Springer Topics in Signal Processing

Jacob Benesty
Mads G. Christensen
Jesper R. Jensen

Signal Enhancement with Variable Span Linear Filters

Springer

Springer Topics in Signal Processing

Volume 7

Series editors

Jacob Benesty, Montreal, Canada
Walter Kellermann, Erlangen, Germany

Jacob Benesty · Mads G. Christensen
Jesper R. Jensen

Signal Enhancement with Variable Span Linear Filters

Springer

Jacob Benesty
INRS-EMT
University of Quebec
Montreal, Quebec
Canada

Jesper R. Jensen
Aalborg University
Aalborg
Denmark

Mads G. Christensen
Aalborg University
Aalborg
Denmark

ISSN 1866-2609 ISSN 1866-2617 (electronic)
Springer Topics in Signal Processing
ISBN 978-981-287-738-3 ISBN 978-981-287-739-0 (eBook)
DOI 10.1007/978-981-287-739-0

Library of Congress Control Number: 2016931188

Printed on acid-free paper

This Springer imprint is published by SpringerNature
The registered company is Springer Science+Business Media Singapore Pte Ltd.

Abstract

The problem of reducing the influence of additive noise on a desired signal occurs in many applications and systems, including also in speech communication systems like cell phones and Internet telephony where it is commonly referred to as speech enhancement. The problem is a difficult one as it is subject to often contradicting requirements, namely that the desired signal, i.e., the speech signal, should be left unharmed while the noise should, ideally, be removed altogether. In practice, it is thus often necessary to sacrifice the speech integrity to achieve a better reduction of the noise. In the classical Wiener filter, this is done in an implicit way, meaning that there is no direct control of the amount of distortion that is incurred on the speech signal. In the theory of optimal linear filtering (like the Wiener filter), the noise reduction problem is stated as a filter design problem while in subspace methods, the problem is, simply put, seen as a geometric one based on eigenvalue decompositions. In this book, the novel concept of variable span signal enhancement filters is introduced, and it is shown how it can be used for noise reduction in various ways. The variable span filters combine the ideas of optimal linear filters with the ideas of subspace methods, as they involve the joint diagonalization of the correlation matrices of the desired signal and the noise. It is shown how some well-known filter designs, like the minimum distortion, maximum signal-to-noise ratio, Wiener, and tradeoff filters along with new generalization of these, can be obtained using the variable span filter framework. It is then shown how the variable span filters can be applied in various contexts, namely in single-channel STFT-based enhancement, in multichannel enhancement in both the time and STFT domains, and, finally, in time-domain binaural enhancement. In these contexts, the properties of these filters are analyzed in terms of noise reduction capabilities and desired signal distortion, and this analysis is validated and further explored in simulations.

Contents

Chapter 1
Introduction

The present book is concerned with a classical problem in signal processing, namely signal enhancement. The problem can be defined as follows: given an observed, noisy signal find, somehow, an estimate of the desired signal. In many applications, the observed signal can be modeled as the sum of the desired signal and some background noise [1], [2]. This is, for example, the case in speech communication (i.e., mobile telephony, voice over IP, etc.), where the desired signal is (usually) a speech signal and all other signals are considered noise. The noise could be background noise from construction work, passing cars, an air conditioning system, or from the sensors or other electrical parts of the communication system. There are plenty of reasons to try to reduce the influence of such noise. In telephony, for example, the presence of noise might, if it is very loud, have a detrimental impact on the intelligibility of the speech. In hearing aids, its presence may cause listener fatigue after extended periods of exposure. The ability of speech coders to accurately reconstruct the speech signal at the receiving end may be ruined by background noise, as codebooks and coding schemes are frequently designed for clean signals. The result is audible, and often quite annoying, with artifacts in the reconstructed signal. Similarly, the performance of speaker and speech recognition systems may be significantly degraded by the presence of noise, as it is difficult to accommodate the multitude of possible noise characteristics in the underlying statistical models while also trying to model the speech signals.

The fundamental question is then how exactly to address the enhancement problem. Historically, the problem has been addressed as an optimal filtering problem [4], wherein the problem is stated as that of finding a filter that, when applied to the observed signal, provides an optimal (in some sense) or at least a good estimate of the desired signal. This then begs the question what exactly good or optimal means for this problem. Ideally, one would prefer that the desired signal is left unchanged while the noise is removed altogether. However, only in rare, pathological cases of little interest it is possible to realize this. Most often, it turns out that we must settle with reducing the influence of the noise as much as possible. However, it is also often possible to achieve additional reduction of the noise by allowing some distortion of the desired signal. Hence, the trick (or perhaps more fittingly, the art) is to come up with a filter design that achieves some reasonable tradeoff between the noise reduction capabilities and the amount of incurred speech distortion. The classical Wiener filter is an example of such an optimal filter, but the aforementioned tradeoff is implicit and only recently have its properties in terms of noise reduction and speech distortion been understood [3].

1.1 Signal Enhancement from a Signal Subspace Perspective

As we have just discussed, the signal enhancement problem can be cast as a filtering problem. This way of looking at it goes back to the very dawn of modern signal processing, i.e., the work of Wiener [4], where the noise reduction problem was originally also stated as linear filtering problem. The problem is thus to find the coefficients of a filter. A different way of looking at the problem emerged in the 80s with [5]. It spawned from the developments in numerical linear algebra and is based on the ideas of matrix approximation and matrix factorizations (or decompositions) (see, e.g., [6] for an overview of these). The idea is, simply put, that the noise reduction problem can be seen as the problem of finding a low-rank approximation of a matrix containing the desired signal from one constructed from the observations or the corresponding correlation matrix. To facilitate this, matrix factorizations, like the eigenvalue or singular value decomposition (or others, more on this later), can be used. This led naturally to the interpretation of methods based on such principles as projections onto subspaces or operations on the eigenvalues or singular values corresponding to different subspaces. These methods were, hence, dubbed subspace methods. For some nice and comprehensive overviews of subspace methods for signal enhancement, we refer the interested reader to [7] and [8]. Early work on subspace methods was limited to the simple white noise case [9], [10], [11], while later work extended the methods to account for colored noise [12], [13], [14]. Different matrix factorizations and decompositions have also been investigated, including the eigenvalue decomposition, the singular value decomposition, the Karhunen-Loève transform, triangular decompositions, the generalized/quotient singular value decomposition, and so on. These approaches may have different numerical properties and have different computational complexities and memory requirements, but other than that, they are all mathematically equivalent [8], meaning that the same result can be achieved using either one. It is of interest to note that the attempts to explore the similarities and differences between subspace methods and filtering methods, and even trying to unify the two ways of looking at the signal enhancement problem, are few and far between, some notable exceptions being [15] and [8]. Aside from signal enhancement, subspace methods have also found many other uses in modern signal processing, including parameter estimation [16], [17], [18], [19], [20], [21], model order selection [22], [23], [24], reduced-rank processing and low-rank adaptive filtering [25], [26], and array processing [27], [28], [29], [30], and much work has been devoted to fast computations of the involved subspaces [31], [32], [33], [34], [35] to make these methods practical.

1.2 From Subspace Methods to Variable Span Linear Filters

As should now be clear, traditional methods based on linear filtering and subspace methods have been considered by the scientific community as two different ways of realizing signal enhancement. In our recent book [36], we took a first step towards unifying linear filtering and subspace methods in one framework. In this book, we take a great leap forward in achieving this end by introducing and exploring what we term variable span filters. In the variable span filter framework, the signal enhancement problem is still seen as a filter design problem, but it makes use of the ideas of subspace methods in finding the filters. More specifically, the variable span filters are formed from linear combinations of the eigenvectors from the joint diagonalization of the the noise and desired signal correlation matrices. By using the joint diagonalization, the color of the noise is taken into account in a simple and elegant way. In forming the linear span filters, different numbers of the eigenvectors corresponding to the largest eigenvalues can be used, and depending on how the chosen number relates to the rank of the correlation matrix of the desired signal, different kinds of filters are obtained. For example, the maximum signal-to-noise ratio (SNR) filter can be obtained by using only the eigenvector corresponding to the largest eigenvalue. If all the eigenvectors are used, then the Wiener filter can be obtained using the variable span filters. If the the number of eigenvectors used is equal to the rank of the desired signal correlation matrix, then the minimum variance distortionless response (MVDR) filter can be obtained, and it is also possible to achieve the tradeoff filter. More importantly, it is possible to obtain generalizations of these filters with the variable span filters, where a variable number of eigenvectors is used, the result being that it is possible to tradeoff distortion on the desired signal for improved noise reduction. Since many well-known filter designs can be obtained as special cases of the variable span filters, and the relation between the various designs is quite simple in this framework, it is quite easy to compare and analyze the various filters, including finding and bounding their performance in terms of desired signal distortion and output SNR. We firmly believe that these ideas will prove valuable to researchers and practitioners working on signal enhancement.

1.3 Organization of the Work

The present book is organized as described next. In Chapter 2, the signal model, basic assumptions, and the problem formulation are first presented, after which the concept of joint diagonalization of the desired signal and noise correlation matrices is introduced and its properties explored. Based on this, the novel concept of variable span filters is introduced. We then proceed to introduce various performance measures, including the output SNR, the desired signal reduction factor, the desired signal distortion index, and the traditional mean-squared error. After this, various optimal variable span filters are derived, including the minimum distortion, MVDR, maximum SNR, Wiener, and tradeoff filters, including also their variable span generalizations. Lastly, the idea of the indirect approach based on variable span filters is introduced. It is based on a two-stage procedure, where, first, an estimate of the noise is found, using variable span filters, which is then subtracted from the observed signal to obtain an estimate of the desired signal. Several such indirect variable span filters are derived, including minimum residual noise, Wiener, and tradeoff filters. In Chapter 3, the variable span filters, which have so far been vectors, are extended to the matrix case to obtain estimates of desired signal vectors rather than just a single sample. The various filter designs introduced previously are generalized to encompass filtering matrices for both the direct and indirect cases, and the performance measures of the various filtering matrices are also derived. In Chapter 4, the concept of the variable span filters is adopted to single-channel signal enhancement in the short-time Fourier transform (STFT) domain. A notable feature of these variable span filters is that, contrary to most approaches, they take interframe correlation into account by using a filter in each subband rather than simply a gain. We then consider multichannel applications of the variable span filters to time-domain signal enhancement in Chapter 5, wherein spatial information is taken into account, before doing the same in the STFT domain in Chapter 6. Moreover, in the end of Chapters 4–6, the presented filter designs are evaluated on real speech signals in terms of noise reduction and signal distortion measures. The codes for running the evaluations are also attached in these chapters. In Chapter 7, we then consider the case of binaural signal enhancement, where two signals are to be extracted from a sensor array, something that is applicable in, for example, hearing aids or headphones. To do this, we apply the widely linear filtering technique to state and solve the problem in a mathematically convenient ways, again using the concept of the variable span filters. Finally, auxiliary MATLAB functions needed in the aforementioned MATLAB scripts are found in Appendix A.

References

1. J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Noise Reduction in Speech Processing*. Berlin, Germany: Springer-Verlag, 2009.
2. P. Loizou, *Speech Enhancement: Theory and Practice*. Boca Raton, FL: CRC Press, 2007.
3. J. Benesty and J. Chen, *Optimal Time-Domain Noise Reduction Filters–A Theoretical Study*, 1st ed. Springer, 2011, no. VII.
4. N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. M.I.T. Press, 1949.
5. D. W. Tufts and R. Kumaresan, “Singular value decomposition and improved frequency estimation using linear prediction,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 30, pp. 671–675, Aug. 1982.
6. G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed. The John Hopkins University Press, 1996.
7. K. Hermus, P. Wambacq, and H. van Hamme, “A review of signal subspace speech enhancement and its application to noise robust speech recognition,” *EURASIP J. Advances Signal Process.*, vol. 2007(1), p. 15, 2007.
8. P. C. Hansen and S. H. Jensen, “Subspace-based noise reduction for speech signals via diagonal and triangular matrix decompositions: survey and analysis,” *EURASIP J. Advances Signal Process.*, vol. 2007(1), p. 24, 2007.
9. J. A. Cadzow, “Signal enhancement–a composite property mapping algorithm,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 30, pp. 671–675, Jan. 1988.
10. M. Dendrinos, S. Bakamidis, and G. Carayannis, “Speech enhancement from noise: a regenerative approach,” *Speech Commun.*, vol. 10, no. 1, pp. 45–57, 1991.
11. B. De Moor, “The singular value decomposition and long and short spaces of noisy matrices,” *IEEE Trans. Signal Process.*, vol. 41, pp. 2826–2838, Sept. 1993.
12. Y. Ephraim and H. L. Van Trees, “A signal subspace approach for speech enhancement,” *IEEE Trans. Speech Audio Process.*, vol. 3, pp. 251–266, Jul. 1995.
13. S. H. Jensen, P. C. Hansen, S. D. Hansen, and J. A. Sørensen, “Reduction of broadband noise in speech by truncated QSVD,” *IEEE Trans. Speech Audio Process.*, vol. 3, pp. 439–448, Nov. 1995.
14. P. S. K. Hansen, *Signal Subspace Methods for Speech Enhancement*. Ph.D. dissertation, Techn. Univ. Denmark, Lyngby, Denmark, 1997.
15. P. Hansen and S. Jensen, “FIR filter representations of reduced-rank noise reduction,” *IEEE Trans. Signal Process.*, vol. 46, pp. 1737–1741, Jun. 1998.
16. V. F. Pisarenko, “The retrieval of harmonics from a covariance function,” *Geophys. J. Roy. Astron. Soc.*, vol. 33, pp. 347–366, 1973.
17. R. O. Scmidt, “Multiple emitter location and signal parameter estimation,” in *Proc. RADC, Spectral Estimation Workshop*, 1979, pp. 243–258.
18. A. Barabell, “Improving the resolution performance of eigenstructure-based direction-finding algorithms,” in *Proc. IEEE ICASSP*, vol. 8, 1983, pp. 336–339.
19. R. O. Schmidt, “Multiple emitter location and signal parameter estimation,” *IEEE Trans. Antennas Propag.*, vol. 34, pp. 276–280, Mar. 1986.
20. R. Roy and T. Kailath, “ESPRIT–estimation of signal parameters via rotational invariance techniques,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, pp. 984–995, Jul. 1989.
21. M. Haardt and J. A. Nossek, “Unitary ESPRIT: how to obtain increased estimation accuracy with a reduced computational burden,” *IEEE Trans. Signal Process.*, vol. 43, pp. 1232–1242, May 1995.
22. M. Wax and T. Kailath, “Detection of the number of signals by information theoretic criterion,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 33, pp. 387–392, Apr. 1985.

23. R. Badeau, B. David, and G. Richard, “A new perturbation analysis for signal enumeration in rotational invariance techniques,” *IEEE Trans. Signal Process.*, vol. 54, pp. 450–458, Feb. 2006.
24. M. G. Christensen, A. Jakobsson, and S. H. Jensen, “Sinusoidal order estimation using angles between subspaces,” *EURASIP J. Advances Signal Process.*, pp. 1–11, vol. 2009(1), p. 11, 2009.
25. L. L. Scharf, “The SVD and reduced rank signal processing,” *Signal Process.*, vol. 25, pp. 113–133, 1991.
26. P. Strobach, “Low-rank adaptive filters,” *IEEE Trans. Signal Process.*, vol. 44, pp. 2932–2947, Dec. 1996.
27. G. Bienvenu and L. Kopp, “Optimality of high resolution array processing using the eigensystem approach,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, pp. 1235–1248, Oct. 1983.
28. R. Kumaresan and D. W. Tufts, “Estimating the angles of arrival of multiple plane waves,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 19, pp. 134–139, Jan. 1983.
29. H. Krim and M. Viberg, “Two decades of array signal processing research: the parametric approach,” *IEEE Signal Process. Mag.*, vol. 13, pp. 67–94, Jul. 1996.
30. M. Viberg, B. Ottersten, and T. Kailath, “Detection and estimation in sensor arrays using weighted subspace fitting,” *IEEE Trans. Signal Process.*, vol. 39, pp. 2436–2449, Nov. 1991.
31. B. Yang, “Projection approximation subspace tracking,” *IEEE Trans. Signal Process.*, vol. 41, pp. 95–107, Jan. 1995.
32. D. Rabideau, “Fast, rank adaptive subspace tracking and applications,” *IEEE Trans. Signal Process.*, vol. 44, pp. 2229–2244, Sept. 1996.
33. R. Badeau, G. Richard, and B. David, “Sliding window adaptive SVD algorithms,” *IEEE Trans. Signal Process.*, vol. 52, pp. 1–10, Jan. 2004.
34. R. Badeau, B. David, and G. Richard, “Fast approximated power iteration subspace tracking,” *IEEE Trans. Signal Process.*, vol. 53, pp. 2931–2941, Aug. 2005.
35. X. Doukopoulos and G. Moustakides, “Fast and stable subspace tracking,” *IEEE Trans. Signal Process.*, vol. 56, pp. 1452–1465, Apr. 2008.
36. J. Benesty, J. R. Jensen, M. G. Christensen, and J. Chen, *Speech Enhancement–A Signal Subspace Perspective*. New York: Academic Press, 2013.

Chapter 2
General Concept with Filtering Vectors

In this chapter, we explain the principle of variable span (VS) linear filters and show how it can be applied to the signal enhancement problem. Then, we derive a large class of optimal filters with a great deal of flexibility in the sense that they can naturally compromise between noise reduction and desired signal distortion. We end this part by also showing how to estimate the noise signal, from which we can also estimate the desired signal with another class of variable span linear filters.

2.1 Signal Model and Problem Formulation

We consider the very general signal model:

$$\mathbf{y} = \mathbf{x} + \mathbf{v}, \tag{2.1}$$

where $\mathbf{y}$ is the observation or noisy signal vector of length M, $\mathbf{x}$ is the desired signal vector, and $\mathbf{v}$ is the noise signal vector. We assume that the components of the two vectors $\mathbf{x}$ and $\mathbf{v}$ are zero mean, stationary, and circular. We further assume that these two vectors are uncorrelated, i.e., $E\left(\mathbf{x}\mathbf{v}^H\right) = E\left(\mathbf{v}\mathbf{x}^H\right) = \mathbf{0}_{M\times M}$, where $E(\cdot)$ denotes mathematical expectation, the superscript ^{H} is the conjugate-transpose operator, and $\mathbf{0}_{M\times M}$ is a matrix of size $M \times M$ with all its elements equal to 0. In this context, the correlation matrix (of size $M \times M$) of the observations is

$$\begin{aligned} \mathbf{\Phi}_{\mathbf{y}} &= E\left(\mathbf{y}\mathbf{y}^H\right) \\ &= \mathbf{\Phi}_{\mathbf{x}} + \mathbf{\Phi}_{\mathbf{v}}, \end{aligned} \tag{2.2}$$

where $\mathbf{\Phi}_{\mathbf{x}} = E\left(\mathbf{x}\mathbf{x}^H\right)$ and $\mathbf{\Phi}_{\mathbf{v}} = E\left(\mathbf{v}\mathbf{v}^H\right)$ are the correlation matrices of $\mathbf{x}$ and $\mathbf{v}$, respectively. In the rest of this chapter, we assume that the rank of

J. Benesty et al., *Signal Enhancement with Variable Span Linear Filters*,
Springer Topics in Signal Processing 7, DOI 10.1007/978-981-287-739-0_2

the desired signal correlation matrix, $\mathbf{\Phi}_\mathbf{x}$, is equal to $P \leq M$ and the rank of the noise correlation matrix, $\mathbf{\Phi}_\mathbf{v}$, is equal to M.

Let x_1 be the first element of $\mathbf{x}$. It is assumed that x_1 is the desired signal sample. Then, the objective of signal enhancement (or noise reduction) is to estimate x_1 from $\mathbf{y}$. This should be done in such a way that the noise is reduced as much as possible with no or little distortion of the desired signal sample [1], [2], [3], [4].

2.2 Joint Diagonalization

The use of the joint diagonalization in noise reduction was first proposed in [5] and then in [6]. In this work, we give a different perspective as it will be shown later.

The two Hermitian matrices $\mathbf{\Phi}_\mathbf{x}$ and $\mathbf{\Phi}_\mathbf{v}$ can be jointly diagonalized as follows [7]:

$$\mathbf{B}^H \mathbf{\Phi}_\mathbf{x} \mathbf{B} = \mathbf{\Lambda}, \tag{2.3}$$

$$\mathbf{B}^H \mathbf{\Phi}_\mathbf{v} \mathbf{B} = \mathbf{I}_M, \tag{2.4}$$

where $\mathbf{B}$ is a full-rank square matrix (of size $M \times M$), $\mathbf{\Lambda}$ is a diagonal matrix whose main elements are real and nonnegative, and $\mathbf{I}_M$ is the $M \times M$ identity matrix. Furthermore, $\mathbf{\Lambda}$ and $\mathbf{B}$ are the eigenvalue and eigenvector matrices, respectively, of $\mathbf{\Phi}_\mathbf{v}^{-1}\mathbf{\Phi}_\mathbf{x}$, i.e.,

$$\mathbf{\Phi}_\mathbf{v}^{-1}\mathbf{\Phi}_\mathbf{x}\mathbf{B} = \mathbf{B}\mathbf{\Lambda}. \tag{2.5}$$

Since the rank of the matrix $\mathbf{\Phi}_\mathbf{x}$ is equal to P, the eigenvalues of $\mathbf{\Phi}_\mathbf{v}^{-1}\mathbf{\Phi}_\mathbf{x}$ can be ordered as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_P > \lambda_{P+1} = \cdots = \lambda_M = 0$. In other words, the last $M - P$ eigenvalues of the matrix product $\mathbf{\Phi}_\mathbf{v}^{-1}\mathbf{\Phi}_\mathbf{x}$ are exactly zero, while its first P eigenvalues are positive, with λ_1 being the maximum eigenvalue. We also denote by $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_P, \mathbf{b}_{P+1}, \ldots, \mathbf{b}_M$, the corresponding eigenvectors. A consequence of this joint diagonalization is that the noisy signal correlation matrix can also be diagonalized as

$$\mathbf{B}^H \mathbf{\Phi}_\mathbf{y} \mathbf{B} = \mathbf{\Lambda} + \mathbf{I}_M. \tag{2.6}$$

We can decompose the matrix $\mathbf{B}$ as

$$\mathbf{B} = \left[\mathbf{B}_Q' \; \mathbf{B}_Q'' \right], \tag{2.7}$$

where

$$\mathbf{B}_Q' = \left[\mathbf{b}_1 \; \mathbf{b}_2 \; \cdots \; \mathbf{b}_Q \right] \tag{2.8}$$

is an $M \times Q$ matrix,

$$\mathbf{B}''_Q = \begin{bmatrix} \mathbf{b}_{Q+1} \; \mathbf{b}_{Q+2} \; \cdots \; \mathbf{b}_M \end{bmatrix} \tag{2.9}$$

is an $M \times (M - Q)$ matrix, and $1 \leq Q \leq M$. For the particular case $Q = P$, the matrices $\mathbf{B}'_P$ and $\mathbf{B}''_P$ span the desired signal-plus-noise subspace and the noise subspace, respectively. It can be verified from (2.3) and (2.4) that

$$\mathbf{B}''^H_P \mathbf{x} = \mathbf{0}_{(M-P)\times 1} \tag{2.10}$$

and

$$\begin{aligned} \mathbf{\Phi}_{\mathbf{v}}^{-1} &= \mathbf{B}\mathbf{B}^H \\ &= \mathbf{B}'_P \mathbf{B}'^H_P + \mathbf{B}''_P \mathbf{B}''^H_P \\ &= \mathbf{B}'_Q \mathbf{B}'^H_Q + \mathbf{B}''_Q \mathbf{B}''^H_Q . \end{aligned} \tag{2.11}$$

Another convenient way to express (2.3) and (2.4) is

$$\mathbf{B}'^H_Q \mathbf{\Phi}_{\mathbf{x}} \mathbf{B}'_Q = \mathbf{\Lambda}'_Q, \tag{2.12}$$

$$\mathbf{B}'^H_Q \mathbf{\Phi}_{\mathbf{v}} \mathbf{B}'_Q = \mathbf{I}_Q, \tag{2.13}$$

where

$$\mathbf{\Lambda}'_Q = \text{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_Q\right) \tag{2.14}$$

is a diagonal matrix containing the first Q eigenvalues of $\mathbf{\Phi}_{\mathbf{v}}^{-1}\mathbf{\Phi}_{\mathbf{x}}$ and $\mathbf{I}_Q$ is the $Q \times Q$ identity matrix. As a consequence of (2.12) and (2.13), we also have

$$\mathbf{B}'^H_Q \mathbf{\Phi}_{\mathbf{y}} \mathbf{B}'_Q = \mathbf{\Lambda}'_Q + \mathbf{I}_Q. \tag{2.15}$$

The joint diagonalization is a very natural tool to use if we want to fully exploit the desired signal-plus-noise and noise subspaces in noise reduction and fully optimize the linear filtering process.

2.3 Variable Span (VS) Linear Filtering

One of the most convenient ways to estimate the desired signal, x_1, from the observation signal vector, $\mathbf{y}$, is through a filtering operation, i.e.,

$$z = \mathbf{h}^H \mathbf{y}, \tag{2.16}$$

where z is the estimate of x_1 and

$$\mathbf{h} = \begin{bmatrix} h_1 \ h_2 \ \cdots \ h_M \end{bmatrix}^T \tag{2.17}$$

is a complex-valued filter of length M. It is always possible to write $\mathbf{h}$ in a basis formed from the vectors $\mathbf{b}_m,\ m = 1, 2, \ldots, M$, i.e.,

$$\begin{aligned} \mathbf{h} &= \mathbf{B}\mathbf{a} \\ &= \mathbf{B}'_Q\mathbf{a}'_Q + \mathbf{B}''_Q\mathbf{a}''_Q, \end{aligned} \tag{2.18}$$

where the components of

$$\begin{aligned} \mathbf{a} &= \begin{bmatrix} a_1 \ \cdots \ a_Q \ a_{Q+1} \ \cdots \ a_M \end{bmatrix}^T \\ &= \begin{bmatrix} \mathbf{a}'^T_Q \ \mathbf{a}''^T_Q \end{bmatrix}^T \end{aligned} \tag{2.19}$$

are the coordinates of $\mathbf{h}$ in the new basis, and $\mathbf{a}'_Q$ and $\mathbf{a}''_Q$ are vectors of length Q and $M - Q$, respectively. Now, instead of estimating the coefficients of $\mathbf{h}$ as in conventional approaches, we can estimate, equivalently, the coordinates $a_m,\ m = 1, 2, \ldots, M$. When $\mathbf{a}$ is estimated, it is then easy to determine $\mathbf{h}$ from (2.18). Furthermore, for $Q = P$, several optimal noise reduction filters with at most P constraints will lead to $\mathbf{a}''_P = \mathbf{0}_{(M-P)\times 1}$ since there is no desired signal in the directions $\mathbf{B}''_P$. Therefore, we can sometimes simplify our problem and force $\mathbf{a}''_P = \mathbf{0}_{(M-P)\times 1}$; as a result, the filter and the estimate are, respectively, $\mathbf{h} = \mathbf{B}'_P\mathbf{a}'_P$ and $z = \mathbf{a}'^H_P\mathbf{B}'^H_P\mathbf{y}$.

From the previous discussion and from (2.18), we see that we can build a more flexible linear filter. We define our variable span (VS) linear filter of length M as

$$\mathbf{h}(Q) = \mathbf{B}'_Q\mathbf{a}'_Q. \tag{2.20}$$

Obviously, $\mathbf{h}(Q) \in \text{Span}\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_Q\}$. As a consequence, the estimate of x_1 is

$$\begin{aligned} z &= \mathbf{a}'^H_Q\mathbf{B}'^H_Q\mathbf{x} + \mathbf{a}'^H_Q\mathbf{B}'^H_Q\mathbf{v} \\ &= x_{\text{fd}} + v_{\text{rn}}, \end{aligned} \tag{2.21}$$

where

$$x_{\text{fd}} = \mathbf{a}'^H_Q\mathbf{B}'^H_Q\mathbf{x} \tag{2.22}$$

is the filtered desired signal and

$$v_{\text{rn}} = \mathbf{a}'^H_Q\mathbf{B}'^H_Q\mathbf{v} \tag{2.23}$$

is the residual noise. We deduce that the variance of z is

$$\begin{aligned}\phi_z &= E\left(|z|^2\right) \\ &= \mathbf{a}_Q'^H \mathbf{\Lambda}_Q' \mathbf{a}_Q' + \mathbf{a}_Q'^H \mathbf{a}_Q'.\end{aligned} \tag{2.24}$$

Notice that the proposed linear processing implies implicitly that we force the last $M - Q$ components of $\mathbf{a}$ to 0.

2.4 Performance Measures

In this section, we briefly define the most useful performance measures for noise reduction with VS linear filters. We can divide these measures into two categories. The first category evaluates the noise reduction performance while the second one evaluates the distortion of the desired signal. We also discuss the very convenient mean-squared error (MSE) criterion, which we tailor for variable span filters, and show how it is related to the performance measures.

2.4.1 Noise Reduction

One of the most fundamental measures in all aspects of signal enhancement is the signal-to-noise ratio (SNR). Since x_1 is the desired signal, we define the input SNR as

$$\text{iSNR} = \frac{\phi_{x_1}}{\phi_{v_1}}, \tag{2.25}$$

where $\phi_{x_1} = E\left(|x_1|^2\right)$ is the variance of x_1 and $\phi_{v_1} = E\left(|v_1|^2\right)$ is the variance of the first component of $\mathbf{v}$, i.e., v_1.

From (2.24), it is easy to find that the output SNR is

$$\begin{aligned}\text{oSNR}\left(\mathbf{a}_Q'\right) &= \frac{\mathbf{a}_Q'^H \mathbf{\Lambda}_Q' \mathbf{a}_Q'}{\mathbf{a}_Q'^H \mathbf{a}_Q'} \\ &= \frac{\sum_{q=1}^{Q} \lambda_q |a_q|^2}{\sum_{q=1}^{Q} |a_q|^2}\end{aligned} \tag{2.26}$$

and it can be shown that

$$\text{oSNR}\left(\mathbf{a}_Q'\right) \leq \lambda_1, \tag{2.27}$$

which means that the output SNR can never exceed the maximum eigenvalue, λ_1. The filters should be derived in such a way that $\text{oSNR}\left(\mathbf{a}_Q'\right) \geq \text{iSNR}$.

The noise reduction factor, which quantifies the amount of noise whose is rejected by the complex filter, is given by

$$\xi_{\text{nr}}\left(\mathbf{a}_Q'\right) = \frac{\phi_{v_1}}{\mathbf{a}_Q'^H \mathbf{a}_Q'}. \tag{2.28}$$

For optimal filters, we should have $\xi_{\text{nr}}\left(\mathbf{a}_Q'\right) \geq 1$.

2.4.2 Desired Signal Distortion

In practice, the complex filter may distort the desired signal. In order to evaluate the level of this distortion, we define the desired signal reduction factor:

$$\xi_{\text{sr}}\left(\mathbf{a}_Q'\right) = \frac{\phi_{x_1}}{\mathbf{a}_Q'^H \mathbf{\Lambda}_Q' \mathbf{a}_Q'}. \tag{2.29}$$

For optimal filters, we should have $\xi_{\text{sr}}\left(\mathbf{a}_Q'\right) \geq 1$. The larger is the value of $\xi_{\text{sr}}\left(\mathbf{a}_Q'\right)$, the more the desired signal is distorted.

By making the appropriate substitutions, one can derive the relationship:

$$\frac{\text{oSNR}\left(\mathbf{a}_Q'\right)}{\text{iSNR}} = \frac{\xi_{\text{nr}}\left(\mathbf{a}_Q'\right)}{\xi_{\text{sr}}\left(\mathbf{a}_Q'\right)}. \tag{2.30}$$

This expression indicates the equivalence between gain/loss in SNR and distortion (for both desired signal and noise).

Another way to measure the distortion of the desired signal due to the complex filter is the desired signal distortion index, which is defined as the mean-squared error between the desired signal and the filtered desired signal, normalized by the variance of the desired signal, i.e.,

$$\upsilon_{\text{sd}}\left(\mathbf{a}_Q'\right) = \frac{E\left(\left|x_1 - \mathbf{a}_Q'^H \mathbf{B}_Q'^H \mathbf{x}\right|^2\right)}{\phi_{x_1}}. \tag{2.31}$$

The desired signal distortion index is usually upper bounded by 1 for optimal filters.

