (d).

The correlation-domain formulation for multirate filter banks is summarized in Figure 3.1. With this formulation, the effect of filtering a random signal with a filter bank can be conveniently described in terms of the effect of the system on the autocorrelation function $\gamma_{uu}(l)$ of the random signal. Specifically, the cross-correlation function $\gamma_{ip}(l)$ between the subband signals can be determined in terms of the input autocorrelation function $\gamma_{uu}(l)$ and the characteristics of the analysis filters $q_{ip}(l)$. This idea is illustrated further in Example 3.1. The concept of correlation-domain formulation is well documented in signal processing books [3, 4, 7, 9]. Here, we have extended the formulation for multirate filter banks to facilitate the analysis of their second-order characteristics, which will be examined closely in Section 3.4 in the case of cosine-modulated filter banks.

Example 3.1 (a complete listing is given in M-file `Example_3P1.m` in the companion CD)

We analyze the second-order characteristics of a cosine-modulated filter bank described in Section 2.5. The input signal is a colored noise generated by filtering a unit-variance white Gaussian noise with an autoregressive model realized as an all-pole infinite impulse response (IIR) filter, which consists of 15 linear-predictive

coding (LPC) coefficients (more details will be given in Section 3.4.2). In the following, we consider the case where $i = 0$ and $p = 1$. The deterministic system correlation sequence $q_{ip}(l)$ and the stochastic autocorrelation sequence $\gamma_{ii}(l)$ are first computed and plotted in Figures 3.1(a) and (b), respectively. In the second part, we compute the cross-correlation sequences before and after critical decimation, $\gamma_{ip}(l)$ and $\gamma_{ip,D}(l)$, using two different methods. In the direct method, the cross-correlation sequences are directly estimated from the subband signals. On the other hand, the indirect method computes the cross-correlation sequences via the correlation-domain formulation. Both methods produce the same results, where identical sequences are plotted in Figures 3.2(c) and (d).

```
%---Design filter bank-----
N = 8; % Number of subbands
L = 8*N; % Length of lowpass prototype filter
[hopt,passedge] = opt_filter(L-1,N); % Prototype filter design
H = make_bank(hopt,N); % Filter bank generation

%---Generate colored noise-----
a = [1.00,-1.32, 0.86,-0.45, ...
      -0.62, 0.98,-0.65, 0.54, ...
      0.37,-0.51, 0.37,-0.40, ...
      0.06,-0.15, 0.12, 0.20]; % Speech LPC coefficients
randn('state',sum(100*clock));
wgn = randn(4*80000,1); % Generate white Gaussian noise
un = filter(1,a,wgn); % IIR filtering
un = un/std(un); % Make unit variance

%---Input autocorrelation sequence--
maxlag = 1000; % Number of time lags for
r_uu = xcorr(un,maxlag,'unbiased'); % correlation computation

%---Deterministic cross-correlation sequence--
k = 0; % Select a subband: 0,1,...,N-1
i = k ; p = k+1 ; % Adjacent subbands i and p
h_i = H(i+1,:); h_p = H(p+1,:); % Analysis filters
q_ip = xcorr(h_i,h_p); % System correlation sequence
% ---Direct method-----
un_i = filter(h_i,1,un); % ith subband signal
un_p = filter(h_p,1,un); % pth subband signal
r_ip = xcorr(un_i,un_p,maxlag,'unbiased');
un_i_D = un_i(1:N:end); % Critical decimation
un_p_D = un_p(1:N:end);
r_ip_D = xcorr(un_i_D,un_p_D,maxlag,'unbiased');

%---Indirect method-----
s_ip = conv(q_ip,r_uu);
ind = -(length(s_ip)-1)/2:1:(length(s_ip)-1)/2;
[foo1,foo2] = find(mod(ind,N)==0);
```

```

s_ip_D = s_ip(foo2); % Critical decimation
ind_D = -(length(s_ip_D)-1)/2:1:(length(s_ip_D)-1)/2;

figure;
subplot(2,1,1); plot(-maxlag:1:maxlag,r_ip); hold on;
subplot(2,1,1); stem(ind,s_ip,'.');
ylabel('Amplitude'); xlabel('Time lag');
axis([-80 80 -0.2 0.2]);
subplot(2,1,2); plot(-maxlag:1:maxlag,r_ip_D); hold on;
subplot(2,1,2); stem(ind_D,s_ip_D,'.');
ylabel('Amplitude'); xlabel('Time lag');
axis([-20 20 -0.05 0.05]);

```

3.2 Cross spectrum

The cross-correlation function $\gamma_{ip}(l)$ represents the time-domain description of the statistical relation between subband signals. The statistical relation can be represented in the frequency domain by taking the Fourier transform of Equation (3.7) to obtain the cross-power spectrum $\Gamma_{ip}(e^{j\omega})$ in the form

$$\begin{aligned}
\Gamma_{ip}(e^{j\omega}) &= \sum_{l=-\infty}^{\infty} \gamma_{ip}(l)e^{-j\omega l} \\
&= \left[\sum_{l=-\infty}^{\infty} q_{ip}(l)e^{-j\omega l} \right] \left[\sum_{l=-\infty}^{\infty} \gamma_{uu}(l)e^{-j\omega l} \right] \\
&= Q_{ip}(e^{j\omega}) \Gamma_{uu}(e^{j\omega}),
\end{aligned} \tag{3.12}$$

where $\Gamma_{uu}(e^{j\omega})$ is the power spectrum of the input signal $u(n)$ and $Q_{ip}(e^{j\omega})$ is the cross-energy spectrum between the two finite-energy impulse responses of the analysis filters.

The cross-energy spectrum $Q_{ip}(e^{j\omega})$ between the i th and p th analysis filters is obtained by taking the Fourier transform of Equation (3.8) as

$$\begin{aligned}
Q_{ip}(e^{j\omega}) &\equiv \sum_{l=-\infty}^{\infty} q_{ip}(l)e^{-j\omega l} \\
&= \left[\sum_{l=0}^{\infty} h_i(l)e^{-j\omega l} \right] \left[\sum_{l=-\infty}^0 h_p(-l)e^{-j\omega l} \right] \\
&= \left[\sum_{l=0}^{\infty} h_i(l)e^{-j\omega l} \right] \left[\sum_{l=0}^{\infty} h_p(l)e^{j\omega l} \right]
\end{aligned} \tag{3.13}$$

$$\begin{aligned}
&= \left[\sum_{l=0}^{\infty} h_i(l) e^{-j\omega l} \right] \left[\sum_{l=0}^{\infty} h_p(l) e^{-j\omega l} \right]^* \\
&= H_i(e^{j\omega}) [H_p(e^{j\omega})]^* \\
&= |H_i(e^{j\omega})| |H_p(e^{j\omega})| e^{j\phi_{ip}(\omega)},
\end{aligned}$$

where $H_i(e^{j\omega})$ and $H_p(e^{j\omega})$ are the frequency responses of the analysis filters, $\phi_{ip}(\omega) = \phi_i(\omega) - \phi_p(\omega)$ is the phase difference and $*$ denotes the complex conjugation. The idea is demonstrated in Example 3.2 and Figure 3.3. Figure 3.3(a) shows the power spectrum of the colored noise with the autocorrelation function as plotted in Figure 3.2(b). To obtain the cross-energy spectrum of the system shown in Figure 3.3(b), we take the Fourier transform of the deterministic correlation sequence $q_{ip}(l)$ shown in Figure 3.2(a). The cross-power spectrum $\Gamma_{ip}(e^{j\omega})$ between the i th and p th subband signals is given by the product $Q_{ip}(e^{j\omega}) \Gamma_{uu}(e^{j\omega})$, as shown in Figure 3.3(c).

Example 3.2 (a complete listing is given in M-file `Example_3P2.m`)

We use the same setup as Example 3.1 to analyze the second-order characteristics of the cosine-modulated filter bank in the frequency domain. The input signal is the same as in Example 3.1. The input spectrum is the squared-magnitude response of the autoregressive model, as shown in Figure 3.3(a). Similar to that in Example 3.1, we consider the case where $i = 0$ and $p = 1$. The cross-energy spectrum gives the frequency-domain representation of the cross-correlation function between the two analysis filters. The cross-power spectrum between the i th and p th subband signals is determined by the characteristics of the input signal and the analysis filters.

```

%---Design filter bank-----
N = 8; % Number of subbands
L = 8*N; % Length of lowpass prototype filter
[hopt,passedge] = opt_filter(L-1,N); % Prototype filter design
H = make_bank(hopt,N); % Filter bank generation

%---Speech LPC coefficients---
a = [1.00,-1.32,0.86,-0.45,
      -0.62,0.98,-0.65,0.54,
      0.37,-0.51,0.37,-0.40,
      0.06,-0.15, 0.12,0.20];

%---Filter bank-----
k = 0; % Subband index, k = 0,1,...,N-1
i = k ; p = k+1; % Adjacent subbands i and p
h_i = H(i+1,:); h_p = H(p+1,:); % Analysis filters
q_ip = xcorr(h_i,h_p); % System correlation sequence

```

```

DFTpoint = 4096;
[A,w] = FreqResp(a,DFTpoint); % Autoregressive model
[Q_ip,w] = FreqResp(q_ip,DFTpoint); % Cross-energy spectrum
[H_i,w] = FreqResp(h_i,DFTpoint); % Freq response of ith filter
[H_p,w] = FreqResp(h_p,DFTpoint); % Freq response of pth filter

InputSpectrum = 1./abs(A).^2; % Input power spectrum
CrossSpectrum = Q_ip.*InputSpectrum; % Cross-power spectrum

```

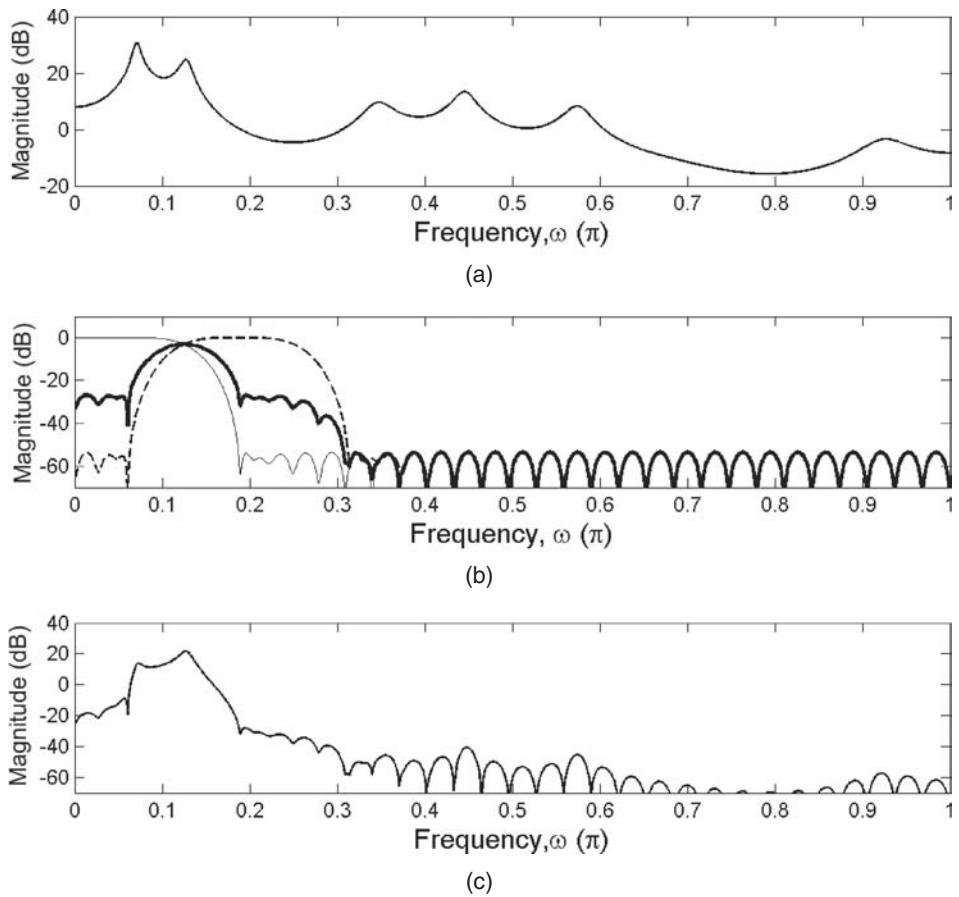


Figure 3.3 Frequency-domain characterization of the filter bank (adapted from Reference [5]): (a) the input power spectrum $|\Gamma_{uu}(e^{j\omega})|$, (b) the cross-energy spectrum $|Q_{ip}(e^{j\omega})|$ between the i th and p th analysis filters where the light solid and dotted lines represent the magnitude responses $|H_i(e^{j\omega})|^2$ and $|H_p(e^{j\omega})|^2$, respectively, and (c) the cross-power spectrum $|\Gamma_{ip}(e^{j\omega})|$ between the subband signals

The cross-correlation function $\gamma_{ip}(l)$ can be recovered from the cross-power spectrum $\Gamma_{ip}(e^{j\omega})$ through the inverse Fourier transform as follows:

$$\gamma_{ip}(l) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \Gamma_{ip}(e^{j\omega}) e^{j\omega l} d\omega. \quad (3.14)$$

From Equation (3.14), we can interpret the cross-power spectrum $\Gamma_{ip}(e^{j\omega_0})$ as a measure of the correlation between two subband signals at a given frequency ω_0 ; i.e. it represents the correlation coefficient in the frequency domain. Recall that the subband signals $u_i(n)$ are generated from a common input signal $u(n)$ by analysis filters $H_i(z)$ with their passbands occupying contiguous portions of the original spectral band. The resulting subband signals $u_i(n)$ and $u_p(n)$ are orthogonal if their cross-correlation function $\gamma_{ip}(l)$ is zero at all lags l . Using Equations (3.12) and (3.13) in Equation (3.14), the characteristic of $\gamma_{ip}(l)$ becomes

$$\begin{aligned} \gamma_{ip}(l) &= \frac{1}{2\pi} \int_{-\pi}^{\pi} H_i(e^{j\omega}) [H_p(e^{j\omega})]^* \Gamma_{uu}(e^{j\omega}) e^{j\omega l} d\omega \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} |H_i(e^{j\omega})| |H_p(e^{j\omega})|^* \Gamma_{uu}(e^{j\omega}) e^{j[\phi_{ip}(\omega)+\omega l]} d\omega. \end{aligned} \quad (3.15)$$

This equation shows that the orthogonality of the subband signals can be assured if the filter bank consists of ideal nonoverlapping bandpass filters $H_i(e^{j\omega})$, whereby their cross-energy spectrum $Q_{ip}(e^{j\omega})$ is zero at all frequencies ω . In the time domain, this condition is equivalent to the case where $q_{ip}(l)$ is zero at all lags l , as can be seen from Equation (3.7).

Therefore, the subband signals are orthogonal if they occupy nonoverlapping portions of the original frequency band. Such a complete decorrelation (or separation) of subbands may be desirable, but is not achievable in practice because realizable filters have a transition band and finite stopband attenuation. In a realizable and lossless filter bank [1], overlap between adjacent bands is unavoidable. Consequently, considerable degrees of correlation exist between adjacent subband signals. In Sections 3.3 and 3.4, we will study a less stringent condition where the subband signals are orthogonal at zero lag only, i.e. the subband signals are only partially decorrelated.

3.2.1 Subband spectrum

By setting $p = i$, the power spectrum of the subband signal $u_i(n)$ can be expressed as

$$\Gamma_{ii}(e^{j\omega}) = |H_i(e^{j\omega})|^2 \Gamma_{uu}(e^{j\omega}), \text{ for } i = 0, 1, \dots, N-1. \quad (3.16)$$

The subband signals $u_0(n), u_1(n), \dots, u_{N-1}(n)$ always have a smaller bandwidth than the original fullband signal $u(n)$. Decimating the subband signals to a lower rate commensurate with their bandwidth reduces the spectral dynamic range (see Section 1.5).

In Section 3.1.1, we have seen that the autocorrelation function of the decimated subband signal $u_{i,D}(k)$ is the decimated version of the autocorrelation function of the subband signal $u_i(n)$. Hence, the power spectrum $\Gamma_{ii,D}(e^{j\omega})$ of the decimated subband signal $u_{i,D}(k)$ can be obtained from Equations (3.16) and (2.2) as

$$\Gamma_{ii,D}(e^{j\omega}) = \frac{1}{N} \sum_{l=0}^{N-1} \Gamma_{ii}(e^{j(\omega-2\pi l)/N}). \quad (3.17)$$

In Equation (3.17), the decimated spectrum $\Gamma_{ii,D}(e^{j\omega})$ is the sum of N uniformly shifted and stretched versions of the subband spectrum $\Gamma_{ii}(e^{j\omega})$ that have been scaled with a factor of $1/N$. Since $\Gamma_{ii}(e^{j\omega})$ is periodic over 2π , the shift operation can be performed by circular shifting on the unit circle. This idea is demonstrated by the MATLAB script given in Example 3.3. The decimation operation can be graphically interpreted as follows:

- (i) Create $N - 1$ copies of $\Gamma_{ii}(e^{j\omega})$ by shifting it uniformly in successive amounts of $2\pi l/N$.
- (ii) Add all the $N - 1$ shifted versions $\Gamma_{ii}(e^{j(\omega-2\pi l/N)})$ to the original $\Gamma_{ii}(e^{j\omega})$ and divide the result by N .
- (iii) Stretch the sum obtained in (ii) with a factor of N .

Figure 3.4(a) shows the power spectrum of the colored noise used in Examples 3.1 and 3.2 for a period of 2π . The power spectrum of the first subband is shown in Figure 3.4(b). Using the procedure outlined above, the subband spectrum after critical decimation is computed and plotted in Figure 3.4(c). Clearly, the spectrum of the subband signal after critical decimation is flatter than the original input spectrum and is closer to that of white noise. Chapter 4 will show that a smaller spectral dynamic range leads to faster convergence for subband adaptive filters.

Example 3.3 (a complete listing is given in M-file `Example_3P3.m`)

This example investigates the effects of decimation to the spectra of subband signals. The first part of the code is similar to the code used in Example 3.2, except that $p = i = 0$ in order to evaluate the spectrum of the subband signal instead of the cross-spectrum between channels. The results are plotted in Figure 3.4. Notice that the frequency axes are from 0 to 2π . All the power spectra are symmetric with respect to π since the input and subband signals are real valued. Also the decimation operation introduces aliasing into the subband signals, which is indicated by the dotted line in Figure 3.4(c).

```
%---Design filter bank-----
N = 8; % Number of subbands
L = 8*N; % Length of lowpass prototype filter
[hopt,passedge] = opt_filter(L-1,N); % Prototype filter design
```

```

H = make_bank(hopt,N);                                % Filter bank generation

%---Speech LPC coefficients-----
a = [1.00,-1.32,0.86,-0.45,
-0.62,0.98,-0.65,0.54,
0.37,-0.51,0.37,-0.40,
0.06,-0.15, 0.12,0.20];

%---Filter bank-----
k = 0;                                              % Subband index, k = 0,1,...,N-1
i = k ; p = k;                                       % Setting i = p
h_i = H(i+1,:); h_p = H(p+1,:); % Analysis filters
q_ip = xcorr(h_i,h_p);                            % System correlation sequence

DFTpoint = 4096;
[A,w] = FreqResp(a,DFTpoint);                      % Autoregressive model
[Q_ip,w] = FreqResp(q_ip,DFTpoint); % Energy spectrum

InputSpectrum = 1./ (abs(A).^2);
SubbandSpectrum = abs(Q_ip).*InputSpectrum
[SubbandSpectrum_D,w_D,XD] = PSDecimation(SubbandSpectrum,DFTpoint,N);

%-----
function [xd,w,XD] = PSDecimation(X,DFTpoint,D)
% X           Power spectrum to be decimated, 0 to 2pi,
%             and a real function of w.

delta_w = 2*pi/DFTpoint;
XD = zeros(D,DFTpoint);
k = 0;
while k <= D-1
    N = round((2*pi*k/D)/delta_w); % No. of points to be shifted
                                    % corresponding to (2*pi*k)/D
    x = X; n = 0;                  % Freq response to be shifted
    while n < N
        x = [x(end) x(1:end-1)];
        n = n + 1;
    end
    XD(k+1,:) = x;
    k = k + 1;
end
Npoint = round(DFTpoint/D);
Npoint = floor(Npoint/4)*4;
XD = XD(:,1:Npoint)/D;          % Stretch the frequency response
xd = sum(XD,1);                % Sum the frequency shifted versions
delta_w = 2*pi/Npoint;
w = (0:Npoint-1)*delta_w;

```

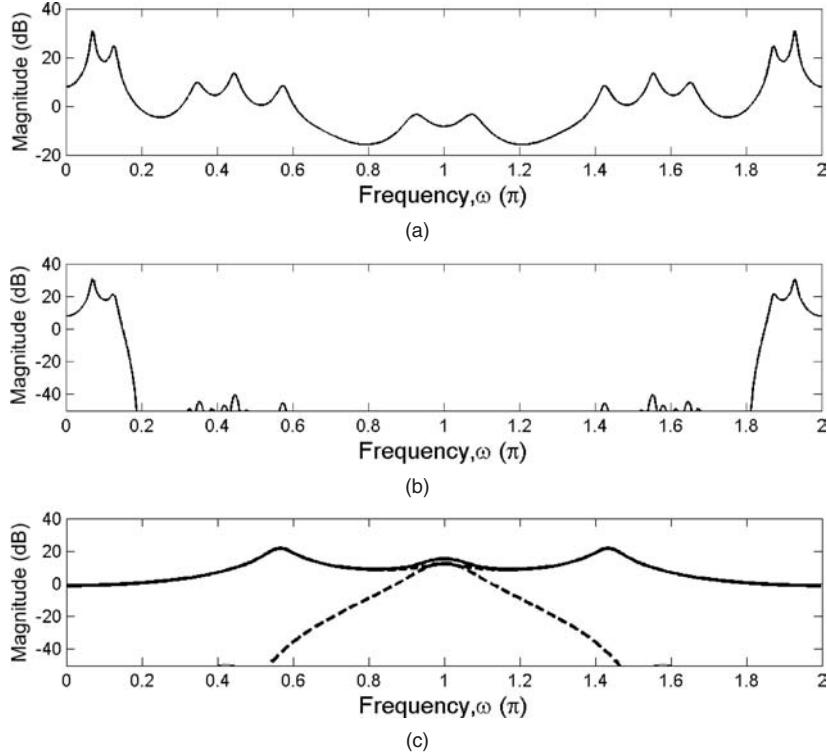


Figure 3.4 Subband spectra before and after decimation: (a) the input power spectrum $\Gamma_{uu}(e^{j\omega})$, (b) the band-limited subband spectrum $\Gamma_{ii}(e^{j\omega})$ and (c) the subband spectrum after critical decimation $\Gamma_{ii,D}(e^{j\omega})$, where the dotted line shows aliasing introduced by decimation

3.3 Orthogonality at zero lag

Assuming that the analysis filters $h_i(n)$ and $h_p(n)$ are causal FIR filters of length L , their cross-correlation function $q_{ip}(l)$, as defined in Equation (3.8), is a noncausal sequence of length $2L - 1$ expressed as

$$q_{ip}(l) = \begin{cases} h_i(l) * h_p(-l), & |l| \leq L - 1, \\ 0, & |l| \geq L. \end{cases} \quad (3.18)$$

Substituting Equation (3.18) into Equation (3.7), the cross-correlation function of the subband signals at zero lag ($l = 0$) can be written as

$$\gamma_{ip}(0) = \sum_{s=-L+1}^{L-1} q_{ip}(s) \gamma_{uu}(-s). \quad (3.19)$$

The autocorrelation function of a wide-sense stationary random process is a symmetric function of the time lag l , i.e. $\gamma_{uu}(l) = \gamma_{uu}(-l)$. If the deterministic cross-correlation

sequence is antisymmetric, i.e. $q_{ip}(l) = -q_{ip}(-l)$, we have $\gamma_{ip}(0) = 0$ in Equation (3.19) because the sum of the product of the symmetric sequence $\gamma_{uu}(-s)$ and the antisymmetric sequence $q_{ip}(s)$ is always zero. In other words, the subband signals can be orthogonal at zero lag, i.e. $\gamma_{ip}(0) = E\{u_i(n)u_p(n)\} = 0$ for $i \neq p$, even though $\gamma_{ip}(l)$ may be large at $l \neq 0$. This property of filter banks can be obtained by manipulating the relative phase between the analysis filters in such a way that the cross-correlation sequence between their impulse responses is antisymmetric.

The practical virtue of orthogonality at zero lag can be seen by considering a random vector $\mathbf{u}_N(n)$ of subband variables shown in Figure 3.5 in the following form:

$$\mathbf{u}_N(n) \equiv [u_0(n), u_1(n), \dots, u_{N-1}(n)]^T. \quad (3.20)$$

Clearly, if the subband signals are orthogonal at zero lag, where $\gamma_{ip}(0) = E\{u_i(n)u_p(n)\} = 0$ for $i \neq p$, the correlation matrix $\mathbf{R}_N \equiv E\{\mathbf{u}_N(n)\mathbf{u}_N^T(n)\}$ of random vectors is diagonal. That is because the random vector $\mathbf{u}_N(n)$ consists of uncorrelated components. In the simplest form, if \mathbf{R}_N is diagonal, the system equation $\mathbf{R}_N \mathbf{x} = \mathbf{y}$ is uncoupled; i.e. the components, $E\{u_i^2(n)\}x_i = y_i$ for $i = 0, 1, \dots, N-1$, do not depend on each other and thus can be solved separately. Here, the variables x_i and y_i are elements of the vectors \mathbf{x} and \mathbf{y} , respectively. This feature can be exploited to improve the efficiency of subband processing algorithms that involve explicit usage of the correlation matrix \mathbf{R}_N or its estimates [13–15]. It should be emphasized that, as a direct consequence of Equation (3.11), the orthogonality of the subband variables, i.e. $E\{u_i(n)u_p(n)\} = 0$, guarantees the orthogonality of the decimated subband variables such that $E\{u_{i,D}(n)u_{p,D}(n)\} = 0$.

3.3.1 Paraunitary condition

In filter bank design, it is common for a paraunitary condition to be imposed on the analysis filters in order to obtain a perfect-reconstruction filter bank. Using the notion of the correlation-domain formulation shown in Figure 3.1, the paraunitary condition

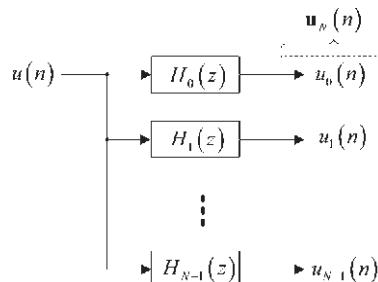


Figure 3.5 An N -channel analysis filter. The random vector $\mathbf{u}_N(n)$ consists of the random variables $u_0(n), u_1(n), \dots, u_{N-1}(n)$

[16, 17] for a filter bank is given by

$$q_{ip}(lN) = \sum_{r=0}^{L-1} h_i(r)h_p(r - lN) = \delta(l)\delta(i - p), \quad (3.21)$$

where

$$\delta(l) = \begin{cases} 1, & \text{for } l = 0, \\ 0, & \text{for } l \neq 0, \end{cases} \quad (3.22)$$

denotes the unit sample sequence. Equation (3.21) indicates that the impulse responses of the paraunitary filter bank, $h_i(n)$ for $i = 0, 1, \dots, N - 1$, as well as the lN sample shifted versions, $h_i(n - lN)$, are orthogonal in a deterministic sense. Substituting $\gamma_{uu}(l) = \sigma_u^2\delta(l)$ into Equation (3.7) for the case where $u(n)$ is white noise with variance σ_u^2 , we obtain

$$\gamma_{ip}(l) = q_{ip}(l) * [\sigma_u^2\delta(l)] = \sigma_u^2 q_{ip}(l). \quad (3.23)$$

From Equations (3.21) and (3.23), it follows that the subband signals are orthogonal at zero lag, i.e. $\gamma_{ip}(0) = \sigma_u^2 q_{ip}(0) = 0$ for $i \neq p$. Furthermore, the cross-correlation function $\gamma_{ip}(l)$ is characterized by periodic zero-crossing separated by N sampling periods, which is a direct consequence of the paraunitary condition defined in Equation (3.21). However, this result holds if and only if the input signal is white. For nonwhite excitation where $\gamma_{uu}(l) \neq \sigma_u^2\delta(l)$, the cross-correlation function $\gamma_{ip}(l)$ is the convolution of the two sequences $q_{ip}(l)$ and $\gamma_{uu}(l)$ as given in Equation (3.7). The characteristics of $q_{ip}(l)$ induced by the paraunitary condition are generally not preserved in $\gamma_{ip}(l)$ due to the convolution operation. Therefore, the paraunitary condition is not sufficient for the orthogonality of subband signals, except when the input signal is white. For example, the discrete cosine transform (DCT) produces transformed variables that are orthogonal under the white input signal because the DCT matrix is orthogonal (or paraunitary [1]). However, under colored signals, the transformed variables are generally not orthogonal [13]. Orthogonality of a filter bank does not ensure the orthogonality of the subband signals. Example 3.4 investigates the orthogonality of the DCT under white and colored signals.

The antisymmetry of the deterministic cross-correlation sequence $q_{ip}(l)$ leads to the orthogonality of the subbands. The paraunitary condition (3.21) only ensures that $q_{ip}(l)$ has periodic zero crossing, therefore not guaranteeing orthogonality at zero lag for arbitrary types of input spectrum. Nevertheless, when the number of subbands N is sufficiently large and the stopband attenuation is high enough, the spectrum of the input signal can be assumed to be white over each subband. Under this situation, the paraunitary property imposed on $q_{ip}(l)$ can almost be preserved in $\gamma_{ip}(l)$, thereby allowing the outputs of the paraunitary filter bank to be approximately orthogonal at zero lag.

Example 3.4 (a complete listing is given in M-file `Example_3P4.m`)

This example investigates the subband orthogonality of a type-IV DCT [16]. Let $\mathbf{A} = [a_{nk}]$ be the DCT matrix with the coefficients defined as

$$a_{nk} = \sqrt{\frac{2}{N}} \cos \left[\left(n + \frac{1}{2} \right) \left(k + \frac{1}{2} \right) \frac{\pi}{N} \right].$$

As discussed in Section 2.4, the basis functions of a unitary transform (i.e. the column of the matrix \mathbf{A}) can be seen as a uniform filter bank. Similar to the previous example, we consider the first and second subbands where $i = 0$ and $p = 1$. The deterministic correlation sequence $q_{ip}(l)$ is depicted in Figure 3.6(a). The stochastic cross-correlation sequence $\gamma_{ip}(l)$ is calculated for white and colored excitation signals. The results are plotted in Figures 3.6(b) and (c). Notice that the subband signals are orthogonal at zero lag, i.e. $\gamma_{ip}(0) = 0$, in Figure 3.6(b) due to the unitary property. However, this property is not preserved when the input signal is colored, which is evident in Figure 3.6(c).

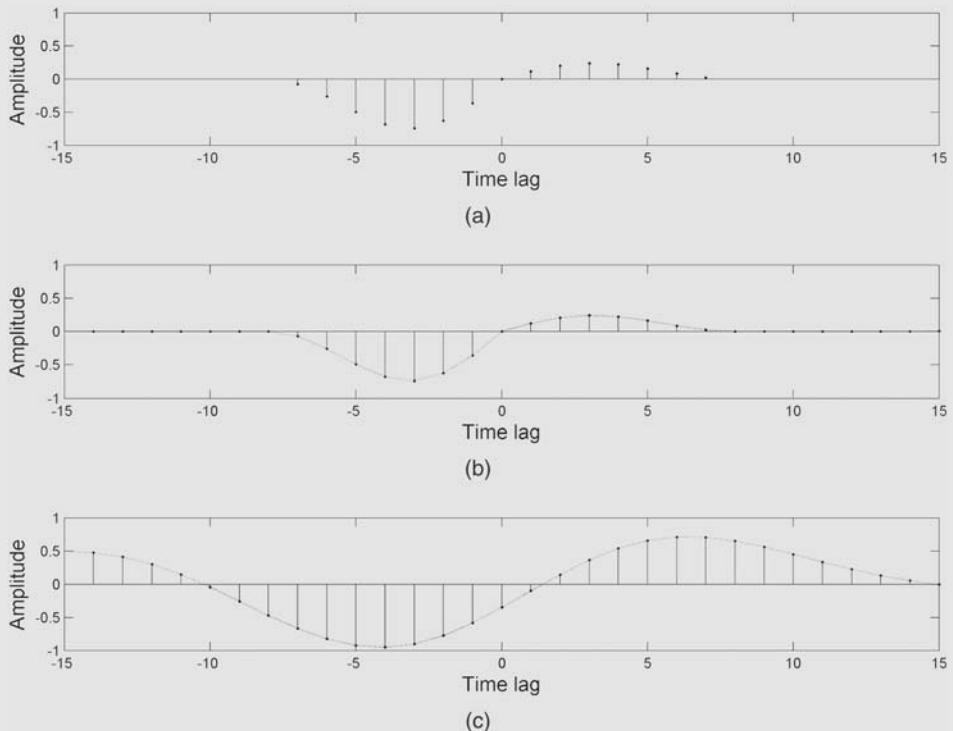


Figure 3.6 Subband orthogonality of DCT: (a) the system correlation sequence $q_{ip}(l)$, (b) the cross-correlation function $\gamma_{ip}(l)$ between the first and second subbands when the input signal is white and (c) the cross-correlation function $\gamma_{ip}(l)$ between the first and second subbands when the input signal is colored

```
%---Type-IV DCT-----
N = 8; L = N;
H = DCTmatrix_TypeIV(N);
%
%---Generate colored noise-----
a = [1.00,-1.32,0.86,-0.45,
      -0.62,0.98,-0.65,0.54,
      0.37,-0.51,0.37,-0.40,
      0.06,-0.15, 0.12,0.20]; % Speech LPC coefficients
randn('state',sum(100*clock));
wgn = randn(4*80000,1); % Generate white Gaussian noise
un = filter(1,a,wgn); % IIR filtering
un = un/std(un); % Make unit variance
%
%---Filter bank-----
k = 0; % Subband index, k = 0,1,...,N-1
i = k ; p = k+1 ; % Adjacent subbands i and p
H_i = H(i+1,:); H_p = H(p+1,:); % Analysis filters
q_ip = xcorr(H_i,H_p); % System correlation sequence

%---Cross-correlation estimation for white signal---
wgn_i = filter(H_i,1,wgn); % ith subband signal
wgn_p = filter(H_p,1,wgn); % pth subband signal
maxlag = 50; % Number of lags for correlation
r_wgn = xcorr(wgn_i,wgn_p,maxlag,'coef'); % estimation

%---Cross-correlation estimation for colored signal---
un_i = filter(H_i,1,un); % ith subband signal
un_p = filter(H_p,1,un); % pth subband signal
maxlag = 50; % Number of lags for correlation
r_un = xcorr(un_i,un_p,maxlag,'coef'); % Estimation
```

3.4 Case study: Subband orthogonality of cosine-modulated filter banks

This section uses the correlation-domain formulation presented in the previous section to analyze the second-order characteristics of cosine-modulated filter banks (see Section 2.5). In particular, we show that subband orthogonality can be obtained by the cosine modulation of a prototype lowpass filter with high stopband attenuation. This feature pertaining to the cosine-modulated filter banks plays an important role in the derivation and analysis of subband adaptive algorithms in Chapter 6.

3.4.1 Correlation-domain analysis

In an N -channel cosine-modulated filter bank, the analysis filters are cosine-modulated versions of a prototype lowpass filter $P(z)$ with a cutoff frequency of $\pi/2N$. The prototype filter $P(z)$ is generally restricted as a linear-phase filter with a symmetric

impulse response [1, 16, 18–20], i.e. $p(n) = p(L - 1 - n)$, so that its frequency response can be written as

$$P(e^{j\omega}) = e^{-j\omega(L-1)/2} P_R(\omega), \quad (3.24)$$

where $P_R(\omega)$ is the real-valued zero-phase response (or amplitude response) of the prototype filter. Evaluating Equation (2.31) on the unit circle and using Equation (3.24) in the expression, the frequency response of the cosine-modulated filter $H_i(z)$ can be formulated in terms of the zero-phase response $P_R(\omega)$ in the following form:

$$H_i(e^{j\omega}) = e^{-j\omega\left(\frac{L-1}{2}\right)} [e^{j\theta_i} U_i(\omega) + e^{-j\theta_i} V_i(\omega)], \quad (3.25)$$

where

$$U_i(\omega) = P_R\left[\omega - \frac{\pi}{N}(i + 0.5)\right] \text{ and } V_i(\omega) = P_R\left[\omega + \frac{\pi}{N}(i + 0.5)\right] \quad (3.26)$$

are the right- and left-shifted versions of the zero-phase response, respectively. Using Equations (3.25) and (3.26) in Equation (3.13), the cross-energy spectrum for the cosine-modulated filters can be written in terms of the shifted zero-phase responses as

$$\begin{aligned} Q_{ip}(e^{j\omega}) &= H_i(e^{j\omega}) [H_p(e^{j\omega})]^* \\ &= [e^{j\theta_i} U_i(\omega) + e^{-j\theta_i} V_i(\omega)] [e^{-j\theta_p} U_p(\omega) + e^{j\theta_p} V_p(\omega)] \\ &= e^{j(\theta_i - \theta_p)} U_i(\omega) U_p(\omega) + e^{-j(\theta_i - \theta_p)} V_i(\omega) V_p(\omega) \\ &\quad + e^{j(\theta_i + \theta_p)} U_i(\omega) V_p(\omega) + e^{-j(\theta_i + \theta_p)} V_i(\omega) U_p(\omega), \end{aligned} \quad (3.27)$$

where $p = i + 1$ and $i = 0, 1, \dots, N - 2$. Now, consider the cross-energy spectrum between adjacent subbands $Q_{i,i+1}(e^{j\omega})$. With the phase alignment factor θ_i in the modulation function as indicated in Equation (2.32), it can be easily seen that

$$\begin{aligned} \theta_i - \theta_{i+1} &= (-1)^i \frac{\pi}{4} - (-1)(-1)^i \frac{\pi}{4} = (-1)^i \frac{\pi}{2} \text{ and} \\ \theta_i + \theta_{i+1} &= (-1)^i \frac{\pi}{4} + (-1)(-1)^i \frac{\pi}{4} = 0. \end{aligned} \quad (3.28)$$

Substituting these results into Equation (3.27), the cross-energy spectrum between adjacent subbands can be written as

$$Q_{i,i+1}(e^{j\omega}) = S_{i,\text{even}}(e^{j\omega}) + j S_{i,\text{odd}}(e^{j\omega}), \quad (3.29)$$

where

$$\begin{aligned} S_{i,\text{even}}(e^{j\omega}) &= U_i(\omega) V_{i+1}(\omega) + V_i(\omega) U_{i+1}(\omega) \text{ and} \\ S_{i,\text{odd}}(e^{j\omega}) &= (-1)^i [U_i(\omega) U_{i+1}(\omega) - V_i(\omega) V_{i+1}(\omega)] \end{aligned} \quad (3.30)$$

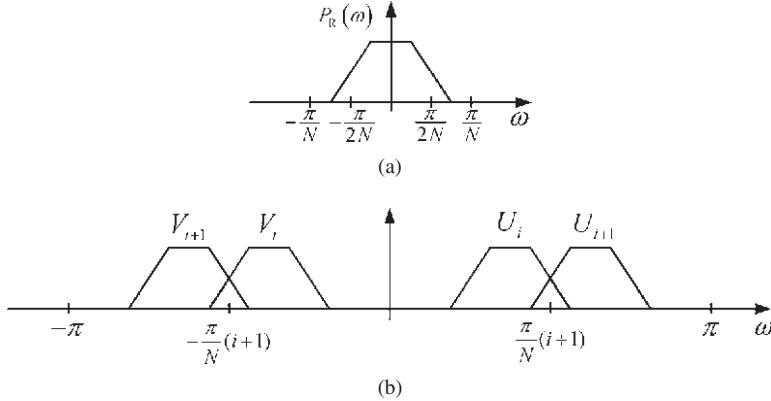


Figure 3.7 A typical prototype filter and its shifted amplitude responses (adapted from Reference [5]): (a) the amplitude response of the prototype filter and (b) overlaps of the shifted amplitude responses, which form a significant amount of the cross-energy spectrum

correspond to the symmetric and antisymmetric components of $q_{i,i+1}(l)$, respectively; i.e. for a real-valued sequence $q_{i,i+1}(l)$, its symmetric component transforms to $S_{i,\text{even}}(e^{j\omega})$, while its antisymmetric component transforms to $jS_{i,\text{odd}}(e^{j\omega})$.

Figure 3.7(a) shows a typical prototype filter with the stopband edge $\omega_s < \pi/N$. Figure 3.7(b) shows that the transition band of $U_i(\omega)$ extends to the frequency range of the adjacent $U_{i+1}(\omega)$. Similarly, significant overlap occurred between $V_i(\omega)$ and $V_{i+1}(\omega)$. These overlaps (i.e. their products) form a significant amount of the antisymmetric component $S_{i,\text{odd}}(e^{j\omega})$, as indicated by the second equation in (3.30). On the other hand, when there is high stopband attenuation, $U_i(\omega)$ and $V_{i+1}(\omega)$ do not overlap. $V_i(\omega)$ and $U_{i+1}(\omega)$ do not overlap either, which implies that the symmetric component $S_{i,\text{even}}(e^{j\omega})$ is negligible according to the first equation in (3.30).

Figure 3.8 illustrates the situation where the stopband edge ω_s of the prototype filter extends beyond π/N . Notice that undesirable overlaps, which contribute to the symmetric component of $S_{i,\text{even}}(e^{j\omega})$, may occur at $i = 0$ and $i = N - 2$, as shown in Figure 3.8(b). However, it is common to have the stopband edge ω_s of the prototype filter less than π/N ; i.e. the constraint

$$|P(e^{j\omega})| = 0, \text{ for } \omega > \pi/N, \quad (3.31)$$

is generally imposed on the prototype filter [1, 16, 18–20]. This constraint ensures that the spectral components from distant (nonadjacent) subbands are sufficiently attenuated, which at the same time annihilates the situation illustrated in Figure 3.8. Consequently, the antisymmetric component $S_{i,\text{odd}}(e^{j\omega})$ always dominates the cross-spectrum $Q_{i,i+1}(e^{j\omega})$. Thus, the cross-correlation sequence $q_{i,i+1}(l)$ can be assumed to be antisymmetric, which in turn ensures that the adjacent subband signals of a cosine-modulated filter bank are always orthogonal at zero lag, regardless of the excitation signal.

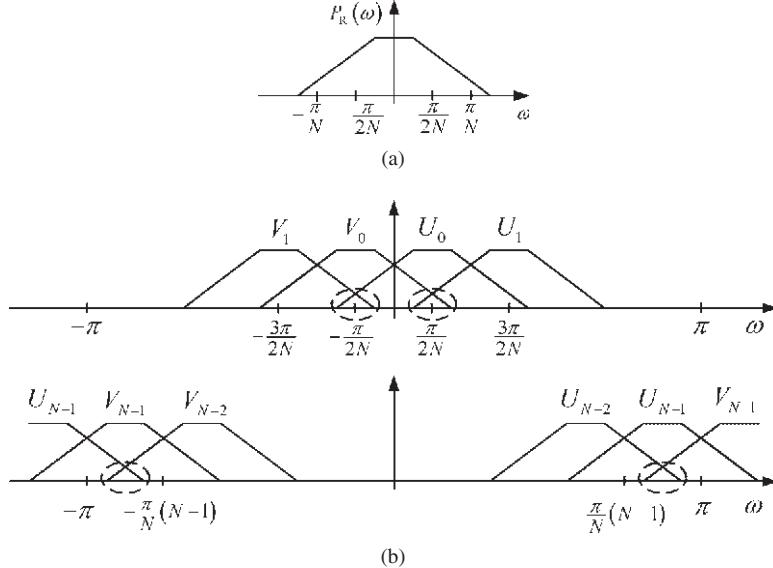


Figure 3.8 Stopband edge ω_s of the prototype filter extends beyond π/N (adapted from Reference [5]): (a) the amplitude response of the prototype filter and (b) the circled regions denoted undesirable overlaps that form the symmetric components $S_{0,\text{even}}(e^{j\omega})$ and $S_{N-1,\text{even}}(e^{j\omega})$

3.4.2 MATLAB simulations

This section uses computer simulations to verify the mathematical analysis of the second-order characteristics of an eight-channel pseudo-QMF cosine-modulated filter bank presented in the previous section. The MATLAB script is given in Example 3.5. The prototype filter length is $L = 128$ and the stopband edge is $\omega_s = \pi/8$. The magnitude responses of all the cosine-modulated analysis filters are plotted in Figure 3.9. Notice that the passband of analysis filters are normalized to 0 dB. The input signal to the analysis filter bank is a colored signal generated by filtering a white Gaussian noise with an autoregressive model of order $P = 15$. The input spectrum plotted in Figure 3.3(a) is given by

$$\Gamma_{uu}(e^{j\omega}) = \frac{\sigma_w^2}{\left|1 + \sum_{l=1}^P a_l e^{-j\omega l}\right|^2}, \quad (3.32)$$

where $\sigma_w^2 = 1$ is the variance of the white Gaussian noise and a_l are the coefficients of the autoregressive model. These coefficients are LPC coefficients of a 30 ms speech segment (sampled at 16 kHz) selected from the TIMIT speech corpus [21].

Figure 3.9 shows that spectral overlaps between nonadjacent filters are negligible due to high stopband attenuation. On the other hand, adjacent filters significantly overlap, implying that a considerable degree of correlation exists between them. The correlation between adjacent filters can be observed from their cross-correlation sequence $q_{i,i+1}(l)$ and the corresponding cross-energy spectrum $|Q_{i,i+1}(e^{j\omega})|$, as shown in Figures 3.10(a)

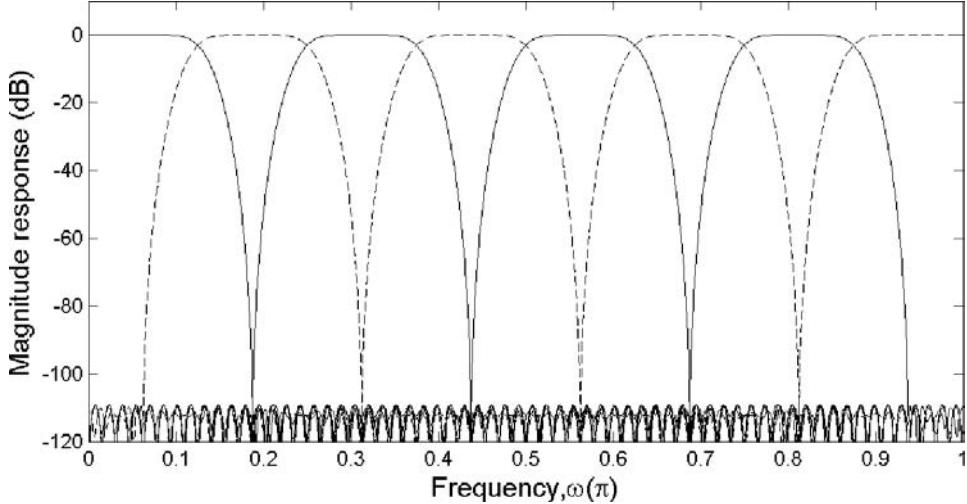


Figure 3.9 Magnitude responses of all the analysis filters in an eight-channel cosine-modulated filter bank (adapted from Reference [5])

and (b), respectively. Notice that the cross-correlation sequence $q_{i,i+1}(l)$ is characterized by periodic zero-crossing separated by $N = 8$ sampling periods. This characteristic implies that the pseudo-QMF design results in an analysis filter bank that nearly satisfies the paraunitary condition (3.21). As mentioned earlier, the paraunitary condition leads to a perfect analysis–synthesis system.

Figure 3.10(b) shows that the adjacent filters are correlated at the frequencies where their passbands overlap. These frequency components appear in the subband signals and thus lead to a high correlation between adjacent subband signals, as illustrated in Figure 3.10(c). The power spectrum shown in Figure 3.10(d) verifies that the subband signals are correlated at the frequencies where the responses of the analysis filter overlap. The normalized cross-correlation sequence shown in Figure 3.10(c) is estimated from the subband samples, $u_i(n)$ and $u_{i+1}(n)$ for $n = 0, 1, \dots, M - 1$, as follows:

$$\hat{\gamma}_{i,i+1}(l) = \frac{1}{\sqrt{\sigma_i^2 \sigma_{i+1}^2}} \left[\frac{1}{M - |l|} \sum_{n=0}^{M-1} u_i(n) u_{i+1}(n - l) \right], \quad (3.33)$$

where M is the length of the data samples used in the estimation and σ_i^2 denotes the variance of the subband signal. A very long data record is used for the estimation since the subband signals are wide-sense stationary. Notice that the purpose of normalization is to have $\hat{\gamma}_{i,i+1}(l) \leq 1$ so that the estimate is independent of scaling of the signals. The cross-correlation estimate $\hat{\gamma}_{i,i+1}(l)$ is truncated by a rectangular window to retain only the cross-correlation coefficients for lags $|l| \leq L$, where $L = 150$ in this case. The magnitude spectrum $|\hat{\Gamma}_{i,i+1}(e^{j\omega})|$, as shown in Figure 3.10(d), is obtained by taking the Fourier transform of the windowed cross-correlation estimate.

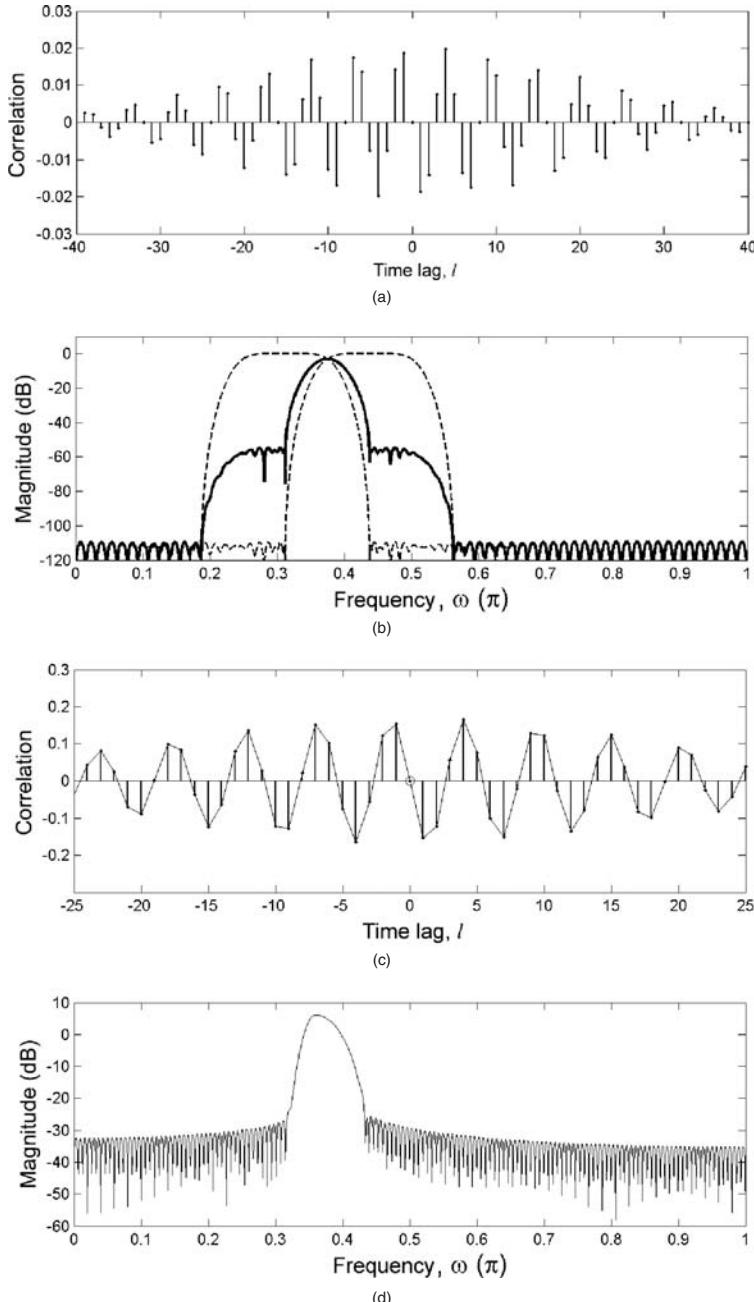


Figure 3.10 Cross-correlation between adjacent bands for a randomly selected case $i = 2$ (adapted from Reference [5]): (a) the deterministic cross-correlation sequence $q_{i,i+1}(l)$ between adjacent filters, (b) the cross-energy spectrum $|Q_{i,i+1}(e^{j\omega})| = |H_i(e^{j\omega})| |H_{i+1}(e^{j\omega})|$, where the dotted lines represent the magnitude responses $|H_i(e^{j\omega})|^2$, (c) the normalized cross-correlation sequence between adjacent subband signals and (d) the estimated cross-power spectrum

Example 3.5

This example examines the subband orthogonality of the cosine-modulated filter bank. The MATLAB code given below decomposes the cross-energy spectrum into odd and even components. The decomposition is performed according to Equations (3.26) and (3.30). The complete listing of the code, which includes time- and frequency-domain correlation analysis, can be found in M-file `Example_3P5.m` in the companion CD.

```
%---Design filter bank-----
N = 8; % Number of subbands
K = 8; L=2*K*N; % Length of lowpass prototype filter
[hopt,passedge] = opt_filter(L-1,N); % Prototype filter design
H = make_bank(hopt,N); % Filter bank generation

%---Ui(w) and Vi(w), the shifted zero-phase responses
ShiftFreq = -(pi/N)*(i + 0.5); % Right shifting, Ui(w)
[Ui,w,theta] = AmpResp(hopt,DFTpoint,2,ShiftFreq);
ShiftFreq = (pi/N)*(i + 0.5); % Left shifting, Vi(w)
[Vi,w,theta] = AmpResp(hopt,DFTpoint,2,ShiftFreq);

%---Up(w) and Vp(w), the shifted zero-phase responses
ShiftFreq = -(pi/N)*(p + 0.5); % Right shifting, Up(w)
[Up,w,theta] = AmpResp(hopt,DFTpoint,2,ShiftFreq);
ShiftFreq = (pi/N)*(p + 0.5); % Left shifting, Vp(w)
[Vp,w,theta] = AmpResp(hopt,DFTpoint,2,ShiftFreq);

%---Cross-energy spectrum, Qip(w)
% S_even and S_odd are real-valued functions of w, which
% represent the real and imaginary parts of the
% cross-energy spectrum
if mod(abs(i-p),2)~=0 % abs(i-p), an odd number
    S_odd = ((-1)^i)*(Ui.*Up - Vi.*Vp);
    S_even = Ui.*Vp + Vi.*Up;
else % abs(i-p), an even number
    S_odd = ((-1)^i)*(Ui.*Vp - Vi.*Up);
    S_even = Ui.*Up + Vi.*Vp;
end
```

As pointed out earlier, the cross-energy spectrum $Q_{i,i+1}(e^{j\omega})$ can be decomposed into symmetric and antisymmetric components $S_{i,\text{even}}(e^{j\omega})$ and $S_{i,\text{odd}}(e^{j\omega})$, as shown in Figure 3.11. It should be emphasized that $S_{i,\text{even}}(e^{j\omega})$ and $S_{i,\text{odd}}(e^{j\omega})$ are real-valued functions of ω . As expected, the symmetric component is small at all frequencies as compared to the antisymmetric component. That is because the cross-energy spectrum $Q_{i,i+1}(e^{j\omega})$ is dominated by $S_{i,\text{odd}}(e^{j\omega})$ and thus the cross-correlation sequence $q_{i,i+1}(l)$ of Figure 3.10(a) can be assumed to be antisymmetric. The antisymmetry of $q_{i,i+1}(l)$ ensures that adjacent subband signals are nearly orthogonal at zero lag, i.e. $\gamma_{i,i+1}(0) \approx 0$, as depicted in Figure 3.10(c) for the colored signal. The absolute values of the normalized

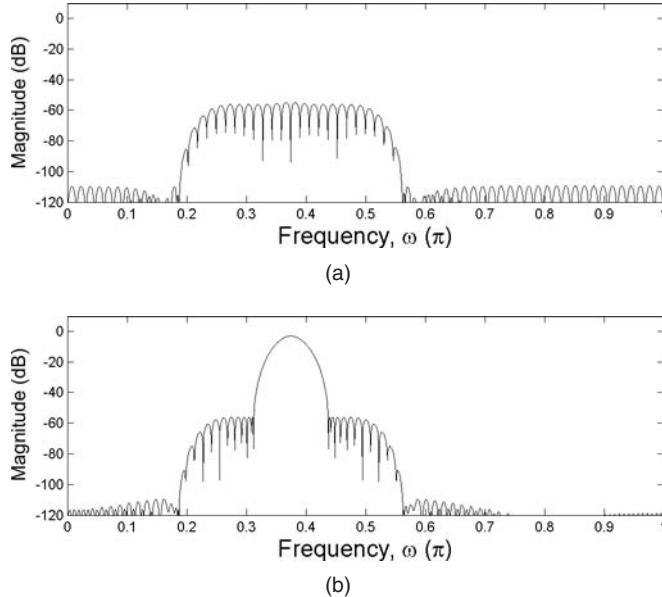


Figure 3.11 Decomposition of the cross-energy spectrum $Q_{i,i+1}(e^{j\omega})$ between two adjacent filters into symmetric and antisymmetric components for a randomly selected case $i = 2$: (a) absolute value of the symmetric component $|S_{i,even}(e^{j\omega})|$ and (b) absolute value of the antisymmetric component $|S_{i,odd}(e^{j\omega})|$

cross-correlation coefficients, $|\hat{\gamma}_{i,i+1}(0)|$ for $i = 0, 1, \dots, N - 2$, are found to be smaller than 0.2×10^{-4} . Note that the cross-correlation may be large for lags where $l \neq 0$.

3.5 Summary

This chapter investigated the second-order characteristics of multirate filter banks. Using the correlation-domain formulation, the effect of filtering a random signal with a filter bank can be described in terms of the effect of the system on the autocorrelation function of the input signal. The outputs of a filter bank are orthogonal if they occupy nonoverlapping portions of the input spectrum. Such a complete separation of subbands may be desirable, but it is not achievable in practice because realizable filters have a nonzero transition band and finite stopband attenuation. Furthermore, it may not be necessary in applications where partial decorrelation is sufficient.

A special case of partial decorrelation was defined where the subband signals are orthogonal at zero lag. The subband signals are orthogonal at zero lag if the deterministic cross-correlation sequence is antisymmetric. This feature can be obtained by manipulating the relative phase between the analysis filters. In the case study given in Section 3.4, orthogonality at zero lag can be approximated by cosine modulation of a prototype filter with high stopband attenuation. In other words, the outputs of a cosine-modulated filter

bank are nearly orthogonal at zero lag for arbitrary input signals as long as the stopband attenuation of the prototype filter is sufficiently high.

References

- [1] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Englewood Cliffs, New Jersey: Prentice Hall, 1993.
- [2] V. K. Ingle and J. G. Proakis, *Digital Signal Processing using MATLAB*, Pacific Grove, California: Brooks/Cole, 2000.
- [3] A. V. Oppenheim, R. W. Schafer and J. R. Buck, *Discrete-Time Signal Processing*, Upper Saddle River, New Jersey: Prentice Hall, 1999.
- [4] J. G. Proakis and M. G. Manolakis, *Digital Signal Processing – Principles, Algorithms, and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 1996.
- [5] K. A. Lee and W. S. Gan, ‘On the subband orthogonality of cosine-modulated filter banks’, *IEEE Trans. Circuits Syst. II*, **53**(8), August 2006, 677–681.
- [6] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.
- [7] S. Haykin, *Adaptive Filter Theory*, 4th edition, Upper Saddle River, New Jersey: Prentice Hall, 2002.
- [8] S. M. Kuo and B. H. Lee, *Real-Time Digital Signal Processing*, New York: John Wiley & Sons, Inc., 2001.
- [9] D. G. Manolakis, V. K. Ingle and S. M. Kogon, *Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering and Array Processing*, New York: McGraw-Hill, 2000.
- [10] C. W. Therrien, ‘Overview of statistical signal processing’, in *Digital Signal Processing Handbook* (eds V. K. Madisetti and D. B. Williams), Boca Raton, Florida: CRC Press, 1999.
- [11] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*, Boston, Massachusetts: McGraw-Hill, 2002.
- [12] C. W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, Englewood Cliffs, New Jersey: Prentice Hall, 1992.
- [13] B. Farhang-Boroujeny and S. Gazor, ‘Selection of orthonormal transforms for improving the performance of the transform domain normalized LMS algorithm’, *IEE Proc., Part F*, **139**(5), October 1992, 327–335.
- [14] K. A. Lee and W. S. Gan, ‘Improving convergence of the NLMS algorithm using constrained subband updates’, *IEEE Signal Processing Lett.*, **11**(9), September 2004, 736–739.
- [15] K. A. Lee, W. S. Gan and Y. Wen, ‘Subband adaptive filtering using a multiple-constraint optimization criterion’, in *Proc. EUSIPCO*, 2004, pp. 1825–1828.
- [16] H. S. Malvar, *Signal Processing with Lapped Transform*, Norwood, Massachusetts: Artech House, 1992.
- [17] R. Merched, P. S. R. Diniz and M. R. Petraglia, ‘A new delayless subband adaptive filter structure’, *IEEE Trans. Signal Processing*, **47**(6), June 1999, 1580–1591.
- [18] C. D. Creusere and S. K. Mitra, ‘A simple method for designing high-quality prototype filters for M -band pseudo-QMF banks’, *IEEE Trans. Signal Processing*, **43**(4), April 1995, 1005–1007.

- [19] T. Q. Nguyen, ‘Near-perfect-reconstruction pseudo-QMF banks’, *IEEE Trans. Signal Processing*, **42**(2), March 1994, 65–76.
- [20] T. Saramäki and R. Bregović, ‘Multirate systems and filter banks’, in *Multirate Systems: Design and Applications* (ed. G. Jovanovic-Dolecek), Hershey, Pennsylvania: Idea Group, 2002.
- [21] *The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus*, National Institute of Standards and Technology, October 1990.

4

Subband adaptive filters

A filter bank partitions the input signal into multiple subbands that occupy contiguous portions of the frequency band. With the pair of perfect-reconstruction analysis–synthesis filter banks described in Chapter 2 that satisfies a certain structural relationship [1–5], a fullband signal can be reconstructed with minimum distortion at the output of the subband system.

The idea of subband filtering evolved primarily from subband coding [6] and has been extended to subband adaptive filtering in References [7] to [10]. In a subband adaptive filter (SAF), the input signal is decomposed into multiple parallel channels. This feature exploits subband properties to facilitate a more effective signal processing. In addition, a subband system also reduces computational complexity since subband signals can be processed with lower-order subfilters at a lower decimated rate commensurate with their narrower bandwidths.

A number of adaptive filtering structures based on subband and multirate techniques have been proposed in the literature [11–19]. This chapter introduces the fundamental principles of subband adaptive filtering. The major problems and some methods for solving them are addressed. The concept of delayless subband adaptive filtering [18, 20] is also presented in this chapter. Finally, the subband adaptive filtering techniques are demonstrated through a practical application – acoustic echo cancellation.

4.1 Subband adaptive filtering

Figure 4.1 shows the conventional SAF structure [7–10, 21, 22] for an application of adaptive system identification. The fullband input signal $u(n)$ and desired response $d(n)$ are decomposed into N spectral bands using analysis filters $H_i(z)$, $i = 0, 1, \dots, N - 1$. These subband signals are decimated to a lower rate using the same factor D and are processed by individual adaptive subfilters $W_i(z)$. Each subfilter has an independent adaptation loop with a locally computed error signal $e_{i,D}(k)$ for updating the tap weights to minimize the corresponding subband error signal. The fullband error signal $e(n)$ is finally obtained by interpolating and recombining all the subband error signals using a

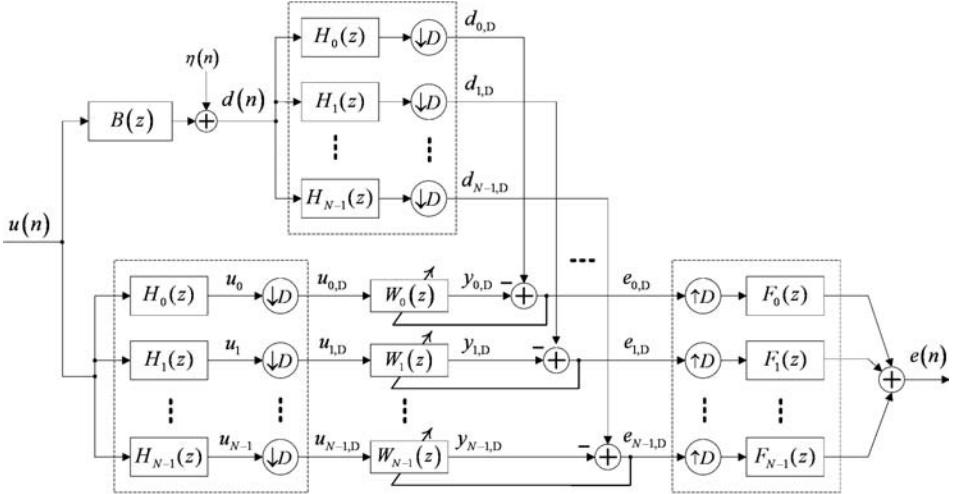


Figure 4.1 A conventional SAF using a set of adaptive subfilters to identify an unknown system $B(z)$

synthesis filter bank. Notice that the variable n is the time index for the fullband signals and k is the time index for the decimated subband signals.

4.1.1 Computational reduction

The primary motivations of using SAF are to reduce the computational complexity and improve the convergence performance for the input signal with a large spectral dynamic range, i.e. the input signal is highly correlated [7, 9, 10]. In Figure 4.1, adaptive subfilters $W_0(z), W_1(z), \dots, W_{N-1}(z)$ are used to model a fullband unknown system $B(z)$. Compared with the equivalent fullband adaptive filters introduced in Chapter 1, computational savings can be obtained by using adaptive subfilters with a shorter length that operate at a lower decimated rate. In addition, the subband signals have a flatter spectrum (i.e. a smaller spectral dynamic range), resulting in faster convergence for the gradient-based adaptive algorithms introduced in Section 1.4.

Consider an application where the unknown system $B(z)$ is modeled by a fullband transversal filter $W(z)$ of length M . Since the subfilters $W_i(z)$ shown in Figure 4.1 operate at a lower rate with the decimation factor D , the length of the adaptive subfilters can be reduced to $M_S = M/D$. There are a total of $M_S \times N = MN/D$ subband tap weights for N subbands. Thus the number of multiplications required by N subfilters is MN/D^2 . Compared with the fullband filtering that requires M multiplications, the computational saving in terms of multiplications can be expressed as

$$\frac{\text{Complexity of fullband filter}}{\text{Complexity of subfilters}} = \frac{D^2}{N}, \quad (4.1)$$

where $D = N$ is used for a critically sampled SAF to achieve the greatest computational savings and $D < N$ for an oversampled SAF to minimize the aliasing distortion. Notice

that the result given in Equation (4.1) does not include the overhead incurred by the filter banks and the adaptation of tap weights. Equation (4.1) serves as a general guideline that indicates the computational savings achieved using subband adaptive filtering as compared with fullband filters.

We have assumed that the analysis filter bank is real valued to derive Equation (4.1). For the DFT filter banks introduced in Chapter 2, the subband signals are complex valued even though the input signal is real valued. We only process the first $N/2 + 1$ subbands (assuming that N is an even number) instead of the N subbands because the analysis filters appear in complex-conjugate pairs except for the two real-valued filters at $i = 0$ and $i = N/2$. Since each complex multiplication needs four real multiplications, the number of multiplications required by N adaptive subfilters is $2MN/D^2$. The computational saving achieved with complex-valued filter banks is only half of its real-valued counterpart. Nevertheless, both the real- and complex-valued filter banks are widely used in subband applications, depending on the signal processing algorithms.

4.1.2 Spectral dynamic range

As discussed in Chapter 1, gradient-based algorithms (e.g. LMS and NLMS) suffer from slow convergence when the input signal is highly correlated. This problem is related to the large eigenvalue spread in the input autocorrelation matrix due to the large spectral dynamic range of the input signal [21, 23–25]. As discussed in Chapter 3, subband decomposition followed by a critical decimation ($D = N$) reduces the spectral dynamic range in each subband. Figure 4.2 shows the power spectrum $\Gamma_{uu}(e^{j\omega})$ of an AR(2) (second-order autoregressive) signal and the magnitude responses $|H_i(e^{j\omega})|$ of a four-channel cosine-modulated filter bank. The length of the analysis filters is $L = 64$.

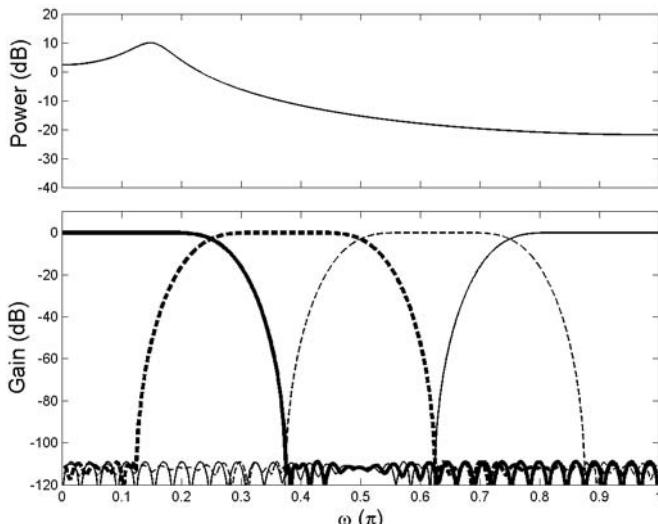


Figure 4.2 Power spectrum of an AR(2) signal (upper panel) and the magnitude responses of analysis filters (lower panel) for a four-channel pseudo-QMF cosine-modulated filter bank

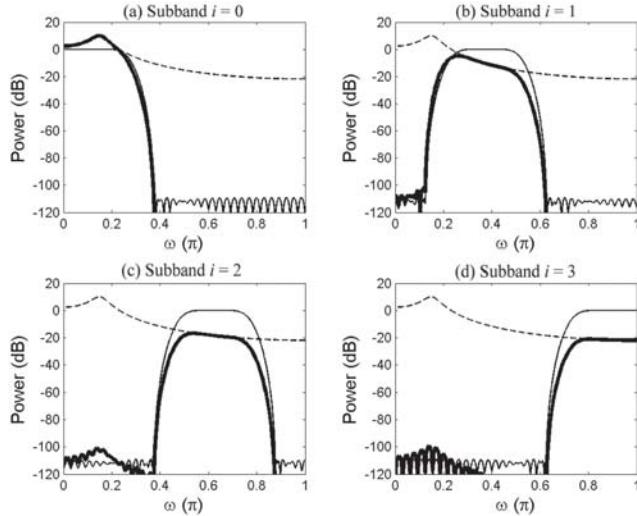


Figure 4.3 Power spectra of the subband signals (dark lines) before decimation. The thin-dashed lines represent the fullband spectrum and the thin-solid lines are squared magnitude responses of analysis filters

Using the correlation-domain formulation defined in Section 3.1, the power spectrum of the subband signal $u_i(n)$ can be expressed as

$$\Gamma_{ii}(e^{j\omega}) = |H_i(e^{j\omega})|^2 \Gamma_{uu}(e^{j\omega}) \text{ for } i = 0, 1, \dots, N-1. \quad (4.2)$$

This equation shows that the power spectrum $\Gamma_{ii}(e^{j\omega})$ is the product of the signal power spectrum $\Gamma_{uu}(e^{j\omega})$ and the squared magnitude response $|H_i(e^{j\omega})|^2$ of the corresponding analysis filter.

The power spectra of subband signals and the associated squared magnitude responses of analysis filters are depicted in Figure 4.3 using the four-channel filter bank shown in Figure 4.2. Notice that the subband signals $u_0(n), u_1(n), \dots, u_{N-1}(n)$ have a narrower bandwidth than the original fullband signal $u(n)$. The frequency contents of subband signals concentrate on the frequency range corresponding to the passband of the analysis filter. Because the autocorrelation function of the decimated subband signal $u_{i,D}(k)$ is the decimated version of the autocorrelation function of the subband signal $u_i(n)$, the power spectrum $\Gamma_{ii,D}(e^{j\omega})$ of $u_{i,D}(k)$ can be obtained from Equation (4.2) as

$$\Gamma_{ii,D}(e^{j\omega}) = \frac{1}{N} \sum_{l=0}^{N-1} \Gamma_{ii}(e^{j(\omega-2\pi l)/N}). \quad (4.3)$$

Decimating the subband signals to a lower rate commensurate with their bandwidth results in a reduced spectral dynamic range. The idea is demonstrated by the MATLAB script given in Example 4.1. As shown in Figure 4.4, the spectrum $\Gamma_{ii,D}(e^{j\omega})$ of the subband signal after critical decimation is closer to that of white noise. As discussed in

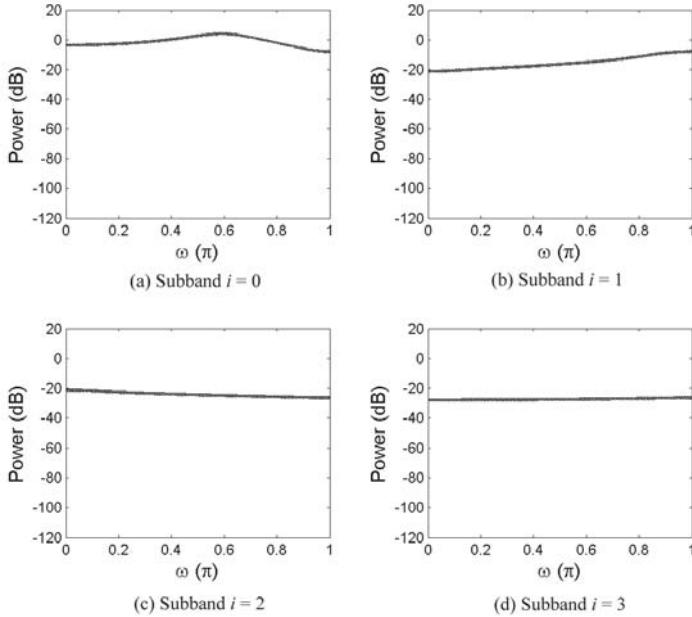


Figure 4.4 Power spectra of the subband signals after critical decimation

Section 1.5, faster convergence can be achieved because the spectral dynamic range of the subband signal is smaller than that of the fullband signal. Furthermore, the length of the adaptive subfilters is shorter than the fullband filter, which allows a larger step size to be used to increase the convergence rate further [26, 27]. However, Section 4.3 will show that aliasing and band-edge effects [8, 23, 28–31] degrade the convergence performance of conventional SAF structures, as shown in Figure 4.1.

Example 4.1 The spectral dynamic range of a random signal is determined by the maximum and minimum of its power spectrum. As discussed earlier in Section 1.5, the spectral dynamic range of the input signal serves as the upper bound for the eigenvalue spread of the input correlation matrix. The convergence rate of the LMS algorithm deteriorates as the spectral dynamic range increases. In this example, we examine the spectral dynamic range of a random signal before and after (i) subband decomposition and (ii) sampling rate decimation. Given the power spectrum of the input signal (denoted as `HzSig` in the M-file), the spectral dynamic range can be computed using the following line of code:

```
SDR = max(10*log10(HzSig)) - min(10*log10(HzSig));
```

A higher spectral dynamic range occurred when a signal has low-power frequency components at some frequency regions. Consequently, the convergence rates of

the adaptive filter decreased at these regions as compared to other frequencies corresponding to the high peaks of the input spectrum. The complete listing of the code can be found in M-file `Example_4P1.m`.

4.2 Subband adaptive filter structures

The main idea of subband adaptive filtering is to decompose a high-order adaptive filtering task using a set of low-order adaptive subfilters. SAF structures can be classified as either open-loop or closed-loop structures depending on the derivation of error signal.

4.2.1 Open-loop structures

In Figure 4.1, the input signal $u(n)$ and desired response $d(n)$ are partitioned into subband signals by the identical analysis filter bank, and then decimated by the same factor D . In each subband, the decimated input signal is filtered by an adaptive subfilter and the output is compared with the corresponding desired response. The subband error signal (i.e. the difference between the subband desired response and the corresponding subfilter output) is used to update the associated adaptive subfilter. These subband error signals are used to form the fullband error signal $e(n)$ using a synthesis filter bank. This type of SAF structure is called the open-loop structure since the adaptation of the subfilters is independent of the fullband error signal.

The open-loop structure independently adapts an individual subfilter in its own adaptation loop to minimize the respective local MSE in each subband. They operate independently without sharing information (input and error signals) with other subfilters between subbands. Instead of using the global MSE defined in Equation (1.5), the objective (or performance) function of the open-loop structure is the sum of all subband MSEs, expressed as

$$J_{\text{SB}} = \sum_{i=0}^{N-1} E \left\{ |e_{i,D}(k)|^2 \right\}, \quad (4.4)$$

where the operator $|\cdot|$ returns the absolute value of its argument since the subband error signals could be complex valued if the DFT filter banks introduced in Section 2.6 are used in the subband decomposition. Because the filter bank is not using ideal analysis filters to produce mutually exclusive subband signals, minimization of the objective function defined in Equation (4.4) based on the subband error signals generally leads to higher levels of fullband MSE. Therefore, the unknown system might not be accurately identified from the system identification perspective.

4.2.2 Closed-loop structures

Figure 4.5 shows a closed-loop structure for subband adaptive filtering. Similar to the open-loop structure shown in Figure 4.1, filtering of the input signal $u(n)$ is performed

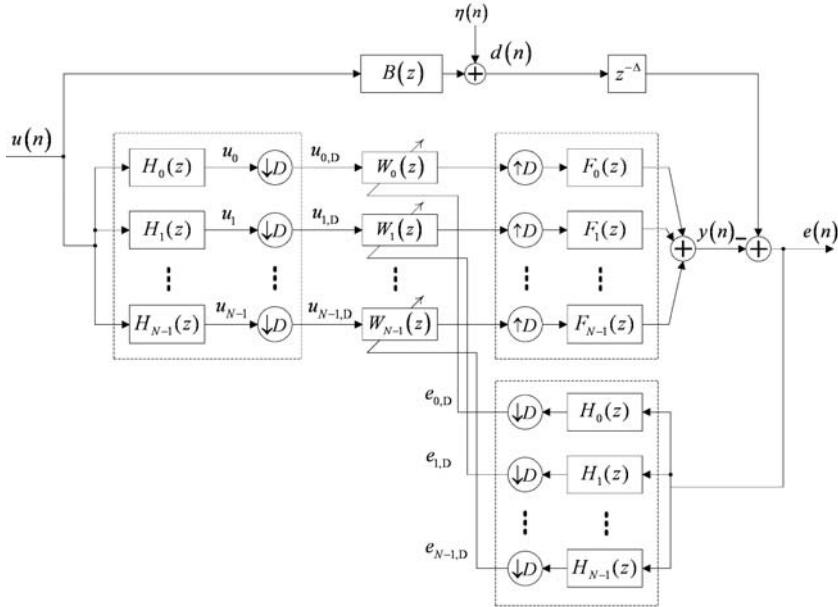


Figure 4.5 A closed-loop structure for subband adaptive filtering

in subbands. The fullband output $y(n)$ is reconstructed using a synthesis filter bank and then compared with the delayed version of the desired response $d(n)$ for generating the error signal $e(n)$. This fullband error signal is partitioned into subbands using the same analysis filter bank, and the decimated subband error signals are used for the adaptation of the respective adaptive subfilters. Notice that the desired response $d(n)$, which is the output of the unknown system, is delayed by Δ samples to account for the combined delay due to the analysis and synthesis filter banks along the input signal path. The delay value is given by [23]

$$\Delta = \left\lfloor \frac{L - 1}{D} \right\rfloor \times D, \quad (4.5)$$

where L is the length of the analysis filters and $\lfloor \cdot \rfloor$ denotes the integer part of its argument.

The closed-loop structure minimizes the fullband MSE as opposed to the sum of subband MSEs in the open-loop counterpart. By using the closed-loop feedback, nonzero frequency components in the fullband error signal will be fed back to update all of the subband adaptive subfilters to minimize those components. Thus the closed-loop structure allows minimization of the fullband error signal and guarantees that the overall adaptive subfilters converge to the optimal Wiener solution. However, in addition to the delay that we purposely impose on the desired response, the combination of the synthesis filter bank applied to the output signal and the analysis filter bank applied to the error signal also incur undesirable delay. This delay reduces the upper bound of the step size; thus the closed-loop structure is required to use a smaller step size, which results in slower convergence.

4.3 Aliasing, band-edge effects and solutions

As shown in Section 4.1, critical decimation decreases the spectral dynamic range in each subband and accelerates the convergence of adaptive subfilters. However, decimation introduces aliasing components into the subband signals. These aliasing components act as additional noise, which disturbs the adaptation process, thus resulting in a higher level of asymptotic MSE in the reconstructed fullband signal. As discussed in Chapter 2, aliasing is unavoidable in individual subbands even though the aliasing components can be collectively cancelled by the synthesis filter bank with perfect-reconstruction analysis–synthesis filters.

In order to reduce the aliasing effects in the subband systems, oversampling structures ($D < N$) are often used to replace critically sampled structures ($D = N$). Notable success has been achieved by using an oversampled DFT filter bank [28, 32–34]. The complex-valued subband signals are decimated with a higher sampling rate (i.e. oversampled) than is required by the signal bandwidth. This oversampling structure reduces severe aliasing components. Nevertheless, oversampling of subband signals produces some small eigenvalues in the subband autocorrelation matrices due to the band edges of the analysis filters. This phenomenon is called the band-edge effect [31]. From the adaptive filtering perspective, the band-edge effect can be observed from the MSE learning curve as a slow asymptotic convergence.