2.4.3 Mean-Squared Error (MSE) Criterion

The error signal between the estimated and desired signals is

$$\begin{aligned} e &= z - x_1 \\ &= \mathbf{a}_Q'^H \mathbf{B}_Q'^H \mathbf{y} - x_1, \end{aligned} \tag{2.32}$$

which can also be written as the sum of two uncorrelated error signals:

$$e = e_{\mathrm{ds}} + e_{\mathrm{rs}}, \tag{2.33}$$

where

$$e_{\mathrm{ds}} = \mathbf{a}_Q'^H \mathbf{B}_Q'^H \mathbf{x} - x_1 \tag{2.34}$$

is the distortion of the desired signal due to the filter and

$$e_{\mathrm{rs}} = \mathbf{a}_Q'^H \mathbf{B}_Q'^H \mathbf{v} \tag{2.35}$$

represents the residual noise. The mean-squared error (MSE) criterion is then

$$\begin{aligned} J\left(\mathbf{a}_Q'\right) &= E\left(|e|^2\right) \\ &= \phi_{x_1} - \mathbf{i}^T \mathbf{\Phi}_{\mathbf{x}} \mathbf{B}_Q' \mathbf{a}_Q' - \mathbf{a}_Q'^H \mathbf{B}_Q'^H \mathbf{\Phi}_{\mathbf{x}} \mathbf{i} + \mathbf{a}_Q'^H \left(\mathbf{\Lambda}_Q' + \mathbf{I}_Q\right) \mathbf{a}_Q' \\ &= J_{\mathrm{ds}}\left(\mathbf{a}_Q'\right) + J_{\mathrm{rs}}\left(\mathbf{a}_Q'\right), \end{aligned} \tag{2.36}$$

where $\mathbf{i}$ is the first column of $\mathbf{I}_M$,

$$\begin{aligned} J_{\mathrm{ds}}\left(\mathbf{a}_Q'\right) &= E\left(|e_{\mathrm{ds}}|^2\right) \\ &= \phi_{x_1} - \mathbf{i}^T \mathbf{\Phi}_{\mathbf{x}} \mathbf{B}_Q' \mathbf{a}_Q' - \mathbf{a}_Q'^H \mathbf{B}_Q'^H \mathbf{\Phi}_{\mathbf{x}} \mathbf{i} + \mathbf{a}_Q'^H \mathbf{\Lambda}_Q' \mathbf{a}_Q' \\ &= \upsilon_{\mathrm{sd}}\left(\mathbf{a}_Q'\right) \phi_{x_1}, \end{aligned} \tag{2.37}$$

and

$$\begin{aligned} J_{\mathrm{rs}}\left(\mathbf{a}_Q'\right) &= E\left(|e_{\mathrm{rs}}|^2\right) \\ &= \mathbf{a}_Q'^H \mathbf{a}_Q' \\ &= \frac{\phi_{v_1}}{\xi_{\mathrm{nr}}\left(\mathbf{a}_Q'\right)}. \end{aligned} \tag{2.38}$$

We deduce that

$$\frac{J_{\text{ds}}\left(\mathbf{a}'_Q\right)}{J_{\text{rs}}\left(\mathbf{a}'_Q\right)} = \text{iSNR} \times \xi_{\text{nr}}\left(\mathbf{a}'_Q\right) \times \upsilon_{\text{sd}}\left(\mathbf{a}'_Q\right) \tag{2.39}$$
$$= \text{oSNR}\left(\mathbf{a}'_Q\right) \times \xi_{\text{sr}}\left(\mathbf{a}'_Q\right) \times \upsilon_{\text{sd}}\left(\mathbf{a}'_Q\right).$$

This shows how the different performances measures are related to the MSEs.

2.5 Optimal VS Linear Filters

In this section, we derive a large class of VS linear filters for noise reduction from the different MSEs developed in the previous section. We will see how all these filters, with different objectives, are strongly connected.

2.5.1 VS Minimum Distortion

The VS minimum distortion filter is obtained by minimizing the distortion-based MSE, $J_{\text{ds}}\left(\mathbf{a}'_Q\right)$. We get

$$\mathbf{a}'_{Q,\text{MD}} = \mathbf{\Lambda}'^{-1}_Q \mathbf{B}'^H_Q \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}, \tag{2.40}$$

where it is assumed that $Q \leq P$. Therefore, the VS minimum distortion filter is

$$\begin{aligned}\mathbf{h}_{\text{MD}}(Q) &= \mathbf{B}'_Q \mathbf{a}'_{Q,\text{MD}} \\ &= \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\lambda_q} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i},\ Q \leq P.\end{aligned} \tag{2.41}$$

One important particular case of (2.41) is $Q = P$. In this situation, we obtain the celebrated minimum variance distortionless response (MVDR) filter:

$$\begin{aligned}\mathbf{h}_{\text{MVDR}} &= \mathbf{h}_{\text{MD}}(P) \\ &= \mathbf{B}'_P \mathbf{a}'_{P,\text{MD}} \\ &= \sum_{p=1}^{P} \frac{\mathbf{b}_p \mathbf{b}_p^H}{\lambda_p} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i} \\ &= \sum_{p=1}^{P} \mathbf{b}_p \mathbf{b}_p^H \mathbf{\Phi}_{\mathbf{v}} \mathbf{i}.\end{aligned} \tag{2.42}$$

Let us show why (2.42) corresponds to the MVDR filter. With $\mathbf{h}_{\text{MVDR}}$, the filtered desired signal is

$$\begin{aligned} x_{\text{fd}} &= \left(\mathbf{B}'_P \mathbf{B}'^H_P \mathbf{\Phi}_{\mathbf{v}} \mathbf{i}\right)^H \mathbf{x} \\ &= \left(\mathbf{i} - \mathbf{B}''_P \mathbf{B}''^H_P \mathbf{\Phi}_{\mathbf{v}} \mathbf{i}\right)^H \mathbf{x} \\ &= x_1 - \mathbf{i}^T \mathbf{\Phi}_{\mathbf{v}} \mathbf{B}''_P \mathbf{B}''^H_P \mathbf{x} \\ &= x_1, \end{aligned} \tag{2.43}$$

where we have used (2.10) and (2.11) in the previous expression. Then, it is clear that

$$\upsilon_{\text{sd}}\left(\mathbf{a}'_{P,\text{MD}}\right) = 0, \tag{2.44}$$

proving that, indeed, $\mathbf{h}_{\text{MVDR}}$ is the MVDR filter.

Another interesting case of (2.41) is $Q = 1$. In this scenario, we obtain the maximum SNR filter:

$$\begin{aligned} \mathbf{h}_{\max,0} &= \mathbf{h}_{\text{MD}}(1) \\ &= \mathbf{b}_1 a_{1,\text{MD}} \\ &= \frac{\mathbf{b}_1 \mathbf{b}_1^H}{\lambda_1} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}. \end{aligned} \tag{2.45}$$

Indeed, it can be verified that

$$\text{oSNR}\left(a_{1,\text{MD}}\right) = \lambda_1. \tag{2.46}$$

We should always have

$$\text{oSNR}\left(\mathbf{a}'_{P,\text{MD}}\right) \leq \text{oSNR}\left(\mathbf{a}'_{P-1,\text{MD}}\right) \leq \cdots \leq \text{oSNR}\left(a_{1,\text{MD}}\right) = \lambda_1 \tag{2.47}$$

and

$$\upsilon_{\text{sd}}\left(\mathbf{a}'_{P,\text{MD}}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{a}'_{P-1,\text{MD}}\right) \leq \cdots \leq \upsilon_{\text{sd}}\left(a_{1,\text{MD}}\right) \leq 1. \tag{2.48}$$

If $\mathbf{\Phi}_{\mathbf{x}}$ is a full-rank matrix, i.e., $P = M$, then

$$\mathbf{h}_{\text{MD}}(M) = \mathbf{i}, \tag{2.49}$$

which is the identity filter. Assume that the rank of $\mathbf{\Phi}_{\mathbf{x}}$ is $P = 1$. In this case, $Q = 1$, the filter is $\mathbf{h}_{\text{MD}}(1) = \mathbf{h}_{\text{MVDR}}$, and the output SNR is maximized, i.e., equal to λ_1. Also, we can write the desired signal correlation matrix as

$$\mathbf{\Phi}_{\mathbf{x}} = \phi_{x_1} \mathbf{d}\mathbf{d}^H, \tag{2.50}$$

where $\mathbf{d}$ is a vector of length M, whose first element is equal to 1. As a consequence,

$$\begin{aligned}\lambda_1 &= \phi_{x_1} \mathbf{d}^H \mathbf{\Phi}_{\mathbf{v}}^{-1} \mathbf{d} \\ &= \phi_{x_1} \left| \mathbf{d}^H \mathbf{b}_1 \right|^2\end{aligned} \tag{2.51}$$

and

$$\mathbf{h}_{\mathrm{MVDR}} = \frac{\mathbf{\Phi}_{\mathbf{v}}^{-1} \mathbf{d}}{\mathbf{d}^H \mathbf{\Phi}_{\mathbf{v}}^{-1} \mathbf{d}}. \tag{2.52}$$

2.5.2 VS Wiener

The VS Wiener filter is obtained from the optimization of the MSE criterion, $J\left(\mathbf{a}_Q'\right)$. The minimization of $J\left(\mathbf{a}_Q'\right)$ leads to

$$\mathbf{a}_{Q,\mathrm{W}}' = \left(\mathbf{\Lambda}_Q' + \mathbf{I}_Q\right)^{-1} \mathbf{B}_Q'^H \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}, \tag{2.53}$$

where $Q \leq M$. We deduce that the VS Wiener filter is

$$\begin{aligned}\mathbf{h}_{\mathrm{W}}(Q) &= \mathbf{B}_Q' \mathbf{a}_{Q,\mathrm{W}}' \\ &= \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{1 + \lambda_q} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}.\end{aligned} \tag{2.54}$$

It is interesting to compare $\mathbf{h}_{\mathrm{W}}(Q)$ to $\mathbf{h}_{\mathrm{MD}}(Q)$. The two VS filters are very close to each other; they differ by the weighting function, which strongly depends on the eigenvalues of the joint diagonalization. For the VS Wiener filter, this function is equal to $(1 + \lambda_q)^{-1}$ while it is equal to λ_q^{-1} for the VS minimum distortion filter. Also, in the latter filter, Q must be smaller than or equal to P, while Q can be greater than P in the former one.

One important particular case of (2.54) is $Q = M$. In this situation, we obtain the classical Wiener filter:

$$\begin{aligned}\mathbf{h}_{\mathrm{W}} &= \mathbf{h}_{\mathrm{W}}(M) \\ &= \mathbf{B}_M' \mathbf{a}_{M,\mathrm{W}}' \\ &= \sum_{m=1}^{M} \frac{\mathbf{b}_m \mathbf{b}_m^H}{1 + \lambda_m} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i} \\ &= \mathbf{\Phi}_{\mathbf{y}}^{-1} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}.\end{aligned} \tag{2.55}$$

For $Q = 1$, we obtain another form of the maximum SNR filter:

$$\begin{aligned}\mathbf{h}_{\max,1} &= \mathbf{h}_{\mathrm{W}}(1) \\ &= \mathbf{b}_1 a_{1,\mathrm{W}} \\ &= \frac{\mathbf{b}_1\mathbf{b}_1^H}{1+\lambda_1}\boldsymbol{\Phi}_{\mathbf{x}}\mathbf{i},\end{aligned} \tag{2.56}$$

since

$$\mathrm{oSNR}\left(a_{1,\mathrm{W}}\right) = \lambda_1. \tag{2.57}$$

We should always have

$$\mathrm{oSNR}\left(\mathbf{a}'_{M,\mathrm{W}}\right) \le \mathrm{oSNR}\left(\mathbf{a}'_{M-1,\mathrm{W}}\right) \le \cdots \le \mathrm{oSNR}\left(a_{1,\mathrm{W}}\right) = \lambda_1 \tag{2.58}$$

and

$$\upsilon_{\mathrm{sd}}\left(\mathbf{a}'_{M,\mathrm{W}}\right) \le \upsilon_{\mathrm{sd}}\left(\mathbf{a}'_{M-1,\mathrm{W}}\right) \le \cdots \le \upsilon_{\mathrm{sd}}\left(a_{1,\mathrm{W}}\right) \le 1. \tag{2.59}$$

2.5.3 VS Tradeoff

Another interesting approach that can compromise between noise reduction and desired signal distortion is the VS tradeoff filter obtained by

$$\min_{\mathbf{a}'_Q} J_{\mathrm{ds}}\left(\mathbf{a}'_Q\right) \quad \text{subject to} \quad J_{\mathrm{rs}}\left(\mathbf{a}'_Q\right) = \beta\phi_{v_1}, \tag{2.60}$$

where $0 \le \beta \le 1$, to ensure that filtering achieves some degree of noise reduction. We easily find that the optimal filter is

$$\mathbf{h}_{\mathrm{T},\mu}(Q) = \sum_{q=1}^{Q} \frac{\mathbf{b}_q\mathbf{b}_q^H}{\mu+\lambda_q}\boldsymbol{\Phi}_{\mathbf{x}}\mathbf{i}, \tag{2.61}$$

where $\mu \ge 0$ is a Lagrange multiplier[1]. Clearly, for $\mu = 0$ and $\mu = 1$, we get the VS minimum distortion and VS Wiener filters, respectively.

For $Q = M$, we obtain the classical tradeoff filter:

$$\begin{aligned}\mathbf{h}_{\mathrm{T},\mu} &= \mathbf{h}_{\mathrm{T},\mu}(M) \\ &= \sum_{m=1}^{M} \frac{\mathbf{b}_m\mathbf{b}_m^H}{\mu+\lambda_m}\boldsymbol{\Phi}_{\mathbf{x}}\mathbf{i} \\ &= \left(\boldsymbol{\Phi}_{\mathbf{x}} + \mu\boldsymbol{\Phi}_{\mathbf{v}}\right)^{-1}\boldsymbol{\Phi}_{\mathbf{x}}\mathbf{i}\end{aligned} \tag{2.62}$$

and for $Q = 1$, we obtain the maximum SNR filter:

[1] For $\mu = 0$, Q must be smaller than or equal to P.

Table 2.1 Optimal VS linear filters for signal enhancement.

VS MD:	$\mathbf{h}_{\mathrm{MD}}(Q) = \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\lambda_q} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i},\ Q \leq P$
MVDR:	$\mathbf{h}_{\mathrm{MVDR}} = \sum_{p=1}^{P} \frac{\mathbf{b}_p \mathbf{b}_p^H}{\lambda_p} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}$
VS Wiener:	$\mathbf{h}_{\mathrm{W}}(Q) = \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{1+\lambda_q} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i},\ Q \leq M$
Wiener:	$\mathbf{h}_{\mathrm{W}} = \sum_{m=1}^{M} \frac{\mathbf{b}_m \mathbf{b}_m^H}{1+\lambda_m} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i} = \mathbf{\Phi}_{\mathbf{y}}^{-1} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}$
VS Tradeoff:	$\mathbf{h}_{\mathrm{T},\mu}(Q) = \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\mu+\lambda_q} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i},\ \mu \geq 0$
Tradeoff:	$\mathbf{h}_{\mathrm{T},\mu} = \sum_{m=1}^{M} \frac{\mathbf{b}_m \mathbf{b}_m^H}{\mu+\lambda_m} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i} = \left(\mathbf{\Phi}_{\mathbf{x}} + \mu \mathbf{\Phi}_{\mathbf{v}}\right)^{-1} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}$
Maximum SNR:	$\mathbf{h}_{\max,\mu} = \frac{\mathbf{b}_1 \mathbf{b}_1^H}{\mu+\lambda_1} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}$

$$\begin{aligned} \mathbf{h}_{\max,\mu} &= \mathbf{h}_{\mathrm{T},\mu}(1) \\ &= \frac{\mathbf{b}_1 \mathbf{b}_1^H}{\mu+\lambda_1} \mathbf{\Phi}_{\mathbf{x}} \mathbf{i}. \end{aligned} \tag{2.63}$$

In Table 2.1, we summarize all optimal VS filters developed in this section, showing how they are strongly related.

2.6 Indirect Optimal VS Linear Filters

The indirect approach is based on two successive stages. In the first stage, we find an estimate of the noise signal. This estimate is then used in the second stage by subtracting it from the observation signal. This will lead to an estimate of the desired signal.

2.6.1 Indirect Approach

Let

$$\mathbf{h}' = \left[h_1' \ h_2' \ \cdots \ h_M' \right]^T \tag{2.64}$$

be a complex-valued filter of length M. By applying this filter to the observation signal vector, we obtain

$$\widehat{v} = \mathbf{h}'^H\mathbf{x} + \mathbf{h}'^H\mathbf{v} \tag{2.65}$$

and the corresponding output SNR is

$$\text{oSNR}_{\widehat{v}}\left(\mathbf{h}'\right) = \frac{\mathbf{h}'^H\mathbf{\Phi}_\mathbf{x}\mathbf{h}'}{\mathbf{h}'^H\mathbf{\Phi}_\mathbf{v}\mathbf{h}'}. \tag{2.66}$$

Then, we find $\mathbf{h}'$ that minimizes $\text{oSNR}_{\widehat{v}}\left(\mathbf{h}'\right)$. It is easy to check that the solution is

$$\mathbf{h}' = \mathbf{B}''_P\mathbf{a}''_P, \tag{2.67}$$

where $\mathbf{B}''_P$ and $\mathbf{a}''_P$ are defined in the previous sections. With (2.67), $\text{oSNR}_{\widehat{v}}\left(\mathbf{h}'\right) = 0$; therefore, $\widehat{v}$ can be seen as the estimate of the noise.

We consider the more general scenario:

$$\mathbf{h}'(\mathtt{Q}) = \mathbf{B}''_\mathtt{Q}\mathbf{a}''_\mathtt{Q}, \tag{2.68}$$

where $0 \leq \mathtt{Q} < M$,

$$\mathbf{B}''_\mathtt{Q} = \left[\, \mathbf{b}_{\mathtt{Q}+1}\ \mathbf{b}_{\mathtt{Q}+2}\ \cdots\ \mathbf{b}_M \,\right] \tag{2.69}$$

is a matrix of size $M \times (M - \mathtt{Q})$,

$$\mathbf{a}''_\mathtt{Q} = \left[\, a_{\mathtt{Q}+1}\ a_{\mathtt{Q}+2}\ \cdots\ a_M \,\right]^T \tag{2.70}$$

is a vector of length $M - \mathtt{Q}$, and $\mathbf{h}'(\mathtt{Q}) \in \text{Span}\left\{\mathbf{b}_{\mathtt{Q}+1}, \mathbf{b}_{\mathtt{Q}+2}, \ldots, \mathbf{b}_M\right\}$. As a consequence,

$$\widehat{v} = \mathbf{h}'^H(\mathtt{Q})\mathbf{x} + \mathbf{h}'^H(\mathtt{Q})\mathbf{v}. \tag{2.71}$$

Now, in general, $\text{oSNR}_{\widehat{v}}\left[\mathbf{h}'(\mathtt{Q})\right] \neq 0$, and this implies distortion as it will become clearer soon. This concludes the first stage.

In the second stage, we estimate the desired signal, x_1, as the difference between the observation, y_1, and the estimate of the noise obtained from the first stage, i.e.,

$$\begin{aligned} z' &= y_1 - \mathbf{h}'^H(\mathtt{Q})\mathbf{x} - \mathbf{h}'^H(\mathtt{Q})\mathbf{v} \\ &= x_1 - \mathbf{a}''^H_\mathtt{Q}\mathbf{B}''^H_\mathtt{Q}\mathbf{x} + v_1 - \mathbf{a}''^H_\mathtt{Q}\mathbf{B}''^H_\mathtt{Q}\mathbf{v} \\ &= \overline{\mathbf{h}}'^H(\mathtt{Q})\mathbf{y}, \end{aligned} \tag{2.72}$$

where

$$\overline{\mathbf{h}}'(\mathtt{Q}) = \mathbf{i} - \mathbf{B}''_\mathtt{Q}\mathbf{a}''_\mathtt{Q} \tag{2.73}$$

is the equivalent filter applied to the observation signal vector. We deduce that the variance of z' is

$$\begin{aligned}\phi_{z'} &= E\left(|z'|^2\right) \\ &= \phi_{x_1} - \mathbf{i}^T \mathbf{\Phi_x} \mathbf{B}''_{\mathrm{Q}} \mathbf{a}''_{\mathrm{Q}} - \mathbf{a}''^H_{\mathrm{Q}} \mathbf{B}''^H_{\mathrm{Q}} \mathbf{\Phi_x} \mathbf{i} + \mathbf{a}''^H_{\mathrm{Q}} \mathbf{\Lambda}''_{\mathrm{Q}} \mathbf{a}''_{\mathrm{Q}} \\ &\quad + \phi_{v_1} - \mathbf{i}^T \mathbf{\Phi_v} \mathbf{B}''_{\mathrm{Q}} \mathbf{a}''_{\mathrm{Q}} - \mathbf{a}''^H_{\mathrm{Q}} \mathbf{B}''^H_{\mathrm{Q}} \mathbf{\Phi_v} \mathbf{i} + \mathbf{a}''^H_{\mathrm{Q}} \mathbf{a}''_{\mathrm{Q}}.\end{aligned} \tag{2.74}$$

2.6.2 MSE Criterion and Performance Measures

We define the error signal between the estimated and desired signals as

$$\begin{aligned}e' &= z' - x_1 \\ &= -\mathbf{a}''^H_{\mathrm{Q}} \mathbf{B}''^H_{\mathrm{Q}} \mathbf{x} + v_1 - \mathbf{a}''^H_{\mathrm{Q}} \mathbf{B}''^H_{\mathrm{Q}} \mathbf{v}.\end{aligned} \tag{2.75}$$

This error can be written as the sum of two uncorrelated error signals, i.e.,

$$e' = e'_{\mathrm{ds}} + e'_{\mathrm{rs}}, \tag{2.76}$$

where

$$e'_{\mathrm{ds}} = -\mathbf{a}''^H_{\mathrm{Q}} \mathbf{B}''^H_{\mathrm{Q}} \mathbf{x} \tag{2.77}$$

is the distortion of the desired signal due to the filtering operation and

$$e'_{\mathrm{rs}} = v_1 - \mathbf{a}''^H_{\mathrm{Q}} \mathbf{B}''^H_{\mathrm{Q}} \mathbf{v} \tag{2.78}$$

is the residual noise. Then, the MSE criterion is

$$\begin{aligned}J\left(\mathbf{a}''_{\mathrm{Q}}\right) &= E\left(|e'|^2\right) \\ &= \phi_{v_1} - \mathbf{i}^T \mathbf{\Phi_v} \mathbf{B}''_{\mathrm{Q}} \mathbf{a}''_{\mathrm{Q}} - \mathbf{a}''^H_{\mathrm{Q}} \mathbf{B}''^H_{\mathrm{Q}} \mathbf{\Phi_v} \mathbf{i} + \mathbf{a}''^H_{\mathrm{Q}} \left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right) \mathbf{a}''_{\mathrm{Q}} \\ &= J_{\mathrm{ds}}\left(\mathbf{a}''_{\mathrm{Q}}\right) + J_{\mathrm{rs}}\left(\mathbf{a}''_{\mathrm{Q}}\right),\end{aligned} \tag{2.79}$$

where $\mathbf{I}_{M-\mathrm{Q}}$ is the $(M-\mathrm{Q}) \times (M-\mathrm{Q})$ identity matrix,

$$\mathbf{\Lambda}''_{\mathrm{Q}} = \mathrm{diag}\left(\lambda_{\mathrm{Q}+1}, \lambda_{\mathrm{Q}+2}, \ldots, \lambda_M\right) \tag{2.80}$$

is a diagonal matrix containing the last $M - \mathrm{Q}$ eigenvalues of $\mathbf{\Phi}_{\mathbf{v}}^{-1}\mathbf{\Phi_x}$,

$$\begin{aligned}J_{\mathrm{ds}}\left(\mathbf{a}''_{\mathrm{Q}}\right) &= E\left(|e'_{\mathrm{ds}}|^2\right) \\ &= \mathbf{a}''^H_{\mathrm{Q}} \mathbf{\Lambda}''_{\mathrm{Q}} \mathbf{a}''_{\mathrm{Q}} \\ &= \upsilon_{\mathrm{sd}}\left(\mathbf{a}''_{\mathrm{Q}}\right) \phi_{x_1}\end{aligned} \tag{2.81}$$

is the distortion-based MSE,

$$\upsilon_{\mathrm{sd}}\left(\mathbf{a}''_{\mathrm{Q}}\right)=\frac{\mathbf{a}''^H_{\mathrm{Q}}\mathbf{\Lambda}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}}{\phi_{x_1}} \tag{2.82}$$

is the desired signal distortion index,

$$\begin{aligned} J_{\mathrm{rs}}\left(\mathbf{a}''_{\mathrm{Q}}\right) &= E\left(\left|e'_{\mathrm{rs}}\right|^2\right) \\ &= \phi_{v_1}-\mathbf{i}^T\mathbf{\Phi}_{\mathbf{v}}\mathbf{B}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}-\mathbf{a}''^H_{\mathrm{Q}}\mathbf{B}''^H_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}+\mathbf{a}''^H_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}} \\ &= \frac{\phi_{v_1}}{\xi_{\mathrm{nr}}\left(\mathbf{a}''_{\mathrm{Q}}\right)} \end{aligned} \tag{2.83}$$

is the MSE corresponding to the residual noise, and

$$\xi_{\mathrm{nr}}\left(\mathbf{a}''_{\mathrm{Q}}\right)=\frac{\phi_{v_1}}{\phi_{v_1}-\mathbf{i}^T\mathbf{\Phi}_{\mathbf{v}}\mathbf{B}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}-\mathbf{a}''^H_{\mathrm{Q}}\mathbf{B}''^H_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}+\mathbf{a}''^H_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}} \tag{2.84}$$

is the noise reduction factor. We deduce that

$$\begin{aligned} \frac{J_{\mathrm{ds}}\left(\mathbf{a}''_{\mathrm{Q}}\right)}{J_{\mathrm{rs}}\left(\mathbf{a}''_{\mathrm{Q}}\right)} &= \mathrm{iSNR}\times\xi_{\mathrm{nr}}\left(\mathbf{a}''_{\mathrm{Q}}\right)\times\upsilon_{\mathrm{sd}}\left(\mathbf{a}''_{\mathrm{Q}}\right) \\ &= \mathrm{oSNR}\left(\mathbf{a}''_{\mathrm{Q}}\right)\times\xi_{\mathrm{sr}}\left(\mathbf{a}''_{\mathrm{Q}}\right)\times\upsilon_{\mathrm{sd}}\left(\mathbf{a}''_{\mathrm{Q}}\right), \end{aligned} \tag{2.85}$$

where

$$\mathrm{oSNR}\left(\mathbf{a}''_{\mathrm{Q}}\right)=\frac{\phi_{x_1}-\mathbf{i}^T\mathbf{\Phi}_{\mathbf{x}}\mathbf{B}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}-\mathbf{a}''^H_{\mathrm{Q}}\mathbf{B}''^H_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i}+\mathbf{a}''^H_{\mathrm{Q}}\mathbf{\Lambda}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}}{\phi_{v_1}-\mathbf{i}^T\mathbf{\Phi}_{\mathbf{v}}\mathbf{B}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}-\mathbf{a}''^H_{\mathrm{Q}}\mathbf{B}''^H_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}+\mathbf{a}''^H_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}} \tag{2.86}$$

is the output SNR and

$$\xi_{\mathrm{sr}}\left(\mathbf{a}''_{\mathrm{Q}}\right)=\frac{\phi_{x_1}}{\phi_{x_1}-\mathbf{i}^T\mathbf{\Phi}_{\mathbf{x}}\mathbf{B}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}-\mathbf{a}''^H_{\mathrm{Q}}\mathbf{B}''^H_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i}+\mathbf{a}''^H_{\mathrm{Q}}\mathbf{\Lambda}''_{\mathrm{Q}}\mathbf{a}''_{\mathrm{Q}}} \tag{2.87}$$

is the desired signal reduction factor.

2.6.3 Optimal Filters

2.6.3.1 Indirect VS Minimum Residual Noise

The indirect VS minimum residual noise filter is derived from $J_{\mathrm{rs}}\left(\mathbf{a}''_{\mathrm{Q}}\right)$. Indeed, by minimizing $J_{\mathrm{rs}}\left(\mathbf{a}''_{\mathrm{Q}}\right)$, we easily get

$$\mathbf{a}''_{\mathrm{Q,MR}}=\mathbf{B}''^H_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}. \tag{2.88}$$

Therefore, the indirect VS minimum residual noise filter is

$$\begin{aligned}\overline{\mathbf{h}}'_{\mathrm{MR}}(\mathtt{Q}) &= \mathbf{i} - \mathbf{B}''_{\mathtt{Q}}\mathbf{B}''^{H}_{\mathtt{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \\ &= \mathbf{B}'_{\mathtt{Q}}\mathbf{B}'^{H}_{\mathtt{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}\end{aligned} \tag{2.89}$$

for $\mathtt{Q} \geq 1$ and $\overline{\mathbf{h}}'_{\mathrm{MR}}(0) = \mathbf{0}_{M\times 1}$. We can express the previous filter as

$$\begin{aligned}\overline{\mathbf{h}}'_{\mathrm{MR}}(\mathtt{Q}) &= \sum_{\mathtt{q}=1}^{\mathtt{Q}} \mathbf{b}_{\mathtt{q}}\mathbf{b}^{H}_{\mathtt{q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \\ &= \sum_{q=1}^{Q} \frac{\mathbf{b}_{q}\mathbf{b}^{H}_{q}}{\lambda_q}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i} + \sum_{\mathtt{i}=Q+1}^{\mathtt{Q}} \mathbf{b}_{\mathtt{i}}\mathbf{b}^{H}_{\mathtt{i}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \\ &= \mathbf{h}_{\mathrm{MD}}(Q) + \sum_{\mathtt{i}=Q+1}^{\mathtt{Q}} \mathbf{b}_{\mathtt{i}}\mathbf{b}^{H}_{\mathtt{i}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \\ &= \overline{\mathbf{h}}'_{\mathrm{MR}}(Q) + \sum_{\mathtt{i}=Q+1}^{\mathtt{Q}} \mathbf{b}_{\mathtt{i}}\mathbf{b}^{H}_{\mathtt{i}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}\end{aligned} \tag{2.90}$$

for $\mathtt{Q} < M$ [and $\overline{\mathbf{h}}'_{\mathrm{MR}}(M) = \mathbf{i}$]. We observe that for $\mathtt{Q} \leq Q \leq P$, $\overline{\mathbf{h}}'_{\mathrm{MR}}(\mathtt{Q}) = \mathbf{h}_{\mathrm{MD}}(\mathtt{Q})$. But for $\mathtt{Q} > P$, the two filters are different since $\mathbf{h}_{\mathrm{MD}}(\mathtt{Q})$ is not defined in this context.

We have at least two interesting particular cases:

- $\overline{\mathbf{h}}'_{\mathrm{MR}}(1) = \mathbf{h}_{\mathrm{max},0}$, which corresponds to the maximum SNR filter; and
- $\overline{\mathbf{h}}'_{\mathrm{MR}}(P) = \mathbf{h}_{\mathrm{MVDR}}$, which corresponds to the MVDR filter.

2.6.3.2 Indirect VS Wiener

The indirect VS Wiener filter is obtained from the optimization of the MSE criterion, $J\left(\mathbf{a}''_{\mathtt{Q}}\right)$. The minimization of $J\left(\mathbf{a}''_{\mathtt{Q}}\right)$ leads to

$$\mathbf{a}''_{\mathtt{Q},\mathrm{W}} = \left(\mathbf{\Lambda}''_{\mathtt{Q}} + \mathbf{I}_{M-\mathtt{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathtt{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}. \tag{2.91}$$

We deduce that the indirect VS Wiener filter is

$$\overline{\mathbf{h}}'_{\mathrm{W}}(\mathtt{Q}) = \mathbf{i} - \mathbf{B}''_{\mathtt{Q}}\left(\mathbf{\Lambda}''_{\mathtt{Q}} + \mathbf{I}_{M-\mathtt{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathtt{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \tag{2.92}$$

for $\mathtt{Q} \geq 1$ and $\overline{\mathbf{h}}'_{\mathrm{W}}(0) = \mathbf{h}_{\mathrm{W}} = \mathbf{\Phi}^{-1}_{\mathbf{y}}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i}$, which is the classical Wiener filter. Expression (2.92) can be rewritten as

$$\begin{aligned}\overline{\mathbf{h}}'_{\mathrm{W}}(\mathrm{Q}) &= \mathbf{i} - \mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{y}}\mathbf{i} \\ &\quad + \mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i} \\ &= \mathbf{B}'_{\mathrm{Q}}\left(\mathbf{\Lambda}'_{\mathrm{Q}} + \mathbf{I}_{\mathrm{Q}}\right)^{-1}\mathbf{B}'^{H}_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{y}}\mathbf{i} + \mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i} \\ &= \mathbf{\Phi}_{\mathbf{y}}^{-1}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i} + \mathbf{B}'_{\mathrm{Q}}\left(\mathbf{\Lambda}'_{\mathrm{Q}} + \mathbf{I}_{\mathrm{Q}}\right)^{-1}\mathbf{B}'^{H}_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \\ &= \mathbf{h}_{\mathrm{W}} + \sum_{\mathrm{q}=1}^{\mathrm{Q}}\frac{\mathbf{b}_{\mathrm{q}}\mathbf{b}_{\mathrm{q}}^{H}}{1+\lambda_{\mathrm{q}}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \\ &= \overline{\mathbf{h}}'_{\mathrm{W}}(0) + \sum_{\mathrm{q}=1}^{\mathrm{Q}}\frac{\mathbf{b}_{\mathrm{q}}\mathbf{b}_{\mathrm{q}}^{H}}{1+\lambda_{\mathrm{q}}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}\end{aligned} \tag{2.93}$$

for $\mathrm{Q} < M$ [and $\overline{\mathbf{h}}'_{\mathrm{W}}(M) = \mathbf{i}$]. It is of interest to observe that $\overline{\mathbf{h}}'_{\mathrm{W}}(P) = \mathbf{h}_{\mathrm{MVDR}}$.

2.6.3.3 Indirect VS Tradeoff

The indirect VS tradeoff filter is obtained from the optimization problem:

$$\min_{\mathbf{a}''_{\mathrm{Q}}} J_{\mathrm{rs}}\left(\mathbf{a}''_{\mathrm{Q}}\right) \quad \text{subject to} \quad J_{\mathrm{ds}}\left(\mathbf{a}''_{\mathrm{Q}}\right) = \beta'\phi_{x_1}, \tag{2.94}$$

where $0 \leq \beta' \leq 1$. We find that

$$\mathbf{a}''_{\mathrm{Q},\mathrm{T},\mu'} = \left(\mu'\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i}, \tag{2.95}$$

where $\mu' \geq 0$ is a Lagrange multiplier. As a result, the indirect VS tradeoff filter is

$$\overline{\mathbf{h}}'_{\mathrm{T},\mu'}(\mathrm{Q}) = \mathbf{i} - \mathbf{B}''_{\mathrm{Q}}\left(\mu'\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \tag{2.96}$$

for $\mathrm{Q} \geq 1$ and

$$\begin{aligned}\overline{\mathbf{h}}'_{\mathrm{T},\mu'}(0) &= \mathbf{i} - \left(\mu'\mathbf{\Phi}_{\mathbf{x}} + \mathbf{\Phi}_{\mathbf{v}}\right)^{-1}\mathbf{\Phi}_{\mathbf{v}}\mathbf{i} \\ &= \left(\mathbf{\Phi}_{\mathbf{x}} + \mu'^{-1}\mathbf{\Phi}_{\mathbf{v}}\right)^{-1}\mathbf{\Phi}_{\mathbf{x}}\mathbf{i} \\ &= \mathbf{h}_{\mathrm{T},1/\mu'}.\end{aligned} \tag{2.97}$$

Obviously, $\overline{\mathbf{h}}'_{\mathrm{T},0}(\mathrm{Q}) = \overline{\mathbf{h}}'_{\mathrm{MR}}(\mathrm{Q})$ and $\overline{\mathbf{h}}'_{\mathrm{T},1}(\mathrm{Q}) = \overline{\mathbf{h}}'_{\mathrm{W}}(\mathrm{Q})$. Also, we have $\overline{\mathbf{h}}'_{\mathrm{T},\mu'}(P) = \mathbf{h}_{\mathrm{MVDR}}$.

References

1. J. Chen, J. Benesty, Y. Huang, and S. Doclo, "New insights into the noise reduction Wiener filter," *IEEE Trans. Audio, Speech, Language Process.*, vol. 14, pp. 1218–1234, July 2006.
2. J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Noise Reduction in Speech Processing.* Berlin, Germany: Springer-Verlag, 2009.
3. P. Loizou, *Speech Enhancement: Theory and Practice.* Boca Raton, FL: CRC Press, 2007.
4. P. Vary and R. Martin, *Digital Speech Transmission: Enhancement, Coding and Error Concealment.* Chichester, England: John Wiley & Sons Ltd, 2006.
5. S. H. Jensen, P. C. Hansen, S. D. Hansen, and J. A. Sørensen, "Reduction of broad-band noise in speech by truncated QSVD," *IEEE Trans. Speech, Audio Process.*, vol. 3, pp. 439–448, Nov. 1995.
6. Y. Hu and P. C. Loizou, "A subspace approach for enhancing speech corrupted by colored noise," *IEEE Signal Process. Lett.*, vol. 9, pp. 204–206, July 2002.
7. J. N. Franklin, *Matrix Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1968.

Chapter 3
General Concept with Filtering Matrices

In the previous chapter, we showed how the first element of the desired signal vector can be estimated with variable span (VS) linear filters. In this chapter, we extend this concept to the estimation of the desired signal vector. This leads to variable span linear filtering matrices.

3.1 Signal Model and Problem Formulation

We consider the same signal model as the one presented in Chapter 2, i.e., $\mathbf{y} = \mathbf{x} + \mathbf{v}$, where the correlation matrix of $\mathbf{y}$ is $\mathbf{\Phi}_{\mathbf{y}} = \mathbf{\Phi}_{\mathbf{x}} + \mathbf{\Phi}_{\mathbf{v}}$. Again, it is assumed that the rank of the desired signal correlation matrix, $\mathbf{\Phi}_{\mathbf{x}}$, is equal to $P \leq M$ while the rank of the noise correlation matrix, $\mathbf{\Phi}_{\mathbf{v}}$, is equal to M. The joint diagonalization of $\mathbf{\Phi}_{\mathbf{x}}$ and $\mathbf{\Phi}_{\mathbf{v}}$ will be used (see Chapter 2).

In this study, it is assumed that $\mathbf{x}$ is the desired signal vector. Then, the objective of signal enhancement is to estimate $\mathbf{x}$ from $\mathbf{y}$. This should be done in such a way that the noise is reduced as much as possible with no or little distortion of the desired signal vector [1], [2].

3.2 VS Linear Filtering with a Matrix

Since we want to estimate the desired signal vector, $\mathbf{x}$, of length M, a square filtering matrix is applied to the observation signal vector, $\mathbf{y}$, to get this estimate:

$$\mathbf{z} = \mathbf{H}\mathbf{y}, \tag{3.1}$$

where

J. Benesty et al., *Signal Enhancement with Variable Span Linear Filters*,
Springer Topics in Signal Processing 7, DOI 10.1007/978-981-287-739-0_3

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1^H \\ \mathbf{h}_2^H \\ \vdots \\ \mathbf{h}_M^H \end{bmatrix} \tag{3.2}$$

is a square filtering matrix of size $M \times M$ and $\mathbf{h}_m,\ m = 1, 2, \ldots, M$ are complex-valued filters of length M. It is always possible to write $\mathbf{h}_m$ in a basis formed from the vectors $\mathbf{b}_i,\ i = 1, 2, \ldots, M$, i.e.,

$$\begin{aligned} \mathbf{h}_m &= \mathbf{B}\mathbf{a}_m \\ &= \mathbf{B}'_Q \mathbf{a}'_{m,Q} + \mathbf{B}''_Q \mathbf{a}''_{m,Q}, \end{aligned} \tag{3.3}$$

where the components of

$$\mathbf{a}_m = \begin{bmatrix} \mathbf{a}'^T_{m,Q} \ \mathbf{a}''^T_{m,Q} \end{bmatrix}^T \tag{3.4}$$

are the coordinates of $\mathbf{h}_m$ in the new basis, and $\mathbf{a}'_{m,Q}$ and $\mathbf{a}''_{m,Q}$ are vectors of length Q and $M - Q$, respectively. Therefore, the filtering matrix can be expressed as

$$\begin{aligned} \mathbf{H} &= \mathbf{A}\mathbf{B}^H \\ &= \mathbf{A}'_Q \mathbf{B}'^H_Q + \mathbf{A}''_Q \mathbf{B}''^H_Q, \end{aligned} \tag{3.5}$$

where

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \mathbf{a}_1^H \\ \mathbf{a}_2^H \\ \vdots \\ \mathbf{a}_M^H \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}'_Q \ \mathbf{A}''_Q \end{bmatrix} \end{aligned} \tag{3.6}$$

is an $M \times M$ matrix and $\mathbf{A}'_Q$ and $\mathbf{A}''_Q$ are matrices of size $M \times Q$ and $M \times (M - Q)$, respectively. Now, instead of estimating $\mathbf{H}$ as in conventional approaches, we can estimate, equivalently, $\mathbf{A}$. When $\mathbf{A}$ is estimated, it is then easy to determine $\mathbf{H}$ from (3.5). The case $Q = P$ is interesting because several optimal noise reduction filtering matrices with at most P constraints will lead to $\mathbf{A}''_P = \mathbf{0}_{M\times(M-P)}$ since there is no desired signal in the directions $\mathbf{B}''_P$. Therefore, we can sometimes simplify our problem and force $\mathbf{A}''_P = \mathbf{0}_{M\times(M-P)}$; as a result, the filtering matrix and the estimate are, respectively, $\mathbf{H} = \mathbf{A}'_P \mathbf{B}'^H_P$ and $\mathbf{z} = \mathbf{A}'_P \mathbf{B}'^H_P \mathbf{y}$.

We see from the previous that we can build a more flexible linear filtering matrix. First, we define our variable span linear filter of length M as

$$\mathbf{h}_m(Q) = \mathbf{B}'_Q \mathbf{a}'_{m,Q},\ m = 1, 2, \ldots, M. \tag{3.7}$$

Obviously, $\mathbf{h}_m(Q) \in \mathrm{Span}\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_Q\}$. Then, we find that the variable span filtering matrix of size $M \times M$ is

$$\mathbf{H}(Q) = \mathbf{A}'_Q \mathbf{B}'^H_Q. \tag{3.8}$$

As a consequence, the estimate of $\mathbf{x}$ is

$$\begin{aligned} \mathbf{z} &= \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{x} + \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{v} \\ &= \mathbf{x}_{\mathrm{fd}} + \mathbf{v}_{\mathrm{rn}}, \end{aligned} \tag{3.9}$$

where

$$\mathbf{x}_{\mathrm{fd}} = \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{x} \tag{3.10}$$

is the filtered desired signal and

$$\mathbf{v}_{\mathrm{rn}} = \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{v} \tag{3.11}$$

is the residual noise. We deduce that the correlation matrix of $\mathbf{z}$ is

$$\begin{aligned} \mathbf{\Phi}_{\mathbf{z}} &= E\left(\mathbf{z}\mathbf{z}^H\right) \\ &= \mathbf{A}'_Q \mathbf{\Lambda}'_Q \mathbf{A}'^H_Q + \mathbf{A}'_Q \mathbf{A}'^H_Q, \end{aligned} \tag{3.12}$$

where $\mathbf{\Lambda}'_Q$ is a diagonal matrix containing the first Q eigenvalues of $\mathbf{\Phi}_{\mathbf{v}}^{-1}\mathbf{\Phi}_{\mathbf{x}}$ (see Chapter 2).

Notice that the proposed linear processing implies implicitly that we force $\mathbf{A}''_Q = \mathbf{0}_{M\times(M-Q)}$.

3.3 Performance Measures

This section is dedicated to the definition of the performance measures with the proposed VS linear filtering matrix for noise reduction.

3.3.1 Noise Reduction

The input SNR is defined as

$$\mathrm{iSNR} = \frac{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{x}}\right)}{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{v}}\right)}, \tag{3.13}$$

where $\mathrm{tr}(\cdot)$ denotes the trace of a square matrix.

The output SNR is obtained from the correlation matrix of $\mathbf{z}$. It is easy to see that it is given by

$$\text{oSNR}\left(\mathbf{A}'_Q\right) = \frac{\text{tr}\left(\mathbf{A}'_Q \boldsymbol{\Lambda}'_Q \mathbf{A}'^H_Q\right)}{\text{tr}\left(\mathbf{A}'_Q \mathbf{A}'^H_Q\right)}. \tag{3.14}$$

The usual objective of noise reduction is to find an appropriate $\mathbf{A}'_Q$ such that the output SNR is greater than the input SNR. It can be verified that

$$\text{oSNR}\left(\mathbf{A}'_Q\right) \leq \max_m \frac{\mathbf{a}'^H_{m,Q} \boldsymbol{\Lambda}'_Q \mathbf{a}'_{m,Q}}{\mathbf{a}'^H_{m,Q} \mathbf{a}'_{m,Q}}. \tag{3.15}$$

As a result,

$$\text{oSNR}\left(\mathbf{A}'_Q\right) \leq \lambda_1, \tag{3.16}$$

showing how the output SNR is upper bounded.

The noise reduction factor is given by

$$\xi_{\text{nr}}\left(\mathbf{A}'_Q\right) = \frac{\text{tr}\left(\boldsymbol{\Phi}_{\mathbf{v}}\right)}{\text{tr}\left(\mathbf{A}'_Q \mathbf{A}'^H_Q\right)}. \tag{3.17}$$

For optimal filtering matrices, we should have $\xi_{\text{nr}}\left(\mathbf{A}'_Q\right) \geq 1$.

3.3.2 Desired Signal Distortion

A distortion measure should be related to the processing we do to reducing the level of the noise. One convenient way to evaluate distortion is via the desired signal reduction factor:

$$\xi_{\text{sr}}\left(\mathbf{A}'_Q\right) = \frac{\text{tr}\left(\boldsymbol{\Phi}_{\mathbf{x}}\right)}{\text{tr}\left(\mathbf{A}'_Q \boldsymbol{\Lambda}'_Q \mathbf{A}'^H_Q\right)}. \tag{3.18}$$

For optimal filtering matrices, we should have $\xi_{\text{sr}}\left(\mathbf{A}'_Q\right) \geq 1$. The larger is the value of $\xi_{\text{sr}}\left(\mathbf{A}'_Q\right)$, the more distortion to the desired signal vector.

Obviously, we have the fundamental relationship:

$$\frac{\text{oSNR}\left(\mathbf{A}'_Q\right)}{\text{iSNR}} = \frac{\xi_{\text{nr}}\left(\mathbf{A}'_Q\right)}{\xi_{\text{sr}}\left(\mathbf{A}'_Q\right)}, \tag{3.19}$$

which, basically, shows that nothing comes for free.

We can also evaluate distortion via the desired signal distortion index:

$$\upsilon_{\text{sd}}\left(\mathbf{A}'_Q\right) = \frac{E\left[\left(\mathbf{x}_{\text{fd}} - \mathbf{x}\right)^H \left(\mathbf{x}_{\text{fd}} - \mathbf{x}\right)\right]}{\text{tr}\left(\mathbf{\Phi}_{\mathbf{x}}\right)}. \tag{3.20}$$

For optimal filtering matrices, we should have $\upsilon_{\text{sd}}\left(\mathbf{A}'_Q\right) \leq 1$.

3.3.3 MSE Criterion

We define the error signal vector between the estimated and desired signals as

$$\begin{aligned} \mathbf{e} &= \mathbf{z} - \mathbf{x} \\ &= \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{y} - \mathbf{x} \\ &= \mathbf{e}_{\text{ds}} + \mathbf{e}_{\text{rs}}, \end{aligned} \tag{3.21}$$

where

$$\begin{aligned} \mathbf{e}_{\text{ds}} &= \mathbf{x}_{\text{fd}} - \mathbf{x} \\ &= \left(\mathbf{A}'_Q \mathbf{B}'^H_Q - \mathbf{I}_M\right) \mathbf{x} \end{aligned} \tag{3.22}$$

represents the desired signal distortion and

$$\begin{aligned} \mathbf{e}_{\text{rs}} &= \mathbf{v}_{\text{rn}} \\ &= \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{v} \end{aligned} \tag{3.23}$$

is the residual noise. We deduce that the MSE criterion is

$$\begin{aligned} J\left(\mathbf{A}'_Q\right) &= \text{tr}\left[E\left(\mathbf{e}\mathbf{e}^H\right)\right] \\ &= \text{tr}\left[\mathbf{\Phi}_{\mathbf{x}} - \mathbf{\Phi}_{\mathbf{x}} \mathbf{B}'_Q \mathbf{A}'^H_Q - \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{\Phi}_{\mathbf{x}} + \mathbf{A}'_Q \left(\mathbf{\Lambda}'_Q + \mathbf{I}_Q\right) \mathbf{A}'^H_Q\right] \\ &= J_{\text{ds}}\left(\mathbf{A}'_Q\right) + J_{\text{rs}}\left(\mathbf{A}'_Q\right), \end{aligned} \tag{3.24}$$

where

$$\begin{aligned} J_{\text{ds}}\left(\mathbf{A}'_Q\right) &= \text{tr}\left[E\left(\mathbf{e}_{\text{ds}}\mathbf{e}^H_{\text{ds}}\right)\right] \\ &= \text{tr}\left(\mathbf{\Phi}_{\mathbf{x}} - \mathbf{\Phi}_{\mathbf{x}} \mathbf{B}'_Q \mathbf{A}'^H_Q - \mathbf{A}'_Q \mathbf{B}'^H_Q \mathbf{\Phi}_{\mathbf{x}} + \mathbf{A}'_Q \mathbf{\Lambda}'_Q \mathbf{A}'^H_Q\right) \\ &= \upsilon_{\text{sd}}\left(\mathbf{A}'_Q\right) \text{tr}\left(\mathbf{\Phi}_{\mathbf{x}}\right) \end{aligned} \tag{3.25}$$

and

$$\begin{aligned} J_{\mathrm{rs}}\left(\mathbf{A}_Q'\right) &= \mathrm{tr}\left[E\left(\mathbf{e}_{\mathrm{rs}}\mathbf{e}_{\mathrm{rs}}^H\right)\right] \\ &= \mathrm{tr}\left(\mathbf{A}_Q'\mathbf{A}_Q'^H\right) \\ &= \frac{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{v}}\right)}{\xi_{\mathrm{nr}}\left(\mathbf{A}_Q'\right)}. \end{aligned} \tag{3.26}$$

We deduce that

$$\begin{aligned} \frac{J_{\mathrm{ds}}\left(\mathbf{A}_Q'\right)}{J_{\mathrm{rs}}\left(\mathbf{A}_Q'\right)} &= \mathrm{iSNR} \times \xi_{\mathrm{nr}}\left(\mathbf{A}_Q'\right) \times \upsilon_{\mathrm{sd}}\left(\mathbf{A}_Q'\right) \\ &= \mathrm{oSNR}\left(\mathbf{A}_Q'\right) \times \xi_{\mathrm{sr}}\left(\mathbf{A}_Q'\right) \times \upsilon_{\mathrm{sd}}\left(\mathbf{A}_Q'\right), \end{aligned} \tag{3.27}$$

showing how the different performances measures are related to the MSEs.

3.4 Optimal VS Linear Filtering Matrices

In this section, a large class of VS filtering matrices for noise reduction are derived and their connections are highlighted.

3.4.1 VS Minimum Distortion

The minimization of $J_{\mathrm{ds}}\left(\mathbf{A}_Q'\right)$ leads to

$$\mathbf{A}_{Q,\mathrm{MD}}' = \mathbf{\Phi}_{\mathbf{x}}\mathbf{B}_Q'\mathbf{\Lambda}_Q'^{-1}, \tag{3.28}$$

where it is assumed that $Q \leq P$. Therefore, the VS minimum distortion filtering matrix is

$$\begin{aligned} \mathbf{H}_{\mathrm{MD}}(Q) &= \mathbf{A}_{Q,\mathrm{MD}}'\mathbf{B}_Q'^H \\ &= \mathbf{\Phi}_{\mathbf{x}}\sum_{q=1}^{Q}\frac{\mathbf{b}_q\mathbf{b}_q^H}{\lambda_q},\ Q \leq P. \end{aligned} \tag{3.29}$$

For $Q = P$, we get the well-known MVDR filtering matrix:

$$\begin{aligned}\mathbf{H}_{\mathrm{MVDR}} &= \mathbf{H}_{\mathrm{MD}}(P) \\ &= \mathbf{A}'_{P,\mathrm{MD}}\mathbf{B}'^{H}_{P} \\ &= \mathbf{\Phi}_{\mathbf{x}}\sum_{p=1}^{P}\frac{\mathbf{b}_p\mathbf{b}_p^H}{\lambda_p} \\ &= \mathbf{\Phi}_{\mathbf{v}}\sum_{p=1}^{P}\mathbf{b}_p\mathbf{b}_p^H.\end{aligned} \tag{3.30}$$

Let us show now that (3.30) is the MVDR filtering matrix. With $\mathbf{H}_{\mathrm{MVDR}}$, the filtered desired signal vector is

$$\begin{aligned}\mathbf{x}_{\mathrm{fd}} &= \mathbf{\Phi}_{\mathbf{v}}\mathbf{B}'_{P}\mathbf{B}'^{H}_{P}\mathbf{x} \\ &= \left(\mathbf{I}_M - \mathbf{\Phi}_{\mathbf{v}}\mathbf{B}''_{P}\mathbf{B}''^{H}_{P}\right)\mathbf{x} \\ &= \mathbf{x} - \mathbf{i}^T\mathbf{\Phi}_{\mathbf{v}}\mathbf{B}''_{P}\mathbf{B}''^{H}_{P}\mathbf{x} \\ &= \mathbf{x},\end{aligned} \tag{3.31}$$

where we have used (2.10) and (2.11) in the previous expression. Then, it is clear that

$$\upsilon_{\mathrm{sd}}\left(\mathbf{A}'_{P,\mathrm{MD}}\right) = 0, \tag{3.32}$$

proving that, indeed, $\mathbf{H}_{\mathrm{MVDR}}$ is the MVDR filtering matrix. If $Q = P = M$, then we obtain the identity filtering matrix, i.e., $\mathbf{H}_{\mathrm{MVDR}}(M) = \mathbf{I}_M$, which does not affect the observation signal vector.

Another interesting case of (3.29) is $Q = 1$. In this scenario, we obtain the maximum SNR filtering matrix:

$$\begin{aligned}\mathbf{H}_{\max,0} &= \mathbf{H}_{\mathrm{MD}}(1) \\ &= \mathbf{A}'_{1,\mathrm{MD}}\mathbf{b}_1^H \\ &= \mathbf{\Phi}_{\mathbf{x}}\frac{\mathbf{b}_1\mathbf{b}_1^H}{\lambda_1}.\end{aligned} \tag{3.33}$$

Indeed, it can be verified that

$$\mathrm{oSNR}\left(\mathbf{A}'_{1,\mathrm{MD}}\right) = \lambda_1. \tag{3.34}$$

We should always have

$$\mathrm{oSNR}\left(\mathbf{A}'_{P,\mathrm{MD}}\right) \leq \mathrm{oSNR}\left(\mathbf{A}'_{P-1,\mathrm{MD}}\right) \leq \cdots \leq \mathrm{oSNR}\left(\mathbf{A}'_{1,\mathrm{MD}}\right) = \lambda_1 \tag{3.35}$$

and

$$\upsilon_{\mathrm{sd}}\left(\mathbf{A}'_{P,\mathrm{MD}}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{A}'_{P-1,\mathrm{MD}}\right) \leq \cdots \leq \upsilon_{\mathrm{sd}}\left(\mathbf{A}'_{1,\mathrm{MD}}\right) \leq 1. \tag{3.36}$$

3.4.2 VS Wiener

The VS Wiener filtering matrix is derived from the minimization of the MSE criterion, $J\left(\mathbf{A}'_Q\right)$. From this optimization, we obtain

$$\mathbf{A}'_{Q,\mathrm{W}} = \mathbf{\Phi}_{\mathbf{x}}\mathbf{B}'_Q\left(\mathbf{\Lambda}'_Q + \mathbf{I}_Q\right)^{-1}, \tag{3.37}$$

where $Q \leq M$. We deduce that the VS Wiener filtering matrix is

$$\begin{aligned}\mathbf{H}_{\mathrm{W}}(Q) &= \mathbf{A}'_{Q,\mathrm{W}}\mathbf{B}'^H_Q \\ &= \mathbf{\Phi}_{\mathbf{x}}\sum_{q=1}^{Q}\frac{\mathbf{b}_q\mathbf{b}_q^H}{1+\lambda_q}.\end{aligned} \tag{3.38}$$

It is interesting to compare $\mathbf{H}_{\mathrm{W}}(Q)$ to $\mathbf{H}_{\mathrm{MD}}(Q)$. The two VS filtering matrices are very close to each other; they differ by the weighting function, which strongly depends on the eigenvalues of the joint diagonalization. For the VS Wiener filtering matrix, this function is equal to $(1+\lambda_q)^{-1}$ while it is equal to λ_q^{-1} for the VS minimum distortion filtering matrix. Also, in the latter filter, Q must be smaller than or equal to P, while Q can be greater than P in the former one.

One important particular case of (3.38) is $Q = M$. In this situation, we obtain the classical Wiener filtering matrix:

$$\begin{aligned}\mathbf{H}_{\mathrm{W}} &= \mathbf{H}_{\mathrm{W}}(M) \\ &= \mathbf{A}'_{M,\mathrm{W}}\mathbf{B}'^H_M \\ &= \mathbf{\Phi}_{\mathbf{x}}\sum_{m=1}^{M}\frac{\mathbf{b}_m\mathbf{b}_m^H}{1+\lambda_m} \\ &= \mathbf{\Phi}_{\mathbf{x}}\mathbf{\Phi}_{\mathbf{y}}^{-1}.\end{aligned} \tag{3.39}$$

For $Q = 1$, we obtain another form of the maximum SNR filtering matrix:

$$\begin{aligned}\mathbf{H}_{\max,1} &= \mathbf{H}_{\mathrm{W}}(1) \\ &= \mathbf{A}'_{1,\mathrm{W}}\mathbf{b}_1^H \\ &= \mathbf{\Phi}_{\mathbf{x}}\frac{\mathbf{b}_1\mathbf{b}_1^H}{1+\lambda_1},\end{aligned} \tag{3.40}$$

since

$$\mathrm{oSNR}\left(\mathbf{A}'_{1,\mathrm{W}}\right) = \lambda_1. \tag{3.41}$$

We should always have

$$\mathrm{oSNR}\left(\mathbf{A}'_{M,\mathrm{W}}\right) \leq \mathrm{oSNR}\left(\mathbf{A}'_{M-1,\mathrm{W}}\right) \leq \cdots \leq \mathrm{oSNR}\left(\mathbf{A}'_{1,\mathrm{W}}\right) = \lambda_1 \tag{3.42}$$

and

$$\upsilon_{\rm sd}\left(\mathbf{A}'_{M,\rm W}\right) \leq \upsilon_{\rm sd}\left(\mathbf{A}'_{M-1,\rm W}\right) \leq \cdots \leq \upsilon_{\rm sd}\left(\mathbf{A}'_{1,\rm W}\right) \leq 1. \tag{3.43}$$

3.4.3 VS Tradeoff

The most practical approach that can compromise between noise reduction and desired signal distortion is the VS tradeoff filtering matrix obtained by

$$\min_{\mathbf{A}'_Q} J_{\rm ds}\left(\mathbf{A}'_Q\right) \quad \text{subject to} \quad J_{\rm rs}\left(\mathbf{A}'_Q\right) = \beta \text{tr}\left(\mathbf{\Phi}_{\mathbf{v}}\right), \tag{3.44}$$

where $0 \leq \beta \leq 1$, to ensure that filtering achieves some degree of noise reduction. We find that the optimal filtering matrix is

$$\mathbf{H}_{{\rm T},\mu}(Q) = \mathbf{\Phi}_{\mathbf{x}} \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\mu + \lambda_q}, \tag{3.45}$$

where $\mu \geq 0$ is a Lagrange multiplier[1]. For $\mu = 0$ and $\mu = 1$, we get the VS minimum distortion and VS Wiener filtering matrices, respectively.

For $Q = M$, we obtain the classical tradeoff filtering matrix:

$$\begin{aligned}\mathbf{H}_{{\rm T},\mu} &= \mathbf{H}_{{\rm T},\mu}(M) \\ &= \mathbf{\Phi}_{\mathbf{x}} \sum_{m=1}^{M} \frac{\mathbf{b}_m \mathbf{b}_m^H}{\mu + \lambda_m} \\ &= \mathbf{\Phi}_{\mathbf{x}} \left(\mathbf{\Phi}_{\mathbf{x}} + \mu \mathbf{\Phi}_{\mathbf{v}}\right)^{-1}\end{aligned} \tag{3.46}$$

and for $Q = 1$, we obtain the maximum SNR filtering matrix:

$$\begin{aligned}\mathbf{H}_{\max,\mu} &= \mathbf{H}_{{\rm T},\mu}(1) \\ &= \mathbf{\Phi}_{\mathbf{x}} \frac{\mathbf{b}_1 \mathbf{b}_1^H}{\mu + \lambda_1}.\end{aligned} \tag{3.47}$$

In Table 3.1, we present all optimal VS filtering matrices developed in this section, showing how they are strongly related.

[1] For $\mu = 0$, Q must be smaller than or equal to P.

Table 3.1 Optimal VS linear filtering matrices for signal enhancement.

VS MD:	$\mathbf{H}_{\text{MD}}(Q) = \mathbf{\Phi}_{\mathbf{x}} \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\lambda_q},\ Q \leq P$
MVDR:	$\mathbf{H}_{\text{MVDR}} = \mathbf{\Phi}_{\mathbf{x}} \sum_{p=1}^{P} \frac{\mathbf{b}_p \mathbf{b}_p^H}{\lambda_p}$
VS Wiener:	$\mathbf{H}_{\text{W}}(Q) = \mathbf{\Phi}_{\mathbf{x}} \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{1+\lambda_q},\ Q \leq M$
Wiener:	$\mathbf{H}_{\text{W}} = \mathbf{\Phi}_{\mathbf{x}} \sum_{m=1}^{M} \frac{\mathbf{b}_m \mathbf{b}_m^H}{1+\lambda_m} = \mathbf{\Phi}_{\mathbf{x}} \mathbf{\Phi}_{\mathbf{y}}^{-1}$
VS Tradeoff:	$\mathbf{H}_{\text{T},\mu}(Q) = \mathbf{\Phi}_{\mathbf{x}} \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\mu+\lambda_q},\ \mu \geq 0$
Tradeoff:	$\mathbf{H}_{\text{T},\mu} = \mathbf{\Phi}_{\mathbf{x}} \sum_{m=1}^{M} \frac{\mathbf{b}_m \mathbf{b}_m^H}{\mu+\lambda_m} = \mathbf{\Phi}_{\mathbf{x}} \left(\mathbf{\Phi}_{\mathbf{x}} + \mu \mathbf{\Phi}_{\mathbf{v}}\right)^{-1}$
Maximum SNR:	$\mathbf{H}_{\max,\mu} = \mathbf{\Phi}_{\mathbf{x}} \frac{\mathbf{b}_1 \mathbf{b}_1^H}{\mu+\lambda_1}$

3.5 Indirect Optimal VS Linear Filtering Matrices

The principle of the indirect approach is to perform noise reduction in two successive stages. First, we find an estimate of the noise signal vector, which is then used in the second stage to get an estimate of the desired signal vector.