A number of techniques have been developed to overcome the aliasing effects encountered in the critically sampled SAFs; they are summarized as follows:

- (i) Reducing overlap of analysis filters by creating spectral gaps between adjacent subbands [35]. This approach introduces spectral holes located at the edges of the subbands in the output of the adaptive filter. In acoustic echo cancellation, the error signal contains local speech, which will be sent to the far-end receiver. These spectral gaps will degrade signal quality, especially when the number of subbands is large; thus this approach is unacceptable in practice [8, 30].
- (ii) Incorporating adaptive cross-filters between adjacent subbands to compensate for the aliasing effects [8, 30]. These cross-filters process the input signal from adjacent subbands to compensate for the aliasing components due to the critical decimation. Section 4.3.2 will show that the adaptive cross-filters can improve the accuracy of system identification.
- (iii) Employing a multiband-structured SAF (MSAF) [11, 12, 14, 15, 19, 36–38]. The aliasing effects are caused by partitioning the fullband adaptive filter into individual adaptive subfilters in each subband. In the multiband structure, the adaptive filter is no longer partitioned into subfilters. Instead, subband signals are used to adapt tap weights of the fullband adaptive filter. This algorithm will be discussed further in Section 4.3.3 and detailed in Chapter 6.
- (iv) Employing a closed-loop delayless structure. The aliasing effects can be reduced using a closed-loop subband structure that minimizes the fullband MSE instead of individual subband MSEs [13, 18, 39]. The capability of delayless closed-loop structure in eliminating the aliasing effects will be investigated in Section 4.3.4.

4.3.1 Aliasing and band-edge effects

Figure 4.6 depicts the power spectrum $\Gamma_{uu}(e^{j\omega})$ of the AR(2) signal (top panel) and the magnitude responses $|H_i(e^{j\omega})|$ of the four-channel DFT filter bank (bottom panel) over the frequency range of $0 \leq \omega \leq 2\pi$. The prototype filter $P(z)$ of length $L = 128$ is designed using the MATLAB function `fir1(127, 1/4)`. The analysis filters $H_i(z)$ are then generated by exponential modulation of $P(z)$, i.e. $H_i(z) = P(ze^{-j2\pi i/N})$ for $i = 0, 1, \dots, N - 1$ (see Section 2.6 for details of DFT filter banks). In general, the analysis filters $H_i(z)$ are complex valued; hence $|H_i(e^{j\omega})|$ is not symmetric with respect to $\omega = \pi$ and $\omega = 0$. If N is an even number, there is a lowpass filter for $i = 0$ and a highpass filter for $i = N/2$. As mentioned earlier, these two filters have real-valued coefficients and are symmetric with respect to $\omega = \pi$ (see Figure 4.6).

The analysis filters $H_i(z)$ partition the input signal into four ($N = 4$) spectral bands. The subband signals are then decimated with a factor of $D = N/2 = 2$, which is two times ($2\times$) higher than the critical sampling rate. The resulting subband spectra using Equations (4.2) and (4.3) are depicted in Figure 4.7. Notice that the subband spectrum exhibits a large spectral dynamic range at the band edges, which causes an ill-conditioned subband autocorrelation matrix, thereby degrading the overall convergence rate. Figure 4.8 shows the spectra of the subband signals for a critically sampled subband structure with $D = N = 4$. Clearly, the band edges are reduced by the critical decimation operation. However, the subband signals contain undesired aliasing components (indicated by the thin lines), which will degrade the convergence of adaptive subfilters. Detailed analysis is illustrated in M-file `Example_4P2.m`, which can be found in the companion CD.

Both oversampled and critically sampled subband adaptive filters are widely used in practice with some modifications to overcome the band-edge and aliasing effects. We

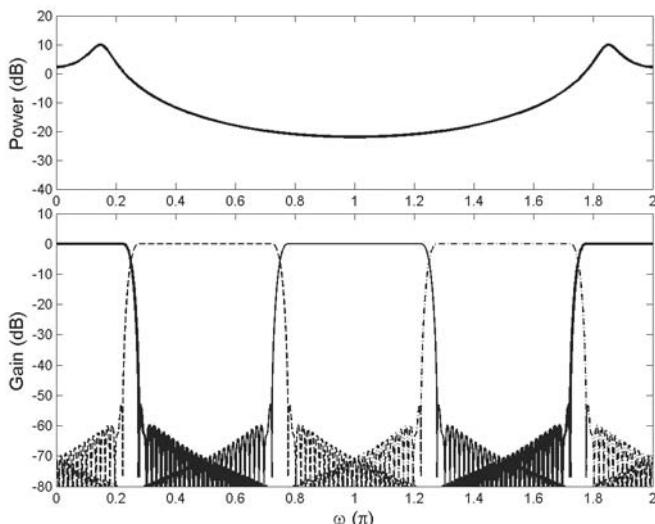


Figure 4.6 Power spectrum of the AR(2) signal (top panel) and the magnitude responses of analysis filters in a four-channel uniform DFT filter bank (bottom panel)

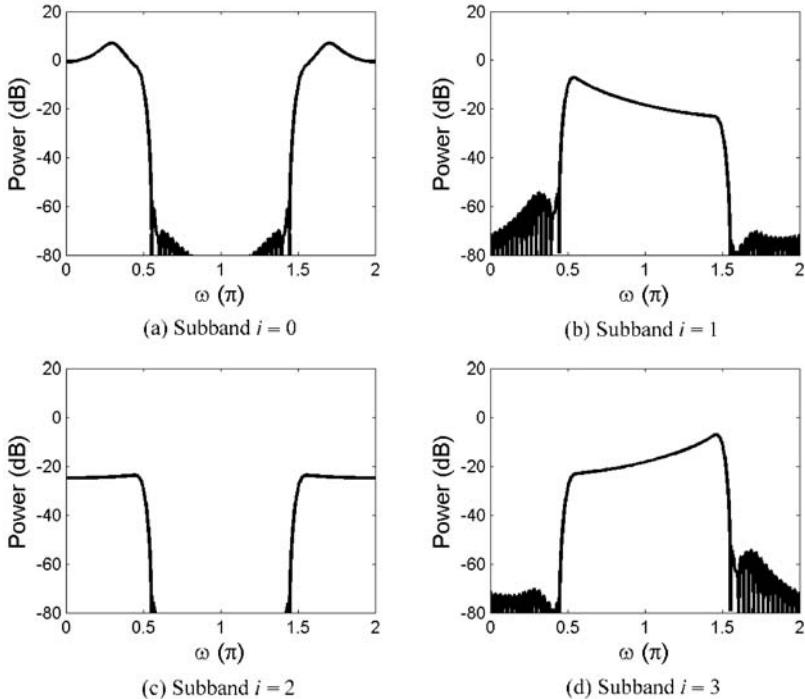


Figure 4.7 Spectra of the subband signals in a $2 \times$ oversampled SAF using a DFT filter bank. Band edges of subband filters introduce small eigenvalues into the subband autocorrelation matrix

focus on the critically sampled structures in this chapter and will discuss the oversampled structures in Chapter 5.

4.3.2 Adaptive cross filters

The adaptive system identification using SAF [8, 16, 17, 30] is illustrated in Figure 4.9, where a matrix of subfilters, $\mathbf{W}(z) = [W_{i,j}(z)]$ for $i, j = 0, 1, \dots, N-1$, are used to model the unknown system $B(z)$ using a critically sampled subband structure. Each element of the model matrix $\mathbf{W}(z)$ represents the transfer function of the adaptive subfilter. The diagonal elements $W_{i,i}(z)$ of the model matrix correspond to the adaptive subfilters $W_i(z)$ shown in Figure 4.1, whereas the off-diagonal elements, $W_{i,j}(z)$ for $j \neq i$, are the adaptive cross-filters between subbands. The model matrix $\mathbf{W}(z)$ generates the output signals $y_{i,D}(k)$ to estimate the desired response components $d_{i,D}(k)$ using the input subband signals $u_{i,D}(k)$. The adaptive cross filter $W_{i,j}(z)$ processes the signal $u_{j,D}(k)$ from other subbands to compensate for the aliasing components in $u_{i,D}(k)$. In this configuration, minimizing the sum of subband MSEs $E\{e_{i,D}^2(k)\}$ is equivalent to minimizing the fullband MSE $E\{e^2(n)\}$, thus resulting in accurate identification of the unknown system. Unfortunately, these cross filters increase computational complexity. Furthermore, they converge slowly to the optimal solution because the subband signals used in adaptation

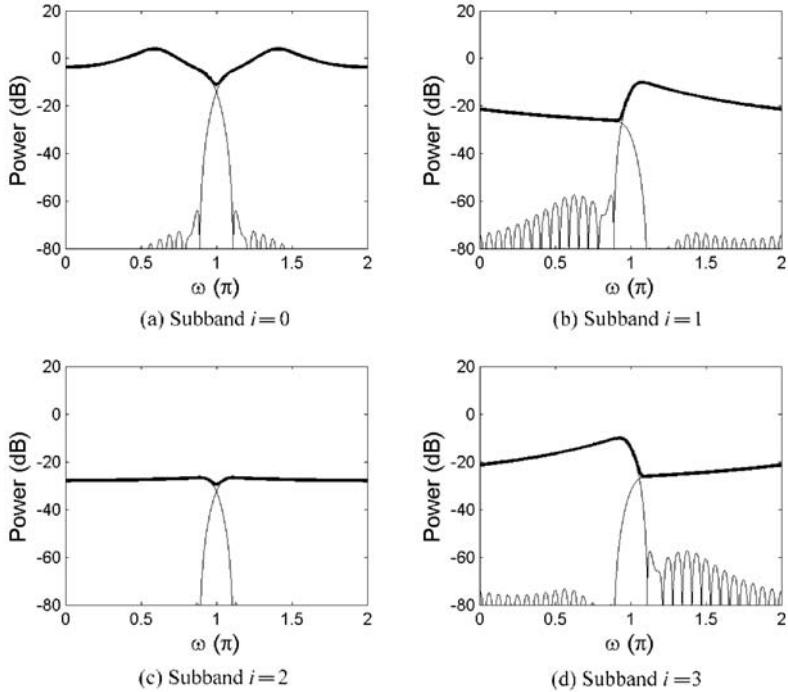


Figure 4.8 Spectra of the subband signals in a critically sampled SAF using a DFT filter bank. Critical decimation reduces the band edges and, at the same time, distorts the subband signals with aliasing (indicated by the thin lines)

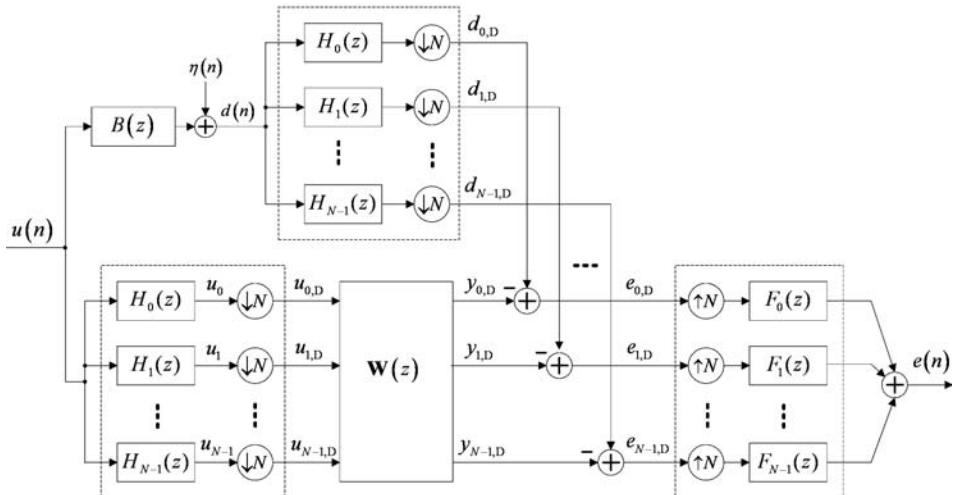


Figure 4.9 Adaptive system identification using a SAF with adaptive cross filters (adapted from Reference [30]). The unknown system $B(z)$ is modeled with a matrix of subfilters $\mathbf{W}(z)$ in the subband domain

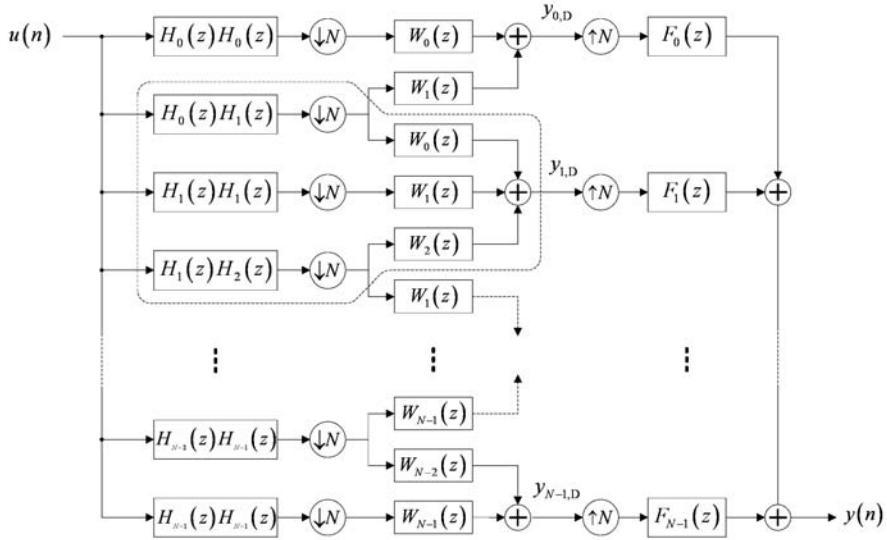


Figure 4.10 An N -band critically sampled SAF structure with cross filters for adjacent subbands (adapted from Reference [40])

are correlated. The critically sampled structure with cross filters has a similar convergence performance to the fullband adaptive filter [30].

Elimination of adaptive cross filters, $W_{i,j}(z)$ for $j \neq i$, reduces the ability of the SAF to cancel aliasing components and degrades the accuracy of modeling an unknown system. Assuming that the analysis filters are designed such that only the adjacent filters are overlapped, inserting adaptive cross filters between adjacent subbands is sufficient to compensate for the aliasing effects. The adaptive elements $W_{i,i}(z)$, $W_{i,i-1}(z)$ and $W_{i,i+1}(z)$ for the i th subband can be adapted using the common error signal $e_{i,D}(k)$. In the new subband structure shown in Figure 4.10 [40], cross-channel information for the adaptive subfilter $W_i(z)$ is derived from adjacent subbands using branches that are composed of analysis filters $H_i(z)H_j(z)$ and adjacent subfilters $W_j(z)$, where $j = i \pm 1$. Similar to the adaptive cross filters shown in Figure 4.9, these branches compensate for the aliasing components due to critical decimation. The major advantage of the structure shown in Figure 4.10 is that the number of adaptive subfilters remains equal to the number of subbands N . However, similar copies of adaptive subfilters are found in the branches where the cross-channel information is derived. Some promising results on the convergence of this structure under colored signals have also been reported in Reference [40].

4.3.3 Multiband-structured SAF

Figure 4.11 shows the MSAF algorithm with the following distinctive features in comparison with the open-loop SAF shown in Figure 4.1:

- (i) A diagonal model matrix $W(z)\mathbf{I}_{N \times N}$ (where $\mathbf{I}_{N \times N}$ is the $N \times N$ identity matrix) is placed before the decimators. The N -input N -output system $W(z)\mathbf{I}_{N \times N}$ is a bank

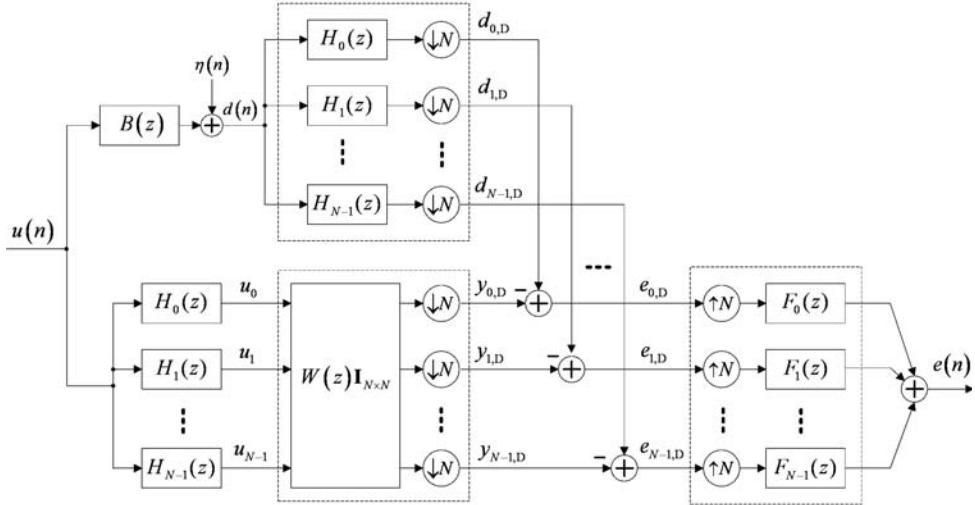


Figure 4.11 A multiband structure for adaptive system identification. Identical fullband adaptive filters are placed in all the subbands (adapted from Reference [41])

of identical filters $W(z)$ of length M . In the conventional SAF shown in Figure 4.1, a set of subfilters $W_0(z), W_1(z), \dots, W_{N-1}(z)$ are placed after the decimators so that the filter length can be reduced to $M_S = M/D$. As discussed in Section 4.3.2, the conventional SAF employs a set of decimated subfilters $W_i(z)$ that are unable to model the unknown system $B(z)$ perfectly. In contrast, the MSAF uses a fullband adaptive filter $W(z)$ to model the unknown system $B(z)$ since identical filters are used in all subbands. Perfect modeling can be achieved if the order of adaptive filters is sufficiently high and the analysis filters are power complementary.

- (ii) A multiband-structured weight-control mechanism. The identical adaptive filters $W(z)$ placed before the decimators are operated on the set of subband signals $u_i(n)$ at the original sampling rate. The tap weights of the fullband adaptive filter, $\{w_m(k)\}_{m=0}^{M-1}$, are updated by a single adaptive algorithm using the subband signals $\{u_i(n), e_{i,D}(k)\}_{i=0}^{N-1}$. In contrast, there are N adaptive subfilters (usually updated by the LMS or NLMS algorithm) operated separately in each subband of the conventional SAF, as illustrated in Figure 4.1.

The multiband nature of the MSAF algorithm can be seen from its weight-control mechanism. The adjustment of fullband tap weights is a weighted combination of the subband signal vectors. From the stochastic interpretation presented later in Section 6.4, the weight adaptation is driven by an equalized spectrum of the form

$$\Gamma_\xi(e^{j\omega}) = \sum_{i=0}^{N-1} \lambda_i^{-1} |H_i(e^{j\omega})|^2 \Gamma_{uu}(e^{j\omega}), \quad (4.6)$$

where λ_i denotes the normalization factor, which is proportional to the respective variance of the i th subband signal. The equalized spectrum $\Gamma_\xi(e^{j\omega})$ given in Equation (4.6) is a linear combination of the normalized subband spectra, $\Gamma_{ii}(e^{j\omega})/\lambda_i$, for $i = 0, 1, \dots, N - 1$.

Consider again the AR(2) signal and the cosine-modulated filter bank as shown in Figure 4.2. The analysis filters partition the input spectrum $\Gamma_{uu}(e^{j\omega})$ into four subbands, as illustrated in Figure 4.3. Here, each subband spectrum $\Gamma_{ii}(e^{j\omega})$ is normalized with

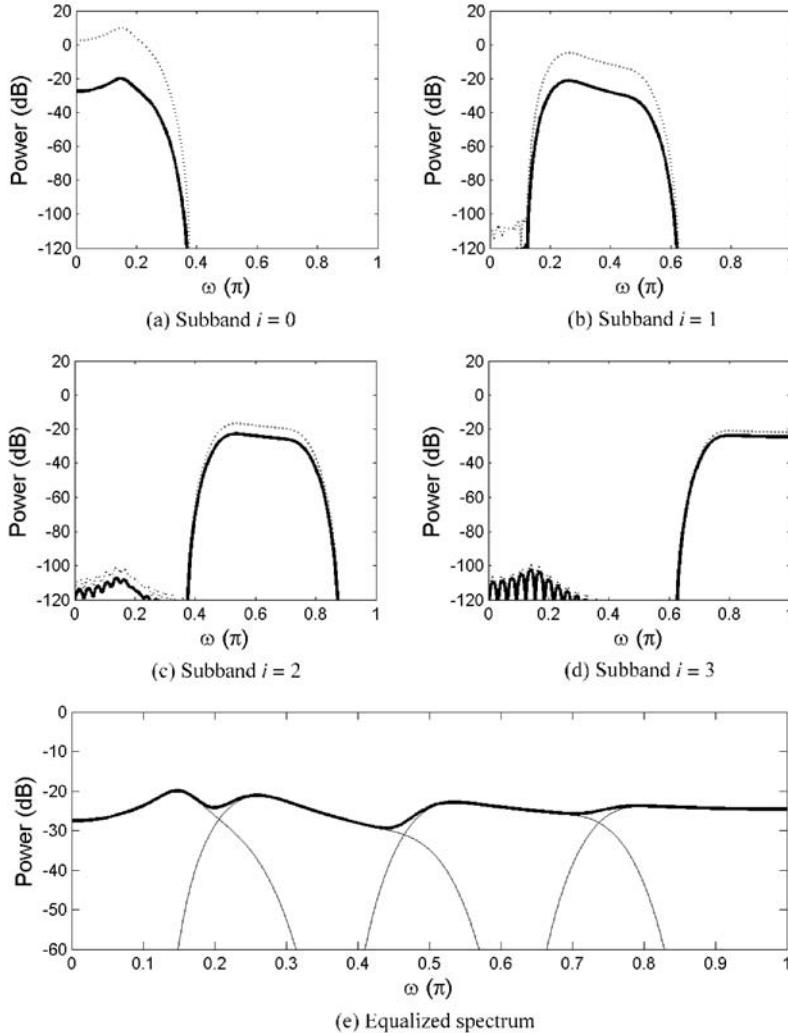


Figure 4.12 The normalized subband spectra ($\Gamma_{ii}(e^{j\omega})/\lambda_i$) are shown in (a), (b), (c) and (d). These normalized spectra are combined to form the equalized spectrum $\Gamma_\xi(e^{j\omega})$, as shown in (e)

a factor of $\lambda_i = \|\mathbf{u}_i(k)\|^2 + \alpha$, as illustrated in Figures 4.12(a) to (d). Notice that the dotted line represents the subband spectrum $\Gamma_{ii}(e^{j\omega})$ before normalization. Finally, by summing $\lambda_i^{-1}\Gamma_{ii}(e^{j\omega})$ using Equation (4.6), we obtain the equalized spectrum $\Gamma_\xi(e^{j\omega})$, as shown in Figure 4.12(e), which shows a reduced spectral dynamic range as compared with the input spectrum $\Gamma_{uu}(e^{j\omega})$ shown in Figure 4.2. Clearly, band edges of the subband spectrum $\Gamma_{ii}(e^{j\omega})$ disappear after the summation. Furthermore, aliasing distortion is avoided as the input signals $u_i(n)$ involved in the weight adaptation are not being decimated. Each narrowband spectrum $\Gamma_{ii}(e^{j\omega})$ is allowed to overlap with neighboring subbands and is used to adapt the fullband filter $W(z)$. Detailed analysis is illustrated in M-file `Example_4P3.m`.

The major motivation for using the MSAF is to improve the convergence of the gradient-based adaptive algorithms with a minimum increase in computational complexity [14]. In the MSAF, computational cost is reduced by:

- (i) exploiting the orthogonality of the subband signals by having the subband signals to be orthogonal at zero lag and
- (ii) decimating the adaptation rate.

The first factor will be discussed in Chapter 6. The rationale behind decimating the adaptation rate is justified by assuming that the analysis filter bank is lossless. Subband signals can be critically decimated while retaining all the information. When the decimation factor is smaller than or equal to the number of subbands, updating tap weights at the decimated rate will not degrade performance since similar amounts of information are used.

4.3.4 Closed-loop delayless structures

In the conventional SAF, the subfilters are adapted by independent adaptive algorithms (LMS or NLMS) to minimize the local MSE in each subband. The subband structure can be seen as a direct form of realizing the adaptive algorithm by dividing it into subbands. Different from the conventional SAF, the delayless structure involves a closed-loop feedback of the fullband error signal for subband weight adaptation. All nonzero frequency components in the error signal will be fed back to the subband adaptive filters for a weight update. Therefore, the closed-loop delayless configuration allows minimization of the fullband error signal and guarantees that the adaptive filter $W(z)$ converges to the optimal Wiener solution. The closed-loop delayless structures will be discussed in the next section.

The original objective of the delayless structures is to eliminate the signal-path delay caused by the analysis and synthesis filter banks, while retaining the advantages of computational saving and convergence speed from subband processing. It was recognized in References [13], [16], [17] and [39] that the aliasing effects encountered in critically sampled SAFs and the band-edge effects encountered in oversampled SAFs [18] can actually be reduced with closed-loop delayless configurations. Such characteristics are closely related to its capability to minimize the fullband MSE instead of the individual subband MSEs.

4.4 Delayless subband adaptive filters

In subband structures presented in previous chapters, the fullband error signal is obtained by interpolating and recombining the decimated subband error signals using a synthesis filter bank. The analysis and synthesis filter banks, which are used to derive the subband signals and to reconstruct the fullband signal, introduce an undesirable delay into the signal path. In applications such as acoustic echo cancellation (AEC) and wideband active noise control, the delay is highly undesirable [18, 32, 34]. In AEC applications, the fullband error signal contains the desired near-end speech that will be transmitted to the far end. The transmission delay has to be limited in order to comply with some telecommunication standards [42].

Signal path delay can be eliminated with the delayless structure shown in Figure 4.13. The basic idea of delayless subband adaptive filtering has been conceived by Morgan and Thi [18, 20]. This section presents the general structures, algorithms, and characteristics of delayless SAFs. Detailed analysis of these delayless structures can be found in References [39] and [43] to [46].

4.4.1 Closed-loop configuration

In the delayless structure illustrated in Figure 4.13, signal-path delay is eliminated by implementing the adaptation process in an auxiliary loop. The adaptive subfilters $W_i(z)$ are adapted in the subbands. The updated subband tap weights are collectively transformed into a set of tap weights for the fullband adaptive filter $W(z)$, which is located in the signal path. Therefore, the filtering process is operated in the time domain as described

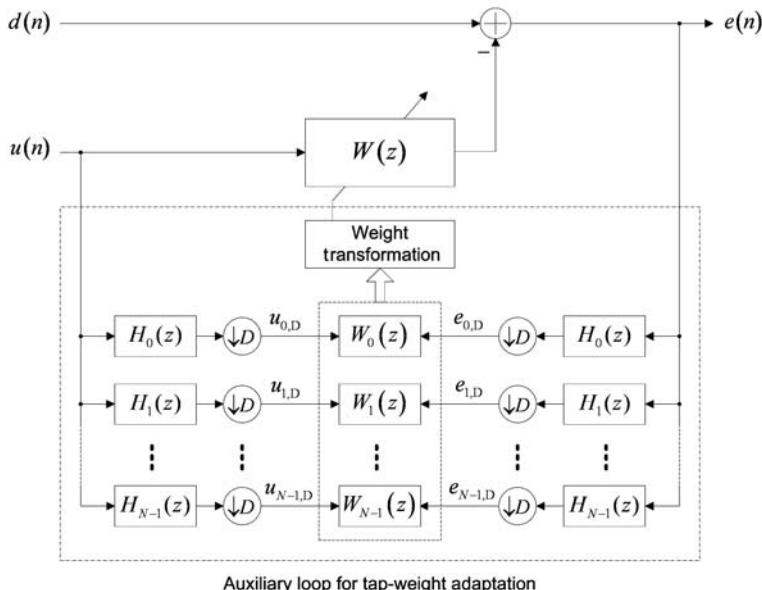


Figure 4.13 A closed-loop delayless SAF (adapted from Reference [18])

in Section 1.2, while the adaptation process is kept in the subbands with a decimated rate. Notice that the conventional SAF of Figure 4.1 requires a pair of analysis and synthesis filter banks. In contrast, the delayless structure shown in Figure 4.13 requires only analysis filter banks. The error signal $e(n)$ is directly obtained from the fullband filter $W(z)$ instead of synthesizing subband error signals using the synthesis filters; thus this structure is called a closed-loop SAF, as discussed in Section 4.2.2.

As depicted in Figure 4.13, the fullband error signal $e(n)$ is generated by subtracting the output of $W(z)$ from the desired response $d(n)$, which is the same as the fullband adaptive filter shown in Figure 1.2. Signal delay is only introduced by the analysis filter bank applied to the fullband error signal $e(n)$; thus the adaptation algorithm uses past error signals presented in the tapped-delay line of the analysis filters. This factor reduces the upper bound of the step size that can be employed for the closed-loop SAF, resulting in slow convergence. Therefore, the SAF in a closed-loop delayless structure behaves differently from the original SAF shown in Figure 4.1.

4.4.2 Open-loop configuration

A delayless SAF structure can also be realized as an open-loop configuration shown in Figure 4.14. The subband error signal $e_{i,D}(k)$ used for adaptation in each subband is derived as the difference between the subband desired response $d_{i,D}(k)$ and the corresponding output of the adaptive subfilter $W_i(z)$. In the open-loop configuration, the fullband error signal $e(n)$ is not fed back for updating the tap weights of adaptive subfilters. The fullband error signal $e(n)$ and the subband error signals $e_{i,D}(k)$ are different

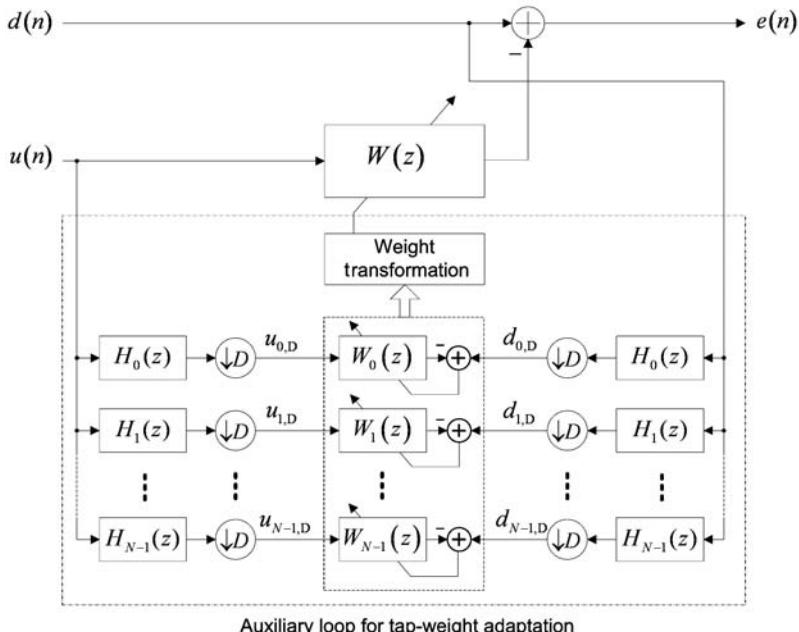


Figure 4.14 An open-loop delayless SAF (adapted from Reference [18])

error quantities. In the case of AEC, $e(n)$ is the near-end signal to be transmitted to the far-end room, while $e_{i,D}(k)$ serves as the estimation error for the weight adaptation. The fullband filtering performed by $W(z)$ does not affect the behavior of the adaptive algorithm. The convergence behavior of the open-loop delayless SAF algorithm is similar to the original SAF structure shown in Figure 4.1, except that the signal-path delay introduced by the filter banks has been eliminated by the delayless structure. Thus the open-loop delayless SAF inherits the structural problems such as aliasing and band-edge effects encountered in the conventional SAF.

4.4.3 Weight transformation

Several techniques have been proposed to perform the subband-to-fullband weight transformation [13, 16–18, 44], which derives tap weights of $W(z)$ from the set of adaptive subfilters $W_i(z)$. The weight transformation techniques have some distinctive features that are greatly dependent on the characteristics of the analysis filter bank used for the subband decomposition.

4.4.3.1 Frequency sampling method

The original delayless structure [18, 20] employs non-critically-decimated DFT filter banks. Recall that an N -channel DFT filter bank is comprised of N complex-modulated bandpass filters $H_i(z) = P(z e^{-j2\pi i}/N)$, where $P(z)$ is the real-valued prototype lowpass filter with a cutoff frequency of π/N . Because the impulse response coefficients of $H_i(z)$ and $H_{N-i}(z)$, for $i = 1, 2, \dots, N/2 - 1$, are complex conjugates, only $N/2 + 1$ subbands are needed for real-valued signals. Similar to the conventional SAF, the adaptive subfilters $W_i(z)$ used in subbands have the length of $M_S = M/D$, where D is the decimation factor.

Since the subband signals are complex valued, the tap weights of the adaptive subfilters are complex valued. The function of the weight transformation is to map the complex subband tap weights into an equivalent set of real-valued fullband tap weights. The weight transformation steps can be summarized as follows:

Step 1. Transform the tap weights of the adaptive subfilters w_i into the DFT domain by computing the M_S -point FFT of w_i for $i = 0, 1, \dots, N/2$. The vectors $\mathbf{w}_i = [w_{i,0}, w_{i,1}, \dots, w_{i,M_S-1}]^T$ can be seen as the time-domain representation of the subfilters $W_i(z) = \sum_{k=0}^{M_S-1} w_{i,k} z^{-k}$. The weight vectors \mathbf{w}_i for the first $N/2 + 1$ subbands are transformed by the FFT to obtain M_S DFT coefficients for each subband.

Step 2. The $M_S \times (N/2 + 1)$ DFT coefficients obtained in the previous step are stacked to form the first $M/2$ points of an M -element vector from index 0 to $(M/2 - 1)$. The vector is completed by setting the $(M/2)$ th point to zero and then using the complex-conjugate values of points from index 1 to $(M/2 - 1)$ in reversed order as the values of points from index $(M/2 + 1)$ to $(M - 1)$. The fullband tap weights are finally obtained by taking the inverse FFT (IFFT) of this M -element vector. The procedure for frequency stacking is illustrated in Table 4.1 for the case of $N = 8$, $M = 128$ and $D = N/2$ for $2\times$ oversampling. This frequency stacking procedure [18] is formulated in equivalent mathematical expressions in References [43] and [47].

Table 4.1 Mapping from subband DFT coefficients to fullband DFT coefficients. This example uses 8 subbands each adaptive subfilter has 32 taps and the fullband adaptive filter has 128 coefficients

Subband DFT coefficient index	Fullband DFT coefficient index				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = \frac{N}{2} = 4$
0	0		32		
1	1		33		
2	2		34		
3	3		35		
4	4		36		
5	5		37		
6	6		38		
7	7		39		
8		8		40	
9		9		41	
10		10		42	
11		11		43	
12		12		44	
13		13		45	
14		14		46	
15		15		47	
16		16		48	
17		17		49	
18		18		50	
19		19		51	
20		20		52	
21		21		53	
22		22		54	
23		23		55	
24			24		56
25			25		57
26			26		58
27			27		59
28			28		60
29			29		61
30			30		62
31			31		63

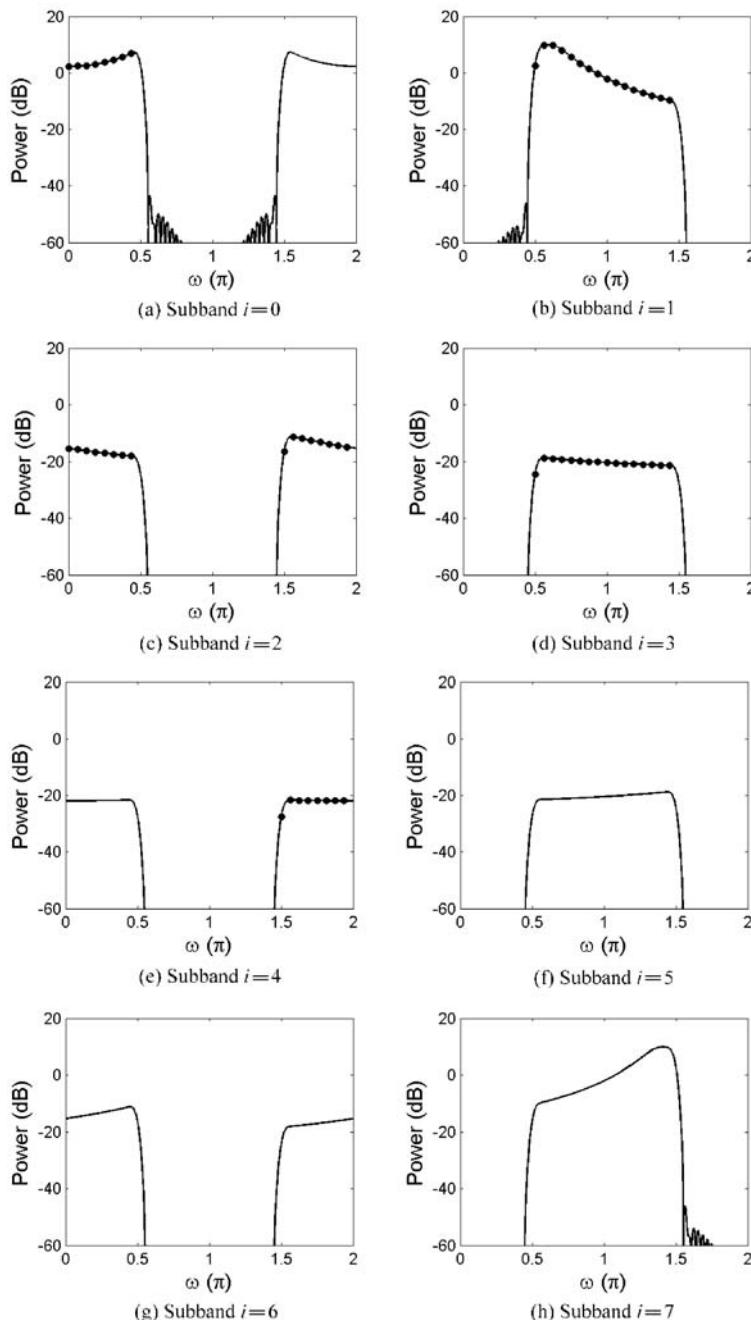


Figure 4.15 Frequency stacking of subbands. The DFT coefficients that appear at the passband (indicated with black dots) are taken and stacked together to form the DFT coefficients of the fullband spectrum

The above weight transformation procedure can be interpreted from the frequency-sampling perspective. The idea of the weight transformation is to transform the subfilter w_i into the DFT domain (i.e. the frequency responses of the subfilters are sampled along the unit circle) and to extract the DFT coefficients of the subfilter at frequencies that appear within the passband of the subband spectrum $\Gamma_{ii,D}(e^{j\omega})$. Since the subband signals are $2\times$ oversampled, only half of the DFT coefficients are within the passband of subband spectrum $\Gamma_{ii,D}(e^{j\omega})$. Furthermore, due to the decimation with a factor of $D = N/2$, the subband spectra exhibit their passbands at $\{0 \rightarrow \pi/2\}$ and $\{3\pi/2 \rightarrow 2\pi\}$ for i is even and $\{\pi/2 \rightarrow 3\pi/2\}$ for i is odd (see Figure 4.15). Hence, the DFT coefficients that appear at the passband are taken according to the order indicated in Table 4.1. The DFT coefficients extracted from consecutive subbands are stacked together to form the DFT coefficients of the fullband filter. Finally, the frequency-domain fullband filter is transformed to the time domain by inverse DFT. The idea of frequency stacking is further illustrated in Example 4.2.

The weight adaptation is applied on the subfilters $W_i(z)$ using the subband signals $\{u_{i,D}(k), e_{i,D}(k)\}$. In the above weight transformation, each of these subfilters $W_i(e^{j\omega})$ forms a portion of the fullband filter $W(e^{j\omega})$. Thus, the set of subband signals are used in the indirect manner of updating the tap weights of the fullband adaptive filter $W(z)$. Clearly, the weight-control mechanism for the delayless structure is similar with that of the multiband-structured SAF in the sense that a set of subband signals are used to adapt the tap weights of the fullband filter $W(z)$.

Example 4.2 We investigate the frequency stacking operation in the DFT domain. The DFT filter bank consists of $N = 8$ subbands and the input signal is an AR(2) signal, as shown in the top panel of Figure 4.6. We partition the input signal into $N = 8$ subbands with $2\times$ oversampling (i.e. the decimation factor $D = N/2$). The spectrum of each subband is shown in Figure 4.15. Notice that subbands $i = 1, 2, 3$ form a conjugate pair with $i = 7, 6, 5$, respectively. Frequency stacking can therefore be done using subbands $i = 0, 1, 2, \dots, 5$. Furthermore, decimation with a factor of $D = N/2$ causes the subband spectra to exhibit their passbands either at the middle or at the both ends of the frequency range from 0 to 2π , as shown in Figure 4.15. Using this characteristic, frequency stacking can be performed using the following script (a complete listing is given in M-file `Example_4P4.m`). The variable `xd` holds the spectrum of the subband signal. The last line of code forms a fullband spectrum as described in step 2 of the frequency-sampling method described in this section.

```

for i = 1:2:(N/2-1) % For i is an odd number
    xd = PSDecimation(HzSig.*Hz(i+1,:),DFTpoint,D);
    x = reshape(xd,Ls/4,4); % Split into four chunks
    x = [x(:,2); x(:,3)]; % Take two chunks at the middle
    X(:,i) = x; % Stacking
end
for i = 2:2:(N/2-1) % For i is an even number
    xd = PSDecimation(HzSig.*Hz(i+1,:),DFTpoint,D);

```

```

x = reshape(xd,Ls/4,4);           % Split into four chunks
x = [x(:,4); x(:,1)];           % Take two chunks at both ends
X(:,i) = x;                      % Stacking
end
X = X(:);                        % Turn matrix into vectors
% Form a fullband filter in the DFT domain
X = [X; 0; conj(flipud(X(2:end)))];

```

4.4.3.2 DFT filter bank with fractional delays

Recent developments [16, 17, 23] show that weight transformation for critically decimated SAF is possible by using a DFT filter bank with its lowpass prototype filter $P(z)$ being an N th-band filter. Recall that the polyphase components $E_l(z)$ of the lowpass filter $P(z) = \sum_{l=0}^{N-1} E_l(z^N) z^{-l}$, with a cutoff frequency of π/N , appear as fractional delays due to their relative phase difference [48, p. 799]. Furthermore, the l th polyphase component $E_l(z)$ generates a forward time shift of l/N samples with respect to the zeroth polyphase component $E_0(z)$. If the prototype filter $P(z)$ is the N th-band filter with its last polyphase component given by $E_{N-1}(z) = z^{-\Delta_{\text{int}}}$, we obtain an analysis DFT filter bank with fractional delays as shown in Figure 4.16. The polyphase components of $P(z)$ have the following forms:

$$\begin{aligned}
E_0(z) &\approx z^{-\Delta_{\text{int}}-(N-1)/N} \\
E_1(z) &\approx z^{-\Delta_{\text{int}}-(N-2)/N} \\
&\vdots \\
E_{N-1}(z) &= z^{-\Delta_{\text{int}}},
\end{aligned} \tag{4.7}$$

where Δ_{int} denotes the integer part of the delay. The idea is illustrated further in Example 4.3.

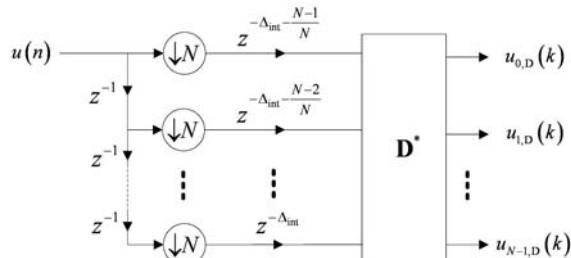


Figure 4.16 DFT filter bank with fractional delays. The matrix $\mathbf{D} = [a_{mn}]$ denotes the DFT matrix with elements $a_{mn} = e^{-j2\pi mn}/N$. Note that \mathbf{D}^* denotes the inverse DFT matrix

Example 4.3 We design an FIR filter of length $L = 10N - 1 = 39$ for a filter bank with $N = 4$ subbands. The impulse response of the filter is obtained using the MATLAB function `fir1(L-1, 1/N)`. The MATLAB function returns a windowed sinc function, as shown in Figure 4.17. We decompose the impulse response into $N = 4$ polyphase components, which can be done efficiently using the `reshape` function, as shown below (a complete listing is given in M-file `Example_4P5.m`). The impulse responses of the polyphase components are plotted in Figure 4.17. Notice that the last polyphase component $E_3(z)$ has many zero samples except at $n = 4$.

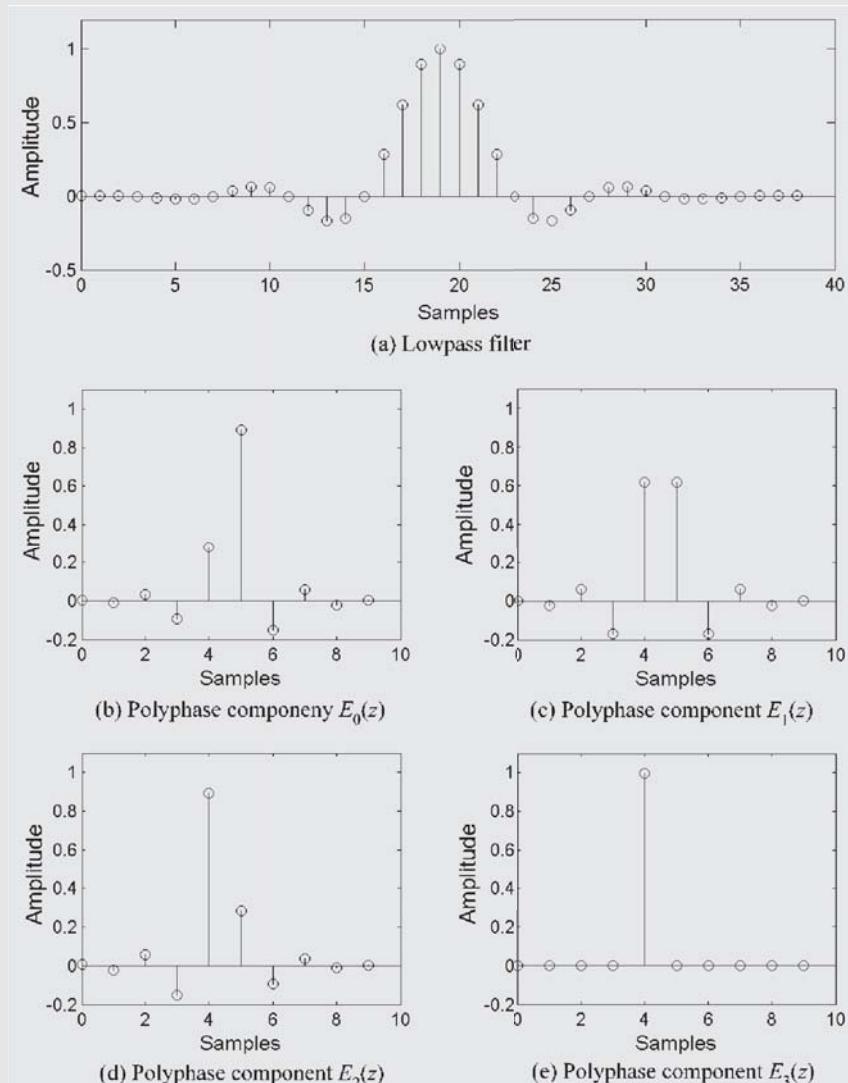


Figure 4.17 A lowpass filter and its polyphase components

This corresponds to the delay of $\Delta_{\text{int}} = 4$ samples. Figure 4.18 shows phase responses of the polyphase components. The polyphase components $E_0(z)$, $E_1(z)$ and $E_2(z)$ generate different amounts of fractional delays relative to $E_3(z)$ according to Equation (4.7).

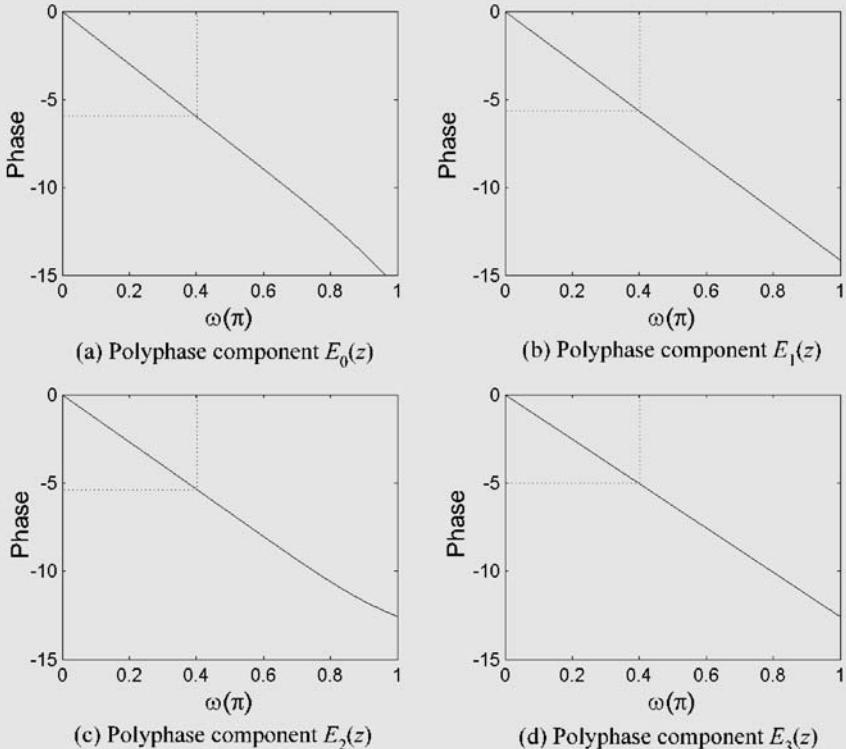


Figure 4.18 Phase responses of polyphase components. The slope of the phase response (which can be observed using the dotted line) indicates the delay relative to the $E_3(z)$

```

hopt = fir1(L-1,1/N);
hopt = hopt*N;
% zero-padding such that the total length L + 1 = 2KN
h = [hopt, 0];
% Polyphase decomposition
E = reshape(h,N,length(h)/N);

```

Besides the fractional delays, the length of adaptive subfilters has to be increased by one sample such that $M_S = M/N + 1$ to allow an accurate modeling of the unknown system [17, 23]. The subband tap weights are mapped into the corresponding fullband tap weights through the following steps:

Step 1. Compute an N -point IFFT on each of M_S columns of the matrix on the right side as follows:

$$\begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,M_S-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,M_S-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{N-1,0} & g_{N-1,1} & \cdots & g_{N-1,M_S-1} \end{bmatrix} = \text{IFFT} \left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,M_S-1} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,M_S-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N-1,0} & w_{N-1,1} & \cdots & w_{N-1,M_S-1} \end{bmatrix} \right). \quad (4.8)$$

Each row of the matrix on the right side of Equation (4.8), i.e. $\mathbf{w}_i^T = [w_{i,0}, w_{i,1}, \dots, w_{i,M_S-1}]$, represents the impulse response of the adaptive subfilter $W_i(z)$. Each row of the matrix on the left side, i.e., $\mathbf{g}_i^T = [g_{i,0}, g_{i,1}, \dots, g_{i,M_S-1}]$, represents the impulse response of the fractionally delayed polyphase component $G'_i(z) = G_i(z)z^{-i/N}$, where $G_0(z), \dots, G_{N-1}(z)$ are the polyphase components of the fullband filter $W(z)$. The IFFT returns real-valued column vectors since the coefficients, $w_{i,k}$ and $w_{N-i,k}$ for $i = 1, 2, \dots, N/2$, are complex conjugates for the real-valued signals $u(n)$ and $d(n)$.

Step 2. For the first polyphase component, we take $G_0(z) = G'_0(z)$. For the subsequent $i = 1, 2, \dots, N - 1$, the impulse response of $G'_i(z)$ is convolved with the fractional delay $E_{i-1}(z)$ as

$$G_i(z)z^{-(\Delta_{\text{int}}+1)} = G'_i(z)E_{i-1}(z), \quad \text{for } i = 1, 2, \dots, N - 1. \quad (4.9)$$

Step 3. Discard extra samples so that the polyphase components $G_i(z)$ of the fullband filter have a length of $M/N = M_S - 1$. For the first polyphase component $G_0(z)$, we simply discard the last sample. For $G_1(z), G_2(z), \dots, G_{N-1}(z)$, we discard the first $\Delta_{\text{int}} + 1$ samples and retain the next $M_S - 1$ samples. The fullband filter can be constructed from the polyphase components as

$$W(z) = \sum_{i=0}^{N-1} G_i(z^N) z^{-i}. \quad (4.10)$$

4.4.4 Computational requirements

The NLMS algorithm is used for adaptation of the subband filters $\mathbf{w}_i(k)$. Since the subband signals and tap weights are complex-valued numbers, the complex NLMS algorithm is employed as follows:

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \frac{\mu \mathbf{u}_i^*(k)}{\|\mathbf{u}_i(k)\|^2 + \alpha} e_{i,D}(k), \quad \text{for } i = 0, 1, \dots, N - 1, \quad (4.11)$$

where $\mathbf{u}_i(k) \equiv [u_{i,D}(k), u_{i,D}(k-1), \dots, u_{i,D}(k-M_S+1)]^T$ is the subband regressor for the subfilter $\mathbf{w}_i(k)$ of length M_S .

The computational complexity of the closed-loop delayless SAFs developed by Morgan and Thi [18, 20] and Merched *et al.* [16, 17] is summarized in Table 4.2. The

Table 4.2 Computational complexity of closed-loop delayless SAFs in terms of the number of real multiplications per input sample

Computational requirements	Closed-loop delayless SAF	
	Morgan and Thi [18, 20]	Merched <i>et al.</i> [16, 17]
Filter bank operations	$4 \left(\frac{L}{N} + \log_2 N \right)$	$2 \left(\frac{L}{N} + \log_2 N \right)$
Subband weight adaptation	$\frac{16M}{N}$	$4 \left(\frac{M}{N} + 1 \right)$
Fullband filtering	M	M
Weight transformation	$[2 \log_2 \left(\frac{2M}{N} \right) + \log_2 M] J$	$\left[\left(1 + \frac{N}{M} \right) \log_2 N + \frac{L(N-1)}{N^2} \right] J$

computational requirements of algorithms are separated into four parts: (i) filter bank operations, (ii) subband weight adaptation, (iii) fullband filtering (or linear convolution) and (iv) weight transformation. Detailed breakdowns of the computational requirements can be found in References [17], [18] and [23]. Note that the weight transformation involves a variable J that determines how often the weight transformation is performed. In particular, the weight transformation is performed once for every M/J input samples, where M is the length of the fullband adaptive filter. The computational complexity can be reduced by using smaller values of J . It has been shown in Reference [18] that the SAFs do not exhibit severe degradation in performance for values of J in the range from one to eight.

4.5 MATLAB examples

This section presents two sets of simulations in the context of adaptive system identification. The impulse response of the unknown system $B(z)$ is depicted in Figure 4.19. The length of the fullband filter $W(z)$ is $M = 1024$ which is identical to the unknown system $B(z)$. The complete listing of MATLAB programs can be found in `Example_4P6.m`, `Example_4P7.m` and `Example_4P8.m`.

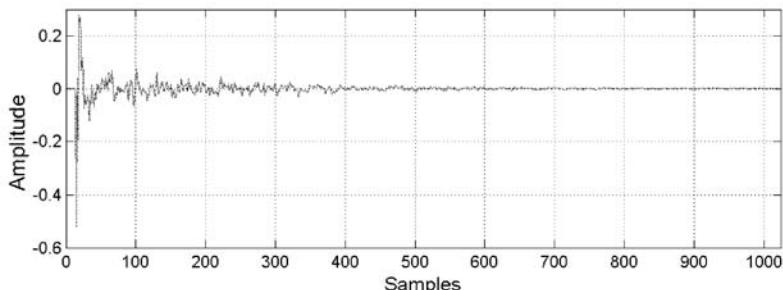


Figure 4.19 Acoustic impulse response of a room used in the simulations

4.5.1 Aliasing and band-edge effects

In the first set of simulations, a noiseless signal model with a white input signal is considered. The purpose of these simulations is to investigate the behavior of the SAF in dealing with the aliasing and band-edge effects. The DFT filter bank shown in Figure 4.6 is used for the simulation. The design specifications of the filter bank are detailed in Section 4.3.1. The length of the adaptive subfilters $W_i(z)$ is set to $M_S = M/N = 256$ for the critically sampled scheme and $M_S = 2M/N = 512$ for the $2\times$ oversampled scheme. The tap weights of the subfilters $W_i(z)$ are adapted using the NLMS algorithm defined in Equation (4.11) with $\mu = 0.1$ and $\alpha = 0.0001$.

Figure 4.20 shows the MSE learning curves of the oversampled SAF and the critically sampled SAF for comparison. These learning curves are obtained by averaging over 200 independent trials and smoothed with a 10-point moving-average filter. The MSE learning curve of the critically sampled scheme converges to a higher level of steady-state MSE due to the aliasing effects; thus it is unable to model the unknown system properly. The learning curve of the critically sampled scheme is characterized by a high asymptotic level of steady-state MSE. The aliasing effects and high asymptotic MSE can be eliminated by using the oversampled scheme. However, the convergence rate of the oversampled scheme is limited by a slow asymptotic convergence (after a fast initial convergence), which is due to some small eigenvalues generated by the band edges of the subband spectrum.

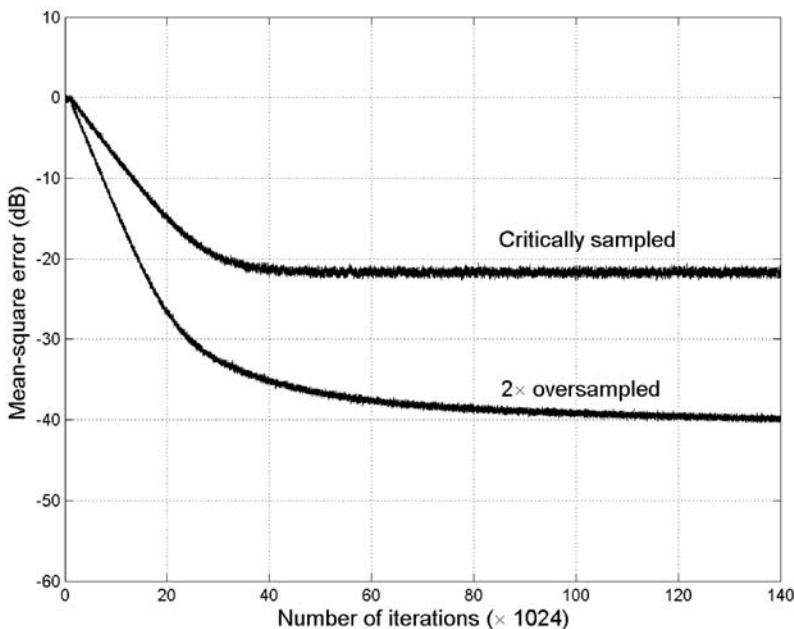


Figure 4.20 MSE learning curves of the critically sampled and oversampled SAFs for $N = 4$ subbands

4.5.2 Delayless alias-free SAFs

A comparison of two delayless SAFs listed below is demonstrated using MATLAB examples:

- (i) Closed-loop delayless SAF ($2\times$ oversampled) proposed by Morgan and Thi [18].
- (ii) Closed-loop delayless SAF (critically sampled) proposed by Merched *et al.* [16, 17].

The first and second delayless SAFs rely on the closed-loop feedback of fullband error signals to overcome the band-edge and aliasing effects, respectively.

Two highly correlated signals, an AR(10) signal and real speech, are used for the simulations. The impulse response of the unknown system $B(z)$ of length $M = 1024$ is shown in Figure 4.19. The delayless SAFs have $N = 16$ subbands and the lengths of the adaptive subfilters are $M_S = 2M/N = 128$ and $M_S = M/N + 1 = 65$ for the delayless SAFs of Morgan and Thi and Merched *et al.*, respectively (see Section 4.4.3). The subband tap weights are collectively transformed into the fullband weight vector $\mathbf{w}(k)$ of length $M = 1024$. The weight transformation is performed for every $M/J = 128$ samples, where $J = 8$ is used to control the update rate of the fullband weight vector $\mathbf{w}(k)$.

The delayless SAFs require a DFT filter bank. The prototype lowpass filter $P(z)$ for the DFT filter bank is designed with the length of $L = 8N - 1 = 127$ using the MATLAB function `fir1(L-1, 1/N)`. For an odd number $L = 127$, the MATLAB routine returns a windowed sinc function, which is exactly an N th-band filter. Figure 4.21 (top panel) shows the impulse response $p(n)$ of the prototype filter. The lower plot depicts the impulse response $e_{15}(n) = p(nN + 15)$ of the last polyphase component $E_{15}(z) = \sum_n e_{15}(n)z^{-n} = z^{-3}$. With reference to Figure 4.16, the integer part of the delay is $\Delta_{\text{int}} = 3$. This quantity is crucial for the subband-to-fullband weight transformation described in Section 4.4.3.2.

Figure 4.22 shows the ensemble-average learning curves of the delayless SAFs under the excitation of the stationary AR(10) signal. The learning curve of a fullband adaptive filter with the NLMS algorithm is also shown in the figure. Step sizes are chosen such that identical steady-state MSEs are achieved for all the adaptive filters. Note that the SNR at the output of the unknown system is set at 50 dB. Clearly, both delayless SAFs outperform the fullband adaptive filter with the NLMS algorithm. For $M = 1024$ and $N = 16$, the fullband NLMS algorithm requires 3072 real multiplications per sampling period, whereas the closed-loop delayless SAFs of Morgan and Merched require 2288 and 1400 real multiplications, respectively. The closed-loop delayless SAFs improve the convergence rate of the NLMS algorithm under colored signals, while achieving some computational savings. Nevertheless, it should be emphasized that the computational complexities of these adaptive filters are of the same order of magnitude.

The performance of the delayless SAFs becomes more apparent for speech signals. In general, speech signals are highly correlated and nonstationary. For such an excitation input, it is convenient to compare the convergence of the adaptive algorithms using the normalized misalignment [32, 49], which is defined as the norm of the weight-error vector $\|\mathbf{w}(k) - \mathbf{b}\|$, normalized by the norm of the optimum weights $\|\mathbf{b}\|$, expressed as

$$\text{Normalized misalignment} = 20 \log_{10} \frac{\|\mathbf{w}(k) - \mathbf{b}\|}{\|\mathbf{b}\|}. \quad (4.12)$$

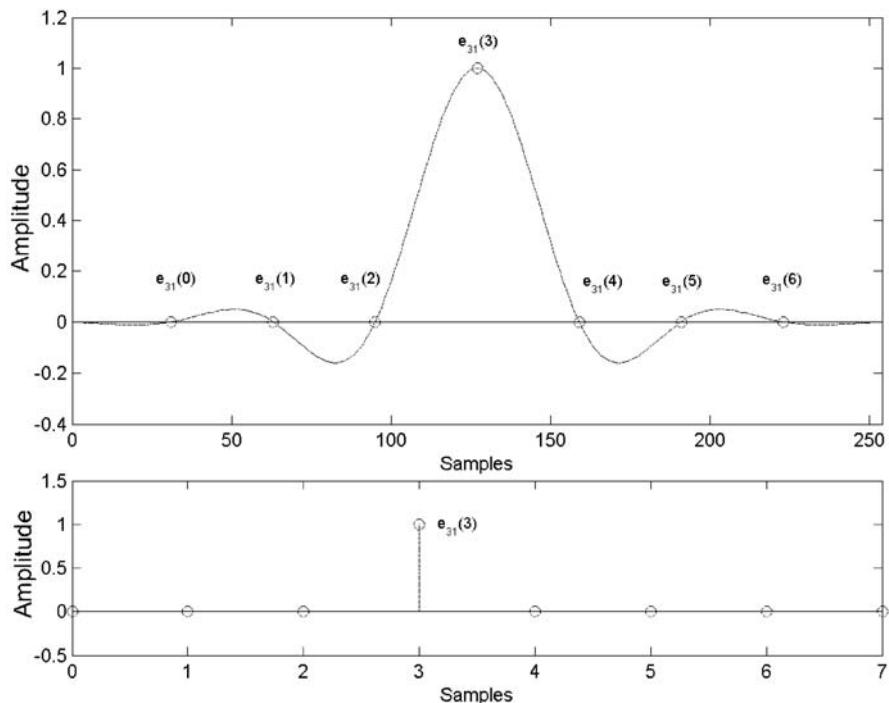


Figure 4.21 An N th-band prototype filter for the DFT filter bank

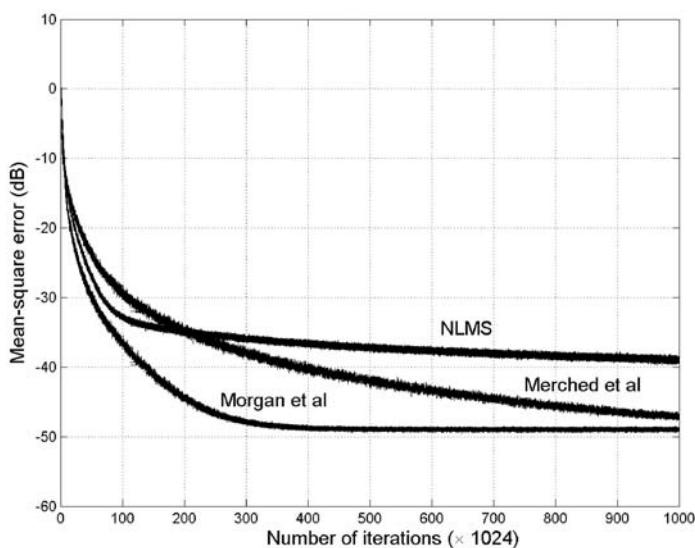


Figure 4.22 MSE learning curves of the fullband NLMS filter ($\mu = 0.10$), closed-loop delayless SAF of Merched *et al.* ($N = 16$, $\mu = 0.20$) and the closed-loop delayless SAF of Morgan and Thi ($N = 16$, $\mu = 0.17$)

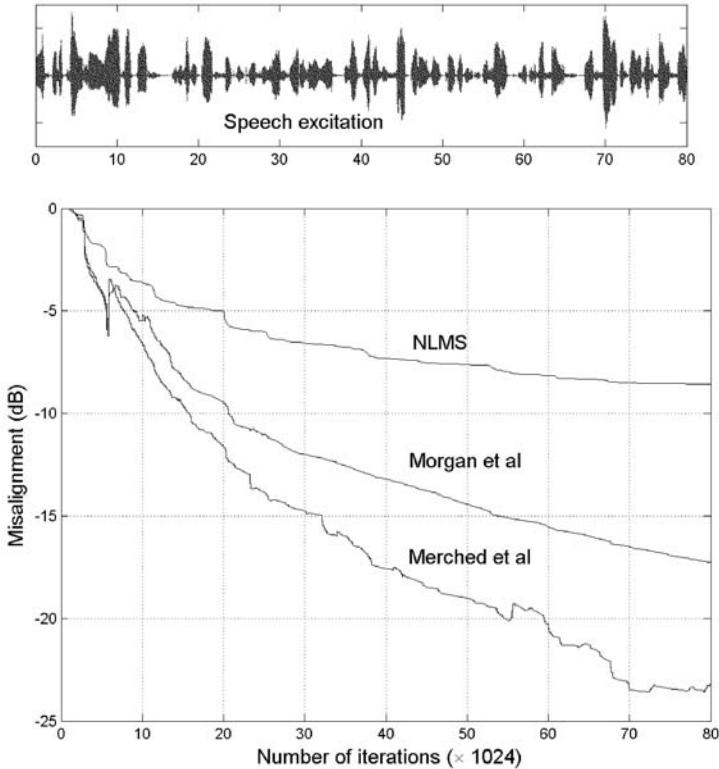


Figure 4.23 Normalized misalignment learning curves of the fullband NLMS adaptive filter ($\mu = 0.10$), closed-loop delayless SAF of Merched *et al.* ($N = 16, \mu = 0.20$) and closed-loop delayless SAF of Morgan and Thi ($N = 16, \mu = 0.17$)

This performance index measures the distance of the adaptive weight vector $\mathbf{w}(k)$ from the unknown system \mathbf{b} at each iteration. Notice that the definition of the normalized misalignment does not involve the expectation operator $E\{\cdot\}$; thus ensemble averaging is not required when obtaining the learning curves.

Figure 4.23 shows the misalignment learning curves of the delayless SAFs using the same step sizes μ and the number of subbands N as in the previous simulations. For simplicity, no plant noise $\eta(n)$ is added to the output of the unknown system. The closed-loop delayless SAF of Merched *et al.* converges faster than the one proposed by Morgan and Thi. This result may be due to the noiseless signal model that has been assumed in the simulations; i.e. the closed-loop delayless SAF of Merched *et al.* performs better under a noiseless situation. Nevertheless, the learning curves of Figure 4.23 show that both delayless SAFs outperform the fullband adaptive filter using the NLMS algorithm.

4.6 Summary

This chapter discussed several subband adaptive filtering structures. The conventional SAF can achieve significant computational savings, but its convergence performance is

limited by aliasing and band-edge effects. These structural problems were analyzed using the correlation-domain formulation presented in Chapter 3. In particular, we showed that the MSAF algorithm is effective in eliminating the aliasing and band-edge effects. Aliasing distortion is annihilated since the subband signals used for the weight adaptation are band-limited without decimation. Furthermore, band edges of the subband spectrum disappear in the summation that produces the normalized spectrum.

The concept of delayless SAFs was presented in Section 4.4. The original goal of the delayless structures was to eliminate the signal-path delay caused by the analysis and synthesis filter banks. The performance of two closed-loop delayless SAFs proposed by Morgan and Thi. [18] and Merched *et al.* [16, 17] were investigated. It was found that that the aliasing and band-edge effects can be reduced with the closed-loop delayless SAF structures. However, the closed-loop feedback of the fullband error signal introduced a delay into the weight-update path. This delay reduced the upper bound of the step size μ , thus degrading the convergence rate by using smaller step sizes.

References

- [1] N. J. Fliege, *Multirate Digital Signal Processing*, New York: John Wiley & Sons, Inc., 1994.
- [2] H. S. Malvar, *Signal Processing with Lapped Transform*, Norwood, Massachusetts: Artech House, 1992.
- [3] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, New York: McGraw-Hill, 2001.
- [4] T. Saramäki and R. Bregović, ‘Multirate systems and filter banks’, in *Multirate Systems: Design and Applications* (ed. G. Jovanovic-Dolecek), Hershey, Pennsylvania: Idea Group, 2002.
- [5] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Englewood Cliffs, New Jersey: Prentice Hall, 1993.
- [6] N. S. Jayant and P. Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video*, Englewoods Cliffs, New Jersey: Prentice Hall, 1984.
- [7] A. Gilloire, ‘Experiments with subband acoustic echo cancellers for teleconferencing’, in *Proc. IEEE ICASSP*, 1987, pp. 2142–2144.
- [8] A. Gilloire and M. Vetterli, ‘Adaptive filtering in subbands’, in *Proc. IEEE ICASSP*, 1988, pp. 1572–1575.
- [9] W. Kellermann, ‘Analysis and design of multirate systems for cancellation of acoustical echoes’, in *Proc. IEEE ICASSP*, 1988, pp. 2570–2573.
- [10] M. M. Sondhi and W. Kellermann, ‘Adaptive echo cancellation for speech signals’, in *Advances in Speech Signal Processing* (eds S. Furui and M. M. Sondhi), New York: Marcel Dekker, 1992.
- [11] M. de Courville and P. Duhamel, ‘Adaptive filtering in subbands using a weighted criterion’, in *Proc. IEEE ICASSP*, 1995, vol. 2, pp. 985–988.
- [12] M. de Courville and P. Duhamel, ‘Adaptive filtering in subbands using a weighted criterion’, *IEEE Trans. Signal Processing*, **46**(9), September 1998, 2359–2371.
- [13] N. Hirayama, H. Sakai and S. Miyagi, ‘Delayless subband adaptive filtering using the Hadamard transform’, *IEEE Trans. Signal Processing*, **47**(6), June 1999, 1731–1734.
- [14] K. A. Lee and W. S. Gan, ‘Improving convergence of the NLMS algorithm using constrained subband updates’, *IEEE Signal Processing Lett.*, **11**(9), September 2004, 736–739.

- [15] K. A. Lee, W. S. Gan and Y. Wen, ‘Subband adaptive filtering using a multiple-constraint optimization criterion’, in *Proc. EUSIPCO*, 2004, pp. 1825–1828.
- [16] R. Merched, P. S. R. Diniz and M. R. Petraglia, ‘A delayless alias-free subband adaptive filter structure’, in *Proc. IEEE ICASSP*, 1997, pp. 2329–2332.
- [17] R. Merched, P. S. R. Diniz and M. R. Petraglia, ‘A new delayless subband adaptive filter structure’, *IEEE Trans. Signal Processing*, **47**(6), June 1999, 1580–1591.
- [18] D. R. Morgan and J. C. Thi, ‘A delayless subband adaptive filter architecture’, *IEEE Trans. Signal Processing*, **43**(8), August 1995, 1819–1830.
- [19] S. S. Pradhan and V. U. Reddy, ‘A new approach to subband adaptive filtering’, *IEEE Trans. Signal Processing*, **47**(3), March 1999, 655–664.
- [20] J. Thi and D. R. Morgan, ‘Delayless subband active noise control’, in *Proc. IEEE ICASSP*, 1993, vol. 1, pp. 181–184.
- [21] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.
- [22] J. J. Shynk, ‘Frequency domain and multirate adaptive filtering filtering’, *IEEE Signal Processing Mag.*, **9**, January 1992, 14–37.
- [23] P. S. R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*, Norwell, Massachusetts: Kluwer Academic Publishers, 2002.
- [24] S. Haykin, *Adaptive Filter Theory*, Upper Saddle River, New Jersey: Prentice Hall, 2002.
- [25] D. G. Manolakis, V. K. Ingle and S. M. Kogon, *Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering and Array Processing*, New York: McGraw-Hill, 2000.
- [26] J. Homer, R. R. Bitmead and I. Mareels, ‘Quantifying the effects of dimension on the convergence rate of the LMS adaptive FIR estimator’, *IEEE Trans. Signal Processing*, **46**(10), October 1998.
- [27] J. Homer, ‘Quantifying the convergence speed of LMS adaptive FIR filter with autoregressive inputs’, *IEE Electronics Lett.*, **36**(6), March 2000, 585–586.
- [28] P. L. de León and D. M. Etter, ‘Experimental results with increased bandwidth analysis filters in oversampled, subband acoustic echo cancellers’, *IEEE Signal Processing Lett.*, **2**(1), January 1995, 1–3.
- [29] P. L. de León and D. M. Etter, ‘Acoustic echo cancellation using subband adaptive filtering’, in *Subband and Wavelet Transforms* (eds A. N. Akansu and M. J. T. Smith), Norwell, Massachusetts: Kluwer Academic Publishers, 1996.
- [30] A. Gilloire and M. Vetterli, ‘Adaptive filtering in subbands with critical sampling: analysis, experiments, and application to acoustic echo cancellation’, *IEEE Trans. Signal Processing*, **40**(8), August 1992, 1862–1875.
- [31] D. R. Morgan, ‘Slow asymptotic convergence of LMS acoustic echo cancellers’, *IEEE Trans. Speech Audio Processing*, **3**(2), March 1995, 126–136.
- [32] C. Breining, P. Dreiseital, E. Hänsler, A. Mader, B. Nitsch, H. Pudar, T. Schertler, G. Schmidt and J. Tilp, ‘Acoustic echo control – an application of very-high-order adaptive filters’, *IEEE Signal Processing Mag.*, **16**(4), July 1999, 42–69.
- [33] B. Farhang-Boroujeny and Z. Wang, ‘Adaptive filtering in subbands: design issues and experimental results for acoustic echo cancellation’, *Signal Processing*, **61**(3), September 1997, 213–223.
- [34] E. Hänsler and G. U. Schmidt, ‘Hands-free telephones – joint control of echo cancellation and postfiltering’, *Signal Processing*, **80**(11), November 2000, 2295–2305.
- [35] H. Yasukawa, S. Shimada and I. Furukawa, ‘Acoustic echo canceller with high speech quality’, in *Proc. ICASSP*, 1987, pp. 2125–2128.

- [36] K. A. Lee and W. S. Gan, ‘Adaptive filtering using constrained subband updates’, in *Proc. ICASSP*, 2005, pp. 2275–2278.
- [37] K. Mayyas and T. Aboulnasr, ‘A fast weighted subband adaptive algorithm’, in *Proc. ICASSP*, 1999, vol. 3, pp. 1249–1252.
- [38] K. Mayyas and T. Aboulnasr, ‘A fast exact weighted subband adaptive algorithm and its application to mono and stereo acoustic echo cancellation’, *J. Franklin Institute*, **342**(3), May 2005, 235–253.
- [39] S. Ohno and H. Sakai, ‘On delayless subband adaptive filtering by subband/fullband transforms’, *IEEE Trans. Signal Processing*, **6**(9), September 1999, 236–239.
- [40] M. R. Petraglia, R. G. Alves and P. S. R. Diniz, ‘New structures for adaptive filtering in subbands with critical sampling’, *IEEE Trans. Signal Processing*, **48**(12), December 2000, 3316–3327.
- [41] K. A. Lee and W. S. Gan, ‘Inherent decorrelating and least perturbation properties of the normalized subband adaptive filter’, *IEEE Trans. Signal Processing*, **54**(11), November 2006, 4475–4480.
- [42] ITU-T Recommendation G.167, *General Characteristics of International Telephone Connections and International Telephone Circuits – Acoustic Echo Controllers*, 1993.
- [43] J. M. de Haan, ‘Convergence and complexity analysis of delayless subband adaptive filters’, Research Report 2004:04, Blekinge Institute of Technology, 2004.
- [44] N. Hirayama and H. Sakai, ‘Analysis of a delayless subband adaptive filter’, in *Proc. IEEE ICASSP*, 1997, pp. 2329–2332.
- [45] S. Miyagi and H. Sakai, ‘Convergence analysis of alias-free subband adaptive filters based on a frequency domain technique’, *IEEE Trans. Signal Processing*, **52**(1), January 2004.
- [46] K. Nishikawa and H. Kiya, ‘Conditions for convergence of a delayless subband adaptive filter and its efficient implementation’, *IEEE Trans. Signal Processing*, **46**(4), April 1998, 1158–1167.
- [47] J. Huo, S. Nordholm and Z. Zang, ‘New weight transform schemes for delayless adaptive filtering’, in *Proc. IEEE Global Telecom. Conf.*, 2001, pp. 197–201.
- [48] J. G. Proakis and M. G. Manolakis, *Digital Signal Processing – Principles, Algorithms, and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 1996.
- [49] E. Hänsler, ‘The hands-free telephone problem’, in *Proc. IEEE ISCAS*, 1992, vol. 4, pp. 1914–1917.

5

Critically sampled and oversampled subband structures

As explained in previous chapters, subband adaptive filters (SAFs) are built upon multi-rate digital filter banks, which involve downsampling analysis and upsampling synthesis sections. By partitioning the fullband input and desired signals into multiple subband signals, individual adaptive subfilters can be applied to their respective subband signals to improve overall system performance. Advantages of performing adaptation in parallel subbands include: (i) the whitening effect of decimating subband signals leads to faster convergence, (ii) downsampling of subband signals and using adaptive subfilters of shorter length operated at the decimated rate reduces overall computation requirements and (iii) processing in an individual subband provides additional degrees of freedom in adaptation, which will be discussed in following sections.

This chapter discusses and analyzes the main characteristics of some subband adaptive filters (SAFs) with focuses on critically sampled and oversampled structures. Each SAF structure has its own filter bank design issues to be considered, and some of these filters will be designed using MATLAB examples to highlight important implementation issues.

Section 5.1 discusses the critically sampled SAFs used in conjunction with affine projection, variable step size and selective partial update techniques. Section 5.2 introduces a class of oversampled and nonuniform subband adaptive filters to reduce in-band aliasing. Section 5.3 introduces the filter bank design methods including exponential-modulated filter banks, alias-free design for critically sampled systems and real-valued oversampled filter banks. Finally, Section 5.4 presents a case study of using the proportionate adaptation technique for SAFs.

5.1 Variants of critically sampled subband adaptive filters

A main drawback of critically sampled SAF structures is the overlapped frequency responses of analysis filters between adjacent subbands. This undesired overlap results in in-band aliasing that will degrade the performance of the SAF.

This section examines some critically sampled adaptive filters using fast-convergence adaptive algorithms to improve their performance. In particular, we study the affine projection algorithms and variable step-size adaptive algorithms. We also investigate selective coefficient update techniques used in conjunction with critically sampled SAFs. These filter structures and their associated adaptive algorithms demonstrate important tradeoffs between the convergence rate and computational complexity. We highlight the performance improvement reported in the literature and compare their advantages and tradeoffs.

5.1.1 SAF with the affine projection algorithm

As introduced in Section 1.4, the affine projection (AP) algorithm derives the update vector from a P -dimensional projection of the current and $P - 1$ previous input signal vectors onto the posteriori error signal vector. This projection generates the update vector to force the set of $P - 1$ previous error signals to zero. There are several works that applied the AP algorithm to subband adaptive filters. Some use critically sampled subband filters [1–3] and others use oversampled filter banks with higher computational complexity [4, 5]. Critically sampled AP algorithms require analysis filters with higher attenuation (which may be difficult to implement) to reduce aliasing between nonadjacent subbands. Alternatively, extra subbands can be incorporated into the SAF to cancel the aliasing distortion.

An acoustic echo cancellation using the critically sampled SAF with the AP algorithm is shown in Figure 5.1, which was introduced by Gordy and Goubran [1] to eliminate aliasing. Unique features of this method are the additional channel between adjacent subbands for the input subband signals that are formed by double filtering the input signal with the set of analysis filters, and the extra delay of Δ samples in the desired

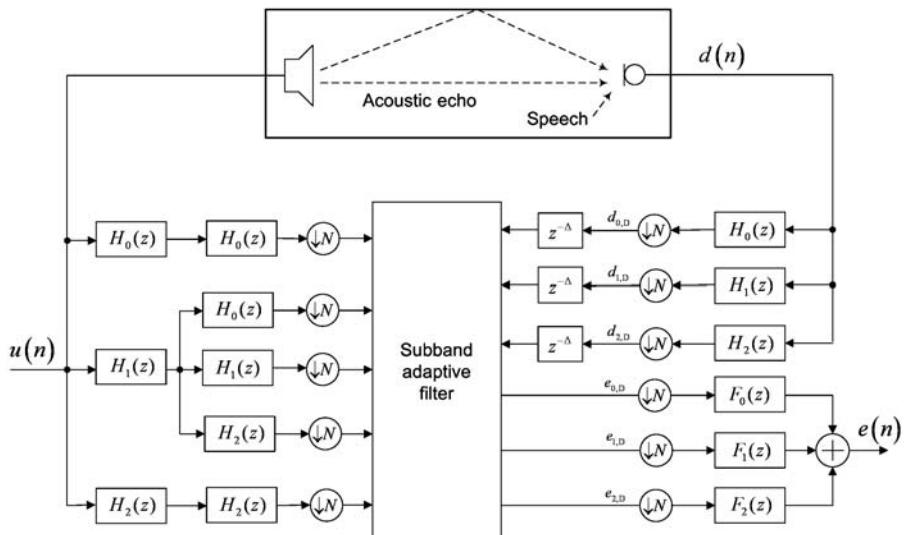


Figure 5.1 Block diagram of the critically sampled SAF structure with adaptive cross filters for $N = 3$ subbands (adapted from Reference [1])

subband signals to compensate delay caused by double filtering the input signal. Note that a similar idea of the double-filtering scheme with adaptive cross filters was introduced in Section 4.3.2 (see Figure 4.10). In general, $(2N-1)$ subbands are required for an N -channel analysis filter bank. As shown in Figure 5.1, an example of a three-band analysis filter bank is expanded to five subbands followed by a downsampling operation with the decimation factor of $D = 3$. This structure assumes the analysis filters have sufficient stopband attenuation such that aliasing between nonadjacent subbands can be neglected. The complete SAF with the AP algorithm is summarized in Table 5.1 [1].