3.5.1 Indirect Approach

In the indirect approach, the noise signal vector is first estimated by

$$\widehat{\mathbf{v}} = \mathbf{H}'\mathbf{x} + \mathbf{H}'\mathbf{v}, \tag{3.48}$$

where $\mathbf{H}'$ is a complex-valued filtering matrix of size $M \times M$. The output SNR corresponding to (3.48) is

$$\text{oSNR}_{\widehat{\mathbf{v}}}\left(\mathbf{H}'\right) = \frac{\text{tr}\left(\mathbf{H}'\mathbf{\Phi}_{\mathbf{x}}\mathbf{H}'^H\right)}{\text{tr}\left(\mathbf{H}'\mathbf{\Phi}_{\mathbf{v}}\mathbf{H}'^H\right)}. \tag{3.49}$$

It is clear that the filtering matrix that minimizes $\text{oSNR}_{\widehat{\mathbf{v}}}\left(\mathbf{H}'\right)$ has the form:

$$\mathbf{H}' = \mathbf{A}_P''\mathbf{B}_P''^H, \tag{3.50}$$

where $\mathbf{A}_P''$ and $\mathbf{B}_P''$ were already defined. With (3.50), $\text{oSNR}_{\widehat{\mathbf{v}}}\left(\mathbf{H}'\right) = 0$ and $\widehat{\mathbf{v}}$ can be seen as the estimate of the noise signal vector.

Let us consider the more general scenario:

$$\mathbf{H}'(\mathsf{Q}) = \mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}}, \tag{3.51}$$

where $0 \leq \mathsf{Q} < M$, and $\mathbf{A}''_{\mathsf{Q}}$ and $\mathbf{B}''_{\mathsf{Q}}$ are matrices of size $M \times (M - \mathsf{Q})$. We observe that $\mathbf{h}'_m(\mathsf{Q}) \in \text{Span}\{\mathbf{b}_{\mathsf{Q}+1}, \mathbf{b}_{\mathsf{Q}+2}, \ldots, \mathbf{b}_M\}$, where $\mathbf{h}'^{H}_m(\mathsf{Q})$ is the mth line of $\mathbf{H}'(\mathsf{Q})$. We deduce that

$$\widehat{\mathbf{v}} = \mathbf{H}'(\mathsf{Q})\mathbf{x} + \mathbf{H}'^{H}(\mathsf{Q})\mathbf{v}. \tag{3.52}$$

But, in general, $\text{oSNR}_{\widehat{\mathbf{v}}}\left[\mathbf{H}'(\mathsf{Q})\right] \neq 0$, implying some distortion to the desired signal vector.

The final step consists of the estimation of the desired signal vector, $\mathbf{x}$, as the difference between the observation signal vector, $\mathbf{y}$, and the estimator $\widehat{\mathbf{v}}$ from (3.52), i.e.,

$$\begin{aligned} \mathbf{z}' &= \mathbf{y} - \mathbf{H}'(\mathsf{Q})\mathbf{x} - \mathbf{H}'(\mathsf{Q})\mathbf{v} \\ &= \mathbf{x} - \mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}}\mathbf{x} + \mathbf{v} - \mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}}\mathbf{v} \\ &= \overline{\mathbf{H}}'(\mathsf{Q})\mathbf{y}, \end{aligned} \tag{3.53}$$

where

$$\overline{\mathbf{H}}'(\mathsf{Q}) = \mathbf{I}_M - \mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}} \tag{3.54}$$

is the equivalent filtering matrix applied to the observation signal vector.

3.5.2 MSE Criterion and Performance Measures

The error signal vector between the estimated and desired signals is

$$\begin{aligned} \mathbf{e}' &= \mathbf{z}' - \mathbf{x} \\ &= -\mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}}\mathbf{x} + \mathbf{v} - \mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}}\mathbf{v}, \end{aligned} \tag{3.55}$$

which can be expressed as the sum of two orthogonal error signal vectors:

$$\mathbf{e}' = \mathbf{e}'_{\text{ds}} + \mathbf{e}'_{\text{rs}}, \tag{3.56}$$

where

$$\mathbf{e}'_{\text{ds}} = -\mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}}\mathbf{x} \tag{3.57}$$

is the distortion of the desired signal due to the filtering matrix and

$$\mathbf{e}'_{\text{rs}} = \mathbf{v} - \mathbf{A}''_{\mathsf{Q}} \mathbf{B}''^{H}_{\mathsf{Q}}\mathbf{v} \tag{3.58}$$

represents the residual noise. Then, the MSE criterion is

$$\begin{aligned} J\left(\mathbf{A}_{\mathbb{Q}}''\right) &= \mathrm{tr}\left[E\left(\mathbf{e}'\mathbf{e}'^{H}\right)\right] \\ &= \mathrm{tr}\left[\mathbf{\Phi}_{\mathbf{v}} - \mathbf{\Phi}_{\mathbf{v}}\mathbf{B}_{\mathbb{Q}}''\mathbf{A}_{\mathbb{Q}}''^{H} - \mathbf{A}_{\mathbb{Q}}''\mathbf{B}_{\mathbb{Q}}''^{H}\mathbf{\Phi}_{\mathbf{v}} + \mathbf{A}_{\mathbb{Q}}''\left(\mathbf{\Lambda}_{\mathbb{Q}}'' + \mathbf{I}_{M-\mathbb{Q}}\right)\mathbf{A}_{\mathbb{Q}}''^{H}\right] \\ &= J_{\mathrm{ds}}\left(\mathbf{A}_{\mathbb{Q}}''\right) + J_{\mathrm{rs}}\left(\mathbf{A}_{\mathbb{Q}}''\right), \end{aligned} \tag{3.59}$$

where $\mathbf{I}_{M-\mathbb{Q}}$ is the $(M-\mathbb{Q}) \times (M-\mathbb{Q})$ identity matrix,

$$\mathbf{\Lambda}_{\mathbb{Q}}'' = \mathrm{diag}\left(\lambda_{\mathbb{Q}+1}, \lambda_{\mathbb{Q}+2}, \ldots, \lambda_{M}\right) \tag{3.60}$$

is a diagonal matrix containing the last $M - \mathbb{Q}$ eigenvalues of $\mathbf{\Phi}_{\mathbf{v}}^{-1}\mathbf{\Phi}_{\mathbf{x}}$,

$$\begin{aligned} J_{\mathrm{ds}}\left(\mathbf{A}_{\mathbb{Q}}''\right) &= \mathrm{tr}\left[E\left(\mathbf{e}_{\mathrm{ds}}'\mathbf{e}_{\mathrm{ds}}'^{H}\right)\right] \\ &= \upsilon_{\mathrm{sd}}\left(\mathbf{A}_{\mathbb{Q}}''\right)\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{x}}\right) \end{aligned} \tag{3.61}$$

is the distortion-based MSE,

$$\upsilon_{\mathrm{sd}}\left(\mathbf{A}_{\mathbb{Q}}''\right) = \frac{\mathrm{tr}\left(\mathbf{A}_{\mathbb{Q}}''\mathbf{\Lambda}_{\mathbb{Q}}''\mathbf{A}_{\mathbb{Q}}''^{H}\right)}{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{x}}\right)} \tag{3.62}$$

is the desired signal distortion index,

$$\begin{aligned} J_{\mathrm{rs}}\left(\mathbf{A}_{\mathbb{Q}}''\right) &= \mathrm{tr}\left[E\left(\mathbf{e}_{\mathrm{rs}}'\mathbf{e}_{\mathrm{rs}}'^{H}\right)\right] \\ &= \frac{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{v}}\right)}{\xi_{\mathrm{nr}}\left(\mathbf{A}_{\mathbb{Q}}''\right)} \end{aligned} \tag{3.63}$$

is the MSE corresponding to the residual noise, and

$$\xi_{\mathrm{nr}}\left(\mathbf{a}_{\mathbb{Q}}''\right) = \frac{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{v}}\right)}{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{v}} - \mathbf{\Phi}_{\mathbf{v}}\mathbf{B}_{\mathbb{Q}}''\mathbf{A}_{\mathbb{Q}}''^{H} - \mathbf{A}_{\mathbb{Q}}''\mathbf{B}_{\mathbb{Q}}''^{H}\mathbf{\Phi}_{\mathbf{v}} + \mathbf{A}_{\mathbb{Q}}''\mathbf{A}_{\mathbb{Q}}''^{H}\right)} \tag{3.64}$$

is the noise reduction factor. We deduce that

$$\begin{aligned} \frac{J_{\mathrm{ds}}\left(\mathbf{A}_{\mathbb{Q}}''\right)}{J_{\mathrm{rs}}\left(\mathbf{A}_{\mathbb{Q}}''\right)} &= \mathrm{iSNR} \times \xi_{\mathrm{nr}}\left(\mathbf{A}_{\mathbb{Q}}''\right) \times \upsilon_{\mathrm{sd}}\left(\mathbf{A}_{\mathbb{Q}}''\right) \\ &= \mathrm{oSNR}\left(\mathbf{A}_{\mathbb{Q}}''\right) \times \xi_{\mathrm{sr}}\left(\mathbf{A}_{\mathbb{Q}}''\right) \times \upsilon_{\mathrm{sd}}\left(\mathbf{A}_{\mathbb{Q}}''\right), \end{aligned} \tag{3.65}$$

where

$$\mathrm{oSNR}\left(\mathbf{A}_{\mathbb{Q}}''\right) = \frac{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{x}} - \mathbf{\Phi}_{\mathbf{x}}\mathbf{B}_{\mathbb{Q}}''\mathbf{A}_{\mathbb{Q}}''^{H} - \mathbf{A}_{\mathbb{Q}}''\mathbf{B}_{\mathbb{Q}}''^{H}\mathbf{\Phi}_{\mathbf{x}} + \mathbf{A}_{\mathbb{Q}}''\mathbf{\Lambda}_{\mathbb{Q}}''\mathbf{A}_{\mathbb{Q}}''^{H}\right)}{\mathrm{tr}\left(\mathbf{\Phi}_{\mathbf{v}} - \mathbf{\Phi}_{\mathbf{v}}\mathbf{B}_{\mathbb{Q}}''\mathbf{a}_{\mathbb{Q}}''^{H} - \mathbf{A}_{\mathbb{Q}}''\mathbf{B}_{\mathbb{Q}}''^{H}\mathbf{\Phi}_{\mathbf{v}} + \mathbf{A}_{\mathbb{Q}}''\mathbf{A}_{\mathbb{Q}}''^{H}\right)} \tag{3.66}$$

is the output SNR and

$$\xi_{\mathrm{sr}}\left(\mathbf{A}''_{\mathtt{Q}}\right)=\frac{\operatorname{tr}\left(\boldsymbol{\Phi}_{\mathbf{x}}\right)}{\operatorname{tr}\left(\boldsymbol{\Phi}_{\mathbf{x}}-\boldsymbol{\Phi}_{\mathbf{x}}\mathbf{B}''_{\mathtt{Q}}\mathbf{A}''^{H}_{\mathtt{Q}}-\mathbf{A}''_{\mathtt{Q}}\mathbf{B}''^{H}_{\mathtt{Q}}\boldsymbol{\Phi}_{\mathbf{x}}+\mathbf{A}''_{\mathtt{Q}}\boldsymbol{\Lambda}''_{\mathtt{Q}}\mathbf{A}''^{H}_{\mathtt{Q}}\right)} \tag{3.67}$$

is the desired signal reduction factor.

3.5.3 Optimal Filtering Matrices

3.5.3.1 Indirect VS Minimum Residual Noise

The indirect VS minimum residual noise filtering matrix is derived from the minimization of $J_{\mathrm{rs}}\left(\mathbf{A}''_{\mathtt{Q}}\right)$. We easily find that

$$\mathbf{A}''_{\mathtt{Q},\mathrm{MR}}=\boldsymbol{\Phi}_{\mathbf{v}}\mathbf{B}''_{\mathtt{Q}}. \tag{3.68}$$

Therefore, the indirect VS minimum residual noise filtering matrix is

$$\begin{aligned}\overline{\mathbf{H}}'_{\mathrm{MR}}(\mathtt{Q})&=\mathbf{I}_M-\boldsymbol{\Phi}_{\mathbf{v}}\mathbf{B}''_{\mathtt{Q}}\mathbf{B}''^{H}_{\mathtt{Q}}\\&=\boldsymbol{\Phi}_{\mathbf{v}}\mathbf{B}'_{\mathtt{Q}}\mathbf{B}'^{H}_{\mathtt{Q}}\end{aligned} \tag{3.69}$$

for $\mathtt{Q}\geq 1$ and $\overline{\mathbf{H}}'_{\mathrm{MR}}(0)=\mathbf{0}_{M\times M}$. We can rewrite the previous filtering matrix as

$$\begin{aligned}\overline{\mathbf{H}}'_{\mathrm{MR}}(\mathtt{Q})&=\boldsymbol{\Phi}_{\mathbf{v}}\sum_{\mathtt{q}=1}^{\mathtt{Q}}\mathbf{b}_{\mathtt{q}}\mathbf{b}_{\mathtt{q}}^{H}\\&=\boldsymbol{\Phi}_{\mathbf{x}}\sum_{q=1}^{Q}\frac{\mathbf{b}_q\mathbf{b}_q^H}{\lambda_q}+\boldsymbol{\Phi}_{\mathbf{v}}\sum_{\mathtt{i}=Q+1}^{\mathtt{Q}}\mathbf{b}_{\mathtt{i}}\mathbf{b}_{\mathtt{i}}^{H}\\&=\mathbf{H}_{\mathrm{MD}}(Q)+\boldsymbol{\Phi}_{\mathbf{v}}\sum_{\mathtt{i}=Q+1}^{\mathtt{Q}}\mathbf{b}_{\mathtt{i}}\mathbf{b}_{\mathtt{i}}^{H}\\&=\overline{\mathbf{H}}'_{\mathrm{MR}}(Q)+\boldsymbol{\Phi}_{\mathbf{v}}\sum_{\mathtt{i}=Q+1}^{\mathtt{Q}}\mathbf{b}_{\mathtt{i}}\mathbf{b}_{\mathtt{i}}^{H}\end{aligned} \tag{3.70}$$

for $\mathtt{Q}<M$ [and $\overline{\mathbf{H}}'_{\mathrm{MR}}(M)=\mathbf{I}_M$]. We observe that for $\mathtt{Q}\leq Q\leq P$, $\overline{\mathbf{H}}'_{\mathrm{MR}}(\mathtt{Q})=\mathbf{H}_{\mathrm{MD}}(\mathtt{Q})$. But for $\mathtt{Q}>P$, the two filtering matrices are different since $\mathbf{H}_{\mathrm{MD}}(\mathtt{Q})$ is not defined in this context.

We have at least two interesting particular cases:

- $\overline{\mathbf{H}}'_{\mathrm{MR}}(1)=\mathbf{H}_{\max,0}$, which corresponds to the maximum SNR filtering matrix; and
- $\overline{\mathbf{H}}'_{\mathrm{MR}}(P)=\mathbf{H}_{\mathrm{MVDR}}$, which corresponds to the MVDR filtering matrix.

3.5.3.2 Indirect VS Wiener

This filtering matrix is obtained from the optimization of the MSE criterion, $J\left(\mathbf{A}''_{\mathrm{Q}}\right)$. The minimization of $J\left(\mathbf{A}''_{\mathrm{Q}}\right)$ leads to

$$\mathbf{A}''_{\mathrm{Q},\mathrm{W}} = \mathbf{\Phi_v}\mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}. \tag{3.71}$$

We deduce that the indirect VS Wiener filtering matrix is

$$\overline{\mathbf{H}}'_{\mathrm{W}}(\mathrm{Q}) = \mathbf{I}_M - \mathbf{\Phi_v}\mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}} \tag{3.72}$$

for $\mathrm{Q} \geq 1$ and $\overline{\mathbf{H}}'_{\mathrm{W}}(0) = \mathbf{H}_{\mathrm{W}} = \mathbf{\Phi_x}\mathbf{\Phi_y}^{-1}$, which is the classical Wiener filtering matrix. Expression (3.72) can be rewritten as

$$\begin{aligned}
\overline{\mathbf{H}}'_{\mathrm{W}}(\mathrm{Q}) &= \mathbf{I}_M - \mathbf{\Phi_y}\mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}} \\
&\quad + \mathbf{\Phi_x}\mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}} \\
&= \mathbf{\Phi_y}\mathbf{B}'_{\mathrm{Q}}\left(\mathbf{\Lambda}'_{\mathrm{Q}} + \mathbf{I}_{\mathrm{Q}}\right)^{-1}\mathbf{B}'^{H}_{\mathrm{Q}} + \mathbf{\Phi_x}\mathbf{B}''_{\mathrm{Q}}\left(\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathrm{Q}} \\
&= \mathbf{\Phi_x}\mathbf{\Phi_y}^{-1} + \mathbf{\Phi_v}\mathbf{B}'_{\mathrm{Q}}\left(\mathbf{\Lambda}'_{\mathrm{Q}} + \mathbf{I}_{\mathrm{Q}}\right)^{-1}\mathbf{B}'^{H}_{\mathrm{Q}} \\
&= \mathbf{H}_{\mathrm{W}} + \mathbf{\Phi_v}\sum_{\mathrm{q}=1}^{\mathrm{Q}}\frac{\mathbf{b}_{\mathrm{q}}\mathbf{b}^{H}_{\mathrm{q}}}{1+\lambda_{\mathrm{q}}} \\
&= \overline{\mathbf{H}}'_{\mathrm{W}}(0) + \mathbf{\Phi_v}\sum_{\mathrm{q}=1}^{\mathrm{Q}}\frac{\mathbf{b}_{\mathrm{q}}\mathbf{b}^{H}_{\mathrm{q}}}{1+\lambda_{\mathrm{q}}}
\end{aligned} \tag{3.73}$$

for $\mathrm{Q} < M$ [and $\overline{\mathbf{H}}'_{\mathrm{W}}(M) = \mathbf{I}_M$]. It is of interest to observe that $\overline{\mathbf{H}}'_{\mathrm{W}}(P) = \mathbf{H}_{\mathrm{MVDR}}$.

3.5.3.3 Indirect VS Tradeoff

The indirect VS tradeoff filtering matrix is obtained from the optimization problem:

$$\min_{\mathbf{A}''_{\mathrm{Q}}} J_{\mathrm{rs}}\left(\mathbf{A}''_{\mathrm{Q}}\right) \quad \text{subject to} \quad J_{\mathrm{ds}}\left(\mathbf{A}''_{\mathrm{Q}}\right) = \beta'\mathrm{tr}\left(\mathbf{\Phi_x}\right), \tag{3.74}$$

where $0 \leq \beta' \leq 1$. We find that

$$\mathbf{A}''_{\mathrm{Q},\mathrm{T},\mu'} = \mathbf{\Phi_v}\mathbf{B}''_{\mathrm{Q}}\left(\mu'\mathbf{\Lambda}''_{\mathrm{Q}} + \mathbf{I}_{M-\mathrm{Q}}\right)^{-1}, \tag{3.75}$$

where $\mu' \geq 0$ is a Lagrange multiplier. As a result, the indirect VS tradeoff filtering matrix is

$$\overline{\mathbf{H}}'_{\mathrm{T},\mu'}(\mathsf{Q}) = \mathbf{I}_M - \mathbf{\Phi}_{\mathbf{v}}\mathbf{B}''_{\mathsf{Q}}\left(\mu'\mathbf{\Lambda}''_{\mathsf{Q}} + \mathbf{I}_{M-\mathsf{Q}}\right)^{-1}\mathbf{B}''^{H}_{\mathsf{Q}} \tag{3.76}$$

for $\mathsf{Q} \geq 1$ and

$$\begin{aligned}\overline{\mathbf{H}}'_{\mathrm{T},\mu'}(0) &= \mathbf{I}_M - \mathbf{\Phi}_{\mathbf{v}}\left(\mu'\mathbf{\Phi}_{\mathbf{x}} + \mathbf{\Phi}_{\mathbf{v}}\right)^{-1} \\ &= \mathbf{\Phi}_{\mathbf{x}}\left(\mathbf{\Phi}_{\mathbf{x}} + \mu'^{-1}\mathbf{\Phi}_{\mathbf{v}}\right)^{-1} \\ &= \mathbf{H}_{\mathrm{T},1/\mu'}.\end{aligned} \tag{3.77}$$

Obviously, $\overline{\mathbf{H}}'_{\mathrm{T},0}(\mathsf{Q}) = \overline{\mathbf{H}}'_{\mathrm{MR}}(\mathsf{Q})$ and $\overline{\mathbf{H}}'_{\mathrm{T},1}(\mathsf{Q}) = \overline{\mathbf{H}}'_{\mathrm{W}}(\mathsf{Q})$. Also, we have $\overline{\mathbf{H}}'_{\mathrm{T},\mu'}(P) = \mathbf{H}_{\mathrm{MVDR}}$.

References

1. J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Noise Reduction in Speech Processing*. Berlin, Germany: Springer-Verlag, 2009.
2. J. Benesty and J. Chen, *Optimal Time-Domain Noise Reduction Filters–A Theoretical Study*. Berlin, Germany: SpringerBriefs in Electrical and Computer Engineering, 2011.

Chapter 4
Single-Channel Signal Enhancement in the STFT Domain

In this chapter, we study the signal enhancement problem in the convenient short-time Fourier transform (STFT) domain. Contrary to most conventional approaches, we do not assume here that successive STFT frames are uncorrelated. As a consequence, the interframe correlation is now taken into account and a filter is used in each subband instead of just a gain to enhance the noisy signal. We show how to apply some of the concepts of variable span (VS) linear filtering to this problem.

4.1 Signal Model and Problem Formulation

The noise reduction problem considered in this chapter is one of recovering the desired signal, $x(t)$, t being the time index, of zero mean from the noisy observation (sensor signal) [1]:

$$y(t) = x(t) + v(t), \tag{4.1}$$

where $v(t)$ is the unwanted additive noise, which is assumed to be a zero-mean random process white or colored but uncorrelated with $x(t)$. All signals are considered to be real, stationary, and broadband.

Using the short-time Fourier transform (STFT), (4.1) can be rewritten in the time-frequency domain as [2]

$$Y(k,n) = X(k,n) + V(k,n), \tag{4.2}$$

where the zero-mean complex random variables $Y(k,n)$, $X(k,n)$, and $V(k,n)$ are the STFTs of $y(t)$, $x(t)$, and $v(t)$, respectively, at frequency bin $k \in \{0,1,\ldots,K-1\}$ and time frame n. Since $x(t)$ and $v(t)$ are uncorrelated by assumption, the variance of $Y(k,n)$ is

J. Benesty et al., *Signal Enhancement with Variable Span Linear Filters*,
Springer Topics in Signal Processing 7, DOI 10.1007/978-981-287-739-0_4

$$\begin{aligned}\phi_Y(k,n) &= E\left[|Y(k,n)|^2\right] \\ &= \phi_X(k,n) + \phi_V(k,n),\end{aligned} \tag{4.3}$$

where $\phi_X(k,n) = E\left[|X(k,n)|^2\right]$ and $\phi_V(k,n) = E\left[|V(k,n)|^2\right]$ are the variances of $X(k,n)$ and $V(k,n)$, respectively.

In this chapter, the interframe correlation is taken into account in order to improve filtering since signals are usually correlated at successive time frames with the STFT [2]. Considering L of these successive frames, we can rewrite the observations as

$$\begin{aligned}\mathbf{y}(k,n) &= \left[\, Y(k,n)\ Y(k,n-1)\ \cdots\ Y(k,n-L+1)\,\right]^T \\ &= \mathbf{x}(k,n) + \mathbf{v}(k,n),\end{aligned} \tag{4.4}$$

where $\mathbf{x}(k,n)$ and $\mathbf{v}(k,n)$ resemble $\mathbf{y}(k,n)$ of length L. The correlation matrix of $\mathbf{y}(k,n)$ is then

$$\begin{aligned}\mathbf{\Phi}_{\mathbf{y}}(k,n) &= E\left[\mathbf{y}(k,n)\mathbf{y}^H(k,n)\right] \\ &= \mathbf{\Phi}_{\mathbf{x}}(k,n) + \mathbf{\Phi}_{\mathbf{v}}(k,n),\end{aligned} \tag{4.5}$$

where $\mathbf{\Phi}_{\mathbf{x}}(k,n) = E\left[\mathbf{x}(k,n)\mathbf{x}^H(k,n)\right]$ and $\mathbf{\Phi}_{\mathbf{v}}(k,n) = E\left[\mathbf{v}(k,n)\mathbf{v}^H(k,n)\right]$ are the correlation matrices of $\mathbf{x}(k,n)$ and $\mathbf{v}(k,n)$, respectively. It is assumed that $\mathbf{\Phi}_{\mathbf{v}}(k,n)$ is a full-rank matrix.

Then, the objective of single-channel noise reduction in the STFT domain is the estimation of the desired signal, $X(k,n)$, from the observation signal vector, $\mathbf{y}(k,n)$, in the best possible way.

4.2 Joint Diagonalization

For convenience, we explain again the joint diagonalization idea in the context and notation of this chapter. The two Hermitian matrices $\mathbf{\Phi}_{\mathbf{x}}(k,n)$ and $\mathbf{\Phi}_{\mathbf{v}}(k,n)$ can be jointly diagonalized as follows [3]:

$$\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{B}(k,n) = \mathbf{\Lambda}(k,n), \tag{4.6}$$

$$\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{v}}(k,n)\mathbf{B}(k,n) = \mathbf{I}_L, \tag{4.7}$$

where $\mathbf{B}(k,n)$ is a full-rank square matrix (of size $L \times L$), $\mathbf{\Lambda}(k,n)$ is a diagonal matrix whose main elements are real and nonnegative, and $\mathbf{I}_L$ is the $L \times L$ identity matrix. Furthermore, $\mathbf{\Lambda}(k,n)$ and $\mathbf{B}(k,n)$ are the eigenvalue and eigenvector matrices, respectively, of $\mathbf{\Phi}_{\mathbf{v}}^{-1}(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)$, i.e.,

$$\mathbf{\Phi}_{\mathbf{v}}^{-1}(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{B}(k,n) = \mathbf{B}(k,n)\mathbf{\Lambda}(k,n). \tag{4.8}$$

The eigenvalues of $\mathbf{\Phi}_{\mathbf{v}}^{-1}(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)$ can be ordered as $\lambda_1(k,n) \geq \lambda_2(k,n) \geq \cdots \geq \lambda_L(k,n) \geq 0$ and we denote by $\mathbf{b}_1(k,n), \mathbf{b}_2(k,n), \ldots, \mathbf{b}_L(k,n)$, the corresponding eigenvectors. The noisy signal correlation matrix can also be diagonalized as

$$\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{y}}(k,n)\mathbf{B}(k,n) = \mathbf{\Lambda}(k,n) + \mathbf{I}_L. \tag{4.9}$$

The fact that the three matrices of interest can be diagonalized in a simple and elegant way simplifies the derivation of a class of optimal filters for single-channel noise reduction in the STFT domain.

4.3 Linear Filtering

The desired signal, $X(k,n)$, is estimated from the observation signal vector, $\mathbf{y}(k,n)$, through a filtering operation, i.e.,

$$Z(k,n) = \mathbf{h}^H(k,n)\mathbf{y}(k,n), \tag{4.10}$$

where $Z(k,n)$ is the estimate of $X(k,n)$ and

$$\mathbf{h}(k,n) = \begin{bmatrix} H_1(k,n) \; H_2(k,n) \; \cdots \; H_L(k,n) \end{bmatrix}^T \tag{4.11}$$

is a complex-valued filter of length L. It is always possible to write $\mathbf{h}(k,n)$ in a basis formed from the vectors $\mathbf{b}_l(k,n),\ l = 1, 2, \ldots, L$, i.e.,

$$\mathbf{h}(k,n) = \mathbf{B}(k,n)\mathbf{a}(k,n), \tag{4.12}$$

where the components of

$$\mathbf{a}(k,n) = \begin{bmatrix} A_1(k,n) \; A_2(k,n) \; \cdots \; A_L(k,n) \end{bmatrix}^T \tag{4.13}$$

are the coordinates of $\mathbf{h}(k,n)$ in the new basis. Now, instead of estimating the coefficients of $\mathbf{h}(k,n)$ as in conventional approaches, we can estimate, equivalently, the coordinates $A_l(k,n),\ l = 1, 2, \ldots, L$. When $\mathbf{a}(k,n)$ is estimated, it is then easy to determine $\mathbf{h}(k,n)$ from (4.12). Substituting (4.12) into (4.10), we get

$$Z(k,n) = \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{x}(k,n) + \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{v}(k,n). \tag{4.14}$$

We deduce that the variance of $Z(k,n)$ is

$$\phi_Z(k,n) = \mathbf{a}^H(k,n)\mathbf{\Lambda}(k,n)\mathbf{a}(k,n) + \mathbf{a}^H(k,n)\mathbf{a}(k,n). \tag{4.15}$$

4.4 Performance Measures

In this section, we briefly define the most useful performance measures for single-channel signal enhancement in the STFT domain. We can divide these measures into two categories: noise reduction and distortion. We also discuss the MSE criterion and show how it is related to the performance measures.

4.4.1 Noise Reduction

We define the subband (at frequency bin k) and fullband input SNRs at time frame n as

$$\mathrm{iSNR}(k, n) = \frac{\phi_X(k, n)}{\phi_V(k, n)}, \tag{4.16}$$

$$\mathrm{iSNR}(n) = \frac{\sum_{k=0}^{K-1} \phi_X(k, n)}{\sum_{k=0}^{K-1} \phi_V(k, n)}. \tag{4.17}$$

From (4.15), we easily deduce the subband output SNR at frequency bin k:

$$\begin{aligned} \mathrm{oSNR}\left[\mathbf{a}(k, n)\right] &= \frac{\mathbf{a}^H(k, n)\mathbf{\Lambda}(k, n)\mathbf{a}(k, n)}{\mathbf{a}^H(k, n)\mathbf{a}(k, n)} \\ &= \frac{\sum_{l=1}^{L} \lambda_l(k, n) \left|A_l(k, n)\right|^2}{\sum_{l=1}^{L} \left|A_l(k, n)\right|^2}, \end{aligned} \tag{4.18}$$

and the fullband output SNR:

$$\mathrm{oSNR}\left[\mathbf{a}(:, n)\right] = \frac{\sum_{k=0}^{K-1} \mathbf{a}^H(k, n)\mathbf{\Lambda}(k, n)\mathbf{a}(k, n)}{\sum_{k=0}^{K-1} \mathbf{a}^H(k, n)\mathbf{a}(k, n)}. \tag{4.19}$$

The filters should be derived in such a way that $\mathrm{oSNR}\left[\mathbf{a}(k, n)\right] \geq \mathrm{iSNR}(k, n)$ and $\mathrm{oSNR}\left[\mathbf{a}(:, n)\right] \geq \mathrm{iSNR}(n)$.

The noise reduction factor quantifies the amount of noise whose is rejected by the complex filter. The subband and fullband noise reduction factors are then

$$\xi_{\mathrm{nr}}\left[\mathbf{a}(k, n)\right] = \frac{\phi_V(k, n)}{\mathbf{a}^H(k, n)\mathbf{a}(k, n)}, \tag{4.20}$$

$$\xi_{\mathrm{nr}}\left[\mathbf{a}(:, n)\right] = \frac{\sum_{k=0}^{K-1} \phi_V(k, n)}{\sum_{k=0}^{K-1} \mathbf{a}^H(k, n)\mathbf{a}(k, n)}. \tag{4.21}$$

For optimal filters, we should have $\xi_{\mathrm{nr}}\left[\mathbf{a}(k, n)\right] \geq 1$ and $\xi_{\mathrm{nr}}\left[\mathbf{a}(:, n)\right] \geq 1$.