The mean-square error (MSE) convergence and computational complexity had been analyzed in Reference [1]. It was found that the MSE convergence can be improved by using a relatively small number of subbands (N from 4 to 8) and low projection orders (P from 2 to 4). The computational complexity of the SAFs with the AP algorithm is approximately in the order of N lower than the fullband AP adaptive filters due to the decimated sampling rate. Further reduction of computational complexity can be achieved by using fast AP algorithms [6].

The AP algorithm can also be applied to oversampled subband adaptive filters [4]. The main advantage of using the oversampled SAF over the critically sampled SAF is the reduced in-band aliasing between subbands to achieve better performance. Further

Table 5.1 The affine projection algorithm for subband adaptive filters

Input and error signal definitions:

$$\mathbf{e}_i(k) = \mathbf{d}_i(k - \Delta) - [\mathbf{U}_{i,i-1}^T(k)\mathbf{w}_{i-1}(k) + \mathbf{U}_{i,i}^T(k)\mathbf{w}_i(k) + \mathbf{U}_{i,i+1}^T(k)\mathbf{w}_{i+1}(k)],$$

for $i = 0, 1, \dots, N-1$,

where

$$\begin{aligned}\mathbf{U}_{i,i-1}(k) &= [\mathbf{u}_{i,i-1}(k), \dots, \mathbf{u}_{i,i-1}(k-P+1)], \\ \mathbf{U}_{i,i}(k) &= [\mathbf{u}_{i,i}(k), \dots, \mathbf{u}_{i,i}(k-P+1)], \\ \mathbf{U}_{i,i+1}(k) &= [\mathbf{u}_{i,i+1}(k), \dots, \mathbf{u}_{i,i+1}(k-P+1)].\end{aligned}\tag{5.1}$$

Note. $\mathbf{U}_{i,i}(k)$ is the input matrix of the i th subband consisting of the current vector $\mathbf{u}_{i,i}(k)$ and $(P-1)$ past vectors for affine projection.

Updating equation:

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \mu \nabla_i(k),\tag{5.2}$$

where

$$\begin{aligned}\nabla_i(k) &= \mathbf{U}_{i,i-1}(k)\mathbf{C}_i^{-1}(k)\mathbf{e}_{i-1}(k) + \mathbf{U}_{i,i}(k)\mathbf{C}_i^{-1}(k)\mathbf{e}_i(k) + \mathbf{U}_{i,i+1}(k)\mathbf{C}_i^{-1}(k)\mathbf{e}_{i+1}(k), \\ \mathbf{C}_i(k) &= U_{i,i-1}^T(k)\mathbf{U}_{i,i-1}(k) + \mathbf{U}_{i,i}^T(k)\mathbf{U}_{i,i}(k) + \mathbf{U}_{i,i+1}^T(k)\mathbf{U}_{i,i+1}(k) + \delta \mathbf{I}.\end{aligned}$$

Note. A small scalar regularization factor δ is used to prevent instability caused by $\mathbf{C}_i^{-1}(k)$ and \mathbf{I} is the identity matrix of order P . The extreme subbands for $i = 0$ and $N-1$ are adjusted to remove terms involving subbands $i-1$ and $i+1$, respectively. Because these subbands are not physically present, there is only one subband adaptation term at the extreme subbands, as shown in Figure 5.1, for $i = 0$ and 2 in a three-subband SAF.

discussion of the oversampled SAF will be presented in Section 5.2.1. In this section, the AP algorithm is used for the oversampled SAF to increase the convergence speed. Unlike the algorithm shown in Figure 5.1, there is no expansion of analysis subbands. For example, a three-band oversampled SAF needs only three subbands with three associated adaptive filters that are updated by the AP algorithm. The difference between the oversampled SAF and the conventional critically sampled SAF (described in Chapter 4) is that the adaptive subfilters in the oversampled SAF operated at a higher rate (f_s/D) because the decimation factor, D , is smaller than the number of subbands, N . The AP algorithm can be efficiently applied to the oversampled SAF with a small projection order of 2 to 3, which is similar to the AP-based critically sampled SAF [4].

5.1.2 SAF with variable step sizes

An alternate method to increase the convergence rate of adaptive algorithms is to apply different step sizes based on the current state of the adaptive filter [7]. A simple example of a variable step-size algorithm is to use larger step sizes for faster convergence during the initial (or transient) state when the error signal is large and smaller step sizes to reduce the misadjustment in the steady state after the algorithm has nearly converged. Variable step-size algorithms have been successfully used in fullband adaptive filters [8–11]. Some of these algorithms [9, 11] use the instantaneous squared error as the criterion to adjust the step-size value. In SAF structures, squared subband error signals are used as multiple quadratic criteria [12] to derive variable step sizes that minimize the following criteria:

$$J_{\text{SB}} = \sum_{i=0}^{N-1} \mu_i E \{ |e_i(k)|^2 \}, \quad (5.3)$$

where μ_i is the step size used for the i th subband. This new criterion leads to the following updating equation

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \sum_{i=0}^{N-1} \mu_i \mathbf{u}_i(k) e_{i,D}^*(k), \quad (5.4)$$

where $\mathbf{w}(k)$ is the fullband weight vector. Note that this algorithm is similar to the multiband-structured SAF (MSAF) introduced in Section 4.3.3. In particular, the step size is set to $\mu_i = 1/\|\mathbf{u}_i(k)\|^2$ for the case of MSAF. It has been shown in Reference [12] that such a step size is in fact optimal in a stationary environment with a white noise as the input signal. However, when the noise is colored, the optimum step size is derived as $\mu_{i,\text{opt}} \leq \sqrt{g_i / \xi_i \sigma_i^2}$, where g_i is the variance of the system (or plant) noise in the i th subband, σ_i^2 is the subband noise variance and ξ_i is the energy of the i th subband signal (which can be obtained as the maximum value of the power spectral density of the input subband signals [12]). A problem of computing this optimal step size is the need to obtain the statistics of these time-varying energy terms. Also, this optimal step size imposes a very limited bound that may not be useful during the transient state. An example of the variable step-size SAF proposed in Reference [12] is summarized in Table 5.2.

Simulation results of this algorithm for system identification applications were reported in Reference [12] using four different cases: (i) white Gaussian noise and a

Table 5.2 The variable step-size algorithm for SAF [12]

Step-size adaptation:

$$\begin{aligned}\beta_i(k) &= \mathbf{v}_i^T \mathbf{u}_i(k) e_{i,D}^*(k), \\ \gamma_i(k) &= \alpha \gamma_i(k-1) + \beta_i(k-1), \\ \mu_i(k) &= \mu_i(k-1) \{1 + \rho_i[\beta_i(k)\gamma_i(k)]\},\end{aligned}\tag{5.5}$$

for $i = 0, 1, 2, \dots, N-1$, where $\mu_i(k)$ is the variable step size for subband i , α is a small constant that is close to and less than 1, ρ_i is the step size for the adaptation of μ_i , $\mathbf{u}_i(k)$ is the input signal vector, \mathbf{v}_i^T is the unitary eigenvector corresponding to the largest N eigenvalues ξ_i . This eigenvector can be predetermined by eigen-decomposition of the subband input autocorrelation matrix \mathbf{R}_i .

Subband weight vector adaptation:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \sum_{i=0}^{N-1} \mu_i(k) \mathbf{u}_i(k) e_{i,D}^*(k)\tag{5.6}$$

colored input signal, (ii) colored noise and input signals with and without similar spectra, (iii) using autoregressive models to generate input and echo signals for a monophonic acoustic echo canceller and (iv) stereophonic acoustic echo cancellation. The variable step-size SAF outperforms the fullband variable step-size algorithm. It was also found that the performance of the SAF with variable step sizes improved with an increased number of subbands, which shows a tradeoff between computational complexity and performance. In summary, the SAF divides the input and desired signals into multiple subbands and uses variable step sizes at each subband to enhance its performance by compensating the eigenvalue spread, especially for the cases of colored signals with a large spectral dynamic range.

5.1.3 SAF with selective coefficient update

Selective coefficient update is a technique commonly used in the time-domain [13–15] and transform-domain [16] adaptive algorithms. In general, these techniques update only a subset of tap weights based on some criteria. The advantage of partial update is the reduced computational cost for applications that need high-order adaptive filters such as echo cancellation for packet-switched telecommunication networks and acoustic echo cancellation for teleconferencing.

To present selective coefficient update operation in subbands, we examine an example of the partial-update discrete cosine transform LMS (DCTLMS) algorithm expressed as [16]

$$W_i(n+1) = \begin{cases} W_i(n) + \frac{\mu}{\lambda_i} e(n) U_i(n), & i \in \{S \text{ maximum values of } U_i^2(n)/\lambda_i\} \\ W_i(n), & \text{otherwise} \end{cases}\tag{5.7}$$

for $i = 0, 1, \dots, M - 1$, where $U_i(n)$ are M -point DCT output signals and λ_i are the eigenvalues of the transformed input autocorrelation matrix. Simulation shows that the partial-update DCTLMS algorithm achieves good results even by using a small value of S (4 to 8 times smaller) as compared to the length of the system impulse response to be identified [16].

A natural extension of the partial-update DCTLMS algorithm is the partial update of the SAF. A fast wavelet transform-domain LMS algorithm with partial-update subband tap weights was proposed in Reference [17]. In this method, only K (where $K < N$) subbands that have the larger ratios of output power $\|\mathbf{u}_i(n)\|^2$ to power estimate σ_i^2 of the subband signals are selected. Therefore, similar to Equation (5.7), the equation used for the selective-update SAF based on K largest values of $\|\mathbf{u}_i(n)\|^2 / \sigma_i^2$ can be expressed as

$$\mathbf{w}_i(n+1) = \begin{cases} \mathbf{w}_i(n) + \frac{\mu}{\sigma_i^2} e(n) \mathbf{u}_i(n), & i \in \{K \text{ maximum values of } \|\mathbf{u}_i(n)\|^2 / \sigma_i^2\} \\ \mathbf{w}_i(n), & \text{otherwise} \end{cases} \quad (5.8)$$

for $i = 0, 1, 2, \dots, N - 1$, where $\mathbf{w}_i(n)$ are the tap weights in the i th subband, $u_i(n)$ is the i th subband signal, σ_i^2 is the power estimate of the i th subband and $0 < \mu \leq 1$.

Simulation results show that this algorithm achieves the same level of performance as the full-update case by updating only half of the subfilters (i.e. $K = N/2$) [17]. In general, the partial update of adaptive subfilters results in significant improvement of performance in the steady state and has a lower computational cost as compared to the conventional NLMS algorithm.

5.2 Oversampled and nonuniform subband adaptive filters

Critically sampled subband adaptive filters suffer from slower convergence and larger steady-state MSE due to the aliasing effects introduced by the decimation of subband signals (see Section 4.5.1). In this section, we examine the oversampling and nonuniform subband adaptive filters to overcome the undesired effects introduced by the critically sampled subband adaptive filters.

5.2.1 Oversampled subband adaptive filtering

As discussed in Section 4.3.2, adaptive cross filters between adjacent subbands can be used to eliminate in-band aliasing in critically sampled SAF structures. Another solution is to use the oversampling filter banks. Oversampled SAF can be implemented as complex- and real-valued filters. An obvious drawback of complex analysis filters is that the generation and processing of complex-valued subband signals will increase the computational cost. Real-valued analysis filters can reduce computational complexity and they can also remove the in-band aliasing by choosing different downsampling rates for different subbands [18].

For example, Figure 5.2 shows a three-channel SAF consisting of two extreme subbands that are downsampled by two (which is less than the number of subbands) and the middle subband is decimated at the critical rate of three. This results in an oversampling ratio (OSR) of 133 %, which is defined as $\text{OSR} = \sum_{i=0}^{N-1} 1/D_i$, where D_i is the

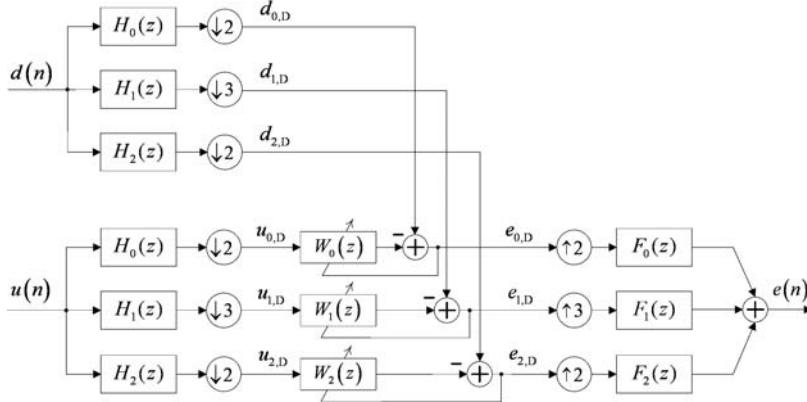


Figure 5.2 Block diagram of a three-channel oversampled SAF with real-valued analysis and synthesis filters using different decimation factors (adapted from Reference [18])

decimation ratio for the i th subband. Each subband has an independent adaptive subfilter $W_i(z)$ and the fullband error signal $e(n)$ is reconstructed from all subband error signals using the synthesis filters $F_i(z)$. The structure shown in Figure 5.2 can be considered as a generic subclass of nonuniform subband adaptive filters, which will be further introduced in the next section.

The computational complexity of the real-valued oversampled filter banks with different decimation ratios are $\sum_{i=0}^{N-1} (1/D_i)^2 \times M$ and $\sum_{i=0}^{N-1} (1/D_i)^3 \times M^2$ [18] for the LMS (or $O(M)$ -type) and RLS (or $O(M^2)$ -type) adaptive algorithms, respectively, where M is the length of the fullband adaptive filters. The assumption made in the complexity analysis is that each SAF uses adaptive subfilters of length $M_S = (1/D_i) \times M$ and the computation needed to perform subband decomposition is negligible as compared to the adaptive filtering operations.

One drawback of oversampled input signals is increased eigenvalue spread (or ill-conditioning) in the input correlation matrix of subband signals, which results in slow convergence of the LMS and NLMS algorithms (see Section 4.3.1 on the band-edge effect). The RLS algorithm can be used to increase the convergence rate at the expense of a higher computational requirement. The AP algorithm introduced in Section 5.1.1 to speed up the convergence of the critically sampled SAF can also be used to increase the convergence rate of the oversampled SAF [4]. However, the computational complexity of the AP-based oversampled SAF is still high. For the oversampled SAF with the identical decimation factor for all subbands, the oversampling factor is the ratio of the number of subbands to the decimation factor, which is recommended to be two or larger [4]. Theoretical analysis and computer simulation of the oversampled subband AP algorithm show that convergence can be improved by increasing the projection order P , but performance gain will be saturated as P is further increased. A good choice of P is two or three.

The oversampled SAF with increased bandwidth of the analysis filters relative to the synthesis filters was proposed to improve its convergence over the oversampled structures [19]. The increased bandwidth removes the slow-convergence spectral components

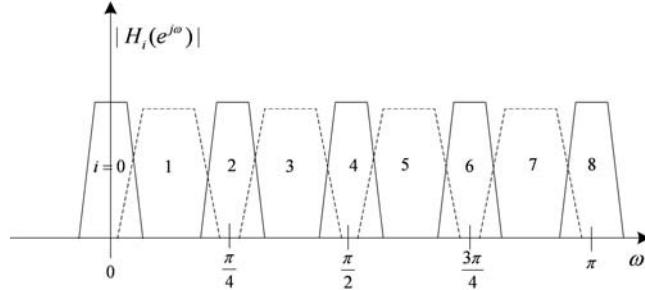


Figure 5.3 A nine-band oversampled SAF using different prototype filters and decimation factors

at the band edges. The increased amount of bandwidth is bounded by the Nyquist frequency of the downsampled signal. Simulation results show that the convergence of MSE improves significantly with increases up to 67 % of the maximum bandwidth as compared to other percentages of increase [19].

Another approach [20] of implementing the oversampled SAF is to use two different prototype filters and different decimation factors. The first prototype filter $p_0(n)$ is cosine modulated to obtain all odd-numbered subbands, while the second prototype filter $p_1(n)$ is cosine modulated to obtain all even-numbered subbands. Thus a total of $N = 2 \times N_0 + 1$ subbands are derived for this type of oversampled SAF, where N_0 is the number of bands derived from the first prototype filter. For example, a nine-band oversampled SAF is shown in Figure 5.3. Subbands 1, 3, 5 and 7 are derived from the prototype filter $p_0(n)$ and decimated by $D_1 = D_3 = D_5 = D_7 = 4$, while subbands 0, 2, 4, 6 and 8 are derived from the prototype filter $p_1(n)$ and decimated by factors of $D_0 = 8$, $D_2 = 6$, $D_4 = 5$, $D_6 = 6$ and $D_8 = 8$, respectively. This nine-band oversampled SAF has an OSR of 178 %. Different decimation factors are used in individual subbands to satisfy the bandpass sampling theorem [21]. An iterative least-squares technique [20] is used to minimize the design performance criteria. These criteria are combinations of a filter bank reconstruction error and stopband energy of the analysis filter. A weighting constant can be used to trade off these criteria. Simulation studies using these oversampling SAFs have shown an improved performance (convergence rate) under colored input signals.

An important consideration in the oversampled SAF is the design of the filter banks. The performance of filter banks depends on the design of the prototype filter, its length, the number of subbands and the oversampling ratio. These topics will be further discussed in Section 5.3.

5.2.2 Nonuniform subband adaptive filtering

The nonuniform filter bank [22] is developed to achieve a better convergence performance by adapting the bandwidth of analysis filters. In general, nonuniform subbands can be classified as (i) spectrally flat subbands and (ii) subbands that account for transition bands when modeling an unknown system. In system identification applications, the spectral characteristics of unknown systems are usually not known *a priori*. One method to improve the performance of system identification is to initially use a uniform filter bank with a large number of filters to identify the unknown system. After the algorithm has

converged, those neighboring subbands with high spectral power are merged and those with low spectral power remain the same. This approach generates narrower subbands around the transition regions of the unknown system and wider subbands elsewhere [22]. Note that the decimation factor must be an integer that is smaller than the number of subbands. This selection results in an oversampled nonuniform filter bank that can reduce the in-band aliasing components, thus achieving faster convergence [22].

5.3 Filter bank design

In this section, we examine some other extensions to the filter bank design techniques described in Chapter 2 and study the design constraints imposed by these filter banks.

5.3.1 Generalized DFT filter banks

The DFT filter banks introduced in Section 2.6 are based on the basic definition of DFT that uses the time-index origin of $n = 0$ and the subband-index origin of $i = 0$. Therefore, the DFT filter bank has an even-channel stacking as shown in Figure 2.13, where the center frequencies of filters are even multiples of π/N . If the frequency origin is chosen as π/N for an N -channel filter bank, the center frequencies are

$$\omega_i = \frac{2\pi}{N}(i + 0.5), \quad i = 0, 1, \dots, N - 1. \quad (5.9)$$

The frequency shifting of 0.5 results in an odd-channel stacking, where the center frequencies of filters are odd multiples of π/N .

A generalized DFT uses these time and frequency references and can be applied to the impulse response $p(n)$ of the prototype analysis filter as

$$h_i(n) = p(n)e^{(j2\pi/N)(i+i_0)(n+n_0)}, \quad i = 0, 1, \dots, N - 1. \quad (5.10)$$

The symbols n_0 and i_0 are the new time and frequency indices, respectively. Typically, these indices are rational fractions that are less than 1. For example, $n_0 = 0$ and $i_0 = 0.5$ result in odd DFT-modulated bandpass filters $h_i(n)$. If N is an even number and the input signal $u(n)$ is real valued, only $N/2$ subbands are needed since the remaining $N/2$ subbands are complex conjugates of the first $N/2$ subbands.

The complex modulation model of the generalized DFT filter bank is shown in Figure 5.4, where $p(n)$ and $f(n)$ are the real-valued prototype filters for the analysis and synthesis filter banks, respectively. The center frequencies of filters are $\omega_i = 2\pi(i + i_0)/N$ for $i = 0, 1, \dots, N - 1$. By modulating the prototype filters $p(n)$ and $f(n)$ with the complex terms $W_N^{-(i+i_0)(n+n_0)}$ and $W_N^{(i+i_0)(n+n_0)}$, respectively, we obtain the complex-valued analysis filters $h_i(n) = p(n)W_N^{-(i+i_0)(n+n_0)}$ and synthesis filters $f_i(n) = f(n)W_N^{(i+i_0)(n+n_0)}$. Note that we usually use the same prototype filter $p(n)$ for designing both analysis and synthesis filters, as presented in Section 2.5. However, we use a different synthesis prototype filter $f(n)$ here to avoid band-edge effects in oversampling SAF cases.

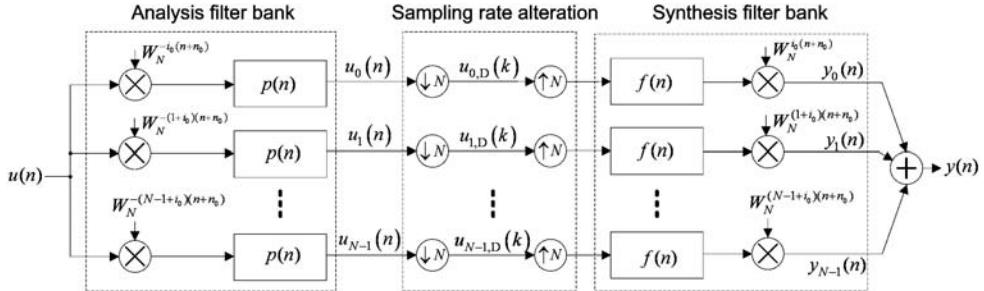


Figure 5.4 Complex modulation model of the generalized DFT filter bank

5.3.2 Single-sideband modulation filter banks

Sections 2.5 and 2.6 illustrate the design and analysis of cosine- and complex-modulated filter banks. The DFT and generalized DFT filter banks are based on quadrature modulation that results in complex-valued signals. Besides the cosine and DFT modulations, there is another class of modulation techniques called single-sideband modulation that results in real-valued subband filters. This section investigates the techniques required to generate these real-valued filter banks.

A single-sideband modulation produces real-valued signals that are often required in practical applications. A major difference between the generalized DFT and single-sideband modulation filter banks is the channel bandwidth ω_Δ . In the complex-modulated filter banks, subband filters occupy only the positive frequency range (i.e. $0 \leq \omega \leq 2\pi$). Therefore, the bandwidth is $\omega_\Delta = 2\pi/N$ in a uniformly spaced filter bank. The single-sideband modulation filter bank occupies both negative and positive frequency ranges (i.e. $-\pi \leq \omega \leq \pi$), thus resulting in a channel bandwidth of $\omega_\Delta = \pi/N$. Compared to Equation (5.9), the center frequencies of filter bank becomes $\omega_i = \pi(i + 0.5)/N$, $i = 0, 1, \dots, N - 1$, for single-sideband modulation. Because they have different bandwidths, different sampling rates can be employed for the complex- and real-valued filter banks.

The single-sideband modulated signals can be derived from the generalized DFT filter bank (see Figure 5.4) by modulating the complex-valued signal, as shown in the top diagram of Figure 5.5. Similarly, the single-sideband synthesis section can be described as shown in the bottom diagram of Figure 5.5. In these block diagrams, the Re(.) operators are used to derive real-valued signals at the output of the analysis and synthesis sections. An alternate way of realizing the DFT filter banks is the weighted overlap-add structure [23].

The overlap of subbands is an important consideration for designing filter banks in subband adaptive filtering. By using different filter design methods, we can obtain subbands with substantial overlapping (Figure 2.12) and slightly overlapping subbands (Figure 2.14). These figures show that spectral leakage to adjacent subbands is unavoidable as the realizable filter has transition band and finite stopband attenuation. Thus the objective of filter bank design is to minimize the spectral leakage or limit it to the adjacent bands only. The second important consideration is to select a right filter length to obtain adequate frequency and time resolutions, which are inversely related.

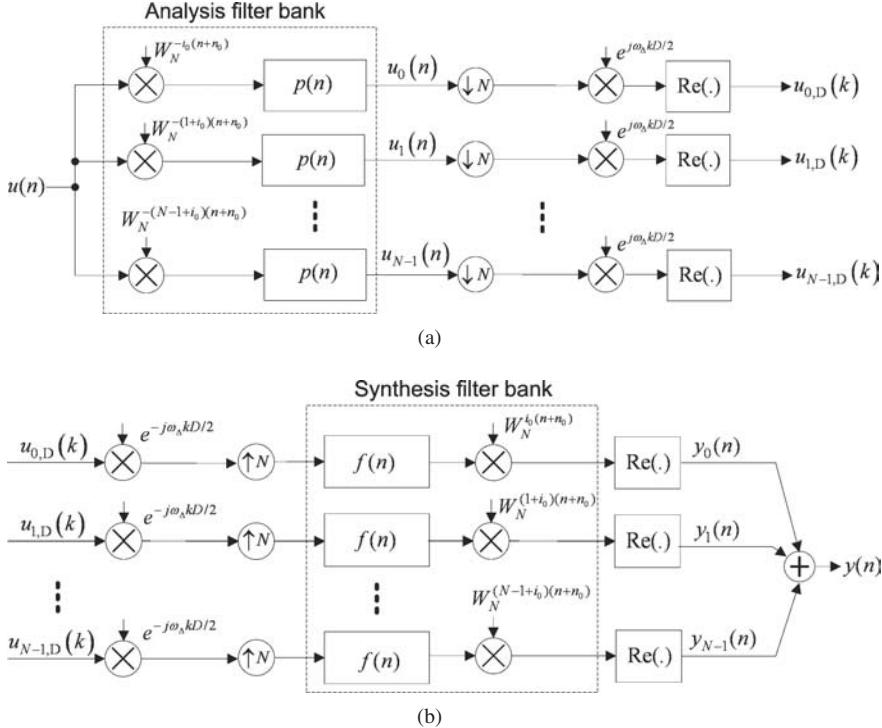


Figure 5.5 Block diagrams of the single-sideband modulation filter bank: analysis section (top) and synthesis section (bottom)

Critical sampling preserves the data rate of subband signals as the original fullband signal because the total number of all subband signal samples is equal to the number of fullband samples. When the decimation factor is less than the number of subbands (i.e. $D < N$), the filter bank is oversampled. In an oversampled filter bank, the total number of subband signal samples is larger than the number of fullband samples, resulting in a higher data rate. In the complex modulation, the sampling rate for each subband signal with bandwidth ω_Δ can be decimated by a decimation factor D that satisfies

$$D \leq \frac{2\pi}{\omega_\Delta}. \quad (5.11)$$

Similarly, we have

$$D \leq \frac{\pi}{\omega_\Delta} \quad (5.12)$$

for the real-valued modulation. In Equations (5.11) and (5.12), D is equal to the upper bound in the critical sampling cases and less than the upper bound in the oversampling systems. These two equations also show that the complex modulation allows a larger decimation factor as compared to the real-valued modulation. Therefore,

the complex-valued filter banks have a lower computational complexity as compared to the real-valued filter banks.

In summary, the characteristics of subband filters depend on the number of subbands N , the decimation factor D and the design of the analysis and synthesis filters. Filter design depends on the filter length, specifications of the prototype filter, such as ripples in the passband and attenuation in the stopband, and design methodology. Furthermore, the design of the analysis and synthesis filters is highly dependent in order to achieve perfect-reconstruction filter banks, which will be further explained in the next few sections.

5.3.3 Filter design criteria for DFT filter banks

Aliasing distortions can be examined in the frequency domain by expressing $z = e^{j\omega}$ and using the complex modulation function $W_N^l = e^{-j2\pi l/N}$. Therefore, the decimated subband signals given in Equation (2.5) can be rewritten as

$$U_{i,D}(e^{j\omega}) = \frac{1}{N} \sum_{l=0}^{N-1} H_i(e^{j(\omega-2\pi l)/N}) U(e^{j(\omega-2\pi l)/N}), \quad i = 0, 1, \dots, N-1. \quad (5.13)$$

The first term (i.e. $l = 0$) in the summation corresponds to the desired signal component and the remaining $N - 1$ terms are the undesired aliasing components.

In order to avoid aliasing in the frequency domain, the subband signals are filtered by the analysis filters $H_i(e^{j\omega})$. These analysis filters can be expressed as shifted versions of the prototype lowpass filter $P(e^{j\omega})$ as $P[e^{j(\omega-\omega_i)}]$, where $\omega_i = 2\pi i/N$ is the center frequency of the i th subband. The ideal magnitude response of the prototype analysis filter with the decimation factor D must satisfy the following ideal lowpass filter specification:

$$|H(e^{j\omega})| = \begin{cases} 1, & 0 \leq |\omega| < \pi/D, \\ 0, & \text{otherwise,} \end{cases} \quad (5.14)$$

where the stopband frequency starts from π/D . A similar filter specification is also imposed on the prototype synthesis filter $F(e^{j\omega})$.

For an overlapped filter bank design, the cascade of analysis and synthesis filters provides an additional degree of freedom for the overall system to function as an identity system with a certain delay. From Equations (2.6) and (2.7), the transfer function of the cascade system can be expressed as

$$\begin{aligned} Y(e^{j\omega}) &= \sum_{i=0}^{N-1} \frac{F_i(e^{j\omega})}{N} \left[H_i(e^{j\omega}) U(e^{j\omega}) + \sum_{l=1}^{N-1} H_i(e^{j\omega} W_N^l) U(e^{j\omega} W_N^l) \right] \\ &= \frac{1}{N} \sum_{i=0}^{N-1} F_i(e^{j\omega}) H_i(e^{j\omega}) U(e^{j\omega}) \\ &\quad + \frac{1}{N} \sum_{l=1}^{N-1} U(e^{j\omega} W_N^l) \sum_{i=0}^{N-1} F_i(e^{j\omega}) H_i(e^{j\omega} W_N^l), \end{aligned} \quad (5.15)$$

where the first term is the desired component and the second term consists of aliasing components caused by decimation and imaging components due to interpolation. Therefore, the inner summation, $\sum_{i=0}^{N-1} F_i(e^{j\omega}) H_i(e^{j\omega} W_N^l)$, in the second term must be summed to zero. To satisfy this condition, the stopband frequency of the analysis and synthesis filters must be lower than π/D . Further relaxation of this stopband frequency condition can be achieved by oversampling, which will be introduced later.

The first term in Equation (5.15) represents the desired component produced by the cascade of analysis and synthesis filters. A necessary condition for $Y(e^{j\omega}) = U(e^{j\omega})$ is that the analysis–synthesis filter product satisfies

$$\frac{1}{N} \sum_{i=0}^{N-1} F_i(e^{j\omega}) H_i(e^{j\omega}) = 1. \quad (5.16)$$

As noted earlier in Section 2.2.2, there is some delay or phase distortion associated with the analysis and synthesis filters. Therefore, we may consider a more relaxed condition that allows phase distortion by requiring only the magnitude of the analysis–synthesis filter product to be unity as

$$\frac{1}{N} \left| \sum_{i=0}^{N-1} F_i(e^{j\omega}) H_i(e^{j\omega}) \right| = 1. \quad (5.17)$$

Further simplification of Equation (5.17) can be achieved if only the adjacent subbands are overlapped as shown in Figure 5.6. In this case, only two terms in Equation (5.17) are required to satisfy the following frequency-domain constraint:

$$\frac{1}{N} |F(e^{j\omega}) H(e^{j\omega}) + F(e^{j(\omega-2\pi/N)}) H(e^{j(\omega-2\pi/N)})| = 1 \quad (5.18)$$

for $0 \leq \omega \leq 2\pi/N$. Figure 5.6 shows that the transition regions between adjacent bands of the analysis–synthesis filter product must be asymmetric at the crossover frequency of $\omega = \pi/N$ in order to satisfy Equation (5.18). At this crossover frequency of π/N , another filter design requirement is

$$\frac{1}{N} |F(e^{j\pi/N}) H(e^{j\pi/N})| = 0.5, \text{ for } \omega = \frac{\pi}{N}. \quad (5.19)$$

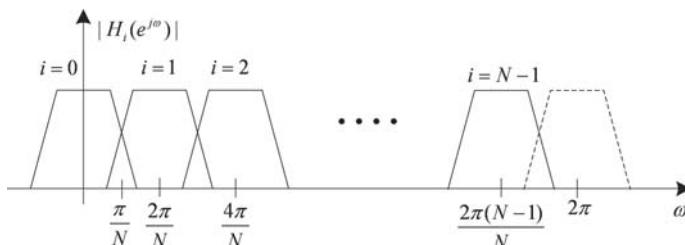


Figure 5.6 N subbands are equally spaced from 0 to 2π

Furthermore, if the transfer functions of analysis and synthesis filters are identical, they become $|F(e^{j\pi/N})| = |H(e^{j\pi/N})| = \sqrt{N/2}$ at the crossover frequency of π/N . If R channels (where $R > 2$) of the filter bank are overlapped, Equation (5.17) becomes

$$\frac{1}{N} \left| \sum_{i=0}^{R-1} F_i(e^{j\omega}) H_i(e^{j\omega}) \right| = 1 \text{ for } 0 \leq \omega \leq \frac{2\pi R}{N}. \quad (5.20)$$

The above filter design constraints are based on critically sampled filter banks, where the decimation factor D equals the number of subbands N . As D is decreased to a value that is smaller than N , the filter bank becomes oversampled with a higher data rate. As D approaches 1, it becomes a filter-bank-sum method, as stated in Reference [23]. Figure 5.7 shows the filter bank design for both critically sampled and oversampled cases. The dash lines indicate the undesired aliasing components.

Figure 5.7(a) shows that the stopband frequencies of the analysis filters (ω_h) and synthesis filters (ω_f) are equal and start from π/D (or π/N). However, when oversampling

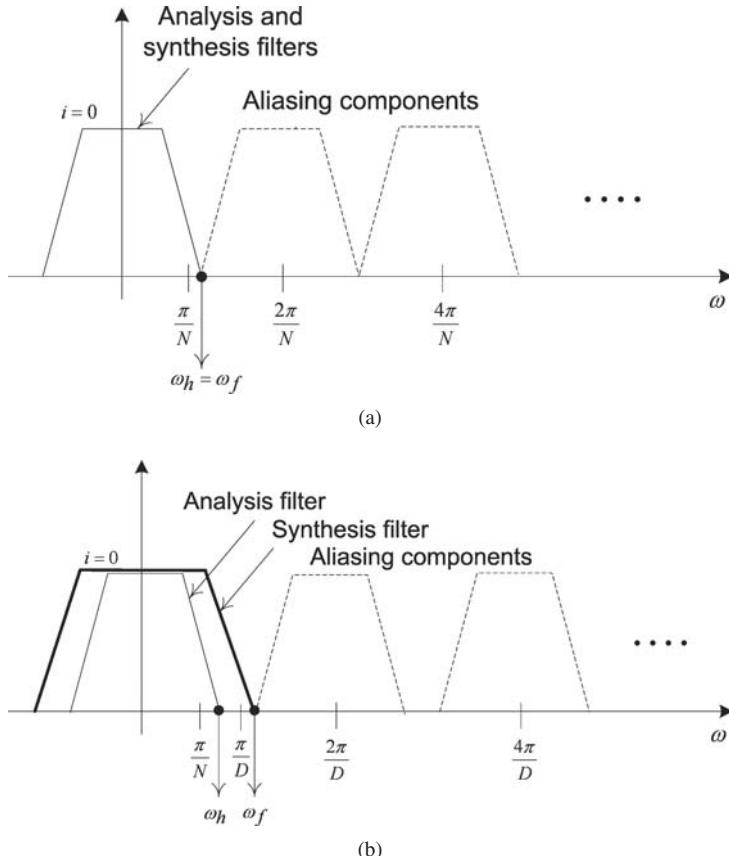


Figure 5.7 Analysis and synthesis filter bank designs for (a) critically sampled ($D = N$) and (b) oversampled ($D < N$) cases

is employed, as in Figure 5.7(b), the stopband frequency of an analysis filter satisfies

$$\frac{\pi}{N} < \omega_h \leq \frac{\pi}{D}. \quad (5.21)$$

Figure 5.7(b) also shows a spectral gap created by oversampling that allows the relaxed synthesis filter design. Therefore, the synthesis filter can be designed with the stopband frequency ω_f that satisfies

$$(\omega_f + \omega_h) \leq \frac{2\pi}{D}. \quad (5.22)$$

This constraint in Equation (5.22) implies that the synthesis filter can be designed with a stopband frequency $\omega_f \leq \pi/D$ instead of frequency bounded by Equation (5.21).

The above constraints in Equations (5.21) and (5.22) avoid the aliasing and imaging distortion of the oversampled filter banks. Example 5.1 shows a design case. In the special case of the filter-bank-sum method ($D = 1$), the synthesis prototype filter becomes an allpass filter, expressed as

$$|F(e^{j\omega})| = 1, \text{ for } |\omega| < \omega_h. \quad (5.23)$$

In this special case, the analysis–synthesis filter product in Equation (5.17) is reduced to

$$\frac{1}{N} \left| \sum_{i=0}^{N-1} H_i(e^{j\omega}) \right| = 1, \quad \forall \omega. \quad (5.24)$$

Unfortunately, oversampling of subband adaptive filters results in a slow asymptotic convergence. This is due to the fact that oversampling increases the eigenvalue spread of subband signals. As stated in Section 5.2.1, increasing the bandwidth of analysis filters relative to the synthesis filters in the oversampled SAF can increase the asymptotic convergence. Therefore, the relaxed constraints stated in Equations (5.21) and (5.22) are not suitable for oversampled subband adaptive filters. A design constraint $\omega_c \leq (\pi/D + \pi\delta)$ is imposed on the cutoff frequency of the analysis filters, where $\pi\delta$ is the bandwidth increment in radians [19]. The largest decimation factor D_{\max} without causing aliasing in the band of interest is given as [3]

$$D_{\max} = \left\lfloor \frac{2\pi}{\omega_h + \omega_f} \right\rfloor, \quad (5.25)$$

where $\lfloor \cdot \rfloor$ is the largest integer that is smaller than its argument. Using this decimation factor results in aliasing distortion in the analysis filter output, but this distortion can be removed by the corresponding synthesis filter, which has a tighter cutoff frequency.

A multicriteria design formulation is used to design a prototype FIR filter based on the window method for reducing the aliasing effect [24]. It shows that the Kaiser window outperforms other window methods. Furthermore, Kaiser windows can use the parameter α to make a tradeoff between the mainlobe width and sidelobe attenuation. The number of subbands also plays a significant role in controlling the in-band aliasing of the filter bank. Aliasing effects can be reduced by using a larger number of subbands. When the

number of subbands N equals the prototype filter length L , increasing the oversampling ratio also results in better aliasing cancellation. When the filter length is increased to $L = KN$, where K is an integer, aliasing decreases with the number of subbands when the oversampling ratio is below 2.5. Beyond the ratio of 2.5, no significant improvement can be achieved.

Example 5.1 (a complete listing is given in M-file `Example_5P1.m`)

This example shows that the oversampled filter banks can be designed with relaxed analysis and synthesis filter design constraints on their stopband frequencies. A window-based FIR filter design method is used to obtain the coefficients of a 128-tap prototype lowpass filter, which is used to generate the DFT filter banks across the frequency range from 300 Hz to 3500 Hz with a passband ripple of 1 dB and a stopband attenuation of 60 dB. The sampling frequency is 8000 Hz. The filter bank design is demonstrated and verified using the following MATLAB script:

```

...
% Create oversampled 8-band DFT filter bank

% Define parameter values
Fs = 8000; % Sampling frequency
Rp = 1; % Passband ripple = 1 dB
Rs = 60; % Stopband attenuation = -60 dB
L = 127; % Order of FIR filter, length = 128
wc = 1/6; % Set cutoff frequency at 1/6

% Design a prototype lowpass filter using a Chebyshev window with
% Rs decibels of relative sidelobe attenuation

b_os = fir1(L,wc,chebwin(L+1,Rs));

figure; % Plot the prototype lowpass filter
freqz(b_os); % Compute frequency response
title('Frequency response of prototype filter (oversampled)');

nbands = 8; % Number of filter bands
D = 6; % Decimation factor (oversampled)

len = length(b_os);
[H,G] = make_bank_DFT(b_os,nbands); % Design DFT filter banks

H = sqrt(D)*H;
G = sqrt(D)*G;
NFFT = 1024; % FFT size
dist_alias(H',G',nbands); % Compute distortion and aliasing
[H,w,delta_w] = FreqResp(H,NFFT); % Compute frequency response
Hz = H';
figure; % Plot results

```

```

subplot(3,1,[1 2]), hold on;
for k = 1:nbands
    if mod(k,2)==0
        plot(w/pi,20*log10(abs(Hz(k,:))+eps));
    else
        plot(w/pi,20*log10(abs(Hz(k,:))+eps),':');
    end
end
axis([0 2 -120 10]); box on; ylabel('Gain (dB)');
title('Frequency response of oversampled 8-band DFT filter bank
');

% Distortion function

Tz = sum(abs(Hz).^2);
subplot(3,1,3); plot(w/pi,10*log10(Tz));
xlabel('Frequency,\omega (\pi)'); ylabel('Gain (dB)');
axis([0 2 0 20]);
title('Combined distortion response (Oversampled)');

```

5.3.4 Quadrature mirror filter banks

So far, we have considered only the uniform filter banks. A multistage structure has not been restricted to uniform filter banks. The type of filter banks that can be employed in multistage structures is the quadrature mirror filter (QMF) bank. Figure 5.8 shows that the two-band QMF bank consists of an analysis lowpass filter $h_0(n)$ and a highpass filter $h_1(n)$ followed by the decimation factor of two at each subband. These lowpass and highpass filters divide the frequency range into a lowpass region $[0 \rightarrow \pi/2]$ and a highpass region $[\pi/2 \rightarrow \pi]$, respectively. The synthesis filters $f_0(n)$ and $f_1(n)$ reconstruct the subband signals into a single fullband output $y(n)$ that approximates the delayed version of the original signal $u(n)$. Note that $h_0(n)$, $h_1(n)$, $f_0(n)$ and $f_1(n)$ can be related to a common prototype lowpass filter $p(n)$ by using $h_0(n) = p(n)$, and these filters can be designed to remove aliasing and imaging distortion completely for perfect reconstruction.

Further decomposition of lowpass and highpass signals is required to obtain filter banks with different bandwidths. For example, the two-stage uniform filter bank is shown in Figure 5.9 and the three-stage tree structure with octave-spaced filter bank is shown in Figure 5.10. These figures show that analysis filters at different stages are operated at different sampling rates. Therefore, the number of subbands N is a power of 2 for

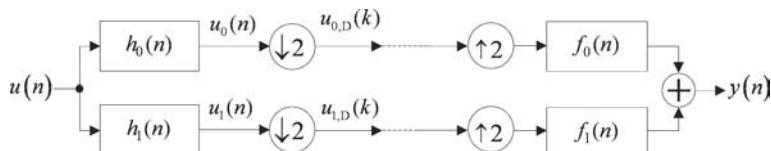


Figure 5.8 Two channels (lowpass and highpass) of analysis and synthesis filters

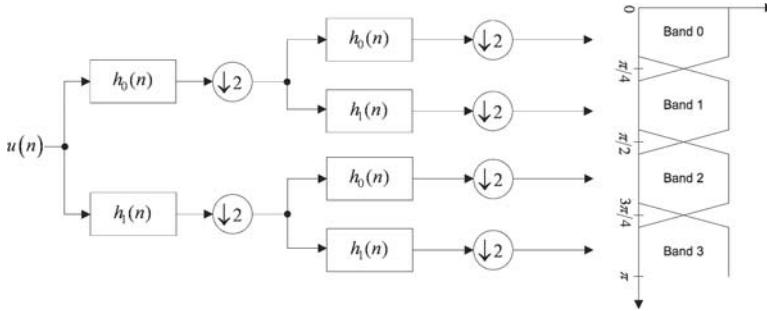


Figure 5.9 A two-stage uniform filter bank results in four uniformly spaced channels

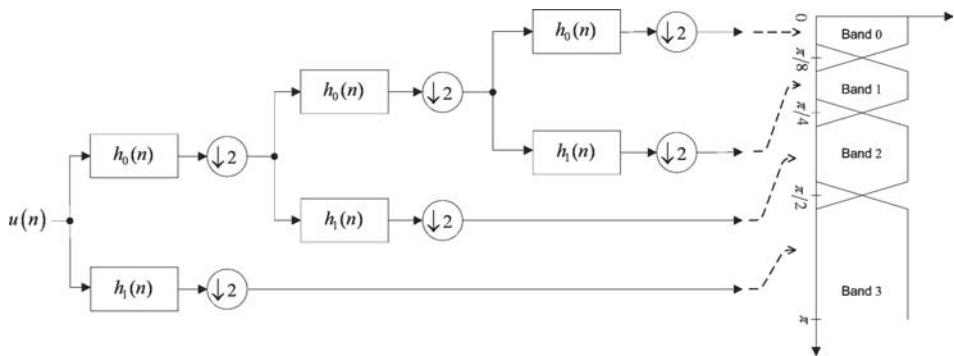


Figure 5.10 A three-stage tree structure with an octave-spaced filter bank

the QMF structures. At the first stage, outputs of filters are operated at $f_s/2$ due to the downsampling by a factor of 2. In general, sampling rates at the i th stage are decimated to $f_s/2^i$. As shown in Figure 5.10, outputs of filters at stage 3 are operated at $f_s/8$, at stage 2 are operated at $f_s/4$ and at stage 1 are operated at $f_s/2$. In order to obtain the same delay in each subband output, appropriate delay units have to be inserted in the high-frequency bands.

The computational complexity for analysis and synthesis filters can be computed by multiplying the number of coefficients with the sampling rate at which the filtering is calculated. In Figure 5.10, the computational complexity for the octave-spaced analysis–synthesis filter bank is L (coefficients) \times 2 (filters) \times 2 (analysis and synthesis) \times ($f_s/8 + f_s/4 + f_s/2$) (which represents the decimated sampling frequency at each stage) operations per second. However, the multistage uniformly spaced filter bank shown in Figure 5.9 has a computational complexity of L (coefficients) $\times 2f_s \times i$ (stages).

The aliasing effects caused by the spectral leakage can be removed in the filter bank framework. The aliasing distortion occurring in the analysis section can be cancelled exactly by the imaging effects generated in the synthesis section. Thus the properly designed perfect reconstruction analysis–synthesis framework contains no distortion. The two-channel perfect reconstruction filter bank shown in Figure 5.8 can be described in

the z -domain as follows:

$$U_{i,D}(z) = \frac{1}{2} [H_i(z^{1/2}) U(z^{1/2}) + H_i(-z^{1/2}) U(-z^{1/2})], \quad i = 0, 1, \quad (5.26)$$

$$Y(z) = U_{0,D}(z^2) F_0(z) + U_{1,D}(z^2) F_1(z). \quad (5.27)$$

Substituting Equation (5.26) into Equation (5.27) results in

$$\begin{aligned} Y(z) &= \frac{1}{2} [H_0(z) F_0(z) + H_1(z) F_1(z)] U(z) \\ &\quad + \frac{1}{2} [H_0(-z) F_0(z) + H_1(-z) F_1(z)] U(-z), \end{aligned} \quad (5.28)$$

where $H_0(z)$ is the lowpass filter with a cutoff frequency of $\pi/2$ and its mirror filter $H_1(z) = H_0(-z)$ is the highpass filter with the same cutoff frequency. The first term on the right side of Equation (5.28) is the desired spectral component at ω , and the second term represents the aliasing components at frequency from $\omega - \pi$. The main objective of designing analysis and synthesis filters is to force the second term to zero, which requires $[H_0(-z) F_0(z) + H_1(-z) F_1(z)] = 0$. This aliasing cancellation design constraint can be achieved by designing the synthesis filter as

$$F_0(z) = \frac{-H_1(-z) F_1(z)}{H_0(-z)}, \quad (5.29)$$

such that the aliasing terms in the analysis section is exactly cancelled by the imaging terms in the synthesis section. Therefore, the synthesis filter is designed according to $H_0(z)$ to satisfy the aliasing cancellation condition.

From Equation (5.29), if $F_1(z) = -2H_0(-z)$, we have $F_0(z) = 2H_0(z)$. The impulse responses of the analysis and synthesis filters are based on the impulse response of the prototype filter $h_0(n) = p(n)$, where $h_1(n) = (-1)^n h_0(n)$, $f_1(n) = -2(-1)^n h_0(n)$ and $f_0(n) = 2h_0(n)$. Applying the z -transform to $h_0(n)$, $h_1(n)$, $f_0(n)$ and $f_1(n)$, and substituting the obtained transfer functions into Equation (5.28), the input–output relationship of the filter bank is obtained as

$$Y(z) = [H_0^2(z) - H_0^2(-z)] U(z). \quad (5.30)$$

Thus the frequency response of the filter bank output is given as

$$Y(e^{j\omega}) = [H_0^2(e^{j\omega}) - H_0^2(e^{j(\omega+\pi)})] U(z). \quad (5.31)$$

This equation shows that a unity gain system can be achieved if $|H_0^2(e^{j\omega}) - H_0^2(e^{j(\omega+\pi)})| = 1$ for all frequencies ω . However, due to the extra delay introduced by the analysis and synthesis filters, there is an overall system delay of Δ samples. If all the analysis and synthesis filters are linear, the term $[H_0(z) F_0(z) + H_1(z) F_1(z)]$ will not have phase distortion. The first term on the right side of Equation (5.28) can be simplified to $z^{-\Delta} U(z)$, which is the delayed version of the input signal. A linear phase filter is commonly used in designing the prototype filter for the QMF, as shown in Example 5.2.

In summary, the process of implementing QMF starts from the design of a linear-phase prototype lowpass filter $h_0(n)$ with a cutoff frequency of $\pi/2$. This lowpass filter must satisfy a power complementary design constraint of $|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega+\pi)})|^2 = 1$. However, this constraint can only be approximated when the linear FIR filters are used in the realization of analysis and synthesis filters [20].

Example 5.2 (a complete listing is given in M-file `Example_5P2.m`)

As explained in Section 5.3.4, a QMF bank can be used to realize uniformly spaced and octave-spaced filter banks. In this example, we build these structures by cascading the QMF lowpass and highpass analysis filters. In addition, we examine the power complementary property of the QMF and observe that the aliasing can be completely removed using the synthesis filters described in Section 5.3.4. The uniformly spaced filter bank design is demonstrated using the following MATLAB script:

```
% QMF 32C[23]
% Number of taps = 32
% Transition bandwidth = 0.0625
% Reconstruction error = 0.009 dB
% Stopband attenuation = 51 dB

% Define first half (16) of symmetric filter coefficients
h0_32C_half = [.69105790e-3 -0.14037930e-2 -0.12683030e-2
.4234195e-2 .1414246e-2 -0.9458318e-2 -0.1303859e-3 .1798145e-1
-0.4187483e-2 -0.3123862e-1 .1456844e-1 .5294745e-1 -0.3934878e-1
-0.9980243e-1 .1285579 .46640530];

% Create both lowpass and highpass symmetric FIR filter from
% the first half of coefficients
h0_32C = [h0_32C_half fliplr(h0_32C_half)]; % Lowpass filter
h1_32C = ((-1).^(0:length(h0_32C)-1)).*h0_32C; % Highpass filter
figure; % Plot frequency response
freqz(h0_32C); hold on;
freqz(h1_32C);
title('Frequency response of QMF 32C');
...
% (a) Plot a uniform filter bank
B1 = h0_32C; % Lowpass filter
B2 = h1_32C; % Highpass filter
B10 = zeros(1, 2*length(B1));
B10(1: 2: length(B10)) = B1;
B11 = zeros(1, 2*length(B2));
B11(1: 2: length(B11)) = B2;
C0 = conv(B1, B10); C1 = conv(B1, B11);
C2 = conv(B2, B10); C3 = conv(B2, B11);

% Compute the frequency responses of digital filters
N=1024; % FFT size
```

```

[H00z, w] = freqz(C0, 1, N);
h00 = abs(H00z);M00 = 20*log10(h00);
[H01z, w] = freqz(C1, 1, N);
h01 = abs(H01z);M01 = 20*log10(h01);
[H10z, w] = freqz(C2, 1, N);
h10 = abs(H10z);M10 = 20*log10(h10);
[H11z, w] = freqz(C3, 1, N);
h11 = abs(H11z);M11 = 20*log10(h11);

figure;
plot(w/pi, M00,'r-','LineWidth',2); hold on;
plot(w/pi, M01,'g--','LineWidth',2);
plot(w/pi, M10,'b--','LineWidth',2);
plot(w/pi,M11,'c-','LineWidth',2);
legend('Band #1','Band #2','Band #3','Band #4','Location','Best');
xlabel('\omega/\pi'); ylabel('Gain (dB)');grid
title('Uniform 4 subbands based on QMF');

```

5.3.5 Pseudo-quadrature mirror filter banks

A pseudo-QMF (PQMF) bank is commonly used in MPEG-1 and MPEG-2 audio coding [25] for performing time-to-frequency mapping. As we have seen in Section 2.5, a PQMF does not achieve perfect reconstruction, hence the name ‘pseudo’. Unlike the QMF banks, the PQMF does not require a cascade structure to divide the frequency spectrum into N uniformly spaced subbands. A prototype lowpass filter $p(n)$ is first designed and the rest of the $N - 1$ filters are derived by modulating the prototype filter with the cosine terms $\cos[(\pi/N)(i + 0.5)(n - (L - 1)/2) + \theta_i]$ for $i = 0, 1, \dots, N - 1$, where $\theta_i = (-1)^i \pi/4$ and L is the length of prototype filter $p(n)$. The synthesis filters are then taken as the time-reversed versions of the analysis filters. As shown in Figure 2.11(a), the passband of the prototype lowpass filter $p(n)$ is from 0 to $\pi/2N$ and is bandlimited to π/N . The frequency responses of the analysis filters contain $2N$ complex-modulated versions of the prototype filter $P(\omega)$ that is shifted along the positive and negative frequency axes between $-\pi$ and π by the step of $\pm[(i + 0.5)/N]\pi$. The right-shifted version forms a conjugate pair with the left-shifted version, leading to the analysis and synthesis filters given in Equation (2.35).

The phase θ_i is determined by an anti-aliasing condition between adjacent subbands that satisfies the following relationship:

$$\theta_i - \theta_{i-1} = \frac{\pi}{2}(2q + 1) \text{ for } i = 0, 1, \dots, N - 1, \quad (5.32)$$

where q is an integer. This equation implies that the phase difference must equal odd multiples of $\pi/2$. In addition, a power complementary design constraint, as given by Equation (2.36), is also satisfied in the pseudo-QMF. In general, the pseudo-QMF has a near-perfect reconstruction due to the aliasing error. Example 5.3 shows the pseudo-QMF prototype filter used in MPEG-1.

Example 5.3 (a complete listing is given in M-file `Example_5P3.m`)

The MPEG-1 audio coder uses a 32-channel pseudo-QMF filter bank. The 513-tap symmetric prototype FIR filter $p(n)$ is given in References [25] and [26]. The phase of the pseudo-QMF used in MPEG is $\theta_i = (L - 1 - N)(i + 0.5)\pi/2N$, which satisfies the aliasing cancellation condition stated in Equation (5.32). This example examines the impulse response and frequency response of the prototype filter and its filter bands. In particular, we examine the passband frequency of the prototype filter. We form the filter bank by modulating the prototype filter with a frequency shift of π/N . The 32-band pseudo-QMF filter-bank design and implementation are illustrated by the following MATLAB script:

```
% Prototype filter h is extracted from MPEG-1 audio coder
% (Refer to the MATLAB file available in the companion CD for the
% complete listing of coefficients)

% Create 32-band PQMF filter bank

nbands = 32;
len = length(h);
[H,G] = make_bank_3(h,nbands); % Create filter banks
dist_alias(H,G,nbands); % Compute distortion and aliasing

% Plot frequency response of filter bank

[Hz,w] = FreqResp(H',NFFT); % Compute frequency response
Hz = Hz.';
figure; subplot(3,1,[1 2]), hold on;
for k = 1:nbands
    if mod(k,2)==0
        plot(w/pi,20*log10(abs(Hz(k,:))+eps));
    else
        plot(w/pi,20*log10(abs(Hz(k,:))+eps),':');
    end
end
axis([0 1 -120 10]); box on; ylabel('Gain (dB)');
title('Frequency response of 32-band PQMF filter bank');

% Distortion function

Tz = sum(abs(Hz).^2);
subplot(3,1,3); plot(w/pi,10*log10(Tz));
xlabel('Frequency, \omega (\pi)'); ylabel('Gain (dB)');
axis([0 1 6.01 6.03]);
title('Combined distortion response');
```

A prototype filter can also be designed by taking into consideration the steady-state MSE when used in an SAF. A nonlinear optimization procedure is introduced in Reference

[27] to generate an optimal prototype filter $p_o(n)$, which satisfies the following cost function:

$$\begin{aligned} \xi = & \alpha_1 \int_0^{\pi/N} \left(|P_o(e^{-j\omega})|^2 + |P_o(e^{-j(\omega-\pi/N)})|^2 - 1 \right)^2 d\omega \\ & + \alpha_2 \int_{\pi/N+\gamma}^{\pi} |P_o(e^{-j\omega})|^2 d\omega + \alpha_3 \left[\frac{\lambda_{\max}}{\lambda_{\min}} - 1 \right], \end{aligned} \quad (5.33)$$

where $P_o(e^{j\omega})$ is the frequency response of the optimal prototype filter, $\alpha_1, \alpha_2, \alpha_3$ are relative weights to control the reconstruction error, aliasing error and eigenvalue spread, respectively, and γ is the small transition bandwidth. The first two weights (α_1, α_2) determine the minimization of the steady-state MSE and the last weight (α_3) determines the convergence rate of the SAF. Therefore, the optimized prototype filter designed based on Equation (5.33) has the ability to trade off the steady-state MSE and convergence rate of SAF algorithms.

5.3.6 Conjugate quadrature filter banks

Like the QMF bank, conjugate quadrature filters [25] consist of an analysis lowpass filter $h_0(n)$ with a cutoff frequency of $\pi/2$. The highpass filter $h_1(n)$ is obtained by modulating the time reverse of the lowpass filter $h_0(n)$ by $(-1)^n$ to get $h_1(n) = (-1)^n h_0(L-1-n)$. The synthesis filters are the time reverses of their respective analysis filters as $f_0(n) = 2h_0(L-1-n)$ and $f_1(n) = 2h_1(L-1-n)$. The transfer functions of the analysis and synthesis filters are

$$\begin{aligned} H_1(z) &= z^{-(N-1)} H_0(-z^{-1}), \\ F_0(z) &= 2z^{-(N-1)} H_0(z^{-1}), \\ F_1(z) &= 2z^{-(N-1)} H_1(z^{-1}). \end{aligned} \quad (5.34)$$

The first equation implies that $H_1(-z) = -z^{-(N-1)} H_0(z^{-1})$ and $H_1(z^{-1}) = z^{N-1} H_0(-z)$. Substituting these equations into Equation (5.28) results in zero aliasing.

The perfect reconstruction condition shows that the analysis filter $H_0(z)$ must satisfy the following constraint:

$$H_0(z)H_0(z^{-1}) + H_0(-z)H_0(-z^{-1}) = 1. \quad (5.35)$$

The input–output relationship of the conjugate quadrature filter bank is stated as

$$Y(z) = z^{-(N-1)} [H_0^2(z) + H_0^2(-z)] X(z). \quad (5.36)$$

Unlike the QMF, the conjugate quadrature filter always satisfies the power complementary design constraint of $|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega+\pi)})|^2 = 1$.

5.4 Case study: Proportionate subband adaptive filtering

The proportionate adaptation technique is capable of dealing with a sparse environment, while the subband adaptive filtering technique improves convergence under colored signals. This case study investigates the integration of these two methods into a single algorithm to improve the performance of adaptive filters in many applications such as network and acoustic echo cancellation and feedback cancellation in hearing aids, as described in Chapter 1, in which sparseness is an additional factor that must be taken into account.

5.4.1 Multiband structure with proportionate adaptation

The idea of proportionate adaptation is to assign an individual step size to each tap weight in proportion to the value of the last estimated tap weight [28]. It uses proportionately larger step sizes for tap weights with larger values and reduces the step sizes to near zero (or no update) for smaller tap weights, thus resulting in a faster convergence when the unknown system to be identified has a sparse impulse response. A sparse impulse response is one that has a small percentage of its components with a significant magnitude while the rest are small or zero [28, 29].

The proportionate adaptation technique can be seamlessly combined with the multiband-structured SAF (MSAF) [30]. We have briefly introduced the multiband adaptation technique in Section 5.1.2. A detailed description of the method is presented in Chapter 6. The main idea of the MSAF algorithm is to adapt the fullband filter using the set of subband signals, in which the proportionate adaptation technique can be applied to the adaptation process. The proportionate MSAF (PMSAF) algorithm is summarized in Table 5.3. Each tap weight of the fullband adaptive filter $\mathbf{w}(k)$ is

Table 5.3 The proportionate MSAF algorithm [31]

Step-size adaptation:

Introduce a diagonal control matrix

$$\mathbf{G}(k) = \text{diag}[g_0(k), g_1(k), \dots, g_{M-1}(k)] \quad (5.37)$$

where diagonal elements of the control matrix are given by

$$g_m(k) = \frac{1 - \alpha}{2L} + (1 + \alpha) \frac{|w_m(k)|}{2\|\mathbf{w}(k)\|_1 + \varepsilon}, \quad m = 0, 1, \dots, M - 1. \quad (5.38)$$

Here, ε is a small constant and $\|\cdot\|_1$ is the l_1 -norm operator.

Subband weight vector adaptation:

$$\mathbf{w}(k + 1) = \mathbf{w}(k) + \mu \mathbf{G}(k) \mathbf{U}(k) \boldsymbol{\Lambda}^{-1}(k) \mathbf{e}_D(k), \quad (5.39)$$

where $\boldsymbol{\Lambda}(k) = \text{diag}[\mathbf{U}^T(k) \mathbf{G}(k) \mathbf{U}(k) + \delta \mathbf{I}]$ is the normalization matrix, $\mathbf{e}_D(k)$ is the error vector at a decimated rate and $\mathbf{U}(k) = [\mathbf{u}_0(k), \mathbf{u}_1(k), \dots, \mathbf{u}_{N-1}(k)]$ is the subband data matrix with the i th subband regressor $\mathbf{u}_i(k) = [u_i(kN), u_i(kN - 1), \dots, u_i(kN - M_S + 1)]^T$ for $i = 0, 1, \dots, N - 1$.

updated independently by adjusting the step size in proportion to its previous estimate. Notice that $g_m(k)$ in Equation (5.38) is the gain applied to the step size for individual tap weights. Clearly, the gain is dependent on the relative scaling of the tap weights as opposed to their absolute value due to the normalization term, $2||\mathbf{w}(k)||_1 + \varepsilon$. The parameter α is used in the control matrix to adjust the scaling of the tap weights. Good choices of α are 0 and -0.5 .

5.4.2 MATLAB simulations

In the following example, we use system identification problems to compare the performance of PMSAF with (i) NLMS, (ii) proportionate NLMS (PNLMS) and (iii) improved PNLMS (IPNLMS) algorithms [29]. The function M-files for implementing the NLMS, IPNLMS and PMSAF algorithms are `NLMSadapt.m`, `IPNLMSadapt.m` and `PMSAFadapt.m`, respectively, with the associated initialization functions. Partial listings of the latter two function M-files that illustrate the implementation of IPNLMS and PMSAF algorithms are shown below. The complete function M-files of these algorithms are available in the companion CD.

The IPNLMS algorithm

The IPNLMS algorithm extends the NLMS algorithm by using different step sizes for different coefficients. The individual step size is computed based on the magnitude of each coefficient. The parameter `alpha` is used in the control matrix to adjust the scaling of the tap weights. In this simulation, `alpha = 0` is used. The normalization term, `(u.*G)'*u + delta`, makes the convergence rate independent of the signal power. An additional constant `delta` is used to avoid division by zero in normalization operations.

```

...
S = IPNLMSinit(zeros(M,1),mu); % Initialization
S.unknownsys = b;
[yn,en,S] = IPNLMSadapt(un,dn,S);% Perform IPNLMS algorithm
...

function S = IPNLMSinit(w0,mu,alpha,leak)

S.coeffs      = w0(:);          % Coefficients of FIR filter
S.step        = mu;             % Step size of IPNLMS algorithm
S.leakage     = leak;           % Leaky factor for leaky IPNLMS
S.iter        = 0;              % Iteration count
S.alpha       = alpha;          % Scaling for coefficients
S.AdaptStart  = length(w0);    % Running effect of adaptive
                               % filter, minimum M

function [yn,en,S] = IPNLMSadapt(un,dn,S)

M = length(S.coeffs);          % Length of FIR filter

```

```

mu = S.step;                                % Step size of IPNLMS algorithm
                                              % (between 0 and 2)
leak = S.leakage;                            % Leaky factor of leaky IPNLMS
alpha = S.alpha;                             % Scaling for coefficients
delta = 1e-4;                               % Small constant
AdaptStart = S.AdaptStart;
w = S.coeffs;                               % Coefficients of FIR filter

u = zeros(M,1);                            % Input signal vector
ITER = length(un);                         % Length of input sequence
yn = zeros(1,ITER);                         % Initialize output vector to zero
en = zeros(1,ITER);                         % Initialize error vector to zero
...
for n = 1:ITER
    u = [un(n); u(1:end-1)];             % Input signal vector
                                         % [u(n),u(n-1),...,u(n-L+1)]';
    yn(n) = w'*u;                      % Estimated output
    en(n) = dn(n) - yn(n);            % Estimation error
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b;% System error norm (normalized)
    end
    if n >= AdaptStart
        g = (1-alpha)/(2*M) +
            abs(w)*(1+alpha)/(2*sum(abs(w))+delta);
        G = repmat(g,1);
        w = (1-mu*leak)*w + (G.*u.*mu*en(n))/((u.*G)'*u + delta);
        S.iter = S.iter + 1;
    end
end

```

The PMSAF algorithm

The PMSAF algorithm listed in Table 5.3 is implemented in MATLAB. The PMSAF algorithm adapts the fullband filter using the set of subband signals, in which the proportionate adaptation technique is applied to the adaptation process. The main difference between the PMSAF and IPNLMS algorithms is that the PMSAF algorithm uses subband signals in the adaptation process that are similar to those of the affine projection algorithm in the time domain. Similar to the IPNLMS algorithm, the individual step size is computed based on the magnitude of each coefficient. The parameters N , L , M denote the number of subbands, length of analysis filter and length of adaptive weight vector, respectively.

```

...
S = PMSAFinit(zeros(M,1),mu,N,L); % Initialization
S.unknownsys = b;
```

```
[en,S] = PMSAFadapt(un,dn,S); % Perform PMSAF algorithm
...
function [en,S] = PMSAFadapt(un,dn,S)

M = length(S.coeffs); % Length of adaptive weight vector
[L,N] = size(S.analysis); % Number and subbands and length
                           % of analysis filter
mu = S.step; % Step size of PMSAF algorithm
alpha = S.alpha; % Scaling for coefficients
delta = S.delta; % Small constant
AdaptStart = S.AdaptStart;
H = S.analysis; % Analysis filter
F = S.synthesis; % Synthesis filter
w = S.coeffs; U = zeros(M,N); % Initialize adaptive filtering
a = zeros(L,1); d = zeros(L,1); % Initialize analysis filtering
A = zeros(L,N); % Initialize synthesis filtering
z = zeros(L,1); % Initialize synthesis filtering
...
for n = 1:ITER
    d = [dn(n); d(1:end-1)]; % Update tapped-delay line of d(n)
    a = [un(n); a(1:end-1)]; % Update tapped-delay line of u(n)
    A = [a, A(:,1:end-1)]; % Update buffer
    ...
    if (mod(n,N)==0) % Tap-weight adaptation at
                      % decimated rate
        U1 = (H'*A)';
        U2 = U(1:end-N,:);
        U = [U1', U2'];
        dD = H'*d; % Subband data matrix
        eD = dD - U'*w; % Partitioning d(n)
        if n >= AdaptStart % Error estimation
            g = (1-alpha)/(2*M) +
                abs(w)*(1+alpha)/(2*sum(abs(w))+1e-4);
            G = repmat(g,1,N); % Replicate and tile an array
            w = w + (U.*G)*(eD./((sum(U.*G.*U)+delta)').*mu);
            % Adaptation of weight vector
            S.iter = S.iter + 1;
        end
        z = F*eD + z;
        en(n-N+1:n) = z(1:N);
        z = [z(N+1:end); zeros(N,1)];
    end
end

```

A script M-file `ECdemo.m` (available in the companion CD) for echo cancellation has been written to call these MATLAB functions in order to compare the performances of these adaptive algorithms. Two different (sparse and dispersive) impulse responses of

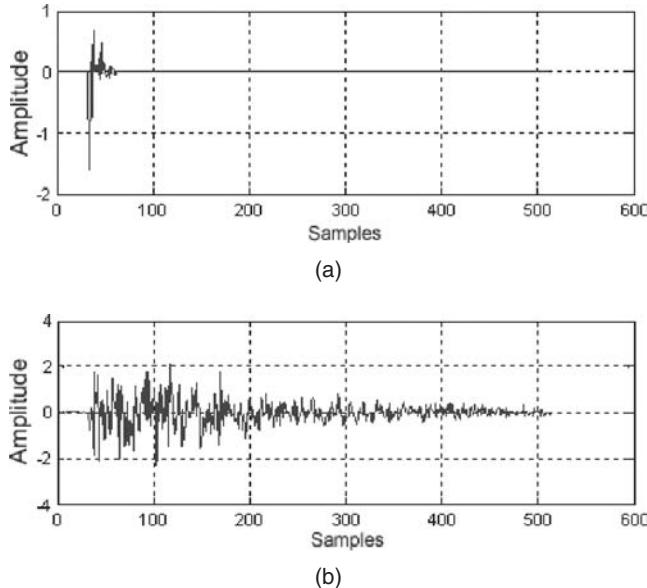


Figure 5.11 Two impulse responses used for the simulation: (a) sparse and (b) dispersive

length 512, as shown in Figure 5.11, are used to represent the acoustic echo path $B(z)$ in the simulation.

A white noise is added to the output of the unknown system with a 40 dB signal-to-noise ratio. Three types of excitation signals (white noise, colored noise and speech) are used in the simulations. The PMSAF algorithm uses a four-channel PQMF bank with a filter length of $L = 32$. An identical step size of $\mu = 0.1$ is used for all algorithms. The MSE learning curves are obtained by ensemble averaging over 100 independent trials.

The MSE plots of these algorithms for the white noise input are shown in Figure 5.12 for the sparse and dispersive impulse responses. In sparse environments with white excitation, proportionate-based algorithms (i.e. PMSAF, PNLMs and IPNLMS) outperform the NLMS algorithm. However, when the impulse response is dispersive under white excitation, there is no advantage in using proportionate adaptation. These results agree with the well-known characteristics of proportionate adaptation that uses larger step sizes for larger tap weights and performs minimum (or no) adaptation for near-zero tap weights to achieve faster convergence and smaller misadjustment.

Figures 5.13(a) and (b) show simulation results of colored input signals under the sparse and dispersive impulse responses, respectively. The PMSAF algorithm outperforms the IPNLMS algorithm in achieving smaller steady-state MSE. Similarly, Figure 5.14 shows misadjustment curves of different algorithms using speech as input signals for different impulse responses. The PMSAF algorithm performs consistently better under both sparse and dispersive impulse responses with colored and speech signals. However, the increase in the convergence rate comes at the expense of additional computational complexity.

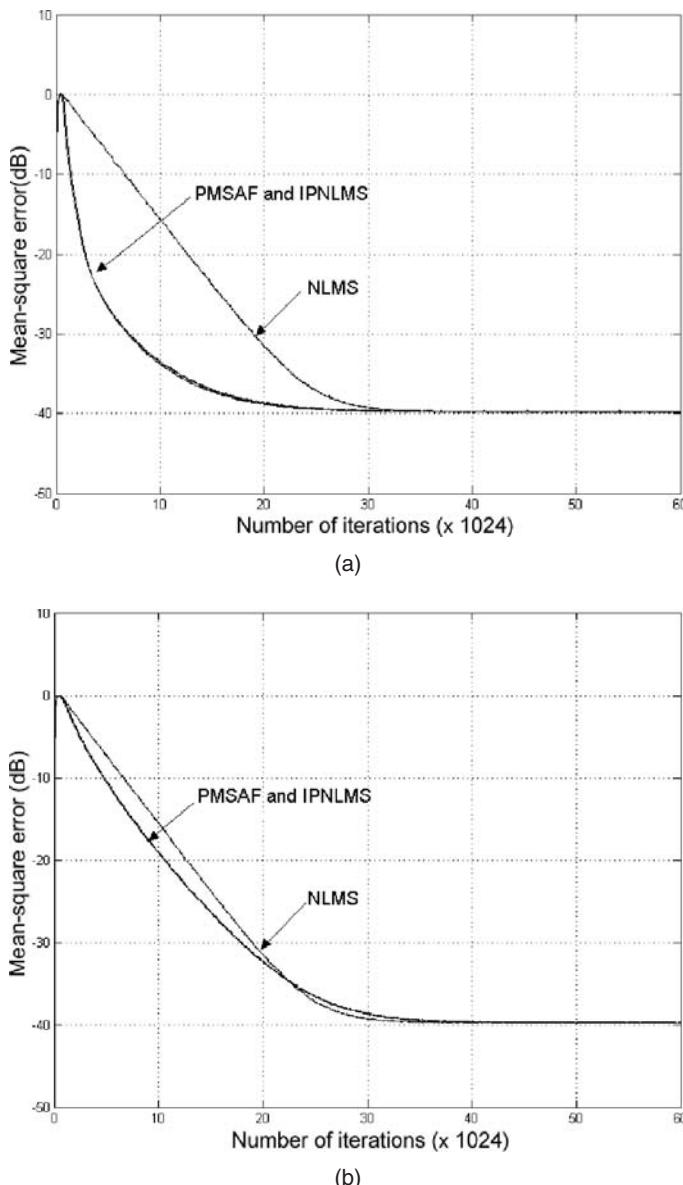


Figure 5.12 MSE plots for the PMSAF, NLMS and IPNLMS algorithms using white excitation under different impulse responses: (a) sparse and (b) dispersive

5.5 Summary

This chapter summarized some important techniques for improving the performance of subband adaptive filters and discussed their design constraints. The affine projection, variable step size and selective coefficient update algorithms were applied to the critically sampled SAFs. The oversampled and nonuniform SAFs were introduced to reduce

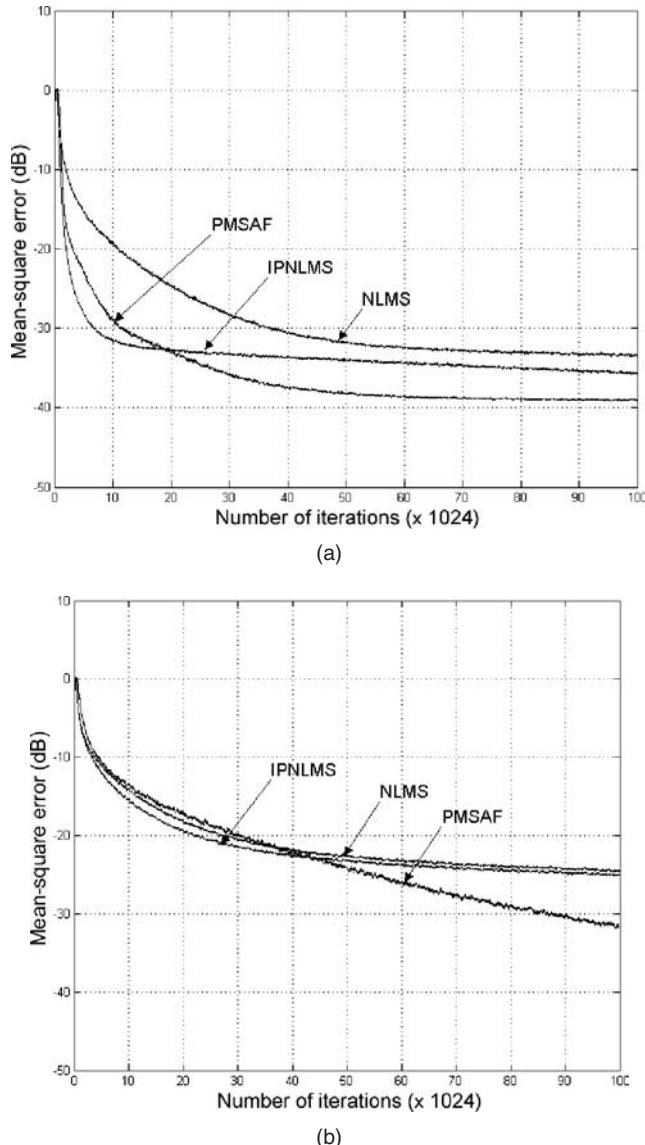


Figure 5.13 MSE plots for the PMSAF, NLMS and IPNLMS algorithms using colored excitation under different impulse responses: (a) sparse and (b) dispersive

in-band aliasing. Several important filter-bank design constraints and techniques were also discussed. In summary, this chapter provided a preliminary introduction to explore new directions in designing efficient SAF structures to achieve faster convergence, lower computational complexity and efficient methods for updating tap weights.

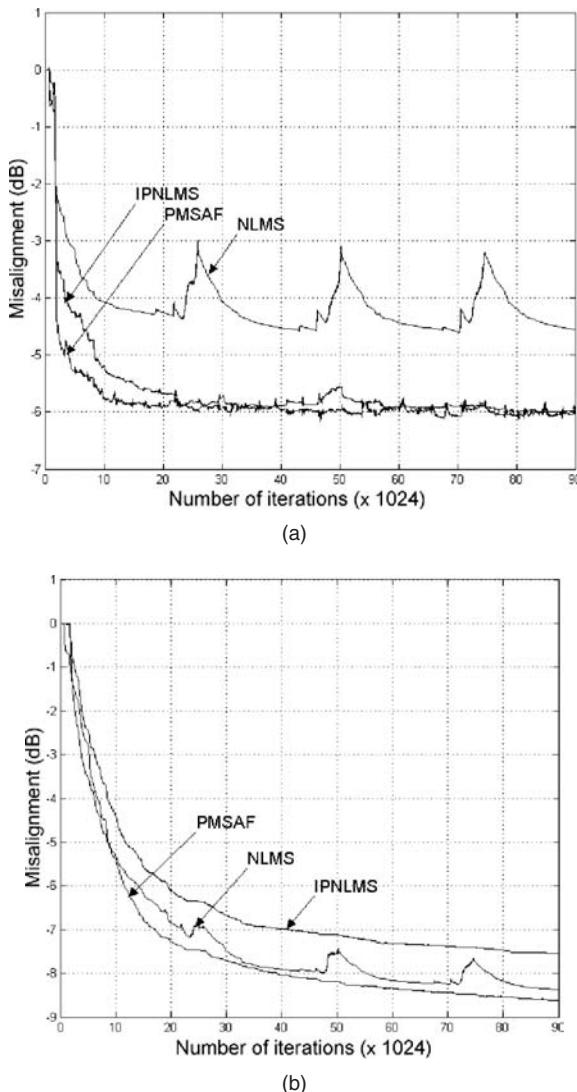


Figure 5.14 Misadjustment plots for the PMSAF, NLMS and IPNLMS algorithms using speech as input signals under different impulse responses: (a) sparse and (b) dispersive

References

- [1] J. D. Gordy and R. A. Goubran, ‘Fast system identification using affine projection and a critically sampled subband adaptive filter’, *IEEE Trans. Instrumentation and Measurement*, **55**(4), August 2006, 1242–1249.