4.4.2 Desired Signal Distortion

In practice, the complex filter distorts the desired signal. In order to evaluate the level of this distortion, we define the subband and fullband desired signal reduction factors:

$$\xi_{\text{sr}}\left[\mathbf{a}(k,n)\right] = \frac{\phi_X(k,n)}{\mathbf{a}^H(k,n)\mathbf{\Lambda}(k,n)\mathbf{a}(k,n)}, \tag{4.22}$$

$$\xi_{\text{sr}}\left[\mathbf{a}(:,n)\right] = \frac{\sum_{k=0}^{K-1}\phi_X(k,n)}{\sum_{k=0}^{K-1}\mathbf{a}^H(k,n)\mathbf{\Lambda}(k,n)\mathbf{a}(k,n)}. \tag{4.23}$$

For optimal filters, we should have $\xi_{\text{sr}}\left[\mathbf{a}(k,n)\right] \geq 1$ and $\xi_{\text{sr}}\left[\mathbf{a}(:,n)\right] \geq 1$. The larger is the value of $\xi_{\text{sr}}\left[\mathbf{a}(:,n)\right]$, the more the desired signal is distorted.

By making the appropriate substitutions, one can derive the relationships:

$$\frac{\text{oSNR}\left[\mathbf{a}(k,n)\right]}{\text{iSNR}(k,n)} = \frac{\xi_{\text{nr}}\left[\mathbf{a}(k,n)\right]}{\xi_{\text{sr}}\left[\mathbf{a}(k,n)\right]}, \tag{4.24}$$

$$\frac{\text{oSNR}\left[\mathbf{a}(:,n)\right]}{\text{iSNR}(n)} = \frac{\xi_{\text{nr}}\left[\mathbf{a}(:,n)\right]}{\xi_{\text{sr}}\left[\mathbf{a}(:,n)\right]}. \tag{4.25}$$

These expressions indicate the equivalence between gain/loss in SNR and distortion for both the subband and fullband cases.

Another way to measure the distortion of the desired signal due to the complex filter is the desired signal distortion index, which is defined as the MSE between the desired signal and the filtered desired signal, normalized by the variance of the desired signal, i.e.,

$$\upsilon_{\text{sd}}\left[\mathbf{a}(k,n)\right] = \frac{E\left\{\left|X(k,n) - \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{x}(k,n)\right|^2\right\}}{\phi_X(k,n)} \tag{4.26}$$

in the subband case and

$$\begin{aligned}\upsilon_{\text{sd}}\left[\mathbf{a}(:,n)\right] &= \frac{\sum_{k=0}^{K-1}E\left\{\left|X(k,n) - \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{x}(k,n)\right|^2\right\}}{\sum_{k=0}^{K-1}\phi_X(k,n)} \\ &= \frac{\sum_{k=0}^{K-1}\upsilon_{\text{sd}}\left[\mathbf{a}(k,n)\right]\phi_X(k,n)}{\sum_{k=0}^{K-1}\phi_X(k,n)}\end{aligned} \tag{4.27}$$

in the fullband case. The desired signal distortion indices are usually upper bounded by 1 for optimal filters.

4.4.3 MSE Criterion

In the STFT domain, the error signal between the estimated and desired signals at the frequency bin k is

$$\begin{aligned}\mathcal{E}(k,n) &= Z(k,n) - X(k,n) \\ &= \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{y}(k,n) - X(k,n),\end{aligned} \tag{4.28}$$

which can also be written as the sum of two uncorrelated error signals:

$$\mathcal{E}(k,n) = \mathcal{E}_{\text{ds}}(k,n) + \mathcal{E}_{\text{rs}}(k,n), \tag{4.29}$$

where

$$\mathcal{E}_{\text{ds}}(k,n) = \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{x}(k,n) - X(k,n) \tag{4.30}$$

is the distortion of the desired signal due to the filter and

$$\mathcal{E}_{\text{rs}}(k,n) = \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{v}(k,n) \tag{4.31}$$

represents the residual noise.

The subband MSE criterion is then

$$\begin{aligned}J\left[\mathbf{a}(k,n)\right] &= E\left[\left|\mathcal{E}(k,n)\right|^2\right] \\ &= \phi_X(k,n) - \mathbf{i}^T\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{B}(k,n)\mathbf{a}(k,n) \\ &\quad - \mathbf{a}^H(k,n)\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i} + \mathbf{a}^H(k,n)\left[\mathbf{\Lambda}(k,n) + \mathbf{I}_L\right]\mathbf{a}(k,n) \\ &= J_{\text{ds}}\left[\mathbf{a}(k,n)\right] + J_{\text{rs}}\left[\mathbf{a}(k,n)\right],\end{aligned} \tag{4.32}$$

where $\mathbf{i}$ is the first column of $\mathbf{I}_L$,

$$\begin{aligned}J_{\text{ds}}\left[\mathbf{a}(k,n)\right] &= E\left[\left|\mathcal{E}_{\text{ds}}(k,n)\right|^2\right] \\ &= \upsilon_{\text{sd}}\left[\mathbf{a}(k,n)\right]\phi_X(k,n),\end{aligned} \tag{4.33}$$

and

$$\begin{aligned}J_{\text{rs}}\left[\mathbf{a}(k,n)\right] &= E\left[\left|\mathcal{E}_{\text{rs}}(k,n)\right|^2\right] \\ &= \frac{\phi_V(k,n)}{\xi_{\text{nr}}\left[\mathbf{a}(k,n)\right]}.\end{aligned} \tag{4.34}$$

We deduce that

$$\begin{aligned}\frac{J_{\text{ds}}\left[\mathbf{a}(k,n)\right]}{J_{\text{rs}}\left[\mathbf{a}(k,n)\right]} &= \text{iSNR}(k,n) \times \xi_{\text{nr}}\left[\mathbf{a}(k,n)\right] \times \upsilon_{\text{sd}}\left[\mathbf{a}(k,n)\right] \\ &= \text{oSNR}\left[\mathbf{a}(k,n)\right] \times \xi_{\text{sr}}\left[\mathbf{a}(k,n)\right] \times \upsilon_{\text{sd}}\left[\mathbf{a}(k,n)\right].\end{aligned} \tag{4.35}$$

This shows how the different subband performances measures are related to the MSEs.

4.5 Optimal Linear Filters

In this section, we derive different optimal filters for single-channel noise reduction in the STFT domain and show how strongly they are connected thanks to the VS linear filtering concept.

4.5.1 Maximum SNR

The maximum SNR filter is obtained by maximizing the subband output SNR. It is clear that (4.18) is maximized if and only if $A_1(k,n) \neq 0$ and $A_2(k,n) = \cdots = A_L(k,n) = 0$. As a consequence, the maximum SNR filter is

$$\mathbf{h}_{\max}(k,n) = A_1(k,n)\mathbf{b}_1(k,n), \tag{4.36}$$

where $A_1(k,n) \neq 0$ is an arbitrary complex number. The optimal value of $A_1(k,n)$ is obtained by minimizing distortion. Substituting (4.36) into (4.33) and minimizing the resulting expression with respect to $A_1(k,n)$, we find the maximum SNR filter with minimum distortion:

$$\mathbf{h}_{\max}(k,n) = \frac{\mathbf{b}_1(k,n)\mathbf{b}_1^H(k,n)}{\lambda_1(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}. \tag{4.37}$$

It can be verified that

$$\text{oSNR}\left[\mathbf{h}_{\max}(k,n)\right] = \lambda_1(k,n), \tag{4.38}$$

which corresponds to the maximum subband output SNR, and

$$\text{oSNR}\left[\mathbf{h}(k,n)\right] \leq \text{oSNR}\left[\mathbf{h}_{\max}(k,n)\right], \ \forall \mathbf{h}(k,n). \tag{4.39}$$

4.5.2 Minimum Distortion

The minimum distortion (MD) filter is obtained by minimizing $J_{\text{ds}}\left[\mathbf{a}(k,n)\right]$. We get

$$\begin{aligned}\mathbf{a}_{\mathrm{MD}}(k,n) &= \left[\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{B}(k,n)\right]^{-1}\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i} \\ &= \mathbf{\Lambda}^{-1}(k,n)\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}.\end{aligned} \tag{4.40}$$

Therefore, the MD filter is

$$\begin{aligned}\mathbf{h}_{\mathrm{MD}}(k,n) &= \mathbf{B}(k,n)\mathbf{a}_{\mathrm{MD}}(k,n) \\ &= \sum_{l=1}^{L}\frac{\mathbf{b}_l(k,n)\mathbf{b}_l^H(k,n)}{\lambda_l(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i} \\ &= \mathbf{i},\end{aligned} \tag{4.41}$$

which in this problem turns out to be the identity filter. In (4.40) and (4.41), it is assumed that $\mathbf{\Phi}_{\mathbf{x}}(k,n)$ is a full-rank matrix. If it's not the case and only $P < L$ eigenvalues are strictly positive, then the MD filter becomes the MVDR filter:

$$\mathbf{h}_{\mathrm{MVDR}}(k,n) = \sum_{p=1}^{P}\frac{\mathbf{b}_p(k,n)\mathbf{b}_p^H(k,n)}{\lambda_p(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}. \tag{4.42}$$

From the obvious relationship between the maximum SNR and MD filters, we propose a class of MD filters:

$$\mathbf{h}_{\mathrm{MD},Q}(k,n) = \sum_{q=1}^{Q}\frac{\mathbf{b}_q(k,n)\mathbf{b}_q^H(k,n)}{\lambda_q(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}, \tag{4.43}$$

where $1 \leq Q \leq L$. We observe that for $Q = 1$ and $Q = L$, we obtain $\mathbf{h}_{\mathrm{MD},1}(k,n) = \mathbf{h}_{\max}(k,n)$ and $\mathbf{h}_{\mathrm{MD},L}(k,n) = \mathbf{i}$, respectively. We should have

$$\mathrm{oSNR}\left[\mathbf{h}_{\mathrm{MD},1}(k,n)\right] \geq \mathrm{oSNR}\left[\mathbf{h}_{\mathrm{MD},2}(k,n)\right] \geq \cdots \geq \mathrm{oSNR}\left[\mathbf{h}_{\mathrm{MD},L}(k,n)\right] \tag{4.44}$$

and

$$\xi_{\mathrm{sr}}\left[\mathbf{h}_{\mathrm{MD},1}(k,n)\right] \geq \xi_{\mathrm{sr}}\left[\mathbf{h}_{\mathrm{MD},2}(k,n)\right] \geq \cdots \geq \xi_{\mathrm{sr}}\left[\mathbf{h}_{\mathrm{MD},L}(k,n)\right]. \tag{4.45}$$

4.5.3 Wiener

The Wiener filter is obtained from the optimization of the MSE criterion, $J\left[\mathbf{a}(k,n)\right]$. The minimization of $J\left[\mathbf{a}(k,n)\right]$ leads to

$$\mathbf{a}_{\mathrm{W}}(k,n) = \left[\mathbf{\Lambda}(k,n) + \mathbf{I}_L\right]^{-1}\mathbf{B}^H(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}. \tag{4.46}$$

We deduce that the Wiener filter is

$$\begin{aligned}\mathbf{h}_{\mathrm{W}}(k,n) &= \mathbf{B}(k,n)\mathbf{a}_{\mathrm{W}}(k,n) \\ &= \sum_{l=1}^{L} \frac{\mathbf{b}_l(k,n)\mathbf{b}_l^H(k,n)}{1+\lambda_l(k,n)} \mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i} \\ &= \mathbf{\Phi}_{\mathbf{y}}^{-1}(k,n)\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}.\end{aligned} \tag{4.47}$$

We can see that the MD and Wiener filters are very close to each other; they only differ by the weighting function, which strongly depends on the eigenvalues of the joint diagonalization. In the first case, it is equal to $\lambda_l^{-1}(k,n)$ while in the second case it is equal to $[1+\lambda_l(k,n)]^{-1}$. The MD filter will always extract the desired signal from all directions while it will be more attenuated with the Wiener filter. We should have

$$\mathrm{oSNR}\left[\mathbf{h}_{\mathrm{W}}(k,n)\right] \geq \mathrm{oSNR}\left[\mathbf{h}_{\mathrm{MD}}(k,n)\right] \tag{4.48}$$

and

$$\xi_{\mathrm{sr}}\left[\mathbf{h}_{\mathrm{W}}(k,n)\right] \geq \xi_{\mathrm{sr}}\left[\mathbf{h}_{\mathrm{MD}}(k,n)\right]. \tag{4.49}$$

4.5.4 Tradeoff

Another interesting approach that can compromise between noise reduction and distortion is the tradeoff filter obtained by

$$\min_{\mathbf{a}(k,n)} J_{\mathrm{ds}}\left[\mathbf{a}(k,n)\right] \quad \text{subject to} \quad J_{\mathrm{rs}}\left[\mathbf{a}(k,n)\right] = \beta\phi_V(k,n), \tag{4.50}$$

where $0 \leq \beta \leq 1$, to ensure that filtering achieves some degree of noise reduction. We easily find that the optimal filter is

$$\mathbf{h}_{\mathrm{T},\mu}(k,n) = \sum_{l=1}^{L} \frac{\mathbf{b}_l(k,n)\mathbf{b}_l^H(k,n)}{\mu+\lambda_l(k,n)} \mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}, \tag{4.51}$$

where $\mu \geq 0$ is a Lagrange multiplier. Clearly, for $\mu = 0$ and $\mu = 1$, we get the MD and Wiener filters, respectively.

From all what we have seen so far, we can propose a very general tradeoff noise reduction filter:

$$\mathbf{h}_{\mu,Q}(k,n) = \sum_{q=1}^{Q} \frac{\mathbf{b}_q(k,n)\mathbf{b}_q^H(k,n)}{\mu+\lambda_q(k,n)} \mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}. \tag{4.52}$$

This form encompasses all known optimal filters. Indeed, it is clear that

- $\mathbf{h}_{0,1}(k,n) = \mathbf{h}_{\max}(k,n)$,

Table 4.1 Optimal linear filters for single-channel signal enhancement in the STFT domain.

Maximum SNR:	$\mathbf{h}_{\max}(k,n) = \dfrac{\mathbf{b}_1(k,n)\mathbf{b}_1^H(k,n)}{\lambda_1(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}$
MVDR:	$\mathbf{h}_{\mathrm{MVDR}}(k,n) = \sum_{p=1}^{P} \dfrac{\mathbf{b}_p(k,n)\mathbf{b}_p^H(k,n)}{\lambda_p(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}$
MD,Q:	$\mathbf{h}_{\mathrm{MD},Q}(k,n) = \sum_{q=1}^{Q} \dfrac{\mathbf{b}_q(k,n)\mathbf{b}_q^H(k,n)}{\lambda_q(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}$
Wiener:	$\mathbf{h}_{\mathrm{W}}(k,n) = \sum_{l=1}^{L} \dfrac{\mathbf{b}_l(k,n)\mathbf{b}_l^H(k,n)}{1+\lambda_l(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}$
Tradeoff:	$\mathbf{h}_{\mathrm{T},\mu}(k,n) = \sum_{l=1}^{L} \dfrac{\mathbf{b}_l(k,n)\mathbf{b}_l^H(k,n)}{\mu+\lambda_l(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}$
General Tradeoff:	$\mathbf{h}_{\mu,Q}(k,n) = \sum_{q=1}^{Q} \dfrac{\mathbf{b}_q(k,n)\mathbf{b}_q^H(k,n)}{\mu+\lambda_q(k,n)}\mathbf{\Phi}_{\mathbf{x}}(k,n)\mathbf{i}$

- $\mathbf{h}_{1,L}(k,n) = \mathbf{h}_{\mathrm{W}}(k,n)$,
- $\mathbf{h}_{0,L}(k,n) = \mathbf{i}$,
- $\mathbf{h}_{0,Q}(k,n) = \mathbf{h}_{\mathrm{MD},Q}(k,n)$,
- $\mathbf{h}_{\mu,L}(k,n) = \mathbf{h}_{\mathrm{T},\mu}(k,n)$.

In Table 4.1, we summarize all optimal filters derived in this section.

4.6 Experimental Results

We then proceed with the evaluation of the filters proposed in Section 4.5. For this evaluation, we considered the enhancement of different speech signals contaminated by different kinds of noise. More specifically, the speech signals considered were constituted by two female and two male speech signals. This amounted to approximately 10 seconds of female speech and 10 seconds of male speech. In Fig. 4.1, plots of the the first five seconds of the concatenated speech signals as well as their spectrograms are found. The different noise signals considered were babble, exhibition, street, and car noise originating from the AURORA database [4]. The spectrograms of the first five seconds of all the noise signals are plotted in Fig. 4.2. Using mixtures of the speech and these different noise types, we then conducted evaluations of the aforementioned filters in terms of their output SNRs and desired signal reduction factors. In each of the evaluations, enhancement of the speech signal mixed with all noise types were considered, hence, the depicted results are the performance measures averaged over time and over the different noise types.

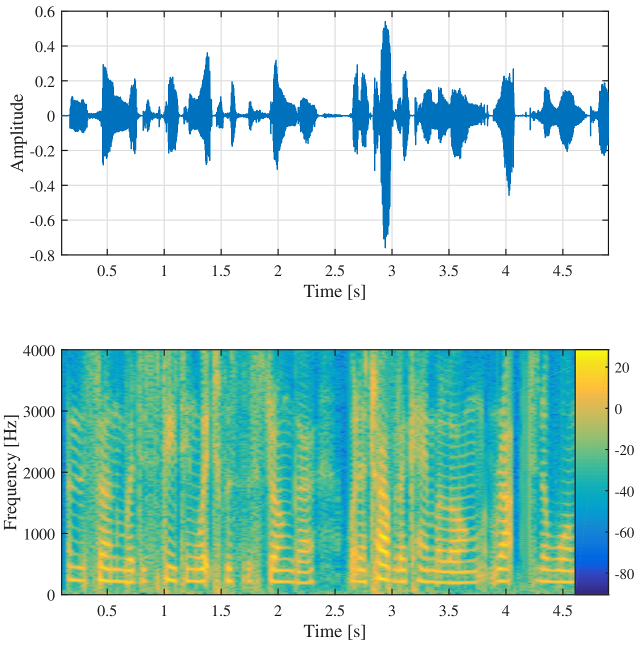


Fig. 4.1 Plot of (top) the first five seconds of the utilized speech signal and (bottom) the spectrogram of it.

As can be observed in the previous section, noise and signal statistics are needed to implement the filters in practice. These are often not readily obtained, but much research has been done on how these statistics can be obtained in practical scenarios (see, e.g., [5], [6], [7]). Noise or signal estimation in practice is not considered in this book, since our focus here is on evaluating the relative performance of the proposed filters. Therefore, the statistics needed in the filter designs were estimated by assuming access to the individual signals, e.g., the noise statistics were estimated from the noise signal. This estimation were conducted recursively as

$$\widehat{\mathbf{\Phi}}_{\mathbf{y}}(k,n) = (1-\xi_{\text{s}})\widehat{\mathbf{\Phi}}_{\mathbf{y}}(k,n-1) + \xi_{\text{s}}\mathbf{y}(k,n)\mathbf{y}^H(k,n), \tag{4.53}$$

$$\widehat{\mathbf{\Phi}}_{\mathbf{x}}(k,n) = (1-\xi_{\text{s}})\widehat{\mathbf{\Phi}}_{\mathbf{x}}(k,n-1) + \xi_{\text{s}}\mathbf{x}(k,n)\mathbf{x}^H(k,n), \tag{4.54}$$

$$\widehat{\mathbf{\Phi}}_{\mathbf{v}}(k,n) = (1-\xi_{\text{n}})\widehat{\mathbf{\Phi}}_{\mathbf{v}}(k,n-1) + \xi_{\text{n}}\mathbf{v}(k,n)\mathbf{v}^H(k,n). \tag{4.55}$$

The MATLAB code for all the evaluations as well as for auxiliary functions can be found in Section 4.A and Appendix A.

The first thing investigated in the evaluations is how to choose the forgetting factors in the recursive expressions above. To investigate this, we

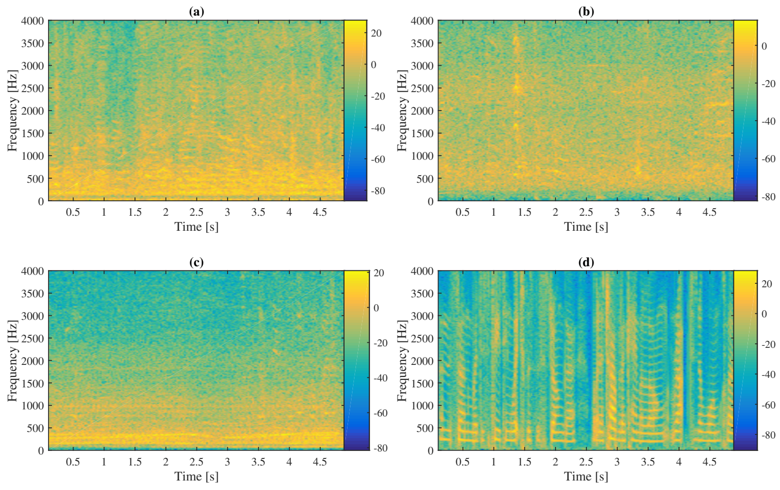
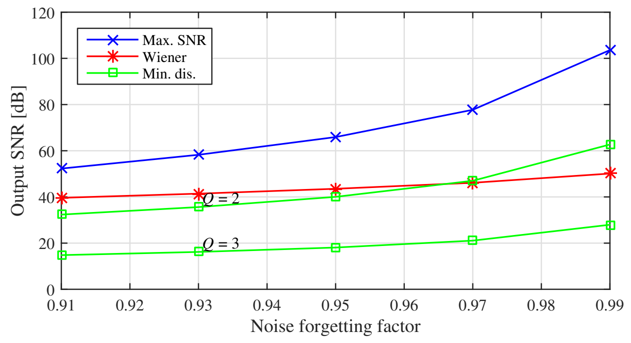


Fig. 4.2 Plot of the spectrograms of the first five seconds of the (a) babble noise, (b) exhibition noise, (c) street noise, and (d) car noise signals used for the evaluations.

considered a scenario with an input SNR of 0 dB, and the STFT of the signals were computed from time-consecutive blocks of 50 samples overlapping by 50 % using an FFT length of 64 and rectangular windows. Moreover, the temporal filter length in the STFT domain was $L = 4$. With this setup, we then first fixed the signal forgetting factor to $\xi_{\mathrm{s}} = 0.95$ and measured the filter performance versus different noise forgetting factors. The outcome is depicted in Fig. 4.3. The filters considered here are the maximum SNR, Wiener, and minimum distortion (for $Q = 2$ and $Q = 3$) filters. Regarding noise reduction, we observe that all filters have an increasing output SNR for an increasing forgetting factor. If we instead look at the desired signal distortion, we see that the filters behave a bit differently. The maximum SNR and minimum distortion filters have a slightly increasing distortion when the noise forgetting factor increases. The Wiener filter has the opposite behavior, i.e., the distortion is decreased when the forgetting factor is increased.

Then, in another series of experiments, we considered the same setup except that the noise forgetting factor was now fixed to $\xi_{\mathrm{n}} = 0.99$, while the signal forgetting factor was varied. The results are shown in Fig. 4.4. This evaluation shows that the output SNRs for the maximum SNR and minimum distortion filters decrease for an increasing signal forgetting factor, whereas it increases for the Wiener filter. The distortion, on the other hand, decreases for an increasing forgetting factor for all the evaluated filters. As a compromise, since the same signal and noise forgetting factors were used for all filters, we chose $\xi_{\mathrm{s}} = 0.95$ and $\xi_{\mathrm{n}} = 0.99$ for the remaining evaluations presented in this section.



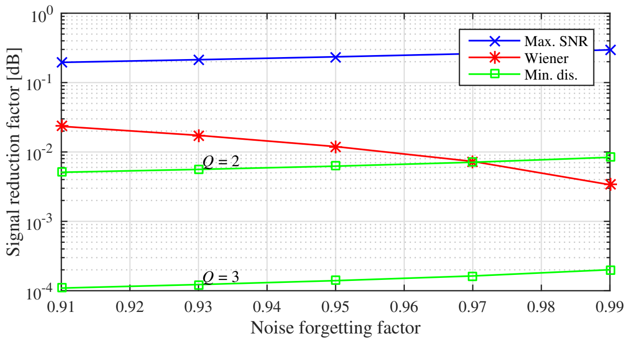


Fig. 4.3 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the noise forgetting factor, ξ_{n}.

We then proceed with the filter evaluation versus the window length used to compute the STFT. Again, the blocks were overlapping by 50 %, and the FFT length for a particular window length was set to the nearest, higher power of 2 (e.g., for a window length of 50, the FFT length was 64). Besides this, the input SNR was 0 dB, and the filter length was $L = 4$. The results depicted in Fig. 4.5 was then obtained using this simulation setup. From these results, we see that the output SNR does not change much versus the window length. From a window length of 40 to 80, the output SNRs of the minimum distortion and Wiener filters are slightly increasing for an increasing window length, but otherwise the output SNRs are almost constant. Regarding distortion, on the other hand, all filters have a decreasing distortion from a window length of 40 to a length of 80. After that, the distortion flattens out or increases slightly for all filters.

The performance versus the input SNR was then evaluated in the next series of experiments. The simulation setup for this evaluation was a follows.

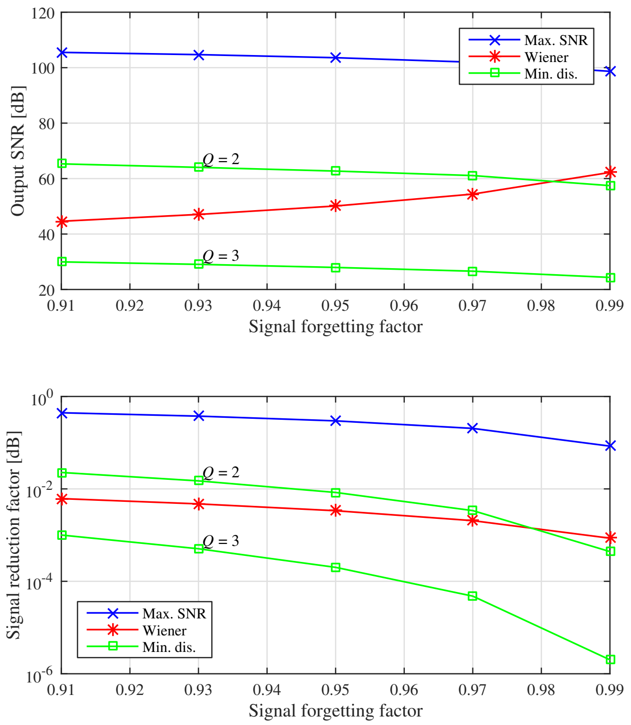


Fig. 4.4 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the signal forgetting factor, ξ_s.

The STFT was computed using windows of length 100 with an FFT length of 128. Moreover, the filter length was set to $L = 4$. With this setup, the performance of the maximum SNR, Wiener, and minimum distortion filters were then evaluated in the input SNR interval [−10 dB;10 dB], yielding the results in Fig. 4.6. From these results, we see that all filters shown an increase in output SNR when the input SNR is increased. However, if we instead consider the SNR gain, i.e., the difference between the output and input SNRs in dB, we see that this remains almost constant for the maximum SNR and minimum distortion filters, whereas it decreases for the Wiener filter. If we then inspect the distortion measurements, we see that the distortion for the maximum SNR and minimum distortion filters remains almost constant for different input SNRs. The Wiener filter, on the other hand, has less and less distortion when the input SNR increases.

We also evaluated the performance versus the filter length, L. For this simulation, the STFT window length was 50, and the STFT was comput-

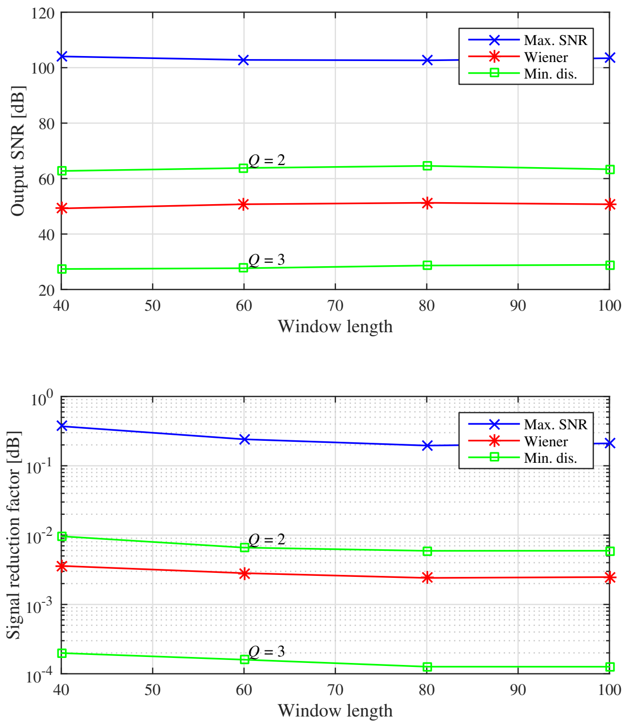


Fig. 4.5 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the STFT window length.

ing using 50 % overlapping windows with an FFT length of 64. The filters were then applied for enhancement of noisy speech signals having an input SNR of 0 dB for different filter lengths. The outcome of this experiment is depicted in Fig. 4.7. These results show that the noise reduction performance in terms of the output SNR is increasing for all filters when the filter length is increased. Moreover, the rate of increase in output SNR is greatest for the maximum SNR and minimum distortion filters. We also see that the output SNR decreases, when the assumed signal subspace rank, Q, is increased for the minimum distortion filter. Regarding the measured distortion, in terms of the signal reduction factor, we see that distortion is more or less constant for the maximum SNR filter. For a fixed Q, the distortion of the minimum distortion filters instead increases when the filter length is increased. The Wiener filter shows the opposite behavior, i.e., its distortion decreases when we decrease the filter length. In summary, for the maximum SNR and Wiener filters we should choose a large filter length, whereas the choice of filter length

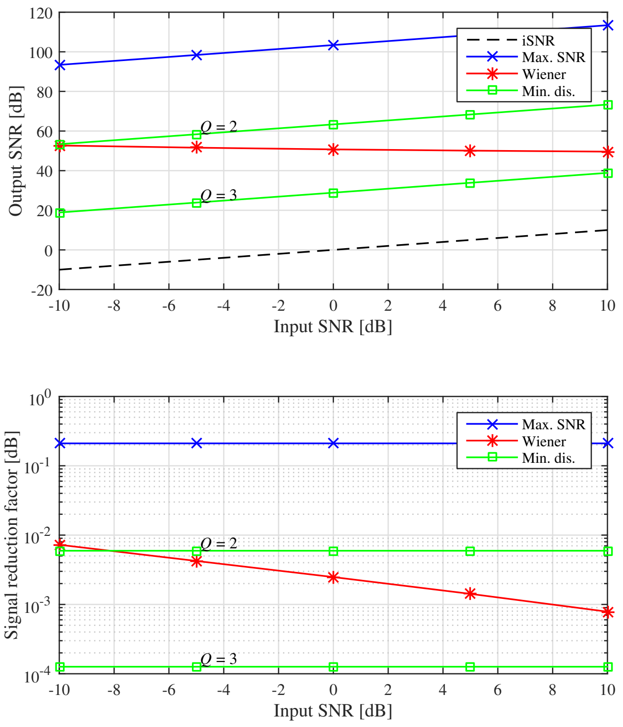


Fig. 4.6 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the input SNR.

is less obvious for the minimum distortion filter since it depends on the importance of noise reduction and signal distortion, respectively.

In the final evaluation, we considered also the tradeoff filter, and measured the performance versus different choices for the tradeoff parameter, μ. As in the previous evaluation, the STFT were computed from 50 % overlapping windows of length 50 using an FFT length of 64. The input SNR was also the same as the previous evaluation, i.e., 10 dB. Basically, the tradeoff filter can be tuned using two variables: the assumed signal rank, Q, and the tradeoff parameter, μ. In the evaluation, using the aforementioned setup, both of these tuning parameters were changed for the tradeoff filter. This resulted in the performances shown in Fig. 4.8. Generally, we observe that the output SNR of the tradeoff filter can be increased by either decreasing Q or increasing μ. Obviously, Q has the biggest impact on the noise reduction performance. Then, regarding distortion, we see that, when we change Q or μ to increase the amount of noise reduction, we get and increased signal reduction factor,

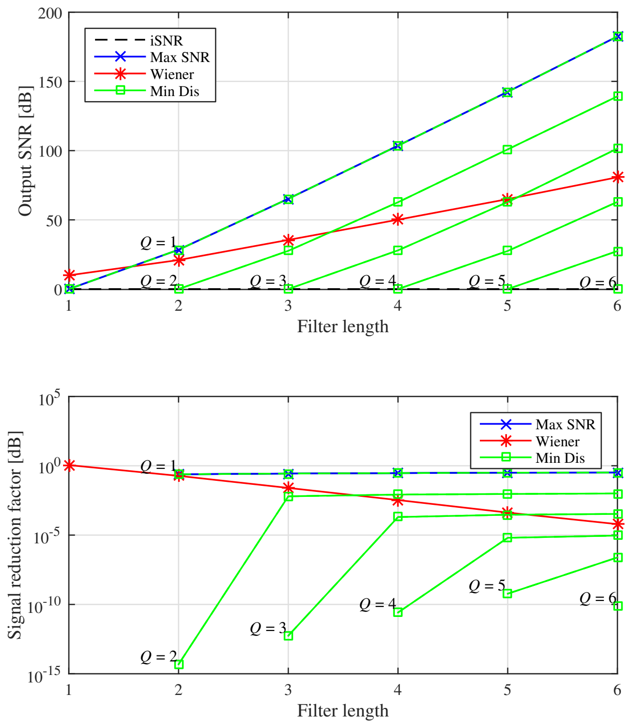


Fig. 4.7 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the filter length, L.

i.e., more distortion. Again, the biggest change in distortion is due to a change in Q. These observations clearly show that the tradeoff filter is indeed able to trade off noise reduction for signal distortion.

References

1. J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Noise Reduction in Speech Processing.* Berlin, Germany: Springer-Verlag, 2009.
2. J. Benesty, J. Chen, and E. Habets, *Speech Enhancement in the STFT Domain.* Springer Briefs in Electrical and Computer Engineering, 2011.
3. J. N. Franklin, *Matrix Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1968.
4. H.-G. Hirsch, and D. Pearce, "The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions," in *Proc. ISCA Tutorial and Research Workshop ASR2000*, 2000.

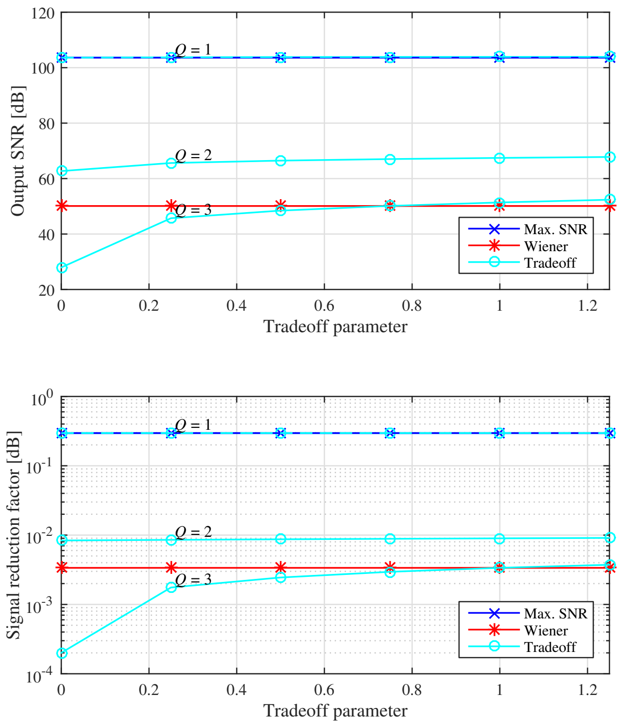


Fig. 4.8 Evaluation of the performance of the maximum SNR, Wiener, and tradeoff filters versus the tradeoff parameter, μ.

5. J. S. Sohn, N. S. Kim, and W. Sung, "A statistical model-based voice activity detection," *IEEE Signal Process. Letters*, vol. 6, pp. 1–3, Jan. 1999.
6. R. Martin, "Noise power spectral density estimation based on optimal smoothing and minimum statistics," *IEEE Trans. Speech Audio Process.*, vol. 9, pp. 504–512, Jul. 2001.
7. T. Gerkmann and R. Hendriks, "Unbiased MMSE-based noise power estimation with low complexity and low tracking delay," *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, pp. 1383–1393, May 2012.

4.A MATLAB Code

4.A.1 Main Scripts

Listing 4.1 Script for evaluating the filter performances versus the noise forgetting factor.

```matlab
clc;clear all;close all;

addpath([cd,'\..\data\']);
addpath([cd,'\..\functions\']);

nWin = 50;
nFft = 64;

iSnr = 0;

nFilt = 4;

filtSetups.minDis.signalRanks = 2:(nFilt-1);

forgetNoiGrid = 0.91:0.02:0.99;
forgetSig = 0.95;

signalString='twoMaleTwoFemale20Seconds.wav';
noiseStrings={'babble30Seconds.wav',...
    'exhibition30Seconds.wav','street30Seconds.wav'...
    ,'car30Seconds.wav'};
filtStrings={'maxSnr','minDis','wiener'};

display(['Running script: ',mfilename]);
display(' ');
for iNoise = 1:length(noiseStrings),
    noiseString = char(noiseStrings(iNoise));
    display(['Noise scenario: ',noiseString,' (',...
        num2str(iNoise),' of ',num2str(length(noiseStrings)...
        ),')']);
    display('   Enhancing...');
    for idx = 1:length(forgetNoiGrid),
        display(['      iter #: ',num2str(idx),' of ',...
            num2str(length(forgetNoiGrid))]);
        forgetNoi = forgetNoiGrid(idx);

        enhancedData = stftEnhanceSignals(signalString,noiseString,...
            iSnr,nFft,nWin,nFilt,forgetSig,forgetNoi,filtStrings,...
            filtSetups);

        performance(idx,iNoise) = stftMeasurePerformance(enhancedData,...
            filtStrings,1);

        setup(idx,iNoise) = enhancedData.setup;
    end

    display('   Measuring performance...');
    for idx = 1:length(forgetNoiGrid),
        iSnrFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).noiseReduction.iSnr.fbMean;
        oSnrMaxSnrFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).noiseReduction.oSnr.maxSnr.fbMean;
        oSnrWienerFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).noiseReduction.oSnr.wiener.fbMean;
        oSnrMinDisFbMean(:,idx,iNoise) = squeeze(...
            performance(idx,iNoise).noiseReduction.oSnr.minDis.fbMean);

        dsdMaxSnrFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).signalDistortion.dsd.maxSnr.fbMean;
        dsdWienerFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).signalDistortion.dsd.wiener.fbMean;
        dsdMinDisFbMean(:,idx,iNoise) = squeeze(...
            performance(idx,iNoise).signalDistortion.dsd.minDis.fbMean);
    end
end

```

```
%% plots
close all;

figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
semilogy(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
semilogy(10*log10(mean(dsdWienerFbMean,3)));
semilogy(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;

%% save

%dateString = datestr(now,30);
% save([mfilename,'_',dateString,'.mat']);
```

Listing 4.2 Script for evaluating the filter performances versus the signal forgetting factor.

```
clc;clear all;close all;

addpath([cd,'\..\data\']);
addpath([cd,'\..\functions\']);

nWin = 50;
nFft = 64;

iSnr = 0;

nFilt = 4;

filtSetups.minDis.signalRanks = 2:(nFilt-1);

forgetNoi = 0.99;
forgetSigGrid = 0.91:0.02:0.99;

signalString = 'twoMaleTwoFemale20Seconds.wav';
noiseStrings = {'babble30Seconds.wav','exhibition30Seconds.wav',...
    'street30Seconds.wav','car30Seconds.wav'};
filtStrings = {'maxSnr','minDis','wiener'};

display(['Running script: ',mfilename]);
display(' ');
for iNoise = 1:length(noiseStrings),
    noiseString = char(noiseStrings(iNoise));
    display(['Noise scenario: ',noiseString,' (',num2str(iNoise),...
        ' of ',num2str(length(noiseStrings)),')']);
    display('   Enhancing...');
    for idx = 1:length(forgetSigGrid),
        display(['      iter #: ',num2str(idx),' of ',num2str(...
            length(forgetSigGrid))]);
        forgetSig = forgetSigGrid(idx);

        enhancedData = stftEnhanceSignals(signalString,noiseString,...
            iSnr,nFft,nWin,nFilt,forgetSig,forgetNoi,filtStrings,...
            filtSetups);

        performance(idx,iNoise) = stftMeasurePerformance(...
            enhancedData,filtStrings,1);

        setup(idx,iNoise) = enhancedData.setup;
    end

    display('    Measuring performance...');
    for idx = 1:length(forgetSigGrid),
        iSnrFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).noiseReduction.iSnr.fbMean;
        oSnrMaxSnrFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).noiseReduction.oSnr.maxSnr.fbMean;
        oSnrWienerFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).noiseReduction.oSnr.wiener.fbMean;
        oSnrMinDisFbMean(:,idx,iNoise) = squeeze(...
            performance(idx,iNoise).noiseReduction.oSnr.minDis.fbMean);

        dsdMaxSnrFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).signalDistortion.dsd.maxSnr.fbMean;
        dsdWienerFbMean(1,idx,iNoise) = ...
            performance(idx,iNoise).signalDistortion.dsd.wiener.fbMean;
        dsdMinDisFbMean(:,idx,iNoise) = squeeze(...
            performance(idx,iNoise).signalDistortion.dsd.minDis.fbMean);
    end
end

%% plots
close all;

figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
semilogy(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
semilogy(10*log10(mean(dsdWienerFbMean,3)));
semilogy(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;

%% save

% dateString = datestr(now,30);
% save([mfilename,'_',dateString,'.mat']);
```

Listing 4.3 Script for evaluating the filter performances versus the STFT window length.

```
clc;clear all;close all;

addpath([cd,'\..\data\']);
addpath([cd,'\..\functions\']);

nWinGrid = 40:20:100;

nFilt = 4;

filtSetups.minDis.signalRanks = 2:(nFilt-1);
iSnr = 0;

forgetNoi = 0.99;
forgetSig = 0.95;

signalString = 'twoMaleTwoFemale20Seconds.wav';
```

```
noiseStrings = {'babble30Seconds.wav','exhibition30Seconds.wav',...
    'street30Seconds.wav','car30Seconds.wav'};
filtStrings = {'maxSnr','minDis','wiener'};

display(['Running script: ',mfilename]);
display(' ');
for iNoise = 1:length(noiseStrings),
    noiseString = char(noiseStrings(iNoise));
    display(['Noise scenario: ',noiseString,' (',num2str(iNoise),...
' of ',num2str(length(noiseStrings)),')']);
    display('   Enhancing...');
    for idx = 1:length(nWinGrid),
        display(['   iter #: ',num2str(idx),' of ',...
num2str(length(nWinGrid))]);
        nWin = nWinGrid(idx);
        nFft = 2^(nextpow2(nWin));

        enhancedData = stftEnhanceSignals(signalString,noiseString,...
iSnr,nFft,nWin,nFilt,forgetSig,forgetNoi,filtStrings,filtSetups);

        performance(idx,iNoise) = stftMeasurePerformance(...
enhancedData,filtStrings,1);

    end

    display('Measuring performance...');
    for idx = 1:length(nWinGrid),
        iSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.iSnr.fbMean;
        oSnrMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.maxSnr.fbMean;
        oSnrWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.wiener.fbMean;
        oSnrMinDisFbMean(:,idx,iNoise) = squeeze(...
performance(idx,iNoise).noiseReduction.oSnr.minDis.fbMean);

        dsdMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.maxSnr.fbMean;
        dsdWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.wiener.fbMean;
        dsdMinDisFbMean(:,idx,iNoise) = squeeze(...
performance(idx,iNoise).signalDistortion.dsd.minDis.fbMean);
    end
end

%% plots
close all;

figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
semilogy(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
semilogy(10*log10(mean(dsdWienerFbMean,3)));
semilogy(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;

%% save

% dateString = datestr(now,30);
%
```

```
% save([mfilename,'_',dateString,'.mat']);
```

Listing 4.4 Script for evaluating the filter performances versus the input SNR.

```
clc;clear all;close all;

addpath([cd,'\..\data\']);
addpath([cd,'\..\functions\']);

nWin = 100;
nFft = 2^nextpow2(nWin);

nFilt = 4;

filtSetups.minDis.signalRanks = 2:(nFilt-1);
iSnrGrid = -10:5:10;

forgetNoi = 0.99;
forgetSig = 0.95;

signalString = 'twoMaleTwoFemale20Seconds.wav';
noiseStrings = {'babble30Seconds.wav','exhibition30Seconds.wav',...
'street30Seconds.wav','car30Seconds.wav'};
filtStrings = {'maxSnr','minDis','wiener'};

display(['Running script: ',mfilename]);
display(' ');
for iNoise = 1:length(noiseStrings),
    noiseString = char(noiseStrings(iNoise));
    display(['Noise scenario: ',noiseString,' (',num2str(iNoise),...
' of ',num2str(length(noiseStrings)),')']);
    display('Enhancing...');
    for idx = 1:length(iSnrGrid),
        display(['   iter #: ',num2str(idx),' of ',...
num2str(length(iSnrGrid))]);
        iSnr = iSnrGrid(idx);

        enhancedData = stftEnhanceSignals(signalString,noiseString,...
iSnr,nFft,nWin,nFilt,forgetSig,forgetNoi,filtStrings,filtSetups);

        performance(idx,iNoise) = stftMeasurePerformance(...
enhancedData,filtStrings,1);

    end
end
%%
for iNoise = 1:length(noiseStrings),
    display('Measuring performance...');
    for idx = 1:length(iSnrGrid),
        iSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.iSnr.fbMean;
        oSnrMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.maxSnr.fbMean;
        oSnrWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.wiener.fbMean;
        oSnrMinDisFbMean(:,idx,iNoise) = squeeze(...
performance(idx,iNoise).noiseReduction.oSnr.minDis.fbMean);

        dsdMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.maxSnr.fbMean;
        dsdWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.wiener.fbMean;
        dsdMinDisFbMean(:,idx,iNoise) = squeeze(...
performance(idx,iNoise).signalDistortion.dsd.minDis.fbMean);
    end
```

```
end
%% plots
close all;

figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
semilogy(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
semilogy(10*log10(mean(dsdWienerFbMean,3)));
semilogy(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;

%% save

% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);
```

Listing 4.5 Script for evaluating the filter performances versus the filter length.

```
clc;clear all;close all;

addpath([cd,'\..\data\']);
addpath([cd,'\..\functions\']);

nWin = 50;
nFft = 2^(nextpow2(nWin));

nFiltGrid = 1:6;

filtSetups.minDis.signalRanks = 1;
iSnr = 0;

forgetNoi = 0.99;
forgetSig = 0.95;

signalString = 'twoMaleTwoFemale20Seconds.wav';
noiseStrings = {'babble30Seconds.wav','exhibition30Seconds.wav',...
    'street30Seconds.wav','car30Seconds.wav'};
filtStrings = {'maxSnr','minDis','wiener'};

display(['Running script: ',mfilename]);
display(' ');

oSnrMinDisFbMean = NaN(length(nFiltGrid),length(nFiltGrid),...
length(noiseStrings));
dsdMinDisFbMean = NaN(length(nFiltGrid),length(nFiltGrid),...
length(noiseStrings));
for iNoise = 1:length(noiseStrings),
    noiseString = char(noiseStrings(iNoise));
    display(['Noise scenario: ',noiseString,' (',num2str(iNoise),...
' of ',num2str(length(noiseStrings)),')']);
    display('   Enhancing...');
    for idx = 1:length(nFiltGrid),
        display(['   iter #: ',num2str(idx),' of ',...
num2str(length(nFiltGrid))]);
        nFilt = nFiltGrid(idx);
        filtSetups.minDis.signalRanks = 1:nFilt;

        enhancedData = stftEnhanceSignals(signalString,noiseString,...
iSnr,nFft,nWin,nFilt,forgetSig,forgetNoi,filtStrings,filtSetups);

        performance(idx,iNoise) = stftMeasurePerformance(...
enhancedData,filtStrings,1);

    end
    %%
    display('Measuring performance...');
    for idx = 1:length(nFiltGrid),
        iSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.iSnr.fbMean;
        oSnrMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.maxSnr.fbMean;
        oSnrWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.wiener.fbMean;

        for nn = 1:idx,
            oSnrMinDisFbMean(nn,idx,iNoise) =  ...
performance(idx,iNoise).noiseReduction.oSnr.minDis.fbMean(nn);
            dsdMinDisFbMean(nn,idx,iNoise) =  ...
performance(idx,iNoise).signalDistortion.dsd.minDis.fbMean(nn);
        end

        dsdMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.maxSnr.fbMean;
        dsdWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.wiener.fbMean;
    end
end

%% plots
close all;

figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)),'b');
plot(10*log10(mean(oSnrWienerFbMean,3)),'r');
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
plot(10*log10(mean(oSnrMaxSnrFbMean,3)),'b--');
hold off;

figure(2);
semilogy((mean(dsdMaxSnrFbMean,3)),'b');
hold on;
semilogy((mean(dsdWienerFbMean,3)),'r');
semilogy((mean(dsdMinDisFbMean,3).'),'g');
semilogy((mean(dsdMaxSnrFbMean,3)),'b--');
hold off;

%% save

% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);
```

Listing 4.6 Script for evaluating the filter performances versus the tradeoff parameter, μ.

```
clc;clear all;close all;

addpath([cd,'\..\data\']);
addpath([cd,'\..\functions\']);

nWin = 50;
nFft = 2^nextpow2(nWin);

nFilt = 4;

muGrid = 0:0.25:1.25;
filtSetups.trOff.signalRanks = 1:nFilt;
iSnr = 0;

forgetNoi = 0.99;
forgetSig = 0.95;

signalString = 'twoMaleTwoFemale20Seconds.wav';
noiseStrings = {'babble30Seconds.wav','exhibition30Seconds.wav',...
    'street30Seconds.wav','car30Seconds.wav'};
filtStrings = {'maxSnr','trOff','wiener'};

display(['Running script: ',mfilename]);
display(' ');
for iNoise = 1:length(noiseStrings),
    noiseString = char(noiseStrings(iNoise));
    display(['Noise scenario: ',noiseString,' (',num2str(iNoise),...
' of ',num2str(length(noiseStrings)),')']);
    display('Enhancing...');
    for idx = 1:length(muGrid),
        display(['   iter #: ',num2str(idx),' of ',num2str(...
length(muGrid))]);
        filtSetups.trOff.mu = muGrid(idx);

        enhancedData = stftEnhanceSignals(signalString,noiseString,...
iSnr,nFft,nWin,nFilt,forgetSig,forgetNoi,filtStrings,filtSetups);

        performance(idx,iNoise) = stftMeasurePerformance(...
enhancedData,filtStrings,1);
    end

    display('Measuring performance...');
    for idx = 1:length(muGrid),
        iSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.iSnr.fbMean;
        oSnrMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.maxSnr.fbMean;
        oSnrWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).noiseReduction.oSnr.wiener.fbMean;
        oSnrTrOffFbMean(:,idx,iNoise) = squeeze(...
performance(idx,iNoise).noiseReduction.oSnr.trOff.fbMean);

        dsdMaxSnrFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.maxSnr.fbMean;
        dsdWienerFbMean(1,idx,iNoise) = ...
performance(idx,iNoise).signalDistortion.dsd.wiener.fbMean;
        dsdTrOffFbMean(:,idx,iNoise) = squeeze(...
performance(idx,iNoise).signalDistortion.dsd.trOff.fbMean);
    end
end
%% plots
close all;

figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)),'b');
plot(10*log10(mean(oSnrWienerFbMean,3)),'r');
plot(10*log10(mean(oSnrTrOffFbMean,3).'),'g');
plot(10*log10(mean(oSnrMaxSnrFbMean,3)),'b--');
```

```
hold off;

figure(2);
semilogy(10*log10(mean(dsdMaxSnrFbMean,3)),'b');
hold on;
semilogy(10*log10(mean(dsdWienerFbMean,3)),'r');
semilogy(10*log10(mean(dsdTrOffFbMean,3).'),'g');
semilogy(10*log10(mean(dsdMaxSnrFbMean,3)),'b--');
hold off;

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);
```

4.A.2 Functions

Listing 4.7 Function for enhancing noisy signals using the single-channel, variable span filters in the STFT domain.

```
function [data] = stftEnhanceSignals(signalString,noiseString,iSnr,...
    nFft,nWin,nFilt,forgetSig,forgetNoi,filtStrings,filtSetups)

[signal,freqSamp] = audioread(signalString);

[noise,freqSampNoise] = audioread(noiseString);

if freqSamp~=freqSampNoise,
    error('The signal and noise are not sampled at the same frequency.');
end
if length(noise)<length(signal),
    error('Signal vector is longer than noise vector.');
end

noise = noise(1:length(signal));

% change noise power to get desired snr
sigPow = var(signal);
noiPow = sigPow/10^(iSnr/10);
noise = sqrt(noiPow)*noise/std(noise);

% generate observation
observed = signal + noise;

% apply stft
[data.raw.sigStft,freqGrid,timeGrid,data.raw.sigBlocks] = stftBatch(...
signal,nWin,nFft,freqSamp);
[data.raw.noiStft,~,~,data.raw.noiBlocks] = stftBatch(noise,nWin,...
nFft,freqSamp);
[data.raw.obsStft,~,~,data.raw.obsBlocks] = stftBatch(observed,nWin,...
nFft,freqSamp);

[nFreqs,nFrames] = size(data.raw.sigStft);

% inits
noiCorr = repmat(eye(nFilt),1,1,nFreqs);
obsCorr = repmat(eye(nFilt),1,1,nFreqs);
sigCorr = repmat(eye(nFilt),1,1,nFreqs);
geigVec = zeros(nFilt,nFilt,nFreqs);
```

```
geigVal = zeros(nFilt,nFilt,nFreqs);
for iFiltStrs = 1:length(filtStrings),
    switch char(filtStrings(iFiltStrs)),
        case 'maxSnr',
        hMaxSnr = zeros(nFilt,nFreqs);
        data.maxSnr.sigStft = zeros(nFreqs,nFrames);
        data.maxSnr.noiStft = zeros(nFreqs,nFrames);
        data.maxSnr.obsStft = zeros(nFreqs,nFrames);
        case 'wiener',
        hWiener = zeros(nFilt,nFreqs);
        data.wiener.sigStft = zeros(nFreqs,nFrames);
        data.wiener.noiStft = zeros(nFreqs,nFrames);
        data.wiener.obsStft = zeros(nFreqs,nFrames);
        case 'minDis',
        hMinDis = zeros(nFilt,nFreqs,length(...
filtSetups.minDis.signalRanks));
        data.minDis.sigStft = zeros(nFreqs,nFrames,length(...
filtSetups.minDis.signalRanks));
        data.minDis.noiStft = zeros(nFreqs,nFrames,length(...
filtSetups.minDis.signalRanks));
        data.minDis.obsStft = zeros(nFreqs,nFrames,length(...
filtSetups.minDis.signalRanks));
        case 'trOff',
        hTrOff = zeros(nFilt,nFreqs,length(filtSetups.trOff.signalRanks));
        data.trOff.sigStft = zeros(nFreqs,nFrames,length(...
filtSetups.trOff.signalRanks));
        data.trOff.noiStft = zeros(nFreqs,nFrames,length(...
filtSetups.trOff.signalRanks));
        data.trOff.obsStft = zeros(nFreqs,nFrames,length(...
filtSetups.trOff.signalRanks));
    end
end
for iFrame=1:nFrames,
    for iFreq=1:nFreqs,
        % extract blocks from signal, noise and observation
        if iFrame<nFilt,
            noiBlock = [data.raw.noiStft(iFreq,iFrame:-1:1).';...
zeros(nFilt-iFrame,1)];
            sigBlock = [data.raw.sigStft(iFreq,iFrame:-1:1).';...
zeros(nFilt-iFrame,1)];
            obsBlock = [data.raw.obsStft(iFreq,iFrame:-1:1).';...
zeros(nFilt-iFrame,1)];
        else
            noiBlock = data.raw.noiStft(iFreq,iFrame:-1:iFrame-nFilt+1).';
            sigBlock = data.raw.sigStft(iFreq,iFrame:-1:iFrame-nFilt+1).';
            obsBlock = data.raw.obsStft(iFreq,iFrame:-1:iFrame-nFilt+1).';
        end

        % estimate statistics
        noiCorr(:,:,iFreq) = (1-forgetNoi)*noiCorr(:,:,iFreq) + ...
(forgetNoi)*(noiBlock*noiBlock');
        obsCorr(:,:,iFreq) = (1-forgetSig)*obsCorr(:,:,iFreq) + ...
(forgetSig)*(obsBlock*obsBlock');
        sigCorr(:,:,iFreq) = (1-forgetSig)*sigCorr(:,:,iFreq) + ...
(forgetSig)*(sigBlock*sigBlock');

        for iFilt = 1:nFilt,
            noiCorr(iFilt,iFilt,iFreq) = real(noiCorr(iFilt,iFilt,iFreq));
            obsCorr(iFilt,iFilt,iFreq) = real(obsCorr(iFilt,iFilt,iFreq));
            sigCorr(iFilt,iFilt,iFreq) = real(sigCorr(iFilt,iFilt,iFreq));
        end

        % save power
        data.raw.sigPow(iFreq,iFrame) = real(sigCorr(1,1,iFreq));
        data.raw.noiPow(iFreq,iFrame) = real(noiCorr(1,1,iFreq));
        data.raw.obsPow(iFreq,iFrame) = real(obsCorr(1,1,iFreq));

```

```matlab
        % joint diagonalization
        [geigVec(:,:,iFreq),geigVal(:,:,iFreq)] = jeig(...
sigCorr(:,:,iFreq),...
            noiCorr(:,:,iFreq),1);

        for iFiltStr=1:length(filtStrings),
            switch char(filtStrings(iFiltStr)),
                case 'maxSnr',
                % max snr filt
                hMaxSnr(:,iFreq) = (geigVec(:,1,iFreq)*...
geigVec(:,1,iFreq)')/geigVal(1,1,iFreq)*sigCorr(:,1,iFreq);
                if norm(hMaxSnr(:,iFreq))==0,
                    hMaxSnr(:,iFreq) = eye(nFilt,1);
                end
                data.maxSnr.sigStft(iFreq,iFrame) = hMaxSnr(:,iFreq)'*...
sigBlock;
                data.maxSnr.noiStft(iFreq,iFrame) = hMaxSnr(:,iFreq)'*...
noiBlock;
                data.maxSnr.obsStft(iFreq,iFrame) = hMaxSnr(:,iFreq)'*...
obsBlock;
                data.maxSnr.sigPow(iFreq,iFrame) = real(...
hMaxSnr(:,iFreq)'*sigCorr(:,:,iFreq)*hMaxSnr(:,iFreq));
                data.maxSnr.noiPow(iFreq,iFrame) = real(...
hMaxSnr(:,iFreq)'*noiCorr(:,:,iFreq)*hMaxSnr(:,iFreq));
                data.maxSnr.obsPow(iFreq,iFrame) = real(...
hMaxSnr(:,iFreq)'*obsCorr(:,:,iFreq)*hMaxSnr(:,iFreq));

                case 'wiener',
                % wiener filt
                blbSubW = zeros(nFilt);
                for iFilt = 1:nFilt,
                    blbSubW = blbSubW + (geigVec(:,iFilt,iFreq)*...
geigVec(:,iFilt,iFreq)')...
                        /(1+geigVal(iFilt,iFilt,iFreq));
                end
                hWiener(:,iFreq) = blbSubW*sigCorr(:,1,iFreq);
                if norm(hWiener(:,iFreq))==0,
                    hWiener(:,iFreq) = eye(nFilt,1);
                end
                data.wiener.sigStft(iFreq,iFrame) = ...
hWiener(:,iFreq)'*sigBlock;
                data.wiener.noiStft(iFreq,iFrame) = ...
hWiener(:,iFreq)'*noiBlock;
                data.wiener.obsStft(iFreq,iFrame) = ...
hWiener(:,iFreq)'*obsBlock;
                data.wiener.sigPow(iFreq,iFrame) = real(...
hWiener(:,iFreq)'*sigCorr(:,:,iFreq)*hWiener(:,iFreq));
                data.wiener.noiPow(iFreq,iFrame) = real(...
hWiener(:,iFreq)'*noiCorr(:,:,iFreq)*hWiener(:,iFreq));
                data.wiener.obsPow(iFreq,iFrame) = real(...
hWiener(:,iFreq)'*obsCorr(:,:,iFreq)*hWiener(:,iFreq));

                case 'minDis',
                % minimum dist filt
                blbSub = zeros(nFilt);
                iterRanks = 1;
                for iRanks = 1:max(filtSetups.minDis.signalRanks),
                    blbSub = blbSub + (geigVec(:,iRanks,iFreq)*...
geigVec(:,iRanks,iFreq)')/geigVal(iRanks,iRanks,iFreq);
                    if sum(iRanks==filtSetups.minDis.signalRanks),
                        hMinDis(:,iFreq,iterRanks) = blbSub*...
sigCorr(:,1,iFreq);
                        if norm(hMinDis(:,iFreq,iterRanks))==0,
                            hMinDis(:,iFreq,iterRanks) = eye(nFilt,1);
                        end
                        iterRanks = iterRanks + 1;
                    end
                end
                for iRanks=1:length(filtSetups.minDis.signalRanks),
                    data.minDis.sigStft(iFreq,iFrame,iRanks) = ...
hMinDis(:,iFreq,iRanks)'*sigBlock;
                    data.minDis.noiStft(iFreq,iFrame,iRanks) = ...
hMinDis(:,iFreq,iRanks)'*noiBlock;
                    data.minDis.obsStft(iFreq,iFrame,iRanks) = ...
hMinDis(:,iFreq,iRanks)'*obsBlock;
                    data.minDis.sigPow(iFreq,iFrame,iRanks) = real(...
hMinDis(:,iFreq,iRanks)'*sigCorr(:,:,iFreq)*hMinDis(:,iFreq,iRanks));
                    data.minDis.noiPow(iFreq,iFrame,iRanks) = real(...
hMinDis(:,iFreq,iRanks)'*noiCorr(:,:,iFreq)*hMinDis(:,iFreq,iRanks));
                    data.minDis.obsPow(iFreq,iFrame,iRanks) = real(...
hMinDis(:,iFreq,iRanks)'*obsCorr(:,:,iFreq)*hMinDis(:,iFreq,iRanks));
                end

                case 'trOff',
                % trade off filt
                blbSub = zeros(nFilt);
                iterRanks = 1;
                for iRanks = 1:max(filtSetups.trOff.signalRanks),
                    blbSub = blbSub + (geigVec(:,iRanks,iFreq)*...
geigVec(:,iRanks,iFreq)')/(filtSetups.trOff.mu+geigVal(iRanks,...
iRanks,iFreq));
                    if sum(iRanks==filtSetups.trOff.signalRanks),
                        hTrOff(:,iFreq,iterRanks) = blbSub*...
sigCorr(:,1,iFreq);
                        if norm(hTrOff(:,iFreq,iterRanks))==0,
                            hTrOff(:,iFreq,iterRanks) = eye(nFilt,1);
                        end
                        iterRanks = iterRanks + 1;
                    end
                end
                for iRanks=1:length(filtSetups.trOff.signalRanks),
                    data.trOff.sigStft(iFreq,iFrame,iRanks) = ...
hTrOff(:,iFreq,iRanks)'*sigBlock;
                    data.trOff.noiStft(iFreq,iFrame,iRanks) = ...
hTrOff(:,iFreq,iRanks)'*noiBlock;
                    data.trOff.obsStft(iFreq,iFrame,iRanks) = ...
hTrOff(:,iFreq,iRanks)'*obsBlock;
                    data.trOff.sigPow(iFreq,iFrame,iRanks) = real(...
hTrOff(:,iFreq,iRanks)'*sigCorr(:,:,iFreq)*hTrOff(:,iFreq,iRanks));
                    data.trOff.noiPow(iFreq,iFrame,iRanks) = real(...
hTrOff(:,iFreq,iRanks)'*noiCorr(:,:,iFreq)*hTrOff(:,iFreq,iRanks));
                    data.trOff.obsPow(iFreq,iFrame,iRanks) = real(...
hTrOff(:,iFreq,iRanks)'*obsCorr(:,:,iFreq)*hTrOff(:,iFreq,iRanks));
                end
            end
        end
    end
end

for iFiltStr=1:length(filtStrings),
    switch char(filtStrings(iFiltStr)),
        case 'maxSnr',
        % stft to time
        data.maxSnr.sig = stftInvBatch(data.maxSnr.sigStft,nWin,nFft);
        data.maxSnr.noi = stftInvBatch(data.maxSnr.noiStft,nWin,nFft);
        data.maxSnr.obs = stftInvBatch(data.maxSnr.obsStft,nWin,nFft);

        case 'wiener',
        data.wiener.sig = stftInvBatch(data.wiener.sigStft,nWin,nFft);
        data.wiener.noi = stftInvBatch(data.wiener.noiStft,nWin,nFft);
        data.wiener.obs = stftInvBatch(data.wiener.obsStft,nWin,nFft);

        case 'minDis',
        for iRanks=1:length(filtSetups.minDis.signalRanks),
            data.minDis.sig(:,iRanks) = stftInvBatch(...
data.minDis.sigStft(:,:,iRanks),nWin,nFft);
            data.minDis.noi(:,iRanks) = stftInvBatch(...
data.minDis.noiStft(:,:,iRanks),nWin,nFft);
            data.minDis.obs(:,iRanks) = stftInvBatch(...
data.minDis.obsStft(:,:,iRanks),nWin,nFft);
        end
        % save to struct
        data.minDis.signalRanks = filtSetups.minDis.signalRanks;

        case 'trOff',
        for iRanks=1:length(filtSetups.trOff.signalRanks),
            data.trOff.sig(:,iRanks) = stftInvBatch(...
data.trOff.sigStft(:,:,iRanks),nWin,nFft);
            data.trOff.noi(:,iRanks) = stftInvBatch(...
data.trOff.noiStft(:,:,iRanks),nWin,nFft);
            data.trOff.obs(:,iRanks) = stftInvBatch(...
data.trOff.obsStft(:,:,iRanks),nWin,nFft);
        end
        % save to struct
        data.trOff.signalRanks = filtSetups.trOff.signalRanks;
    end
end

% save setup
data.setup.stftFreqGrid = freqGrid;
data.setup.stftTimeGrid = timeGrid;
data.setup.freqSamp = freqSamp;
data.setup.nWin = nWin;
data.setup.nFft = nFft;

% save raw signals
data.raw.sig = signal;
data.raw.noi = noise;
data.raw.obs = observed;
```