- [2] R. G. Alves, J. A. Apolinario and M. R. Petraglia, ‘Subband adaptive filtering with critical sampling using the data selective affine projection’, in *Proc. ISCAS*, 2004, pp. 257–260.
- [3] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.
- [4] H. R. Abutalebi, H. Sheikhzadeh, R. L. Brennan and G. H. Freeman, ‘Affine projection algorithm for oversampled subband adaptive filters’, in *Proc. ICASSP*, 2003, pp. 209–212.
- [5] E. Chau, H. Sheikhzadeh and R. L. Brennan, ‘Complexity reduction and regularization of a fast affine projection algorithm for oversampled subband adaptive filters’, in *Proc. ICASSP*, 2004, pp. 109–112.
- [6] S. L. Gay and S. Tavathia, ‘The fast affine projection algorithm’, in *Proc. ICASSP*, 1995, pp. 3023–3026.
- [7] S. Haykin, *Adaptive Filter Theory*, 4th edition, Upper Saddle River, New Jersey: Prentice Hall, 2002.
- [8] T. Aboulnasr and K. Mayyas, ‘A robust variable step-size LMS-type algorithm: analysis and simulations’, *IEEE Trans. Signal Processing*, **45**(3), March 1997, 631–639.
- [9] A. Benveniste, M. Metivier and P. Priouret, *Adaptive Algorithms and Stochastic Approximation*, New York: Springer-Verlag, 1990.
- [10] R. H. Kwong and E. W. Johnston, ‘A variable step-size LMS algorithm’, *IEEE Trans. Signal Processing*, **40**(7), July 1992, 1633–1642.
- [11] V. J. Mathews and Z. Xie, ‘A stochastic gradient adaptive filter with gradient adaptive step-size’, *IEEE Trans. Signal Processing*, **41**(6), June 1993, 2075–2087.
- [12] T. Liu and S. Gazor, ‘A variable step-size pre-filter-bank adaptive algorithm’, *IEEE Trans. Speech and Audio Processing*, **13**(5), September 2005, 905–916.
- [13] T. Aboulnasr and K. Mayyas, ‘Selective coefficient update of gradient based adaptive algorithm’, in *Proc. ICASSP*, 1997, pp. 1929–1932.
- [14] K. Dogancay and O. Tanrikulu, ‘Adaptive filtering algorithms with selective partial updates’, *IEEE Trans. Circuits and Systems – II*, **48**(8), August 2001, 762–769.
- [15] S. C. Douglas, ‘Adaptive filters employing partial updates’, *IEEE Trans. Circuits and Systems – II*, **44**(3), March 1997, 209–216.
- [16] S. Attallah and S. W. Liaw, ‘Analysis of DCTLMS algorithm with a selective coefficient updating’, *IEEE Trans. Circuits and Systems II*, **48**(6), June 2001, 628–632.
- [17] S. Attallah, ‘The wavelet transform-domain LMS adaptive filter with partial subband-coefficient updating’, *IEEE Trans. Circuits and Systems – II*, **53**(1), January 2006, 8–12.
- [18] M. Harteneck, J. M. Paez-Borallo and R. W. Stewart, ‘An oversampled subband adaptive filter without cross adaptive filters’, *Signal Processing*, **64**(1), January 1998, 93–101.
- [19] P. L. de León and D. M. Etter, ‘Experimental results with increased bandwidth analysis filters in oversampled, subband acoustic echo cancelers’, *IEEE Signal Processing Letters*, **2**(1), January 1995, 1–3.
- [20] M. Harteneck, S. Weiss and R. W. Stewart, ‘Design of near perfect reconstruction oversampled filter banks for subband adaptive filters’, *IEEE Trans. Circuits and Systems – II*, **46**(8), August 1999, 1081–1085.
- [21] A. J. Coulson, ‘A generalization of nonuniform bandpass sampling’, *IEEE Trans. Signal Processing*, **43**(3), March 1995, 694–704.
- [22] J. D. Griesbach, M. Lightner and D. M. Etter, ‘Subband adaptive filtering decimation constraints for ovesampled nonuniform filter banks’, *IEEE Trans. Circuits and Systems – II*, **49**(10), October 2002, 677–681.

- [23] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Englewood Cliffs, New Jersey: Prentice Hall, 1983.
- [24] K. F. C. Yiu, N. Grbic, S. Nordholm and K. L. Teo, ‘Multicriteria design of oversampled uniform DFT filter banks’, *IEEE Signal Processing Lett.*, **11**(6), June 2004, 541–544.
- [25] M. Bosi, *Introduction to Digital Audio Coding and Standards*, Norwell, Massachusetts: Kluwer Academic Publishers, 2003.
- [26] ISO/IEC 11172-3, Information Technology, *Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s, Part 3: Audio*, 1993.
- [27] M. R. Petraglia and P. B. Batalheiro, ‘Filter bank design for a subband adaptive filtering structure with critical sampling’, *IEEE Trans. Circuit and Systems – I*, **51**(6), June 2004, 1194–1202.
- [28] D. L. Duttweiler, ‘Proportionate normalized least mean square adaptation in echo cancellers,’ *IEEE Trans. Speech Audio Processing*, **8**(5), September 2000, 508–518.
- [29] J. Benesty and S. L. Gay, ‘An improved PNLMS algorithm’, in *Proc. ICASSP*, 2002, pp. 1881–1884.
- [30] K. A. Lee and W. S. Gan, ‘Improving convergence of the NLMS algorithm using constrained subband updates’, *IEEE Signal Processing Lett.*, **11**(9), September 2004, 736–739.
- [31] X. Y. See, K. A. Lee and W. S. Gan, ‘Proportionate subband adaptive filtering’, in *Proc. ICALIP*, 2008, pp. 128–132.

6

Multiband-structured subband adaptive filters

Subband and multirate techniques [1–7] have been employed in designing computationally efficient adaptive filters with improved convergence performance for input signals having a large spectral dynamic range. One important application is acoustic echo cancellation [8–15], which involves colored (speech) signals and requires high-order filters to model long impulse responses of unknown and time-varying room acoustic paths. In the conventional subband adaptive filter (SAF) introduced in Chapter 4, each subband uses an individual adaptive subfilter in its own adaptation loop. Section 4.3 shows that the convergence rate of the SAF is decreased by aliasing and band-edge effects. To solve these structural problems, a multiband weight-control mechanism has been developed [16–22], in which the fullband adaptive filter is updated by input subband signals normalized by their respective variances.

Based on the multiband structure, a recursive weight-control mechanism called the multiband-structured SAF (MSAF) is derived from a deterministic optimization criterion. This chapter introduces and analyzes the MSAF algorithm to demonstrate its capability of dealing with colored input signals. In particular, we show that the MSAF algorithm is equivalent to the general form of the NLMS algorithm presented in Section 1.4.

6.1 Multiband structure

Figure 6.1 shows an example of a multiband structure for adaptive system identification, where the desired response $d(n)$ and filter output $y(n)$ are partitioned into N subbands by means of analysis filters $H_0(z), H_1(z), \dots, H_{N-1}(z)$. The subband signals, $d_i(n)$ and $y_i(n)$ for $i = 0, 1, \dots, N - 1$, are critically decimated to a lower rate commensurate with their reduced bandwidth. The sampling rate of the critically decimated signal is $1/N$ of the original one, thereby preserving the effective sampling rate and the total number of data samples to be processed.

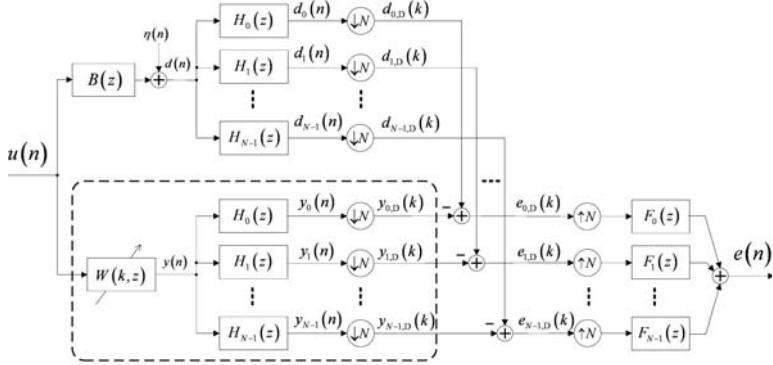


Figure 6.1 Multiband structure showing the subband desired responses $d_{i,D}(k)$, filter outputs $y_{i,D}(k)$ and estimation errors $e_{i,D}(k)$, where the fullband adaptive filter $W(k, z)$ is adapted by subband signals at the decimated rate (adapted from Reference [18])

Figure 6.2 shows two equivalent structures for the selected input portion of the multiband system as indicated by a dotted box in Figure 6.1. Assuming that the fullband adaptive filter $W(k, z)$ of length M is stationary (i.e. the adaptation is frozen), it can be transposed with the analysis filter bank as depicted in Figure 6.2(a). Each subband signal occupies only a portion of the original frequency range. The N subband signals

$$y_i(n) = \sum_{m=0}^{M-1} w_m(k) u_i(n-m), \text{ for } i = 0, 1, \dots, N-1, \quad (6.1)$$

are the outputs of the FIR filter, $W(k, z) = \sum_{m=0}^{M-1} w_m(k) z^{-m}$, in response to the input subband signals $u_i(n)$. The $N:1$ decimator retains only those samples of $y_i(n)$ that occur at time instants equal to multiples of N . Thus the decimated output subband signals can be written as

$$\begin{aligned} y_{i,D}(k) &= y_i(kN) \\ &= \sum_{m=0}^{M-1} w_m(k) u_i(kN - m) \\ &= \mathbf{w}^T(k) \mathbf{u}_i(k), \end{aligned} \quad (6.2)$$

where $\mathbf{w}(k) = [w_0(k), w_1(k), \dots, w_{M-1}(k)]^T$ is the weight vector of the fullband adaptive filter $W(k, z)$ and

$$\begin{aligned} \mathbf{u}_i(k) &= [u_i(kN), u_i(kN - 1), \dots, u_i(kN - N + 1), \\ &\quad u_i(kN - N), \dots, u_i(kN - M + 1)]^T \end{aligned} \quad (6.3)$$

is the regression (signal) vector for the i th subband. Equations (6.2) and (6.3) indicate that at every time instant k , each subband signal vector $\mathbf{u}_i(k)$ is packed with N new samples

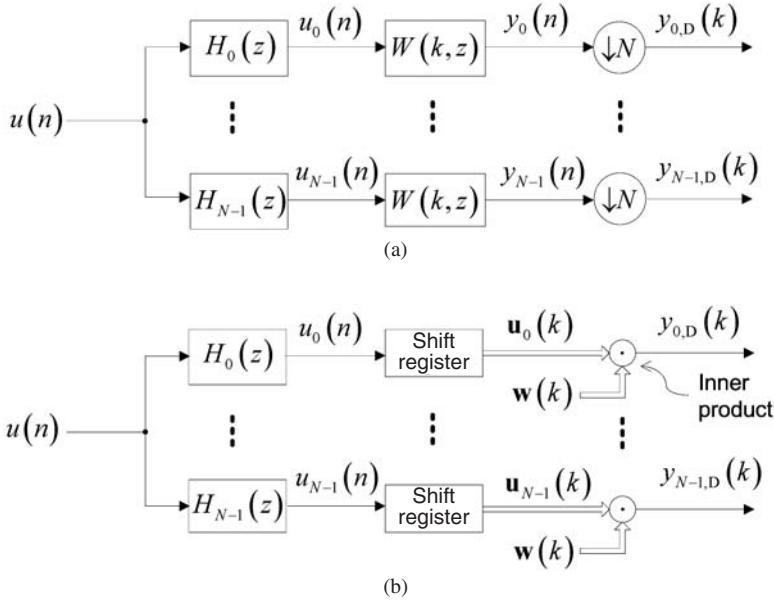


Figure 6.2 Two equivalent subband structures: (a) the input signal is divided into subband signals before the filter $W(k, z)$ and decimators (adapted from Reference [18]) and (b) the sample-based filtering process followed by decimation is equivalent to the block-based filtering process at the decimated rate

(from $u_i(kN)$ to $u_i(kN - N + 1)$) and $M - N$ old samples to produce an output sample $y_{i,D}(k)$. The block-based operation is illustrated in Figure 6.2(b), where the shift register performs both decimation and packing operations to form the subband vectors defined in Equation (6.3).

By employing the subband structure of Figure 6.2(a) in Figure 6.1, we obtain the subband structure shown in Figure 6.3. The subband error signal $e_{i,D}(k)$ is the difference between the filter output $y_{i,D}(k)$ and the desired response $d_{i,D}(k)$, expressed as

$$e_{i,D}(k) = d_{i,D}(k) - \mathbf{w}^T(k)\mathbf{u}_i(k) \text{ for } i = 0, 1, \dots, N - 1. \quad (6.4)$$

The error signal vector for N subbands, $\mathbf{e}_D(k) \equiv [e_{0,D}(k), e_{1,D}(k), \dots, e_{N-1,D}(k)]^T$, can be expressed in a compact form as

$$\mathbf{e}_D(k) = \mathbf{d}_D(k) - \mathbf{U}^T(k)\mathbf{w}(k), \quad (6.5)$$

where the $M \times N$ subband signal matrix $\mathbf{U}(k)$ and the $N \times 1$ desired response vector $\mathbf{d}_D(k)$ are defined as follows:

$$\mathbf{U}(k) \equiv [\mathbf{u}_0(k), \mathbf{u}_1(k), \dots, \mathbf{u}_{N-1}(k)] \text{ and} \quad (6.6)$$

$$\mathbf{d}_D(k) \equiv [d_{0,D}(k), d_{1,D}(k), \dots, d_{N-1,D}(k)]^T. \quad (6.7)$$

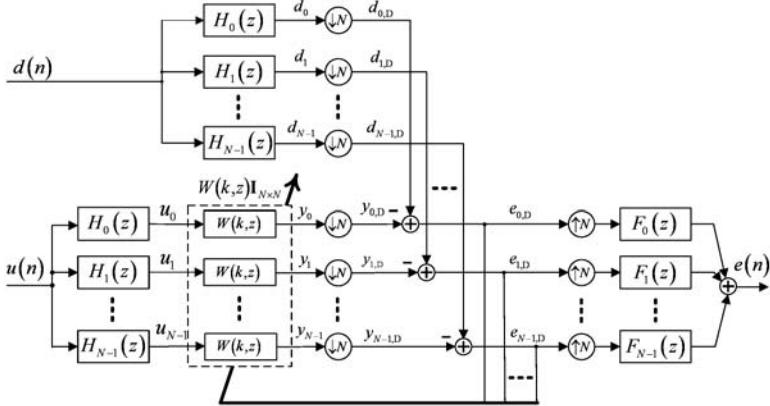


Figure 6.3 Multiband-structured SAF. Subband input and error signals are used to adapt the tap weights of the fullband filter $W(k, z)$ at the decimated rate (adapted from Reference [19])

In Figure 6.3, the diagonal N -input N -output system $W(k, z)\mathbf{I}_{N \times N}$ (where $\mathbf{I}_{N \times N}$ is the $N \times N$ identity matrix) represents a bank of parallel filters with an identical transfer function $W(k, z)$, as shown in the dotted box. These N adaptive filters are placed before the decimators (and after the analysis filter bank), which is different from the conventional SAF structure (see Figure 4.1) where a set of adaptive subfilters are placed after the decimators. In this new configuration, the fullband adaptive filter $W(k, z)$ operates on the set of bandlimited signals $u_i(n)$ at the original sampling rate; the tap weights, $\{w_m(k)\}_{m=0}^{M-1}$, are iteratively updated by subband data sets $\{\mathbf{U}(k), \mathbf{e}_D(k)\}$ at the decimated rate.

The tap weights $\{w_m(k)\}_{m=0}^{M-1}$ are considered as fullband filter coefficients because the subband signals $u_i(n)$ actuate the weight adaptation collectively to cover the original bandwidth. The fullband nature of the adaptive filter $W(k, z)$ is also clearly shown in Figure 6.1. By decimating the adaptation rate, the adaptive filter $W(k, z)$ performs filtering of N subband signals, $u_i(n)$ for $i = 0, 1, \dots, N - 1$, at $1/N$ th of the original sampling rate, as indicated by Equation (6.2). Therefore, the computational complexity remains the same even though there are N replicas of $W(k, z)$ in the MSAF shown in Figure 6.3, as opposed to a single $W(k, z)$ used in Figure 6.1.

6.1.1 Polyphase implementation

In Figure 6.3, the diagonal system $W(k, z)\mathbf{I}_{N \times N}$ is a bank of identical filters $W(k, z)$. The FIR filter outputs (reproduced from Equation (6.2))

$$y_{i,D}(k) = y_i(kN) = \mathbf{w}^T(k)\mathbf{u}_i(k), \text{ for } i = 0, 1, \dots, N - 1 \quad (6.8)$$

are computed once for every N input samples since the tap weights are updated at the critically decimated rate. The realization of an FIR filter with critical decimation is illustrated in Figure 6.4. Notice that there are M decimators embedded within the FIR

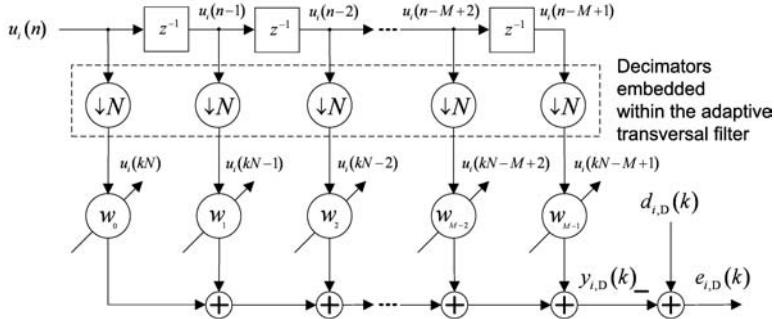


Figure 6.4 Direct-form realization of an adaptive FIR filter with a decimated output

filter shown in Figure 1.3. For every N input samples, the tapped-delay line and the decimators gather a block of M samples for the subband input vector

$$\mathbf{u}_i(k) = [u_i(kN), u_i(kN - 1), \dots, u_i(kN - N + 1), u_i(kN - N), \dots, u_i(kN - M + 1)]^T, \quad (6.9)$$

which consists of N new samples and $M - N$ old samples for the adaptive filter $\mathbf{w}(k)$ of length M .

An alternative realization of $W(k, z)$, based on polyphase decomposition, was proposed in Reference [22]. In this structure, the FIR filter $W(k, z)$ is represented in polyphase form as

$$W(k, z) = \sum_{r=0}^{N-1} W_r(k, z^N) z^{-r}, \quad (6.10)$$

where $W_r(k, z)$ is the r th polyphase filter (see Section 2.2.3) of $W(k, z)$, expressed as

$$W_r(k, z) = \sum_{n=0}^{K-1} w_{nN+r}(k) z^{-n}, \text{ for } r = 0, 1, \dots, N-1. \quad (6.11)$$

In Equations (6.10) and (6.11), K is the length of polyphase filters and the fullband adaptive FIR filter $W(k, z)$ is assumed to be of length $M = NK$ for simplicity. The polyphase realization is depicted in Figure 6.5.

The structures shown in Figures 6.4 and 6.5 are equivalent. Both realizations are computationally efficient when all the computations are performed at the decimated rate. The advantage of the direct-form realization shown in Figure 6.4 over the polyphase realization given in Figure 6.5 is that it allows a more compact representation of filtering operations using the matrix notation defined in Equation (6.8). Example 6.1 uses MATLAB implementations to illustrate both methods.

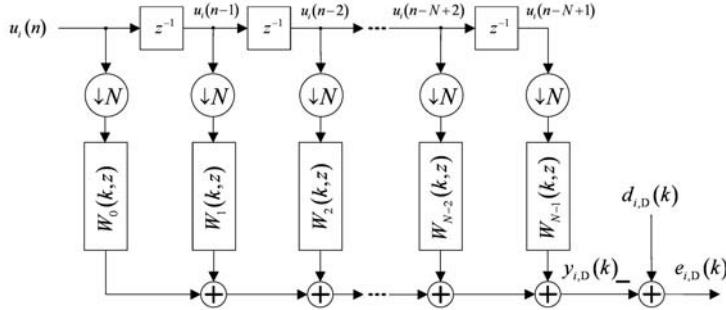


Figure 6.5 Polyphase realization of an adaptive FIR filter with a decimated output

Example 6.1 Both the direct-form and polyphase realizations shown in Figures 6.4 and 6.5 are implemented in MATLAB. In this example, the subband signal u_i is a sinusoidal signal and the adaptive FIR filter is assumed to be stationary (i.e. adaptation is frozen). The variables x and xN represent the tapped-delay lines of the direct-form and polyphase realizations, respectively. These tapped-delay lines are updated for every input sample and used to compute the outputs $y_{i,D1}$ and $y_{i,D2}$ at a decimated rate. At the end of the code, the output $y_{i,D3}$ is computed by filtering the signal u_i and decimation (as shown in Figure 6.2(a)). These realizations produce the same outputs. A complete listing of the code is given in M-file `Example_6P1.m`.

```

N = 8; % Number of subbands
M = 256; % Length of adaptive filter
w = fir1(M-1, 1/4); % Adaptive tap weights (assumed stationary)
x = zeros(M, 1); % Tapped-delay line buffer (direct form)
W = reshape(w, N, M/N); % Polyphase decomposition (in each row)
X = reshape(x, N, M/N); % Polyphase buffer
xN = zeros(N, 1); % Tapped-delay line buffer (polyphase form)

ITER = length(ui);
k = 0;
for n = 1:ITER
    x = [ui(n); x(1:end-1)]; % Update tapped-delay line
    xN = [ui(n); xN(1:end-1)]; % Update tapped-delay line
    if mod(n-1,N) == 0
        k = k + 1;
        yID1(k) = w*x; % Direct-form realization
        X = [xN, X(:,1:end-1)]; % Update buffer
        yID2(k) = sum(sum(W.*X)); % Polyphase realization
    end
end

% Filtering followed by decimation
yID3 = filter(w, 1, ui);
yID3 = yID3(1:N:end);

```

6.2 Multiband adaptation

This section derives a recursive weight-control mechanism for the MSAF based on the unconstrained optimization using the method of Lagrange multipliers [23, 24] and analyzes its computational complexity.

6.2.1 Principle of minimal disturbance

The principle of minimal disturbance [23, p. 321, 25] states that ‘from one iteration to the next, coefficients of adaptive filter should be changed in a minimal manner, subject to a (set of) constraint(s) imposed on the updated filter output.’ This principle is used to formulate a constraint optimization criterion for deriving the NLMS algorithm [23].

In the MSAF shown in Figure 6.3 with the subband signals sets $\{\mathbf{U}(k), \mathbf{d}_D(k)\}$, a criterion that ensures convergence to the optimum solution after a sufficient number of iterations is to have the updated weight vector $\mathbf{w}(k+1)$ nulling the a posteriori errors in all N subbands at each iteration k as follows:

$$\begin{aligned}\xi_{0,D}(k) &= d_{0,D}(k) - \mathbf{w}^T(k+1)\mathbf{u}_0(k) = 0, \\ \xi_{1,D}(k) &= d_{1,D}(k) - \mathbf{w}^T(k+1)\mathbf{u}_1(k) = 0, \\ &\vdots \\ \xi_{N-1,D}(k) &= d_{N-1,D}(k) - \mathbf{w}^T(k+1)\mathbf{u}_{N-1}(k) = 0.\end{aligned}\tag{6.12}$$

On the other hand, by adjusting the tap weights in a minimal fashion, the effects of noise $\eta(n)$ on the weight adaptation can be minimized as the adaptive filter converges to the optimum weight vector \mathbf{w}_o to identify the unknown system $B(z)$ shown in Figure 6.1. The term $\eta(n)$ includes measurement noise, system dynamics that cannot be modeled and other kinds of noise within the system.

6.2.2 Constrained subband updates

Based on the principle of minimal disturbance, we formulate the following constraint optimization problem for the multiband structure of Figure 6.3:

Minimize the squared Euclidean norm of the change in the weight vector

$$f[\mathbf{w}(k+1)] = \|\mathbf{w}(k+1) - \mathbf{w}(k)\|^2\tag{6.13}$$

subject to the set of N subband constraints imposed on the filter outputs as

$$d_{i,D}(k) = \mathbf{w}^T(k+1)\mathbf{u}_i(k) \text{ for } i = 0, 1, \dots, N-1.\tag{6.14}$$

This constrained optimization problem can be converted to unconstrained optimization by the method of Lagrange multipliers [23, 24]. Specifically, the quadratic function of Equation (6.13) is combined with the multiple constraints in Equation (6.14) to form the Lagrangian function

$$J[\mathbf{w}(k+1), \lambda'_0, \dots, \lambda'_{N-1}] = f[\mathbf{w}(k+1)] + \sum_{i=0}^{N-1} \lambda'_i [d_{i,D}(k) - \mathbf{w}^T(k+1)\mathbf{u}_i(k)],\tag{6.15}$$

where λ'_i are the Lagrange multipliers pertaining to multiple constraints.

Taking the derivative of the Lagrangian function with respect to the weight vector $\mathbf{w}(k+1)$ and setting this derivative to zero, we get

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \frac{1}{2} \sum_{i=0}^{N-1} \lambda'_i \mathbf{u}_i(k). \quad (6.16)$$

Using Equations (6.16) and (6.4) in Equation (6.14) as follows:

$$\begin{aligned} d_{i,D}(k) - \left[\mathbf{w}(k) + \frac{1}{2} \sum_{j=0}^{N-1} \lambda'_j \mathbf{u}_j(k) \right]^T \mathbf{u}_i(k) &= 0, \\ 2 \left[d_{i,D}(k) - \mathbf{w}^T(k) \mathbf{u}_i(k) \right] - \mathbf{u}_i^T(k) \sum_{j=0}^{N-1} \lambda'_j \mathbf{u}_j(k) &= 0, \\ 2e_{i,D}(k) - \mathbf{u}_i^T(k) \sum_{j=0}^{N-1} \lambda'_j \mathbf{u}_j(k) &= 0, \end{aligned}$$

we obtain a linear system of N equations expressed as

$$\mathbf{u}_i^T(k) \sum_{j=0}^{N-1} \lambda'_j \mathbf{u}_j(k) = 2e_{i,D}(k) \text{ for } i = 0, 1, \dots, N-1. \quad (6.17)$$

Using the notation defined in Equations (6.5) and (6.6), these N linear equations in Equation (6.17) can be formulated into the following matrix forms:

$$\begin{bmatrix} \mathbf{u}_0^T(k) \mathbf{u}_0(k) & \cdots & \mathbf{u}_0^T(k) \mathbf{u}_{N-1}(k) \\ \vdots & \ddots & \vdots \\ \mathbf{u}_{N-1}^T(k) \mathbf{u}_0(k) & \cdots & \mathbf{u}_{N-1}^T(k) \mathbf{u}_{N-1}(k) \end{bmatrix} \begin{bmatrix} \lambda'_0 \\ \vdots \\ \lambda'_{N-1} \end{bmatrix} = 2 \begin{bmatrix} e_{0,D}(k) \\ \vdots \\ e_{N-1,D}(k) \end{bmatrix}, \quad (6.18)$$

$$[\mathbf{U}^T(k) \mathbf{U}(k)] \boldsymbol{\lambda} = 2\mathbf{e}_D(k),$$

where $\boldsymbol{\lambda} = [\lambda'_0, \lambda'_1, \dots, \lambda'_{N-1}]^T$ is the $N \times 1$ Lagrange vector. Solving the linear system of the Lagrange multipliers, we obtain

$$\boldsymbol{\lambda} = 2 [\mathbf{U}^T(k) \mathbf{U}(k)]^{-1} \mathbf{e}_D(k). \quad (6.19)$$

Assuming that subband signals are orthogonal at zero lag, the off-diagonal elements of the matrix $\mathbf{U}^T(k) \mathbf{U}(k)$ are negligible (this will be further discussed in Section 6.3). With this diagonal assumption, Equation (6.19) can be reduced to a simple form

$$\lambda'_i = 2 \frac{e_{i,D}(k)}{\|\mathbf{u}_i(k)\|^2} \text{ for } i = 0, 1, \dots, N-1. \quad (6.20)$$

Combining the results given in Equations (6.16) and (6.20), a recursive equation for updating the weight vector can be obtained as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \sum_{i=0}^{N-1} \frac{\mathbf{u}_i(k)}{\|\mathbf{u}_i(k)\|^2 + \alpha} e_{i,D}(k), \quad (6.21)$$

where α is a small positive constant used to avoid possible division by zero. In the absence of disturbance, the necessary and sufficient condition for the convergence in the mean-squares sense is that the step size μ must satisfy the following double inequality [26]:

$$0 < \mu < 2. \quad (6.22)$$

The choice of step size will be discussed in Section 7.4.

Using the notation defined in Equations (6.5) and (6.6), the algorithm given in Equation (6.21) can be expressed more compactly as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \mathbf{U}(k) \boldsymbol{\Lambda}^{-1}(k) \mathbf{e}_D(k), \quad (6.23)$$

where

$$\boldsymbol{\Lambda}(k) = \text{diag}[\mathbf{U}^T(k) \mathbf{U}(k) + \alpha \mathbf{I}] \quad (6.24)$$

is the diagonal matrix of normalization factors $\lambda_i(k) = \|\mathbf{u}_i(k)\|^2 + \alpha$. This adaptive algorithm is called the multiband-structured SAF algorithm to highlight its multiband nature. The simplest interpretation of Equation (6.21) or (6.23) is that it is a method of making the fullband weight vector $\mathbf{w}(k)$ converge equally for all spectral components by normalizing the subband regressors $\mathbf{u}_i(k)$ with the respective variance estimates $\lambda_i(k) = \|\mathbf{u}_i(k)\|^2 = M\hat{\gamma}_{ii}(0)$, where $\hat{\gamma}_{ii}(0) = \mathbf{u}_i^T(k) \mathbf{u}_i(k)/M$ is the running-average estimate of the subband variance $\gamma_{ii}(0) = E\{u_i^2(n)\}$.

The above optimization problem involves N constraints. The number of constraints N (i.e. the number of subbands) must be smaller than the number of unknowns M (i.e. the length of fullband weight vector) for the minimum norm solution to exist. This requirement sets an upper bound on the number of subbands allowed in the MSAF algorithm. In the adaptive echo cancellation applications, the upper bound is not particularly limiting because the filter length M is usually a large number.

6.2.3 Computational complexity

The MSAF algorithm is summarized in Table 6.1 along with the associated computational cost in terms of number of multiplications per sampling period T_s . Since the adaptation process is performed at the decimated rate of $1/NT_s$, the number of multiplications needed is divided by N . The MSAF algorithm requires $3M + 2$ multiplications for the adaptation process and $(N + 2)L$ multiplications for the analysis and synthesis filter banks; thus the total computational cost is $O(M) + O(NL)$. In Table 6.1, the analysis and synthesis filter banks are represented as \mathbf{H} and \mathbf{F} , respectively, and they are implemented as block transforms with memory. The columns of the matrices, $\mathbf{h}_i = [h_i(0), h_i(1), \dots, h_i(L-1)]^T$ and

Table 6.1 Summary of the MSAF algorithm

Computation	Multiplications per T_s
For $k = 0, 1, 2, \dots$, where $kN = n$,	
Error estimation: $\mathbf{e}_D(k) = \mathbf{d}_D(k) - \mathbf{U}^T(k)\mathbf{w}(k)$	$\frac{N \times M}{N} = M$
Normalization matrix: $\mathbf{A}(k) = \text{diag}[\mathbf{U}^T(k)\mathbf{U}(k) + \alpha\mathbf{I}]$	$\frac{N \times M}{N} = M$
Tap-weight adaptation: $\mathbf{w}(k+1) = \mathbf{w}(k) + \mu\mathbf{U}(k)\mathbf{A}^{-1}(k)\mathbf{e}_D(k)$	$\frac{2N + NM}{N} = M + 2$
Band partitioning: $\mathbf{U}_1^T(k) = \mathbf{H}^T\mathbf{A}(kN)$ $\mathbf{d}_D(k) = \mathbf{H}^T\mathbf{d}(kN)$	$\frac{N^2L + NL}{N} = (N+1)L$
Synthesis: $\mathbf{e}(kN) = \mathbf{F}\mathbf{e}_D(k)$	$\frac{N \times L}{N} = L$
<i>Parameters:</i>	
M - number of adaptive tap weights	
N - number of subbands	
L - length of the analysis and synthesis filters	
<i>Variables:</i>	
$\mathbf{U}^T(k) = [\mathbf{U}_1^T(k), \mathbf{U}_2^T(k-1)]$	
$\mathbf{U}_2^T(k-1)$ - first $M-N$ columns of $\mathbf{U}^T(k-1)$	
$\mathbf{A}(kN) = [\mathbf{a}(kN), \mathbf{a}(kN-1), \dots, \mathbf{a}(kN-N+1)]$	
$\mathbf{a}(kN) = [u(kN), u(kN-1), \dots, u(kN-L+1)]^T$	
$\mathbf{d}(kN) = [d(kN), d(kN-1), \dots, d(kN-L+1)]^T$	
$\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{N-1}]$ - analysis filter bank	
$\mathbf{F} = [\mathbf{f}_0, \mathbf{f}_1, \dots, \mathbf{f}_{N-1}]$ - synthesis filter bank	

$\mathbf{f}_i = [f_i(0), f_i(1), \dots, f_i(L-1)]^T$, represent the analysis and synthesis filters, respectively. A direct implementation of the filter banks is possible but would require $3NL$ multiplications [1]. MATLAB implementation of the MSAF algorithm is outlined in Example 6.2.

The weight adaptation equation (6.21) is comparable to the NLMS algorithm defined in Equation (1.13). In the MSAF algorithm, the fullband weight vector $\mathbf{w}(k)$ is adapted by N normalized subband regressors $\mathbf{u}_i(k)$ instead of a single fullband regressor $\mathbf{u}(n)$ in the NLMS algorithm. By virtue of critical decimation, the weight vector is adapted at the decimated rate. Thus the number of multiplications required for the weight adaptation is $3M + 2$ for an arbitrary number of subbands N . Compared with the NLMS algorithm, the MSAF algorithm requires additional $(N+2)L$ multiplications for the implementation of filter banks. In acoustic echo cancellation applications, M is typically much larger than NL . Clearly, the MSAF algorithm exploits the properties of subband signals to accelerate the convergence, while remaining computationally efficient by means of a multirate structure.

Example 6.2 The MSAF algorithm can be implemented in MATLAB using matrix notation as summarized in Table 6.1. In the following lines of code, the fullband weight vector $\mathbf{w}(k)$ is iteratively updated using a set of subband quantities derived from the input signal $u(n)$ and the desired response $d(n)$. Before the adaptation process is started, we initialize all variables to zero values as follows:

(i) For adaptive filter

```
w = zeros(M,1); % Adaptive weight vector
U = zeros(M,N); % Subband signal matrix
```

(ii) For analysis filter banks

```
d = zeros(L,1); % Tapped-delay line for d(n)
a = zeros(L,1); % Tapped-delay line for u(n)
A = zeros(L,N); % Buffer for U1
```

(iii) For synthesis filter bank

```
z = zeros(L,1); % Tapped-delay line for e(n)
```

For the given input signal u_n and desired response d_n , the weight vector w is iteratively updated as (see Table 6.1):

```
for n = 1:ITER
    d = [dn(n); d(1:end-1)]; % Update tapped-delay line of d(n)
    a = [un(n); a(1:end-1)]; % Update tapped-delay line of u(n)
    A = [a, A(:,1:end-1)]; % Update buffer
    if (mod(n,N)==0)
        U1 = (H'*A)';
        U2 = U(1:end-N,:);
        U = [U1', U2'];
        dD = dD - U'*w; % Error estimation
        eD = dD - U*(eD./(sum(U.*U)+alpha)')*mu;
        w = w + U*(eD./(sum(U.*U)+alpha)')*mu;
        z = F*eD + z; % Synthesis
        en(n-N+1:n) = z(1:N);
        z = [z(N+1:end); zeros(N,1)];
    end
end
```

6.3 Underdetermined least-squares solutions

For the case where $\mu = 1$, $\alpha = 0$ (i.e. an unregularized MSAF algorithm with unity step size) and the diagonal assumption is valid, i.e. $\mathbf{U}^T(k)\mathbf{U}(k) \approx \Lambda(k)$, the weight adaptation equation (6.23) becomes

$$\delta\mathbf{w}(k+1) = \mathbf{U}(k)[\mathbf{U}^T(k)\mathbf{U}(k)]^{-1}\mathbf{e}_D(k), \quad (6.25)$$

where $\delta\mathbf{w}(k+1) = \mathbf{w}(k+1) - \mathbf{w}(k)$ is the weight-adjustment vector. Since $\mathbf{U}(k)[\mathbf{U}^T(k)\mathbf{U}(k)]^{-1}$ is the pseudo-inverse of $\mathbf{U}^T(k)$, the weight-adjustment vector

$\delta\mathbf{w}(k+1)$ can be regarded as the minimum-norm solution for the linear system of N equations with M unknowns, expressed as

$$\mathbf{U}^T(k)\delta\mathbf{w}(k+1) = \mathbf{e}_D(k). \quad (6.26)$$

Using Equation (6.5) in Equation (6.26) leads to

$$\mathbf{U}^T(k)\mathbf{w}(k+1) = \mathbf{d}_D(k). \quad (6.27)$$

Hence, the MSAF algorithm can be seen from the deterministic perspective as a recursive estimator that updates the weight vector at each iteration in a minimal manner (i.e. the squared Euclidean norm of $\delta\mathbf{w}(k+1)$ is minimized) while nulling the a posteriori error, $\xi(k) = \mathbf{d}_D(k) - \mathbf{U}^T(k)\mathbf{w}(k+1)$, in all N subbands.

Equations (6.25) and (6.26) define an underdetermined case of the least-squares estimation problem where the number of parameters M is more than the number of constraints N . It has been shown that the NLMS and AP algorithms can also be derived as underdetermined least-squares solutions listed in Table 6.2 [23, 27, 28]. Compare the minimum-norm solutions given in Equation (6.25) and Table 6.2. Both the AP and MSAF algorithms can be seen as generalized forms of the NLMS algorithm. The AP algorithm generalizes the NLMS algorithm along the time axis using multiple time-domain constraints, whereas the MSAF algorithm generalizes the NLMS algorithm along the frequency axis using multiple subband constraints. These multiple constraints contribute to the decorrelation properties of the algorithms that accelerate convergence under colored input signals. Detailed discussions of decorrelation properties of the AP algorithm can be found in References [29] to [31].

6.3.1 NLMS equivalent

For a special case of $N = 1$, the MSAF algorithm given in Equation (6.21) is reduced to the NLMS algorithm, expressed as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \frac{\mathbf{u}(k)}{\|\mathbf{u}(k)\|^2 + \alpha} e(k). \quad (6.28)$$

Table 6.2 Deterministic interpretations of the NLMS and AP algorithms

Algorithm	Underdetermined least-squares solution	Complexity
NLMS	$\delta\mathbf{w}(n+1) = \mathbf{u}(n) [\mathbf{u}^T(n)\mathbf{u}(n)]^{-1} e(n),$ where $e(n) = d(n) - \mathbf{u}^T(n)\mathbf{w}(n)$ and $\mathbf{u}(n) \equiv [u(n), u(n-1), \dots, u(n-M+1)]^T.$	$O(M)$
AP	$\delta\mathbf{w}(n+1) = \mathbf{A}(n) [\mathbf{A}^T(n)\mathbf{A}(n)]^{-1} \mathbf{e}(n)$ where $\mathbf{A}(n) \equiv [\mathbf{u}(n), \mathbf{u}(n-1), \dots, \mathbf{u}(n-P+1)],$ $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{A}^T(n)\mathbf{w}(n)$, and $\mathbf{d}(n) \equiv [d(n), d(n-1), \dots, d(n-P+1)]^T.$	$O(P^2M)$

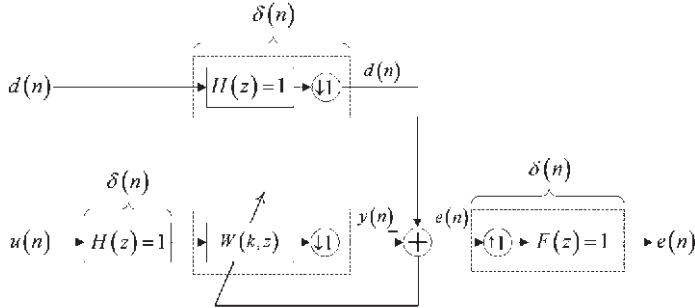


Figure 6.6 The NLMS algorithm can be regarded as the special case of the MSAF algorithm for $N = 1$

As illustrated in Figure 6.6, the filter bank is reduced to a single filter with the unit impulse $\delta(n)$ as the impulse response because the band partitioning and the subsequent signal reconstruction are no longer required. Furthermore, since $k = n$ for $N = 1$, the weight adaptation of Equation (6.28) is performed at the original sampling rate.

6.3.2 Projection interpretation

From a projection interpretation [32] viewpoint, the MSAF algorithm can be seen as a subband variant of the AP algorithm, where subband bases $\mathbf{u}_0(k), \dots, \mathbf{u}_{N-1}(k)$ are used to replace the time-domain bases $\mathbf{u}(n), \dots, \mathbf{u}(n - N + 1)$ in the iterative projection. The projection interpretation of the MSAF algorithm will be further discussed in Section 7.3. The major difference is the nature of bases used in the projection. The advantage of the subband bases is their orthogonal properties; i.e. the inner product $\mathbf{U}^T(k)\mathbf{U}(k)$ is diagonal, which greatly reduces the computational burden of computing its inverse $[\mathbf{U}^T(k)\mathbf{U}(k)]^{-1}$ for the weight adaptation. The use of subband bases is an effective solution to the matrix inversion problem encountered in the AP algorithm. It can also be regarded as a viable alternative to the fast implementation of the AP algorithm, which calculates the matrix inversion recursively with numerical instability problems, thus requiring proper regularization and re-initialization strategies.

6.4 Stochastic interpretations

In this section, the behavior of the MSAF algorithm is analyzed from two stochastic viewpoints: a diagonalized Newton method and an equalization of error signal. The MSAF algorithm can be shown to have fundamental decorrelation properties that whiten the input signal prior to weight adaptation.

6.4.1 Stochastic approximation to Newton's method

Newton's method is an iterative optimization process that can be cast for the quadratic case [23, 33–35] as

$$\mathbf{w}(l+1) = \mathbf{w}(l) + \mu [\mathbf{R} + \alpha' \mathbf{I}]^{-1} [\mathbf{p} - \mathbf{R}\mathbf{w}(l)], \quad (6.29)$$

where the autocorrelation matrix \mathbf{R} and the cross-correlation vector \mathbf{p} are signal statistics that characterize the mean-square error (MSE) function $J = E\{e^2(n)\}$ to be minimized. In Equation (6.29), the index l denotes the step of the iterative process, and $\alpha' = \alpha/N$ is the regularization parameter. The regularized Newton recursion given in Equation (6.29) is called the Levenberg–Marquardt method [28, 36]. To develop an adaptive algorithm for the MSAF given in Figure 6.3 from Equation (6.29), we replace \mathbf{R} and \mathbf{p} by their estimates in terms of $\{\mathbf{U}(k), \mathbf{d}_D(k)\}$ as follows:

$$\hat{\mathbf{R}}(k) = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{u}_i(k) \mathbf{u}_i^T(k) = \frac{1}{N} \mathbf{U}(k) \mathbf{U}^T(k) \quad (6.30)$$

and

$$\hat{\mathbf{p}}(k) = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{u}_i(k) d_{i,D}(k) = \frac{1}{N} \mathbf{U}(k) \mathbf{d}_D(k). \quad (6.31)$$

These estimates are unbiased if the filter bank is power complementary. Substituting Equations (6.30) and (6.31) into Equation (6.29) for \mathbf{R} and \mathbf{p} , and using Equation (6.5) in the expression, we obtain

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) + \mu \left[\frac{1}{N} \mathbf{U}(k) \mathbf{U}^T(k) + \alpha' \mathbf{I} \right]^{-1} \\ &\quad \times \left[\frac{1}{N} \mathbf{U}(k) \mathbf{d}_D(k) - \frac{1}{N} \mathbf{U}(k) \mathbf{U}^T(k) \mathbf{w}(k) \right] \\ &= \mathbf{w}(k) + \mu [\mathbf{U}(k) \mathbf{U}^T(k) + N\alpha' \mathbf{I}]^{-1} \mathbf{U}(k) [\mathbf{d}_D(k) - \mathbf{U}^T(k) \mathbf{w}(k)] \\ &= \mathbf{w}(k) + \mu [\mathbf{U}(k) \mathbf{U}^T(k) + \alpha \mathbf{I}]^{-1} \mathbf{U}(k) \mathbf{e}_D(k). \end{aligned} \quad (6.32)$$

Note that the iteration index l has been replaced by the decimated time index k because the weight recursion coincides with the time update of the estimates $\hat{\mathbf{R}}(k)$ and $\hat{\mathbf{p}}(k)$.

The recursion shown in Equation (6.32) requires the inversion of the $M \times M$ matrix $[\mathbf{U}(k) \mathbf{U}^T(k) + \alpha I]$ at each iteration, which can be simplified using the following matrix inversion lemma [23, 28]

$$[\mathbf{A} + \mathbf{BCD}]^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{B} [\mathbf{C}^{-1} + \mathbf{D} \mathbf{A}^{-1} \mathbf{B}]^{-1} \mathbf{D} \mathbf{A}^{-1}. \quad (6.33)$$

Let $\mathbf{A} = \alpha \mathbf{I}$, $\mathbf{B} = \mathbf{U}(k)$, $\mathbf{C} = \mathbf{I}$ and $\mathbf{D} = \mathbf{U}^T(k)$. The matrix inversion can be obtained as

$$\begin{aligned} [\mathbf{U}(k) \mathbf{U}^T(k) + \alpha \mathbf{I}]^{-1} &= \alpha^{-1} \mathbf{I} - \alpha^{-1} \mathbf{I} \mathbf{U}(k) [\mathbf{I} + \mathbf{U}^T(k) \alpha^{-1} \mathbf{I} \mathbf{U}(k)]^{-1} \mathbf{U}^T(k) \alpha^{-1} \mathbf{I} \\ &= \alpha^{-1} \mathbf{I} - \alpha^{-1} \mathbf{U}(k) [\alpha \mathbf{I} + \mathbf{U}^T(k) \mathbf{U}(k)]^{-1} \mathbf{U}^T(k). \end{aligned} \quad (6.34)$$

Multiplying both sides of Equation (6.34) by $\mathbf{U}(k)$, we have

$$\begin{aligned} &[\mathbf{U}(k) \mathbf{U}^T(k) + \alpha \mathbf{I}]^{-1} \mathbf{U}(k) \\ &= \alpha^{-1} \mathbf{U}(k) - \alpha^{-1} \mathbf{U}(k) [\alpha \mathbf{I} + \mathbf{U}^T(k) \mathbf{U}(k)]^{-1} \mathbf{U}^T(k) \mathbf{U}(k) \\ &= \alpha^{-1} \mathbf{U}(k) \{ \mathbf{I} - [\alpha \mathbf{I} + \mathbf{U}^T(k) \mathbf{U}(k)]^{-1} \mathbf{U}^T(k) \mathbf{U}(k) \} \end{aligned} \quad (6.35)$$

$$\begin{aligned}
&= \alpha^{-1} \mathbf{U}(k) [\alpha \mathbf{I} + \mathbf{U}^T(k) \mathbf{U}(k)]^{-1} [\alpha \mathbf{I} + \mathbf{U}^T(k) \mathbf{U}(k) - \mathbf{U}^T(k) \mathbf{U}(k)] \\
&= \mathbf{U}(k) [\mathbf{U}^T(k) \mathbf{U}(k) + \alpha \mathbf{I}]^{-1}.
\end{aligned}$$

Using this result in Equation (6.32) leads to a mathematically equivalent form of the recursive equation:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \mathbf{U}(k) [\mathbf{U}^T(k) \mathbf{U}(k) + \alpha \mathbf{I}]^{-1} \mathbf{e}_D(k). \quad (6.36)$$

The recursion given in Equation (6.36) is more efficient than Equation (6.32) because it involves the inversion of the lower dimension $N \times N$ matrix $[\mathbf{U}^T(k) \mathbf{U}(k) + \alpha \mathbf{I}]$, where $N < M$ for a high-order adaptive filter of length M . Furthermore, because of the diagonal assumption, the off-diagonal elements of the matrix $[\mathbf{U}^T(k) \mathbf{U}(k) + \alpha \mathbf{I}]$ can be assumed as negligible; thus the weight adaptation can be further simplified to

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \sum_{i=0}^{N-1} \mathbf{u}_i(k) [\mathbf{u}_i^T(k) \mathbf{u}_i(k) + \alpha]^{-1} e_{i,D}(k). \quad (6.37)$$

This equation is identical to the weight adaptation of the MSAF algorithm given in Equation (6.21). The difference is that the weight update recursion of Equation (6.37) is obtained from a stochastic approach as an approximation of Newton's method, as summarized in Figure 6.7(a).

It should be emphasized that a power complementary filter bank preserves the second-order statistics of the input signal $u(n)$ and desired response $d(n)$, thus resulting in the unbiased estimates $\hat{\mathbf{R}}(k)$ and $\hat{\mathbf{p}}(k)$ in Equations (6.30) and (6.31), respectively. The use of these unbiased estimates in the recursive estimation allows the algorithm to converge to the optimal Wiener solution, $\mathbf{w}_o = \mathbf{R}^{-1} \mathbf{p}$.

6.4.2 Weighted MSE criterion

An MSE function defined as the weighted sum of the mean-squared values of subband error signals [16, 17, 22] is expressed as

$$J_{SB} = \sum_{i=0}^{N-1} \lambda_i^{-1} E\{e_{i,D}^2(k)\}, \quad (6.38)$$

where λ_i^{-1} denotes the weighting factors (i.e. the reciprocal of the normalization factors λ_i defined earlier). The MSAF algorithm given in Equation (6.21) can be derived by minimizing the weighted MSE function (6.38) using the steepest-descent algorithm, then taking the instantaneous estimates of the ensemble averages that appear in the recursion and finally setting the weighting factors λ_i^{-1} to $[\|\mathbf{u}_i(k)\|^2 + \alpha]^{-1}$. However, in a method different from that presented in Section 6.1, this new approach formulates the MSAF algorithm as a stochastic approximation to the steepest-descent method, as summarized in Figure 6.7(b). The weighting factors λ_i^{-1} are proportional to the inverse of the variances of the respective input subband signals. Hence, the weighted MSE function gives a heavier weight to the error signal corresponding to the subband with lower signal power. This weighting scheme can be seen as a decorrelation (or equalization) procedure.

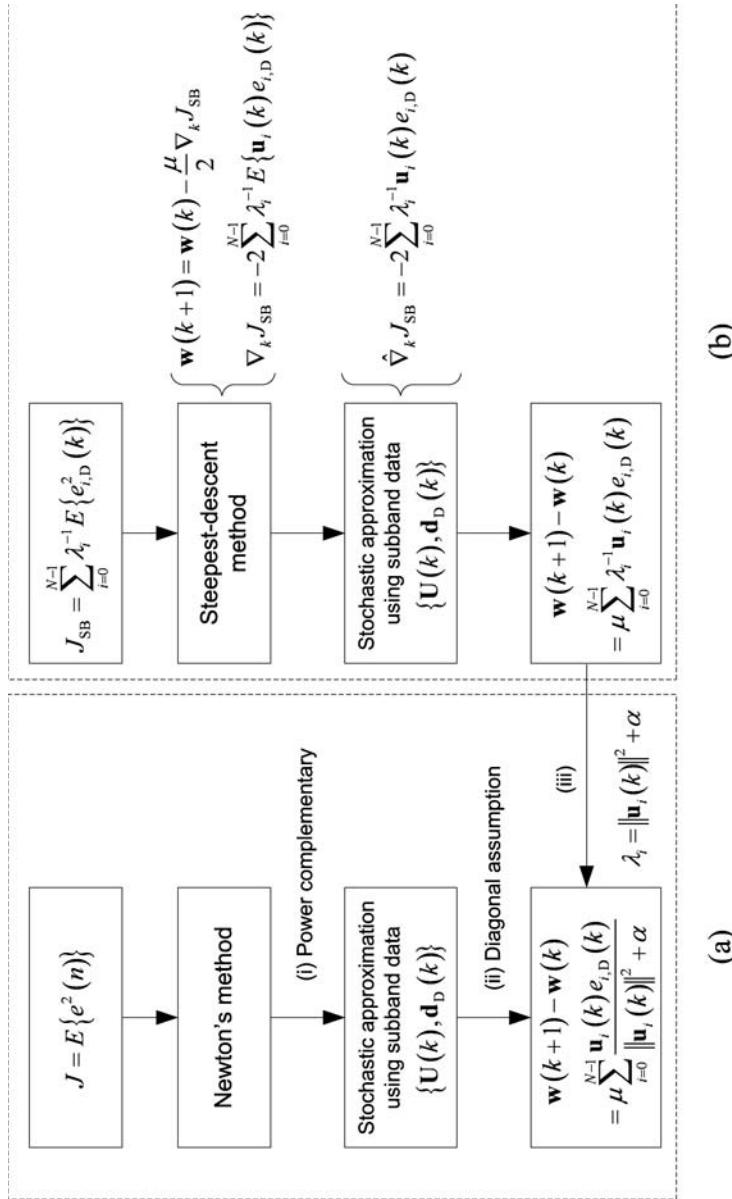


Figure 6.7 Stochastic interpretations of the MSAF algorithm (adapted from Reference [19]). Stochastic approximation to (a) Newton's method in minimizing the classical MSE function and (b) the steepest-descent algorithm in minimizing a weighted MSE function

Assume that all signals are wide-sense stationary and the adaptive filter $W(k, z)$ is time invariant (i.e. the weight adaptation is frozen). The estimation error $e_{i,D}(k)$ measures the difference between the desired response $d_{i,D}(k)$ and the filter output $y_{i,D}(k)$ at the decimated rate, as defined in Equation (6.4) and illustrated in Figure 6.8(a). Since the addition and decimation operations are commutative (as explained in Section 2.1), the estimation error can be alternatively obtained as $e_{i,D}(k) = e_i(kN)$, as shown in Figure 6.8(b), where

$$\begin{aligned} e_i(n) &= d_i(n) - \sum_{m=0}^{M-1} w_m u_i(n-m) \\ &= d_i(n) - w_n * u_i(n) \end{aligned} \quad (6.39)$$

is the subband error signal at the original sampling rate. Notice that the iteration index k has been dropped from the tap weights, $\{w_m(k)\}_{m=0}^{M-1}$, because we assume that the weight adaptation is frozen. The subband error signal $e_i(n)$ given in Equation (6.39) can be expressed in terms of the fullband error signal $e(n)$ by considering the analysis filter as follows:

$$\begin{aligned} e_i(n) &= h_i(n) * d(n) - w_n * [h_i(n) * u(n)] \\ &= h_i(n) * [d(n) - w_n * u(n)] \\ &= h_i(n) * e(n), \end{aligned} \quad (6.40)$$

where $h_i(n)$ is the impulse response of the i th analysis filter. Note that we have applied the associative, commutative and distributive laws of the linear time-invariant system in deriving Equation (6.40). This equation implies that the subband error signals $e_i(n)$ can be obtained by partitioning the fullband error signal $e(n)$, as illustrated in Figure 6.8(c).

Applying the correlation-domain formulation (as described in Section 3.1) to Equation (6.40), the autocorrelation function $g_i(l) \equiv E\{e_i(n)e_i(n-l)\}$ of the subband error signal $e_i(n)$ can be written as

$$g_i(l) = h_i(l) * h_i(-l) * g_e(l), \text{ for } i = 0, 1, \dots, N-1, \quad (6.41)$$

where $g_e(l) \equiv E\{e(n)e(n-l)\}$ is the autocorrelation function of the fullband estimation error $e(n)$. Next, define the weighted sum of subband autocorrelation functions $g_\xi(l)$ in the following form:

$$g_\xi(l) \equiv \sum_{i=0}^{N-1} \lambda_i^{-1} g_i(l) = \sum_{i=0}^{N-1} \lambda_i^{-1} E\{e_i(n)e_i(n-l)\}. \quad (6.42)$$

Substituting Equation (6.41) into Equation (6.42), we obtain

$$\begin{aligned} g_\xi(l) &= \sum_{i=0}^{N-1} \lambda_i^{-1} [h_i(l) * h_i(-l) * g_e(l)] \\ &= \left[\sum_{i=0}^{N-1} \lambda_i^{-1} h_i(l) * h_i(-l) \right] * g_e(l) \\ &= [h_\xi(l) * h_\xi(-l)] * g_e(l). \end{aligned} \quad (6.43)$$

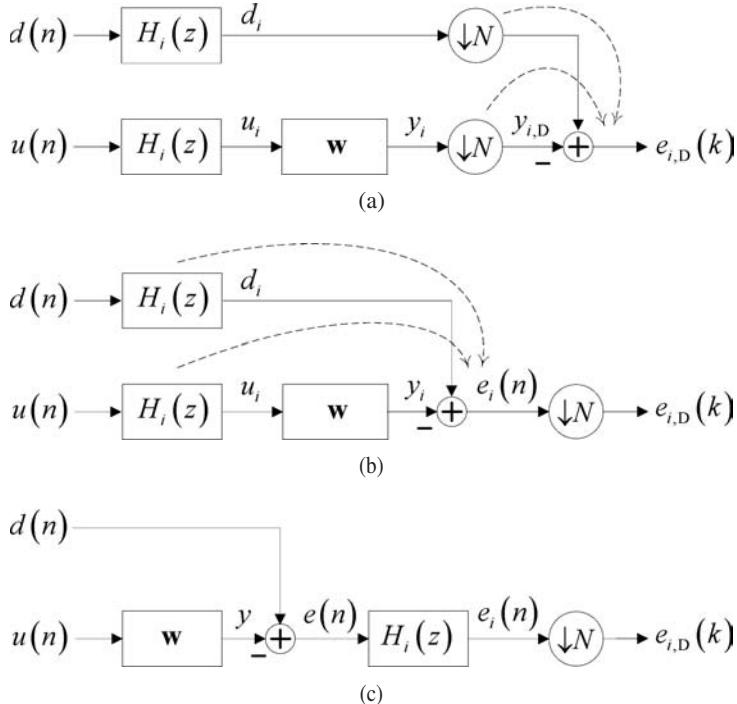


Figure 6.8 Analysis of the weighted MSE function J_{SB} by assuming that all signals and systems are stationary and time invariant

The resulting filter $h_{\xi}(n)$ can be regarded as an equalization filter with its autocorrelation function given by

$$h_{\xi}(n) * h_{\xi}(-n) = \sum_{i=0}^{N-1} \lambda_i^{-1} [h_i(n) * h_i(-n)]. \quad (6.44)$$

Taking the Fourier transform of Equation (6.44) results in the frequency-domain equation

$$|H_{\xi}(e^{j\omega})|^2 = \sum_{i=0}^{N-1} \lambda_i^{-1} |H_i(e^{j\omega})|^2, \quad (6.45)$$

where $|H_i(e^{j\omega})|$ is the magnitude response of the i th analysis filter.

As shown in Figure 6.9(a), the filter $h_{\xi}(n)$ operates on the fullband error signal $e(n)$ to generate the preconditioned error signal $\xi(n) = h_{\xi}(n) * e(n)$. The autocorrelation function of $\xi(n)$ can be written as

$$\begin{aligned} E\{\xi(n)\xi(n-l)\} &= [h_{\xi}(l) * h_{\xi}(-l)] * g_e(l) \\ &= g_{\xi}(l). \end{aligned} \quad (6.46)$$

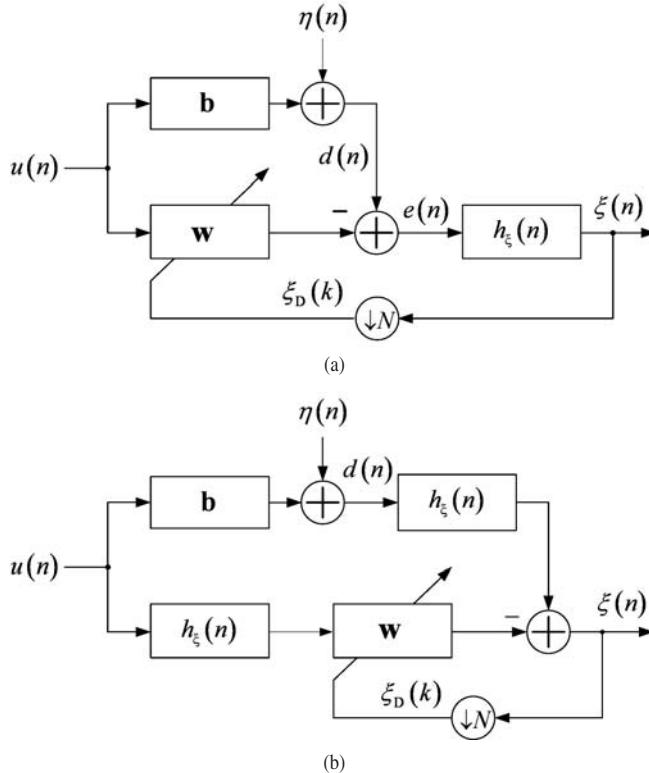


Figure 6.9 (a) Minimizing the weighted MSE function is equivalent to minimizing the equalized error function. (b) An equivalent structure obtained from (a) by moving $h_\xi(n)$ into the branches and also across the adaptive filter \mathbf{w} . The vector $\mathbf{b} = [b_0, b_1, \dots, b_{M-1}]^T$ is the parameter vector of the unknown system $B(z)$ (adapted from Reference [19])

Let $l = 0$ in Equations (6.42) and (6.46), we obtain

$$\sum_{i=0}^{N-1} \lambda_i^{-1} E\{e_i^2(n)\} = E\{\xi^2(n)\}. \quad (6.47)$$

Because the variance of the subband signal is equivalent to its decimated version, we have

$$E\{e_{i,D}^2(k)\} = E\{e_i^2(kN)\} \text{ and } E\{\xi_D^2(k)\} = E\{\xi^2(kN)\}, \quad (6.48)$$

where $\xi_D(k) = \xi(kN)$ is the decimated version of the preconditioned error signal $\xi(n)$. Now, rewrite the MSE function defined in Equation (6.38) using Equations (6.47) and (6.48), we obtain an equivalent representation of the weighted MSE function as

$$J_\xi = \sum_{i=0}^{N-1} \lambda_i^{-1} E\{e_i^2(kN)\} = E\{\xi^2(kN)\} = E\{\xi_D^2(k)\}. \quad (6.49)$$

Clearly, minimizing the weighted MSE function of Equation (6.38) is equivalent to minimizing the preconditioned error function defined in Equation (6.49). The idea is illustrated in Figure 6.9(a). Since we assume that the input signal is wide-sense stationary and the weight adaptation is frozen, the filter $h_\xi(n)$ can be moved into the branches and across the adaptive filter $W(k, z)$, as shown in Figure 6.9(b). In that respect, the filter $h_\xi(n)$ can be regarded as an equalization filter (i.e. an estimate of the inverse of the coloring filter for the input signal) that flattens the input spectrum $\Gamma_{uu}(e^{j\omega})$ to reduce its spectral dynamic range. Using Equation (6.45) in Figure 6.9(b), the equalized spectrum $\Gamma_\xi(e^{j\omega})$ can be written as

$$\begin{aligned}\Gamma_\xi(e^{j\omega}) &= |H_\xi(e^{j\omega})|^2 \Gamma_{uu}(e^{j\omega}) \\ &= \sum_{i=0}^{N-1} \lambda_i^{-1} |H_i(e^{j\omega})|^2 \Gamma_{uu}(e^{j\omega}).\end{aligned}\quad (6.50)$$

For random signals, equalization can be interpreted as whitening, thereby decorrelating the input signals. Thus the subband decomposition and normalization features shown in Equation (6.38) essentially form a decorrelation filter $h_\xi(n)$ that whitens the input signal before passing it to the adaptive filter. The idea is somewhat similar to the pre-whitening type of adaptive algorithms [8, pp. 50–53, 11, 12, 37], but the difference is that the decorrelation filter is embedded in the algorithm to form a self-designing decorrelation mechanism.

6.4.3 Decorrelating properties

From stochastic interpretation, the MSAF algorithm possesses some fundamental decorrelation properties. At one end, the decorrelation properties manifest themselves as an orthogonalization matrix $[\mathbf{U}(k)\mathbf{U}^T(k) + \alpha\mathbf{I}]^{-1}$, given in Equation (6.32), which forces the direction of adaptation at each iteration pointing to the minimum MSE. At the other end, the decorrelation properties appear in the form of a decorrelation filter $h_\xi(n)$, which whitens the input signal prior to weight adaptation. In both cases, the underlying principle remains the same. The MSAF algorithm decorrelates the input signal and follows the steepest-descent path on the preconditioned error-performance surface given in Equation (6.49).

The effectiveness of the inherent decorrelation filter $h_\xi(n)$ in dealing with colored signals is greatly dependent on the characteristics of analysis filters $h_0(n), \dots, h_{N-1}(n)$, which form the basis in constructing $h_\xi(n) * h_\xi(-n)$ as indicated by Equation (6.44). Taking a closer look at Figure 6.7, this shows that the steepest-descent-based algorithm can be translated into the Newton-based algorithm when the following conditions are satisfied:

- (i) the analysis filters are power complementary,
- (ii) the diagonal assumption is valid,
- (iii) the weighting factor $\lambda_i^{-1} = [\|\mathbf{u}_i(k)\|^2 + \alpha]^{-1}$ is employed.

Newton's method is expected to have better transient behavior (i.e. faster convergence) than the steepest-descent method. Therefore, the above requirements define an appropriate basis and weighting factors in forming an efficient decorrelation filter $h_\xi(n)$.

6.5 Filter bank design issues

This section discusses several issues related to design specific analysis and synthesis filter banks for the MSAF.

6.5.1 The diagonal assumption

The diagonal assumption states that the off-diagonal elements of the matrix

$$\mathbf{U}^T(k)\mathbf{U}(k) = \begin{bmatrix} \mathbf{u}_0^T(k)\mathbf{u}_0(k) & \cdots & \mathbf{u}_0^T(k)\mathbf{u}_{N-1}(k) \\ \vdots & \ddots & \vdots \\ \mathbf{u}_{N-1}^T(k)\mathbf{u}_0(k) & \cdots & \mathbf{u}_{N-1}^T(k)\mathbf{u}_{N-1}(k) \end{bmatrix} \quad (6.51)$$

are negligible; thus the matrix can be approximated by its diagonal counterpart, which contains only the diagonal elements $\lambda_i(k) = \|\mathbf{u}_i(k)\|^2$. This assumption holds if the filter bank outputs $u_i(n)$ and $u_p(n)$ are orthogonal at zero lag, i.e.

$$\gamma_{ip}(0) \equiv E\{u_i(n)u_p(n)\} = 0 \text{ for } i \neq p, \quad (6.52)$$

while the filter bank is persistently excited at all frequencies. This orthogonal condition can be achieved using cosine-modulated filter banks, as proven in Section 3.4.

Assuming that the input signal $u(n)$ is ergodic (and thus wide-sense stationary), the cross-correlation functions, $\gamma_{ip}(0)$ for $i, p = 0, 1, \dots, N - 1$, can be approximated with the time average $\hat{\gamma}_{ip}(0) = \mathbf{u}_i^T(k)\mathbf{u}_p(k)/M$. In the case where $i = p$, the time average $\hat{\gamma}_{ii}(0) = \|\mathbf{u}_i(k)\|^2/M$ is the variance estimate of the subband signal $u_i(n)$. Under the situation where the filter bank is persistently excited at all frequencies, the orthogonal condition (i.e. orthogonality at zero lag) specified in Equation (6.52) ensures that the off-diagonal elements $\mathbf{u}_i^T(k)\mathbf{u}_p(k) = M\hat{\gamma}_{ip}(0)$ of the matrix $\mathbf{U}^T(k)\mathbf{U}(k)$ are much smaller than its diagonal elements $\|\mathbf{u}_i(k)\|^2 = M\hat{\gamma}_{ii}(0)$. Therefore, the matrix $\mathbf{U}^T(k)\mathbf{U}(k)$ can be simplified to a diagonal matrix to justify the diagonal assumption. It should be emphasized that the algorithm given in Equation (6.21) is recursive in nature, which effectively averages the estimates of $\hat{\gamma}_{ip}(0)$ during adaptation. Hence, the length of weight vector M may not restrict the validity of the diagonal assumption; i.e. the diagonal assumption would still hold for lower orders of adaptive FIR filters.

6.5.2 Power complementary filter bank

From Equations (6.3) and (6.30), the expected value of the instantaneous estimate $\hat{\mathbf{R}}(k)$ can be expressed in an expanded form as

$$E\{\hat{\mathbf{R}}(k)\} = \begin{bmatrix} \gamma(0) & \cdots & \gamma(M-1) \\ \vdots & \ddots & \vdots \\ \gamma(-M+1) & \cdots & \gamma(0) \end{bmatrix}, \quad (6.53)$$

where the elements, $\gamma(l)$ for $|l| = 0, 1, \dots, M - 1$, are given by

$$\gamma(l) = \frac{1}{N} \sum_{i=0}^{N-1} E\{u_i(kN)u_i(kN-l)\}. \quad (6.54)$$

Next, consider the subband structure in Figure 6.3. For N filter bank outputs $u_i(n)$, their autocorrelation function $\gamma_{ii}(l) \equiv E\{u_i(n)u_i(n-l)\}$ can be formulated in terms of the input autocorrelation function $\gamma_{uu}(l) \equiv E\{u(n)u(n-l)\}$ as

$$\gamma_{ii}(l) = h_i(l) * h_i(-l) * \gamma_{uu}(l). \quad (6.55)$$

Using Equation (6.55) in Equation (6.54), we obtain

$$\gamma(l) = \left[\frac{1}{N} \sum_{i=0}^{N-1} h_i(l) * h_i(-l) \right] * \gamma_{uu}(l). \quad (6.56)$$

If the analysis filters are power complementary, i.e.

$$\frac{1}{N} \sum_{i=0}^{N-1} h_i(l) * h_i(-l) = \delta(l), \quad (6.57)$$

Equation (6.56) reduces to $\gamma(l) = \gamma_{uu}(l)$, which implies that $\hat{\mathbf{R}}(k)$ is an unbiased estimator of the ensemble average \mathbf{R} . A similar procedure can be used to show that $\hat{\mathbf{p}}(k)$ is an unbiased estimator of \mathbf{p} . In Equation (6.57), we assumed that all the analysis filters have been scaled to obtain a magnitude of \sqrt{N} in the frequency response $|H(e^{j\omega})|$.

6.5.3 The number of subbands

The quality of the orthogonalization matrix depends on the accuracy of using the instantaneous estimate $\hat{\mathbf{R}}(k)$ defined in Equation (6.30) to approximate \mathbf{R} . Assuming that $\mathbf{U}(k)$ has full column rank, it is clear that $\hat{\mathbf{R}}(k)$ will have the rank equal to the number of subbands N . In this case, the regularization constant α' added to the diagonal elements of $\hat{\mathbf{R}}(k)$ prevents the undesired behavior in inverting a rank defective matrix. The rank of $\hat{\mathbf{R}}(k)$ approaches \mathbf{R} of rank M as the number of subbands N approaches the length of the adaptive filter M . This fact suggests that a more efficient orthogonalization matrix, and thus a faster convergence, can be attained with more subbands.

The number of subbands N to be used in a given application depends on the spectral distribution of the input signal. Generally, more subbands are required for input spectra with more peaks and valleys. For a power spectrum $\Gamma_{uu}(e^{j\omega})$ with N peaks and valleys, we need N analysis filters to decompose the spectrum into N subband spectra $|H_i(e^{j\omega})|^2 \Gamma_{uu}(e^{j\omega})$, such that each subband contains only a single peak or valley. These subbands are normalized and recombined to form the equalized spectrum $\Gamma_\xi(e^{j\omega})$ given in Equation (6.50). The spectral dynamic range of $\Gamma_\xi(e^{j\omega})$ is reduced because these N peaks and valleys are aligned to have a similar level by normalization. Notice that for real-valued input signals and filter coefficients, both $|H_i(e^{j\omega})|^2$ and $\Gamma_{uu}(e^{j\omega})$ are even

and periodic functions with a fundamental period of 2π . Therefore, only the positive frequency region $0 \leq \omega \leq \pi$ has to be considered.

For autoregressive (AR) signals, the number of peaks in the power spectrum depends on the order and locations of the poles. Generally, a higher-order AR model has more peaks (and valleys as well) in its spectrum. In that sense, the number of subbands N can be related to the order of the AR signal; i.e. the number of subbands to be used can be determined by some known order-selection criteria, such as the Akaike information criterion and the final prediction error criterion [38], which are commonly used in a parametric spectrum estimation.

6.6 Delayless MSAF

The MSAF algorithm can be implemented in open-loop and closed-loop delayless configurations. For both delayless configurations, the subband-to-fullband weight transformation (discussed in Section 4.4) is not required since the MSAF algorithm derives the fullband tap weights directly from the subband signals. The tap weights updated by the subband signals are directly copied to a fullband adaptive filter in order to eliminate the delay inflicted by the filter banks. It should be emphasized that the MSAF algorithm does not suffer from aliasing and band-edge effects. Hence, it does not require the closed-loop feedback of the fullband error signal to overcome the structural problems discussed in Chapter 4.

6.6.1 Open-loop configuration

Figure 6.10 shows the open-loop implementation of the delayless MSAF algorithm. The steps, parameters and variables of the open-loop algorithm are described in

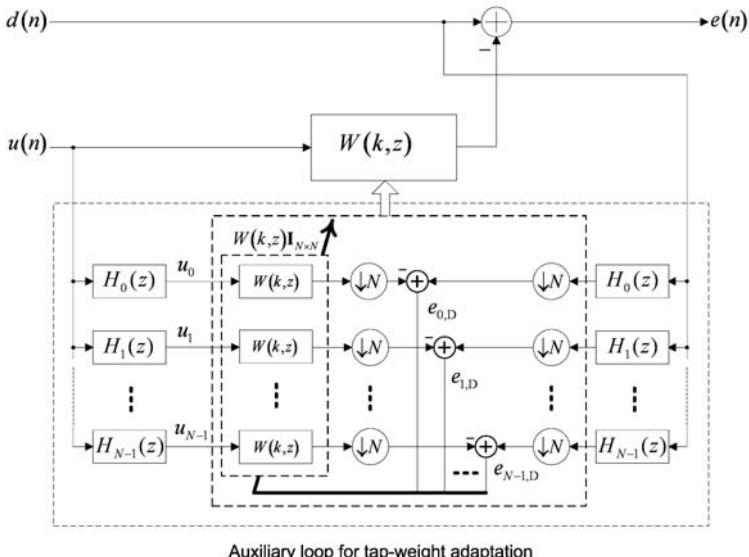


Figure 6.10 An open-loop delayless MSAF algorithm (adapted from Reference [39])

Table 6.3 Open-loop implementation of the delayless MSAF algorithm

Computation	Multiplications per T_s
For $n = 0, 1, 2, \dots$,	
<i>Filtering and error formation:</i>	
$e(n) = d(n) - \mathbf{w}^T(k)\mathbf{u}(n)$	M
For $k = 0, 1, 2, \dots$, where $kN = n$,	
<i>Error estimation:</i>	$\frac{N \times M}{N} = M$
$\mathbf{e}_D(k) = \mathbf{d}_D(k) - \mathbf{U}^T(k)\mathbf{w}(k)$	$\frac{N \times M}{N} = M$
<i>Normalization matrix:</i>	$\frac{N \times M}{N} = M$
$\mathbf{A}(k) = \mathbf{U}^T(k)\mathbf{U}(k) + \alpha\mathbf{I}$	$\frac{2N + NM}{N} = M + 2$
<i>Tap-weight adaptation:</i>	
$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu\mathbf{U}(k)\mathbf{A}^{-1}(k)\mathbf{e}_D(k)$	$\frac{N^2L + NL}{N} = (N+1)L$
<i>Band partitioning:</i>	
$\mathbf{U}_1^T(k) = \mathbf{H}^T\mathbf{A}(kN)$	
$\mathbf{d}_D(k) = \mathbf{H}^T\mathbf{d}(kN)$	
<i>Parameter:</i>	
α - small positive constant	
<i>Variables:</i>	
$\mathbf{U}^T(k) = [\mathbf{U}_1^T(k), \mathbf{U}_2^T(k-1)]$	
$\mathbf{U}_2^T(k-1)$ - first $M-N$ columns of $\mathbf{U}^T(k-1)$	
$\mathbf{A}(kN) = [\mathbf{a}(kN), \mathbf{a}(kN-1), \dots, \mathbf{a}(kN-N+1)]$	
$\mathbf{a}(kN) = [u(kN), u(kN-1), \dots, u(kN-L+1)]^T$	
$\mathbf{d}(kN) = [d(kN), d(kN-1), \dots, d(kN-L+1)]^T$	
$\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{N-1}]$ is analysis filter bank	

Table 6.3. Notice that the matrix \mathbf{H} denotes the analysis filter bank with its columns $\mathbf{h}_i = [h_i(0), h_i(1), \dots, h_i(L-1)]^T$ representing the analysis filters. In the open-loop configuration, $e(n)$ is not fed back for updating tap weights. The fullband error signal $e(n)$ and the subband estimation error vector $\mathbf{e}_D(k) \equiv [e_{0,D}(k), e_{1,D}(k), \dots, e_{N-1,D}(k)]^T$ are two different error quantities. In the case of acoustic echo cancellation, $e(n)$ is the near-end signal to be transmitted to the far-end room, while $\mathbf{e}_D(k)$ serves as the estimation error signal for the tap-weight adaptation. The fullband filtering process is an unrelated operation that does not affect the convergence behavior of the MSAF algorithm. The convergence of the open-loop delayless MSAF is therefore similar to that of the original MSAF algorithm shown in Figure 6.3, except for the signal-path delay (imposed by the filter banks) that has been eliminated with the delayless structure.

The computational cost of the open-loop delayless MSAF algorithm in terms of the number of multiplications per sampling period T_s is also summarized in Table 6.3. Notice that the number of multiplications incurred at each iteration is divided by N because the adaptation process is performed at the critically decimated rate of $1/NT_s$. The MSAF algorithm in the open-loop delayless configuration requires $3M + 2$ multiplications for the adaptation process, M multiplications for the fullband filtering process and $(N+1)L$ multiplications for the two analysis filter banks. Thus the total number of multiplications needed in one sampling period is $4M + NL + L + 2$. Clearly, the computational cost of

the delayless MSAF is higher than that of the delayless SAFs presented in Section 4.4, which are based on the conventional SAF structures. However, the delayless MSAF would be preferable in terms of convergence performance, especially for the colored input signals.

6.6.2 Closed-loop configuration

The closed-loop implementation of the delayless MSAF algorithm is described in Figure 6.11 and in Table 6.4. It is different from the open-loop configuration as the subband estimation error $e_D^{cl}(k) = \mathbf{H}^T \mathbf{e}(kN)$ is derived from the fullband error $\mathbf{e}(kN) \equiv [e(kN), e(kN - 1), \dots, e(kN - L + 1)]^T$ by the analysis filter bank \mathbf{H} . The fullband error signal $e(n)$ is generated by taking the difference between the desired response $d(n)$ and the output of the modeling filter $W(z)$. A delay is introduced by the analysis filter bank applied to the fullband error $e(n)$. Furthermore, the adaptation algorithm uses past information from previous estimations $e(kN - 1), e(kN - 2), \dots, e(kN - L + 1)$ that appear in the tapped-delay line of the analysis filters. These two factors limit the convergence rate and reduce the upper bound of the step size that can be employed for the closed-loop MSAF algorithm [17]; i.e. the delayless MSAF algorithm implemented in a closed-loop structure behaves differently from that of the original MSAF algorithm shown in Figure 6.3.

The closed-loop delayless MSAF algorithm is summarized in Table 6.4. It can be noted that the closed-loop delayless MSAF algorithm requires a total number of $3M + NL + L + 2$ multiplications in one sampling period, which is less (about M multiplications) than the open-loop delayless MSAF. The computational saving is achieved by

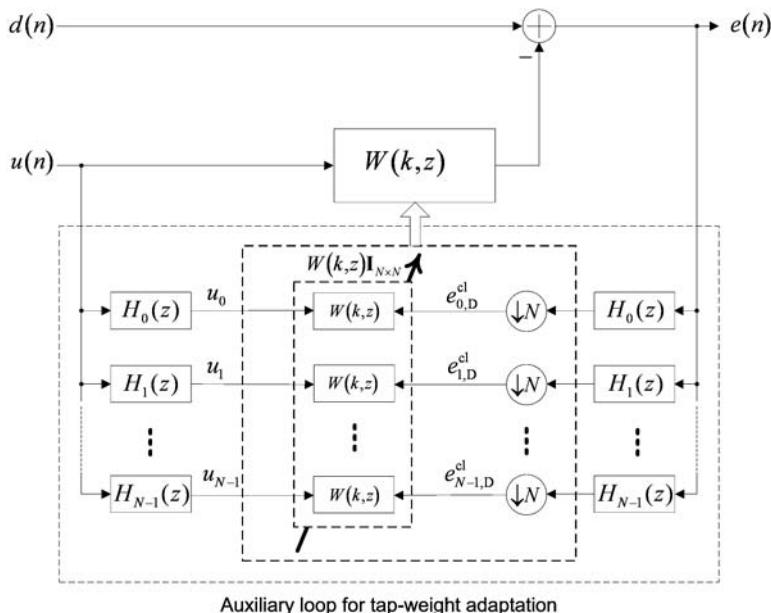


Figure 6.11 A closed-loop delayless MSAF algorithm (adapted from Reference [39])

Table 6.4 Open-loop implementation of the delayless MSAF algorithm

Computation	Multiplications per T_s
For $n = 0, 1, 2, \dots$, <i>Filtering and error formation:</i> $e(n) = d(n) - \mathbf{w}^T(k)\mathbf{u}(n)$	M
For $k = 0, 1, 2, \dots$, where $kN = n$, <i>Normalization matrix:</i> $\mathbf{A}(k) = \mathbf{U}^T(k)\mathbf{U}(k) + \alpha\mathbf{I}$	$\frac{N \times M}{N} = M$
<i>Tap-weight adaptation:</i> $\mathbf{w}(k+1) = \mathbf{w}(k) + \mu\mathbf{U}(k)\mathbf{A}^{-1}(k)\mathbf{e}_D^{cl}(k)$	$\frac{2N + NM}{N} = M + 2$
<i>Band partitioning:</i> $\mathbf{U}_1^T(k) = \mathbf{H}^T\mathbf{A}(kN)$ $\mathbf{e}_D^{cl}(k) = \mathbf{H}^T\mathbf{e}(kN)$	$\frac{N^2L + NL}{N} = (N+1)L$
<i>Parameter:</i> α - small positive constant	
<i>Variables:</i> $\mathbf{U}^T(k) = [\mathbf{U}_1^T(k), \mathbf{U}_2^T(k-1)]$ $\mathbf{U}_2^T(k-1)$ - first $M-N$ columns of $\mathbf{U}^T(k-1)$ $\mathbf{A}(kN) = [\mathbf{a}(kN), \mathbf{a}(kN-1), \dots, \mathbf{a}(kN-N+1)]$ $\mathbf{a}(kN) = [u(kN), u(kN-1), \dots, u(kN-L+1)]^T$ $\mathbf{d}(kN) = [d(kN), d(kN-1), \dots, d(kN-L+1)]^T$ $\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{N-1}]$ is analysis filter bank	

deriving the subband estimation error $\mathbf{e}_D^{cl}(k)$ from the fullband error $\mathbf{e}(kN)$ instead of using the subband error estimation $\mathbf{e}_D(k) = \mathbf{d}_D(k) - \mathbf{U}^T(k)\mathbf{w}(k)$. The price paid for the computational reduction is the degradation of the convergence rate.

The closed-loop delayless MSAF is called the weighted SAF in [16, 17]. In fact, the formulation and the approach taken in deriving the weighted SAF algorithm are different from those of the closed-loop delayless MSAF algorithm. The weighted SAF algorithm is derived as a stochastic-gradient algorithm, whereas the MSAF is derived with the deterministic approach based on the principle of minimal disturbance. Nevertheless, both algorithms are identical in the sense that they have similar structures for filtering and weight adaptation.