Listing 4.8 Function for measuring the performance of single-channel, variable span filters in the STFT domain.

```
function [performance] = stftMeasurePerformance(data,filtStrings,...
    flagFromSignals)

[nFreqs,nFrames] = size(data.raw.sigStft);
nWin = data.setup.nWin;
nBlockSkip = 5;

if flagFromSignals,

    % raw signal powers
    [performance.power.raw.sigPowNb,performance.power.raw.sigPowNbMean,...
        performance.power.raw.sigPowFb,...
        performance.power.raw.sigPowFbMean] = ...
        calculatePowers(data.raw.sigStft,nFreqs,nWin,nBlockSkip);

    % raw noise powers
    [performance.power.raw.noiPowNb,performance.power.raw.noiPowNbMean,...
        performance.power.raw.noiPowFb,...
        performance.power.raw.noiPowFbMean] = ...
        calculatePowers(data.raw.noiStft,nFreqs,nWin,nBlockSkip);

    [performance.noiseReduction.iSnr.nb,...
        performance.noiseReduction.iSnr.fb,...
        performance.noiseReduction.iSnr.nbMean, ...
        performance.noiseReduction.iSnr.fbMean] ...
```

```
            = measurePerformance(performance,'raw');

        for iFiltStr=1:length(filtStrings),
            switch char(filtStrings(iFiltStr)),
                case 'maxSnr',
                % signal and noise powers (max snr)
                [performance.power.maxSnr.sigPowNb,...
                    performance.power.maxSnr.sigPowNbMean,...
                    performance.power.maxSnr.sigPowFb,...
                    performance.power.maxSnr.sigPowFbMean] = ...
                    calculatePowers(data.maxSnr.sigStft,nFreqs,nWin,...
                    nBlockSkip);

                [performance.power.maxSnr.noiPowNb,...
                    performance.power.maxSnr.noiPowNbMean,...
                    performance.power.maxSnr.noiPowFb,...
                    performance.power.maxSnr.noiPowFbMean] = ...
                    calculatePowers(data.maxSnr.noiStft,nFreqs,nWin,...
                    nBlockSkip);

                [performance.noiseReduction.oSnr.maxSnr.nb,...
                    performance.noiseReduction.oSnr.maxSnr.fb,...
                    performance.noiseReduction.oSnr.maxSnr.nbMean,...
                    performance.noiseReduction.oSnr.maxSnr.fbMean,...
                    performance.signalDistortion.dsd.maxSnr.nb,...
                    performance.signalDistortion.dsd.maxSnr.fb,...
                    performance.signalDistortion.dsd.maxSnr.nbMean,...
                    performance.signalDistortion.dsd.maxSnr.fbMean]...
                    = measurePerformance(performance,char(...
                    filtStrings(iFiltStr)));

                case 'wiener',
                [performance.power.wiener.sigPowNb,...
                    performance.power.wiener.sigPowNbMean,...
                    performance.power.wiener.sigPowFb,...
                    performance.power.wiener.sigPowFbMean] = ...
                    calculatePowers(data.wiener.sigStft,nFreqs,nWin,...
                    nBlockSkip);

                [performance.power.wiener.noiPowNb,...
                    performance.power.wiener.noiPowNbMean,...
                    performance.power.wiener.noiPowFb,...
                    performance.power.wiener.noiPowFbMean] = ...
                    calculatePowers(data.wiener.noiStft,nFreqs,nWin,...
                    nBlockSkip);

                [performance.noiseReduction.oSnr.wiener.nb,...
                    performance.noiseReduction.oSnr.wiener.fb,...
                    performance.noiseReduction.oSnr.wiener.nbMean,...
                    performance.noiseReduction.oSnr.wiener.fbMean,...
                    performance.signalDistortion.dsd.wiener.nb,...
                    performance.signalDistortion.dsd.wiener.fb,...
                    performance.signalDistortion.dsd.wiener.nbMean,...
                    performance.signalDistortion.dsd.wiener.fbMean]...
                    = measurePerformance(performance,char(...
                    filtStrings(iFiltStr)));
                case 'minDis',
                    for iRank = 1:size(data.minDis.sigStft,3),
                        [performance.power.minDis.sigPowNb(:,:,iRank),...
                        performance.power.minDis.sigPowNbMean(:,:,iRank),...
                        performance.power.minDis.sigPowFb(:,:,iRank),...
                        performance.power.minDis.sigPowFbMean(:,:,iRank)] ...
                        = calculatePowers(data.minDis.sigStft(:,:,iRank),...
                        nFreqs,nWin,nBlockSkip);

                        [performance.power.minDis.noiPowNb(:,:,iRank), ...
                        performance.power.minDis.noiPowNbMean(:,:,iRank), ...
                        performance.power.minDis.noiPowFb(:,:,iRank), ...
                        performance.power.minDis.noiPowFbMean(:,:,iRank)] =...
                        calculatePowers(data.minDis.noiStft(:,:,iRank),...
                        nFreqs,nWin,nBlockSkip);

                [performance.noiseReduction.oSnr.minDis.nb(:,:,iRank),...
                performance.noiseReduction.oSnr.minDis.fb(:,:,iRank),...
                performance.noiseReduction.oSnr.minDis.nbMean(:,:,iRank),...
                performance.noiseReduction.oSnr.minDis.fbMean(:,:,iRank),...
                performance.signalDistortion.dsd.minDis.nb(:,:,iRank),...
                performance.signalDistortion.dsd.minDis.fb(:,:,iRank),...
                performance.signalDistortion.dsd.minDis.nbMean(:,:,iRank),...
                performance.signalDistortion.dsd.minDis.fbMean(:,:,iRank)]...
                = measurePerformance(performance,char(...
                filtStrings(iFiltStr)),iRank);
                    end
                case 'trOff',
                    for iRank = 1:size(data.trOff.sigStft,3),
                        [performance.power.trOff.sigPowNb(:,:,iRank),...
                        performance.power.trOff.sigPowNbMean(:,:,iRank),...
                        performance.power.trOff.sigPowFb(:,:,iRank),...
                        performance.power.trOff.sigPowFbMean(:,:,iRank)] = ...
                        calculatePowers(data.trOff.sigStft(:,:,iRank),...
                        nFreqs,nWin,nBlockSkip);

                        [performance.power.trOff.noiPowNb(:,:,iRank), ...
                        performance.power.trOff.noiPowNbMean(:,:,iRank), ...
                        performance.power.trOff.noiPowFb(:,:,iRank), ...
                        performance.power.trOff.noiPowFbMean(:,:,iRank)] = ...
                        calculatePowers(data.trOff.noiStft(:,:,iRank),...
                        nFreqs,nWin,nBlockSkip);

                [performance.noiseReduction.oSnr.trOff.nb(:,:,iRank),...
                performance.noiseReduction.oSnr.trOff.fb(:,:,iRank),...
                performance.noiseReduction.oSnr.trOff.nbMean(:,:,iRank),...
                performance.noiseReduction.oSnr.trOff.fbMean(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.nb(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.fb(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.nbMean(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.fbMean(:,:,iRank)]...
                = measurePerformance(performance,char(...
                filtStrings(iFiltStr)),iRank);
                    end
            end
        end
    end
end
%%
function [powNb,powNbMean,powFb,powFbMean] = calculatePowers(...
    stftData,fftLen,winLen,nSkip)

powNb = abs([stftData(:,:);conj(flipud(stftData(2:end-1,:)))]).^2...
/fftLen/winLen;
powNbMean = mean(powNb(:,nSkip+1:end),2);
powFb = sum(powNb,1);
powFbMean = mean(powFb(1,nSkip+1:end));

end

function [snrNb,snrFb,snrNbMean,snrFbMean,dsdNb,dsdFb,dsdNbMean...
    ,dsdFbMean] = measurePerformance(performance,filtStr,iRank)

if nargin < 3,
    iRank = 1;
end

snrNb = eval(['performance.power.',filtStr,'.sigPowNb(:,:,iRank)'])...
```

```
    ./eval(['performance.power.',filtStr,'.noiPowNb(:,:,iRank)']);
snrFb = eval(['performance.power.',filtStr,'.sigPowFb(:,:,iRank)'])...
    ./eval(['performance.power.',filtStr,'.noiPowFb(:,:,iRank)']);

snrNbMean = eval(['performance.power.',filtStr,...
    '.sigPowNbMean(:,:,iRank)'])./eval(['performance.power.',...
    filtStr,'.noiPowNbMean(:,:,iRank)']);
snrFbMean = eval(['performance.power.',filtStr,...
    '.sigPowFbMean(:,:,iRank)'])./eval(['performance.power.',...
    filtStr,'.noiPowFbMean(:,:,iRank)']);

dsdNb = performance.power.raw.sigPowNb...
    ./eval(['performance.power.',filtStr,'.sigPowNb(:,:,iRank)']);
dsdFb = performance.power.raw.sigPowFb...
    ./eval(['performance.power.',filtStr,'.sigPowFb(:,:,iRank)']);

dsdNbMean = performance.power.raw.sigPowNbMean...
    ./eval(['performance.power.',filtStr,'.sigPowNbMean(:,:,iRank)']);
dsdFbMean = performance.power.raw.sigPowFbMean...
    ./eval(['performance.power.',filtStr,'.sigPowFbMean(:,:,iRank)']);

end
```

Chapter 5
Multichannel Signal Enhancement in the Time Domain

After studying the single-channel signal enhancement problem in the STFT domain, we now propose to study the multichannel case, where the spatial information is taken into account, but in the time domain. We show how to exploit the ideas of variable span (VS) linear filtering to derive different kind of optimal filtering matrices.

5.1 Signal Model and Problem Formulation

We consider the conventional signal model in which an array with M sensors captures a convolved source signal in some noise field. The received signals, at the discrete-time index t, are expressed as [1], [2]

$$\begin{aligned} y_m(t) &= g_m(t) * s(t) + v_m(t) \\ &= x_m(t) + v_m(t),\ m = 1, 2, \ldots, M, \end{aligned} \tag{5.1}$$

where $g_m(t)$ is the acoustic impulse response from the unknown desired source, $s(t)$, location to the mth sensor, $*$ stands for linear convolution, and $v_m(t)$ is the additive noise at sensor m. We assume that the signals $x_m(t) = g_m(t) * s(t)$ and $v_m(t)$ are uncorrelated, zero mean, stationary, real, and broadband. By definition, the terms $x_m(t),\ m = 1, 2, \ldots, M$ are coherent across the array while the noise signals, $v_m(t),\ m = 1, 2, \ldots, M$, are typically only partially coherent across the array. We assume that the sensor signals are aligned so that the array looks in the direction of the source, which is considered to be known or can be estimated. This preprocessing step is not required but it may be needed in practice in order to avoid using very large filtering matrices.

By processing the data by blocks of L samples, the signal model given in (5.1) can be put into a vector form as

J. Benesty et al., *Signal Enhancement with Variable Span Linear Filters*,
Springer Topics in Signal Processing 7, DOI 10.1007/978-981-287-739-0_5

$$\mathbf{y}_m(t) = \mathbf{x}_m(t) + \mathbf{v}_m(t),\ m = 1, 2, \ldots, M, \tag{5.2}$$

where

$$\mathbf{y}_m(t) = \left[y_m(t)\ y_m(t-1) \cdots\ y_m(t-L+1) \right]^T \tag{5.3}$$

is a vector of length L, and $\mathbf{x}_m(t)$ and $\mathbf{v}_m(t)$ are defined similarly to $\mathbf{y}_m(t)$ from (5.3). It is more convenient to concatenate the M vectors $\mathbf{y}_m(t),\ m = 1, 2, \ldots, M$ together as

$$\begin{aligned} \underline{\mathbf{y}}(t) &= \left[\mathbf{y}_1^T(t)\ \mathbf{y}_2^T(t) \cdots\ \mathbf{y}_M^T(t) \right]^T \\ &= \underline{\mathbf{x}}(t) + \underline{\mathbf{v}}(t), \end{aligned} \tag{5.4}$$

where vectors $\underline{\mathbf{x}}(t)$ and $\underline{\mathbf{v}}(t)$ of length ML are defined in a similar way to $\underline{\mathbf{y}}(t)$. Since $x_m(t)$ and $v_m(t)$ are uncorrelated by assumption, the correlation matrix (of size $ML \times ML$) of the sensor signals is

$$\begin{aligned} \mathbf{R}_{\underline{\mathbf{y}}} &= E\left[\underline{\mathbf{y}}(t)\underline{\mathbf{y}}^T(t) \right] \\ &= \mathbf{R}_{\underline{\mathbf{x}}} + \mathbf{R}_{\underline{\mathbf{v}}}, \end{aligned} \tag{5.5}$$

where $\mathbf{R}_{\underline{\mathbf{x}}} = E\left[\underline{\mathbf{x}}(t)\underline{\mathbf{x}}^T(t) \right]$ and $\mathbf{R}_{\underline{\mathbf{v}}} = E\left[\underline{\mathbf{v}}(t)\underline{\mathbf{v}}^T(t) \right]$ are the correlation matrices of $\underline{\mathbf{x}}(t)$ and $\underline{\mathbf{v}}(t)$, respectively.

In this work, our desired signal vector is designated by the clean (but convolved) source signal samples received at sensor 1, namely $\mathbf{x}_1(t)$. Obviously, any vector $\mathbf{x}_m(t)$ could be taken as the reference. Our problem then may be stated as follows: given M mixtures of two uncorrelated signal vectors $\mathbf{x}_m(t)$ and $\mathbf{v}_m(t)$, our aim is to preserve $\mathbf{x}_1(t)$ while minimizing the contribution of the noise signal vectors, $\mathbf{v}_m(t),\ m = 1, 2, \ldots, M$, at the array output.

Since $\mathbf{x}_1(t)$ is the desired signal vector, $\underline{\mathbf{x}}(t)$ needs to be written as a function of $\mathbf{x}_1(t)$. Indeed, by decomposing $\underline{\mathbf{x}}(t)$ into two orthogonal components, one proportional to the desired signal vector, $\mathbf{x}_1(t)$, and the other corresponding to the interference, we get [3]

$$\begin{aligned} \underline{\mathbf{x}}(t) &= \mathbf{R}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}^{-1}\mathbf{x}_1(t) + \underline{\mathbf{x}}_{\mathrm{i}}(t) \\ &= \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{x}_1(t) + \underline{\mathbf{x}}_{\mathrm{i}}(t) \\ &= \underline{\mathbf{x}}'(t) + \underline{\mathbf{x}}_{\mathrm{i}}(t), \end{aligned} \tag{5.6}$$

where

$$\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} = \mathbf{R}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}^{-1} \tag{5.7}$$

is the time-domain steering matrix, $\mathbf{R}_{\underline{\mathbf{x}}\mathbf{x}_1} = E\left[\underline{\mathbf{x}}(t)\mathbf{x}_1^T(t) \right]$ is the cross-correlation matrix (of size $ML \times L$) between $\underline{\mathbf{x}}(t)$ and $\mathbf{x}_1(t)$, $\mathbf{R}_{\mathbf{x}_1} = E\left[\mathbf{x}_1(t)\mathbf{x}_1^T(t) \right]$ is the correlation matrix (of size $L \times L$) of $\mathbf{x}_1(t)$, $\underline{\mathbf{x}}'(t) = \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{x}_1(t)$, and $\underline{\mathbf{x}}_{\mathrm{i}}(t)$ is the interference signal vector. Obviously, the first L

components of $\underline{\mathbf{x}}'(t)$ are equal to $\mathbf{x}_1(t)$, while the first L components of $\underline{\mathbf{x}}_{\mathrm{i}}(t)$ are equal to $\mathbf{0}_{L\times 1}$. It can be verified that $\underline{\mathbf{x}}'(t)$ and $\underline{\mathbf{x}}_{\mathrm{i}}(t)$ are orthogonal, i.e.,

$$E\left[\underline{\mathbf{x}}'(t)\underline{\mathbf{x}}_{\mathrm{i}}^T(t)\right] = \mathbf{0}_{ML\times ML}. \tag{5.8}$$

Therefore, (5.4) can be rewritten as

$$\underline{\mathbf{y}}(t) = \underline{\mathbf{x}}'(t) + \underline{\mathbf{x}}_{\mathrm{i}}(t) + \underline{\mathbf{v}}(t) \tag{5.9}$$

and the correlation matrix of $\underline{\mathbf{y}}(t)$ from the previous expression is

$$\begin{aligned}\mathbf{R}_{\underline{\mathbf{y}}} &= \mathbf{R}_{\underline{\mathbf{x}}'} + \mathbf{R}_{\underline{\mathbf{x}}_{\mathrm{i}}} + \mathbf{R}_{\underline{\mathbf{v}}} \\ &= \mathbf{R}_{\underline{\mathbf{x}}'} + \mathbf{R}_{\mathrm{in}},\end{aligned} \tag{5.10}$$

where $\mathbf{R}_{\underline{\mathbf{x}}'} = \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T = \mathbf{R}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}^{-1}\mathbf{R}_{\underline{\mathbf{x}}\mathbf{x}_1}^T$ and $\mathbf{R}_{\underline{\mathbf{x}}_{\mathrm{i}}}$ are the correlation matrices of $\underline{\mathbf{x}}'(t)$ and $\underline{\mathbf{x}}_{\mathrm{i}}(t)$, respectively, and

$$\mathbf{R}_{\mathrm{in}} = \mathbf{R}_{\underline{\mathbf{x}}_{\mathrm{i}}} + \mathbf{R}_{\underline{\mathbf{v}}} \tag{5.11}$$

is the interference-plus-noise correlation matrix. It is clear that the rank of $\mathbf{R}_{\underline{\mathbf{x}}'}$ is L, while the rank of $\mathbf{R}_{\mathrm{in}}$ is assumed to be equal to ML.

Using the joint diagonalization technique [4], the two symmetric matrices $\mathbf{R}_{\underline{\mathbf{x}}'}$ and $\mathbf{R}_{\mathrm{in}}$ can be jointly diagonalized as follows:

$$\mathbf{B}^T\mathbf{R}_{\underline{\mathbf{x}}'}\mathbf{B} = \mathbf{\Lambda}, \tag{5.12}$$

$$\mathbf{B}^T\mathbf{R}_{\mathrm{in}}\mathbf{B} = \mathbf{I}_{ML}, \tag{5.13}$$

where $\mathbf{B}$ is a full-rank square matrix (of size $ML \times ML$), $\mathbf{\Lambda}$ is a diagonal matrix whose main elements are real and nonnegative, and $\mathbf{I}_{ML}$ is the $ML \times ML$ identity matrix. Furthermore, $\mathbf{\Lambda}$ and $\mathbf{B}$ are the eigenvalue and eigenvector matrices, respectively, of $\mathbf{R}_{\mathrm{in}}^{-1}\mathbf{R}_{\underline{\mathbf{x}}'}$, i.e.,

$$\mathbf{R}_{\mathrm{in}}^{-1}\mathbf{R}_{\underline{\mathbf{x}}'}\mathbf{B} = \mathbf{B}\mathbf{\Lambda}. \tag{5.14}$$

Since the rank of the matrix $\mathbf{R}_{\underline{\mathbf{x}}'}$ is equal to L, the eigenvalues of $\mathbf{R}_{\mathrm{in}}^{-1}\mathbf{R}_{\underline{\mathbf{x}}'}$ can be ordered as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_L > \lambda_{L+1} = \cdots = \lambda_{ML} = 0$. In other words, the last $ML - L$ eigenvalues of the matrix product $\mathbf{R}_{\mathrm{in}}^{-1}\mathbf{R}_{\underline{\mathbf{x}}'}$ are exactly zero, while its first L eigenvalues are positive, with λ_1 being the maximum eigenvalue. We also denote by $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{ML}$, the corresponding eigenvectors. Therefore, the noisy signal correlation matrix can also be diagonalized as

$$\mathbf{B}^T\mathbf{R}_{\underline{\mathbf{y}}}\mathbf{B} = \mathbf{\Lambda} + \mathbf{I}_{ML}. \tag{5.15}$$

We can decompose the matrix $\mathbf{B}$ as

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}' \ \mathbf{B}'' \end{bmatrix}, \tag{5.16}$$

where

$$\mathbf{B}' = \begin{bmatrix} \mathbf{b}_1 \ \mathbf{b}_2 \ \cdots \ \mathbf{b}_L \end{bmatrix} \tag{5.17}$$

is an $ML \times L$ matrix that spans the desired signal-plus-noise subspace and

$$\mathbf{B}'' = \begin{bmatrix} \mathbf{b}_{L+1} \ \mathbf{b}_{L+2} \ \cdots \ \mathbf{b}_{ML} \end{bmatrix} \tag{5.18}$$

is an $ML \times (ML - L)$ matrix that spans the noise subspace. As a result,

$$\mathbf{B}'^T \mathbf{R}_{\underline{\mathbf{x}}'} \mathbf{B}' = \mathbf{\Lambda}', \tag{5.19}$$

$$\mathbf{B}'^T \mathbf{R}_{\text{in}} \mathbf{B}' = \mathbf{I}_L, \tag{5.20}$$

$$\mathbf{B}'^T \mathbf{R}_{\underline{\mathbf{y}}} \mathbf{B}' = \mathbf{\Lambda}' + \mathbf{I}_L, \tag{5.21}$$

where $\mathbf{\Lambda}'$ is an $L \times L$ diagonal matrix containing the (nonnull) positive eigenvalues of $\mathbf{R}_{\text{in}}^{-1} \mathbf{R}_{\underline{\mathbf{x}}'}$ and $\mathbf{I}_L$ is the $L \times L$ identity matrix. Matrices $\mathbf{B}'$ and $\mathbf{\Lambda}'$ will be very useful to manipulate in the rest of the chapter.

5.2 Linear Filtering with a Rectangular Matrix

Multichannel noise reduction is performed by applying a linear transformation to $\underline{\mathbf{y}}(t)$. We get

$$\begin{aligned} \mathbf{z}(t) &= \mathbf{H}\underline{\mathbf{y}}(t) \\ &= \mathbf{x}_{\text{fd}}(t) + \mathbf{x}_{\text{ri}}(t) + \mathbf{v}_{\text{rn}}(t), \end{aligned} \tag{5.22}$$

where $\mathbf{z}(t)$ is the estimate of $\mathbf{x}_1(t)$,

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \vdots \\ \mathbf{h}_L^T \end{bmatrix} \tag{5.23}$$

is a rectangular filtering matrix of size $L \times ML$, $\mathbf{h}_l, \ l = 1, 2, \ldots, L$ are filters of length ML,

$$\mathbf{x}_{\text{fd}}(t) = \mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} \mathbf{x}_1(t) \tag{5.24}$$

is the filtered desired signal,

$$\mathbf{x}_{\text{ri}}(t) = \mathbf{H}\underline{\mathbf{x}}_{\text{i}}(t) \tag{5.25}$$

is the residual interference, and

$$\mathbf{v}_{\mathrm{rn}}(t) = \mathbf{H}\underline{\mathbf{v}}(t) \tag{5.26}$$

is the residual noise.

It is always possible to write $\mathbf{h}_l$ in a basis formed from the vectors $\mathbf{b}_i,\ i = 1, 2, \ldots, L$ that span the desired signal-plus-noise subspace, i.e.,

$$\begin{aligned}\mathbf{h}_l &= \sum_{i=1}^{L} a_{li}\mathbf{b}_i \\ &= \mathbf{B}\mathbf{a}_l,\end{aligned} \tag{5.27}$$

where $a_{li},\ i = 1, 2, \ldots, L$ are the components of the vector $\mathbf{a}_l$ of length L. Notice that we completely ignore the noise-only subspace as many optimal filtering matrices will do the same. We will see that this choice is reasonable and will lead to interesting filtering matrices for noise reduction. Therefore, the filtering matrix can be expressed as

$$\mathbf{H} = \mathbf{A}\mathbf{B}'^T, \tag{5.28}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_L^T \end{bmatrix} \tag{5.29}$$

is an $L \times L$ matrix. Now, instead of estimating $\mathbf{H}$ (of size $L \times ML$) as in conventional approaches, we estimate $\mathbf{A}$ (of size $L \times L$).

The correlation matrix of $\mathbf{z}(t)$ is then

$$\mathbf{R}_{\mathbf{z}} = \mathbf{R}_{\mathbf{x}_{\mathrm{fd}}} + \mathbf{R}_{\mathbf{x}_{\mathrm{ri}}} + \mathbf{R}_{\mathbf{v}_{\mathrm{rn}}}, \tag{5.30}$$

where

$$\begin{aligned}\mathbf{R}_{\mathbf{x}_{\mathrm{fd}}} &= \mathbf{A}\mathbf{B}'^T\mathbf{R}_{\underline{\mathbf{x}}'}\mathbf{B}'\mathbf{A}^T \\ &= \mathbf{A}\boldsymbol{\Lambda}'\mathbf{A}^T,\end{aligned} \tag{5.31}$$

$$\begin{aligned}\mathbf{R}_{\mathbf{x}_{\mathrm{ri}}} + \mathbf{R}_{\mathbf{v}_{\mathrm{rn}}} &= \mathbf{A}\mathbf{B}'^T\mathbf{R}_{\mathrm{in}}\mathbf{B}'\mathbf{A}^T \\ &= \mathbf{A}\mathbf{A}^T.\end{aligned} \tag{5.32}$$

5.3 Performance Measures

We explain the performance measures in the context of multichannel noise reduction in the time domain with sensor 1 as the reference. We start by deriving measures related to noise reduction. In the second subsection, we discuss the evaluation of desired signal distortion. Finally, we present the MSE criterion, which is very convenient to use in signal enhancement applications.

5.3.1 Noise Reduction

Since sensor 1 is the reference, the input SNR is computed from the first L components of $\underline{\mathbf{y}}(t)$ as defined in (5.9). We easily find that

$$\mathrm{iSNR} = \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}, \tag{5.33}$$

where $\mathbf{R}_{\mathbf{v}_1}$ is the correlation matrix of $\mathbf{v}_1(t)$.

The output SNR is obtained from (5.30). It is given by

$$\begin{aligned}\mathrm{oSNR}\left(\mathbf{A}\right) &= \frac{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\underline{\mathbf{x}}'}\mathbf{H}^T\right)}{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\mathrm{in}}\mathbf{H}^T\right)} \\ &= \frac{\mathrm{tr}\left(\mathbf{A}\mathbf{\Lambda}'\mathbf{A}^T\right)}{\mathrm{tr}\left(\mathbf{A}\mathbf{A}^T\right)}.\end{aligned} \tag{5.34}$$

Then, the main objective of multichannel signal enhancement is to find an appropriate $\mathbf{A}$ that makes the output SNR greater than the input SNR. Consequently, the quality of the noisy signal may be enhanced. It can be checked that

$$\mathrm{oSNR}\left(\mathbf{A}\right) \leq \max_l \frac{\mathbf{a}_l^T\mathbf{\Lambda}'\mathbf{a}_l}{\mathbf{a}_l^T\mathbf{a}_l}. \tag{5.35}$$

As a result,

$$\mathrm{oSNR}\left(\mathbf{A}\right) \leq \lambda_1. \tag{5.36}$$

This shows how the output SNR is upper bounded.

The noise reduction factor is defined as

$$\begin{aligned}\xi_{\mathrm{nr}}(\mathbf{A}) &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\mathrm{in}}\mathbf{H}^T\right)} \\ &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}{\mathrm{tr}\left(\mathbf{A}\mathbf{A}^T\right)}.\end{aligned} \tag{5.37}$$

For optimal filtering matrices, we should have $\xi_{\mathrm{nr}}(\mathbf{A}) \geq 1$. The noise reduction factor is not upper bounded and can go to infinity if we allow infinite distortion.

5.3.2 Desired Signal Distortion

The distortion of the desired signal vector can be measured with the desired signal reduction factor:

$$\begin{aligned}\xi_{\mathrm{sr}}(\mathbf{A}) &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\underline{\mathbf{x}}'}\mathbf{H}^T\right)} \\ &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}{\mathrm{tr}\left(\mathbf{A}\mathbf{\Lambda}'\mathbf{A}^T\right)}.\end{aligned} \tag{5.38}$$

For optimal filtering matrices, we should have $\xi_{\mathrm{sr}}(\mathbf{A}) \geq 1$. In the distortionless case, we have $\xi_{\mathrm{sr}}(\mathbf{A}) = 1$. Hence, a rectangular filtering matrix that does not affect the desired signal requires the constraint:

$$\begin{aligned}\mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} &= \mathbf{A}\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} \\ &= \mathbf{I}_L.\end{aligned} \tag{5.39}$$

It is obvious that we always have

$$\frac{\mathrm{oSNR}(\mathbf{A})}{\mathrm{iSNR}} = \frac{\xi_{\mathrm{nr}}(\mathbf{A})}{\xi_{\mathrm{sr}}(\mathbf{A})}. \tag{5.40}$$

The distortion can also be measured with the desired signal distortion index:

$$\begin{aligned}\upsilon_{\mathrm{sd}}(\mathbf{A}) &= \frac{E\left\{\left[\mathbf{x}_{\mathrm{fd}}(t) - \mathbf{x}_1(t)\right]^T\left[\mathbf{x}_{\mathrm{fd}}(t) - \mathbf{x}_1(t)\right]\right\}}{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)} \\ &= \frac{\mathrm{tr}\left[\left(\mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{R}_{\mathbf{x}_1}\left(\mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} - \mathbf{I}_L\right)^T\right]}{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)} \\ &= \frac{\mathrm{tr}\left[\left(\mathbf{A}\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{R}_{\mathbf{x}_1}\left(\mathbf{A}\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} - \mathbf{I}_L\right)^T\right]}{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}.\end{aligned} \tag{5.41}$$

For optimal rectangular filtering matrices, we should have

$$0 \leq \upsilon_{\text{sd}}(\mathbf{A}) \leq 1 \tag{5.42}$$

and a value of $\upsilon_{\text{sd}}(\mathbf{A})$ close to 0 is preferred.