6.7 MATLAB examples

This section presents several computer simulations based on the application of system identification as shown in Figure 6.1. These simulations are designed to illustrate further the analytical results given in the previous sections and to show the improved performance of the MSAF algorithm in dealing with colored signals and high-order identification problems. The unknown system $B(z)$ to be identified is an actual room acoustic path measured at 8 kHz sampling frequency. The impulse response of the FIR system $B(z)$

is represented by vector \mathbf{b} of length 1024, as shown in Figure 4.19. Furthermore, three different input signals, including white Gaussian noise, AR signals and real speech signals, are used for the simulations.

In all simulations, the length of weight vector $\mathbf{w}(k)$ is $M = 1024$, which is identical to the length of vector \mathbf{b} . Tap weights of the adaptive filter are initialized to zero. White noise that is uncorrelated with the input signal is added to the output of the unknown system as $\eta(n)$ shown in Figure 6.1, giving a 40 dB signal-to-noise ratio. The added noise disrupts the weight vector $\mathbf{w}(k)$ to identify perfectly the unknown system represented by \mathbf{b} ; thus the steady-state MSE is a nonzero value bounded by the power of noise $\eta(n)$.

6.7.1 Convergence of the MSAF algorithm

In the first set of simulations, the convergence of the MSAF algorithm with different numbers ($N = 4, 8, 16$ and 32) of subbands is investigated. Pseudo-QMF cosine-modulated filter banks are used for the analysis and synthesis filters with a filter length of $L = 8N$. The length L of the analysis filters increases with N to maintain 60 dB of stopband attenuation. This high stopband attenuation significantly reduces overlap between adjacent analysis filters. The cross-correlation between nonadjacent subbands can also be completely neglected. On the other hand, by virtue of the cosine-modulated analysis filters, adjacent subband signals are orthogonal at zero lag. Recall that the diagonal assumption (stated in Section 6.5.1) is justified if the subband signals are orthogonal at $l = 0$.

The MSE learning curves of the NLMS and MSAF algorithms using a white noise as input signal are shown in Figure 6.12. The MSE learning curves are obtained by ensemble averaging over 200 independent trials and smoothed by a 10-point moving-average filter.

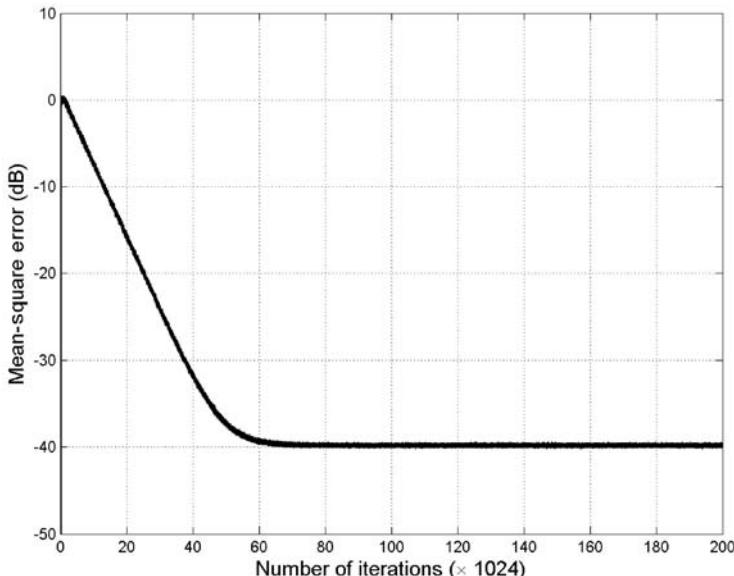


Figure 6.12 MSE learning curves of the NLMS algorithm and the MSAF algorithm with $N = 4, 8, 16$ and 32 subbands under white noise excitation. These five learning curves are overlapped, which shows that they have almost identical convergence performance

For all cases, the step size used for both algorithms is $\mu = 0.1$ and the regularization parameter α is 0.0001. An identical steady-state MSE is achieved for both algorithms, thus allowing a fair comparison of their transient behavior. Figure 6.12 shows that the learning curves of the NLMS and MSAF algorithms with different numbers of subbands N are the same when the input signal is white noise. The reason is that the inherent decorrelation mechanism of the MSAF algorithm has no effect on the uncorrelated white signals.

Figure 6.13 shows the advantages of using the MSAF algorithm as compared with the NLMS algorithm for the colored AR(2) signal. The spectrum of the AR(2) signal is shown in Figure 6.14(a). Because the input signal has a large spectral dynamic range, the NLMS algorithm exhibits a slow asymptotic convergence after a fast initial convergence. On the other hand, the MSAF exhibits consistently faster convergence rates (i.e. the rate of learning curves decays to the steady-state MSE) due to the reduced spectral dynamic range of the equalized spectrum $\Gamma_{\xi}(e^{j\omega})$. Furthermore, the convergence rate improves considerably with the increased number of subbands. It can be observed that the convergence improvement obtained by increasing the number of subbands N tapers off with higher numbers of subbands. The number of subbands to be used in any specific application is determined by the ability to design an inherent decorrelation filter $|H_{\xi}(e^{j\omega})|^2$ to match the inverse of the coloring filter. Generally more subbands are required for input signals with more peaks and valleys in the magnitude spectrum. The complete listing of the MATLAB code can be found in `Example_6P3.m`.

The MSAF algorithm converges faster than the NLMS algorithm mainly due to its inherent decorrelation mechanism, which whitens the input signal prior to weight adaptation. Figure 6.14(a) shows the power spectrum of the AR(2) signal $\Gamma_{uu}(e^{j\omega})$ with a spectral dynamic range of 25.11 dB. The inherent decorrelation filter $|H_{\xi}(e^{j\omega})|^2$ of the MSAF algorithm is adjusted in response to this colored excitation, as shown in Figure 6.14(b). More precisely, it partitions the input spectrum $\Gamma_{uu}(e^{j\omega})$ into $N = 4$ overlapped spectral bands, normalizes each subband with its respective subband energy

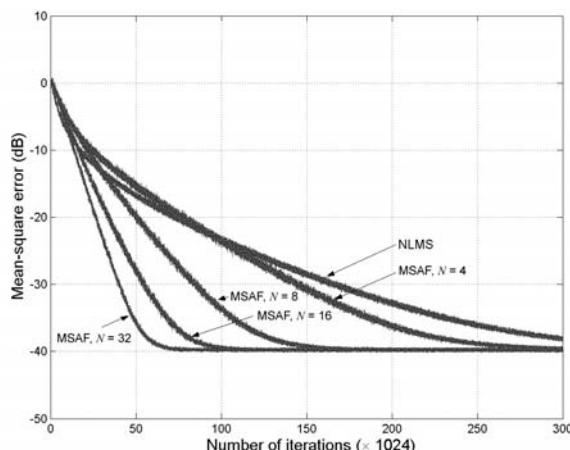


Figure 6.13 MSE learning curves of the NLMS algorithm and the MSAF algorithm with $N = 4, 8, 16$ and 32 subbands under the colored AR(2) signal. The step size and regularization factor are $\mu = 0.1$ and $\alpha = 0.0001$, respectively

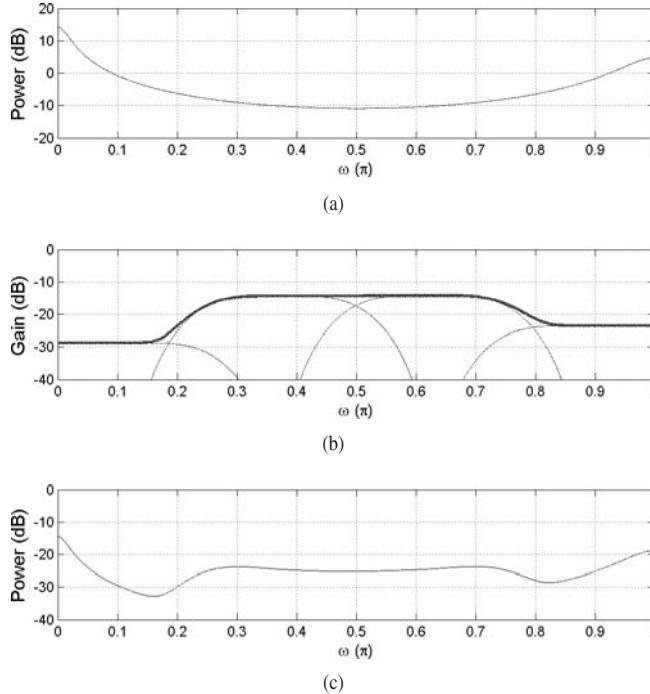


Figure 6.14 The inherent decorrelating mechanism of the MSAF algorithm: (a) the power spectrum of the AR(2) signal used in the simulations, (b) the magnitude response (indicated by the thick line) of the inherent decorrelation filter with $N = 4$ and (c) the resulting equalized spectrum

$\lambda_i(k) = \|\mathbf{u}_i(k)\|^2 + \alpha$ and then recombines the normalized subbands to yield the equalized power spectrum $\Gamma_{\xi}(e^{j\omega}) = |H_{\xi}(e^{j\omega})|^2 \Gamma_u(e^{j\omega})$, as shown in Figure 6.14(c), with a reduced spectral dynamic range of 18.34 dB.

Figure 6.15 shows the decorrelation filter for the cases of $N = 8, 16$ and 32 subbands. Clearly, a better match to the inverse of the input spectrum can be obtained by increasing the number of subbands. The performance of the decorrelation filter can be observed from the resulting equalized spectra shown in Figure 6.16. The spectral dynamic range of the equalized spectrum reduced significantly when the number of subbands N increased. For the cases of $8, 16$ and 32 subbands, the spectral dynamic ranges are 12.99 dB, 7.94 dB and 3.86 dB, respectively. Details of the analysis can be found in `Example_6P4.m`.

6.7.2 Subband and time-domain constraints

In the second set of simulations, the convergence of the MSAF algorithm is compared with the NLMS and AP algorithms. Recall that the NLMS algorithm updates the tap weights on the basis of an input vector $\mathbf{u}(n)$. The AP algorithm (introduced in Section 1.4) extends the NLMS algorithm by introducing additional $P - 1$ past input vectors, $\mathbf{u}(n - 1), \dots, \mathbf{u}(n - P + 1)$, into the weight adaptation. On the other hand, the MSAF algorithm extends the NLMS algorithm by decomposing the input vector $\mathbf{u}(n)$ into N subband vectors $\mathbf{u}_0(k), \mathbf{u}_1(k), \dots, \mathbf{u}_{N-1}(k)$. It should be emphasized that, in all cases, the coefficients to

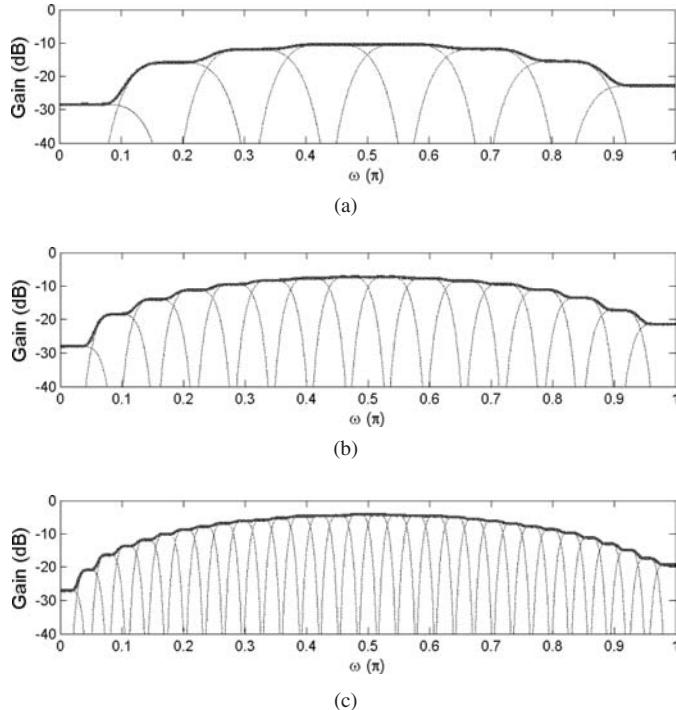


Figure 6.15 The magnitude responses of inherent decorrelation filters of the MSAF algorithm for (a) $N = 8$, (b) $N = 16$ and (c) $N = 32$

be adapted are the fullband tap weights $\{w_m(k)\}_{m=0}^{M-1}$. The aim of the simulations is to compare the merit of employing multiband constraints and time-domain constraints in terms of the convergence rate and computational complexity. The complete listing of the MATLAB code can be found in `Example_6P5.m` and `Example_6P6.m`.

Two highly correlated excitation signals, an AR(10) signal and real speech, are used in the simulations. The AR coefficients are obtained using the Yule–Walker method on a segment of unit-variance speech. The power spectrum of the AR(10) signal closely resembles the power spectrum of speech, as shown in Figure 6.17. The MSAF algorithm has $N = 10$ subbands using pseudo-QMF cosine-modulated filter banks with an analysis filter length of $L = 8N = 80$. Two variations of the sample- and block-based AP algorithm are considered, both with the order of $P = 10$ (note that we use $P = N = 10$ for a fair comparison). The block-based AP algorithm is commonly known as the partial-rank algorithm (PRA) [27, 28]. The step size and regularization parameter are $\mu = 0.1$ and $\alpha = 0.0001$, respectively, such that an identical steady-state MSE is achieved by all the algorithms for the stationary excitation.

Figure 6.18 shows the ensemble-averaged learning curves for using the AR(10) signal as the excitation input. It can be observed that the convergence speed of the MSAF algorithm is faster than the NLMS algorithm and close to the AP algorithm and PRA. The subband decomposition and adaptation process of the MSAF algorithm altogether required a computational cost of $O(M) + O(NL)$. On the other hand, the computational complexities of the NLMS, PRA and AP algorithms are $O(M)$, $O(PM)$ and $O(P^2M)$,

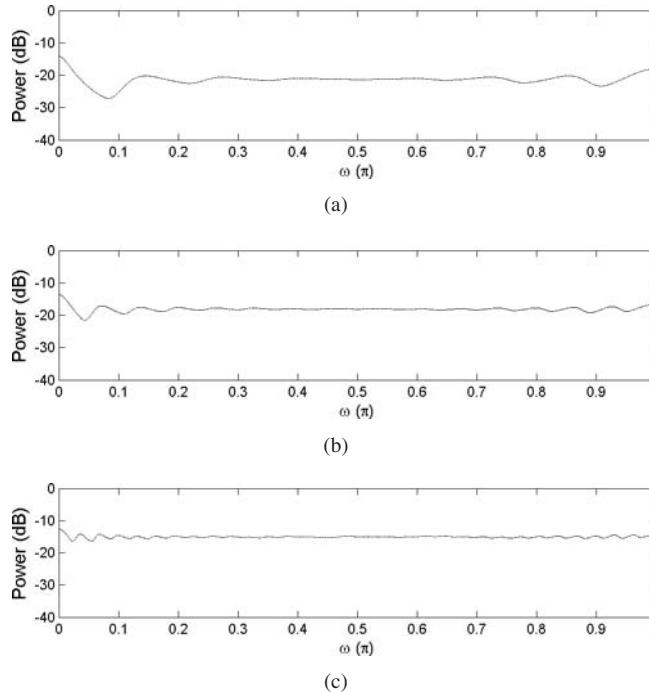


Figure 6.16 The equalized spectra of the MSAF algorithm for (a) $N = 8$, (b) $N = 16$ and (c) $N = 32$

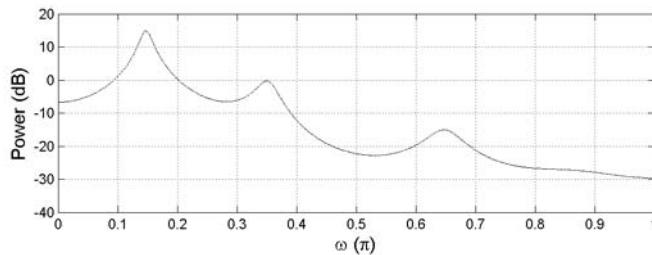


Figure 6.17 Power spectrum of the AR(10) signal derived from a segment of speech

respectively. For $M = 1024$ and $P = N = 10$, the NLMS, MSAF, PRA and AP algorithms, respectively, require 3073, 4034, 12 299 and 122 990 multiplications per sampling period T_s . Clearly, the MSAF algorithm achieves a similar convergence rate to the AP algorithm, with computational complexity close to the NLMS algorithm.

Figure 6.19 shows the convergence behaviors of the NLMS, PRA, MSAF and AP algorithms for the speech signal shown on the top panel. The iteration index k is taken as n for the cases of the NLMS and AP algorithms since the weight adaptation is performed at the original sampling rate. For the cases of the MSAF and PRA algorithms, the misalignment learning curves (defined in Equation (4.12)) are plotted as functions of kN .

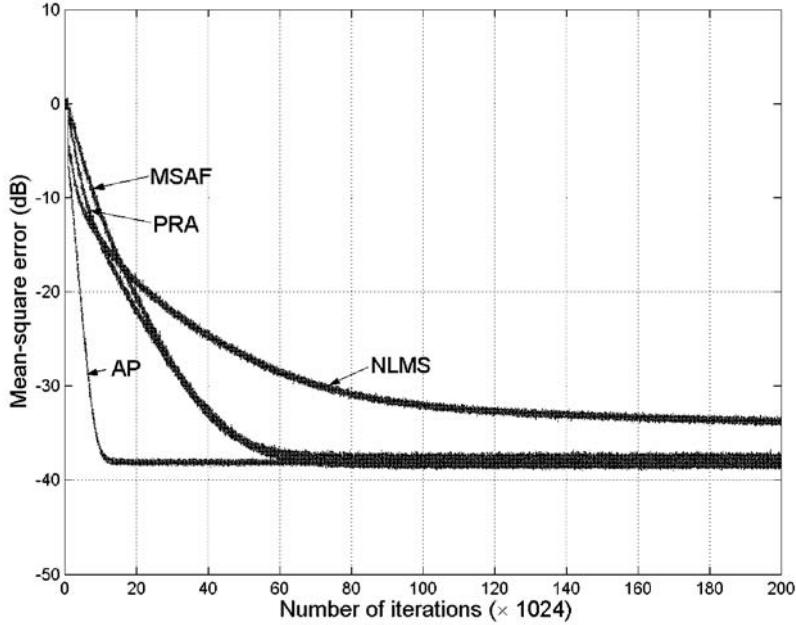


Figure 6.18 MSE learning curves of the NLMS, PRA, MSAF and AP algorithms. The step size and regularization factor are $\mu = 0.1$ and $\alpha = 0.0001$, respectively

Furthermore, a noiseless situation is considered for simplicity; i.e. no disturbance $\eta(n)$ is added to the output of the unknown system $B(z)$.

These normalized misalignment curves show that the MSAF algorithm performs equally well with the speech signal as compared with the stationary colored signals. The MSAF algorithm outperforms the NLMS and PRA algorithms. Its convergence speed is close to the AP algorithm with a far lower computational complexity. These results indicate that the analytical results obtained by assuming that the input signal is wide-sense stationary hold for the nonstationary input signal as well. The capability of the MSAF algorithm in dealing with nonstationary signals is mainly achieved by employing the running-average normalization matrix $\Lambda(k)$ defined in Equation (6.24) for the weight adaptation.

6.8 Summary

This chapter presented the MSAF, which uses the fullband adaptive filter updated by subband signals that are normalized by their respective variances. The characteristics and performance of the algorithm were analyzed and compared with other adaptive algorithms and verified by computer simulations. The MSAF algorithm was formulated in Section 6.3 as the minimum-norm solution to an underdetermined least-squares estimation problem. Within these deterministic frameworks, the MSAF algorithm can be seen as a recursive estimator that iteratively updates the weight vector in a minimal manner, while nulling the a posteriori error in all subbands. Similar least-perturbation properties were also found in the NLMS and AP algorithms. Both the AP and MSAF algorithms can be seen

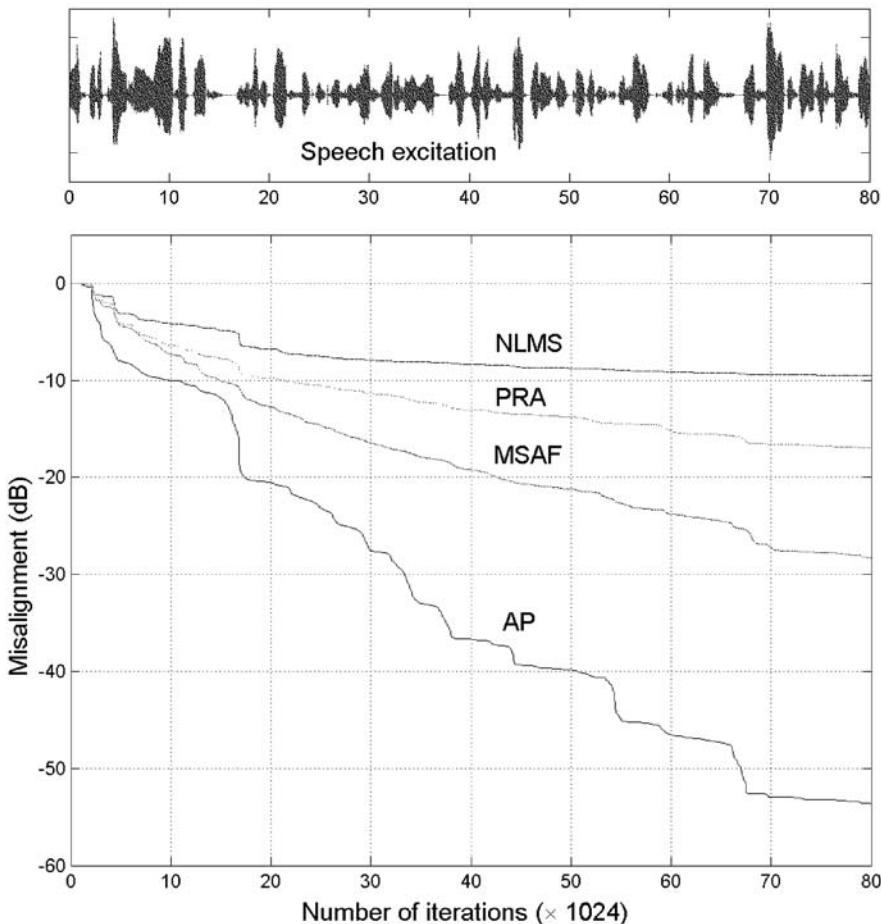


Figure 6.19 Normalized misalignment learning curves for the NLMS, PRA (order 10), MSAF (10 subbands) and AP (order 10) algorithms under speech excitation. The step size is $\mu = 0.1$

as generalized forms of the NLMS algorithm. The AP algorithm generalizes the NLMS along the time axis using multiple time-domain constraints, whereas the MSAF algorithm generalizes the NLMS along the frequency axis using multiple subband constraints.

References

- [1] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Englewood Cliffs, New Jersey: Prentice Hall, 1983.
- [2] E. A. B. da Silva and P. S. R. Diniz, ‘Time-varying filters’, in *Encyclopedia of Electrical and Electronics Engineering* (ed. J. G. Webster), New York: John Wiley & Sons, Inc., 1999.
- [3] N. J. Fliege, *Multirate Digital Signal Processing*, New York: John Wiley & Sons, Inc., 1994.

- [4] H. S. Malvar, *Signal Processing with Lapped Transform*, Norwood, Massachusetts: Artech House, 1992.
- [5] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, New York: McGraw-Hill, 2001.
- [6] T. Saramäki and R. Bregović, ‘Multirate systems and filter banks’, in *Multirate Systems: Design and Applications* (ed. G. Jovanovic-Dolecek), Hershey, Pennsylvania: Idea Group, 2002.
- [7] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Englewood Cliffs, New Jersey: Prentice Hall, 1993.
- [8] C. Breining, P. Dreiseitl, E. Hänsler, A. Mader, B. Nitsch, H. Puder, T. Schertler, G. Schmidt and J. Tilp, ‘Acoustic echo control – an application of very-high-order adaptive filters’, *IEEE Signal Processing Mag.*, **16**(4), July 1999, 42–69.
- [9] P. Dreiseitel, E. Hänsler and H. Puder, ‘Acoustic echo and noise control – a long lasting challenge’, in *Proc. EUSIPCO*, 1998, vol. 2, pp. 945–952.
- [10] S. L. Gay and J. Benesty (eds), *Acoustic Signal Processing for Telecommunication*, Norwell, Massachusetts: Kluwer Academic Publishers, 2000.
- [11] E. Hänsler, ‘Adaptive echo compensation applied to the hands-free telephone problem’, in *Proc. IEEE ICAS*, 1990, vol. 1, pp. 279–282.
- [12] E. Hänsler, ‘The hands-free telephone problem’, in *Proc. IEEE ICAS*, 1992, vol. 4, pp. 1914–1917.
- [13] E. Hänsler, ‘The hands-free telephone problem – an annotated bibliography’, *Signal Processing*, **27**(3), June 1992, 259–271.
- [14] A. Mader, H. Puder and G. U. Schmidt, ‘Step-size control for acoustic echo cancellation filters – an overview’, *Signal Processing*, **80**(9), September 2000, 1697–1719.
- [15] G. Schmidt, ‘Applications of acoustic echo control – an overview’, in *Proc. EUSIPCO*, 2004, pp. 9–16.
- [16] M. de Courville and P. Duhamel, ‘Adaptive filtering in subbands using a weighted criterion’, in *Proc. IEEE ICASSP*, 1995, vol. 2, pp. 985–988.
- [17] M. de Courville and P. Duhamel, ‘Adaptive filtering in subbands using a weighted criterion’, *IEEE Trans. Signal Processing*, **46**(9), September 1998, 2359–2371.
- [18] K. A. Lee and W. S. Gan, ‘Improving convergence of the NLMS algorithm using constrained subband updates’, *IEEE Signal Processing Lett.*, **11**(9), September 2004, 736–739.
- [19] K. A. Lee and W. S. Gan, ‘Inherent decorrelating and least perturbation properties of the normalized subband adaptive filter’, *IEEE Trans. Signal Processing*, **54**(11), November 2006, 4475–4480.
- [20] K. Mayyas and T. Aboulnasr, ‘A fast weighted subband adaptive algorithm’, in *Proc. ICASSP*, 1999, vol. 3, pp. 1249–1252.
- [21] K. Mayyas and T. Aboulnasr, ‘A fast exact weighted subband adaptive algorithm and its application to mono and stereo acoustic echo cancellation’, *J. Franklin Institute*, **342**(3), May 2005, 235–253.
- [22] S. S. Pradhan and V. U. Reddy, ‘A new approach to subband adaptive filtering’, *IEEE Trans. Signal Processing*, **47**(3), March 1999, 655–664.
- [23] S. Haykin, *Adaptive Filter Theory*, 4th edition, Upper Saddle River, New Jersey: Prentice Hall, 2002.
- [24] D. G. Manolakis, V. K. Ingle and S. M. Kogon, *Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering and Array Processing*, New York: McGraw-Hill, 2000.

- [25] B. Widrow and M. A. Lehr, ‘30 years of adaptive neural networks: perceptron, Madaline, and backpropagation’, *Proc. IEEE*, Special Issue on Neural Networks, **78**(9), September 1990, 1415–1442.
- [26] K. A. Lee, W. S. Gan and S. M. Kuo, Mean-square performance analysis of the normalized subband adaptive filter’, in *Proc. Asilomar Conf.*, 2006, pp. 248–252.
- [27] D. R. Morgan and S. G. Kratzer, ‘On a class of computationally efficient, rapidly convergence, generalized NLMS algorithms’, *IEEE Signal Processing Lett.*, **3**(8), August 1996, 245–247.
- [28] A. H. Sayed, *Fundamentals of Adaptive Filtering*, New York: John Wiley & Sons, Inc., 2003.
- [29] M. Rupp, ‘A family of adaptive filter algorithms with decorrelating properties’, *IEEE Trans. Signal Processing*, **46**(3), March 1998, 771–775.
- [30] D. T. M. Slock, ‘The block underdetermined covariance (BUC) fast transversal filter (FTF) algorithm for adaptive filtering’, in *Proc. Asilomar Conf.*, 1992, pp. 550–554.
- [31] D. T. M. Slock, ‘Underdetermined growing and sliding covariance fast transversal filter RLS algorithms’, in *Proc. EUSIPCO*, 1992, pp. 1169–1172.
- [32] S. L. Gay, ‘Affine projection algorithms’, in *Least-Mean-Square Adaptive Filters* (eds S. Haykin and B. Widrow), New York: John Wiley & Sons, Inc., 2004.
- [33] P. S. R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*, Norwell, Massachusetts: Kluwer Academic Publishers, 1997.
- [34] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.
- [35] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Englewood Cliffs, New Jersey: Prentice Hall, 1985.
- [36] G. O. Glentis, K. Berberidis and S. Theodoridis, ‘Efficient least squares adaptive algorithms for FIR transversal filtering’, *IEEE Signal Processing Mag.*, **16**(4), July 1999, 13–41.
- [37] R. Frenzel and M. E. Hennecke, Using prewhitening and stepsize control to improve the performance of the LMS algorithm for acoustic echo compensation’, in *Proc. IEEE ICASSP*, 1992, vol. 4, pp. 1930–1932.
- [38] J. G. Proakis and M. G. Manolakis, *Digital Signal Processing – Principles, Algorithms, and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 1996.
- [39] K. A. Lee and W. S. Gan, ‘On delayless architecture for the normalized subband adaptive filter’, in *Proc. IEEE ICME*, 2007, pp. 1595–1598.

Stability and performance analysis

This chapter analyzes the convergence of subband adaptive filters in the mean and mean-square senses. The performance of the SAF is characterized in terms of the stability, convergence speed and steady-state performance after the algorithm has converged. The mean-square performance is analyzed based on the energy conservation relation [1–7]. Stability bounds for the step size and the steady-state MSE are determined by manipulating various error terms in the energy conservation relation. The theoretical analysis provides a set of working rules for designing the SAF in practical applications.

7.1 Algorithm, data model and assumptions

This section briefly reviews the multiband-structured SAF (MSAF) for statistical analysis, and defines various terms, data models and assumptions that will be used later in the convergence analysis as well as an evaluation of the weight recursion.

7.1.1 The MSAF algorithm

The MSAF algorithm introduced in Chapter 6 (see Figure 6.3) is an effective subband adaptive filtering algorithm that partitions the input signal $u(n)$ and desired response $d(n)$ into multiple subbands for weight adaptation. However, different from the conventional SAF introduced in Chapter 4 where each subband has an individual adaptive subfilter in its own adaptation loop [8–11], the MSAF algorithm collectively adapts the fullband adaptive filter $W(k, z) = \sum_{m=0}^{M-1} w_m(k)z^{-m}$ using the complete set of subband signals. The adaptive algorithm updates the fullband weight vector $\mathbf{w}(k) \equiv [w_0(k), w_1(k), \dots, w_{M-1}(k)]^T$ iteratively as follows:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu \mathbf{U}(k) \boldsymbol{\Lambda}^{-1}(k) \mathbf{e}_D(k), \quad (7.1)$$

$$\mathbf{e}_D(k) = \mathbf{d}_D(k) - \mathbf{U}^T(k) \mathbf{w}(k), \quad (7.2)$$

where

$$\mathbf{A}(k) = \text{diag} [\mathbf{U}^T(k)\mathbf{U}(k) + \alpha\mathbf{I}], \quad (7.3)$$

μ is the step size and α is the regularization parameter. The $M \times 1$ subband signal vector (regressor) $\mathbf{u}_i(k)$, $M \times N$ data matrix $\mathbf{U}(k)$ and $N \times 1$ desired response vector $\mathbf{d}_D(k)$ are defined as follows:

$$\mathbf{u}_i(k) \equiv [u_i(kN), u_i(kN - 1), \dots, u_i(kN - M + 1)]^T, \quad (7.4)$$

$$\mathbf{U}(k) \equiv [\mathbf{u}_0(k), \mathbf{u}_1(k), \dots, \mathbf{u}_{N-1}(k)], \quad (7.5)$$

$$\mathbf{d}_D(k) \equiv [d_{0,D}(k), d_{1,D}(k), \dots, d_{N-1,D}(k)]^T. \quad (7.6)$$

Although the subband regressors $\mathbf{u}_i(k)$ defined in Equation (7.4) are bandlimited, they contain the subband signals $u_i(n)$ at the original sampling rate. Thus the fullband adaptive filter $W(k, z)$ operates on the subband signals at the original sampling rate while its tap weights are iteratively updated at the decimated rate. The filtering operation will not become the bottleneck in real-time implementation of the MSAF algorithm since most digital signal processors have optimized architecture to perform FIR filtering [12].

7.1.2 Linear data model

As shown in Figure 7.1, the physical mechanism for generating the desired response $d(n)$ can be described by the linear model expressed as [3, 13, 14]

$$d(n) = \mathbf{b}^T \mathbf{u}(n) + \eta(n), \quad (7.7)$$

where $\mathbf{b} = [b_0, b_1, \dots, b_{M-1}]^T$ is the parameter vector of the unknown system $B(z)$ that is represented by an FIR filter of length M , $\mathbf{u}(n) = [u(n), u(n - 1), \dots, u(n - M + 1)]^T$ is the input signal vector and $\eta(n)$ is the additive white noise that is statistically independent of signal $u(n)$. Furthermore, it is also assumed that the length M of the adaptive FIR filter $W(k, z)$ is exactly equal to the length of the impulse response of $B(z)$ contained in the vector \mathbf{b} .

Figure 7.1(a) shows the linear data model followed by a multirate filter bank. By virtue of the linearity of the analysis filters and decimators as depicted in Figure 7.1(b), the desired response vector $\mathbf{d}_D(k)$ defined in Equation (7.6) that is generated by the linear data model of Equation (7.7) can be written as

$$\mathbf{d}_D(k) = \mathbf{U}^T(k)\mathbf{b} + \boldsymbol{\eta}_D(k), \quad (7.8)$$

where $\boldsymbol{\eta}_D(k) \equiv [\eta_{0,D}(k), \eta_{1,D}(k), \dots, \eta_{N-1,D}(k)]^T$ is the noise vector comprised of the critically decimated subband noise

$$\eta_{i,D}(k) = \eta_i(kN) = \sum_{l=0}^{L-1} h_i(l)\eta(kN - l). \quad (7.9)$$

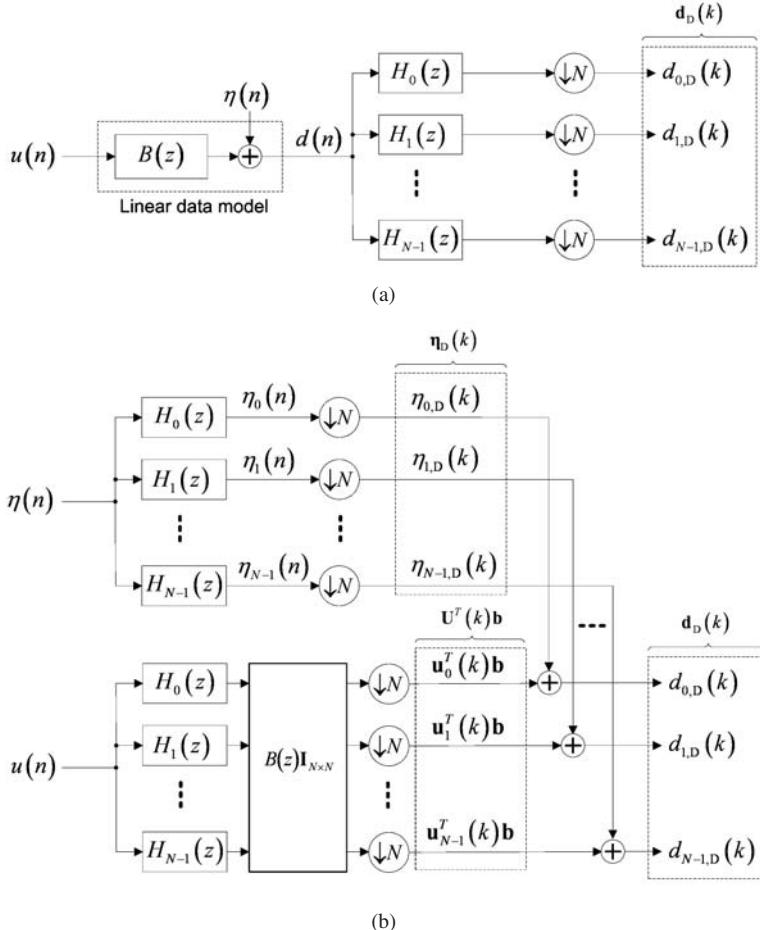


Figure 7.1 Analysis of the linear data model: (a) the desired response generated by a linear data model, which is decomposed into subbands and decimated, and (b) an equivalent structure obtained by virtue of linearity of the filter and the decimator

Using Equation (7.8) in Equation (7.2), the subband error signal vector $\mathbf{e}_D(k)$ can be expressed as

$$\begin{aligned}\mathbf{e}_D(k) &= [\mathbf{U}^T(k)\mathbf{b} + \boldsymbol{\eta}_D(k)] - \mathbf{U}^T(k)\mathbf{w}(k) \\ &= \mathbf{U}^T(k)[\mathbf{b} - \mathbf{w}(k)] + \boldsymbol{\eta}_D(k) \\ &= \mathbf{U}^T(k)\boldsymbol{\varepsilon}(k) + \boldsymbol{\eta}_D(k),\end{aligned}\quad (7.10)$$

where the weight-error vector, $\boldsymbol{\varepsilon}(k) \equiv \mathbf{b} - \mathbf{w}(k)$, measures how far the weight vector $\mathbf{w}(k)$ is from the model vector \mathbf{b} to be identified. Equation (7.10) shows that the error vector $\mathbf{e}_D(k)$ consists of two components: the modeling error $\mathbf{U}^T(k)\boldsymbol{\varepsilon}(k)$ and the measurement

noise $\eta_D(k)$. The vector $\mathbf{U}^T(k)\mathbf{e}(k)$ is the undisturbed estimation error that measures how well the adaptive filter output $\mathbf{U}^T(k)\mathbf{w}(k)$ approximates $\mathbf{U}^T(k)\mathbf{b}$, which is the undisturbed portion of $\mathbf{d}_D(k)$.

7.1.3 Paraunitary filter banks

The paraunitary condition for filter banks is defined in the transform domain by Equation (2.25). Imposing the paraunitary condition on the filter banks that partition $u(n)$ and $d(n)$ leads to the following two facts:

- (i) the filter banks are power complementary,
- (ii) the noise vector $\eta_D(k)$ in Equation (7.8) at any time instant k is uncorrelated with all the weight estimates $\{\mathbf{w}(k), \mathbf{w}(k-1), \mathbf{w}(k-2), \dots\}$.

The first fact is well explained in the literature; for examples see References [15, p. 124] and [16, p. 296]. The second fact greatly simplifies the mathematical analysis. A detailed discussion of these two facts will be presented in this section.

7.1.3.1 Paraunitary, lossless and power complementary

An analysis filter bank can be represented by matrix notation as defined in Equations (2.13) and (2.18), which is reproduced here for convenience:

$$\begin{aligned}\mathbf{h}(z) &\equiv [H_0(z), H_1(z), \dots, H_{N-1}(z)]^T \\ &= \mathbf{E}(z^N) \mathbf{e}(z),\end{aligned}\tag{7.11}$$

where $\mathbf{E}(z)$ is the polyphase-component matrix of the filter bank and $\mathbf{e}(z)$ is the delay chain defined by Equation (2.20). The filter bank $\mathbf{h}(z)$ is paraunitary if the following condition is satisfied:

$$\tilde{\mathbf{E}}(z)\mathbf{E}(z) = \mathbf{I},\tag{7.12}$$

where $\tilde{\mathbf{E}}(z)$ is the paraconjugate of $\mathbf{E}(z)$.

If the filter bank is causal and stable, in addition to paraunitary, it is also a lossless system [16, p. 288]. For a causal FIR filter, the terms paraunitary and lossless are interchangeable. The relation of the paraunitary condition with the perfect-reconstruction analysis–synthesis systems is explained in Section 2.3. In addition to the perfect-reconstruction property, the paraunitary condition also leads to a power-complementary filter bank.

Using Equation (7.11) in Equation (2.11) leads to

$$\begin{aligned}\mathbf{H}_m(z) &= \left[\mathbf{E}(z^N) \mathbf{e}(z), \mathbf{E}(z^N W_N^N) \mathbf{e}(z W_N), \dots, \right. \\ &\quad \left. \mathbf{E}(z^N W_N^{N(N-1)}) \mathbf{e}(z W_N^{N-1}) \right]^T.\end{aligned}\tag{7.13}$$

Because

$$(W_N^l)^N = (e^{-j2\pi l/N})^N = 1, \text{ for } l = 0, 1, \dots, N-1,$$

Equation (7.13) can be further simplified to

$$\mathbf{H}_m(z) = \mathbf{E}(z^N) \left[\mathbf{e}(z), \mathbf{e}(zW_N), \dots, \mathbf{e}(zW_N^{N-1}) \right]^T. \quad (7.14)$$

Let $\Delta(z) \equiv \text{diag}[1, z^{-1}, \dots, z^{-N+1}]$ denote a diagonal delay matrix. From Equation (2.20), it follows that

$$\mathbf{e}(zW_N^{-l}) = \Delta(z) [1, W_N^{-l}, \dots, W^{-l(N-1)}]^T. \quad (7.15)$$

Substituting Equation (7.15) into Equation (7.14) and taking the transposition leads to

$$\mathbf{H}_m(z) = [\mathbf{E}(z^N) \Delta(z) \mathbf{D}^*]^T = \mathbf{D}^H \Delta(z) \mathbf{E}^T(z^N), \quad (7.16)$$

where $\mathbf{D} = [a_{mn}]$ is the DFT matrix with elements $a_{mn} = e^{-j2\pi mn/N}$ and the superscript H denotes Hermitian transposition. Computing the product of $\mathbf{H}_m(z)$ and its paraconjugate,

$$\tilde{\mathbf{H}}_m(z) = \tilde{\mathbf{E}}^T(z^N) \tilde{\Delta}(z) \mathbf{D}, \quad (7.17)$$

and using the facts that $\tilde{\mathbf{E}}(z^N) \mathbf{E}(z^N) = \mathbf{I}$, $\mathbf{D}^H \mathbf{D} = N\mathbf{I}$ and $\Delta(z) \tilde{\Delta}(z) = \mathbf{I}$, we obtain

$$\begin{aligned} \mathbf{H}_m(z) \tilde{\mathbf{H}}_m(z) &= \mathbf{D}^H \Delta(z) [\tilde{\mathbf{E}}(z^N) \mathbf{E}(z^N)]^T \tilde{\Delta}(z) \mathbf{D} \\ &= N\mathbf{I}. \end{aligned} \quad (7.18)$$

This equation shows that the first diagonal element of $\mathbf{H}_m(z) \tilde{\mathbf{H}}_m(z)$ is

$$\sum_{i=0}^{N-1} H_i(z) \tilde{H}_i(z) = N. \quad (7.19)$$

Evaluating Equation (7.19) along the unit circle, we obtain

$$\begin{aligned} \sum_{i=0}^{N-1} [H_i(e^{j\omega})] [H_i(e^{j\omega})]^* &= N \\ \frac{1}{N} \sum_{i=0}^{N-1} |H_i(e^{j\omega})|^2 &= 1. \end{aligned} \quad (7.20)$$

This equation indicates that the paraunitary filter bank $\mathbf{h}(z)$ is power complementary.

7.1.3.2 Uncorrelated noise vectors

Define the cross-correlation function between the impulse responses of the analysis filters, $h_i(n)$ and $h_p(n)$ for $i, p = 0, 1, \dots, N - 1$, as follows:

$$q_{ip}(l) \equiv \sum_{n=0}^{L-1} h_i(n)h_p(n-l) = h_i(l) * h_p(-l), \quad (7.21)$$

where L is the length of the analysis filters and $*$ denotes linear convolution. The paraunitary condition [15, 17] ensures that the N -fold decimated version of $q_{ip}(l)$ appears in the following form:

$$q_{ip}(lN) = \delta(l)\delta(i-p). \quad (7.22)$$

Equations (7.21) and (7.22) indicate that the cross-correlation sequences $q_{ip}(l)$ of a paraunitary filter bank are characterized by periodic zero-crossings separated by N sampling periods. Notice that $q_{ip}(l)$ is a noncausal sequence of length $2L - 1$ as we assume that the analysis filters are causal FIR filters of length L .

Recall that the disturbance $\eta(n)$ in the linear data model (7.7) is assumed to be an uncorrelated white noise such that

$$E\{\eta(n)\eta(n-l)\} = \sigma_\eta^2\delta(l), \quad (7.23)$$

where σ_η^2 denotes the variance of $\eta(n)$. Applying the correlation-domain formulation (see Section 3.1) on the upper branch of Figure 7.1(b), the correlation function $\gamma_{ip}(l) \equiv E\{\eta_i(n)\eta_j(n-l)\}$ between the subband noises $\eta_i(n)$, for $i = 0, 1, \dots, N - 1$, can be expressed as

$$\gamma_{ip}(l) = q_{ip}(l) * [\sigma_\eta^2\delta(l)] = \sigma_\eta^2 q_{ip}(l). \quad (7.24)$$

Let $\eta_{i,D}(k) = \eta_i(kN)$ be the critically decimated subband noises. According to Section 3.1.1, the cross-correlation function $\gamma_{ip,D}(l) \equiv E\{\eta_{i,D}(k)\eta_{j,D}(k-l)\}$ between the critically decimated subband noises is the N -fold decimated version of $\gamma_{ip}(l)$ in Equation (7.24), expressed as

$$\gamma_{ip,D}(l) = \gamma_{ip}(lN) = \sigma_\eta^2 q_{ip}(lN). \quad (7.25)$$

Using Equation (7.22) in Equation (7.25), we obtain

$$E\{\eta_{i,D}(k)\eta_{j,D}(k-l)\} = \begin{cases} \sigma_\eta^2\delta(l), & \text{for } i = j, \\ 0, & \text{for } i \neq j, \end{cases} \quad (7.26)$$

which implies that the critically decimated subband noises, $\eta_{i,D}(k)$ for $i = 0, 1, \dots, N - 1$, are white and uncorrelated to each other. It should be emphasized that the subband noises, $\eta_i(n)$ for $i = 0, 1, \dots, N - 1$, before critical decimation are generally colored and correlated to each other. As can be seen from Equations (7.21) and (7.24), the correlation function $\gamma_{ip}(l)$ is characterized by the analysis filters $h_i(n)$ and $h_p(n)$.

Using the result given in Equation (7.26), fact (ii) as stated in Section 7.1.3 can be observed from the MSAF algorithm. Substituting Equation (7.10) into Equation (7.1) and setting $k = k - 1$, the current weight estimate $\mathbf{w}(k)$ can be expressed as

$$\mathbf{w}(k) = \mathbf{w}(k - 1) + \mu \mathbf{U}(k - 1) \boldsymbol{\Lambda}^{-1}(k) [\mathbf{U}^T(k - 1) \mathbf{e}(k - 1) + \boldsymbol{\eta}_D(k - 1)]. \quad (7.27)$$

Clearly, $\mathbf{w}(k)$ is dependent on the past noise vectors $\{\boldsymbol{\eta}_D(k - 1), \boldsymbol{\eta}_D(k - 2), \dots\}$ and the past input matrices $\{\mathbf{U}(k - 1), \mathbf{U}(k - 2), \dots\}$. However, the current noise vector $\boldsymbol{\eta}_D(k)$ is uncorrelated with the past noises, as indicated by Equation (7.26). Furthermore, $\boldsymbol{\eta}_D(k)$ is also assumed to be statistically independent (see Section 7.1.2) of the input matrices $\mathbf{U}(k)$ for all k . Therefore, $\boldsymbol{\eta}_D(k)$ is uncorrelated with the current and all the previous weight vectors $\{\mathbf{w}(k), \mathbf{w}(k - 1), \mathbf{w}(k - 2), \dots\}$.

7.2 Multiband MSE function

This section introduces the multiband MSE function as a performance measure for the MSAF algorithm. By means of the Wiener filter theory, the multiband MSE function can be shown to be equivalent to the classical fullband MSE function. Hence, use of the multiband MSE function allows a fair comparison of the MSAF algorithm with other adaptive algorithms that are measured by the MSE function.

7.2.1 MSE functions

The multiband MSE function is defined as the average of the mean-squared values of the subband estimation errors expressed as

$$J_M \equiv \frac{1}{N} \sum_{i=0}^{N-1} E \{e_{i,D}^2(k)\} = \frac{1}{N} E \{\|\mathbf{e}_D(k)\|^2\}. \quad (7.28)$$

Substituting Equation (7.2) into Equation (7.28), the multiband MSE function J_M can be written in the following expanded form:

$$\begin{aligned} J_M &= \frac{1}{N} E \left\{ [\mathbf{d}_D(k) - \mathbf{U}^T(k) \mathbf{w}]^T [\mathbf{d}_D(k) - \mathbf{U}^T(k) \mathbf{w}] \right\} \\ &= \frac{1}{N} E \{ \|\mathbf{d}_D(k)\|^2 - 2 [\mathbf{U}(k) \mathbf{d}_D(k)]^T \mathbf{w} + \mathbf{w}^T \mathbf{U}(k) \mathbf{U}^T(k) \mathbf{w} \} \\ &= \frac{1}{N} [E \{\|\mathbf{d}_D(k)\|^2\} - 2 E \{\mathbf{U}(k) \mathbf{d}_D(k)\}^T \mathbf{w} + \mathbf{w}^T E \{\mathbf{U}(k) \mathbf{U}^T(k)\} \mathbf{w}]. \end{aligned} \quad (7.29)$$

The iteration index k has been dropped from the vector $\mathbf{w}(k)$ for the context where the MSE function is treated as an error-performance surface depending on the tap weights. Assuming that the filter banks used for the input signal $u(n)$ and desired response $d(n)$ are identical and power complementary, and using the similar procedure described in Section 6.5.2, the second-order moments in Equation (7.29) can be shown to be equivalent to the second-order moments characterizing the classical MSE function expressed as

$$J = E \{e^2(n)\} = \sigma_d^2 - 2 \mathbf{p}^T \mathbf{w} + \mathbf{w}^T \mathbf{R} \mathbf{w}, \quad (7.30)$$

where

$$\begin{aligned}\sigma_d^2 &\equiv E \{d^2(n)\} = \frac{1}{N} E \{\|\mathbf{d}_D(k)\|^2\}, \\ \mathbf{p} &\equiv E \{\mathbf{u}(n) d(n)\} = \frac{1}{N} E \{\mathbf{U}(k) \mathbf{d}_D(k)\} \text{ and} \\ \mathbf{R} &\equiv E \{\mathbf{u}(n) \mathbf{u}^T(n)\} = \frac{1}{N} E \{\mathbf{U}(k) \mathbf{U}^T(k)\}\end{aligned}\quad (7.31)$$

are the second-order moments pertaining to the MSE functions J and J_M .

A power-complementary filter bank preserves the second-order moments of its input signal. By virtue of the power-complementary property, the multiband MSE function J_M is equivalent to the classical MSE function J defined in Equation (1.6). Both the MSE functions J and J_M are equivalent in the sense that they are quadratic functions of the weight vector \mathbf{w} that describes an identical error-performance surface characterized by the second-order moments $\{\sigma_d^2, \mathbf{p}, \mathbf{R}\}$ of Equation (7.31). In that sense, minimizing the MSE functions defined in Equations (7.28) and (7.30) with any iterative optimization methods would lead to the same optimal Wiener solution $\mathbf{w}_o = \mathbf{R}^{-1}\mathbf{p}$, which is located at the bottom (minimum) of the error-performance surface.

7.2.2 Excess MSE

The linear data model defined in Equation (7.7), or equivalently in Equation (7.8), describes a stationary environment for generating the desired response. Substituting Equation (7.10) into Equation (7.28), the multiband MSE function under the linear-data-model assumption can be expressed as

$$\begin{aligned}J_M(k) &= \frac{1}{N} E \{[\mathbf{e}_a(k) + \boldsymbol{\eta}_D(k)]^T [\mathbf{e}_a(k) + \boldsymbol{\eta}_D(k)]\} \\ &= \frac{1}{N} E \{\|\boldsymbol{\eta}_D(k)\|^2 + 2\boldsymbol{\eta}_D^T(k)\mathbf{e}_a(k) + \|\mathbf{e}_a(k)\|^2\} \\ &= \frac{1}{N} E \{\|\boldsymbol{\eta}_D(k)\|^2\} + 2 \left[\frac{1}{N} E \{\boldsymbol{\eta}_D^T(k)\mathbf{e}_a(k)\} \right] + \frac{1}{N} E \{\|\mathbf{e}_a(k)\|^2\},\end{aligned}\quad (7.32)$$

where $\mathbf{e}_a(k) \equiv [e_{0,a}(k), e_{1,a}(k), \dots, e_{N-1,a}(k)]^T = \mathbf{U}^T(k)\boldsymbol{\varepsilon}(k)$ is the undisturbed estimation error. The cross term $E\{\boldsymbol{\eta}_D^T(k)\mathbf{e}_a(k)\}$ in Equation (7.32) can be eliminated by exploiting the fact that the noise vector $\boldsymbol{\eta}_D(k)$ is uncorrelated with the current weight vector $\mathbf{w}(k)$ due to the paraunitary condition imposed on the filter banks (as explained earlier in Section 7.1.3). Thus the multiband MSE function reduces to

$$J_M(k) = \sigma_\eta^2 + \frac{1}{N} E \{\|\mathbf{e}_a(k)\|^2\}, \quad (7.33)$$

where

$$\sigma_\eta^2 \equiv E \{\eta^2(n)\} = \frac{1}{N} E \{\|\boldsymbol{\eta}_D(k)\|^2\} = \frac{1}{N} \sum_{i=0}^{N-1} E \{\eta_{i,D}(k)\} \quad (7.34)$$

is the variance of the additive white noise $\eta(n)$ by considering that the analysis filter bank is power complementary (due to the paraunitary condition, as explained earlier in Section 7.1.3). Unlike the error-performance surface defined in Equation (7.28), we include the iteration index k in Equations (7.32) and (7.33) in order to consider the evolution of the estimation error during the adaptation process.

Perfect adaptation is achieved when $\mathbf{w}(k)$ converges to the optimum weight vector \mathbf{w}_o , which results in the minimum MSE J_{\min} , as defined in Equation (1.9). Under this condition, the weight-error vector $\mathbf{e}(k)$ and the undisturbed estimation error $\mathbf{e}_a(k) = \mathbf{U}^T \mathbf{e}(k)$ in Equation (7.33) are essentially zero vectors. By substituting the perfect-adaptation condition $\mathbf{e}_a(k) = \mathbf{0}$ into Equation (7.33), we show that the minimum MSE under the linear data model is determined by the variance σ_η^2 of the disturbance $\eta(n)$. Nevertheless, due to the stochastic nature of adaptive algorithms, the computed weight vector will never be fixed at the bottom of the error surface $\{J_{\min}, \mathbf{w}_o\}$. Instead, it exhibits a random motion around $\{J_{\min}, \mathbf{w}_o\}$, thus increasing the MSE and producing the so-called steady-state excess MSE J_{ex} , which can be written as

$$J_{\text{ex}} \equiv J(\infty) - J_{\min} = J_M(\infty) - \sigma_\eta^2 = \frac{1}{N} E \{ \| \mathbf{e}_a(\infty) \|^2 \}, \quad (7.35)$$

where $J(\infty)$ is the average MSE in the steady state. In Equation (7.35), it is assumed that the multiband MSE function J_M is identical to the classical MSE function J by virtue of the paraunitary condition of the filter banks. In Section 7.4, we will use the excess MSE J_{ex} to evaluate the steady-state performance of the MSAF algorithm.

7.3 Mean analysis

In this section, we begin the mean analysis of MSAF recursion using projection interpretation. The mean behavior of the MSAF algorithm is then observed from the mean characteristics of the projection matrix.

7.3.1 Projection interpretation

The following analysis assumes that the desired response $d(n)$ is derived from a noiseless linear data model, where the disturbance $\eta(n)$ in Equation (7.7) (and in Figure 7.1) is negligible. Furthermore, we also assume that the regularization parameter α in Equation (7.3) is zero.

Substituting Equations (7.1) and (7.2) into Equation (7.8), the MSAF algorithm can be expressed as

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) + \mu \mathbf{U}(k) \Lambda^{-1}(k) [\mathbf{U}^T(k) \mathbf{b} - \mathbf{U}^T(k) \mathbf{w}(k)] \\ &= \mathbf{w}(k) - \mu \mathbf{U}(k) \Lambda^{-1}(k) \mathbf{U}^T(k) \mathbf{w}(k) + \mu \mathbf{U}(k) \Lambda^{-1}(k) \mathbf{U}^T(k) \mathbf{b} \\ &= [\mathbf{I} - \mu \mathbf{P}(k)] \mathbf{w}(k) + \mu \mathbf{P}(k) \mathbf{b}, \end{aligned} \quad (7.36)$$

where $\mathbf{P}(k)$ is a projection matrix [18] of the form

$$\mathbf{P}(k) \equiv \mathbf{U}(k) \Lambda^{-1}(k) \mathbf{U}^T(k) = \sum_{i=0}^{N-1} \frac{\mathbf{u}_i(k) \mathbf{u}_i^T(k)}{\mathbf{u}_i^T(k) \mathbf{u}_i(k)}. \quad (7.37)$$

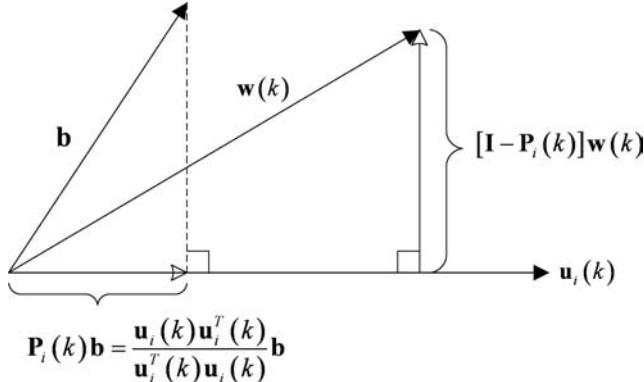


Figure 7.2 Projection interpretation of the MSAF recursion. The updated vector $\mathbf{w}(k+1)$ is the sum of projections on to two perpendicular subspaces

Clearly, the projection matrix $\mathbf{P}(k)$ is uniquely determined by the subband data matrix $\mathbf{U}(k)$, which consists of N subband regressors $\mathbf{u}_i(k)$ defined in Equation (7.5). The subband vectors are orthogonal if the diagonal assumption (see Section 6.5.1) is valid. Therefore, the column space of $\mathbf{U}(k)$ is spanned by an orthogonal basis consisting of $\mathbf{u}_0(k), \mathbf{u}_1(k), \dots, \mathbf{u}_{N-1}(k)$.

The weight recursion given in Equation (7.36) can be interpreted as an iterative projection operation. At each iteration, the projection matrix $\mathbf{P}(k)$ projects the true system impulse response \mathbf{b} on to the column space of $\mathbf{U}(k)$, while the old estimate $\mathbf{w}(k)$ is projected on to the subspace that is perpendicular to the column space of $\mathbf{U}(k)$. The sum of these two projections results in the updated weight vector $\mathbf{w}(k+1)$. The iterative projection operation given in Equation (7.36) is illustrated in Figure 7.2. Since we assume that subband regressors $\mathbf{u}_0(k), \mathbf{u}_1(k), \dots, \mathbf{u}_{N-1}(k)$ are orthogonal (by virtue of the diagonal assumption), the projection matrix $\mathbf{P}(k)$ can be decomposed into N projection matrices $\mathbf{P}_i(k) = \mathbf{u}_i(k)\mathbf{u}_i^T(k)/\mathbf{u}_i^T(k)\mathbf{u}_i(k)$ of rank one. Projection on to the column space of $\mathbf{U}(k)$ is then taken as the sum of N projections on to the vectors $\mathbf{u}_i(k)$, for $i = 0, 1, \dots, N-1$. A similar concept applies to the complement projection.

The complement projection $[\mathbf{I} - \mu\mathbf{P}(k)]\mathbf{w}(k)$ in Equation (7.36) can be interpreted as a way to discard the old information $\mathbf{P}(k)\mathbf{w}(k)$ (i.e. the component of the old estimate $\mathbf{w}(k)$ that lies in the column space of $\mathbf{U}(k)$) from $\mathbf{w}(k)$. On the other hand, the projection $\mathbf{P}(k)\mathbf{b}$ serves as the new information to replace the old information $\mathbf{P}(k)\mathbf{w}(k)$ that has been discarded from $\mathbf{w}(k)$. The amount of information flow from one iteration to the next is regulated by the step size μ , for which a stability bound will be established in the next section. For the case where $N = M$, $\mathbf{P}(k)\mathbf{w}(k) = \mathbf{w}(k)$ and $\mathbf{P}(k)\mathbf{b} = \mathbf{b}$; this is because the projection of a vector of length M on to the column space of the full rank $M \times M$ matrix $\mathbf{U}(k)$ is equal to the vector itself.

If we further assume that $\mu = 1$, the adaptive weight vector $\mathbf{w}(k)$ converges in one step as

$$\begin{aligned}\mathbf{w}(k+1) &= \mathbf{w}(k) - \mathbf{P}(k)\mathbf{w}(k) + \mathbf{P}(k)\mathbf{b} \\ &= \mathbf{w}(k) - \mathbf{w}(k) + \mathbf{b} \\ &= \mathbf{b}.\end{aligned}\tag{7.38}$$

Convergence in one step, though attractive, is not practical since the noise $\eta_D(k)$ exists in real applications. Furthermore, for $N = M$, the filter banks would introduce an unacceptably long delay that is generally greater than that introduced by the adaptive filter.

7.3.2 Mean behavior

The effectiveness of the MSAF algorithm in dealing with colored signals can be observed from the mean behavior of the projection matrix $\mathbf{P}(k)$. Subtracting Equation (7.36) from \mathbf{b} , rearranging terms, taking the expectation of both sides and making the independence assumption [13, 14, 19, 20] (i.e. the weight vector $\mathbf{w}(k)$ is statistically independent of the subband regressors $\mathbf{u}_i(k)$), the mean behavior of the weight-error vector $\mathbf{\epsilon}(k) \equiv \mathbf{b} - \mathbf{w}(k)$ can be described as

$$E\{\mathbf{\epsilon}(k+1)\} = [I - \mu \mathbf{R}_w] E\{\mathbf{\epsilon}(k)\}, \quad (7.39)$$

where

$$\mathbf{R}_w \equiv E\{\mathbf{P}(k)\} = \sum_{i=0}^{N-1} E\left\{ \frac{\mathbf{u}_i(k)\mathbf{u}_i^T(k)}{\mathbf{u}_i^T(k)\mathbf{u}_i(k)} \right\} \quad (7.40)$$

is the mean of the projection matrix $\mathbf{P}(k)$. Clearly, Equation (7.39) for the MSAF algorithm has the same mathematical form as the LMS algorithm [13, 14, 19, 20], except for the correlation matrices. Equation (7.39) suggests that the convergence of the MSAF algorithm in mean is driven by M natural modes, which are characterized by the eigenvalues of \mathbf{R}_w .

Assuming that M is large, the fluctuations of subband energy $\|\mathbf{u}_i(k)\|^2$ from one iteration to the next will be small enough to justify the following approximation:

$$\begin{aligned} E\left\{ \frac{\mathbf{u}_i(k)\mathbf{u}_i^T(k)}{\|\mathbf{u}_i(k)\|^2} \right\} &\approx \frac{E\{\mathbf{u}_i(k)\mathbf{u}_i^T(k)\}}{E\{\|\mathbf{u}_i(k)\|^2\}} \\ &= \frac{E\{\mathbf{u}_i(k)\mathbf{u}_i^T(k)\}}{M\gamma_{ii}(0)}, \end{aligned} \quad (7.41)$$

where $\gamma_{ii}(0)$ is the variance of the subband signal $E\{u_i^2(n)\}$. Defining $\mathbf{R}_i = E\{\mathbf{u}_i(k)\mathbf{u}_i^T(k)\}$ as the subband autocorrelation matrix, the mean of the projection matrix can be seen as a weighted sum of N subband autocorrelation matrices, expressed as

$$\mathbf{R}_w \approx \sum_{i=0}^{N-1} \lambda_i^{-1} \mathbf{R}_i, \quad (7.42)$$

where $\lambda_i^{-1} = 1/M\gamma_{ii}(0)$ are the weighting factors.

From Equations (7.41) and (7.42), it is obvious that the weighted autocorrelation matrix \mathbf{R}_w is determined by a weighted autocorrelation function $\gamma_\xi(l)$ of the form

$$\gamma_\xi(l) = \left[\sum_{i=0}^{N-1} \lambda_i^{-1} h_i(l) * h_i(-l) \right] * \gamma_{uu}(l) \quad (7.43)$$

or in the frequency domain by an equalized spectrum

$$\Gamma_{\xi}(e^{j\omega}) = \left[\sum_{i=0}^{N-1} \lambda_i^{-1} |H_i(e^{j\omega})|^2 \right] \Gamma_{uu}(e^{j\omega}). \quad (7.44)$$

In the above equations, $h_i(n)$ and $|H_i(e^{j\omega})|$ are the impulse and magnitude responses of the i th analysis filter, respectively; $\gamma_{uu}(l)$ and $\Gamma_{uu}(e^{j\omega})$ are the autocorrelation function and power spectrum of signal $u(n)$, respectively. Notice that the results given in Equations (7.43) and (7.44) are consistent with Equation (6.50) derived in Section 6.4.2, where we analyzed the inherent decorrelation properties of the MSAF algorithm.

The matrix \mathbf{R}_w is completely determined by $\gamma_{\xi}(l)$. Therefore, the condition number of the weighted correlation matrix \mathbf{R}_w is bounded by the spectral dynamic range of the equalized spectrum $\Gamma_{\xi}(e^{j\omega})$ expressed as

$$\kappa(\mathbf{R}_w) \leq \frac{\max_{\omega} \Gamma_{\xi}(e^{j\omega})}{\min_{\omega} \Gamma_{\xi}(e^{j\omega})}. \quad (7.45)$$

The partitioning and normalization features shown in Equation (7.44) flatten the input spectrum $\Gamma_{uu}(e^{j\omega})$, resulting in the equalized $\Gamma_{\xi}(e^{j\omega})$ with a reduced spectral dynamic range. The equalized spectrum $\Gamma_{\xi}(e^{j\omega})$ decreases the eigenvalue spread of the autocorrelation matrix \mathbf{R}_w , thereby reducing the sensitivity of the MSAF algorithm to the colored input signals.

7.4 Mean-square analysis

In this section, stability bounds for the step size and an expression for the excess MSE are derived based on the energy conservation relation [1–3, 5–7].

7.4.1 Energy conservation relation

The first step in establishing an energy conservation relation for the MSAF algorithm is to represent the weight recursion in terms of the weight-error vector, $\mathbf{e}(k) \equiv \mathbf{w}_o - \mathbf{w}(k)$. Subtracting Equation (7.1) from \mathbf{w}_o , we get

$$\mathbf{e}(k+1) = \mathbf{e}(k) - \mu \mathbf{U}(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k). \quad (7.46)$$

Let $\mathbf{e}_a(k) = \mathbf{U}^T(k) \mathbf{e}(k)$ and $\mathbf{e}_p(k) = \mathbf{U}^T(k) \mathbf{e}(k+1)$ denote an a priori error and an a posteriori error, respectively. Multiplying both sides of Equation (7.46) by $\mathbf{U}^T(k)$ from the left, $\mathbf{e}_a(k)$ and $\mathbf{e}_p(k)$ can be related as follows:

$$\begin{aligned} \mathbf{U}^T(k) \mathbf{e}(k+1) &= \mathbf{U}^T(k) \mathbf{e}(k) - \mu \mathbf{U}^T(k) \mathbf{U}(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k), \\ \mathbf{e}_p(k) &= \mathbf{e}_a(k) - \mu [\mathbf{U}^T(k) \mathbf{U}(k)] \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k). \end{aligned} \quad (7.47)$$

The a priori error $\mathbf{e}_a(k)$ represents the undisturbed estimation error produced by the current weight vector, whereas the a posteriori error $\mathbf{e}_p(k)$ indicates the undisturbed estimation error produced by the updated weight vector. Assuming that the subband signal matrix $\mathbf{U}(k)$ has full column rank, the matrix $\mathbf{U}^T(k)\mathbf{U}(k)$ is invertible, and Equation (7.47) can be solved for $\mu\Lambda^{-1}(k)\mathbf{e}_D(k)$ as follows:

$$\begin{aligned}\mu [\mathbf{U}^T(k)\mathbf{U}(k)] \Lambda^{-1}(k) \mathbf{e}_D(k) &= \mathbf{e}_a(k) - \mathbf{e}_p(k), \\ \mu \Lambda^{-1}(k) \mathbf{e}_D(k) &= [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} [\mathbf{e}_a(k) - \mathbf{e}_p(k)].\end{aligned}$$

Substituting the result into Equation (7.46) and rearranging terms, the MSAF algorithm can be written in terms of the weight-error vectors and the undisturbed estimation errors at two successive iterations in the following form:

$$\begin{aligned}\mathbf{\varepsilon}(k+1) + \mathbf{U}(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_a(k) \\ = \mathbf{\varepsilon}(k) + \mathbf{U}(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_p(k).\end{aligned}\quad (7.48)$$

Evaluating the energy by taking the squared Euclidean norm on the left-hand side of Equation (7.48), we obtain

$$\begin{aligned}\|\mathbf{\varepsilon}(k+1) + \mathbf{U}(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_a(k)\|^2 \\ = \left\{ \begin{array}{l} \|\mathbf{\varepsilon}(k+1)\|^2 + 2\mathbf{e}_p^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_a(k) \\ + \mathbf{e}_a^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_a(k) \end{array} \right\}.\end{aligned}\quad (7.49)$$

Similarly, the expression for energy on the right-hand side of Equation (7.48) is given by

$$\begin{aligned}\|\mathbf{\varepsilon}(k) + \mathbf{U}(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_p(k)\|^2 \\ = \left\{ \begin{array}{l} \|\mathbf{\varepsilon}(k)\|^2 + 2\mathbf{e}_a^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_p(k) \\ + \mathbf{e}_p^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_p(k) \end{array} \right\}.\end{aligned}\quad (7.50)$$

By equating the energy on both sides of Equation (7.48), we obtain the following energy conservation relation:

$$\begin{aligned}\|\mathbf{\varepsilon}(k+1)\|^2 + \mathbf{e}_a^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_a(k) \\ = \|\mathbf{\varepsilon}(k)\|^2 + \mathbf{e}_p^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_p(k).\end{aligned}\quad (7.51)$$

This energy conservation relation shows how the energy of the weight-error vectors at two successive iterations are related to the weighted energy of the a priori and a posteriori errors. It has been shown in References [3] and [5] to [7] that a similar energy conservation relation can be established for other adaptive algorithms, such as the LMS, NLMS, AP and RLS algorithms.

7.4.2 Variance relation

Define the mean-square deviation as

$$c(k) = E \{ \| \boldsymbol{\varepsilon}(k) \|^2 \}. \quad (7.52)$$

Taking the expectation of both sides of Equation (7.51), the energy relation can be translated into the variance relation in terms of $\{c(k), c(k+1)\}$ as follows:

$$\begin{aligned} c(k+1) &+ E \left\{ \mathbf{e}_a^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_a(k) \right\} \\ &= c(k) + E \left\{ \mathbf{e}_p^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_p(k) \right\}. \end{aligned} \quad (7.53)$$

Since $\mathbf{e}_p(k)$ is related to $\mathbf{e}_a(k)$ via Equation (7.47), the second term of the expression on the right-hand side of Equation (7.53) can be expanded into the following form:

$$\begin{aligned} &E \left\{ \mathbf{e}_p^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_p(k) \right\} \\ &= E \left\{ \mathbf{e}_a^T(k) [\mathbf{U}^T(k)\mathbf{U}(k)]^{-1} \mathbf{e}_a(k) \right\} \\ &\quad - 2\mu E \left\{ \mathbf{e}_a^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k) \right\} \\ &\quad + \mu^2 E \left\{ \mathbf{e}_D^T(k) \mathbf{\Lambda}^{-1}(k) [\mathbf{U}^T(k)\mathbf{U}(k)] \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k) \right\}. \end{aligned} \quad (7.54)$$

Using Equation (7.54) in Equation (7.53), the variance relation can be simplified to

$$\begin{aligned} c(k+1) &= c(k) - 2\mu E \left\{ \mathbf{e}_a^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k) \right\} \\ &\quad + \mu^2 E \left\{ \mathbf{e}_D^T(k) \mathbf{\Lambda}^{-1}(k) [\mathbf{U}^T(k)\mathbf{U}(k)] \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k) \right\}. \end{aligned} \quad (7.55)$$

Several aspects of the convergence of the MSAF algorithm can be analyzed by evaluating the mean-square values of the error quantities $\{\boldsymbol{\varepsilon}(k), \boldsymbol{\varepsilon}(k+1), \mathbf{e}_a(k), \mathbf{e}_D(k)\}$ that appear in the variance relation given in Equation (7.55). Note that no approximation has been made to establish the energy conservation relation in Equation (7.51) and the variance relation in Equation (7.55). Hence, reliable results for the mean-square performance can be obtained by using the variance relation.

7.4.3 Stability of the MSAF algorithm

Assuming that the diagonal assumption is valid (i.e. the off-diagonal elements of the matrix $\mathbf{U}^T(k)\mathbf{U}(k)$ are negligible) and the regularization parameter α has been set to zero such that $\mathbf{U}^T(k)\mathbf{U}(k) \approx \mathbf{\Lambda}(k)$, the variance relation of Equation (7.55) can be further simplified to

$$c(k+1) - c(k) \approx \mu^2 E \left\{ \mathbf{e}_D^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k) \right\} - 2\mu E \left\{ \mathbf{e}_a^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k) \right\}. \quad (7.56)$$

The underlying principle of the MSAF algorithm is to have the updated weight vector $\mathbf{w}(k+1)$ as close as possible to the previous weight vector $\mathbf{w}(k)$ while subject to a set of

subband constraints. For the algorithm to be stable, the mean-square deviation $c(k)$ must decrease monotonically with increasing k , i.e. $c(k+1) < c(k)$ from one iteration to the next. Equation (7.56) shows that the step size μ has to satisfy the following condition for $c(k+1)$ to be less than $c(k)$:

$$0 < \mu < 2 \left\{ \frac{E \{ \mathbf{e}_a^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_a(k) \}}{E \{ \mathbf{e}_D^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_D(k) \}} \right\}. \quad (7.57)$$

Consider the situation where the disturbance $\eta_D(k)$ in Equation (7.10) is negligible. Under the noiseless situation, it is obvious that the estimation error $\mathbf{e}_D(k)$ is equal to the undisturbed error $\mathbf{e}_a(k) = \mathbf{U}^T(k) \boldsymbol{\varepsilon}^T(k)$. Hence, in the absence of disturbance, the necessary and sufficient condition for the convergence in the mean-square sense is that the step size must satisfy the following double inequality:

$$0 < \mu < 2. \quad (7.58)$$

7.4.4 Steady-state excess MSE

The excess MSE J_{ex} defined in Equation (7.35) indicates the difference between the average MSE $J(\infty)$ produced by the adaptive algorithm operated in the steady state and the minimum MSE J_{min} pertaining to the optimum Wiener solution \mathbf{w}_o . The adaptive algorithm reaches a steady state if its mean-squared deviation $c(k)$ converges to a finite constant c as k increases to infinity, i.e. [3, 6, 7]

$$c(k+1) = c(k) = c < \infty, \text{ as } k \rightarrow \infty. \quad (7.59)$$

It should be emphasized that the step size μ has to be small enough, as bounded in Equation (7.57) or (7.58), in order to guarantee the convergence of the algorithm to the steady state.

The expression for the steady-state excess MSE J_{ex} is derived for the MSAF algorithm. For mathematical simplicity, we assume that the filter banks are paraunitary, the diagonal assumption is valid and the regularization parameter is set to zero. Furthermore, it is also common to assume that the desired response arises from the linear data model of Equation (7.8). Substituting Equation (7.10) into Equation (7.56) and exploiting the fact that the noise vector $\eta_D(k)$ is uncorrelated with the weight-error vector $\boldsymbol{\varepsilon}(k)$ due to the paraunitary filter banks, we obtain

$$\begin{aligned} c(k+1) - c(k) &\approx (\mu^2 - 2\mu) E \{ \mathbf{e}_a^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_a(k) \} \\ &\quad + \mu^2 E \{ \eta_D^T(k) \mathbf{\Lambda}^{-1}(k) \eta_D(k) \}. \end{aligned} \quad (7.60)$$

Introducing the steady-state condition (7.59) into (7.60), we obtain the steady-state relation between the subband undisturbed error signals $e_{i,a}(k)$ and the subband disturbances $\eta_{i,D}(k)$ as follows:

$$E \{ \mathbf{e}_a^T(k) \mathbf{\Lambda}^{-1}(k) \mathbf{e}_a(k) \} \approx \frac{\mu}{2-\mu} E \{ \eta_D^T(k) \mathbf{\Lambda}^{-1}(k) \eta_D(k) \}, \text{ as } k \rightarrow \infty. \quad (7.61)$$

By virtue of the diagonal assumption, the normalization matrix $\Lambda^{-1}(k)$ can be assumed as a diagonal matrix with elements $1/\|\mathbf{u}_i(k)\|^2$ for the case of the zero regularization parameter. Using this result in the relation (7.61), the steady-state variance relation can be equivalently written as

$$\sum_{i=0}^{N-1} E \left\{ \frac{e_{i,a}^2(k)}{\|\mathbf{u}_i(k)\|^2} \right\} \approx \frac{\mu}{2-\mu} \sum_{i=0}^{N-1} E \left\{ \frac{\eta_{i,D}^2(k)}{\|\mathbf{u}_i(k)\|^2} \right\}, \text{ as } k \rightarrow \infty. \quad (7.62)$$

If we further assume that the subband signals are uncorrelated such that the variance $E\{e_{i,a}^2(k)\}$ of the undisturbed error signal in the i th subband is only due to the variance $E\{\eta_{i,D}^2(k)\}$ of disturbance in the same subband, we may deduce that the i th terms of the summations on both sides of the relation (7.62) correspond to each other as follows:

$$E \left\{ \frac{e_{i,a}^2(k)}{\|\mathbf{u}_i(k)\|^2} \right\} \approx \frac{\mu}{2-\mu} E \left\{ \frac{\eta_{i,D}^2(k)}{\|\mathbf{u}_i(k)\|^2} \right\}, \text{ for } i = 0, 1, \dots, N-1, \text{ as } k \rightarrow \infty. \quad (7.63)$$

For a high-order filter where $M \gg 1$, the fluctuation in the subband energy $\|\mathbf{u}_i(k)\|^2$ from one iteration to the next can be assumed to be small enough, thereby justifying the assumptions

$$E \left\{ \frac{e_{i,a}^2(k)}{\|\mathbf{u}_i(k)\|^2} \right\} \approx \frac{E\{e_{i,a}^2(k)\}}{E\{\|\mathbf{u}_i(k)\|^2\}} \text{ and } E \left\{ \frac{\eta_{i,D}^2(k)}{\|\mathbf{u}_i(k)\|^2} \right\} \approx \frac{E\{\eta_{i,D}^2(k)\}}{E\{\|\mathbf{u}_i(k)\|^2\}}. \quad (7.64)$$

Inserting this approximation into the relation (7.63), we obtain

$$E \{e_{i,a}^2(k)\} \approx \frac{\mu}{2-\mu} E \{\eta_{i,D}^2(k)\}, \text{ for } i = 0, 1, \dots, N-1, \text{ as } k \rightarrow \infty. \quad (7.65)$$

As defined in Equation (7.35), the excess MSE J_{ex} in the steady state is given by the average of the variance of the undisturbed subband error signals. Using the relation (7.65) in Equation (7.35), the excess MSE for the MSAF algorithm can be expressed as

$$J_{\text{ex}} \approx \frac{\mu}{2-\mu} \left[\frac{1}{N} \sum_{i=0}^{N-1} E \{\eta_{i,D}^2(k)\} \right] = \frac{\mu \sigma_\eta^2}{2-\mu}, \quad (7.66)$$

where σ_η^2 denotes the variance of the additive white noise as defined in Equation (7.34). This expression holds as long as the frequency responses of the analysis filters do not overlap significantly and their stopband attenuation is sufficiently high. Otherwise, the excess MSE would generally be higher than that given in Equation (7.66) because the variance of the undisturbed error signal $E\{e_{i,a}^2(k)\}$ will be increased from cross-channel disturbances. Recall that the NLMS algorithm can be seen as a special case of the MSAF algorithm with $N = 1$. The expression for excess MSE given by Equation (7.66) is consistent with that of the NLMS algorithm reported in the literature.

From Equation (7.66), it is obvious that the excess MSE J_{ex} is independent of the number of subbands N . Hence, the convergence of the MSAF algorithm can be accelerated by increasing the number of subbands N while maintaining the same level of excess

MSE in the steady state with the same step size. The drawback is that additional delay is introduced into the signal path because higher orders of analysis filters are required to achieve sufficient stopband attenuation. Note that the signal-path delay can be eliminated with the delayless structures described in Chapter 6.

7.5 MATLAB examples

This section demonstrates and verifies the analytical results derived in previous sections using MATLAB simulations that are performed in the context of adaptive system identification. The desired response $d(n)$ is generated using the linear data model given in Equation (7.7). The impulse response of unknown system $B(z)$ (represented by vector \mathbf{b}) is of length $M = 1024$, as shown in Figure 4.19, and the variance of additive noise $\eta(n)$ is $\sigma_\eta^2 = 1 \times 10^{-4}$. Two types of input signals, white Gaussian noise and the AR(2) signal, are used in the simulations. The spectrum of the AR(2) signal is shown in Figure 6.14(a). The MSAF algorithm uses different numbers of subbands $N = 4, 8, 16$ and 32 , and the regularization parameter α is set to 0.0001 for all cases. The length of the adaptive weight vector $\mathbf{w}(k)$ is $M = 1024$ and the tap weights are initialized to zero in all the simulations.

Paraunitary cosine-modulated filter banks [15] with a length of $L = 8N$ are used for the analysis and synthesis filters since we impose the paraunitary condition on the analysis filters in order to simplify the mathematical analysis, as shown in Section 7.1.3. Nevertheless, pseudo-QMF cosine-modulated filter banks can also be used. It has been confirmed that the pseudo-QMF design results in an analysis filter bank that is almost paraunitary [21]. The complete listing of MATLAB programs can be found in the companion CD.

7.5.1 Mean of the projection matrix

The major motivation of using the MSAF algorithm is to improve the convergence of the NLMS algorithm under colored input signals. The effectiveness of the MSAF algorithm in dealing with colored signals can be seen from the mean behavior of the projection matrix $\mathbf{P}(k)$, i.e. the weighted autocorrelation matrix \mathbf{R}_w . Table 7.1 shows the condition numbers of \mathbf{R}_w for $N = 1, 4, 8, 16$ and 32 subbands with white and colored input signals. Recall that the special case of the MSAF algorithm with $N = 1$ corresponds to

Table 7.1 Condition numbers of the weighted autocorrelation matrix \mathbf{R}_w for both white and colored input signals

Number of subbands, N	Condition number, $\kappa(\mathbf{R}_w)$	
	White signal	Colored signal
1	1.471	414.997
4	1.488	68.139
8	1.480	19.518
16	1.488	6.372
32	1.486	2.298

the NLMS algorithm in which the convergence of the algorithm is determined by the normalized autocorrelation matrix $\mathbf{R}_w \approx \mathbf{R}/M\gamma_{uu}(0)$, where $\mathbf{R} \equiv E\{\mathbf{u}(n)\mathbf{u}^T(n)\}$ is the fullband autocorrelation matrix and $\gamma_{uu}(0) \equiv E\{u^2(n)\}$ is the variance of the input signal.

The condition numbers of the weighted autocorrelation matrices for the white input signal are listed in the second column of Table 7.1. These matrices for $N = 1, 4, 8, 16$ and 32 subbands are well conditioned, with their condition numbers approximately equal to one. This explains why the NLMS and MSAF algorithms exhibit identical learning curves (as shown in Figure 6.12) when the input signal is white. The convergence of the NLMS algorithm deteriorates for colored signals, where the condition number increases tremendously from 1.471 to 414.997 for $N = 1$. By introducing more constraints on the weight adaptation (i.e. by increasing the number of subbands, N), the condition number of the weighted autocorrelation matrix decreases accordingly, as can be observed from the third column of Table 7.1. A smaller eigenvalue spread gives a faster convergence rate. This fact can be observed from the learning curves shown in Figure 6.13. Table 7.1 also shows that the convergence of the algorithm becomes less sensitive to colored input signals, with an increased number of subbands. The analysis is illustrated in the MATLAB code given in Example 7.1.

Example 7.1 The condition number of the weighted autocorrelation matrix \mathbf{R}_w is completely determined by the weighted autocorrelation function γ_ξ given by Equation (7.43). In the following code, we first decompose the input signal u_n into N subbands using the analysis filter bank H . The weighted correlation sequence r_w is then computed by taking the weighted sum of the autocorrelation sequences r_i . Finally, we construct the matrix R_w from r_w by exploiting its Toeplitz structure and compute the condition numbers. A complete listing is given in M-file Example_7P1.m.

```

U = upfirdn(un,H); % Analysis filter bank
rw = 0; % Initialization
for i = 1:N
    % Subband autocorrelation
    ri = xcorr(U(:,i),M-1,'unbiased');
    % Normalization factor
    lambda(i) = M*var(U(:,i));
    % Weighted autocorrelation sequence
    rw = rw + ri/lambda(i);
end

c = flipud(rw(1:M)); r = rw(M:end);
Rw = toeplitz(c,r);
CondNumber = cond(Rw);

```

7.5.2 Stability bounds

The steady-state MSE curves of the MSAF algorithm with $N = 4, 8, 16$ and 32 subbands, plotted as a function of the step size μ , are shown in Figures 7.3 and 7.4 for the white

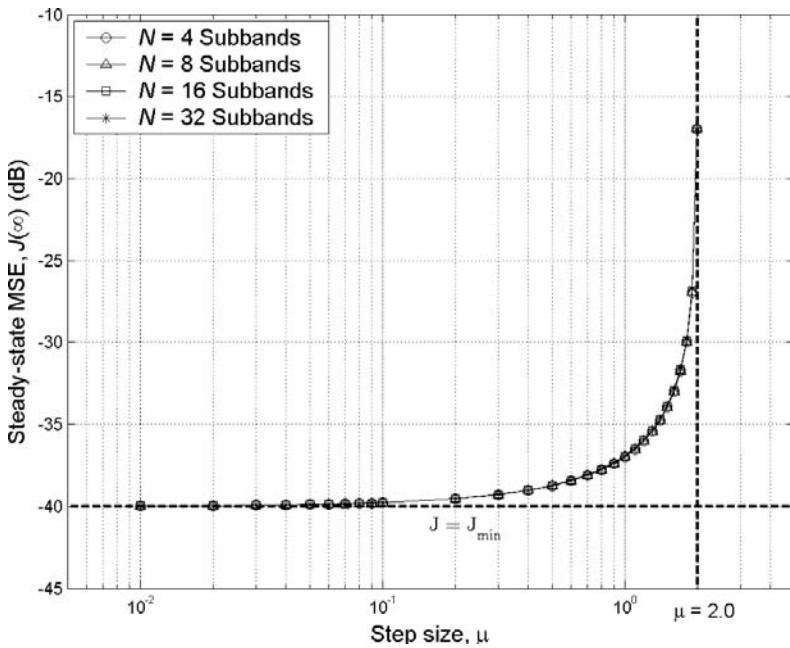


Figure 7.3 Steady-state MSE of the MSAF algorithm (for $N = 4, 8, 16$ and 32 subbands) under white noise excitation

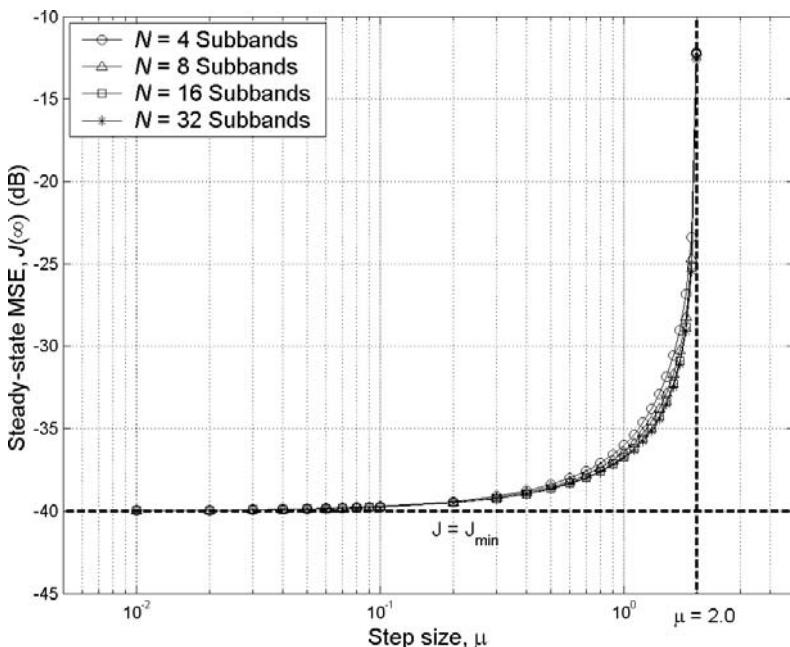


Figure 7.4 Steady-state MSE of the MSAF algorithm (for $N = 4, 8, 16$ and 32 subbands) under colored excitation (adapted from Reference [22])

noise and AR(2) excitation signals, respectively. For each step size μ that varies from 0.01 to 1.99, the ensemble-averaged learning curve is obtained by averaging the instantaneous squared error curve over 200 independent trials. For each trial, the MSAF algorithm is iterated until it reaches the steady state. The time average of the last 15 000 samples of the ensemble-averaged learning curve is then used to calculate the steady-state MSE.

The steady-state MSE curve, treated as the function of the step size, is bounded by two asymptotes, $J = J_{\min} = -40$ dB and $\mu = 2.0$, as illustrated in Figures 7.3 and 7.4. When the step size approaches the stability bound of $\mu = 2.0$, the steady-state MSE increases tremendously. The algorithm diverges when the step size μ is larger than the stability bound defined in the inequality (7.58). The minimum MSE, J_{\min} , is determined by the variance of the disturbance, $\sigma_{\eta}^2 = 1 \times 10^{-4}$. A smaller step size leads to a lower steady-state MSE towards the vicinity of the horizontal asymptote $J = J_{\min}$. Nevertheless, the achievable steady-state MSE, $J(\infty)$, is always higher than J_{\min} , which reflects the stochastic nature of the algorithm.

7.5.3 Steady-state excess MSE

Figures 7.5 and 7.6 show the experimental and theoretical values of the steady-state excess MSE for the MSAF algorithm using various step sizes. The theoretical values are calculated using Equation (7.66). In Figure 7.5, the MSAF algorithm (for $N = 4, 8, 16$ and 32 subbands) is excited by white Gaussian noise, whereas the results in Figure 7.6 is for the AR(2) signal. Observe that Equation (7.66) leads to a good fit between theory and practice over the range of step sizes from 0.01 to 1.00 for both white and colored

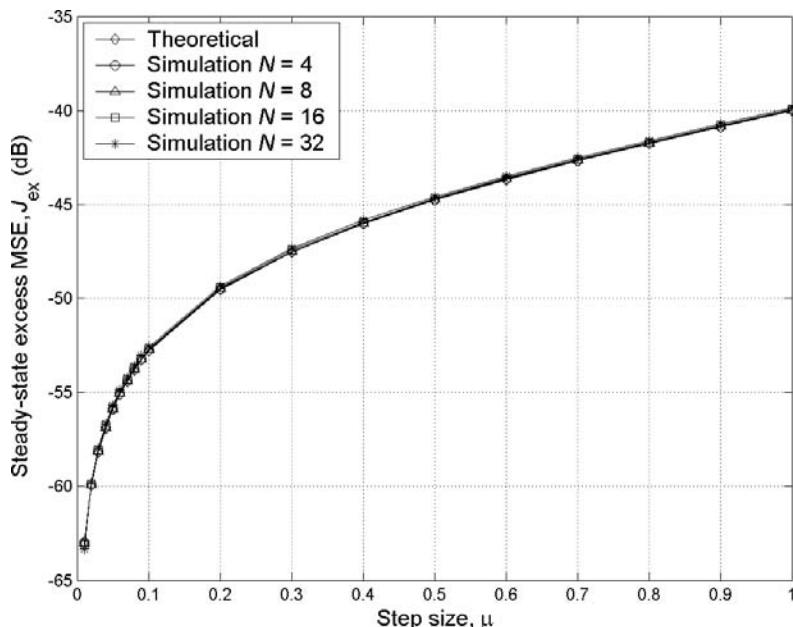


Figure 7.5 Steady-state excess MSE of the MSAF algorithm (for $N = 4, 8, 16$ and 32 subbands) under the white input signal

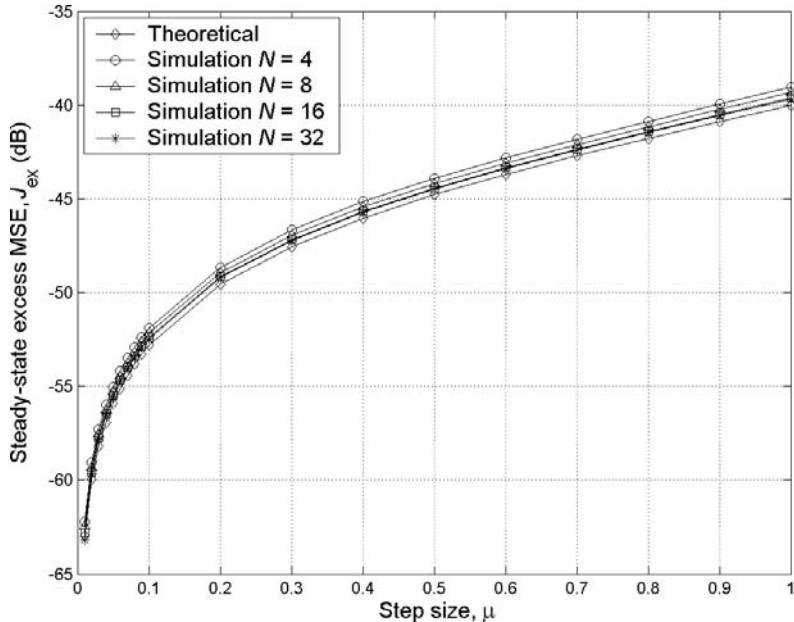


Figure 7.6 Steady-state excess MSE of the MSAF algorithm (for $N = 4, 8, 16$ and 32 subbands) under the colored input signal (adapted from Reference [22])

input signals. Both theoretical and experimental results also show that the excess MSE J_{ex} is independent of the number of subbands, N .

Comparing the results shown in Figures 7.5 and 7.6, it is observed that a better fit between theoretical and experimental results can be obtained for the case of white signals. This is because we have assumed that subband signals are uncorrelated in order to derive the relation (7.63) from (7.62). Consequently, the experimental results in Figure 7.6 deviated from the values predicted by Equation (7.66). However, when the input signal is white, all the subband quantities in the relation (7.62) are normalized by the same factor, where

$$E \left\{ \| \mathbf{u}_i(k) \|^2 \right\} = M \sigma_u^2, \text{ for } i = 0, 1, \dots, N - 1. \quad (7.67)$$

Therefore, the relation (7.62) directly leads to (7.63) without the assumption that the subband signals are uncorrelated, which is generally not true even for the case of white signals. Consequently, the experimental values shown in Figure 7.5 present a better match with the theoretical values given by Equation (7.66).

7.6 Summary

This chapter analyzed and evaluated the convergence behavior of the MSAF algorithm in the mean and mean-square senses. The existence of filter banks complicates the convergence analysis of SAF algorithms. In order to make the mathematical analysis tractable, filter banks were not explicitly included in the convergence analysis presented in this

chapter. Instead, the properties that the filter bank would impose on the subband signals were identified and taken into consideration. In particular, the multiband MSE function was used as the performance measure for the MSAF algorithm. By virtue of the paraunitary property of the analysis filter bank, the multiband MSE function was shown to be equivalent to the classical MSE function. Since the multiband MSE function is defined in terms of subband estimation errors, mathematical terms pertaining to the filter banks were avoided in the analytical analysis.

Mean-square performance of the MSAF algorithm was analyzed based on the energy conservation relation. By using various error terms in the energy relation, stability bounds for the step size and the steady-state MSE were derived. In spite of using several assumptions, a good match between theoretical and experimental results was confirmed through simulations.

References

- [1] M. Rupp and A. H. Sayed, ‘A time-domain feedback analysis of filtered-error adaptive gradient algorithms’, *IEEE Trans. Signal Processing*, **44**(6), June 1996, 1428–1439.
- [2] A. H. Sayed and M. Rupp, ‘Robustness issues in adaptive filtering’, in *Digital Signal Processing Handbook* (eds V. K. Madisetti and D. B. Williams), Boca Raton, Florida: CRC Press, 1999.
- [3] A. H. Sayed, *Fundamentals of Adaptive Filtering*, New York: John Wiley & Sons, Inc., 2003.
- [4] A. H. Sayed and V. H. Nascimento, ‘Energy conservation and the learning ability of LMS adaptive filters’, in *Least-Mean-Square Adaptive Filters* (eds S. Haykin and B. Widrow), New York: John Wiley & Sons, Inc., 2004.
- [5] H. C. Shin and A. H. Sayed, ‘Mean-square performance of a family of affine projection algorithms’, *IEEE Trans. Signal Processing*, **52**(1), January 2004, 90–102.
- [6] N. R. Yousef and A. H. Sayed, ‘A unified approach to the steady-state and tracking analyses of adaptive filtering algorithms’, in *Proc. IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing*, 1999, vol. 2, pp. 699–703.
- [7] N. R. Yousef and A. H. Sayed, ‘A unified approach to the steady-state and tracking analyses of adaptive filters’, *IEEE Trans. Signal Processing*, **49**(2), February 2001, 314–324.
- [8] P. L. de León and D. M. Etter, ‘Experimental results with increased bandwidth analysis filters in oversampled, subband acoustic echo cancellers’, *IEEE Signal Processing Lett.*, **2**(1), January 1995, 1–3.
- [9] A. Gilloire and M. Vetterli, ‘Adaptive filtering in subbands with critical sampling: analysis, experiments, and application to acoustic echo cancellation’, *IEEE Trans. Signal Processing*, **40**(8), August 1992, 1862–1875.
- [10] W. Kellermann, ‘Analysis and design of multirate systems for cancellation of acoustical echoes’, in *Proc. IEEE ICASSP*, 1988, pp. 2570–2573.
- [11] J. J. Shynk, ‘Frequency domain and multirate adaptive filtering’, *IEEE Signal Processing Mag.*, **9**, January 1992, 14–37.
- [12] S. M. Kuo and W. S. Gan, *Digital Signal Processors: Architectures, Implementation, Applications*, Upper Saddle River, New Jersey: Prentice Hall, 2005.
- [13] S. Haykin, *Adaptive Filter Theory*, 4th edition, Upper Saddle River, New Jersey: Prentice Hall, 2002.
- [14] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Englewood Cliffs, New Jersey: Prentice Hall, 1985.

- [15] H. S. Malvar, *Signal Processing with Lapped Transform*, Norwood, Massachusetts: Artech House, 1992.
- [16] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Englewood Cliffs, New Jersey: Prentice Hall, 1993.
- [17] A. Mertins, *Signal Analysis: Wavelets, Filter Banks, Time-Frequency Transforms and Applications*, New York: John Wiley & Sons, Inc., 1999.
- [18] G. Strang, *Introduction to Linear Algebra*, Wellesley, Massachusetts: Wellesley-Cambridge Press, 2003.
- [19] S. C. Douglas and M. Rupp, ‘Convergence issues in the LMS adaptive filter’, in *Digital Signal Processing Handbook* (eds V. K. Madisetti and D. B. Williams), Boca Raton, Florida: CRC Press, 1999.
- [20] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*, New York: John Wiley & Sons, Inc., 1998.
- [21] R. D. Koilpillai and P. P. Vaidyanathan, ‘Cosine-modulated FIR filter banks satisfying perfect reconstruction’, *IEEE Trans. Signal Processing*, **40**(4), April 1992, 770–783.
- [22] K. A. Lee, W. S. Gan and S. M. Kuo, ‘Mean-square performance analysis of the normalized subband adaptive filter’, in *Proc. Asilomar Conf.*, 2006, pp. 248–252.

8

New research directions

This final chapter surveys recent literature to summarize briefly some research works in the area of subband adaptive filtering. We discuss several important developments and extensions in the areas of filter bank design, structures and algorithms, theoretical analysis and potential applications for subband adaptive filters. This chapter also suggests some open research topics including new research directions and trends in developing and applying subband adaptive filtering to inspire more research and development activities in this important field. We have attempted to cover most of the recent published works in this chapter, but there will still be some omissions through an unfortunate oversight.

8.1 Recent research on filter bank design

Fundamental concepts of designing prototype lowpass filters for perfect-reconstruction filter banks were discussed in previous chapters. There are many excellent references [1–3] that describe methods in designing prototype filters specific for SAFs to minimize the aliasing errors between adjacent subbands.

An aliasing-free subband decomposition can only be achieved using a filter-bank-sum method, i.e. without decimation or the decimation factor $D = 1$. If $D > 1$, aliasing will arise. In a through-connect (trivial) filter bank without processing in between the analysis and synthesis filters (see Figure 2.9), aliasing can be canceled in the synthesis section of the filter bank with the perfect-reconstruction design. Unfortunately, the aliasing is always present in the analysis section and subsequently degrades the convergence of subband adaptive filters that follow the analysis filters. The undesired aliasing errors can be reduced by using oversampling filter banks, i.e. $D < N$, where N is the number of subbands [4]. As described in previous chapters, other techniques (summarized in Section 4.3) such as introducing adaptive cross-filters between adjacent subbands [5, 6], using multiband-structured SAF [7–9] and employing the closed-loop delayless structures [10, 11] can reduce the aliasing errors with a tradeoff of system computational complexity.

It is always preferable to have filter banks with a linear-phase response (or identical group delay) for all frequencies. A method to achieve an approximately linear phase is to design the synthesis prototype filter as the time-reversed (or mirror-image)

impulse response of the analysis prototype filter, such as the pseudo-QMF described in Section 5.3.5.

In Chapter 5, several perfect-reconstruction subband systems are discussed. However, the perfect-reconstruction property no longer holds with intermediate subband operations such as filtering and coding. The perfect-reconstruction property can be relaxed to near-perfect reconstruction by allowing some minor distortions. Furthermore, a near-perfect-reconstruction filter bank allows some flexibility in the design of analysis and synthesis filters. For example, it uses lower stopband attenuation as compared with the perfect-reconstruction filter banks.

A near-perfect-reconstruction filter bank was designed in Reference [12] to minimize the overall amplitude distortion together with weighted stopband attenuation for the analysis and synthesis filter banks. The resulting analysis and synthesis prototype filters have different characteristics. It also found that the near-perfect-reconstruction filter bank result in lower error suppression and better modeling as compared with the perfect-reconstruction filter bank for adaptive system identification shown in Figure 4.1 [12].

The use of different types of analysis filter banks for adaptive system identification was proposed in Reference [13]. The first type of analysis filter bank partitions the desired signal into subbands to achieve perfect reconstruction with the corresponding specifically designed synthesis filter bank. The second type of analysis filter bank is designed to decompose the input signal based on the statistics of the signal. The use of different analysis filter banks achieves a smaller MSE as compared with the conventional methods of using identical analysis filter banks for both the input and desired signals [14]. In addition, an adaptive algorithm can be employed to design the second type of analysis filter bank without requiring the statistics of the input signal [13].

A comprehensive description of the prototype filter design for oversampled near-perfect-reconstruction generalized DFT filter banks was presented in Reference [15]. It showed that the design of an optimal prototype filter can be transformed into a convex optimization problem. The design criteria are explicitly bounded by the aliasing components in the subbands, distortion induced by the filter bank and image subband errors in the output.

8.2 New SAF structures and algorithms

As discussed in the previous section, an important consideration in designing an SAF is the aliasing effects due to decimation. This critical problem is still open for research to find alternate SAF structures. This section briefly introduces some recent works on in-band aliasing cancellation and tap-length adjustment in adaptive subfilters.

8.2.1 In-band aliasing cancellation

A linear-phase bandwidth-increased analysis filter $H'_i(e^{j\omega})$ of order $2KN$, where K is an integer, was recently proposed in Reference [16] for the cancellation of in-band aliasing. As illustrated in Figure 8.1, the output of filter $U'_i(e^{j\omega})$ consists of inter-band aliasing components. This output signal is decimated by a factor N to form $U'_{i,D}(e^{j\omega})$ and subtracted from the subband signal $U_{i,D}(e^{j\omega})$ to obtain the almost alias-free signal $U''_{i,D}(e^{j\omega})$. Notice that the subband signal $U_{i,D}(e^{j\omega})$ is delayed by $\Delta = K$ samples to compensate for the delay caused by the bandwidth-increased filter $H'_i(e^{j\omega})$. The subtraction causes a spectral

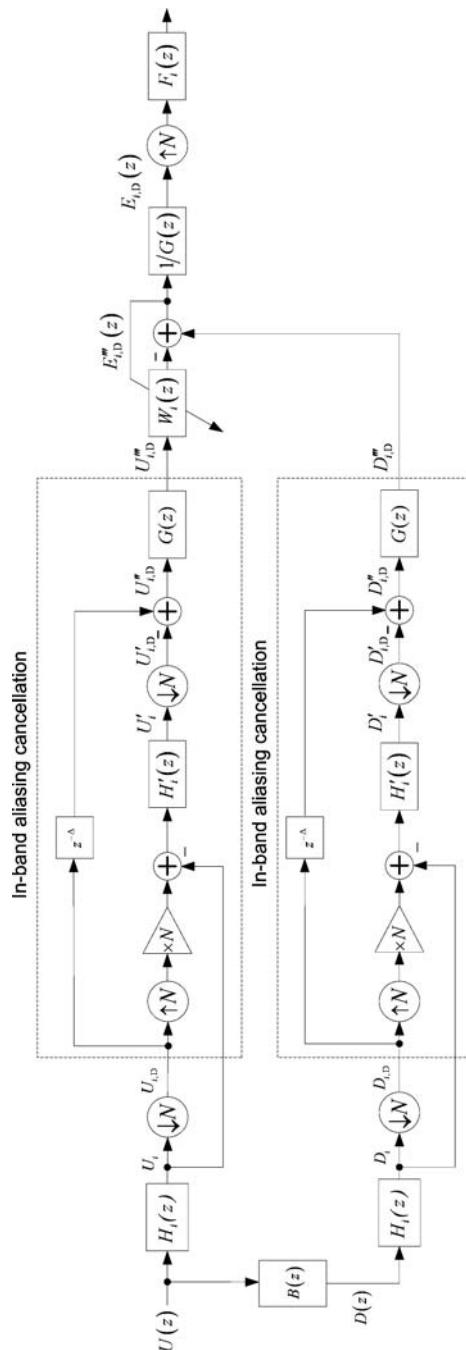


Figure 8.1 Block diagram of the alias-free SAF with critical sampling in the k th subband (adapted from Reference [16])

dip in $U''_{i,D}(e^{j\omega})$, which can be reduced using a minimum-phase filter $G(e^{j\omega})$. The filtered signal $U'''_{i,D}(e^{j\omega})$ is processed by the adaptive subfilter $W_i(e^{j\omega})$. In order to compensate for the effect of the minimum-phase filter, the error signal is filtered by the inverse filter, $1/G(e^{j\omega})$. Detailed simulation results given in Reference [16] illustrated the magnitude responses at each stage of the alias-free critically sampled SAF.

The performance of this alias-free structure is better than the classical fullband adaptive filters under colored input signals such as speech. However, there is no improvement of performance under white signals. The computational complexity of this structure is approximately reduced by a factor of N as compared with the fullband adaptive algorithm. Note that additional computational costs are required for implementation of the bandwidth-increased analysis filter, minimum-phase filter and inverse filter.

Another SAF structure that does not require adaptive cross filters was proposed in Reference [17]. This critically decimated delayless structure uses the fullband error signal and a specially designed uniform DFT analysis filter bank to avoid the need for adaptive cross filters. This SAF structure has improved the convergence rate and reduced computational complexity as compared with the fullband adaptive filters.

The use of a nonuniform, critically sampled SAF for adaptive system identification was developed in Reference [18]. This structure employs normalized step sizes for each subband and operates at the critically decimated rate, thus resulting in significant improvement in terms of the convergence rate and modeling accuracy as compared with the fullband adaptive filters with the LMS algorithm and other uniform SAF algorithms under colored input signals.

8.2.2 Adaptive algorithms for the SAF

The affine projection (AP) algorithm improves the performance of SAFs, as discussed in Section 5.1.1. Many fast AP algorithms were proposed in Reference [19] to reduce computational complexity and improve stability of adaptation. One recent fast AP algorithm presented in Reference [20] uses the Gauss–Seidel iteration to compute the inverse of the input covariance matrix for adaptation in the oversampled SAF. In addition, the selective coefficient update (presented in Section 5.1.3) was used to reduce further the complexity of the SAF. The oversampled SAF with the fast AP algorithm achieved a comparable performance as compared with the fullband AP algorithm in adaptive echo cancellation.

A useful variant of the AP algorithm proposed in Reference [21] incorporates the variable step-size method discussed in Section 5.1.2. The proposed variable step-size AP algorithm for the SAF is based on the optimum approach presented in Reference [22]. This algorithm achieves a better misalignment performance as compared with the fullband AP algorithms under highly correlated input signals. A further reduction of the computational cost can be obtained by using polyphase implementation of subband analysis and synthesis filters.

8.2.3 Variable tap lengths for the SAF

In conventional critically decimated SAFs, the adaptive subfilters $W_i(z)$ usually have the same filter length, $M_S = M/D$. Because a decrease in the filter length will increase

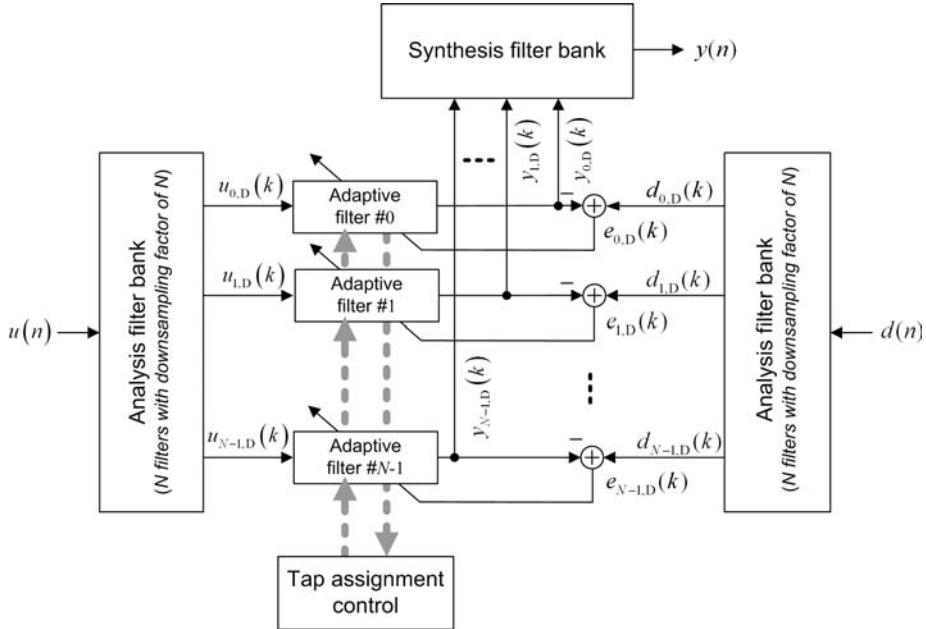


Figure 8.2 A SAF with tap-assignment control

the spectral dynamic range due to decimation of subband signals [23], the length of the adaptive subfilters needs to be increased. Also, an extra delay was inserted in the desired signal path to improve the convergence of the SAF [23].

Adaptive adjustment of the filter length in each subband can enhance the performance of the SAF as well as allocating adequate computational resources to subbands that require more processing power. Several techniques for automatic tap assignment have been proposed in References [24] to [26]. Figure 8.2 shows an additional block called the tap-assignment control, which is introduced in the SAF to vary the adaptive subfilter lengths for different subbands. The variable tap lengths in the SAF are most suitable for system identification applications, where the magnitude spectrum of the unknown system has unequal magnitude gains at different subbands. For example, in AEC, low-frequency bands usually have higher power as compared with high-frequency bands due to the higher absorption in the high-frequency range; thus more taps are required for the adaptive subfilters in lower-frequency bands. In other applications, longer filter lengths can be assigned to subbands with higher power of error signals and shorter filter lengths for subbands with lower power.

One of the effective tap-assignment algorithms is developed in Reference [24] to balance the subband error signals, while the other algorithm [26] performs tap allocation based on a mixed criterion of input signal and trailing tap weights of different subbands. A low-complexity version given in Reference [26] was proposed in Reference [27]. Another algorithm [28] incorporated perceptual criteria based on human hearing to assign different filter lengths for each subband.

8.3 Theoretical analysis

The aliasing effects in subband filtering degrade the convergence performance of SAFs. Due to the increased spectral dynamic range (or larger eigenvalue spread) at band edges, the asymptotic convergence rate is decreased for a typical adaptive algorithm (such as the NLMS algorithm) used in SAFs [29]. Lower bounds for the minimum MSE and modeling accuracy were reported in Reference [30]. Aliasing and filter-bank distortion measurement techniques based on the Heinle–Schüssler method [31] was carried out in Reference [30] without any explicit knowledge of the filter bank.

A more comprehensive analysis by Weiss *et al.* [32] examined the bound of minimum MSE using the power spectral density of aliased signal components, and linked the modeling accuracy of the SAFs to the filter-bank mismatch in perfect reconstruction. In particular, they found that both the minimum MSE and modeling accuracy bounds are linked to the prototype filter in the modulated filter banks. An equivalent model of the SAF was proposed in Reference [33] and the theoretical optimum tap weights and the minimum MSE were derived. The minimum MSE of the SAF was also derived in Reference [34], which shows that the MSE performance is highly dependent on the prototype filter.

A detailed MSE analysis of both uniform and nonuniform SAFs for adaptive system identification was conducted by Griesbach, Lightner and Etter [35]. Both time-domain and frequency-domain techniques are used to derive the subband minimum MSE for both uniform and nonuniform SAFs.

Adaptive system identification using a SAF was analyzed and compared with the fullband adaptive filters in Reference [36]. An important finding is that both the critically sampled and oversampled subband systems can achieve the same performance (in terms of the convergence rate and asymptotic MSE) as the fullband methods. However, SAFs provide computational saving, especially in the case of a critically sampled SAF. Subband filtering parameters such as the number of subbands and decimation factors can be optimized to reduce the computational complexity while keeping the MSE performance compatible with the fullband adaptive filters.

A geometric insight into the sources of errors in adaptive system identification using SAFs was presented in Reference [14]. It was found that the minimum MSE bound is related to the error in analysis filter design using the least-squares estimation and is independent of the response of an unknown system. This estimation error can be reduced by designing two different analysis filters for the desired and input branches of the adaptive system identification. This proposal also coincides with the use of a minimum MSE as a criterion for analysis filter design reported in Reference [13]. Finally, the MSE performance was derived for the MSAF structures in Reference [37]. The mean and mean-square analysis is detailed in Chapter 7.

8.4 Applications of the SAF

Subband adaptive filtering has found many applications in the areas of speech enhancement [38, 39], acoustic echo cancellation [40, 41], speech dereverberation [42], noise cancellation [43, 44], channel equalization [45, 46], microphone array processing [47, 48], active noise control [49–51], interference cancellation [52, 53], hearing aids [54, 55], acoustic shock limitation [56], audio coding and processing [57–60], ultrasonic

underwater target detection [61, 62], image processing [63–65], biomedical systems [66, 67], communication systems [68–70] and much more. The main motivations for applying an SAF for practical applications include: (i) the ability to reduce computational complexity, (ii) flexibility in using different adaptive strategies (such as adaptive algorithms, step sizes and filter lengths) for different subbands, (iii) improved convergence and misadjustment performance under colored signals and (iv) an inherent parallel structure that is suitable for parallel processing.

The aliasing effects in SAFs have been greatly reduced by several filter-bank design techniques described in this book, and the signal delay inherent in SAFs can be reduced by the delayless SAF structures introduced in Section 4.4. With the proposed solutions introduced in this book, the diverse applications of SAFs clearly indicate the strong potential of employing SAFs in many applications that require processing attributes, such as reduced computational complexity and an increased convergence rate. With the advancement of digital signal processors coupled with optimized library functions for subband adaptive filtering, SAFs with reasonable complexity can be implemented to meet the real-time requirements [71] given by a specific application. Some practical considerations such as low delay, computational complexity and power consumption, as well as real-time implementation and objective–subjective evaluation of SAF performance in applications, were presented in References [41], [44] and [72] to [74].

Several joint filter-bank structures were proposed in References [75] and [76], such as two filter banks being combined into a single filter-bank operation. This joint filter-bank structure has many advantages in low-power implementation by reducing computational complexity and memory usage.

Arising from the need to use more microphones and loudspeakers in some applications such as acoustic echo cancellation and noise reduction, the development of more advanced array processing algorithms coupled with SAF structures provides an attractive solution to handle the problem in the time–space subband domain. Some of these applications use nonlinear adaptive filtering [77], and the subband processing has an added flexibility in achieving better control (in terms of order of nonlinearity, nonlinear algorithms and parameters) of subband signal processing.

8.5 Further research on a multiband-structured SAF

The analysis filter bank partitions the fullband signals into subbands for adaptive processing in the subband domain, whereas the synthesis filter bank reconstructs a fullband signal after processing. These filter banks are essential elements in subband adaptive filtering, but they increase the undesired end-to-end delay of the overall system. The signal-path delay inflicted by the filter banks can be eliminated by delayless structures, discussed in Section 6.6. However, the subband signals used for weight adaptation (in the auxiliary path) are still delayed by the analysis filter banks, even though fullband filtering is delayless. This weight-update delay may be negligible for a high-order adaptive filter, where the length of adaptive filter is much longer than that of the analysis filters. Nevertheless, the delay may have a considerable impact on the tracking performance of the MSAF algorithm for nonstationary signals, as postulated in Reference [78, p. 15] for the general case of subband adaptive filtering. Thus the tracking performance of the MSAF algorithm under nonstationary signals deserves further analysis. For this purpose,

the energy conservation relation presented in Section 7.4 can be extended to the MSAF structures using the first-order random-walk model [79–82].

As shown by mathematical analysis and simulations presented in Section 3.4, cosine-modulated filter banks are able to generate subband signals that are orthogonal at zero lag. The MSAF algorithm relies on the subband orthogonality of the cosine-modulated filter banks to justify the use of the diagonal assumption (see Section 6.5.1), which greatly simplifies the MSAF recursion. This restriction on the filter banks that can be used by the MSAF algorithm may limit its practical applications. Therefore, further research is needed to investigate the feasibility of using other types of filter banks for the MSAF algorithm. One possible candidate is the frequency-sampling filter [83, pp. 214–218], which leads to a computationally efficient filter bank. It should be emphasized that the filter banks used by the MSAF algorithm must satisfy the following conditions: (i) the outputs of the filter bank are orthogonal at zero lag and (ii) the analysis filters are power complementary. Recall that there is no need to use the synthesis filter bank if the delayless structure is employed in the MSAF algorithm.

The MSAF algorithm can be applied to the multichannel case with additional constraints on channel decorrelation. The resulting algorithm is of particular interest in stereophonic and multichannel acoustic echo cancellation (AEC) applications [84–89]. The additional constraints for channel decorrelation are useful in dealing with the nonuniqueness problem encountered in stereophonic and multichannel AEC. In addition, the MSAF algorithm can be integrated with the subband audio coders in communication systems. In such an integrated system, the cosine-modulated filter bank for the MSAF algorithm can be shared by the front-end echo cancellation and subband audio coders [90], with a minimum increase in the overall computational cost. The basic idea of a joint filter bank for AEC and audio coding was reported in Reference [76]. The strategy for resource sharing between signal-processing blocks can also be extended to include spectral subtraction and beamforming techniques for noise reduction in communication systems.

Several extensions of the MSAF algorithm include: (i) changing the fixed regularization parameter α in Equation (6.21) to variable regularization parameters α_i in each subband in order to increase potentially the convergence rate and reduce misadjustment of the MSAF algorithm, (ii) incorporate the proportionate algorithm to improve the convergence under dispersive and sparse impulse responses with colored signals (as presented Section 5.4) and (iii) use the optimal variable-step-size approach as discussed in Section 5.1.2.

8.6 Concluding remarks

Adaptive signal processing is becoming one of the most important fields in digital signal processing, which is evident by the publication of many books and papers on this subject listed in the references of the book and the many practical applications outlined in Sections 1.6 and 8.4. The research works on adaptive filters can be classified into two categories: (i) developing more advanced filter structures and adaptation algorithms for improving transient, steady-state, computational complexity and/or numerical performance and (ii) analyzing the developed adaptive filters. However, most practical applications are still using the FIR filter with the LMS-type algorithms, which suffers from slow convergence for colored signals and high complexity for applications that require high-order filters such as acoustic echo cancellation.

Subband adaptive filtering has gained extensive attention in the last two decades as an effective technique for practical applications that need to process colored signals with a large spectral dynamic range and/or require very high-order adaptive filters. Subband adaptive filters offer the advantages of computational savings and fast convergence for these problems. Many important SAF structures and algorithms were discussed in previous chapters with theoretical analysis and MATLAB simulations, and some recent developments and applications were briefly introduced in this chapter to provide some hints of future research directions for subband adaptive filtering. As stated in the preface, we would like to acknowledge all the researchers who work in this emerging field of adaptive signal processing and hope that this book will inspire more research and applications in subband adaptive filtering.

References

- [1] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Englewood Cliffs, New Jersey: Prentice Hall, 1983.
- [2] P. Vaidyanathan, *Multirate Systems and Filterbanks*, Englewood Cliffs, New Jersey: Prentice Hall, 1993.
- [3] G. Strang and T. Nguyen, *Wavelets and Filterbanks*, Wellesley, Massachusetts: Wellesley-Cambridge, 1997.
- [4] E. Hänsler and G. Schmidt, *Acoustic Echo and Noise Control*, Hoboken, New Jersey: John Wiley & Sons, Inc., 2004.
- [5] A. Gilloire and M. Vetterli, ‘Adaptive filtering in subbands with critical sampling: analysis, experiments, and application to acoustic echo cancellation’, *IEEE Trans. Signal Processing*, **40**(8), August 1992, 1862–1875.
- [6] M. R. Petraglia, R. G. Alves and P. S. R. Diniz, ‘New structures for adaptive filtering in subbands with critical sampling’, *IEEE Trans. Signal Processing*, **48**(12), December 2000, 3316–3327.
- [7] K. A. Lee and W. S. Gan, ‘Improving convergence of the NLMS algorithm using constrained subband updates’, *IEEE Signal Processing Lett.*, **11**(9), September 2004, 736–739.
- [8] M. de Courville and P. Duhamel, ‘Adaptive filtering in subbands using a weighted criterion’, *IEEE Trans. Signal Processing*, **46**(9), September 1998, 2359–2371.
- [9] S. S. Pradhan and V. U. Reddy, ‘A new approach to subband adaptive filtering’, *IEEE Trans. Signal Processing*, **47**(3), March 1999, 655–664.
- [10] D. R. Morgan and J. C. Thi, ‘A delayless subband adaptive filter architecture’, *IEEE Trans. Signal Processing*, **43**(8), August 1995, 1819–1830.
- [11] S. Ohno and H. Sakai, ‘On delayless subband adaptive filtering by subband/fullband transforms’, *IEEE Trans. Signal Processing*, **6**(9), September 1999, 236–239.
- [12] K. Eneman and M. Moonen, ‘DFT modulated filter bank design for oversampled subband systems’, *Signal Processing*, **81**(9), 2001, 1947–1973.
- [13] Y. Higa, H. Ochi and S. Kinjo, ‘A subband adaptive filter with the statistically optimum analysis filter bank’, *IEEE Trans. Circuits and Systems – II*, **45**(8), August 1998, 1150–1154.
- [14] J. H. Gunther and G. Wilson, ‘Mean-squared error analysis of adaptive subband-based system identification’, *IEEE Trans. Audio, Speech, and Language Processing*, **16**(8), November 2008, 1452–1465.
- [15] M. R. Wilbur, T. N. Davidson and J. P. Reilly, ‘Efficient design of oversampled NPR GDFT filterbanks’, *IEEE Trans. Signal Processing*, **52**(7), July 2004, 1947–1963.

- [16] S. G. Kim, C. D. Yoo and T. Q. Nguyen, ‘Alias-free subband adaptive filtering with critical sampling’, *IEEE Trans. Signal Processing*, **56**(5), May 2008, 1894–1904.
- [17] R. Merched, P. S. R. Diniz and M. R. Petraglia, ‘A new delayless subband adaptive filter structure’, *IEEE Trans. Signal Processing*, **47**(6), June 1999, 1580–1591.
- [18] M. P. Petraglia and P. B. Batalheiro, ‘Nonuniform subband adaptive filtering with critical sampling’, *IEEE Trans. Signal Processing*, **56**(2), February 2008, 565–575.
- [19] S. Haykin and B. Widrow (eds), *Least-Mean-Square Adaptive Filters*, Hoboken, New Jersey: John Wiley & Sons, Inc., 2003.
- [20] E. Chau, H. Sheikhzadeh and R. L. Brennan, ‘Complexity reduction and regularization of a fast affine projection algorithm for oversampled subband adaptive filters’, in *Proc. ICASSP*, 2004, pp. 17–21.
- [21] H. Choi, S. W. Sohn, A. C. Youn, J. W. Suh and H. D. Bae, ‘Subband affine projection algorithm using variable step size’, in *Proc. ICASSP*, 2006, pp. 189–192.
- [22] H. C. Shin, A. H. Sayed and W. J. Song, ‘Variable step-size NLMS and affine projection algorithms’, *IEEE Signal Processing Lett.*, **11**(2), February 2004, 132–135.
- [23] K. Eneman and M. Moonen, ‘Adding causal and anti-causal filter taps to enhance the performance of subband adaptive filters’, in *Proc. IEEE Benelux Signal Processing Symp.*, 2002, pp. 1–4.
- [24] Z. Ma, K. Nakayama and A. Sugiyama, ‘Automatic tap assignment in sub-band adaptive filter’, *IEICE Trans. Communication*, **E76-B**(7), July 1993, 751–754.
- [25] A. Sugiyama and A. Hirano, ‘A subband adaptive filtering with adaptive intersubband tap-assignment’, *IEICE Trans. Fundamentals of Electronics, Communications, and Computer Science*, **E77-A**(9), September 1994, 1432–1438.
- [26] A. Sugiyama and F. Landais, ‘A new adaptive intersubband tap-assignment algorithm for subband adaptive filters’, in *Proc. ICASSP*, 1995, pp. 3051–3054.
- [27] S. Weiss, U. Sorgel and R. W. Stewart, ‘Computationally efficient adaptive system identification in subbands with intersubband tap assignment for undermodelled problems’, in *Proc. Asilomar Conf.*, 1996, pp. 818–822.
- [28] M. Vukadinovic and T. Aboulnasr, ‘A study of adaptive intersubband tap assignment algorithms from a psychoacoustic point of view’, in *Proc. ISCAS*, 1996, pp. 65–68.
- [29] D. R. Morgan, ‘Slow asymptotic convergence of LMS acoustic echo cancelers’, *IEEE Trans. Speech and Audio Processing*, **3**(2), March 1995, 126–136.
- [30] A. Stenger, R. Rabenstein, S. Weiss and R. W. Stewart, ‘Measuring performance limits of subband adaptive systems’, in *Proc. Asilomar Conf.*, 1998, pp. 1176–1180.
- [31] F. A. Heinle and H. W. Schuessler, ‘Measuring the performance of implemented multirate systems’, in *Proc. ICASSP*, 1996, pp. 2754–2757.
- [32] S. Weiss, A. Stenger, R. W. Stewart and R. Rabenstein, ‘Steady-state performance limitations of subband adaptive filters’, *IEEE Trans. Signal Processing*, **49**(9), September 2001, 1982–1991.
- [33] Y. Ono and H. Kiya, ‘Performance analysis of subband adaptive systems using an equivalent model’, in *Proc. ICASSP*, 1994, pp. 19–22.
- [34] H. Munemoto and H. Ocji, ‘Total error performance analysis of subband adaptive digital filters’, in *Proc. Asilomar Conf.*, 1998, pp. 1386–1391.
- [35] J. D. Griesbach, M. R. Lightner and D. M. Etter, ‘Mean square error analysis of nonuniform subband adaptive filters for system modeling’, *IEEE Trans. Signal Processing*, **53**(2), February 2005, 550–563.
- [36] D. Marelli and M. Y. Fu, ‘Performance analysis for subband identification’, *IEEE Trans. Signal Processing*, **52**(1), January 2004, 142–154.

- [37] K. A. Lee, W. S. Gan and S. M. Kuo, ‘Mean-square performance analysis of the normalized subband adaptive filter’, in *Proc. Asilomar Conf.*, 2006, pp. 248–252.
- [38] W. R. Wu and P. C. Chen, ‘Subband Kalman filtering for speech enhancement’, *IEEE Trans. Circuits and Systems – II*, **45**(8), August 1998, 1072–1083.
- [39] H. R. Abutalebi, H. Sheikhzadeh, R. L. Brennan and G. H. Freeman, ‘A hybrid subband adaptive system for speech enhancement in diffuse noise fields’, *IEEE Signal Processing Lett.*, **11**(1), January 2004, 44–47.
- [40] S. L. Gay and J. Benesty (eds), *Acoustic Signal Processing for Telecommunication*. Norwell, Massachusetts: Kluwer Academic Publishers, 2000.
- [41] K. Steinert, M. Schonle, C. Beaugeant and T. Fingscheidt, ‘Hands-free system with low-delay subband acoustic echo control and noise reduction’, in *Proc. ICASSP*, 2008, pp. 1521–1524.
- [42] B. W. Gillespie, H. S. Malvar and D. A. F. Florencio, ‘Speech dereverberation via maximum-kurtosis subband adaptive filtering’, in *Proc. ICASSP*, 2001, pp. 3701–3704.
- [43] B. H. Lee and S. M. Kuo, ‘Sub-band adaptive IIR noise cancellation’, in *Proc. ISCAS*, 1990, pp. 775–778.
- [44] H. Sheikhzadeh, H. R. Abutalebi, R. L. Brennan and J. Sollazzol, ‘Performance limitations of a new subband adaptive system for noise and echo reduction’, in *Proc. Electronics, Circuits and Systems*, 2003, pp. 459–462.
- [45] R. W. Stewart, S. Weiss, D. Garcia-Alis and G. C. Freeland, ‘Subband adaptive equalization of time-varying channels’, in *Proc. Asilomar Conf.*, 1999, pp. 534–538.
- [46] T. Miyagi and H. Ochi, ‘An alias-free subband adaptive equalizer for OFDM system’, in *Proc. Asilomar Conf.*, 2000, pp. 1811–1814.
- [47] J. M. de Haan, N. Grbic, I. Claesson and S. E. Nordholm, ‘Filter bank design for subband adaptive microphone arrays’, *IEEE Trans. Speech and Audio Processing*, **11**(1), January 2003, 14–23.
- [48] A. Davis, S. Y. Low, S. Nordholm and N. Grbic, ‘A subband space constrained beamformer incorporating voice activity detection’, in *Proc. ICASSP*, 2005, pp. 65–68.
- [49] A. A. Milani, I. M. S. Panahi and R. Briggs, ‘Performance analysis of sub-band nLMS, APA and RLS in fMRI ANC with a non-minimum phase secondary path’, in *Proc. ICASSP*, 2007, pp. 353–356.
- [50] S. J. Park, J. H. Yun, Y. C. Park and D. H. Youn, ‘A delayless subband active noise control system for wideband noise control’, *IEEE Trans. Speech and Audio Processing*, **9**(8), November 2001, 892–899.
- [51] S. M. Kuo, E. K. Yenduri and A. Gupta, ‘Frequency-domain delayless active sound quality control algorithm’, *J. Sound and Vibration*, **318**(4–5), 2008, 715–725.
- [52] D. J. Rabideau, ‘Optimal linear constraints and subspace adaptive filtering’, in *Proc. Sensor Array and Multichannel Signal Processing Workshop*, 2000, pp. 310–314.
- [53] M. R. Edalatzadeh and M. H. V. Samiei, ‘A new eigenspace subband beamformer for cancellation of broadband interferences using subband adaptive filtering’, in *Proc. Int. Symp. Telecommunications*, 2008, pp. 284–289.
- [54] V. Parsa and D. G. Jamieson, ‘Adaptive modeling of digital hearing aids using a subband affine projection algorithm’, in *Proc. ICASSP*, 2002, pp. 1937–1940.
- [55] M. G. Siqueira, R. Speece, E. Petsalis, A. Alwan, S. Soli and S. Gao, ‘Subband adaptive filtering applied to acoustic feedback reduction in hearing aids’, in *Proc. Asilomar Conf.*, 1996, pp. 788–792.
- [56] G. Choy, D. Hermann, R. L. Brennan, T. Schneider, H. Sheikhzadeh and E. Cornu, ‘Subband-based acoustic shock limiting algorithm on a low-resource DSP system’, in *Proc. EUROSPEECH*, 2003, pp. 2869–2872.

- [57] P. Dymarski, ‘Watermarking of audio signals using adaptive subband filtering and manchester signaling’, in *Proc. EURASIP Conf. Speech and Image Processing, Multimedia Communications and Services*, 2007, pp. 221–224.
- [58] P. F. Driessen and L. Yan, ‘An unsupervised adaptive filtering approach of 2-to-5 channel upmix’, in *Proc. 119 Audio Engineering Convention*, 2005, pp. 1–7.
- [59] E. Camberlein, P. Philippe and F. Bimbot, ‘Adaptive filter banks using fixed size MDCT and subband merging for audio coding comparison with the MPEG AAC filter banks’, in *Proc. 121 Audio Engineering Convention*, 2006, pp. 1–9.
- [60] A. Rimell and M. Hawksford, ‘Reduction of loudspeaker polar response aberrations using psychoacoustically motivated adaptive subband algorithm’, in *Proc. 97 Audio Engineering Convention*, 1994, pp. 1–34.
- [61] J. Mao and M. X. Li, ‘Application of subband adaptive filtering techniques to ultrasonic detection in multilayers’, *J. Acoust. Soc. Am.*, **109**(5), May 2001, 2318.
- [62] C. H. Yuan, M. R. Azimi-Sadjadi, J. Wilbur and G. J. Dobeck, ‘Underwater target detection using multichannel subband adaptive filtering and high-order correlation schemes’, *IEEE J. Oceanic Engng*, **25**(1), January 2000, 192–205.
- [63] D. W. Zhou, ‘An image denoising algorithm with an adaptive window’, in *Proc. Int. Conf. Image Processing*, 2007, pp. 333–336.
- [64] D. M. Monro, W. Huo and X. P. Wang, ‘Subband adaptive dictionaries for wavelet/matching pursuits image coding’, in *Proc. Int. Conf. Image Processing*, 2006, pp. 2133–2136.
- [65] S. Gezici, I. Yilmaz, O. N. Gerek and E. Cetin, ‘Image denoising using adaptive subband decomposition’, in *Proc. Int. Conf. Image Processing*, 2001, pp. 261–264.
- [66] A. Jimenez, M. R. Ortiz, M. A. Pena, S. Charleston, A. T. Aljama and R. Gonzalez, ‘Performance of two adaptive subband filtering schemes for processing fetal phonocardiograms: influence of the wavelet and the level of decomposition’, in *Proc. Computers in Cardiology*, 2000, pp. 427–430.
- [67] M. Karrakchou, C. van den Branden Lambrecht and M. Kunt, ‘Analyzing pulmonary capillary pressure’, *IEEE Engineering in Medicine and Biology Mag.*, **14**(2), March–April 1995, 179–185.
- [68] M. V. Tazebay and A. N. Akansu, ‘Adaptive subband transforms in time-frequency excisers for DSSS communications systems’, *IEEE Trans. Signal Processing*, **43**(11), November 1995, 2776–2782.
- [69] Y. Zhang, S. Weiss and L. Hanzo, ‘Subband adaptive antenna array for wideband wireless communications’, in *Proc. Microwave and Millimeter Wave Technology*, 2002, pp. 693–696.
- [70] P. Z. Tu, X. J. Huang and E. Dutkiewicz, ‘Subband adaptive filtering for efficient spectrum utilization in cognitive radios’, in *Proc. Cognitive Radio Oriented Wireless Networks and Communications*, 2008, pp. 1–4.
- [71] S. M. Kuo and W. S. Gan, *Digital Signal Processors: Architectures, Implementations, and Applications*, Upper Saddle River, New Jersey: Prentice Hall, 2005.
- [72] P. Enerothe, S. L. Gay, T. Gänslar and J. Benesty, ‘A real-time implementation of a stereophonic acoustic echo canceller’, *IEEE Trans. Speech and Audio Processing*, **9**(5), July 2001, 513–523.
- [73] E. Cornu, A. Dufaux and D. Hermann, ‘A high performance, low latency, low power audio processing system for wideband speech over wireless links’, in *Proc. EUSIPCO*, 2006.
- [74] D. Hermann, E. Cornu, T. Soltani and H. Sheikhzadeh, ‘Low-power implementation of a subband fast affine projection algorithm for acoustic echo cancellation’, in *Proc. EUSIPCO*, 2005.

- [75] R. Dong, D. Hermann, R. Brennan and E. Chau, ‘Joint filterbank structures for integrating audio coding into hearing aid applications’, in *Proc. ICASSP*, 2008, pp. 1533–1536.
- [76] P. Enero, ‘Joint filterbanks for echo cancellation and audio coding’, *IEEE Trans. Speech Audio Processing*, **11**(4), July 2003, 342–354.
- [77] S. Haykin, *Adaptive Filter Theory*, 4th edition, Upper Saddle River, New Jersey: Prentice Hall, 2002.
- [78] J. J. Shynk, ‘Frequency-domain and multirate adaptive filtering’, *IEEE Signal Processing Mag.*, **9**, January 1992, 14–37.
- [79] A. H. Sayed, *Fundamentals of Adaptive Filtering*, New York: John Wiley & Sons, Inc., 2003.
- [80] H. C. Shin and A. H. Sayed, ‘Mean-square performance of a family of affine projection algorithms’, *IEEE Trans. Signal Processing*, **52**(1), January 2004, 90–102.
- [81] N. R. Yousef and A. H. Sayed, ‘A unified approach to the steady-state and tracking analyses of adaptive filtering algorithms’, in *Proc. IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing*, 1999, vol. 2, pp. 699–703.
- [82] N. R. Yousef and A. H. Sayed, ‘A unified approach to the steady-state and tracking analyses of adaptive filters’, *IEEE Trans. Signal Processing*, **49**(2), February 2001, 314–324.
- [83] S. M. Kuo and B. H. Lee, *Real-Time Digital Signal Processing*, New York: John Wiley & Sons, Inc., 2001.
- [84] J. Benesty, D. R. Morgan and M. M. Sondhi, ‘A better understanding and an improved solution to the specific problems of stereophonic acoustic echo cancellation’, *IEEE Trans. Speech Audio Processing*, **6**(2), March 1998, 156–165.
- [85] T. Gänsl and J. Benesty, ‘Stereophonic acoustic echo cancellation and two-channel adaptive filtering: an overview’, *Int. J. Adaptive Control and Signal Processing*, **14**, 2000, 565–586.
- [86] T. Gänsl and J. Benesty, ‘Multichannel acoustic echo cancellation: what’s new’, in *Proc. Int. Workshop on Acoustic Echo and Noise Control*, 2001.
- [87] K. A. Lee, W. S. Gan, J. Yang and F. Sattar, ‘Multichannel teleconferencing system with multi spatial region acoustic echo cancellation’, in *Proc. Int. Workshop on Acoustic Echo and Noise Control*, 2003, pp. 51–54.
- [88] M. M. Sondhi and D. R. Morgan, ‘Acoustic echo cancellation for stereophonic teleconferencing’, in *Proc. IEEE Workshop Applications of Signal Processing to Audio and Acoustics*, 1991, pp. 141–142.
- [89] M. M. Sondhi and D. R. Morgan, ‘Stereophonic acoustic echo cancellation – an overview of the fundamental problem’, *IEEE Signal Processing Lett.*, **2**(8), August 1995, 148–151.
- [90] T. Painter and A. Spanias, ‘Perceptual coding of digital audio’, *Proc. IEEE*, **88**(4), April 2000, 451–513.

Appendix A

Programming in MATLAB

This appendix provides a quick reference on the fundamentals of MATLAB, which is used in the examples, applications and implementation of adaptive algorithms in this book. This appendix also covers useful topics related to digital signal processing (DSP) in MATLAB, including Signal Processing Toolbox and Filter Design Toolbox. A more detailed description is documented in the MATLAB and toolboxes user guides. These user guides are accessible from the on-line help browser of MATLAB using the `helpwin` command.

A.1 MATLAB fundamentals

MATLAB stands for MATrix LABoratory and has become a widely used technical computing language in engineering and scientific fields, especially for DSP research and development. It allows users to perform numerical computation, simulation, acquisition and visualization of data for algorithm design, analysis and implementation. Unlike other high-level programming languages such as C, MATLAB provides a comprehensive suite of mathematical, statistical and engineering functions. The functionality is further enhanced with interactive and flexible graphical capabilities for creating two-dimensional (2D) and three-dimensional (3D) plots.

Furthermore, various toolboxes are available for working under the MATLAB environment. These toolboxes are collections of algorithms written by experts in their fields and provide application-specific capabilities. These toolboxes greatly enhance MATLAB functionality in signal processing, data analysis and statistics, mathematical modeling, control systems design, etc.

A.1.1 Starting MATLAB

To start MATLAB, double click on the icon  on the desktop. The MATLAB Desktop appears as shown in Figure A.1, which provides an integrated environment in developing and testing MATLAB programs. Note that this Desktop is based on Version 7.0 and

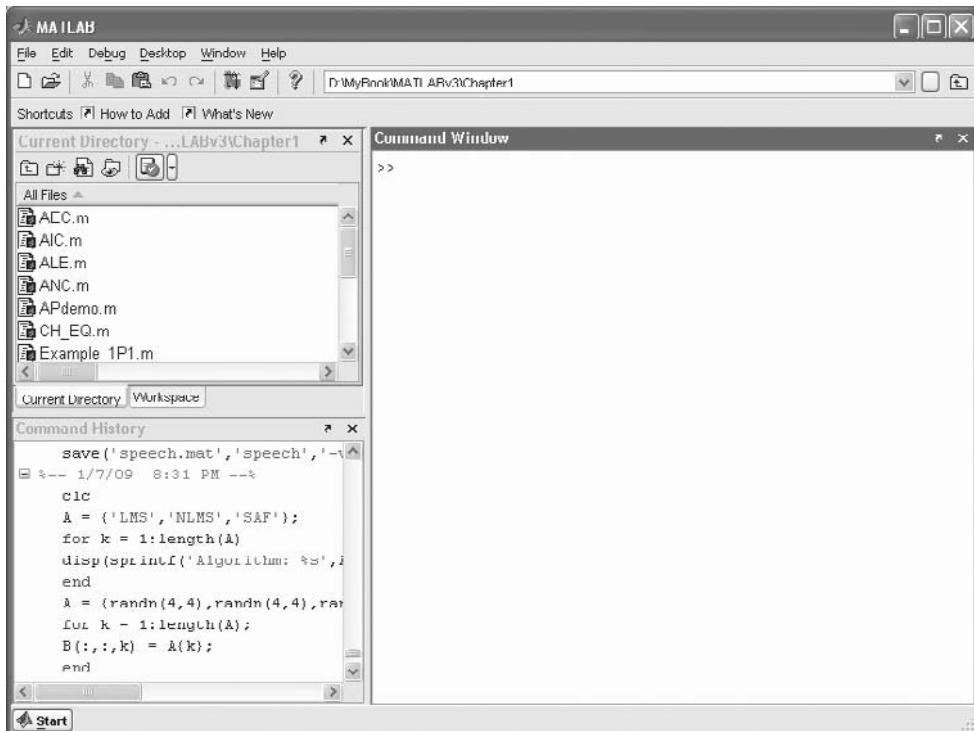


Figure A.1 MATLAB Desktop (Version 7.0) using the default layout

has some minor differences with other versions of MATLAB. In this appendix, we use Version 7.0 to illustrate basic MATLAB programming concepts.

The Command Window (on the right side of the Desktop) is the main window that allows the user to enter MATLAB commands after the prompt `>>`. Some commonly used commands are listed as follows:

1. `help`: to display help text in the Command Window by listing topics that are available. In addition, it shows a list of toolboxes and add-on programs installed under the MATLAB environment.
2. `help function`: to display the usage of a particular MATLAB `function`. Alternatively, we may use `helpwin` to see the output of help displayed in the Help browser window instead of the Command Window.
3. `whos`: to display a list of variables, together with their sizes and types, currently loaded into the workspace. Alternatively, we may use `who` to just display the list of variables.
4. `what dir`: to display the files in the directory `dir`.
5. `which function`: to display the full pathname of `function` used. This command is particularly useful to ensure that the right version of the M-file is called.

6. `clear all`: to clear all the variables in the workspace. It is always good practice to have this command at the beginning of an M-file so that unrelated variables used earlier are removed from the workspace.
7. `clear x`: to clear the variable `x` from the workspace.
8. `save('result.mat','x','y')`: to save variables `x` and `y` to the binary disk file `result.mat`, which can be retrieved with the `load` function.
9. `load('result.mat','x')`: to load variable `x` from the disk file `result.mat`. If the variable name is not specified, all variables are loaded and overwrite existing variables in the workspace.
10. `addpath 'c:\project\common'`: to add a specific directory to the MATLAB search path. In many MATLAB programs given in the companion CD, we use this command to add the folder ‘Common’, which contains commonly used functions and data files to the MATLAB search path. Use the `path` command to see the current MATLAB search path.
11. `clc`: to clear the Command Window and home the cursor. This command is useful to clear up the Command Window when it becomes too messy.
12. `quit`: to exit MATLAB and return all allocated memory to the system.

Besides the Command Window, there is the Command History window located at the bottom left of the Desktop, as shown in Figure A.1. The Command History window records all the executed commands and the date and time when these commands were executed. This is a useful feature in recalling what commands have been executed previously. To the upper left of the Desktop, there are two defaulted windows: Current Directory and Workspace windows. The Current Directory window keeps track of the files inside the current directory. The Workspace window is used to organize the loaded variables, which also displays the size of variables, bytes and class. Located at the bottom-left corner, the **Start** button provides access to tools, toolboxes, demos and MATLAB documentation.

A set of toolbars at the top of the Desktop performs the following functions:

	New file, open file
	Cut, copy, paste
	Undo last action, redo last action
	Simulink library browser
	Open help browser

MATLAB Desktop also includes the following features under the **File** pull-down menu:

1. *Import data from* a file directory into the MATLAB workspace.
2. *Save* MATLAB workspace to a file.

3. *Set path* allows commonly used files in the set directory to be searched.
4. *Preferences* allows the user to specify the window's color and font size, number display formats, editor used, print options, figure copy options and many others.

A.1.2 Constructing and manipulating matrices

MATLAB uses simple programming notations to execute mathematical statements. MATLAB syntax is expressed in a matrix-style operation. In MATLAB, a matrix is an array of numbers. A scalar is a one-by-one matrix, and a matrix with one column or row is a vector. The user can represent and manipulate scalars, vectors and matrices in the MATLAB workspace as examples, summarized in Table A.1.

MATLAB does not require any data-type declarations, and we even don't need to specify the dimension of the matrix or vector. When MATLAB encounters a new variable, it automatically creates the variable in the Workspace with an appropriate amount of storage. MATLAB is case sensitive; it differentiates between variable names with upper case from those with lower case.

If we type a statement listed in the table and press **Enter**, MATLAB automatically displays the results on screen. (In this sense, MATLAB is like a calculator with advance functionality.) To suppress the display, we can simply end a statement with a semicolon. In this case, MATLAB performs the computation, assigns the results to variables and does not display the output. This is particularly useful when we generate large matrices and execute MATLAB programs with many variables and intermediate results.

A.1.3 The colon operator

The colon operator ‘:’ is one of the most useful MATLAB operators. In Table A.1, we use the colon operator to construct vectors and to access submatrices. The general syntax of using colon operator is

```
z = StartValue:IncrementValue:EndValue;
```

For example, in Table A.1, we use

```
z = 0:2:10;
```

to generate vector `z`, which contains six elements: 0, 2, 4, 6, 8, 10. Note that if the increment is 1, it can be skipped as `z = StartValue:EndValue`. For example, In Table A.1, we use `z = 1:10` to create a vector that contains 10 elements from 1 to 10.

In signal processing, the colon operator can be used to implement a decimator (or downsample) for multirate signal processing. A decimator with a decimation factor D , where D is a positive integer, reduces the sampling rate of an input sequence (see Section 2.1 for more details) by a factor of D . The decimator is implemented by keeping every D th sample of the input sequence, while removing other samples. For example, this operation can be easily implemented in MATLAB as

```
x = randn(1,1000);
D = 4;
x_D = x(1:D:end);
```

Table A.1 Basic MATLAB commands for constructing and manipulating matrices

Input	Comment
<code>x = [1 2 3];</code>	<code>x</code> is defined as a row vector with three elements. The elements in a row vector are separated with blanks (or alternatively, commas) and surrounded with squared brackets. A semicolon ‘;’ is used to indicate the end of a statement and to suppress the display of vector <code>x</code> .
<code>y = [1; 2; 3];</code>	<code>y</code> is defined as a column vector with three elements. The elements in a column vector are separated with semicolons.
<code>y = [1 2 3]';</code>	We can transpose a real-valued row vector using an apostrophe operator ‘’ to convert a row vector into a column vector. The apostrophe operator performs a Hermitian transposition (i.e. a complex-conjugate transposition).
<code>y = [1 2 3]';;</code>	The apostrophe-dot operator ‘.’ transposes a matrix without affecting the signs of the imaginary parts of complex numbers.
<code>x = [1 2 3; 4 5 6; 7 8 9];</code>	<code>x</code> is defined as a 3-by-3 matrix. A semicolon inside the brackets [] indicates the end of each row.
<code>x*x'</code>	Inner-product operation (assume <code>x</code> is a 1-by- N row vector; thus <code>x'</code> is an N -by-1 column vector).
<code>x.*x</code>	Element-by-element (of the vector <code>x</code>) multiplication.
<code>x+x</code>	Element-by-element addition.
<code>x+1</code>	MATLAB allows addition between matrix and scalar, where all elements in matrix <code>x</code> are added by 1 in the example.
<code>x*x</code>	Matrix-vector multiplication (Note: their dimensions must be compatible; e.g. if <code>x</code> is an N -by- N matrix, then <code>x</code> must be an N -by-1 column vector).
<code>z = 1:10;</code>	<code>z</code> is defined as a row vector with 10 elements [1 2 3 4 5 6 7 8 9 10]. The colon operator ‘:’ separates the start and end values. The default increment is 1.
<code>z = 0:2:10;</code>	<code>z</code> is defined as a row vector with six elements [0 2 4 6 8 10] by using a skip value 2 in the statement.

(continued overleaf)

Table A.1 (*continued*)

Input	Comment
<code>X(1, :)</code>	Select the first row of <code>x</code> .
<code>X(:, 1)</code>	Select the first column of <code>x</code> .
<code>X(end, :)</code>	Select the last row of <code>x</code> .
<code>X(:, end)</code>	Select the last column of <code>x</code> . The keyword <code>end</code> refers to the last row or column.
<code>X(1:2, 1:2)</code>	Select the upper 2-by-2 submatrix of <code>x</code> .
<code>X^2</code>	Perform x^2 (i.e. $x*x$ of the square matrix <code>x</code>).
<code>X.^2</code>	Perform an element-by-element square. In general, the operator ' <code>^</code> ' performs exponentiation.
<code>[x, x]</code>	Concatenate the row vector.
<code>[x; x]</code>	Place the vector in the next row.
<code>[x; X]</code>	Place the matrix in the next rows.
<code>ones(2, 3)</code>	Create a 2-by-3 vector of all elements = 1.
<code>zeros(4, 3)</code>	Create a 4-by-3 matrix of all elements = 0.
<code>rand(2, 3)</code>	Create a 2-by-3 matrix of uniformly distributed random numbers.
<code>randn(2, 3)</code>	Create a 2-by-3 matrix of normally distributed random numbers.
<code>x(1) = [];</code>	Remove the first element of vector <code>x</code> .
<code>x(2, :) = [];</code>	Remove the second row of matrix <code>x</code> .
<code>flipud(x)</code>	Flip the matrix <code>x</code> in the up/down direction.
<code>fliplr(x)</code>	Flip the matrix <code>x</code> in the left/right direction.
<code>diag(x)</code>	Return a vector that contains the main diagonal of matrix <code>x</code> .
<code>diag(x)</code>	Construct a diagonal matrix (i.e. a matrix with all its off-diagonal elements equal to zero) with its diagonal elements given by vector <code>x</code> .
<code>Eye(2, 3)</code>	Create a 2-by-3 matrix with ones on its main diagonals.
<code>foo = sum(X);</code>	Sum all columns of <code>x</code> . The variable <code>foo</code> is a row vector.
<code>foo = sum(X, 1);</code>	Same as <code>sum(x)</code> .
<code>foo = sum(X, 2);</code>	Sum all rows of <code>x</code> . The variable <code>foo</code> is a column vector.
<code>foo = sum((sum(X)));</code>	Sum all elements of <code>x</code> . <code>foo</code> is a scalar.
<code>mean(X, dim)</code>	These three functions return the average (<code>mean</code>),
<code>var(X, dim)</code>	variance (<code>var</code>) and standard deviation (<code>std</code>) of each
<code>std(X, dim)</code>	column (if <code>dim</code> = 1) or row (if <code>dim</code> = 2) of <code>x</code> . The syntax is similar to the function <code>sum</code> .

Table A.1 (Continued)

Input	Comment
<code>foo = eye(3);</code>	<code>foo</code> is a 3-by-3 identity matrix. The command <code>eye(N)</code> creates an N -by- N identity matrix.
<code>size(x)</code>	Return a two-element row vector <code>[M N]</code> , where <code>M</code> indicates the number of rows while <code>N</code> indicates the number of columns in the M -by- N matrix <code>x</code> .
<code>size(x,dim)</code>	Return the number of columns (if <code>dim = 1</code>) or rows (if <code>dim = 2</code>) in <code>x</code> .
<code>foo = randn(size(x));</code>	<code>foo</code> is a random matrix that has the same numbers of columns and rows as in <code>x</code> .
<code>ones(4,1)*randn(1,4)</code>	<code>ones(4,1)</code> returns a four-element column vector of all ones and <code>randn(1,4)</code> returns a four-element random vector (row). Multiplication of these two vectors results in a 4-by-4 matrix with the first random vector duplicated in each row of the matrix.
<code>repmat(randn(1,4),4,1)</code>	Use <code>repmat</code> (replicate and tile an array) to create the same random matrix.
<code>y.*diag(y)</code> <code>y.*y</code>	Element-by-element multiplication (<code>y</code> is a column vector as used in the previous example).

In this example, the row vector `x` represents a random signal with 1000 samples. We then select every D th samples of `x` using `[1:D:end]`, where `D` indicates the incremental step and the keyword `end` indicates the end of the row vector `x`. We do not explicitly specify the sampling rate in this example. Consider that both `x` and `x_D` span the same duration; the sampling rate of the signal in vector `x_D` is therefore $(1/D)$ th that of the input sequence `x` since the number of random samples in `x_D` is $(1/D)$ th of the original sequence `x`.

The keyword `end` is particularly useful to access a specific sample relative to the end of a vector. For example,

```
x = randn(1,10);
y = x(end-1);
```

The second-last element of vector `x` is assigned to variable `y`, which is a scalar.

The colon operator works for both column and row vectors. However, in most situations, we work on column vectors. (For example, we always define the tap-weight vector and input signal vector in adaptive filters as column vectors (see Section 1.2 for details). This can be accomplished with the colon operator as follows:

```
y = x(:);
```

The variable `y` is defined as a column vector regardless of whether `x` is a column or row vector. If `x` is a matrix, `y` is then a column vector obtained by concatenating columns of `x`.

A.1.4 Data types

The data types used to define variables will affect the precision, dynamic range and arithmetic errors in computation. By default, all MATLAB computations are carried out in a double-precision floating-point format (64-bit). Nevertheless, data can also be stored in single precision (32-bit) or integer (8-bit, 16-bit and 32-bit). Lower precision requires less memory, but it introduces larger quantization errors in the computation. In some cases, we might deliberately reduce the precision, not to ease the memory requirement but to analyze the effects of quantization errors on a given algorithm in finite word-length implementation. Table A.2 lists data formats supported in MATLAB.

The precision of variables can be specified using the data type as a prefix. For example,

```
x = [1 2 3]; % A double precision row vector
y = single(x); % Convert to single precision
z = uint8(x); % Convert to unsigned int (8-bit)
```

Note that we can find the number of bytes used in representing an array by looking at the Workspace window. Besides storing numbers in the required precision, we can also change the displayed numeric format without affecting the data. This can be done by specifying the display format in File → Preferences (select Command Window, Text display, Numeric format). Alternatively, we can use the `format` function to control the numeric format of the values displayed in the Command Window. Type `helpwin format` in the Command Window to see available display formats.

A.1.5 Working with strings

Strings are sequences of characters. For example, we might use a string to keep an account on the name of the algorithm used in an experiment. A string can be entered in MATLAB using single quote. For example, the statement

```
s = 'Subband';
```

creates a character array with seven elements. Individual elements of `s` can be accessed via `s(1), s(2), ..., s(7)`, similar to that for numeric arrays. A larger string can be formed by concatenating shorter strings as follows:

```
saf = [s, 'adaptive filter'];
```

Table A.2 Data types supported in MATLAB

Precision	Description
<code>single</code>	Single precision floating-point (32-bit)
<code>double</code>	Double precision floating-point (64-bit)
<code>int8, uint8</code>	Signed, unsigned integer (8-bit)
<code>int16, uint16</code>	Signed, unsigned integer (16-bit)
<code>int32, uint32</code>	Signed, unsigned integer (32-bit)

Alternatively, we can use the `sprintf` function to construct a string from shorter strings and numeric scalars as follows:

```
year = 2009;
saf_year = sprintf('%s, %d', saf, year);
```

In the above statement, `%s` and `%d` are the conversion specifiers. The first `%s` indicates that `saf` is a string, while the second `%d` indicates that `year` is a numeric scalar. The `sprintf` function (write formatted data to string) is frequently used in the MATLAB programs. For example, in `SYSID.m` (see Chapter 1), we use the following statement:

```
disp(sprintf('LMS, step size = %.5f', mulms));
```

to display the algorithm and step size being used by the program at run time. We also use it to display some important simulation results, such as

```
disp(sprintf('Total time = %.3f mins', toc/60));
```

In some cases, we can convert a string to a number as follows:

```
year_string = '2009';
year_number = str2num(year_string);
```

We create a character array `year_string`. To assign the value to a numeric scalar `year_number`, we convert the string to a number using the `str2num` function.

A.1.6 Cell arrays and structures

MATLAB allows data to be grouped under a single variable known as the cell array. For example, we can group three different variables as an array `A` and specify the cell array using the curly braces `{}` as follows:

```
A = {50, 'Smith', eye(3)};
```

In the above statement, we create a 1-by-3 cell array. To access the contents of cells, we use subscripts in curly braces. (Note: subscripts always start from 1 in MATLAB.) The first cell `A{1}` is a scalar, the second cell `A{2}` contains the string of a character and the third cell `A{3}` is a 3-by-3 identity matrix (see Table A.1 for more MATLAB commands for constructing and manipulating matrices). It should be mentioned that cell arrays are not pointers since they contain exact copies of matrices or strings assigned to them.

A cell array can be used to store a sequence of matrices or character strings of different sizes. For example,

```
A = {'LMS', 'NLMS', 'SAF'};
for k = 1:length(A)
    disp(sprintf('Algorithm: %s', A{k}));
end
```

The cell array `A` contains three (given by `length(A)`) character strings (i.e. the name of an algorithm such as LMS, NLMS and SAF) of different sizes. Storing those strings in a cell array simplifies the programming. For example, a `for` loop can be used to access the

individual cell in `A` one by one. Similar to the C language, the `for` loop is commonly used to repeat statements a specific number of times. The general form of a `for` statement is

```
for variable = expression
    statement;
    ...
    statement;
end
```

In the following example, `A` is a cell array containing four 4-by-4 random matrices. Alternatively, we may store the random matrices using a three-dimensional matrix `B`. (Note: this may not be possible if `A` contains matrices of different sizes.) Thus,

```
A = {randn(4,4),randn(4,4),randn(4,4),randn(4,4)};
for k = 1:length(A)
    B(:,:,k) = A{k};
end
```

The three-dimensional matrix `B` (and all other numeric arrays) is stored in a contiguous block in the computer memory. On the other hand, cell elements of `A` are stored as separate contiguous blocks in the memory. In other words, with a cell array, we could split a three-dimensional matrix into three parts to be stored at different locations in the memory. This is particularly useful for handling large matrices as the size of the largest contiguous block (and therefore the largest matrix) is capped by a certain value in MATLAB depending on the operating systems (Windows, Linux, etc.).

An alternate form in specifying the above cell array `A` by name is to use the structure array as follows:

```
S.name = 'Smith'; % X{2}
S.age = 50; % X{1}
S.matrix = eye(3); % X{3}
```

`S` is a structure array with three fields: `name`, `age` and `matrix`. We may also initialize a structure with the following single statement:

```
S = struct('name','Smith','age',50,'matrix', eye(3));
```

To access individual fields in `S`, simply type `S.name`, `S.age` or `S.matrix`. We might expand the structure for a new entry as follows:

```
S(2).name = 'Rebecca';
S(2).age = 33;
S(2).matrix = randn(3);
```

Similar to a cell array, a structure array provides additional flexibility in writing MATLAB programs. For example, we can access individual entries and fields of a structure with simple indexing:

```
for k = 1:length(S)
    disp(sprintf('Name: %s, Age: %d',...
    S(k).name,S(k).age));
end
```

Notice that we may use ellipses to break a statement into smaller parts when it becomes too long to fit into a single line.

A.1.7 MATLAB scripting with M-files

We can directly enter and execute MATLAB commands presented earlier in the Command Window. Since we usually need to perform a similar task repeatedly and use several commands in sequence for a specific task, a better way is to write those commands in a text file as a MATLAB program. We call these text files that contain a sequence of commands (or statements) in MATLAB language as M-files with filename extension **.m**. The M-file editor (or other text editors) can be used to write and save MATLAB commands into an M-file with an ‘.m’ extension. The M-file editor can be activated by

clicking on either the **New M-file** icon  or the **Open File** icon .

There are two kinds of text M-files: scripts and functions. A script M-file has no input and output arguments, and contains a sequence of MATLAB statements that use the workspace data globally. To execute a MATLAB script, simply type in the name of the M-file (without extension) in the Command Window and press **Enter** (or **Return**). MATLAB executes the commands in the file in a sequential order. The user can interrupt a running program by pressing **Ctrl-C** or **Ctrl+Break** at any time. Since MATLAB scripts share a global workspace, scripts can operate on existing variables in the workspace or they can create new variables on which to operate. Any variable that they create remains in the workspace (unless it is cleared by the `clear` command), which can be used in subsequent computations.

A MATLAB function accepts input arguments and returns output arguments. A function M-file must contain the keyword `function` (must in lower case) in the first line of the file. The general syntax of the function definition is

```
function [OutputVariables] = FunctionName (InputVariables)
```

The function name `FunctionName` should be identical to the filename of the function M-file, without the ‘.m’ extension. (For the case where the function name is not the same as the filename, MATLAB uses the latter.) For example, a function `getstat.m` is listed as follows:

```
function [mu,sig,sig2] = getstat(x)
% getstat    Computes mean, variance, and std deviation
% x          Row or column vector of numbers
% mu         Mean of the elements in vector x
% sig        Standard deviation of the vector x
% sig2       Variance of the vector x
%
if nargin ~= 1
    error('Number of input argument must be one');
end
if nargin ~= 3
    error('Number of output arguments must be three');
end
mu = mean(x); % Compute mean of x
sig = std(x); % Compute standard deviation of x
sig2 = var(x); % Compute variance of x
```

As shown in the example, the function `getstat` contains three output arguments `mu`, `sig` and `sig2`, which are enclosed in square brackets. The input argument `x` is enclosed in parentheses. The next several lines start with a percent sign % and after the first line and up to the first blank or executable line are comments that provide the help text; i.e. if we type `help getstat` in the Command Window and the `getstat.m` file is in the search path, MATLAB displays those help texts. Comments can also be inserted anywhere inside an M-file by starting with %. It is good programming practice to document the code with the help text and comments.

The first few lines of the code check the number of input and output arguments using the `nargin` and `nargout` (number of arguments provided as input and output, respectively) variables in the function M-files. These variables come in handy during programming when we want to make sure that the correct numbers of input and output arguments are given by users.

Different from script M-files, all variables inside the function M-file are created in the local workspace and are not shared with the calling script M-file in the global workspace. Therefore variables used in a function M-files will never be mixed up with those in the global workspace.

A.1.8 Plotting in MATLAB

MATLAB is known for its outstanding graphics capabilities. Both 2D and 3D graphics can be plotted and manipulated using some commonly used commands, summarized in Table A.3. The most commonly used and flexible graphic function is `plot`, which creates a 2D line plot. The general syntax is

```
plot(x, y, 'styleOption');
```

which plots vector `y` versus vector `x`, and the `styleOption` specifies the various colors, line types, line markers, etc. These options are summarized in Table A.4. For example,

```
plot(x,y,'-.g*');
```

where ‘-.’ specifies the dashdot line, ‘g’ specifies green color and ‘*’ specifies a star mark on those sample points on the line curve. In addition, MATLAB supports the following:

```
legend(string1,string2,string3, ...);
```

to put a legend on the current plot using the specified strings as labels. For example, in `SYSID.m`, the following statements:

```
plot(0:iter-1,MSELms,0:iter-1,MSEnlms);
legend('MSE(LMS)', 'MSE(NLMS)');
```

to identify the first line plot is the MSE of the LMS algorithm and the second line is for the NLMS algorithm.

In Example 1.1 of Chapter 1, we plot a 3D plot using the `meshc` function. We first define the indices for the x and y axes as follows:

```
x = 0:0.1:4;
y = -4:0.1:0;
```

Table A.3 MATLAB commands and examples for 2D and 3D plots

Input	Comment
<code>t = 0:0.001:1;</code>	Set up the time index in vector t ; start from 0 to 1 with an incremental step of 0.001.
<code>x = exp(-10.*t);</code>	Create a vector x using the exponential function <code>exp</code> .
<code>plot(t,x);</code> <code>plot(t,x,'ro');</code>	Line plot of vector x versus vector t . Line plot using red color of the line with a circle to indicate each data point.
<code>figure;</code> <code>stem(t,x);</code> <code>stem(t,x,'filled');</code>	Create a new figure plot. Discrete sequence plot terminated with an empty circle or a full circle with the option ‘ <code>filled</code> ’.
<code>subplot(M,N,I);</code>	Divide the figure window into M (rows) x N (columns) smaller figure windows and select the I th figure window for display.
<code>subplot(1,2,1);</code> <code>plot(t1,x1);</code>	Divide the figure window into two smaller windows in a row and plot the first one with x_1 versus t_1 .
<code>subplot(1,2,2);</code> <code>plot(t2,x2);</code>	Plot the second figure (on the right side) with x_2 versus t_2 .
<code>close;</code> <code>close all;</code>	Close the current active figure. Close all the figures.
<code>x = 0:1:50;</code>	X-axis index from 0 to 50 with an incremental step of 1.
<code>y1 = sin(2*pi*x/8);</code> <code>y2 = cos(2*pi*x/8);</code> <code>plot(x,y1,x,y2);</code> <code>plot(x,y1,'r');</code> hold on; <code>plot(x,y2,'b');</code> hold off;	Generate sinewave sequence in vector y_1 . Generate cosinewave sequence in vector y_2 . Plot sine and cosine waves in the same plot. Plot sine (with red color) and cosine (with blue color) waves, one at a time in the same plot.
<code>axis([0 10 -1 1]);</code>	Set the display limits for the x axis and y axis using the general syntax <code>axis([x_min x_max y_min y_max])</code> .
<code>grid on;</code>	Show grid lines.
<code>xlabel('Time')</code> <code>ylabel('Amplitude')</code> <code>title('Two sine waves')</code>	Annotate the x axis as ‘Time’. Annotate the y axis as ‘Amplitude’. Specify the title of the plot as ‘Two sine waves’.
<code>t = 0:pi/50:5*pi;</code> <code>plot3(sin(t),cos(t),t);</code>	Prepare the time-index values. Perform a 3D plot whose coordinates are elements given in the x, y and z vectors.
<code>axis square;</code> <code>grid on;</code>	Change to a square plot with grid.

(continued overleaf)

Table A.3 (*continued*)

Input	Comment
[X, Y] = meshgrid([-2:0.1:2]);	Transform the vector into an array x and y that can be used for 3D surface plots.
Z = X.*exp(-X.^2-Y.^2);	Generate another array z.
mesh(X, Y, Z);	Mesh surface plot.
surf(X, Y, Z);	Surface plot.
meshc(X, Y, Z);	Surface plot with contour beneath.
meshz(X, Y, Z);	Surface plot with curtain.
pcolor(X, Y, Z);	Pseudo-color plot.
surfl(X, Y, Z);	3D shaded surface with lighting.

Table A.4 Style options available for MATLAB plots

Colors	Line types	Line markers
b	Blue	.
g	Green	o
r	Red	x
c	Cyan	+
m	Magenta	*
y	Yellow	s
k	Black	d
w	White	

For each pair of x (corresponding to the first tap-weight) and y (the second tap-weight) indices, we compute the corresponding MSE value and save it in the respective element of matrix z for the z axis as follows:

```

Z = zeros(length(x),length(y));
for m = 1:length(x)
    for n = 1:length(y)
        w = [x(m) y(n)]';
        Z(m,n) = 3-2*[0.5272; -0.4458]'*w...
            + w'*[1.1 0.5; 0.5 1.1]*w;
    end
end

```

The first `for` loop iterates through each element of vector x , while the second `for` loop does the same for vector y . The 3D plot is created using the following command:

```
meshc(x,y,Z');
```

The generated 3D MSE plot is shown in Figure A.2.

We can change the orientation (viewing angle) using the `view` command to obtain Figure 1.4. The general syntax is given as

```
view(az, el);
```

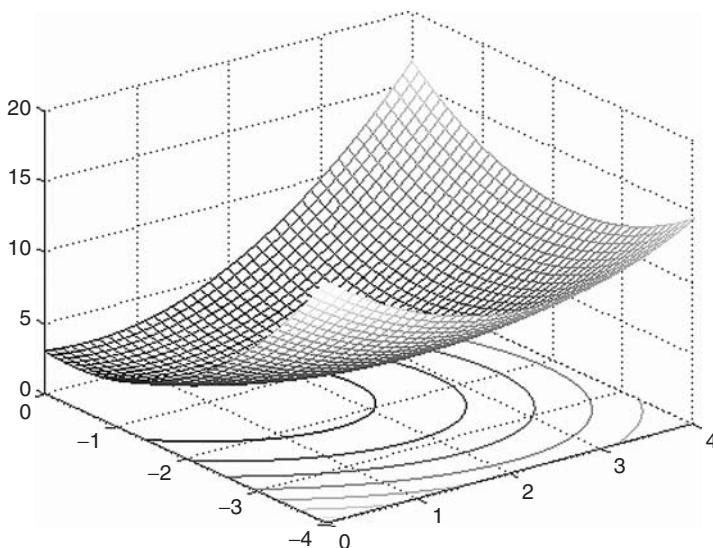


Figure A.2 An example of a three-dimensional plot

where `az` is the azimuth (or horizontal) rotation and `el` is the vertical elevation in degrees. For example, `az = 0` and `el = 90` is directly overhead, which is the same as a 2D view; `az = el = 0` looks directly up the first column of the matrix and `az = 180` is behind the matrix.

More recent versions of MATLAB (version 6 and higher) provide a set of powerful editing tools to edit the graphics. It supports a point-and-click editing mode to modify the plots. Graphical objects can be selected by clicking on the object and multiple objects can be selected by **shift-click**. Several editing and navigating features are highlighted in Figure A.3 (select *View* → *Plot Edit Toolbar* and → *Figure Toolbar* to display the toolbars). Notably, the Data Cursor allows users to look at the values at specific points on the plot. The Pan tool moves (by click and drag) the graph within a MATLAB figure window.

Besides selecting, moving, resizing, changing color, cutting, copying and pasting the graphic objects, MATLAB also provides tools for zoom-in, zoom-out, basic data fitting and displaying data statistics. These tools can be executed from the **Tools** pull-down menu. Once the diagram has been edited, the user can export the diagram into different graphical file types such as `bmp`, `tiff`, etc., or save it as a MATLAB `fig` file. A more detailed explanation can be found in the MATLAB help.

A.1.9 Other useful commands and tips

There are many useful MATLAB functions that are commonly used for performing linear algebra operations and Fourier analysis of data. These functions are summarized in Table A.5. More signal processing functions will be discussed in the section on Signal Processing Toolbox. Table A.6 lists some useful MATLAB tips and commands.

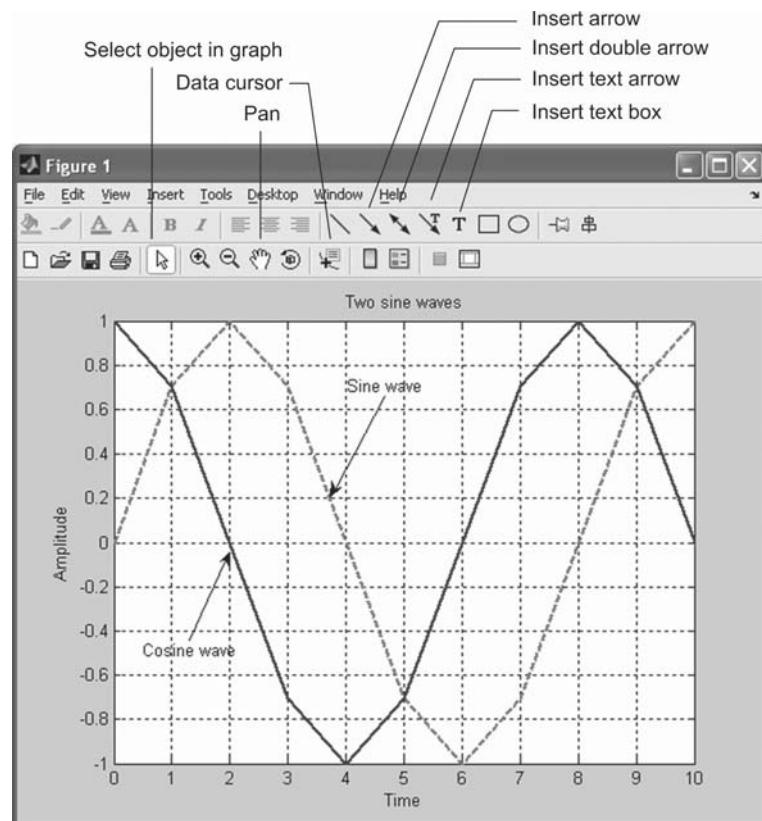


Figure A.3 An example of graphic editing

Table A.5 MATLAB functions for linear algebra and Fourier analysis

Functions	Comments
norm	Matrix or vector norm
rank	Matrix rank
det	Determinant of matrix
trace	Sum of diagonal elements
null	Null space
orth	Orthogonalization
eig	Eigenvalues of matrix
fft	Fast Fourier transform (FFT)
ifft	Inverse FFT (IFFT)
abs	Magnitude
angle	Phase angle
unwrap	Unwrap phase angle in radians

Table A.6 Some useful MATLAB tips and commands

Commands	Comments
<code>clear all;</code>	It is advisable to start a script M-file to clear all variables created earlier in the workspace by other M-files, if these variables are not required anymore.
<code>find</code>	Find indices and values of nonzero elements. For example, the command <code>length(find(x>0))</code> finds the number of positive elements in the matrix <code>x</code> .
<code>linspace</code>	Generate linearly spaced vectors. Similar to that using the colon “ <code>:</code> ” operator, but gives direct control over the number of points.
<code>exist</code>	Check if variables and functions are defined. We may also use <code>exist</code> to check if a file exists.
<code>wavread</code>	Read a Microsoft WAVE (.wav) sound file. Note that the command <code>[y, Fs, Nbits] = wavread(FileName);</code> also returns the sample rate <code>Fs</code> in Hz and the number of bits per sample <code>Nbits</code> used to encode the data samples in the wavefile.
<code>wavwrite</code>	Write a Microsoft WAVE (.wav) sound file. The general syntax is <code>wavwrite(y, Fs, Nbits, FileName);</code> which writes sound samples in vector <code>y</code> to a Windows WAVE file <code>FileName</code> with a sample rate of <code>Fs</code> Hz and <code>Nbits</code> number of bits per data sample.
<code>sound</code>	Convert vector into sound and send to the sound card. The general syntax is <code>sound(y, Fs)</code> , which sends the signal in vector <code>y</code> with sample frequency <code>Fs</code> to the sound card on computers that support sound. Values in <code>y</code> are assumed to be in the range $-1.0 \leq y \leq 1.0$. Values outside that range are clipped.
<code>soundsc</code>	<code>soundsc</code> scales the data before plays the data as sound. This command avoids clipping by using <code>sound</code> .
<code>wavplay(y, Fs)</code>	Output the signal in vector <code>y</code> with sample frequency of <code>Fs</code> Hz to the Windows wave audio device. Standard audio rates are 8000, 11025, 22050 and 44100 Hz.
<code>delete</code>	Delete files. It is different from <code>clear</code> , which deletes variables from workspace.
<code>round, ceil, floor, fix</code>	Round toward nearest integer, infinity, negative infinity and zero.
<code>Ctrl+c</code>	Copy selected text in M-file editor.
<code>Ctrl+x</code>	Cut selected text in M-file editor.
<code>Ctrl+v</code>	Paste text in M-file editor.
<code>Ctrl+r</code>	Comment selected text in M-file editor. This comes in handy when we wish to temporarily comment a large piece of code.
<code>Ctrl+t</code>	Uncomment selected text in M-file editor.
<code>Ctrl+[</code>	Decrease indent in M-file editor.
<code>Ctrl+]</code>	Increase indent in M-file editor.
<code>Ctrl+z</code>	Undo previous action in M-file editor.
<code>Ctrl+y</code>	Redo previous action in M-file editor.

A.2 Signal processing toolbox

As we develop more M-files, we can organize them into different folders. Such collections of M-files for solving some application-specific problems are referred to as toolboxes. As mentioned earlier, MATLAB comes with a comprehensive range of toolboxes (though MATLAB toolboxes are sold separately) written by experts in their field. Commonly used DSP-related toolboxes include Signal Processing Toolbox, Filter Design Toolbox, Wavelet Toolbox, Communication Toolbox, Image Processing Toolbox, etc.

This section summarizes some useful signal processing functions and tools in the Signal Processing Toolbox. These functions can be categorized in the following DSP groups:

- Digital filter design, analysis and implementation
- Analog filter design, analysis and implementation
- Linear system transformations
- Windowing functions
- Spectral analysis
- Transforms
- Statistical signal processing
- Parametric modeling
- Linear prediction
- Multirate signal processing
- Waveform generation

Readers can refer to the user guides, available through the MATLAB help browser (type `helpwin` in the Command Window) and website, for more details.

A.2.1 Quick fact about the signal processing toolbox

The Signal Processing Toolbox contains many functions for performing DSP operations. In addition to these DSP functions, the toolbox also contains several useful interactive (graphical user interface) tools including (i) a Signal Processing Tool (SPTool), which provides interactive tools in analyzing and filtering signals, (ii) a Filter Design and Analysis Tool (FDATool), which provides advanced filter design tools for designing digital filters, quantizing filter coefficients and analyzing the quantization effects, and (iii) a Window Design and Analysis Tool (WINTool), which supports design and analysis of windows. These tools will be explained in the following sections.

Table A.7 lists some commonly used signal-processing functions in the toolbox. For examples, `fir1`, `filter`, `upfirdn`, `remez` and `fft` are used frequently in this book. This list is grouped under different categories. To learn the detailed usage of these functions, simply type

```
help functionName;
```

Table A.7 Some commonly used functions provided in the Signal Processing Toolbox

Filter analysis	Description
freqz freqzplot grpdelay impz unwrap zplane	Frequency response of a digital filter Plot frequency response data Group delay of a filter Impulse response of a digital filter Unwrap phase angle Zero-pole plot
Filter implementation	Description
conv deconv fftfilt filter filtfilt latcfilt medfilt1 sgolayfilt upfirdn	Convolution and polynomial multiplication Deconvolution and polynomial division FFT-based FIR filtering using overlap-add FIR or IIR filter filtering Zero-phase digital filtering Lattice and lattice ladder filtering 1D median filtering Savitzky-Golay filtering Upsample, FIR filtering and downsample.
FIR filter design	Description
convmtx cremez fir1 fir2 firls firrcos intfilt kaiserord remez	Convolution matrix Complex and nonlinear phase equiripple FIR Window-based FIR filter design Frequency sampling-based FIR filter design Least-square linear-phase FIR filter design Raised cosine FIR filter design Interpolation FIR filter design FIR filter design using a Kaiser window Parks-McClellan optimal FIR filter design
IIR filter design	Description
bilinear butter cheby1 cheby2 ellip impinvar maxflat prony yulewalk	Bilinear transformation Butterworth filter design Chebyshev type I filter design Chebyshev type II filter design Elliptic filter design Impulse invariance method Generalized digital Butterworth filter design Prony's method for IIR filter design Recursive digital filter design (least-square)
Windows	Description
bartlett barthannwin blackman bohmanwin boxcar	Bartlett window Modified Bartlett-Hanning window Blackman window Bohman window Rectangular window

(continued overleaf)

Table A.7 (*Continued*)

Windows	Description
chebwin	Chebyshev window
gausswin	Gaussian window
hamming	Hamming window
hann	Hann (Hanning) window
Kaiser	Kaiser window
triang	Triangular window
tukeywin	Tukey window
Transforms	Description
bitrevorder	Permute input into bit-reversed order
czt	Chirp z -transform
dct	Discrete cosine transform
dftmtx	DFT matrix
fft	1D FFT
fft2	2D FFT
fftshift	Move zeroth lag to left of spectrum
idct	Inverse discrete cosine transform
ifft	1D IFFT
ifft2	2D IFFT
Cepstral analysis	Description
cceps	Complex cepstral analysis
icceps	Inverse complex cepstrum
rceps	Real cepstrum, minimum phase reconstruction
Statistical and spectrum analysis	Description
cohere	Estimate magnitude square coherence function
corrcoef	Correlation coefficients
corrmtx	Autocorrelation matrix
cov	Covariance matrix
csd	Cross spectral density
pburg	PSD (power spectral density) estimate using Burg's method
pcov	PSD estimate using the covariance method
peig	PSD estimate using the eigenvector method
periodogram	PSD estimate using the periodogram method
pmcov	PSD estimate using the modified covariance method
pmtm	PSD estimate using the Thomson multitaper method
pmusic	PSD estimate using the MUSIC method
psdplot	Plot PSD data
pwelch	PSD estimate using Welch's method.
pyulear	PSD estimate using the Yule–Walker autoregressive method
rootmusic	Frequency and power estimation using the MUSIC algorithm

Table A.7 (continued)

Statistical and spectrum analysis	Description
<code>xcorr</code>	Cross-correlation function
<code>xcorr2</code>	2D cross-correlation
<code>xcov</code>	Covariance function
Linear prediction	Description
<code>ac2rc</code>	Autocorrelation sequence to reflection coefficients
<code>ac2poly</code>	Autocorrelation sequence to prediction polynomial
<code>is2rc</code>	Inverse sine parameters to reflection coefficients
<code>lar2rc</code>	Log area ratios to reflection coefficients conversion
<code>levinson</code>	Levinson–Durbin recursion
<code>lpc</code>	Linear predictive coefficients using autocorrelation
<code>lsf2poly</code>	Line spectral frequencies to prediction polynomial
<code>poly2ac</code>	Prediction polynomial to autocorrelation sequence
<code>poly2lsf</code>	Prediction polynomial to line spectral frequencies
<code>poly2rc</code>	Prediction polynomial to reflection coefficients
<code>rc2ac</code>	Reflection coefficients to autocorrelation sequence
<code>rc2is</code>	Reflection coefficients to inverse sine parameters
<code>rc2lar</code>	Reflection coefficients to log area ratios
<code>rc2poly</code>	Reflection coefficients to prediction polynomial
<code>rlevinson</code>	Reverse Levinson–Durbin recursion
<code>schurrc</code>	Schur algorithm
Multirate signal processing	Description
<code>decimate</code>	Resample at a lower sampling rate (decimation)
<code>downsample</code>	Downsample input signal
<code>interp</code>	Resample data at a higher sample rate (interpolation)
<code>interp1</code>	General 1D interpolation
<code>resample</code>	Change sampling rate by any rational factor
<code>spline</code>	Cubic spline interpolation
<code>upsample</code>	Upsample input signal
Waveform generation	Description
<code>chirp</code>	Swept-frequency cosine
<code>diric</code>	Dirichlet or periodic sinc function
<code>gauspuls</code>	Gaussian-modulated sinusoidal pulse
<code>gmonopuls</code>	Gaussian monopulse
<code>pulstran</code>	Pulse train
<code>rectpuls</code>	Sampled aperiodic rectangle
<code>sawtooth</code>	Sawtooth (triangle) wave
<code>sinc</code>	Sinc function
<code>square</code>	Square wave
<code>tripuls</code>	Sampled aperiodic triangle
<code>vco</code>	Voltage-controlled oscillator

in the Command Window to display the help text for that particular function. The Signal Processing Toolbox further provides an interactive tool that integrates the functions with the graphical user interface, which will be introduced in the next section. The signal processing functions summarized in Table A.7 together with the powerful graphical tools provide a comprehensive tool for signal processing.

Table A.7 lists seven functions that support multirate signal processing. Two functions, `decimate` and `downsample`, are used to reduce the sampling rate of discrete-time signals (as discussed in Chapter 2). For example, the following statement

```
y = decimate(x,D);
```

resamples the input sequence in vector `x` at $1/D$ times the original sample rate. The resulting vector `y` is D times shorter. By default, the `decimate` function filters the data with an 8th-order Chebyshev type I lowpass filter before resampling. Similarly, interpolation can be implemented using two functions: `interp` and `upsample`.

A.2.2 Signal processing tool

The Signal Processing Tool (SPTool) provides several tools to enable the user to analyze signals, design and analyze filters, filter the signals and analyze the spectrum of signals. The user can open this tool by typing

```
sptool;
```

in the MATLAB Command Window. The SPTool main window appears as shown in Figure A.4. There are four applications that can be accessed via the SPTool main GUI (graphical user interface). These applications are listed below.

A.2.2.1 Signal browser

The Signal Browser is used to view the input signals. Signals from the workspace or file can be loaded into the SPTool by clicking on **File → Import**. An **Import to SPTool** GUI appears (as shown in Figure A.5), which allows the user to select the data from the file or workspace. Figure A.5 shows that the ‘SpeechSample.mat’ file contains one variable with the name `un` (the MAT file can be found in the common folder on the companion CD). Set the sampling frequency to 8000 and let MATLAB know that `un` is imported as a signal by selecting the **Signal** option in the list box at the upper right corner. Click **OK** to import the data to the MATLAB workspace. To view the signal, simply highlight the signal and click **View**. The Signal Browser window is shown in Figure A.6, which allows the user to zoom-in and zoom-out the signal, read the data values via markers, display format and even play back the selected signal using the loudspeakers.

A.2.2.2 Filter designer

The **Filter Designer** is used for designing digital FIR and IIR filters. The user simply clicks on the **New** button for a new filter or the **Edit** button for an existing filter under the Filter column in the SPTool (the first window that appears after we type in the `sptool` in the Command Window) to open the **Filter Designer**, as shown in Figure A.7. The user can design lowpass, highpass, bandpass and bandstop filters using different filter design algorithms. In addition, the user can also design a filter using the **Pole/Zero Editor** to

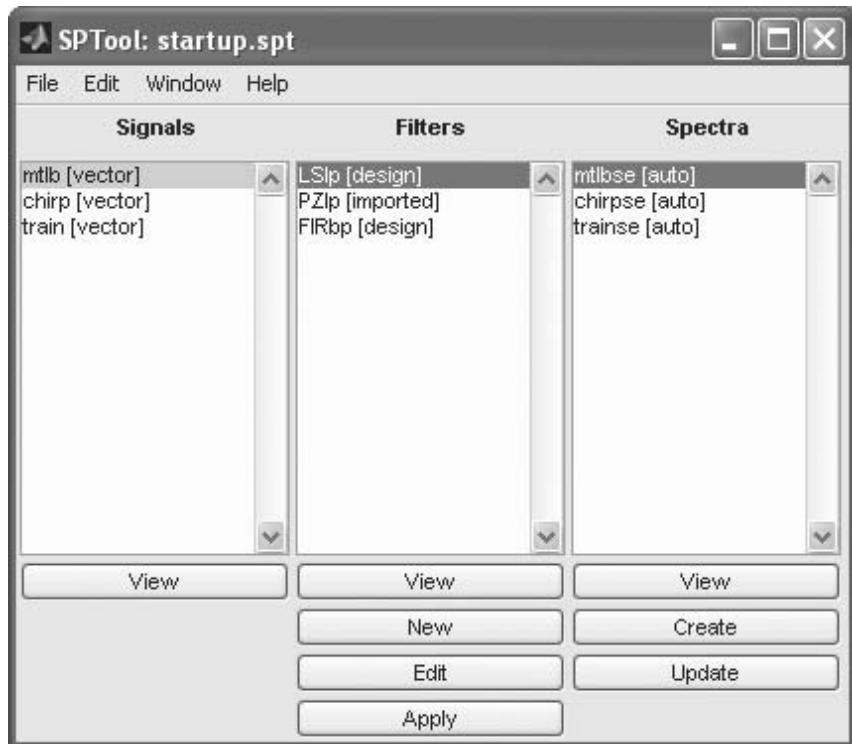


Figure A.4 SPTool provided by the Signal Processing Toolbox

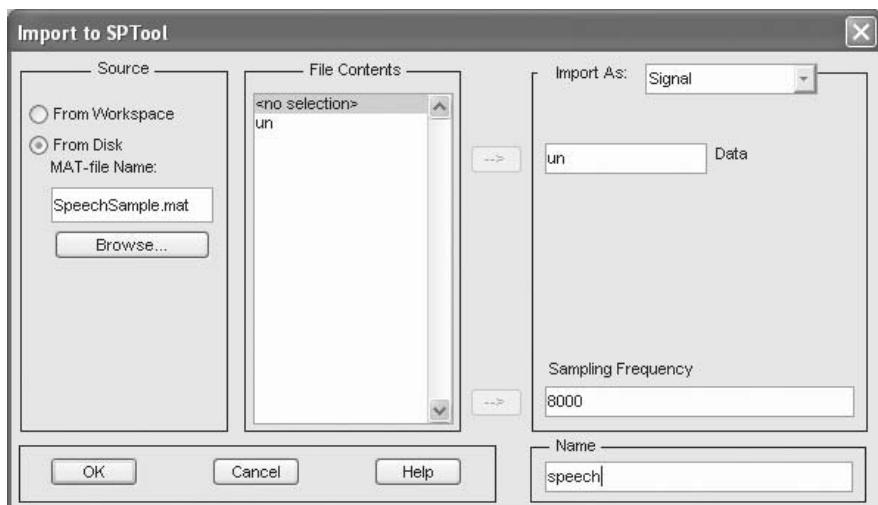


Figure A.5 Import to the SPTool GUI allows the user to select data from the file or workspace

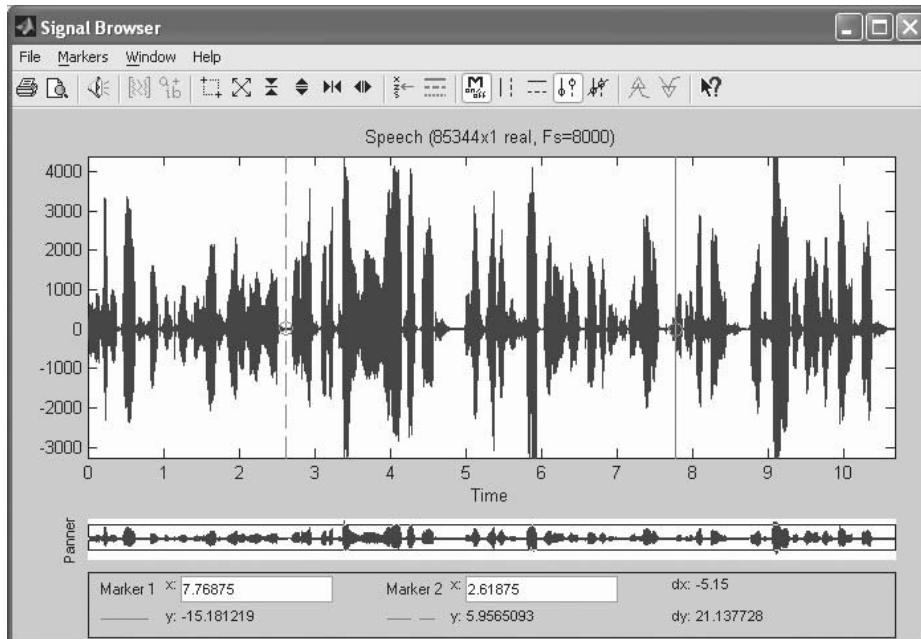


Figure A.6 Users can zoom-in and zoom-out the signal, read the data values via markers and even play back the selected signal

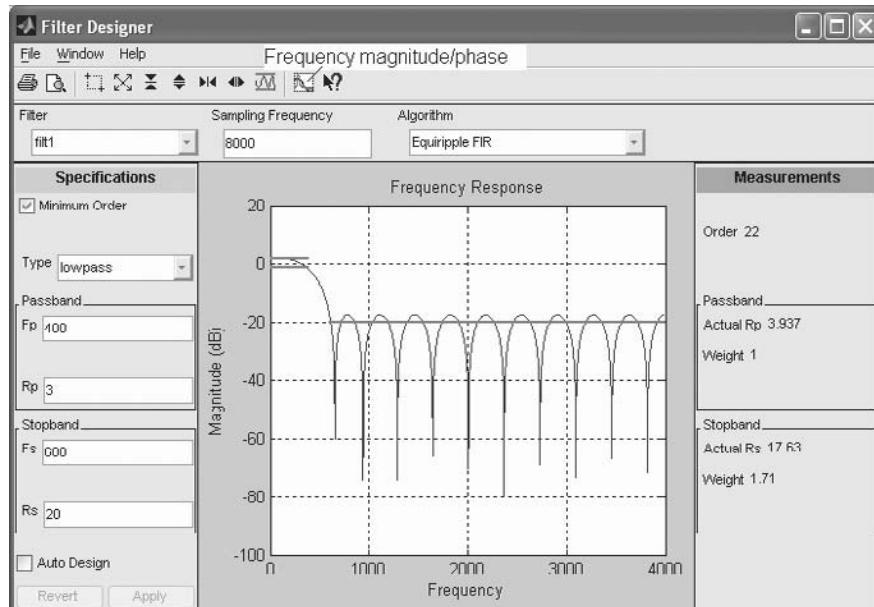


Figure A.7 The Filter Designer available under the SPTool. We set the sampling frequency to 8000 Hz in this example

place the poles and zeros graphically in the z -plane. The **Pole/Zero Editor** can be activated by making a selection in the **Algorithm** list box. The **Filter Designer** also provided a useful feature, which overlays the input spectrum on to the frequency response of the filter by clicking on the frequency magnitude/phase icon (as indicated in Figure A.7).

A.2.2.3 Filter visualization tool

Once the filter has been designed, the frequency specification and other filter characteristics can be verified by using the **Filter Visualization Tool**. Selecting the name of the designed filter and clicking on the View icon under the Filter column in the SPTool will open the window, as shown in Figure A.8. Users can analyze the filter in terms of its magnitude response, phase response, group delay, zero-pole plot (as shown in the figure), impulse response and step response.

When the filter characteristics have been confirmed, users can then select the input signal and the designed filter. Click on the **Apply** button to perform filtering and generate the output signal (which appears in the **Sig**nals column of the SPTool window). The **Apply Filter** GUI is shown in Figure A.9, which allows users to specify the variable name of the output signal. The algorithm list provides a choice of several filter structures.

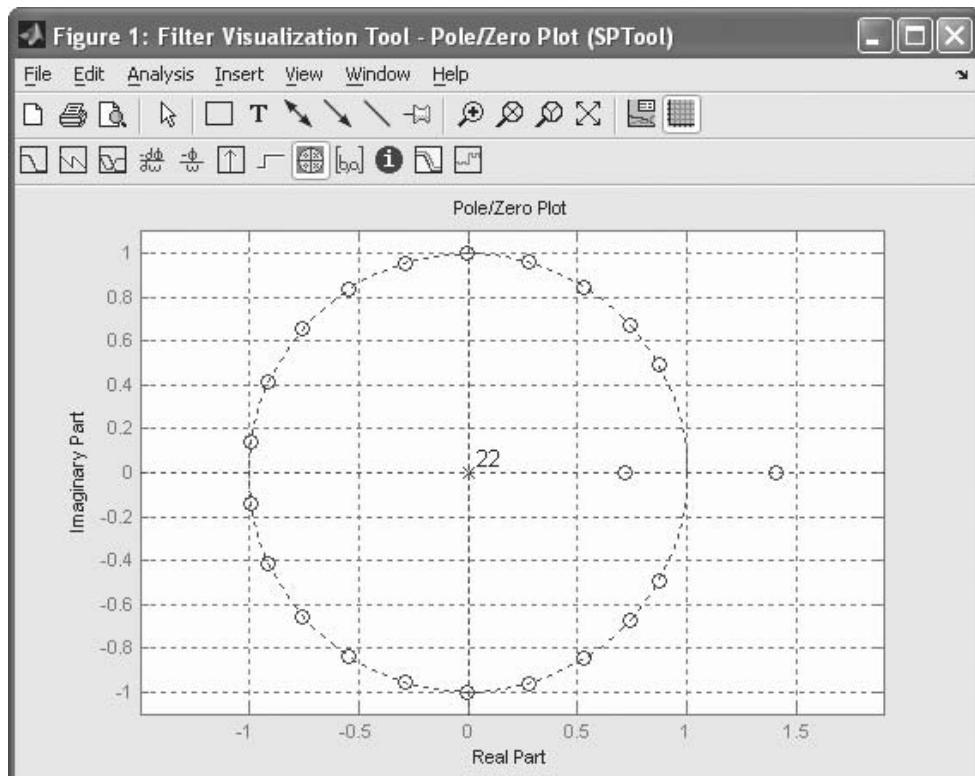


Figure A.8 The Filter Visualization Tool available under the SPTool

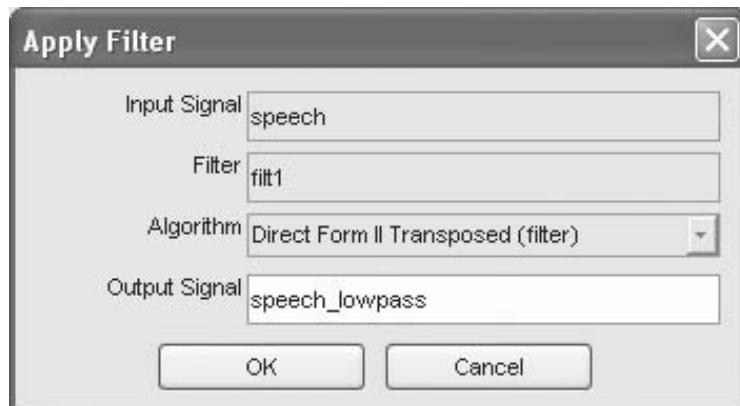


Figure A.9 The Apply Filter tool available under the SPTool

A.2.2.4 Spectrum viewer

The final GUI is the **Spectrum Viewer**, as shown in Figure A.10. Users can view the existing spectra by clicking on file names, followed by the **View** button. Select the signal and click on the **Create** button to enter the **Spectrum Viewer** window. The user can also select one of the many spectral estimation methods, such as Burg, covariance, FFT, modified covariance, MUSIC, Welch, Yule–Walker autoregressive (AR), etc., for the spectrum estimation. In addition, the size of the FFT, window functions and overlapping samples can be selected to complete the power spectrum density (PSD) estimation.

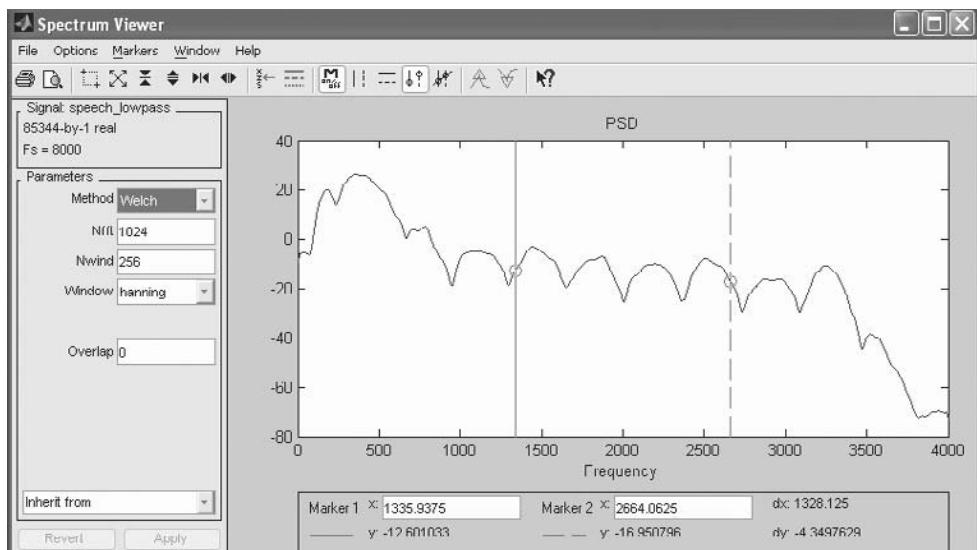


Figure A.10 The Spectrum Viewer available under the SPTool

SPTool also allows the signals, filter parameters and spectra to be exported to the MATLAB workspace or files. These saved parameters are represented in MATLAB as a structure of signals, filters and spectra.

A.2.3 Window design and analysis tool

The Signal Processing Toolbox also provides two graphical user interface tools, the Window Visualization Tool (WVTool) and the Window Design and Analysis Tool (WINTool), to design and analyze windows. In the MATLAB command window, the following command:

```
wvtool( WinName (N) ) ;
```

opens the WVTool with the time- and frequency-domain plots of the n -length window specified in `WinName`, which can be any window in the Signal Processing Toolbox.

A more powerful tool for designing and analyzing the window is the WINTool, which can be launched as

```
wintool;
```

It opens with a default 64-point Hamming window, as shown in Figure A.11.

As shown in Figure A.11, WINTool has three panels: **Window Viewer** displays the time-domain and frequency-domain representations of the selected window(s). Three window measurements are shown below the plots. (1) The leakage factor indicates the ratio of power in the sidelobes to the total window power. (2) Relative sidelobe attenuation shows the difference in height from the mainlobe peak to the highest sidelobe peak. (3) The mainlobe width (-3 dB) is the bandwidth of the mainlobe at 3 dB below the mainlobe peak.

The second panel, called **Window List**, lists the window functions available for display in the **Window Viewer**. Highlight one or more windows to display them. The four **Window List** buttons are as follows. (1) **Add a new window** allows the user to add a default Hamming window with length 64 and symmetric sampling. We can change the information for this window by applying changes made in the **Current Window Information** panel. (2) **Copy window** copies the selected window(s). (3) **Save to workspace** saves the selected window(s) as vector(s) to the MATLAB workspace. The name of the window in WINTool is used as the vector name. (4) **Delete** removes the selected window function(s) from the window list.

Current Window Information displays information about the currently active window. The active window name is shown in the **Name** field. Each window is defined by the parameters in the **Current Window Information** panel. We can change the current window's characteristics by changing its parameters and clicking **Apply**. The first parameter **Name** shows the name of the window. We can either select a name from the menu or type the desired name in the edit box. The parameter **Type** presents the algorithm for the window. Select the type from the menu. All windows in the Signal Processing Toolbox are available. The parameter **Length** indicates the total number of samples.

For example, click on the **Add a new window** button at the **Window List** panel, select **Kaiser** from the **Type** menu in the **Current Window Information** panel and click on **Apply**. Both time-domain and frequency-domain representations in the **Window Viewer** panel are updated. Highlight both **window_1** and **window_2** from the **Window List** panel; the time- and frequency-domain representations of both windows are then

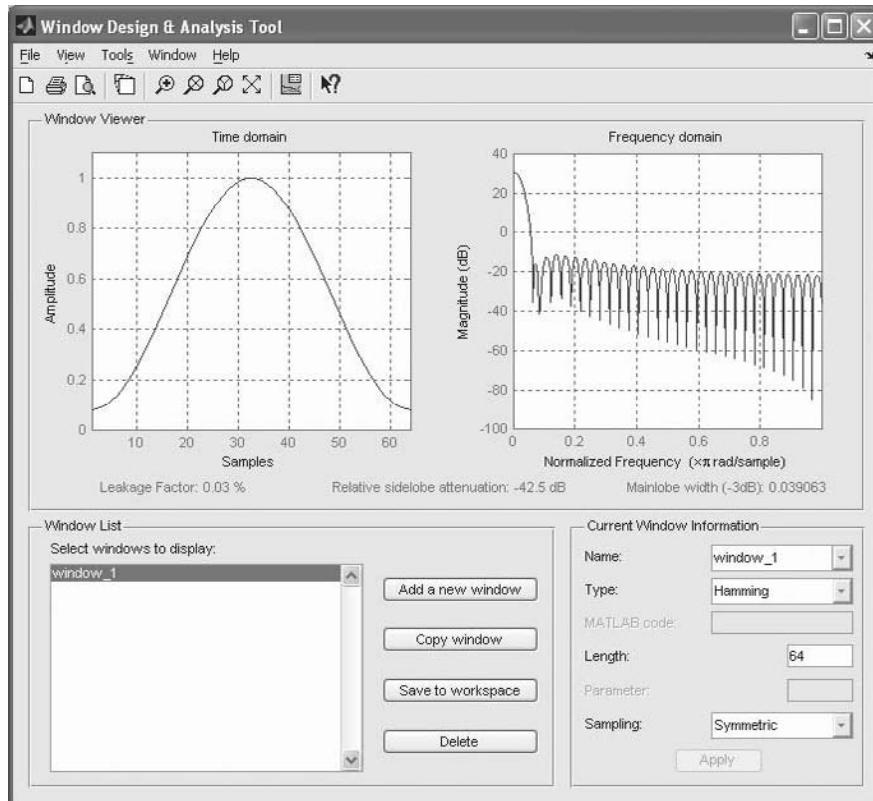


Figure A.11 The WINTool GUI with a default 64-point Hamming window

displayed. After verifying the designed window, we can use **Export** from the **File** menu to export window coefficients to the MATLAB workspace as a text file or a MAT-file.

A.3 Filter design toolbox

The Filter Design Toolbox is a collection of tools that provides advanced techniques for designing, simulating and analyzing digital filters. It extends the capabilities of the Signal Processing Toolbox presented earlier by adding filter structures and design methods for complex DSP applications. This toolbox also provides functions that simplify the design of fixed-point filters and the analysis of quantization effects.

A.3.1 Quick fact about the filter design toolbox

The Filter Design Toolbox provides the following key features for digital filter designs:

1. Advanced FIR filter design methods. The advanced equiripple FIR design automatically determines the minimum filter order required. It also provides constrained

ripple, minimum phase, extra ripple and maximal ripple designs. In addition, the least- P th norm FIR design allows the user to adjust the tradeoff between the minimum stopband energy and the minimum order equiripple characteristics.

2. Advanced IIR filter design methods. Allpass IIR filter design with arbitrary group delay enables equalization of nonlinear group delays of other IIR filters to obtain an overall approximate linear phase passband response. The least- P th norm IIR design creates optimal IIR filters with arbitrary magnitude and the constrained least- P th norm IIR design constrains the maximum radius of the filter poles to improve the robustness of the quantization.
3. Quantization. It provides quantization functions for signals, filters and FFT. It also supports quantization of filter coefficients, including coefficients created using the Signal Processing Toolbox.
4. Analysis of quantized filters. It provides analysis of the frequency response, zero-pole plot, impulse response, group delay, step response and phase response of quantized filters. In addition, it supports limit cycle analysis for fixed-point IIR filters.
5. Adaptive filter design, analysis, and implementation. It supports LMS-based, RLS-based, lattice-based, frequency-domain, fast transversal filter and affine projection algorithms. These functions will be introduced in Section A.3.3.

A.3.2 Filter design and analysis tool

The Filter Design Toolbox includes a GUI tool called the Filter Design and Analysis Tool (FDATool). This interactive tool allows the user to design optimal FIR and IIR filters from scratch, import previously designed filters, quantize floating-point filters and analyze quantization effects. In addition to the existing functions listed in Table A.6, the FDATool contains additional filter design functions listed in Table A.8.

The FDATool can be activated by typing `fdatool` in the Command Window. The FDATool GUI shown in Figure A.12 is similar to the Filter Designer (see Figure A.7) in the SPTool. The design steps and features to view the filter characteristics are also similar. However, the FDATool is a more advanced filter design tool that includes additional filter types such as the differentiator, Hilbert transform, multiband, arbitrary magnitude and group delay. The FDATool also has an additional option in converting (**Edit → Convert Structure**) the default filter structure to different structures, such as the direct-form state-space structure and its lattice equivalents.

Table A.8 Additional filter design functions in the FDATool

Filter design function	Description
<code>firlpnorm</code>	Design minimax FIR filters using the least- P th algorithm.
<code>gremez</code>	Design optimal FIR filters (Remez exchange).
<code>iirgrpdelay</code>	Design IIR filters (specify group delay in the passband).
<code>iirlpnorm</code>	Design minimax IIR filters using the least- P th algorithm.
<code>iirlpnormc</code>	Design minimax IIR filters using the least- P th algorithm, which restricts the filter poles and zeros within a fixed radius.

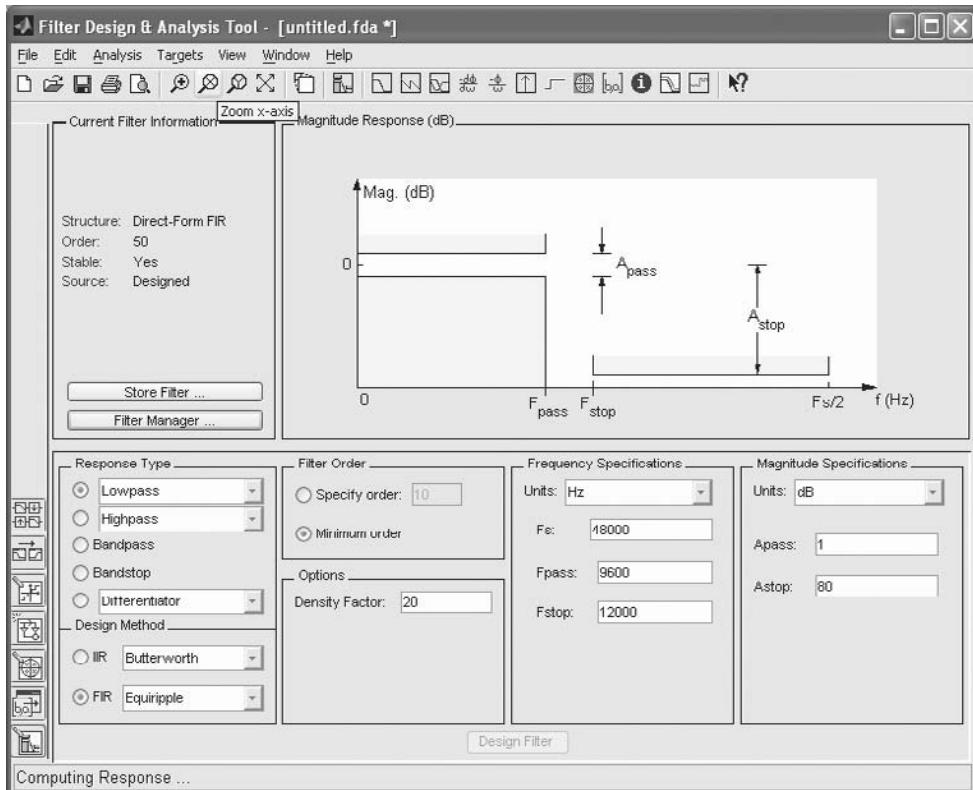


Figure A.12 The FDATool GUI

A.3.3 MATLAB functions for adaptive filtering

The introduction of adaptive filtering algorithms, applications and MATLAB implementation are presented in Chapter 1. However, the MATLAB programs listed in this book (discussed in detail in Appendix B and available in the companion CD) are written without using adaptive filtering functions provided in the Filter Design Toolbox, so the user can run those programs with MATLAB only. This section introduces the implementation of adaptive filtering algorithms for readers with the toolbox.

The MATLAB functions in the Filter Design Toolbox support adaptive FIR filter structure using the LMS-based, RLS-based and AP algorithms. The toolbox also provides functions to implement frequency-domain and lattice filter structures. The general syntax of function `adaptfilt` for implementing adaptive filters is

```
ha = adaptfilt.algorithm(arguments);
```

This function returns an adaptive filter object `ha` of type `algorithm`. The LMS-type adaptive algorithms based on FIR filters include `LMS`, `NLMS`, etc., and are summarized in Table A.9.

Table A.9 Summary of LMS-type adaptive algorithms for FIR filters

algorithm	Description
lms	Direct-form LMS
nlms	Direct-form Normalized LMS
dlms	Direct-form delayed LMS
blms	Block LMS
blmsfft	FFT-based block LMS
ss	Direct-form sign-sign LMS
se	Direct-form sign-error LMS
sd	Direct-form sign-data LMS
filtxlms	Filtered-X LMS
adjlms	Adjoint LMS

Table A.10 Input arguments for `adaptfilt.lms`

Input arguments	Description
l	Filter length (defaults to 10)
step	Step size (defaults to 0.1)
leakage	Leaky factor (defaults to 1)
coeffs	Initial filter coefficients (defaults to 0)
states	Initial filter states (defaults to 0)

For example, we can construct an adaptive FIR filter with the LMS algorithm object as follows:

```
ha = adaptfilt.lms(l,step,leakage,coeffs,states);
```

Table A.10 describes the input arguments such as filter length, step size, etc., available for the `adaptfilt.lms`. The adaptive FIR filter with the LMS algorithm can be implemented as follows:

```
hlms = adaptfilt.lms(32,mu);
[yn,en] = filter(hlms,un,dn);
```

The first statement constructs an adaptive filter object `hlms` using the LMS algorithm with length 32 and step size `mu`. The second statement performs the adaptive filtering with the input arguments `un` and `dn` that represent the signal vectors for $u(n)$ and $d(n)$, respectively, and produce two output vectors `yn` and `en` that are filter output $y(n)$ and error sequence $e(n)$, respectively.

The RLS-type adaptive algorithms based on FIR filter structures are summarized in Table A.11 and the AP-type adaptive algorithms based on FIR filters are summarized in Table A.12. For example, the syntax for the direct-form RLS algorithm is

```
ha = adaptfilt.rls(l,lambda,invcov,coeffs,states);
```

Table A.11 Summary of RLS-type adaptive algorithms for FIR filters

algorithm	Description
ftrf	Fast transversal filter
hrls	Householder RLS
hswrls	Householder sliding window RLS
qrdrls	QR-decomposition-based RLS
rls	Direct-form RLS
swrls	Window RLS

Table A.12 Summary of AP-type adaptive algorithms for FIR filters

algorithm	Description
ap	Direct matrix inversion
apru	Recursive matrix updating
bap	Block affine projection

The forgetting factor (λ) `lambda` is a scalar in the range (0, 1] and its default value is

1. The argument `invcov` is the inverse of the input signal covariance matrix.

A.3.4 A case study: adaptive noise cancellation

The application of adaptive filtering for noise cancellation is introduced in Section 1.6.1.3, and its block diagram is shown in Figure 1.8. This section illustrates the usage of adaptive filtering functions from the Filter Design Toolbox introduced in Section A.3.3 for adaptive noise cancellation. We implement the adaptive noise canceller using an FIR filter with the normalized LMS (NLMS) summarized in Tables A.9 and A.10. The script M-file can be modified to use RLS-type and AP-type algorithms listed in Tables A.11 and A.12, respectively.

A.3.4.1 Principle of adaptive noise cancellation

As shown in Figure A.13, the adaptive noise canceller has two inputs: the primary signal $d(n)$ from the primary sensor (which is placed close to the signal source) and the reference signal $u(n)$ from the reference sensor. Ideally, the reference signal $u(n)$ contains noise only. The primary signal $d(n)$ consists of the desired signal $s(n)$ plus noise $v'(n)$, i.e. $d(n) = s(n) + v'(n)$. The noise $v'(n)$ is highly correlated with $u(n)$ since they are coming from the same noise source with an adequate distance between the primary and reference sensors. The objective of the adaptive filter $W(z)$ is to compensate the amplitude and phase of the reference signal $u(n)$ in order to estimate the noise $v'(n)$. The filter output $y(n)$, which is an estimate of noise $v'(n)$, is then subtracted from the primary signal $d(n)$, producing $e(n)$ as the desired signal $s(n)$ plus reduced noise. Therefore, the ANC output

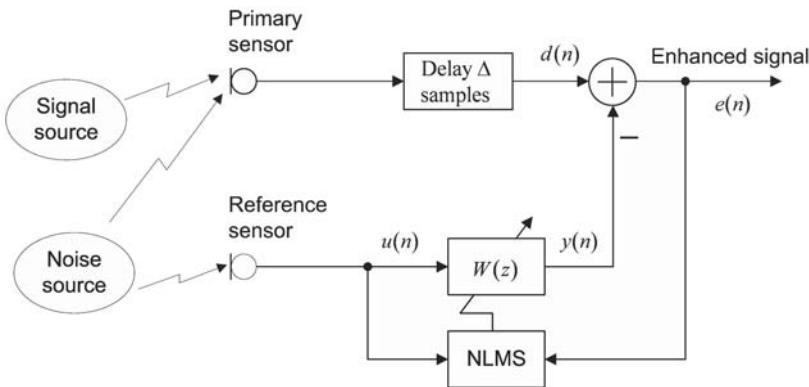


Figure A.13 Adaptive noise cancellation using the NLMS filter

$e(n)$ contains an enhanced signal. Note that in Figure A.13, the delay of Δ samples is inserted in the primary channel to produce $d(n)$, which can guarantee causality of the adaptive FIR filter $W(z)$.

To apply the adaptive noise cancellation effectively, the reference noise picked up by the reference sensor must be highly correlated with the noise components in the primary signal. This can be achieved by placing the primary and reference sensors close to each other. Unfortunately, it is also critical to avoid the desired signal from the signal source being picked up by the reference sensor. This crosstalk problem may be eliminated by using an acoustic barrier between the signal source and reference sensor to reduce the desired signal being picked up by the reference sensor.

A.3.4.2 MATLAB implementation and simulation

The script M-file that implements the adaptive noise cancellation shown in Figure A.13 using the adaptive filtering function `adaptfilt.algorithm` provided in the Filter Design Toolbox is listed below. This MATLAB script shows that adaptive FIR filtering can be easily implemented using functions available in the Filter Design Toolbox.

```
% DelayANC.m    Performs Adaptive Noise Cancellation
%
% Noise: radio or music
% Signal: breathing sound
%
% Delay is inserted in the primary channel to guarantee causality
% of adaptive FIR filter W(z)

Fs = 24000;                      % Sampling frequency
N = 512;                          % Length of adaptive FIR filter
D = 10;                           % Number of delay samples

% Read primary signal

load PrimaryMIC.txt -ascii;      % Load data file, primary channel
```

```

tn = PrimaryMIC';
L = length(tn); % Length of data file
tn = tn - mean(tn); % Compensate for the DC offset caused
% by data acquisition system
dn=[zeros(1,D) tn(:,1:L-D)]; % Insert delay into the primary
% channel to produce d(n)
plot(dn); % Plot the primary signal
title('Time waveform of primary signal');
xlabel('Time index'); ylabel('Amplitude');

% soundsc(dn,Fs); % Listen to the sound
pause;
% spectrogram(dn,1024,256,1024,Fs);
% Display spectrogram of primary
% signal
pause;
% Reference channel

load ReferenceMIC.txt -ascii; % Load data file, reference channel
tn = ReferenceMIC'; % u(n)
un = tn - mean(tn); % Compensate for the DC offset
% plot(un); % Plot the reference signal
% soundsc(un,Fs); % Listen to the sound
% pause;

mu = 1; % Step size
h = adaptfilt.nlms(N,mu); % Normalized LMS algorithm
[yn,en] = filter(h,un,dn); % Adaptive filtering

% Users are encouraged to try different algorithms listed in
% Tables A.11 and A.12 for comparing performance of algorithms
plot(en); % Plot error signal e(n)
title('Time waveform of error signal');
xlabel('Time index'); ylabel('Amplitude');

pause;
spectrogram(en,1024,256,1024,Fs);
% Display spectrogram of error signal
% soundsc(en,Fs); % Listen to the sound
% save enSig.txt en -ascii; % Save enhanced signal in data file

```

The noisy primary signal is shown in Figure A.14, where its spectrogram is given in Figure A.15. The following statement:

```
spectrogram(en,1024,256,1024,Fs);
```

computes and displays the spectrogram using a short-time Fourier transform of the signal specified by vector `en` using a default Hamming window. Both the time-domain waveform and the frequency-domain spectrogram show that the primary signal is a very noisy signal

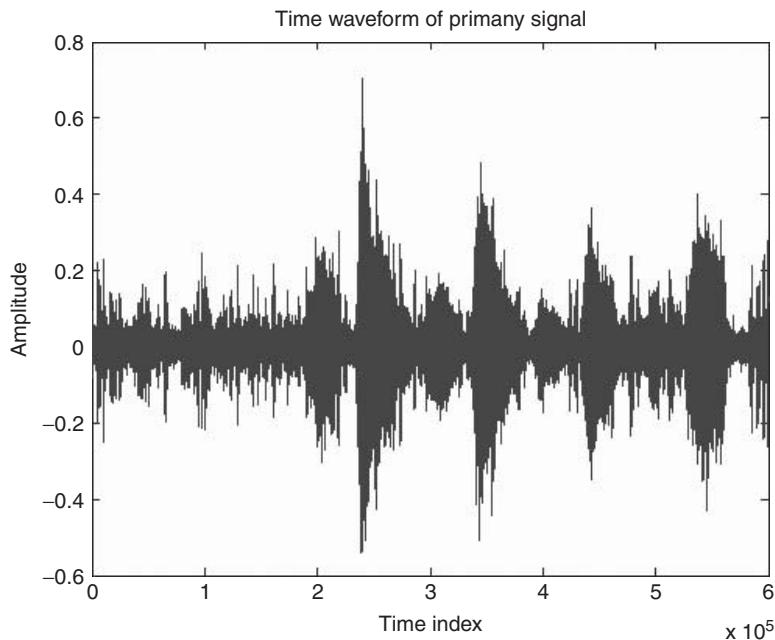


Figure A.14 Time waveform of primary signal $d(n)$

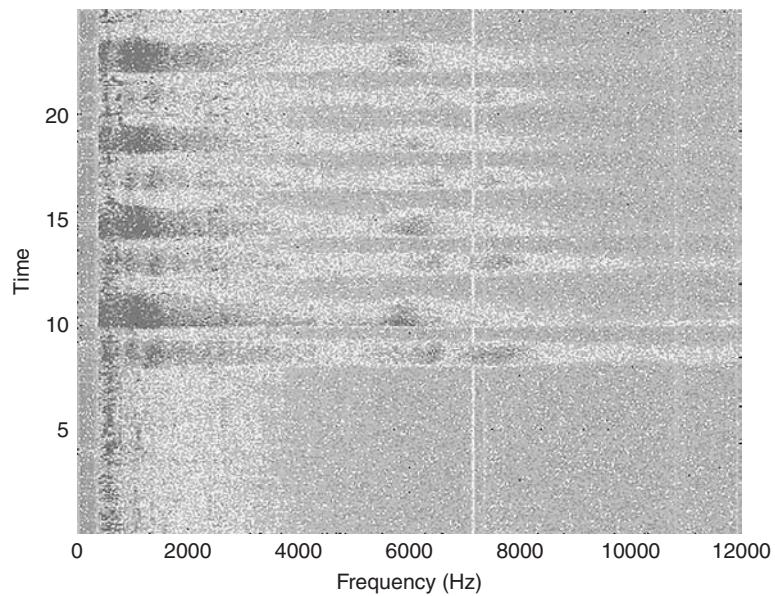


Figure A.15 Spectrogram of the primary signal

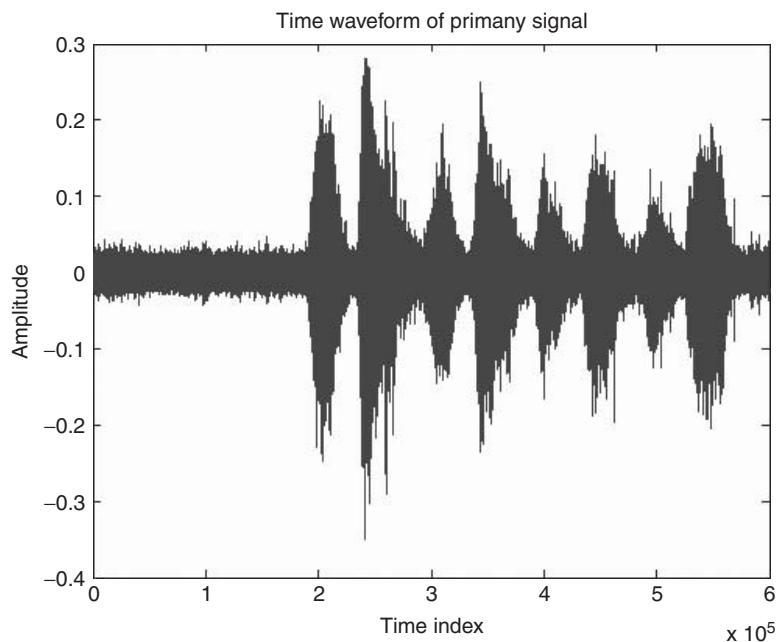


Figure A.16 Enhanced (error) signal $e(n)$

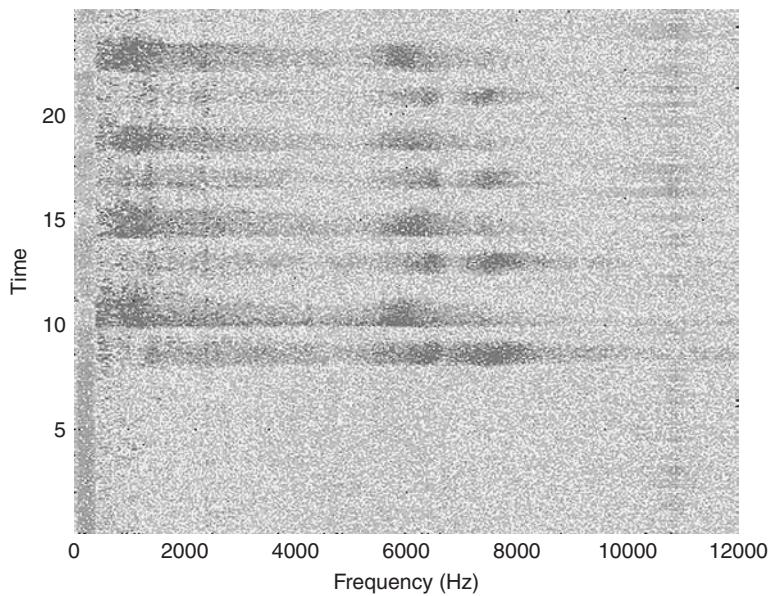


Figure A.17 Spectrogram of the enhanced breathing signal

and it is very difficult to detect the desired breathing sound from the background noise generated by a radio talk show.

As shown in the MATLAB script, the adaptive noise canceller uses the NLMS algorithm to enhance the desired breathing sound. The enhanced output signal $e(n)$ is displayed in Figure A.16 and its spectrogram is shown in Figure A.17. The performance of adaptive noise cancellation can be evaluated by comparing the primary signal $d(n)$ and the error signal $e(n)$ in both the time domain and frequency domain.

Appendix B

Using MATLAB for adaptive filtering and subband adaptive filtering

This appendix provides an introduction to using MATLAB for digital signal processing (DSP), adaptive filtering and subband adaptive filtering. New MATLAB users are encouraged to read the first section of Appendix A for a quick hands-on guide to MATLAB fundamentals.

B.1 Digital signal processing

This section introduces the basic concepts of digital signal processing and how to represent digital signals in MATLAB.

B.1.1 Discrete-time signals and systems

Signals can be divided into three broad categories: (i) continuous-time (or analog) signals, (ii) discrete-time signals and (iii) digital signals. Signals that we encounter daily are mostly analog, which are defined continuously in time and amplitude. Discrete-time signals are defined only at a set of time instances with a continuous range of amplitudes. This is different from digital signals, which have discrete values in both time and amplitude.

Discrete-time and digital signals share a common characteristic; i.e. both can be treated as sequences of numbers. However, each discrete-time signal sample needs an infinite number of bits (or word-length) to represent its continuous-amplitude value. Signals processed on digital hardware (or computers) are represented in the form of digital signals, which are obtained by quantization of discrete-time signals in amplitude using a finite number of bits. Analysis of quantization (of signal samples and system parameters) effects is known as the finite word-length analysis. For development and analysis of DSP systems

and algorithms, we use discrete-time signals for mathematical simplicity. In most cases, it is acceptable to represent digital signals with a sufficient number of bits; i.e. the quantization and arithmetic errors are negligible. For example, MATLAB uses 64 bits (by default) to represent numbers. See Section A.1.4 for other supported data types in MATLAB. Remember that, in practice, only digital signals can be represented and processed by digital systems.

A DSP system is a digital hardware (or software) that implements some computational operations to process digital signals. Given one or more digital signals as inputs, a DSP system transforms, manipulates, extracts information or modifies those signals and produces outputs for some predefined purposes. Figure 1.1 depicts a block diagram of an adaptive digital system. As described in Chapter 1, and briefly discussed in this appendix, the output signal is generated by filtering the input signal such that the output is an approximation to the reference (desired) signal in a statistical sense.

B.1.2 Signal representations in MATLAB

In DSP systems, signals are sequences of numbers (or samples) created, for example, by sampling or measurement of some physical events. In MATLAB, we represent signals as finite-duration sequences in vectors due to finite memory limitations (i.e. it is not possible to save, store or load an infinite number of samples into memory on a computer). Assuming that all signals are causal (start at the time index $n = 0$), we can represent a signal sequence as a row vector in MATLAB, as in the following example:

```
un = [6,-2,-21,1,16,10,-16,-1,-7,-10,-12,3,-4,1,-4];
```

We may also define a time-index vector n , which has the same number of samples as un , to indicate the position (in discrete time) of each sample in the vector un as follows:

```
n = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14];
```

Note that this time-index vector can be easily generated by the MATLAB command as follows:

```
n = [0:length(un)-1];
```

The time-index vector is crucial, for example, when plotting the signal un (see Table A.3 for details).

A discrete-time signal is defined only for integer values of the time index n and is left undefined for other time values in between those integers. Therefore, a more precise way of plotting a discrete-time signal is shown in Figure B.1, generated by the MATLAB function `stem` (see Section A.1.8) as `stem(n,un)`. We also superimpose a dotted-continuous line generated by the `plot` function, which connects samples of the digital signal. In some cases, it would be difficult to see the shape of the signal by just plotting sample points using `stem`. We can also use the style options supported by the `plot` function to indicate those discrete sample values. For example, we can use the following statement to generate a similar plot as shown in Figure B.1:

```
plot(n,un,:o');
```

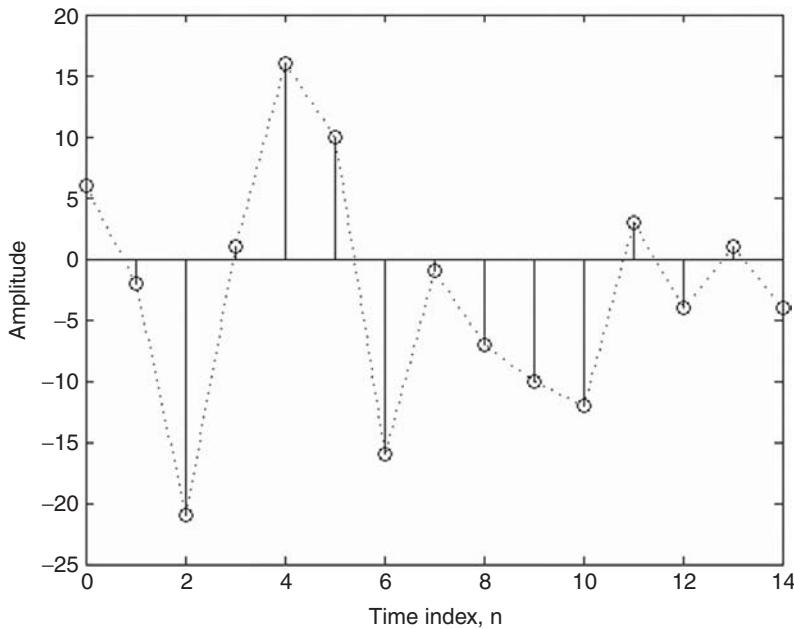


Figure B.1 Graphical representation of a discrete-time signal

Nevertheless, we should also be aware of the pitfall that connecting the sample points may give the impression that the signal is defined for continuous values of the time index n , which is not the case for discrete-time and digital signals. The MATLAB code for plotting the signal in Figure B.1 is given below:

```
un = [6,-2,-21,1,16,10,-16,-1,-7,-10,-12,3,-4,1,-4];
n = [0:length(un)-1];% Time index for un
stem(n,un);           % Plot digital signal
hold on;
plot(n,un,':');       % Superimpose with continuous curve
```

As mentioned earlier, MATLAB is a powerful tool for simulation of DSP algorithms and applications. The data files (such as `engine_1`, `engine_2`, `Timit`, etc., in the Common folder) used for simulation are best coming from data acquisition in real experimental setups, but many users use MATLAB to generate digital signals for computer simulations. Two most frequently used digital signals are sinusoidal and random signals, which can be generated, for example, as follows:

```
fs = 8000;                      % Define sampling rate
f = 500;                         % Frequency of sinewave
n = 0:1:1023;                    % Time index covers 1024
                                  % samples
un = sin(2*pi*f*n/fs);          % Generate sinewave
vn = randn(size(n));             % Generate random signal
```

B.2 Filtering and adaptive filtering in MATLAB

An adaptive filter, as shown in Figure 1.2, is a time-varying system that uses a recursive (adaptive) algorithm to continuously adjust its tap weights for operation in an unknown environment. An adaptive filter typically consists of two functional blocks: (i) a digital filter to perform the desired filtering and (ii) an adaptive algorithm to adjust the tap weights of the filter.

As shown in Figure 1.1, an adaptive filter takes two inputs and produces one output. Given an input signal $u(n)$ and desired response $d(n)$, the task of an adaptive filter is to compute the output $y(n)$ from the input signal $u(n)$ such that $y(n)$ approximates the desired response $d(n)$ in some statistical sense. For example, the LMS algorithm minimizes the mean-squared difference between the two signals. The difference between the desired response $d(n)$ and the input signal $y(n)$ is called the error signal $e(n)$. The error signal is used by the adaptive algorithm to adjust the tap weights of the digital filter. Note that for some applications, such as adaptive noise cancellation, the error signal contains an enhanced signal that becomes the desired output of the adaptive system.

Figure 1.3 shows the most commonly used FIR filter structure for adaptive filtering. Its counterpart, the infinite impulse response (IIR) filter, complicates the design of adaptive algorithms due to stability issues. Thus the FIR filter is widely used in practical applications. We restrict the discussion here to the class of adaptive FIR filters, either in the time, subband, frequency or transform domain.

B.2.1 FIR filtering

Figure B.2 shows the block diagram of an adaptive FIR filter using the LMS algorithm. In this section, we focus on the implementation of the filtering block. As depicted in the figure, the FIR filtering block consists of a tapped-delay line and a set of tap weights. The tapped-delay line gathers a block of M samples represented mathematically in vector form as

$$\mathbf{u}(n) \equiv [u(n), u(n - 1), \dots, u(n - M + 1)]^T. \quad (\text{B.1})$$

Similarly, the tap weights can be represented in vector form as

$$\mathbf{w}(n) \equiv [w_0(n), w_1(n), \dots, w_{M-1}(n)]^T. \quad (\text{B.2})$$

Both $\mathbf{u}(n)$ and $\mathbf{w}(n)$ are defined as M -by-1 column vectors, where the index n in the parenthesis represents the time index. The parameter M is called the length of the FIR filter. It indicates the number of adjustable coefficients (i.e. the tap weights) in the filter. Note that an FIR filter of length M has the order of $M - 1$. For a causal system, the time index n increases with a step of unity from 0, 1, 2, ... to the end of operation.

Assuming that we use a four-tap FIR filter (i.e. $M = 4$), Figure B.3 illustrates the tapped-delay line operations at the time index $n = 1, 2, 3, 4, 5$ and 6 (recall that the MATLAB index starts from 1). At every time instance, the tapped-delay line gathers a block of $M = 4$ samples consisting of one new samples and three (i.e. $M - 1 = 3$) old

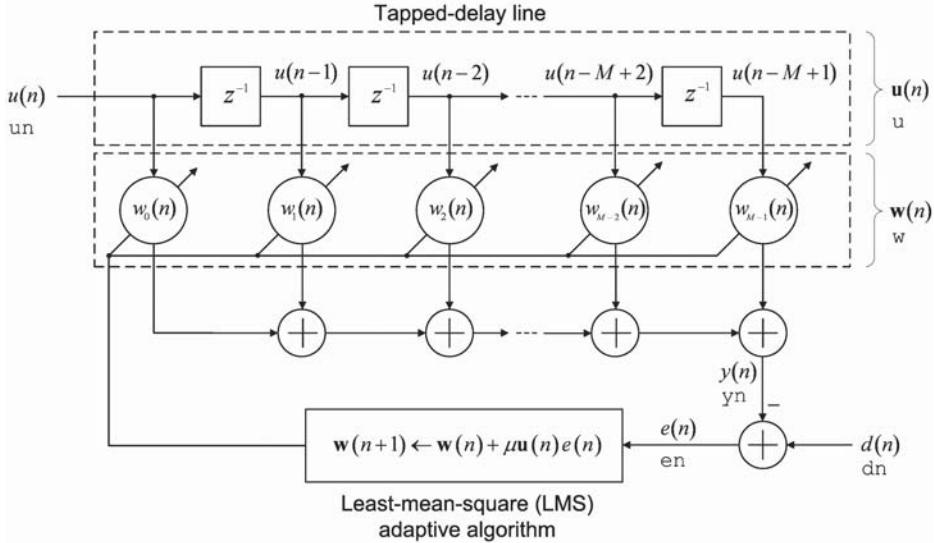


Figure B.2 The LMS algorithm is built based on the FIR filter. The signals $u(n)$, $d(n)$, $y(n)$ and $e(n)$ are represented by the row vectors u_n , d_n , y_n and e_n in MATLAB. The column vectors $\mathbf{u}(n)$ and $\mathbf{w}(n)$ of the adaptive filter are represented as column vectors u and w , respectively

```

Initialize,u = [ 0 0 0 0 ]T
n = 1,u = [ 6 0 0 0 ]T
n = 2,u = [ -2 6 0 0 ]T
n = 3,u = [-21 -2 6 0 ]T
n = 4,u = [ 1 -21 -2 6 ]T
n = 5,u = [ 16 1 -21 -2 ]T
n = 6,u = [ 10 16 1 -21 ]T

```

Figure B.3 Tapped-delay line refreshing operation of a four-tap FIR filter

samples. The oldest sample was pushed out from the vector. The following MATLAB code implements the tapped-delay line refreshing operation:

```

un = [6, -2, -21, 1, 16, 10, -16, -1, -7, -10, -12, 3, -4, 1, -4];
ITER = length(un);
u = zeros(4,1);
for n = 1:ITER
    u = [un(n); u(1:end-1)];
    disp(sprintf('%d\t\t',u));
end

```

The input signal consists of 15 samples represented as a row vector u_n (we used the same vector earlier in Section B.1.2). We check the length of the input signal using the

`length` function. The index `n` starts from 1 as the MATLAB index always starts from 1 instead of 0. At each iteration of the `for` loop, the input vector `u` is updated with a new sample `un(n)`. The oldest sample `un(end)` was pushed out from the vector `u` (recall that the keyword `end` indicates the end of a vector; see Section A.1.3 for details). We initialize the elements of the input vector to zeros in the third line of the code. For this reason, the first vector in Figure B.3 consists of zero-valued elements.

As shown in Figure B.2, the output $y(n)$ of the FIR filter is computed as the weighted sum of the M input samples, where the weights are the adjustable coefficients $w_0(n), w_1(n), \dots, w_{M-1}(n)$ of the filter. The output $y(n)$ can be expressed in both scalar and vector notations as follows:

$$y(n) = \sum_{m=0}^{M-1} w_m(n)u(n-m) = \mathbf{w}^T(n)\mathbf{u}(n). \quad (\text{B.3})$$

To compute one output sample $y(n)$, the input vector $\mathbf{u}(n)$ is updated with a new input sample (as described above) and the output is given by the inner product of the tap-weight and input vectors. The filtering operation can be implemented in MATLAB by inserting two additional lines into the tapped-delay line code as follows:

```
un = [6,-2,-21,1,16,10,-16,-1,-7,-10,-12,3,-4,1,-4];
ITER = length(un); % No. of iterations
u = zeros(4,1); % Clear signal buffer
w = randn(4,1);
for n = 1:ITER
    u = [un(n); u(1:end-1)]; % Update signal buffer
    yn(n) = w'*u; % Compute filter output
end
```

In this example, we initialize the elements of the tap-weight vector to some random values and the tap weights are held constant throughout the process. As we shall see in the next section, the tap weights of an adaptive filter are updated at every iteration by the adaptive algorithm.

B.2.2 The LMS adaptive algorithm

The coefficients of a digital filter determine the filter characteristics. Based on some specifications (e.g. cut-off frequency and stopband ripples), we can design digital filters (that contain the set of coefficients conforming to the given specifications) and plug the coefficients into the filter structure. Nowadays, filter design can be easily done using software packages such as the Signal Processing Tool (SPTool) and the Filter Design and Analysis Tool (FDATool) presented in Appendix A.

In many practical applications, filter specifications are unknown at the design time. For example, one might need a digital filter that could model the acoustic response of a room. The acoustic responses are different from one room to another. Furthermore, acoustic responses change with time. A better solution for solving this problem is to use a digital filter with time-varying coefficients (i.e. an adaptive filter) to track the unknown yet changing environments. Different from the fixed-coefficient filters, designing an adaptive

filter requires some considerations, such as the filter length, step size, regularization parameters and the adaptive algorithm to be used.

One of the most widely used adaptive algorithms is the LMS algorithm introduced in Section 1.4. The LMS algorithm updates the filter coefficients (i.e. tap weights) as follows:

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \underbrace{\mu \mathbf{u}(n) e(n)}_{\text{adjustment}}, \quad (\text{B.4})$$

where μ is the step size that determines the stability and the convergence rate of the algorithm. The time-varying nature of the weight vector $\mathbf{w}(n)$ is indicated by having an index n to show its value as a function of time. Equation (B.4) can be understood as follows. At the current iteration n , the amount of adjustment to the current tap weights $\mathbf{w}(n)$ is determined by the input vector $\mathbf{u}(n)$ (formed with signal samples in the tapped-delay line) scaled with the estimation error $e(n)$ and the step size μ . As shown in Figure B.2, the estimation error (or error signal) $e(n)$ measures the difference between the adaptive filter output $y(n)$ and the desired response $d(n)$. The following MATLAB code implements the LMS algorithm:

```
for n = 1:ITER
    u = [un(n); u(1:end-1)]; % Update signal buffer
    yn(n) = w'*u; % Compute filter output
    en(n) = dn(n) - yn(n); % Compute error signal
    w = w + (mu*en(n))*u; % Update weight vector
end
```

At each iteration of the `for` loop, the input vector `u` is updated with the new sample `un(n)`. The filter output `yn(n)` and estimation error `en(n)` are computed using the current weight vector `w`. We then update the tap weights by adding the update term `(mu*en(n))*u` to `w` and assign the result back to `w` for the next iteration. The same operation is repeated until the end of the `for` loop. Notice that `yn` and `en` are defined as vectors. We also keep an account of the changes in the filter output and estimation error for further analysis.

B.2.3 Anatomy of the LMS code in MATLAB

Available on the companion CD, the script and function M-files implementing the LMS algorithm are `LMSinit.m`, `LMSadapt.m` and `LMSdemo.m`. The first function, `LMSinit`, initializes the parameters of the LMS algorithm. The second function, `LMSadapt`, performs the actual computation of the LMS algorithm. The third M-file, `LMSdemo`, is provided as an example of using `LMSinit` and `LMSadapt`.

The function M-files `LMSinit.m` and `LMSadapt.m` are kept in the folder called `Common`, while the demo script `LMSdemo.m` can be found in the folder ‘Chapter1’ on the companion CD. See Appendix C for a brief description of the organization of M-files. Complete listings of the code for these three M-files are shown in Tables B.1, B.2 and B.3. In these tables, we have divided the programs into smaller segments. An explanation of the individual segment is given in the following subsection.

B.2.3.1 The `lmsdemo` script

In this script M-file, the LMS algorithm is demonstrated using an FIR filter for adaptive system identification (see Figure 1.5). The complete M-file is listed in Table B.1. A brief description for each statement is given below:

- (a) The `addpath` command adds the folder ‘Common’ to the MATLAB search path since the initialization and adaptation functions (`LMSinit` and `LMSadapt`) are located in the `Common` folder. The `clear all` command removes all the variables from the workspace.

Table B.1 List of LMSdemo script M-file

```
% LMSdemo           Least Mean Square (LMS) Algorithm Demo

addpath '..\Common';
clear all;                                     (a)

mu = 0.001;
M = 256;                                       (b)

iter = 8.0*80000;
b = load('h1.dat');
b = b(1:M);
[un,dn] = GenerateResponses(iter,b);
stepcheck_lms(un,M,mu);

tic;
S = LMSinit(zeros(M,1),mu);
S.unknownsys = b;
[yn,en,S] = LMSadapt(un,dn,S);
EML = S.eml.^2;
err_sqr = en.^2;
disp(sprintf('Total time = %.3f mins',toc/60)); (d)

figure;
q = 0.99; MSE = filter((1-q),[1 -q],err_sqr);
hold on; plot((0:length(MSE)-1)/1024,10*log10(MSE));
axis([0 iter/1024 -60 10]);
xlabel('Number of iterations (\times 1024 input samples)');
ylabel('Mean-square error (with delay)');
title('LMSdemo');
grid on;                                         (e)

figure;
hold on; plot((0:length(EML)-1)/1024,10*log10(EML));
xlabel('Number of iterations (\times 1024 input samples)');
ylabel('Misalignment (dB)');
title('LMSdemo');
grid on;                                         (f)
```

- (b) Set the length m of the FIR filter to 256 and step-size μ_u to 0.001. The length of the FIR filter is selected so that it is sufficiently long to cover the impulse response of the unknown plant. The step size μ_u has to be bounded within a certain range according to (1.12).
- (c) In practical applications, the unknown system b is a physical plant with both input and output connected to the adaptive filter. For experimental purposes, we simulate the unknown system with an FIR filter. The coefficients of the FIR filter are taken from the `h1.dat` data file using the `load` command. We further truncate the length of vector b to m so that the unknown plant and the adaptive filter have the same length of impulse response. This useful shortcut (only applicable in a controlled simulation) allows us to investigate the convergence behavior of the adaptive filter without considering the over- or undermodeling problem.

Given the coefficients of the unknown plant in vector b , the `GenerateResponse` function generates two signal vectors: u_n and d_n . The input signal u_n is a zero-mean unit-variance random noise. The desired response d_n is obtained by filtering u_n through the unknown system defined as vector b . By default, white noise is added to d_n as a plant noise with a 40 dB signal-to-noise ratio (SNR). With reference to Figure 1.5, we can now completely simulate the adaptive system identification using u_n as the input signal and d_n as the desired response (i.e. the reference signal) to the adaptive filter. The variable `iter` determines the duration of the simulation (i.e. number of samples of the signal used for simulation). The `stepcheck_lms` function checks whether a given step size μ_u is appropriate based on the variance of u_n . See Section 1.4 for a discussion on choosing the step size.

- (d) The `LMSinit` function performs the initialization of the LMS algorithm. The argument `zeros(M, 1)` sets the initial tap weights to zero. The `LMSadapt` function performs the actual computation of the LMS algorithm, where u_n and d_n are the input and desired signal vectors, respectively. The structure array s , from the `LMSinit` function stores the initial state of the LMS algorithm. The same structure array is passed to the `LMSadapt` function. An output signal y_n , an error signal e_n and the final state s are returned by the `LMSadapt` function at the end of the adaptation. The convergence behavior of the adaptive filter can now be analyzed by looking at the squared error `err_sqr` and the misalignment error `EML` (see Equation (4.12)). The last statement displays the total execution time for all operations between the `tic` and `toc` commands. Note that `tic` and `toc` functions work together to measure elapsed time. The `tic` saves the current time that `toc` uses later to measure the elapsed time. The sequence of commands

```
tic;
...
toc;
```

measures the amount of time MATLAB takes to complete the operations and displays the time in seconds.

- (e) The `err_sqr` is smoothed along the time axis to approximate the mean-square error (MSE). The plot of the MSE versus time is called the learning curve. For the case where convergence is achieved, the learning curve shows that the MSE gradually

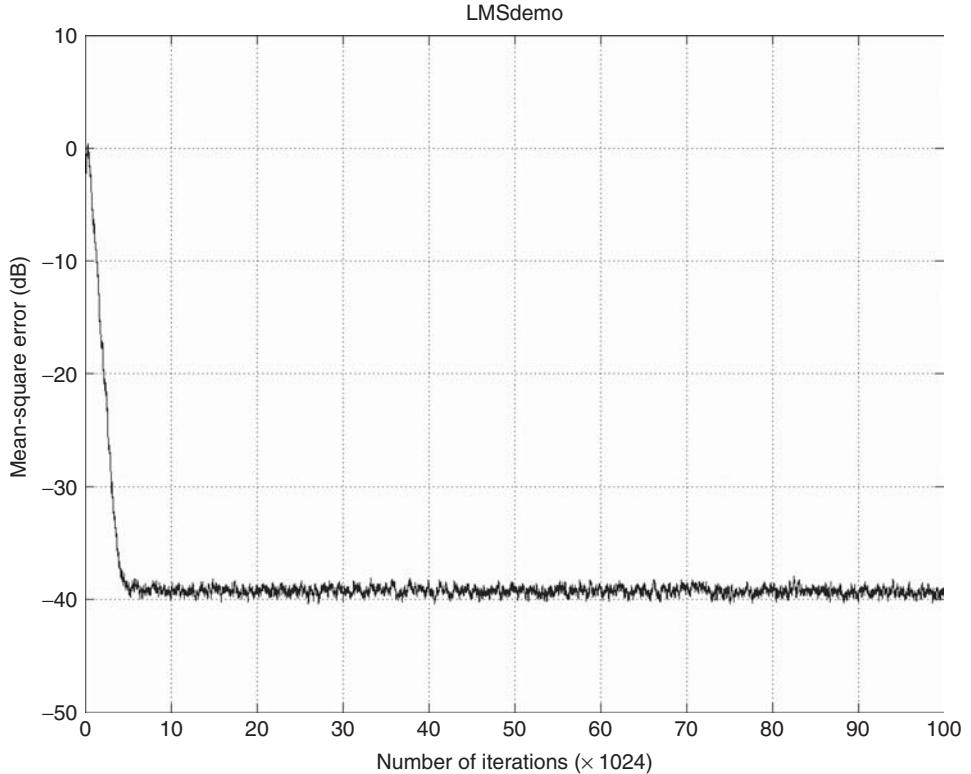


Figure B.4 Mean-square error curve showing the convergence of the LMS algorithm

converges (or decays) to a constant value. We contract the time axis of the MSE plot by 1024 samples in order to analyze the convergence behavior for a longer duration. The resulting MSE plot is shown in Figure B.4.

- (f) The misalignment plot indicates the difference between the unknown plant and the adaptive filter at every iteration. The misalignment curve is shown in Figure B.5.

B.2.3.2 The `LMSinit` function

The `LMSinit` function uses the structure array `s` to set up parameters such as the step size, leaky factor and initial weights. These parameters have to be set accordingly in order for the LMS algorithm to estimate the unknown system `b` with sufficient accuracy. The `LMSinit` function is listed in Table B.2. A brief description of each statement is given below:

- (a) The `LMSinit` function accepts three input arguments: the initial weights `w0`, the step size `mu` and the leaky factor `leak`. The leaky factor is set to a default value of zero (i.e. the normal form of LMS without leakage) if only two input arguments are given. The variable `nargin` indicates the number of input arguments given to a

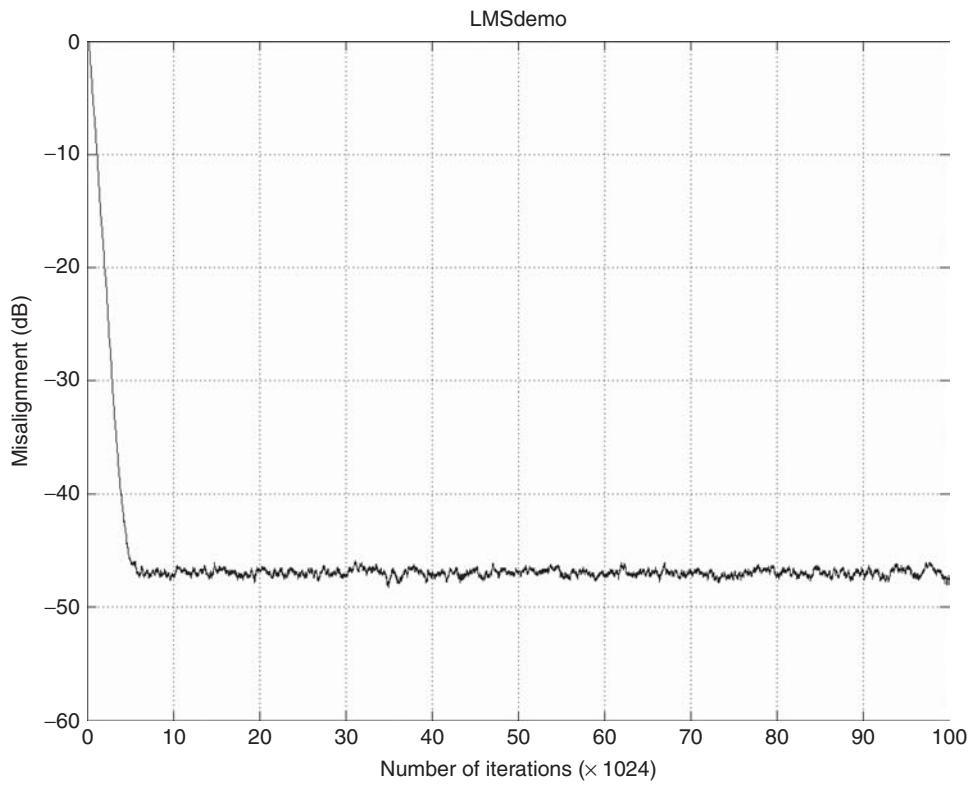


Figure B.5 Misalignment curve showing the convergence of the LMS algorithm

Table B.2 List of `LMSinit` function M-file

```
% function S = LMSinit(w0,mu,leak)

if nargin < 3
    leak = 0;
end

S.coeffs      = w0 (:);
S.step        = mu;
S.leakage     = leak;
S.iter        = 0;
S.AdaptStart  = length (w0);
```

(a)

(b)

MATLAB function. The `LMSinit` function returns the structure array `S` containing all the parameters required by the `LMSadapt` function, which performs the system identification.

- (b) `s` is a structure array with five fields: `coeffs`, `step`, `leakage`, `iter` and `AdaptStart`. The first three fields are set according to the value passed to the function. Notice that the length `m` of the adaptive filter is automatically determined by the length of the initial vector `w0` given to the `LMSinit` function. The `iter` field is used to record the number of iterations of the weight adaptation. The last field, `AdaptStart`, is used in the `LMSadapt` function to start the weight adaptation when the iteration number exceeds the value assigned to `AdaptStart`. The initial value of `AdaptStart` is set equal to the length `m` of the adaptive filter. By doing so, the weight adaptation is performed after the FIR filter has completely ‘run on to’ the input sequence `un`.

B.2.3.3 The `LMSadapt` function

The `LMSadapt` function, as shown in Table B.3, accepts three input arguments: the input signal `un`, the desired response `dn` and the parameter array `s`. In the `LMSdemo` script presented earlier, `dn` is generated by filtering `un` using the ‘unknown’ plant `b`. Given `un` and `dn` as input arguments, the `LMSadapt` function adjusts the coefficients of its FIR filter such that the filter output `yn` is close to the desired response `dn` in an MSE sense. The adaptive tap weights `w` are updated iteratively toward the unknown plant `b` using samples of `un` and `dn` in the adaptation. A brief description for each statement is given below:

- (a) The structure array, `s`, provides a convenient way of passing the parameters between the `LMSinit` and `LMSadapt` functions. Instead of a bunch of MATLAB variables, the parameters of the LMS filter are initialized and packed into the structure array `s`, which are then unpacked and used in the `LMSadapt` function.
- (b) The LMS algorithm given in Equation (B.4) is a sample-based algorithm, where the weight adaptation is performed sample by sample. The number of samples contained in the input signal vector `un` determines the number of iterations, `ITER`. We also pre-allocate two row vectors, `yn` and `en`, to store the output and error signals generated during the adaptation process. The column vector `u` represents the input signal vector `u(n)` defined in Equation (B.1), the length of which is the same as the weight vector `w(n)`. Notice that we use `un` to represent the input signal `u(n)`, as shown in Figure B.2.
- (c) The normalized misalignment is a measure of the difference between the unknown plant and the adaptive filter at each iteration. If `b` is assigned to the structure array `s` prior to calling the `LMSadapt` function (see statement (d) of Table B.1), the `ComputeEML` flag is set to 1 and the misalignment `EML` will be computed during the adaptation process.
- (d) The LMS adaptation is implemented using a `for` loop. At each iteration, the following operations are performed in sequence:
 - (i) Update the tap-input vector `u` with a new sample `un(n)`; the oldest sample `u(end)` is pushed out from the vector.

Table B.3 List of function M-file LMSadapt

```
% function [yn,en,S] = LMSadapt(un,dn,S)

M = length(S.coeffs);
mu = S.step;
leak = S.leakage;
AdaptStart = S.AdaptStart;
w = S.coeffs; (a)

u = zeros(M,1);
ITER = length(un);
yn = zeros(1,ITER);
en = zeros(1,ITER); (b)

if isfield(S,'unknownsys')
    b = S.unknownsys;
    norm_b = norm(b);
    eml = zeros(1,ITER);
    ComputeEML = 1; (c)
else
    ComputeEML = 0;
end

for n = 1:ITER
    u = [un(n); u(1:end-1)];
    yn(n) = w'*u;
    en(n) = dn(n) - yn(n);
    if ComputeEML == 1;
        eml(n) = norm(b-w)/norm_b; (d)
    end
    if n >= AdaptStart
        w = (1-mu*leak)*w + (mu*en(n))*u;
        S.iter = S.iter + 1;
    end
end

S.coeffs = w;
if ComputeEML == 1;
    S.eml = eml; (e)
end
```

- (ii) Compute the FIR filter output y_n , which is given by the inner product between the current adaptive weight vector w and the tap-input vector u .
- (iii) Calculate the error e_n between the filter output and the desired response.
- (iv) Update the adaptive filter coefficients. Equation (B.4) gives the adaptation equation for the case where the leaky factor $\text{leak} = 0$. The filter coefficients are updated by adding $(\mu * e_n) * u$ to w and assign the result back to w for the next iteration. By setting `AdaptStart` equal to the length of the adaptive filter M , the weight adaptation will only be performed after the filter has completely ‘run on to’ the input sequence u_n .
- (e) Finally, the updated filter coefficients w and the misalignment error EML are packed on to the structure array s and passed back to the calling script.

B.3 Multirate and subband adaptive filtering

In this section, we describe the MATLAB implementation of multirate filter banks and subband adaptive filters. Readers are encouraged to refer to Chapter 2 for a more detailed description of subband and multirate systems and Chapter 4 for the fundamental principles of subband adaptive filtering.

B.3.1 Implementation of multirate filter banks

B.3.1.1 Decimator and interpolator

Digital signal processing systems that use more than one sampling rate are referred to as multirate systems. Two basic sampling rate conversion devices employed in multirate systems are the decimator and interpolator (see Section B.1 for more details).

A decimator with a decimation factor D , where D is a positive integer, reduces the sampling rate of an input sequence by a factor of D . The decimator is implemented by keeping every D th sample of the input sequence, while removing other samples. On the other hand, an interpolator with an interpolation factor of I , where I is a positive integer, increases the sampling rate of the input signal by inserting $(I - 1)$ zero samples between an adjacent pair of input samples. The decimation and interpolation can be easily implemented in MATLAB using the colon ‘`:`’ operator. The MATLAB scripts `Example_2P1.m` and `Example_2P2.m` demonstrate the operations of decimation and interpolation using a sinusoid as the input signal. Table B.4 lists these two files. The explanation for each code segment is given below:

- (a) We set the decimation and interpolation factors, d and i , to be the same. In the following, we first decimate the signal x with a factor of 2 and then perform interpolation on the decimated sequence with the same factor. L is the length of the lowpass filter h_{LP} , which removes unwanted high-frequency components resulting from zero-insertions.
- (b) The input signal x is a 50 Hz ($f = 50$) sinusoid with a sampling rate of 1 kHz ($fs = 1000$). The term $2*pi*(freq/fs)$ in the `sin` function gives a normalized frequency of 0.1π .

Table B.4 List of MATLAB scripts from Example_2P1 and Example_2P2

% Example_2P1 and Example_2P2	
clear all;	(a)
D = 2;	
I = 2;	
L = 17;	
h_LP = fir1(L-1,1/2);	
fs = 1000; freq = 50; N = 100;	(b)
n = 0:N-1;	
x = sin(2*pi*(freq/fs)*n);	
x_D = x(1:D:end);	(c)
x_I = zeros(1,length(x));	
x_I(1:I:end) = x_D;	
y = filter(h_LP,1,x_I);	
figure;	(d)
subplot(4,1,1); stem(0:20, x(1:21));	
xlabel('Time index, n');	
subplot(4,1,2); stem(0:10, x_D(1:11));	
xlabel('Time index, k');	
subplot(4,1,3); stem(0:20, x_I(1:21));	
xlabel('Time index, n');	
subplot(4,1,4); stem(0:20, y((1:21)+(L-1)/2)*I);	
xlabel('Time index, n');	

- (c) For the decimation, every D th samples of x is selected using $[1:D:end]$, where D indicates the incremental step and the keyword `end` indicates the end of the row vector x . The signal x_D is then interpolated with the same factor of 2 by inserting a zero sample between an adjacent pair of samples in x_D . The original, decimated and interpolated sequences are shown in the first three panels of Figure B.5. The following observations can be made:
- (i) The sampling rate of x_D (in the second panel) is half of the sampling rate of the original sequence x and the interpolated sequence x_I .
 - (ii) The sampling rate is recovered in x_I by inserting zeros. However, interpolation alone does not reconstruct the original signal x .
 - (iii) The original input x can be reconstructed with interpolation followed by low-pass filtering. The lowpass filter h_{LP} removes the high-frequency images resulting from inserting zero-valued samples. The reconstructed signal y is shown in the last panel of Figure B.6.

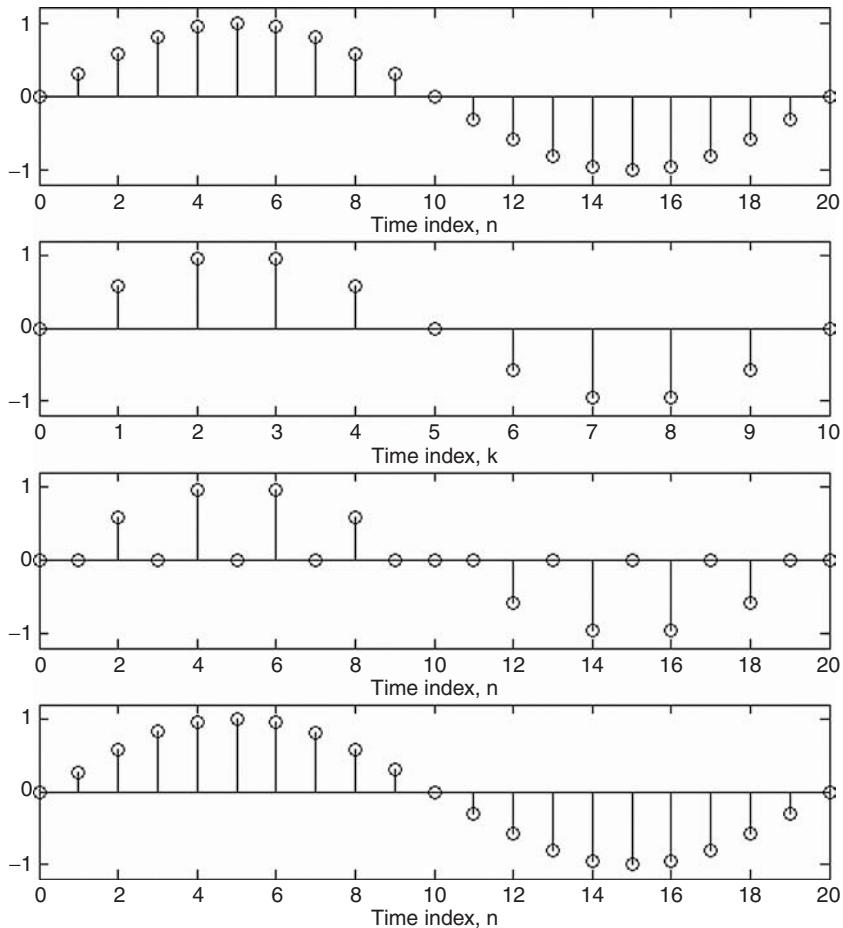


Figure B.6 The input signal x (first panel) is decimated with a factor of 2. The decimated signal x_D (second panel) is then interpolated with a factor of 2. The interpolated signal x_I (third panel) is finally lowpass filtered to obtain the output signal y (fourth panel), which is the reconstructed signal after sampling rate conversion

- (d) We plot the first cycle of the sinusoids. The output signal y shows that there is a delay of $(L-1)/2$ samples caused by the lowpass filter. We compensate this delay in the `plot` function.

B.3.1.2 DFT Filter bank as a block transform

Figure 2.3 shows an N -channel filter bank. The left side of the figure is the analysis filter bank consisting of a set of bandpass filters with a common input. The right side of the figure is the synthesis filter bank with a summed output from another set of bandpass filters. Filter banks can be implemented as a block transform with memory. The idea

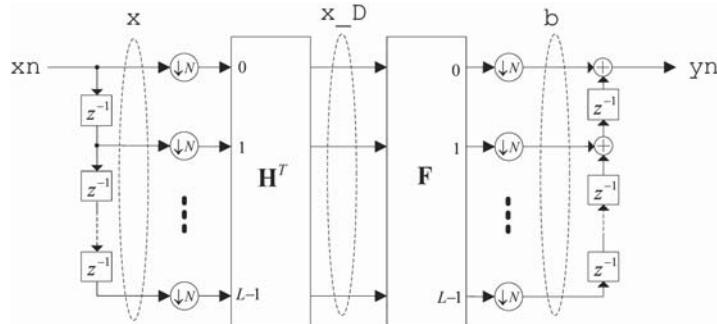


Figure B.7 A DFT filter bank can be implemented as a block transform with memory

is illustrated in Figure B.7 and Table B.5. In the following, we give a step-by-step explanation of the MATLAB implementation for the DFT filter bank:

- Design a prototype lowpass filter `hopt` using the `fir1` function available from the Signal Processing Toolbox. The cutoff frequency (second argument in the `fir1` function) is set at $1/N$, which is determined by the number of subbands $N = 8$. The DFT filter bank is then generated via complex modulation (see Section 2.6) of the prototype filter using the `make_bank_DFT` function. The length of the analysis filter is $L = 256$. The magnitude response of the filter bank is shown in Figure 2.14.
The coefficients of the analysis and synthesis are stored in matrices H and F , respectively. Both H and F are 256-by-8 matrices. Each column of H holds the coefficients of an analysis filter, with the first column corresponding to the first analysis filter and so on. Similar representation applies for the synthesis filter bank F . The scaling factor `sqrt(N)` ensures a unity gain for the combined analysis–synthesis filter bank.
- The input signal xn to the filter bank is a sinusoid with normalized frequency of 0.02π . With this frequency (see Figure 2.14), the signal xn essential passes through only the first subband while attenuated in other subbands. We pre-allocate another row yn for the synthesized output of the filter bank. We also define two row vectors, x and b , as the signal buffers at the analysis and synthesis sections, respectively. The length of the buffers is determined by the length L of the prototype filter.
- The forward and inverse transformations (i.e. the analysis and synthesis filter banks) of the input signal xn is implemented by a `for` loop. The number of `ITER` is given by the number of samples in xn . At each iteration, the following operations are performed in sequence:
 - Update the input buffer x with a new input sample $xn(n)$. The oldest sample in the buffer $xn(end)$ is discarded.
 - Block transformations are performed once for every N input samples. This can be accomplished using the `mod(n, N)` function, which returns a 0 when the iteration index n is multiples of N . The result of the forward transform $H.^*x$ is assigned to an N -by-1 column vector x_D . Elements of the transformed vector

Table B.5 DFT filter bank as a block transform

```

addpath '..\common';
clear all;

N = 8;
k = 16;
L = 2*K*N;
hopt = fir1(L-1,1/N);
[H,F] = make_bank_DFT(hopt,N);
H = sqrt(N)*H; F = sqrt(N)*F;

xn = sin(2*pi*0.01*(0:1000));
ITER = length(xn);
yn = zeros(ITER,1);
x = zeros(length(H),1);
b = zeros(length(F),1);

for n = 1:ITER
    x = [xn(n); x(1:end-1)];
    if mod(n,N)==0
        x_D = H.'*x;
        %=====
        % Insert subband processing
        % algorithm here
        %=====
        b = F*x_D + b;
    end
    yn(n) = real(b(1));
    b = [b(2:end); 0];
end

figure;
plot(0:length(xn)-1,xn); hold on;
plot(0:length(yn)-1,yn,'r:');
xlabel('Time index, n'); ylabel('Amplitude');

```

x_D are in fact the subband signals, which serve as inputs to subband processing algorithms. The inverse transform is given by F^*x_D . The result is added to the output buffer b .

- (d) The fullband output sample $yn(n)$ is given by the first element $b(1)$ of the output signal buffer. After pushing out the first element, the output buffer b is then updated with a 0 sample (recall that an interpolator inserts zero samples) to be used in the next iteration. The input and output signals are shown in Figure B.8. Notice that the output signal is delayed by $L - 1 = 255$ samples due to the filter bank.

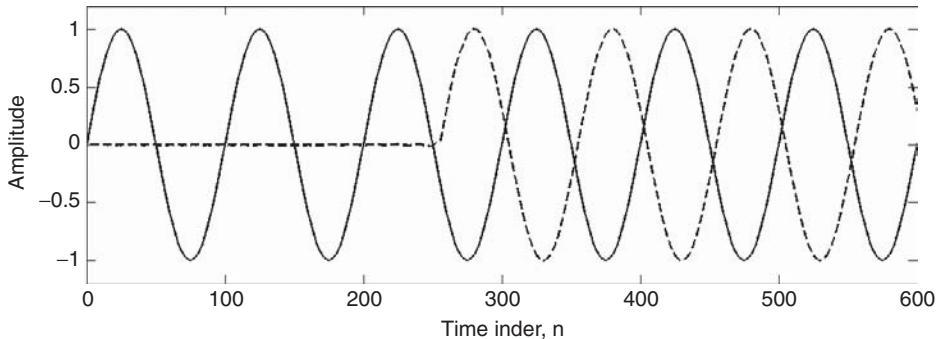


Figure B.8 The output signal (dashed line) is a delayed version of the input signal (solid line)

B.3.2 Implementation of a subband adaptive filter

An analysis filter bank decomposes a fullband signal into multiple subbands. The fullband signal can be reconstructed via a synthesis filter bank, as we have seen in Section B.3.1. In its conventional form, a subband adaptive filter (SAF) is constructed by inserting N adaptive subfilters (N is the number of subbands or channels), one between each of the analysis–synthesis filter pairs. Recall that an adaptive filter requires a reference signal $d(n)$ (or desired response) in addition to the input signal $u(n)$. Therefore, we need two analysis filter banks and one synthesis filter bank. The conventional form of the SAF is shown in Figure 4.1.

Available on the companion CD, the script and function M-files implementing the SAF are `SAFinit.m`, `SAFadapt.m` and `SAFdemo.m`. The first function, `SAFinit`, initializes the parameters (e.g. step size, initial weights, number of subbands, filter banks, etc.) of the SAF. The second function, `SAFadapt`, performs the actual computation of the SAF algorithm. The third M-file, `SAFdemo`, is provided as an example of using functions `SAFinit` and `SAFadapt`.

The function M-files `SAFinit.m` and `SAFadapt.m` are kept in the folder ‘`Common`’, while the demo script `SAFdemo.m` can be found in the folder ‘`Chapter4`’ on the companion CD. The structures of the `SAFdemo` and `SAFinit` are similar to that of the LMS algorithm presented in Section B.2.3. We will only focus on the `SAFadapt` function in this section. Partial listing of the MATLAB function is shown in Table B.6, where we focus only on the adaptation loop. The explanation for each block is given below. Figure B.9 shows the location of various signal vectors:

- (a) Similar to that in Section B.3.1.2, x and y are column vectors representing the signal buffers of analysis filter banks for the input signal u_n and desired response d_n , respectively. The length of these buffers is determined by the length L of the analysis and synthesis filters. For each iteration in the `for` loop, the signal buffers, x and y , are updated with new samples.
- (b) For the input signal, the transformed vector $x' * H$ is used to update the matrix U . Each column of U represents the signal vector for an adaptive subfilter. Similarly,

Table B.6 Partial listing of the SAFadapt function M-file

```

function [en,S] = SAFadapt (un,dn,S)

for n = 1:ITER
    x = [un(n); x(1:end-1)];
    y = [dn(n); y(1:end-1)]; (a)

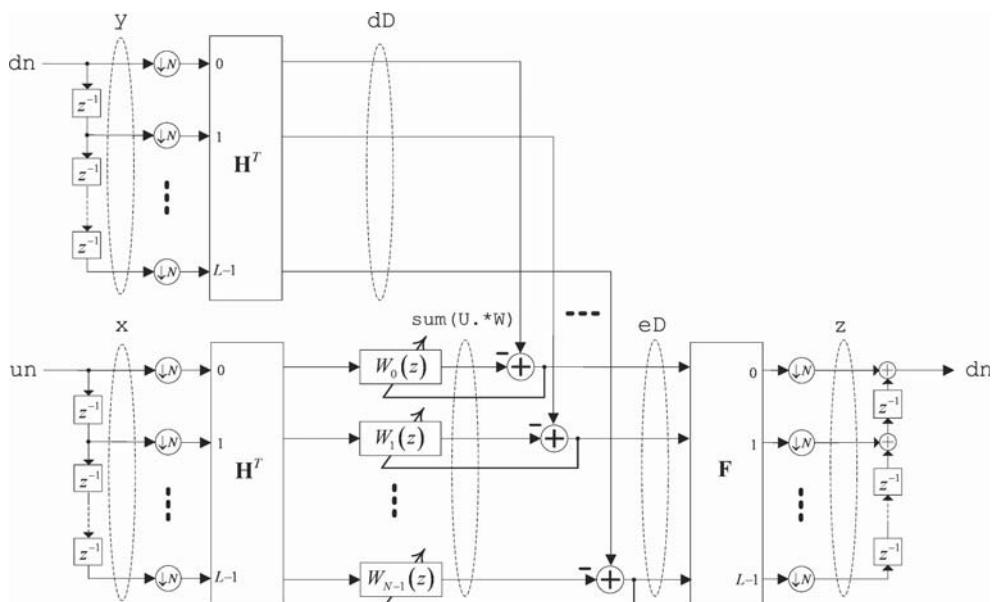
    if (mod(n-1,D)==0)
        U = [x'*H; U(1:end-1,:)];
        dD = y'*H;
        eD = dD - sum(U.*w);
    end (b)

    if n >= AdaptStart
        W = W + conj(U)*diag(eD./ (sum(U.*conj(U))+alpha))*mu;
        S.iter = S.iter + 1;
    end (c)

    eD = [eD, conj(fliplr(eD(2:end-1)))].';
    z = F*eD + z;
end (d)

en(n) = real(z(1));
z = [z(2:end); 0];
end

```

**Figure B.9** Subband adaptive filter using DFT filter banks

each column of the matrix \mathbf{w} represents the weight vector of an adaptive subfilter. The subband estimation errors e_D are computed once in every D iterations.

- (c) Tap-weight adaptation is performed using the NLMS algorithm (see Section 4.4.4). Remember that each column of \mathbf{w} represents an adaptive subfilter.
- (d) Inverse transformation of the subband estimation errors gives rise to the fullband error signal e_n . Notice that we only process the first $N/2 + 1$ subbands (instead of N subbands) by means of the complex-conjugate relation between subbands (see Section 4.1.1 for details). The estimation errors in the remaining subbands are obtained from other subbands using the complex-conjugate properties.

Appendix C

Summary of MATLAB scripts, functions, examples and demos

This appendix lists script and function M-files available on the companion CD and briefly describes the organization of these M-files and their tasks. These MATLAB programs illustrate examples given in main chapters, implement some adaptive algorithms and demonstrate some important applications. The codes have been developed and tested using MATLAB version 6.5.

We use three M-files to illustrate the adaptive algorithms introduced in the book. For example, the LMS algorithm introduced in Section 1.4 is implemented and demonstrated using the following three M-files:

```
LMSinit.m  
LMSadapt.m  
LMSdemo.m
```

The first function, `LMSinit`, initializes the FIR filter with the LMS algorithm. By default, the tap weights of the adaptive filter are initialized to zeros. The step size and leakage factor can also be set using the `LMSinit` function. The second function, `LMSadapt`, performs the actual computation of the LMS algorithm. The script M-file, `LMSdemo`, is provided as an example to use the `LMSinit` and `LMSadapt` functions for solving a system identification problem. The function M-files `LMSinit.m` and `LMSadapt.m` are kept in the common folder called `Common`, which contains commonly used functions and data files, while the demo script M-files, such as `LMSdemo.m`, are kept in the folder of the respective chapter where the algorithm is introduced. For the case of the LMS algorithm, the demo file can be found in the folder ‘`Chapter1`’ of the companion CD. We use the same convention for all the algorithms introduced in the book. To access the MATLAB functions stored in the common folder, we add the following statement in the demo M-files:

```
addpath '... \Common';
```

The `addpath` command adds the folder ‘`Common`’ to the MATLAB search path for the required function M-files and/or data files.

There are many MATLAB examples presented in the book to demonstrate some important concepts. The M-files for these MATLAB examples are named as `Example_cPn.m` for the `n`th example in Chapter `c`. For example, the M-file for Example 1.1 is called `Example_1P1.m`. We also introduce several applications of adaptive filtering in the book. These MATLAB applications include system identification (SYSID), acoustic echo cancellation (AEC), adaptive interference cancellation (AIC), adaptive line enhancer (ALE), active noise control (ANC), inverse system modeling (INVSYSID), adaptive channel equalizer (CH_EQ) and acoustic feedback reduction for hearing aids (HA). These MATLAB applications provide a useful platform to evaluate the performance of different adaptive algorithms. The M-files can be found in the folder for the respective chapter where the examples or applications are introduced.

We have tried to use only basic functions provided with the standard MATLAB package. However, some of the M-files included in the companion CD still require functions `filter`, `firl`, `impz`, `mscohere`, `remez`, `spectrogram`, `spectrum`, `upfirdn` and `xcorr` from the Signal Processing Toolbox (see Appendix A for details). A summary is given of the M-files used in the book in the following table.

M-file	Brief description
Chapter 1	
AEC.m	Demo of using the LMS and NLMS algorithms for acoustic echo cancellation
AIC.m	Demo of using the LMS and NLMS algorithms for adaptive interference cancellation. In AIC, a speech signal is corrupted by broadband noise and the purpose of the adaptive filter is to estimate the broadband noise and subtracted from the corrupted speech to form an enhanced speech
ALE.m	Demo of using the LMS and NLMS algorithms for adaptive line enhancement. ALE is an adaptive prediction problem, where we use an adaptive filter to separate broadband noise or speech from a narrowband signal
ANC.m	Demo of using the FXLMS and FXNLMS algorithms for active noise control
CH_EQ.m	Perform adaptive channel equalization with an initial training mode and switching to a decision-directed mode subsequently. During the training mode, the desired signal is derived from a delayed version of the input signal. During the decision-directed mode, the desired signal is a sign operator of the adaptive filter output
APdemo.m	Perform a simple system identification using the AP-based adaptive FIR filter
Example_1P1.m	Plot the mean-square error (MSE) of a two-tap adaptive FIR filter
FDAFdemo.m	Demo for the transform-domain adaptive filter
HA.m	Demo of using the LMS and NLMS algorithms for acoustic feedback cancellation in hearing aids

M-file	Brief description
INVSYSID.m	Perform inverse system identification with only the training mode; the desired signal is derived from a delayed version of the input signal
LMSdemo.m	Adaptive system identification using the FIR filter with the LMS algorithm
NLMSdemo.m	Adaptive system identification using the FIR filter with the NLMS algorithm
RLSdemo.m	Adaptive system identification using the FIR filter with the RLS algorithm
SOAFdemo.m	Demo of using the self-orthogonalizing adaptive filter
SYSID.m	Perform adaptive system identification using the LMS and NLMS algorithms
Chapter 2	
Example_2P1.m	Decimation
Example_2P2.m	Interpolation
Example_2P3.m	Filter bank
Example_2P4.m	Polyphase implementation of type-I and type-II polyphase filter banks
Example_2P5.m	Filter bank as block transform with memory
Example_2P6.m	Frequency response and distortion function of filter bank
Example_2P7.m	DFT filter bank and its frequency response
Chapter 3	
Example_3P1.m	Correlation-domain formulation
Example_3P2.m	Cross power spectrum between two analysis filters
Example_3P3.m	Decimated subband spectrum
Example_3P4.m	Orthogonality of DCT
Example_3P5.m	Subband orthogonality of cosine-modulated filter banks
Chapter 4	
DSAFdemo_	Closed-loop delayless subband adaptive filter demo for the algorithm proposed by Merched <i>et al.</i>
Merched_c1lp.m	Closed-loop delayless subband adaptive filter demo for the algorithm proposed by Morgan and Thi
DSAFdemo_	Spectral dynamic range of fullband and subband signals after critical decimation
Example_4P1.m	Aliasing and band-edge effects
Example_4P2.m	Spectrum equalization in MSAF
Example_4P3.m	Frequency stacking for delayless SAF
Example_4P4.m	N th-band filter
Example_4P5.m	Aliasing and band-edge effects (convergence analysis)
Example_4P6.m	Delayless SAFs (convergence analysis)
Example_4P7.m	Delayless SAFs under speech excitation
Example_4P8.m	Subband adaptive filter demo
SAFdemo.m	
Chapter 5	
ECdemo.m	Echo cancellation demonstration using (i) the NLMS, (ii) IPNLMS and (iii) PMSAF algorithms under different echo-path impulse responses

M-file	Brief description
Example_5P1.m	Oversampled filter bank using relaxed analysis/synthesis filter design constraints
Example_5P2.m	Cascade QMF analysis and synthesis filters
Example_5P3.m	32-channel PQMF for the MPEG-1 audio coder
IPNLMSdemo.m	Improved proportionate normalized LMS (IPNLMS) algorithm demo
PMSAFdemo.m	Proportionate multiband-structured subband adaptive filter (PMSAF) demo
Chapter 6	
Example_6P1.m	Polyphase implementation
Example_6P2.m	Implementation of the MSAF algorithm
Example_6P3.m	Convergence of the MSAF algorithm under white and colored excitations
Example_6P4.m	Decorrelation property of the MSAF algorithm
Example_6P5.m	Comparison of subband and time-domain constrained adaptive filters under stationary and colored excitations
Example_6P6.m	Comparison of subband and time-domain constrained adaptive filters under speech excitation
MSAFdemo.m	Multiband-structured SAF (MSAF) demo
MSAFdemo_c1lp.m	Closed-loop delayless multiband-structured SAF demo
MSAFdemo_o1lp.m	Open-loop delayless multiband-structured SAF demo
PRAdemo.m	Partial-rank algorithm (PRA) demo
Chapter 7	
Example_7P1.m	Mean of projection matrix
Example_7P2.m	Mean-square performance of the MSAF algorithm
DecorrFilter.m	Decorrelation properties of the MSAF algorithm
ParaunitaryCMFB.m	Generate coefficients of the paraunitary cosine-modulated filter bank
make_bank_para.m	Creates filters for a paraunitary cosine-modulated filter banks with N subbands
ButterFlyAngles_0p8.m	Butterfly angles for the stopband edge at $0.8\pi/N$
ButterFlyAngles_1p0.m	Butterfly angles for the stopband edge at $1.0\pi/N$
ButterFlyAngles_1p2.m	Butterfly angles for the stopband edge at $1.2\pi/N$
Appendix A	
DelayANC.m	Adaptive noise cancellation using MATLAB functions provided in the Filter Design Toolbox
Appendix B	
Table_BP4.m	Decimation and interpolation processes
Table_BP5.m	DFT filter bank as a block transform with memory
Common M-files	
AmpResp.m	Calculate the amplitude response of a symmetric FIR filter

M-file	Brief description
APadapt.m	Affine projection (AP) algorithm
APinit.m	Initialize parameter structure for the affine projection (AP) algorithm
DCTmatrix_TypeIV.m	Construct an N -by- N type-IV DCT matrix
dist_alias.m	Calculate and plot distortion and aliasing of filter banks
DSAAdapt_Merched_cllp.m	Closed-loop delayless subband adaptive filter using the weight transformation proposed by Merched <i>et al.</i>
DSAFinit_Merched_cllp.m	Initialize the parameter structure for the delayless subband adaptive filter proposed by Merched <i>et al.</i>
DSAdapt_Morgan_cllp.m	Closed-loop delayless subband adaptive filter using the weight transformation proposed by Morgan and Thi
DSAFinit_Morgan_cllp.m	Initialize the parameter structure for the delayless subband adaptive filter proposed by Morgan and Thi
fdaf.m	Frequency-domain adaptive filter algorithm
FDAFadapt.m	Frequency-domain adaptive filter (FDAF) algorithm
FDAFinits.m	Initialize parameter structure for the FDAF algorithm
FreqResp.m	Calculate the frequency response of the FIR system using an N -point FFT
FXLMSadapt.m	Filtered-X LMS (FXLMS) algorithm
FXLMSinit.m	Initialize the parameter structure for the FXLMS algorithm
FXNLMSadapt.m	Normalized FXLMS algorithm
FXNLMSinit.m	Initialize the parameter structure for the FXNLMS algorithm
GenerateResponse.m	Generate input and desired responses
GenerateResponse_speech.m	Generate input and desired responses with a speech signal
HALMSadapt.m	Adaptive FIR filter with the LMS algorithm for hearing aids
HALMSinit.m	Initialize the parameter structure of the LMS algorithm for hearing aids
HANLMSadapt.m	NLMS algorithm for hearing aids
HANLMSinit.m	Initialize the parameter structure of the NLMS algorithm for hearing aids
IPNLMSadapt.m	Adaptive FIR filter with the IPNLMS algorithm
IPNLMSinit.m	Initialize the parameter structure for the IPNLMS algorithm
LMSadapt.m	Adaptive FIR filtering with the LMS algorithm
LMSadapt_dec.m	LMS algorithm for decision-directed channel equalization
LMSinit.m	Initialize the parameter structure for the LMS algorithm
Make_bank.m	Generate an analysis filter bank using cosine modulation
make_bank_3.m	Standard type of cosine modulation where the phase reference is $(L - 1)/2$
make_bank_DFT.m	Generate a DFT filter bank with N subbands
MSAFadapt.m	Multiband-structured SAF (MSAF) algorithm
MSAFadapt_cllp.m	Closed-loop implementation of the delayless multiband-structured SAF
MSAFadapt_oplp.m	Open-loop implementation of the delayless multiband-structured SAF

M-file	Brief description
MSAFinit.m	Initialize the parameter structure for the MSAF algorithm
NLMSadapt.m	Adaptive FIR filter with the NLMS algorithm
NLMSadapt_dec.m	NLMS algorithm for decision-directed channel equalization
NLMSinit.m	Initialize the parameter structure for the NLMS algorithm
opt_filter.m	Create a prototype lowpass filter for the pseudo-QMF filter bank with N subbands
PMSAFadapt.m	Proportionate multiband-structured subband adaptive filter (PMSAF)
PMSAFinint.m	Initialize the parameter structure for the PMSAF algorithm
PRAadapt.m	Partial-rank algorithm (PRA)
PRAinit.m	Initialize the parameter structure for the PRA
PSDecimation.m	Decimation of power spectrum density
recursive_power.m	Recursive estimation of signal power
RLSadapt.m	Recursive least-squares algorithm (RLS)
RLSinit.m	Initialize the parameter structure for the RLS algorithm
SAFadapt.m	Subband adaptive filter (SAF)
SAFinint.m	Initialize the parameter structure for the subband adaptive filter
smooth.m	Smoothing input signal using moving averaging
SOAFadapt.m	Adaptive FIR filtering with the self-orthogonalizing (SOAF) algorithm
SOAFadapt_DFT.m	Adaptive FIR filtering using the SOAF (DFT) algorithm
SOAFinit.m	Initialize the parameter structure for the SOAF algorithm
Stepcheck_lms.m	Performs step-size checking
WeightTransform_Merched.m	Mapping of subband adaptive filter coefficients to a fullband filter using the method proposed by Merched <i>et al.</i>
WeightTransform_Morgan.m	Mapping of subband adaptive filter coefficients to a fullband filter using the method proposed by Morgan and Thi
Common data files	
engine_1.wav	Engine noise for active noise control
engine_2.wav	
far.wav	Far-end speech for acoustic echo cancellation
fb.dat	Feedback path for hearing aids
h1.dat	Room impulse response (small room)
h2.dat	Room impulse response (big room)
Homer.mat	AR parameters used in the simulation
near.wav	Near-end speech for acoustic echo cancellation.
p_p.asc	Primary path (denominator) for active noise control
p_z.asc	Primary path (numerator) for active noise control
s_p.asc	Secondary path (denominator) for active noise control
s_z.asc	Secondary path (numerator) for active noise control
SparseImpulse_512_32.mat	Sparse impulse response
speech.mat	Speech files
SpeechSample.mat	
Timit.wav	

Appendix D

Complexity analysis of adaptive algorithms

This section summarizes the adaptive algorithms presented in this book and analyzes their computational complexities in terms of the number of multiplications, additions/subtractions and some special operations. In addition, we also refer to the MATLAB files that are used in this book. The adaptive algorithms are separated into three different parts including (i) filtering, (ii) error computation and (iii) weight updating, to highlight the computational complexity and memory usage in each section. Furthermore, we include the data memory usage that is an important consideration when implementing adaptive algorithms in embedded systems with limited memory. This breakdown of computational complexity and memory usage across all algorithms provides a more accountable benchmark, which can be verified by readers and allows them to identify the computational and memory hot-spots in adaptive algorithms. As there are many approaches and fast algorithms in performing computation of different tasks, our analysis is based on the computational complexity and memory usage of the MATLAB implementation in this book. Other implementation tricks can be further exploited by readers who are skilled in the digital implementation.

Tables D.1, D.2 and D.3 summarize the computational complexity and memory usage of time-domain adaptive filters, frequency-domain adaptive filters and subband adaptive filters, respectively.

Table D.1 Computational complexity and data memory usage of time-domain adaptive filters

Adaptive algorithms	Operations	Computational load (in terms of multiplications, additions, divisions and other special operations for every new data sample. All computation is carried out in real-valued arithmetic)	Data memory usage (in units of B -bit words) (Note: sample processing is assumed in this memory usage profile, unless otherwise stated.)
LMS algorithm using an M -tap FIR adaptive filter Refer to: LMSinit.m LMSadapt.m	Filtering: $y(n) = \mathbf{w}^T(n)\mathbf{u}(n).$ Error signal: $e(n) = d(n) - y(n).$ Coefficients update: $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\mathbf{u}(n)e(n).$	M multiplications, ($M - 1$) additions 1 addition (or subtraction) $(M+1)$ multiplications, M additions	M words for signal vector $\mathbf{u}(n),$ M words for coefficient vector $\mathbf{w}(n)$ and 1 word for output signal $y(n)$ 1 word each for $e(n)$ and $d(n)$
NLMS algorithm using an M -tap FIR adaptive filter Refer to: NLMSinit.m NLMSadapt.m	Filtering: $y(n) = \mathbf{w}^T(n)\mathbf{u}(n).$ Error signal: $e(n) = d(n) - y(n).$ Coefficient update: $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \frac{\mathbf{u}(n)}{\mathbf{u}^T(n)\mathbf{u}(n)} e(n).$	M multiplications, ($M - 1$) additions 1 addition (or subtraction) $(2M + 1)$ multiplications and $2M$ additions	1 word for μ and in-place computation is used for updating tapped delay-line and coefficient vectors $(2M + 4)$ words

<p><i>Total operations/sample = $O(M)$</i></p> <p>$(3M + 1)$ multiplications, $(3M - 1)$ additions, 1 division</p>	<p>$(2M + 5)$ words</p>
<p>AP algorithm using an M-tap FIR adaptive filter, with P constraints. Consider that P constraints are less than M taps (underdetermined)</p> <p>Refer to: APinit.m APadapt.m</p>	<p>Data matrix ($M \times N$): $\mathbf{A}^T = [\mathbf{u}(n), \mathbf{u}(n-1), \dots, \mathbf{u}(n-P+1)]$</p> <p>Error computation: $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n)$. Note: error computation is performed based on P constraints.</p> <p>Coefficient update: $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \mathbf{A}^T(n)(\mathbf{A}(n)\mathbf{A}^T(n))^{-1}\mathbf{e}(n)$.</p>
	<p>M words for signal vector $\mathbf{u}(n)$, thus $M \times P$ words are required for saving the data matrix \mathbf{A}.</p> <p>M words each for $\mathbf{e}(n)$ and $\mathbf{d}(n)$, M words for weight vector $\mathbf{w}(n)$</p> <p>P words for $(M \times N)$ matrix, M words for weight vector $\mathbf{w}(n)$</p> <p>P^2 words for housing inverse matrix operations. In-place computation can be applied to the updating term. M words for updating vector</p>
	<p><i>Total operations/sample</i></p> $ \begin{aligned} &= O(P^3) + O(P^2M) \\ &\approx O(P^2M) \\ &(2P^2M + 2PM + M) \text{ multiplications,} \\ &(2P^2M + PM - P^2) \text{ additions,} \\ &\text{and } O(P^3) \text{ matrix inverse operations} \end{aligned} $

Continued overleaf

Table D.1 (Continued)

Adaptive algorithms	Operations	Computational load (in terms of multiplications, additions, divisions and other special operations for every new data sample. All computation is carried out in real-valued arithmetic)	Data memory usage (in units of B -bit words) (Note: sample processing is assumed in this memory usage profile, unless otherwise stated.)
RLS algorithm using an M -tap FIR adaptive filter. λ is the forgetting factor that is equal to unity in this case	Filtering: $y(n) = \mathbf{w}^T(n)\mathbf{u}(n)$	M multiplications, $(M - 1)$ additions	M words for signal vector $\mathbf{u}(n)$, M words for coefficient vector $\mathbf{w}(n)$ and 1 word for output $y(n)$
Refer to: RLSinit.m RLSadapt.m	Error signal: $e(n) = d(n) - y(n).$	1 addition (or subtraction)	1 word each for $e(n)$ and $d(n)$
	Adaptive gain computation: $\mathbf{r}(n) = \mathbf{P}(n - 1)\mathbf{u}(n),$ $\mathbf{r}(n)$ $\mathbf{g}(n) = \frac{\lambda + \mathbf{u}^T(n)\mathbf{r}(n)}{\lambda + \mathbf{u}^T(n)\mathbf{r}(n)}.$	M^2 multiplications, $M(M - 1)$ additions. $2M$ multiplications, M additions and 1 division	M^2 words for matrix $\mathbf{P}(n)$ M words for vector $\mathbf{r}(n)$, 1 word each for λ and $\mathbf{u}^T(n)\mathbf{r}(n)$, M words for vector $\mathbf{g}(n)$
	Coefficient update: $\mathbf{w}(n + 1) = \mathbf{w}(n) + \mathbf{g}(n)e(n). (M \times 1)$	M multiplications, M additions	In-place computation for weight update
	Inverse correlation matrix update: $\mathbf{P}(n) = (\mathbf{P}(n - 1) - \mathbf{g}(n)\mathbf{u}^T(n)\mathbf{P}(n - 1)).(M \times M)$	M^2 multiplications, M^2 additions, (Note: $\mathbf{r}(n)$ has been computed in adaptive gain computation.)	M^2 words for updating matrix
<i>Total operations/sample</i> = $O(M^2)$ $(2M^2 + 4M)$ multiplications, $(2M^2 + 2M)$ additions and 1 division			$(2M^2 + 4M + 5)$ words

Table D.2 Computational complexity and data memory usage of transform-domain adaptive filters

Adaptive algorithms	Operations	Computational load (in terms of real multiplications, additions, divisions and other special operations for every new data sample. All computation is carried out in real-valued arithmetic)	Data memory usage (in units of B -bit words) (Note: sample processing is assumed in this memory usage profile, unless otherwise stated.)
Frequency domain adaptive filter (FDAF) using a $2M$ -point FFT and overlap-save method with 50 % overlap (with gradient constraint). Refer to Figure 1.14 for details	Filtering: $\mathbf{Y}(n) = \mathbf{W}(n) \cdot \mathbf{U}(n)$ Note: frequency-domain multiplications Refer to: <code>FDAFinit.m</code> <code>FDAFadapt.m</code>	4 $M \log_2 2M$ multiplications and 4 $M \log_2 2M$ additions for $\text{FFT}[\mathbf{u}(n)]$ 8 M multiplications and 4 M additions for (element-by-element) vector multiplication.	4 M words each for $\mathbf{Y}(n)$, $\mathbf{W}(n)$ and $\mathbf{U}(n)$
Error signal: $\mathbf{e}(n)_M = \mathbf{d}(n)_M - \text{IFFT}[\mathbf{Y}(n)]$.		4 $M \log_2 2M$ multiplications and 4 $M \log_2 2M$ additions for $\text{IFFT}[\mathbf{Y}(n)]$; 2 M additions (subtractions)	2 M words each for $\mathbf{e}(n)_M$ and $\mathbf{d}(n)_M$
Coefficient update: $W_k(n+M) = W_k(n) + \mu U_k^*(n) E_k(n), k = 0, 1, \dots, 2M-1.$		4 $M \log_2 2M$ multiplications and 4 $M \log_2 2M$ additions for $\text{FFT}[\mathbf{e}(n)]$ 8 $M \log_2 2M$ multiplications and 8 $M \log_2 2M$ additions for gradient constraint ^a .	4 M words for $E_k(n)$ and (4 M +1) words for updating $\mu U_k^*(n) E_k(n)$ In-place computation for weight update

Continued overleaf

Table D.2 (*continued*)

Adaptive algorithms	Operations	Computational load	Data memory usage
		(in terms of real multiplications, additions, divisions and other special operations for every new data sample. All computation is carried out in real-valued arithmetic) (^a Note: For the constrained frequency-domain adaptive filter only. Ignore this computation for unconstrained frequency-domain adaptive filters.)	(in units of B -bit words) (Note: sample processing is assumed in this memory usage profile, unless otherwise stated.)
		$Total\ operations/M\ samples = O(M \log_2 2M) + O(M)$ $(20M \log_2 2M + 18M)$ multiplications and $(20M \log_2 2M + 14M)$ additions (Note: computation complexity of the FDAF algorithm is performed in a block of M samples.) $Total\ operations/sample = O(\log_2 2M)$	$(24M + 1)$ words
Self-orthogonalizing adaptive filter (SOAF) using DCT. Refer to Figure 1.15 for details	Transformation of input vector: $\mathbf{U}(n) = \text{DCT}[\mathbf{u}(n)]$. Output signal: $y(n) = \mathbf{w}^T(n)\mathbf{U}(n).$	M^2 multiplications and $M(M - 1)$ additions for $\text{DCT}[\mathbf{u}(n)]$ M multiplications, $(M - 1)$ additions	M^2 words for DCT basis and M words for signal vector $\mathbf{u}(n)$ M words for coefficient vector $\mathbf{w}(n)$ and 1 word for output signal $y(n)$. (Note: $\mathbf{U}(n)$ and $\mathbf{u}(n)$ share the same memory.)
Refer to: <code>SOAFinit.m</code> <code>SOAFadapt.m</code>	Power normalization:	$U'_k(n) = \frac{U_k(n)}{\sqrt{P_k(n) + \delta}},$ $P_k(n) = \beta P_k(n - 1)$ $+ (1 - \beta) U_k^2(n),$ $k = 0, 1, \dots, M - 1.$	M words each for power estimation $P_k(n)$ and $U'_k(n)$ 2 words for β and $(1 - \beta)$

Error signal: $e(n) = d(n) - y(n)$.	1 addition (or subtraction)	1 word each for $e(n)$ and $d(n)$
Coefficients update: $W_k(n+1) = W_k(n) + \mu_k$ $U'_k(n)e(n)$, for $k = 0, 1, \dots, M-1$.	$(M+1)$ multiplications, M additions	M words for μ_k
Total operations/sample = $O(M^2) + O(M)$ $(M^2 + 5M + 1)$ multiplications, $(M^2 + 3M)$ additions, M divisions and M square-root operations	$(M^2 + 5M + 5)$ words	

Table D.3 Computational complexity and data memory usage of subband adaptive filters

Adaptive algorithms	Operations	Computational load (in terms of multiplications, additions, divisions and other special operations for every new data sample. All computation is carried out in real-valued arithmetic)	Data memory usage (in units of B -bit words) (Note: sample processing is assumed in this memory usage profile, unless otherwise stated.)
Subband adaptive filter (SAF)	<p>Analysis filters are used to partition the input and desired signals</p> <p>A synthesis filter is used to reconstruct the error signal</p> <p>(Note: analysis and synthesis filters are implemented as a block transform. The forward and reverse transformation are performed once for every D input samples (see Appendix B).)</p>	$2NL/D$ multiplications, $N(L - 1)/D$ additions NL/D multiplications, $2N(L - 1)/D$ additions	$2NL$ words for two sets of analysis filter coefficients and $2NL$ words for their subband signal vectors
Refer to: <code>SAFInit.m</code> <code>SAFadapt.m</code>	<p>Subband filtering:</p> $y_{i,D}(n) = \mathbf{w}_i^T(k)\mathbf{u}_{i,D}(k)$ <p>for $i = 0, 1, \dots, N - 1$.</p> <p>(Note: subband filtering is performed at a sampling frequency of f_s/D. For comparison with the fullband filter, complexity is computed at f_s and $M_S = M/D$.)</p>	MN/D^2 multiplications, MN/D^2 additions	MN/D words for subband filter coefficients $\mathbf{w}_i(k)$; MN/D words for signal vector $\mathbf{u}_{i,D}(k)$

	Error estimation: $\mathbf{e}_D(k) = \mathbf{d}_D(k) - \mathbf{y}_D(k)$	N/D additions	N words each for $\mathbf{d}_D(k)$, $\mathbf{y}_D(k)$ and $\mathbf{e}_D(k)$
	Subbands' coefficients update: $\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \frac{\mu \mathbf{u}_i(k)}{\ \mathbf{u}_i(k)\ ^2 + \alpha} e_{i,D}(k)$ for $i = 0, 1, \dots, N-1$	$2MN/D^2$ multiplications, $2MN/D^2$ additions (Neglect scalar multiplication and division at the decimated rate.)	1 word each for μ, α and $\ \mathbf{u}_i(k)\ ^2$, and in-place computation is used for updating tapped delay-line and coefficient vectors
	Total operations/sample = $O(NL/D) + O(MN/D^2)$ $(3NL/D + 3MN/D^2)$ multiplications, $[3N(L-1)/D + 3MN/D^2 + N/D]$ additions		$[6NL + N(2M/D + 3) + 3]$ words
	If $D = N$ (critical sampling), $O(L) + O(M/N)$ $(3L + 3M/N)$ multiplications, $[3(L-1) + 3M/N + 1]$ additions		If $D = N$ (critical sampling), $(6NL + 2M + 3N + 3)$ words
Multiband-structured subband adaptive filter (MSAF) with critical sampling ($D = N$). Refer to Figure 6.3 for details	Analysis and synthesis: $\mathbf{U}_1^T(k) = \mathbf{H}^T \mathbf{A}(kN)$, $\mathbf{d}_D(k) = \mathbf{H}^T \mathbf{d}(kN)$ $\mathbf{e}(kN) = \mathbf{F}\mathbf{e}_D(k)$	$(N+1)L$ multiplications, $(N+1)(L-1)$ additions $(NL)/N = L$ multiplications, $(L-1)$ additions	NL words each for \mathbf{H} , $\mathbf{A}(kN)$ and \mathbf{F} ; NM words for $\mathbf{U}^T(k)$; N words each for $\mathbf{d}(kN)$ and $\mathbf{e}_D(k)$ These N locations are reused in $\mathbf{d}_D(k)$ and $\mathbf{e}(kN)$
Refer to: <code>MSAFinit.m</code> <code>MSAFadapt.m</code>	(Note: analysis and synthesis filters are implemented as a block transform.)		

Table D.3 (*Continued*)

	Error estimation: $\mathbf{e}_D(k) = \mathbf{d}_D(k) - \mathbf{U}^T(k)\mathbf{w}(k)$	$NM/N = M$ multiplications M additions	N words to save $\mathbf{U}^T(k)\mathbf{w}(k)$
Normalization matrix: $\Lambda(k) = \text{diag}[\mathbf{U}^T(k)\mathbf{U}(k) + \alpha\mathbf{I}]$	$(NM)/N = M$ multiplications, M additions (Note: $\mathbf{U}^T(k)\mathbf{U}(k)$ can be approximated with a diagonal matrix; i.e. the diagonal assumption, and the off-diagonal elements need not be computed in the normalization matrix.)	N words to save $\mathbf{U}^T(k)\mathbf{U}(k)$, as the off-diagonal elements are negligible; 1 word for α	
Tap-weight adaptation: $\mathbf{w}(k+1) = \mathbf{w}(k) + \mu\mathbf{U}(k)\boldsymbol{\Lambda}^{-1}(k)\mathbf{e}_D(k)$	$(2N + NM)/N = (M+2)$ multiplications, M additions (Note: the operation $\boldsymbol{\Lambda}^{-1}(k)\mathbf{e}_D(k)$ is considered as N multiplications.)	M words for holding the updating vector and 1 word for μ	$[M(N+1) + N(3L+4)+2]$ words

Index

A

- Acoustic
echo cancellation, 15–18, 31, 38, 106, 114, 134, 137, 156, 167, 232, 302
feedback cancellation, 15, 21–22, 30, 302
noise, 18–19
- Active noise control, 15, 18–19, 114, 232, 302
- Adaptive
algorithm, 1–2, 6–13, 15, 31–37, 111, 126, 134, 136–139, 159, 175–176, 186, 203, 230, 270–272, 282–285, 307
array processing, 28–30, 233
channel equalizer, 25–28, 302
cross-filter, 106, 108–110, 135, 138, 230
filtering, 1–2, 14, 15, 28, 30, 31, 38, 99, 139, 270–272, 282, 302
inverse modeling/inverse system identification, 15, 25–27, 30, 302
line enhancer, 15, 24, 26, 30, 302
noise (interference) cancellation, 15, 19, 20–21, 272–277, 302
notch filter, 25
prediction, 15, 23, 30, 302
subfilter, 57, 99–111, 114–117, 133, 138, 139, 203, 230–231, 297
- system identification (modeling), 15–17, 20–22, 30, 99, 106, 108, 136, 140, 157, 192, 228, 231, 232, 287, 302
- Adaptive filter
frequency-domain, 31–34, 307, 311–312
structure, 1–3, 31, 38, 104
time-domain, 1–2, 31, 33, 35, 39, 307–308
- Affine projection (AP) algorithm, 9–12, 134–136, 139, 158, 178–179, 195–198, 230, 302, 305, 309
- Aliasing
cancellation, 151, 154, 228
component, 46–48, 61, 66, 106–116, 144–146, 151
distortion, 100, 113, 129, 134, 144, 147, 150
effect, 106–116, 125, 129, 138, 147, 150, 189, 232
error, 153, 155, 227, 232–233
in-band (*see also* inband aliasing)
- Amplitude distortion, 26, 61, 228
- Analysis filter bank, 38, 44, 47–49, 54–55, 93, 104–105, 116, 135, 168, 206, 228, 297
- Analysis-synthesis
filter product, 145, 147
framework, 150
system, 59, 63, 93, 206

- Anti-aliasing condition, 153
 Autocorrelation function, 73, 75, 77, 80, 83, 85, 102, 183–184, 188, 213–214
 Autocorrelation matrix, 4–5, 7, 11, 12, 13–14, 56, 101, 107, 137, 138, 180, 214, 220
 ill-conditioned, 14, 107
 subband, 106, 213
 weighted, 213, 219, 220
 Autoregressive model, 77, 80, 92, 137
- B**
 Bandwidth, 44, 82, 102, 139, 140, 142, 143, 147, 155, 167, 267
 Bandpass
 filter, 30, 38, 39, 44, 55, 63, 82, 116, 141, 294
 sampling theorem, 140
 Band-edge effect, 57, 103, 106–125, 129, 139, 189
 Block processing, 31, 32
 Block transform, 55, 294
 implementation, 58, 295, 314–315
 with memory, 55, 175, 294
 Broadband noise, 21, 23–24, 302
- C**
 Channel equalization, 25, 27, 30, 232, 302
 Closed-loop, 104–105
 delayless structure, 106, 113–115, 123–124, 126, 128–129, 189, 191, 227, 303
 Colored noise, 77, 80, 83, 137, 160
 Complex conjugate, 33, 63, 66, 101, 116, 123, 141
 Complex LMS algorithm, 33, 36
 Complex modulated/modulation, 63, 65, 66, 116, 141–144, 153, 295
 Complex-valued subband signal, 106, 138, 142
 Computational complexity (cost), 9, 10, 12, 63, 100, 124, 135, 137, 139, 144, 150, 160, 175, 178, 233, 307–316
 Condition number, 13–14, 214, 219–220
 Constraint,
 $2N$ th band, 60, 62
 filter design, 146–148, 151–152
 flatness, 59, 62, 64
 gradient (*see also* gradient constraint)
 optimization, 173, 175
 orthogonality, 62
 time, frequency and subband domain, 145, 178, 195, 199, 217
 Convergence rate (curve), 2, 6, 8, 9, 14, 56–57, 103, 125, 134, 136, 139–140, 155, 157, 160, 194, 220, 232–233, 285
 Convolution
 circular, 32, 33
 fast, 33
 linear, 31, 32, 74, 124, 208
 Conjugate quadrature filter, 155
 Control matrix, 156–157
 Correlation, 22, 23, 55
 auto (*see also* autocorrelation matrix)
 circular, 32
 coefficient, 82, 93
 cross (*see also* cross-correlation vector)
 domain, 73–79, 86, 89, 102, 183, 208
 linear, 32–33
 Cosine modulated/modulation, 60, 63, 67–68
 critical decimation, 77–79
 filter bank, 45, 53, 59–68, 80, 89–95, 101, 112, 140, 187, 193, 196, 219, 234
 Critically-sampled
 affine projection algorithm, 134
 decimated, 41, 45, 54, 74, 113, 116, 120, 167, 190, 204, 208, 230
 subband adaptive filter, 57, 100, 106–110, 113, 125, 133–136, 138, 139, 146, 161, 230, 232

- Cross-correlation
 function, 74, 75, 77–82, 85–88,
 93–95, 187, 208
 vector, 4, 5, 12, 180
- Cross-energy spectrum, 79–82, 90–96
- Cross filter, 1065, 108–110, 135, 138,
 227
- Crossover frequency, 145, 146
- D**
- Decimate/decimation, 39, 42, 44,
 50–52, 77, 83, 106, 150, 156,
 168, 170–171, 183, 204, 227,
 244, 292–293
 factor, 42, 135, 136, 139, 140, 141,
 143, 144, 146, 147, 149
 graphical interpretation, 83
 linear operation, 43
 rate, 99–100, 57, 99, 175, 183, 190
 ratio, 139
 spectrum, 83
 subband signal, 38, 44, 46, 58, 77,
 83, 100, 138, 144, 208
- Decision-directed mode, 27–28, 31,
 302
- Decision feedback, 27
- Decorrelate, 22, 24, 35, 55, 82
- Decorrelation filter, 181, 186, 194–195
- Delay, 22–23, 24–26, 30, 54–55, 57,
 105, 134, 135, 144, 145, 150,
 151, 213, 228, 231, 233
 chain, 49, 50, 55, 58, 206
 signal path, 38, 114, 129, 219, 233
- Delayless subband adaptive filter, 38,
 56, 106, 113–116, 119, 123–124,
 126–129, 189–192, 234
- Desired signal, 17, 20–32, 36, 38, 133,
 137, 144, 228, 272–273, 287
 subband, 104, 115
 vector, 6, 10
- Deterministic, 39, 75, 80, 85, 87, 178,
 192
- Diagonal assumption, 174, 177, 181,
 187, 193, 212, 216–218, 234
- Digital filter, 1–2, 258, 268, 282, 284
- Discrete cosine transform (DCT),
 35–37, 55, 138
- LMS algorithm, 137–138, 312
- matrix, 87–88
- transformed variable, 87
- Discrete Fourier transform (DFT),
 35–36, 39, 55
- coefficients, 116–119
- filter bank, 65–67, 101, 104, 107,
 116, 120, 125–126, 141–142,
 144, 148, 228, 230, 294–296
- matrix, 207
- modulated bandpass filter, 141
- Distortion transfer function (*see also*
 filter bank distortion transfer
 function)
- Disturbance, 15, 20, 175, 208, 211,
 217–218
- Double-filtering scheme, 135
- Double-talk condition, 17
- Downsample, 138, 140, 244
- E**
- Echo return loss, 15, 17
- Eigen-decomposition, 137
- Eigenvalue, 7, 14, 57, 106, 125, 137,
 138, 213
 spread, 13, 14, 26, 35, 101, 103,
 137, 139, 147, 155, 214, 220,
 232
 upper bound, 103
- Eigenvector, 137
- Electroacoustic domain, 19
- Energy conservation relation, 214–216,
 234
- Ensemble averaging, 125, 128, 160,
 193, 222
- Error signal, 1–4, 6, 9–11, 15, 19–30,
 33, 36, 38–39, 57, 106, 110, 135,
 136, 179, 218, 231, 282, 285, 290
- estimation error, 57, 116, 183,
 190–192, 206, 209–210, 215,
 232, 285, 299
- fullband, subband, 39, 99, 104–105,
 113–115, 126, 136, 139, 169,
 181–184, 190–191, 205
- microphone, 18–19
- preconditioned, 184–185
- Euclidean norm, 173, 178, 215

- Even-channel stacking, 141
 Excess mean-square error, 14,
 210–211, 217–218, 222
 Extended lapped transform, 63
- F**
 Fast Fourier transform (FFT)/Inverse FFT (IFFT), 31–33, 116, 123, 266
 Fast transversal filter, 12, 269
 Feedback/feedforward path, 22
 Filter bank, 38, 133, 140
 aliasing cancellation, 61, 151
 aliasing free, 46, 47
 amplitude distortion, 61, 228
 analysis, (*see also* analysis filter bank)
 complex-modulated, 66, 116, 142
 cosine-modulated (*see also* cosine-modulated filter bank)
 critically (maximally) decimated
 (*see also* critically decimated)
 definition, 44
 design, 86, 141–148, 187, 227
 DFT (*see also* DFT filter bank)
 distortion transfer function, 48, 55,
 60–61, 64–65
 input-output relation, 46–47, 151,
 155
 linear-time invariant, 47, 74, 183
 magnitude distortion, 48, 61
 multirate (*see also* multirate filter bank)
 paraunitary, 41, 54–55, 60–62,
 86–87, 93, 206, 208,
 210–211, 217, 219
 perfect reconstruction (*see also* perfect reconstruction filter bank)
 phase distortion, 48, 61, 145, 151
 polyphase, 48, 60–62, 120–123,
 170–172, 206
 realizable and lossless, 45, 73, 82,
 142
 reconstruction error (*see also* reconstruction error)
- single-sideband modulation (*see also* single-sideband modulation)
 synthesis (*see also* synthesis filter bank)
 uniform (*see also* uniform(spaced) filter bank)
 vector form, 47
- Filter-bank-sum, 146, 147, 227
 Filter coefficients, 1, 12, 16, 19, 26, 33,
 34, 36, 54–55, 74, 116, 150, 170,
 173, 257, 284
 Filtered-x LMS algorithm, 19
 Finite-impulse-response (FIR) filter,
 2–10, 16, 18, 26, 29, 38–39, 54,
 68, 85, 147–148, 152, 154,
 168–169, 171, 268–273,
 282–284
 Forgetting factor, 9, 11, 12, 35
 Fractional delay, 120–123
 Frequency-domain adaptive filter
 (FDAF), 31–34, 311
 Frequency sampling method, 116–119,
 234
 Frequency stacking, 116–119
 Fullband filter/filtering, 100–101, 116,
 124, 136, 139, 156, 158, 190
- G**
 Gaussian noise, 8, 16, 21, 22, 92, 136,
 193, 219, 222
 Global minimum, 4, 104
 Gradient-based algorithm, 56–57,
 100–101, 113, 192
 Gradient constraint, 33–34, 311
 Group delay, 227, 265, 269
- H**
 Hands-free, 2, 16
 Hearing aids, 16, 21–22, 30, 156
 Hyperboloid, 4
- I**
 Ideal filter, 48, 82, 144
 Improved proportionate normalized LMS algorithm, 157–158, 160
 Impulse response, 16, 18, 48, 54, 66,
 123, 126, 138, 141, 151, 154,
 179, 183, 204, 228, 287

- antisymmetric, 68
- conjugate pairs, 63, 116
- dispersive, 159, 160
- sparse, 156, 159, 160
- symmetric, 60, 67, 68, 90
- time-reversed, 54
- Independence assumption, 213
- In-band aliasing, 133, 135, 138, 141, 147, 162, 228
- Interference cancellation, 20–21, 232
- Interpolate/interpolation, 43, 145, 292–293, 262
 - factor, 43, 292
 - linear operation, 43
- Inverse matrix, 5
- Inverse modeling (*see also* adaptive inverse modeling)
- Iterative
 - least-squares, 140
 - optimization, 179, 210
 - projection, 179, 212
- L**
 - l_1 -norm, 156
 - Lagrange\Lagrangian
 - function, 173–174
 - multiplier, 173–174
 - vector, 174
 - Least-mean-square (LMS) algorithm, 6–14, 16, 17, 31, 39, 111, 139, 213, 234, 272, 284–292, 301, 308
 - Least-square (LS)
 - error, 11, 39
 - method (approach), 11, 178
 - Linear data model, 204, 208, 210–211, 217, 219
 - Lossless, 73, 82, 113, 206
 - LPC coefficients, 78, 92
- M**
 - Matrix inversion lemma, 12, 180
 - Mean-square deviation, 216–217
 - Mean-square error (MSE), 4–8, 14, 125, 135, 140, 160, 180, 182, 185, 193, 232, 287
 - excess (*see also* excess mean-square error)
 - fullband/subband, 104–106, 108, 113
 - minimum, 4–6, 14, 186, 211, 222, 232
 - multiband, 209–211
 - steady-state (*see also* steady-state mean-square error)
 - weighted, 181, 185
 - Mirror filter (*see also* quadrature mirror filter)
 - Misadjustment, 14, 136, 160, 233
 - Misalignment, 126, 128, 197, 230, 287, 292
 - Minimization, 4, 11, 104, 113, 155
 - Minimum norm, 175, 178
 - Minimum-phase filter, 230
 - MPEG-1 audio coder, 153–154
 - Multiband
 - adaptation, 156, 173
 - delayless, 189–192
 - structured SAF (MSAF), 38, 55, 57, 106, 110–113, 136, 156, 170, 173, 175–182, 186, 189–198, 203, 209, 211–223, 232–234, 315–316
 - Multirate
 - filter bank, 73, 77, 204, 292
 - system, 41, 176, 292
 - N**
 - Narrowband
 - component/signal, 23, 24, 30
 - interference, 23
 - spectrum, 113
 - Newton's method, 179–182, 186
 - Noble identities, 50
 - Nonlinear decision device, 31
 - Nonstationary signal (environment), 8, 11, 18, 35, 126, 198, 233
 - Nonuniform filter bank/subband, 138, 139, 140, 141, 230, 232
 - Norm of weight error, 15, 126
 - Normalized LMS (NLMS) algorithm, 8–9, 10–12, 14, 57, 111, 123, 126, 157, 173, 176, 178, 193–198, 218–220, 272, 308

Normalization matrix, 156, 176, 190, 192, 198, 218
 Normalized step size, 36, 230
 Nyquist frequency, 140

O

Octave-spaced filter bank, 149, 152
 Odd-channel stacking, 141
 Open-loop, 104, 115
 delayless structure, 116, 189–190, 192
 Optimum (weight) vector, 5–6, 12, 14, 173, 211
 Orthogonality
 constraint, 62
 subband, 82, 87, 89, 113, 234
 subband signal, 82
 at zero lag, 63, 85–86, 234
 Orthogonalization matrix, 186, 188
 Overlap-add method, 32, 142
 Overlap-save method, 32,
 Overlapped frequency band, 38, 110, 133, 144–146, 194
 Oversampling/oversampled
 filter banks, 106, 134, 139, 143, 147, 148, 228, 232
 ratio, 138, 140, 148
 subband adaptive filter, 100, 113, 135, 136, 138–143, 147, 230

P

Paraconjugate, 50, 54, 206, 207
 Paraunitary (*see also* filter bank, paraunitary)
 Parks-McClellan algorithm, 63, 259
 Parallel-to-serial converter, 51
 Partial update, 137–138
 Perfect-reconstruction, 54, 59–60, 64, 149, 153, 155, 206
 condition, 155
 filter bank, 47–48, 51, 86, 144, 150, 228
 Performance
 criteria, 1, 4, 140
 surface, 4, 5, 6, 186, 209, 211

Phase, 19, 86, 120, 122, 153, 154, 272
 difference, 80, 120, 153

distortion, 48, 61, 145, 151
 linear, 60, 67, 68, 89, 152, 228
 minimum (see minimum-phase filter)
 preserving, 61
 shift, 24
 zero (see zero-phase response)

Power (*see also* signal power)
 Power complementary, 111, 152, 153, 155, 180–181, 187, 206, 209–211, 234

Power spectrum, 79, 80–83, 93, 101–107, 188, 194–197, 214

Prediction error, 23, 189

Primary path, 19

Principle of minimal disturbance, 173, 192

Projection, 134, 179, 211–213

Propagation delay, 25

Proportionate

 multi-band subband adaptive filter (PMSAF), 156–160

 normalized LMS algorithm, 157

Prototype filter, 45, 59–68, 91–92, 107, 120, 140, 141, 144, 147–155, 228, 232, 295
 real-valued, 63, 66, 90, 101, 107, 116, 138, 141–142
 stopband edge, 92, 144–147

Pseudo-inversion, 10

Pseudo quadrature mirror filter (PQMF), 60–62, 92–93, 153–154, 160, 193, 219, 228

Pseudo-random, 27

Q

Quadratic
 criteria, 136
 function, 4, 173, 179, 210
 Quadrature mirror filter (QMF), 46–48, 50, 55, 149–152, 155
 Quadrature modulation, 142

R

Reconstruction error, 140, 155

Recursive

 algorithm/equation, 2, 6, 9, 173, 175, 178, 181, 187, 282

- least square (RLS) algorithm, 11–13, 35, 139, 271, 309–310
- Reference signal (*see also* desired signal)
- Regularization factor, 135, 180, 188, 194, 196, 204, 211, 216, 219, 234
- Reverberation time, 18
- Room transfer function, 17
- S**
- Sampling rate, 38, 39, 57, 106, 111, 135, 142–143, 149–150, 167, 170
conversion/alteration, 42–45, 55, 292–294
effective, 44, 167
- Second-order statistics, 5, 181
- Secondary path, 19
- Selective coefficient update, 134, 137, 230
- Self-orthogonalizing adaptive filter (SOAF), 35–37, 312–313
- Serial-to-parallel converter, 51
- Settling time, 14
- Shift-variant operation, 43
- Signal power, 8–9, 35, 102, 181
- Single-sideband modulation, 142
- Sparse environment (*see also* impulse response, sparse)
- Spectral dynamic range, 13–14, 82, 100, 101–103, 106, 113, 137, 186, 194–195, 214
- Spectral gap, 106, 147
- Spectral leakage, 142, 150
- Spectrogram, 15, 274
- Spectrum
cross, 79–85, 90–92, 95
energy, 79–85
equalized, 111–113, 186, 188, 194, 214
power, 79–85, 93, 101–102, 188, 194, 214
- Stationary, 4, 8, 73, 126, 136, 168, 196, 210
wide-sense, 73–75, 77, 85, 93, 183, 186–187
- Steady-state mean-square error, 14, 125, 138, 154–155, 160, 193, 196, 220–222
- Steepest-descent method, 6, 181–182, 186
- Step size, 6–8, 14, 18, 36, 103, 105, 115, 126, 136, 156–157, 175, 177, 191, 212, 217, 220–222, 230
- stability bound, 7, 8, 203, 214, 220
- Subband
coding, 41, 99
error (*see also* error, subband)
filter/filtering, 38, 99, 123, 134, 144, 232
orthogonality, 73, 89, 234
regressor, 123, 156, 168, 175–176, 204, 212–213
spectrum, 82–85, 57, 107, 113, 119
to fullband weight transformation, 116–123, 126, 189
- Subband adaptive filter (SAF)/filtering, 38–39, 57, 99, 104, 134, 138, 142, 203, 292–299, 314–315
- closed-loop (*see also* closed-loop)
conventional, 99–104
critically sampled (*see also* critically sampled subband adaptive filtering)
delayless (*see also* delayless subband adaptive filtering)
multiband structured (*see also* multiband structured subband adaptive filtering)
open-loop (*see also* opened-loop)
oversampled (*see also* oversampled subband adaptive filtering)
- Synthesis filter bank, 38–39, 44–47, 54, 57, 104–106, 114, 139, 141, 144–155, 187
- System modeling (*see also* adaptive system modeling)
- T**
- Tap assignment, 231
- Tap weight (*see also* weight vector)
- Tapped-delay line, 35, 49, 115, 171, 172, 191, 282–285

- Time-reversed, 54, 55, 153, 227
 Time-varying system, 2, 27, 47, 136, 282, 285
 Transform-domain adaptive filter (TDAF), 31, 55–57, 137
 Transient, 136, 187, 194
 Transition bands, 48, 82, 91, 142, 155
 Transversal filter (*see also* FIR filter)
- U**
 Underdetermined case, 10, 177–178
 Uniform (spaced) filter bank, 44, 66, 140, 142, 149–153, 230
 Unitary, 55, 88, 137
 Unknown system, 15–16, 20, 25–27, 30–100, 105, 108, 128, 140, 156, 160, 204, 287
 modeling, 110, 111, 122, 125, 140, 173
 Upsample, 38, 262
- V**
 Variable
 step-size, 134, 136–137, 230
 tap length, 230–231
- Variance, 175
 error signal, 218
 noise, 87, 92, 136, 208, 211, 222
 relation, 216, 218
 subband signal, 93, 112, 181, 185, 187, 213
 weight, 8
- W**
 Wavelet transform-domain, 138
 Weight-error vector, 126, 205, 211, 213, 217
 Weighted least-square, 11
 Weight vector, 3–6, 11, 12, 14, 16, 33–34, 36, 56–57, 116, 126, 136–137, 158, 168, 173, 175–178, 187, 203, 209, 211–216, 285, 290
 Weight variance, 8
 Wiener-Hopf equation (solution), 5, 7, 11
 Wide-sense stationary (*see also* stationary, wide-sense)
- Z**
 Zero-phase response, 68, 90