5.3.3 MSE Criterion

It is clear that the error signal vector between the estimated and desired signals is

$$\begin{aligned}\mathbf{e}(t) &= \mathbf{z}(t) - \mathbf{x}_1(t) \\ &= \mathbf{H}\underline{\mathbf{y}}(t) - \mathbf{x}_1(t) \\ &= \mathbf{e}_{\text{ds}}(t) + \mathbf{e}_{\text{rs}}(t),\end{aligned} \tag{5.43}$$

where

$$\begin{aligned}\mathbf{e}_{\text{ds}}(t) &= \mathbf{x}_{\text{fd}}(t) - \mathbf{x}_1(t) \\ &= \left(\mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{x}_1(t)\end{aligned} \tag{5.44}$$

represents the signal distortion and

$$\begin{aligned}\mathbf{e}_{\text{rs}}(t) &= \mathbf{x}_{\text{ri}}(t) + \mathbf{v}_{\text{rn}}(t) \\ &= \mathbf{H}\underline{\mathbf{x}}_{\text{i}}(t) + \mathbf{H}\underline{\mathbf{v}}(t)\end{aligned} \tag{5.45}$$

represents the residual interference-plus-noise. We deduce that the MSE criterion is

$$\begin{aligned}J(\mathbf{A}) &= \text{tr}\left\{E\left[\mathbf{e}(t)\mathbf{e}^T(t)\right]\right\} \\ &= \text{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right) + \text{tr}\left(\mathbf{H}\mathbf{R}_{\underline{\mathbf{y}}}\mathbf{H}^T\right) - 2\text{tr}\left(\mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\right) \\ &= \text{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right) + \text{tr}\left[\mathbf{A}\left(\mathbf{\Lambda}' + \mathbf{I}_L\right)\mathbf{A}^T\right] - 2\text{tr}\left(\mathbf{A}\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\right).\end{aligned} \tag{5.46}$$

Since $E\left[\mathbf{e}_{\text{ds}}(t)\mathbf{e}_{\text{rs}}^T(t)\right] = \mathbf{0}_{L\times L}$, $J(\mathbf{A})$ can also be expressed as

$$\begin{aligned}J(\mathbf{A}) &= \text{tr}\left\{E\left[\mathbf{e}_{\text{ds}}(t)\mathbf{e}_{\text{ds}}^T(t)\right]\right\} + \text{tr}\left\{E\left[\mathbf{e}_{\text{rs}}(t)\mathbf{e}_{\text{rs}}^T(t)\right]\right\} \\ &= J_{\text{ds}}(\mathbf{A}) + J_{\text{rs}}(\mathbf{A}),\end{aligned} \tag{5.47}$$

where

$$\begin{aligned}J_{\text{ds}}(\mathbf{A}) &= \text{tr}\left[\left(\mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{R}_{\mathbf{x}_1}\left(\mathbf{H}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1} - \mathbf{I}_L\right)^T\right] \\ &= \text{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)\upsilon_{\text{sd}}(\mathbf{A})\end{aligned} \tag{5.48}$$

and

$$\begin{aligned} J_{\mathrm{rs}}(\mathbf{A}) &= \mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\mathrm{in}}\mathbf{H}^T\right) \\ &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}{\xi_{\mathrm{nr}}(\mathbf{A})}. \end{aligned} \tag{5.49}$$

Finally, we have

$$\begin{aligned} \frac{J_{\mathrm{ds}}(\mathbf{A})}{J_{\mathrm{rs}}(\mathbf{A})} &= \mathrm{iSNR} \times \xi_{\mathrm{nr}}(\mathbf{A}) \times \upsilon_{\mathrm{sd}}(\mathbf{A}) \\ &= \mathrm{oSNR}(\mathbf{A}) \times \xi_{\mathrm{sr}}(\mathbf{A}) \times \upsilon_{\mathrm{sd}}(\mathbf{A}). \end{aligned} \tag{5.50}$$

This shows how the MSEs are related to the most fundamental performance measures.

5.4 Optimal Rectangular Linear Filtering Matrices

In this section, we derive the most important rectangular filtering matrices that can help reduce the level of the noise. We will see how these optimal matrices are very closely related thanks to the proposed formulation.

5.4.1 Maximum SNR

From Subsection 5.3.1, we know that the output SNR is upper bounded by λ_1, which we can consider as the maximum possible output SNR. Then, it is easy to verify that with

$$\mathbf{A}_{\max} = \begin{bmatrix} a_{11}\mathbf{i}^T \\ a_{21}\mathbf{i}^T \\ \vdots \\ a_{L1}\mathbf{i}^T \end{bmatrix}, \tag{5.51}$$

where $a_{l1},\ l = 1, 2, \ldots, L$ are arbitrary real numbers with at least one of them different from 0 and $\mathbf{i}$ is the first column of the $L \times L$ identity matrix, $\mathbf{I}_L$, we have

$$\mathrm{oSNR}(\mathbf{A}_{\max}) = \lambda_1. \tag{5.52}$$

As a consequence,

$$\begin{aligned}\mathbf{H}_{\max} &= \mathbf{A}_{\max}\mathbf{B}'^T \\ &= \begin{bmatrix} a_{11}\mathbf{b}_1^T \\ a_{21}\mathbf{b}_1^T \\ \vdots \\ a_{L1}\mathbf{b}_1^T \end{bmatrix}\end{aligned} \tag{5.53}$$

is considered to be the maximum SNR filtering matrix. Clearly,

$$\mathrm{oSNR}\left(\mathbf{H}_{\max}\right) \geq \mathrm{iSNR} \tag{5.54}$$

and

$$0 \leq \mathrm{oSNR}\left(\mathbf{H}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\max}\right), \forall \mathbf{H}. \tag{5.55}$$

The choice of the values of $a_{l1},\ l = 1, 2, \ldots, L$ is extremely important in practice. A poor choice of these values leads to high distortions of the desired signal. Therefore, the a_{l1}'s should be found in such a way that distortion is minimized. Substituting (5.51) into the the distortion-based MSE, we get

$$J_{\mathrm{ds}}\left(\mathbf{H}_{\max}\right) = \mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right) + \lambda_1 \sum_{l=1}^{L} a_{l1}^2 - 2\sum_{l=1}^{L} a_{l1}\mathbf{i}^T\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\mathbf{i}_{l,L}, \tag{5.56}$$

where $\mathbf{i}_{l,L}$ is the lth column of $\mathbf{I}_L$, and minimizing the previous expression with respect to the a_{l1}'s, we find

$$a_{l1} = \mathbf{i}_{l,L}^T\mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T\frac{\mathbf{b}_1}{\lambda_1},\ l = 1, 2, \ldots, L. \tag{5.57}$$

Plugging these optimal values in (5.53), we obtain the optimal maximum SNR filtering matrix with minimum desired signal distortion:

$$\mathbf{H}_{\max} = \mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T\frac{\mathbf{b}_1\mathbf{b}_1^T}{\lambda_1}. \tag{5.58}$$

5.4.2 Wiener

If we differentiate the MSE criterion, $J\left(\mathbf{A}\right)$, with respect to $\mathbf{A}$ and equate the result to zero, we find

$$\mathbf{A}_{\mathrm{W}} = \mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T\mathbf{B}'\left(\mathbf{\Lambda}' + \mathbf{I}_L\right)^{-1}. \tag{5.59}$$

We deduce that the Wiener filtering matrix for the estimation of the vector $\mathbf{x}_1(t)$, which is confined in the desired signal-plus-noise subspace[1], is

$$\mathbf{H}_{\mathrm{W}} = \mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{l=1}^{L} \frac{\mathbf{b}_l\mathbf{b}_l^T}{1+\lambda_l}. \tag{5.60}$$

From the proposed formulation, we see clearly how $\mathbf{H}_{\mathrm{W}}$ and $\mathbf{H}_{\max}$ are related. Besides a (slight) different weighting factor, $\mathbf{H}_{\mathrm{W}}$ considers all directions where speech is present, while $\mathbf{H}_{\max}$ relies only on the direction where the maximum of speech energy is present.

Property 5.1. The output SNR with the Wiener filtering matrix is always greater than or equal to the input SNR, i.e., $\mathrm{oSNR}\left(\mathbf{H}_{\mathrm{W}}\right) \geq \mathrm{iSNR}$.

Obviously, we have

$$\mathrm{oSNR}\left(\mathbf{H}_{\mathrm{W}}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\max}\right) \tag{5.61}$$

and, in general,

$$\upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{W}}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\max}\right). \tag{5.62}$$

5.4.3 MVDR

The MVDR filtering matrix is obtained directly from the constraint (5.39). Since $\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}$ is a full-rank square matrix, we deduce that

$$\mathbf{A}_{\mathrm{MVDR}} = \left(\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\right)^{-1}. \tag{5.63}$$

As a result, the MVDR filtering matrix is

$$\mathbf{H}_{\mathrm{MVDR}} = \left(\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\right)^{-1}\mathbf{B}'^T. \tag{5.64}$$

From (5.19), we find that

$$\left(\mathbf{B}'^T\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}\right)^{-1} = \mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T\mathbf{B}'\mathbf{\Lambda}'^{-1}, \tag{5.65}$$

suggesting that we can formulate the MVDR as

$$\mathbf{H}_{\mathrm{MVDR}} = \mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{l=1}^{L} \frac{\mathbf{b}_l\mathbf{b}_l^T}{\lambda_l}. \tag{5.66}$$

[1] This Wiener filtering matrix is different from the conventional one given in [3] within the same context.

It is worth comparing $\mathbf{H}_{\text{MVDR}}$ with $\mathbf{H}_{\max}$ and $\mathbf{H}_{\text{W}}$.

Property 5.2. The output SNR with the MVDR filtering matrix is always greater than or equal to the input SNR, i.e., $\text{oSNR}\left(\mathbf{H}_{\text{MVDR}}\right) \geq \text{iSNR}$.

We have

$$\text{oSNR}\left(\mathbf{H}_{\text{MVDR}}\right) \leq \text{oSNR}\left(\mathbf{H}_{\text{W}}\right) \leq \text{oSNR}\left(\mathbf{H}_{\max}\right) \tag{5.67}$$

and, obviously, with the MVDR filtering matrix, we have no distortion, i.e.,

$$\xi_{\text{sr}}\left(\mathbf{H}_{\text{MVDR}}\right) = 1, \tag{5.68}$$

$$\upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MVDR}}\right) = 0. \tag{5.69}$$

From the obvious relationship between the MVDR and maximum SNR filtering matrices, we can deduce a whole class of minimum distortion filtering matrices:

$$\mathbf{H}_{\text{MD},Q} = \mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{q=1}^{Q} \frac{\mathbf{b}_q\mathbf{b}_q^T}{\lambda_q}, \tag{5.70}$$

where $1 \leq Q \leq L$. We observe that $\mathbf{H}_{\text{MD},1} = \mathbf{H}_{\max}$ and $\mathbf{H}_{\text{MD},L} = \mathbf{H}_{\text{MVDR}}$. Also, we have

$$\text{oSNR}\left(\mathbf{H}_{\text{MD},L}\right) \leq \text{oSNR}\left(\mathbf{H}_{\text{MD},L-1}\right) \leq \cdots \leq \text{oSNR}\left(\mathbf{H}_{\text{MD},1}\right) = \lambda_1 \tag{5.71}$$

and

$$0 = \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MD},L}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MD},L-1}\right) \leq \cdots \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MD},1}\right). \tag{5.72}$$

5.4.4 Tradeoff

By minimizing the speech distortion index with the constraint that the noise reduction factor is equal to a positive value that is greater than 1, we get the tradeoff filtering matrix:

$$\mathbf{H}_{\text{T},\mu} = \mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{l=1}^{L} \frac{\mathbf{b}_l\mathbf{b}_l^T}{\mu + \lambda_l}, \tag{5.73}$$

where $\mu \geq 0$ is a Lagrange multiplier. We observe that $\mathbf{H}_{\text{T},0} = \mathbf{H}_{\text{MVDR}}$ and $\mathbf{H}_{\text{T},1} = \mathbf{H}_{\text{W}}$.

Property 5.3. The output SNR with the tradeoff filtering matrix is always greater than or equal to the input SNR, i.e., $\text{oSNR}\left(\mathbf{H}_{\text{T},\mu}\right) \geq \text{iSNR}, \ \forall \mu \geq 0$.

Table 5.1 Optimal linear filtering matrices for multichannel signal enhancement in the time domain.

Maximum SNR:	$\mathbf{H}_{\max} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \frac{\mathbf{b}_1 \mathbf{b}_1^T}{\lambda_1}$
Wiener:	$\mathbf{H}_{\mathrm{W}} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^T}{1+\lambda_l}$
MVDR:	$\mathbf{H}_{\mathrm{MVDR}} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^T}{\lambda_l}$
MD,Q:	$\mathbf{H}_{\mathrm{MD},Q} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^T}{\lambda_q}$
Tradeoff:	$\mathbf{H}_{\mathrm{T},\mu} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^T}{\mu+\lambda_l}$
General Tradeoff:	$\mathbf{H}_{\mu,Q} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^T}{\mu+\lambda_q}$

We should have for $\mu \geq 1$,

$$\mathrm{oSNR}\left(\mathbf{H}_{\mathrm{MVDR}}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\mathrm{W}}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\mathrm{T},\mu}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\max}\right), \tag{5.74}$$

$$0 = \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{MVDR}}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{W}}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{T},\mu}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\max}\right), \tag{5.75}$$

and for $\mu \leq 1$,

$$\mathrm{oSNR}\left(\mathbf{H}_{\mathrm{MVDR}}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\mathrm{T},\mu}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\mathrm{W}}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\max}\right), \tag{5.76}$$

$$0 = \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{MVDR}}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{T},\mu}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{W}}\right) \leq \upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\max}\right). \tag{5.77}$$

From all what we have seen so far, we can propose a very general noise reduction filtering matrix:

$$\mathbf{H}_{\mu,Q} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\mathbf{x}}\mathbf{x}_1}^T \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^T}{\mu+\lambda_q}. \tag{5.78}$$

This form encompasses all known optimal filtering matrices. Indeed, it is clear that

- $\mathbf{H}_{0,1} = \mathbf{H}_{\max}$,
- $\mathbf{H}_{1,L} = \mathbf{H}_{\mathrm{W}}$,
- $\mathbf{H}_{0,L} = \mathbf{H}_{\mathrm{MVDR}}$,
- $\mathbf{H}_{0,Q} = \mathbf{H}_{\mathrm{MD},Q}$,
- $\mathbf{H}_{\mu,L} = \mathbf{H}_{\mathrm{T},\mu}$.

In Table 5.1, we give all optimal filtering matrices studied in this chapter.

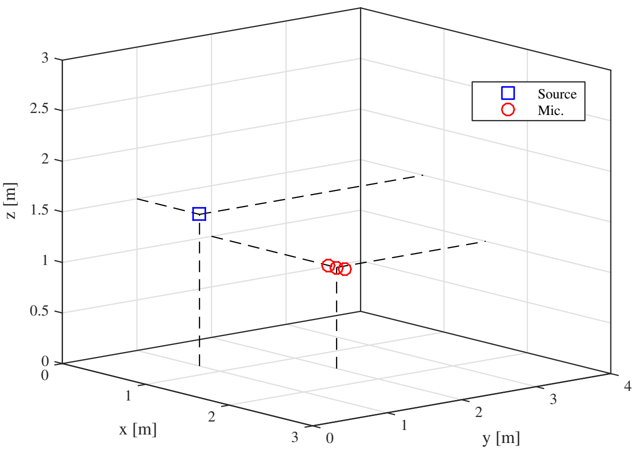


Fig. 5.1 Illustration of the room setup used in the evaluations in Section 5.5 with $M = 3$ microphones and a microphone spacing of $d = 0.1$ m.

5.5 Experimental Results

In this section, we present the experimental evaluations of the filters proposed in Section 5.4. For this evaluation, we considered the enhancement of different reverberant speech signals contaminated by babble noise. More specifically, the speech signals considered were two female and two male speech signals. This amounted to approximately 10 seconds of female speech and 10 seconds of male speech. The signals were all single-channel speech signals, so to obtain multichannel speech signals for the evaluation, we used a room impulse response (RIR) generator [5]. The setup for generating the multichannel signal were the same for all evaluations in this section and were as follows. The simulated room had dimensions $3 \times 4 \times 3$ m. In this room, the source was located at $(0.75, 1, 1.5)$ m, while the microphones where placed at $(1.5 + d[m - \frac{M-1}{2}], 2, 1)$ m for $m = 0, \ldots, M-1$ with d denoting the microphone spacing. The sensor spacing was 0.1 m in all evaluations. In Fig. 5.1, the source and microphone organization is illustrated for a scenario with $M = 3$ microphones and a sensor spacing $d = 0.1$ m. Besides this, the speed of sound was assumed to be 343 m/s, the 60 dB reverberation time was 0.2 s for all frequencies, the room impulse response length was 2,048, the microphone types was omnidirectional, and highpass filtering of the RIRs was enabled. With this setup, we then generated our clean, multichannel speech signals including reverberation. An example of the first five seconds of the speech signal obtained by one of the microphones in a three-microphone setup, like in Fig. 5.1, is shown in Fig. 5.2.

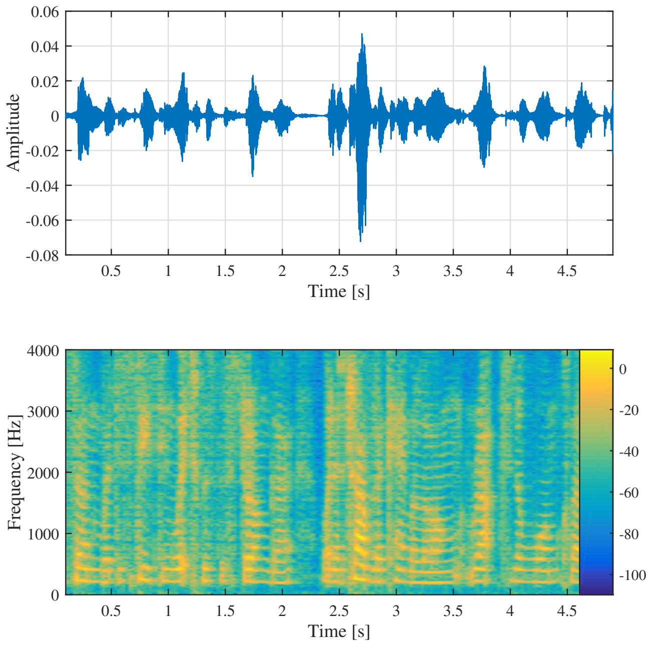


Fig. 5.2 Plot of (top) the first five seconds of the utilized speech signal measured by a single microphone with the setup depicted in Fig. 5.1 and (bottom) the spectrogram of it.

To generate noisy multichannel signals for the evaluations, we added two types of noise to the clean signal: simulated sensor noise and simulated diffuse noise. The sensor noise was constituted by white Gaussian noise in each channel, while the diffuse noise was babble noise. The diffuse babble noise was generated by synthesizing multichannel diffuse babble noise using a single channel babble noise signal from the AURORA database [6] and assuming a spherical noise field. The procedure for generating multichannel diffuse noise followed herein are described in [7], and a MATLAB implementation of this noise generation method is available online[2]. An example of the first five seconds of the diffuse noise signal obtained by the one of the microphones in a three-microphone setup like in Fig. 5.1, is shown in Fig. 5.3.

Using mixtures of the speech and these different noise types (white sensor noise and diffuse background noise), we then conducted evaluations of the aforementioned filters in terms of their output SNRs and speech reduction factors. In each of the evaluations, enhancement of the speech signal simultaneously mixed with the two noise types were considered, and the performance of the filters were then measured over time and averaged. Hence, the depicted

[2] http://www.audiolabs-erlangen.de/fau/professor/habets/software/noise-generators

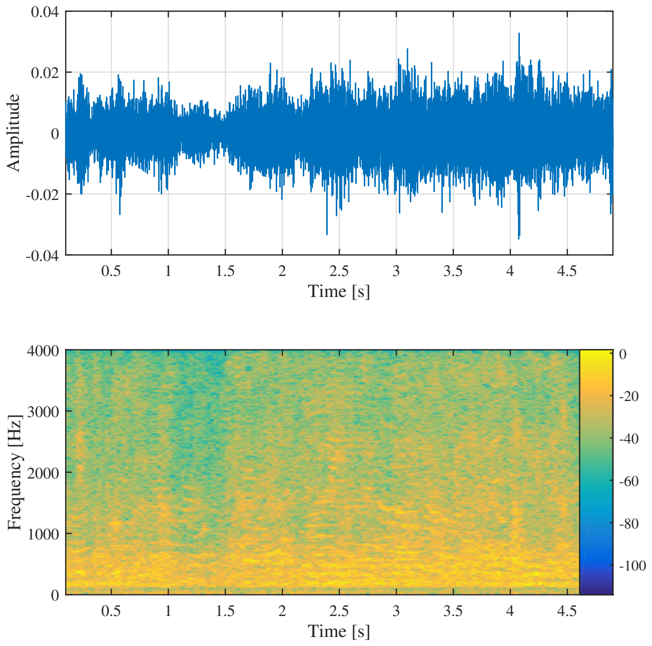


Fig. 5.3 Plot of (top) the first five seconds of the utilized diffuse noise signal captured by a single microphone in the setup depicted in Fig. 5.1 and (bottom) the spectrogram of it.

results in the remainder of the chapter are the performance measures averaged over time for different simulation settings.

It appeared from the previous section that, in order to design the optimal variable span based filters for multichannel enhancement in the time domain, we need estimates of the time-domain steering matrix which is related to the cross-correlation matrix between $\underline{\mathbf{x}}(t)$ and $\mathbf{x}_1(t)$ as well as the correlation matrix of $\mathbf{x}_1$. Moreover, knowledge about the interference-plus-noise correlation matrix, $\mathbf{R}_{\text{in}}$, is needed. The focus herein is not on noise or signal estimation, but rather on the relative performance of the presented filter designs, so the necessary statistics are estimated directly from the desired signal and the noise signal. To estimate the statistics in practice, techniques such as VAD, minimum statistics or sequential methods could be pursued [8], [9], [10], [11]. The statistics estimation directly from the speech and noise signals, respectively, was conducted recursively using the following general equation for approximating the (cross-)correlation, $\mathbf{R}_{\mathbf{ab}}$, between two vectors, $\mathbf{a}(t)$ and $\mathbf{b}(t)$:

$$\widehat{\mathbf{R}}_{\mathbf{ab}}(t) = (1-\xi)\widehat{\mathbf{R}}_{\mathbf{ab}}(t-1) + \xi\mathbf{a}(t)\mathbf{b}^T(t), \tag{5.79}$$

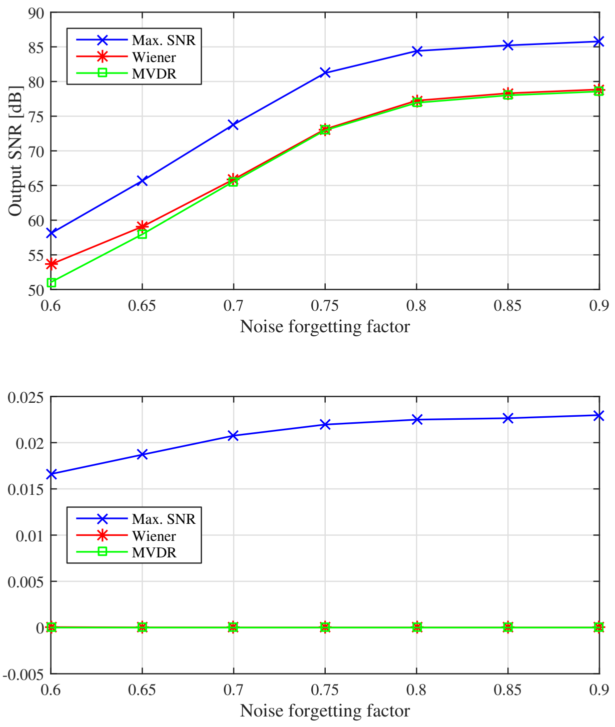


Fig. 5.4 Evaluation of the performance of the maximum SNR, Wiener, and MVDR filters versus the noise forgetting factor, ξ_{n}.

where ξ is the forgetting factor, and $\widehat{\mathbf{R}}_{\mathbf{ab}}(t)$ denotes the estimate of $\mathbf{R}_{\mathbf{ab}}$ at time instance t. The MATLAB code used for conducting the evaluations can be found in Section 5.A and in the Appendix A.

In the evaluations, two forgetting factors were used: a signal forgetting factor, ξ_{s}, which was applied in the estimation of signal related correlation matrices (i.e., $\mathbf{R}_{\mathbf{x}_1}$, $\mathbf{R}_{\underline{\mathbf{x}}\mathbf{x}_1}$, $\mathbf{R}_{\underline{\mathbf{x}}'}$, and $\mathbf{R}_{\underline{\mathbf{x}}_{\text{i}}}$), and, a noise forgetting factor, ξ_{n}, which was applied for estimation of $\mathbf{R}_{\underline{\mathbf{v}}}$. The first evaluations were therefore considering the performances of the optimal filter designs versus the forgetting factors. For this investigation, we considered a scenario with an average input signal-to-diffuse-noise ratio (iSDNR) of 0 dB, and an average input signal-to-sensor-noise ratio (iSSNR) of 30 dB. The number of microphones was $M = 3$, and the number of temporal samples was $L = 60$. The spatio-temporal blocks of observed samples considered for enhancement were overlapping by 50 % in time. Therefore, to obtain the enhanced signal using a particular filter, the output vectors were combined using overlap-add with

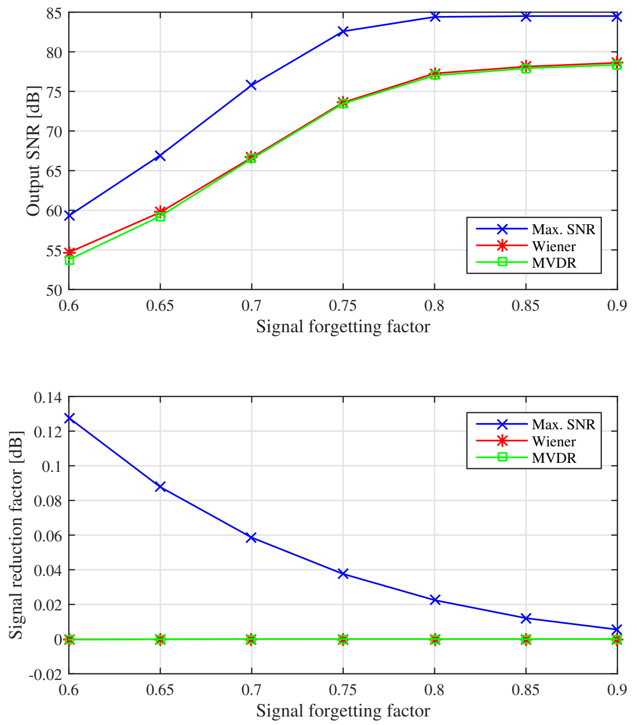


Fig. 5.5 Evaluation of the performance of the maximum SNR, Wiener, and MVDR filters versus the signal forgetting factor, ξ_s.

Hanning windowing. This overlap-add procedure was used all evaluations in this chapter.

With the simulation setup described above, we then fixed the signal forgetting factor to 0.8, and varied the noise forgetting factor to obtain the results depicted in Fig. 5.4. From the figure, we see that the output SNRs of all the filters increase as we increase the noise forgetting factor. However, the performance improvement is largest from a noise forgetting factor of 0.6 up to 0.8, after which the output SNR flattens out. Regarding distortion, the Wiener and MVDR filter have almost no distortion for all forgetting factors, whereas the maximum SNR filter has an increasing distortion, when we increase the noise forgetting factor. We then conducted another evaluation using the same simulation setup except that the noise forgetting factor was now fixed to 0.8, while the signal forgetting factor was varied. The results from this evaluation are plotted in Fig. 5.5. Similar to the previous evaluation, the output SNRs of all filters increase when the signal forgetting factor is increased. The output

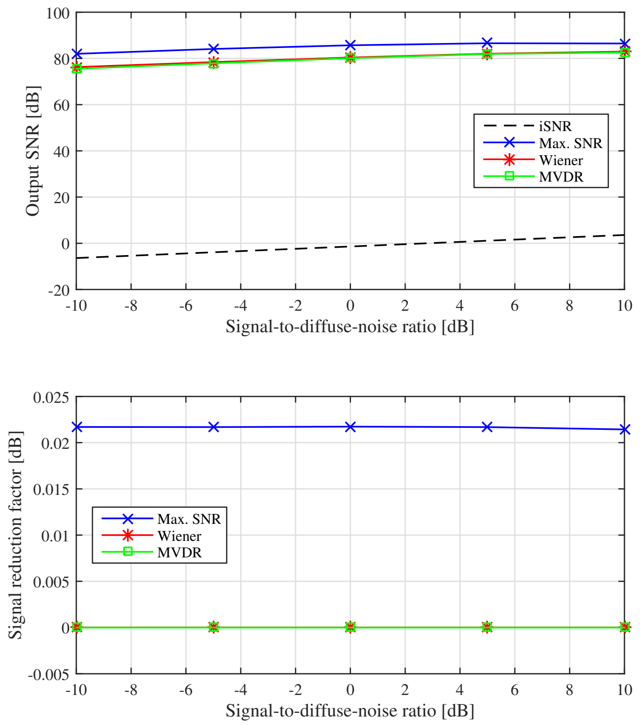


Fig. 5.6 Evaluation of the performance of the maximum SNR, Wiener, and MVDR filters versus the input SDNR.

SNRs, however, flatten out for a signal forgetting factor larger than 0.8. When it comes to the signal reduction factor, it is decreasing for the maximum SNR filter, when the signal forgetting factor is increasing. The Wiener and MVDR filters have much smaller distortions close to 0 for all signal forgetting factors. Based on these simulations, we choose signal and noise forgetting factors of $\xi_s = 0.8$ and $\xi_n = 0.8$ for the remaining evaluations in this chapter.

In the next evaluation, we investigated the filter performances versus the input SDNR. For this evaluation, we considered a scenario, where the input SSNR was 30 dB, the number of microphones was $M = 3$, and the temporal sample length was $L = 80$. With this setup, the output SNR and signal reduction factor were then measured as a function of the input SDNR, yielding the results provided in Fig. 5.6. The output SNRs generally increase for an increasing input SDNR. However, the SNR gain (i.e., the difference between input and output SNR) is smaller for higher input SDNRs. The relation between the filters is as expected, i.e., the maximum SNR filter has the highest

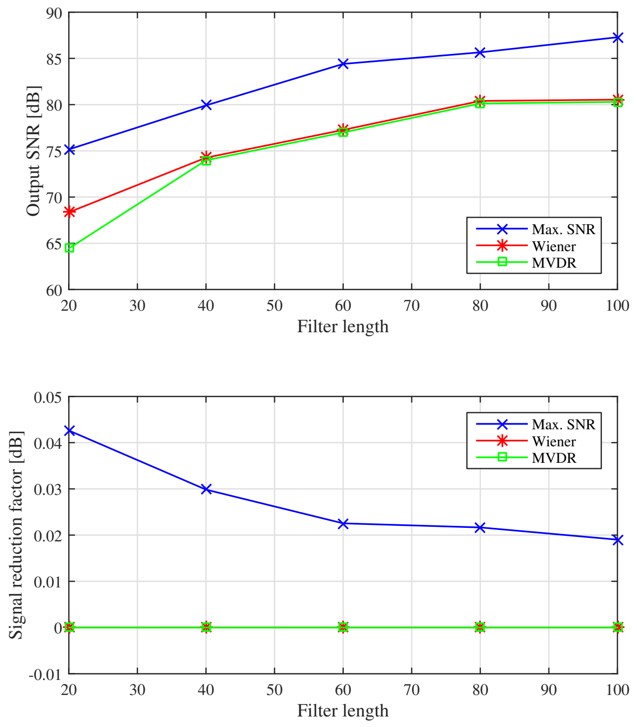


Fig. 5.7 Evaluation of the performance of the maximum SNR, Wiener, and MVDR filters versus the filter length, L.

output SNR followed by the Wiener and MVDR filters. The signal reduction factors do not change much versus the input SDNR. The Wiener and MVDR filters have nearly no distortion, while the maximum SNR filter has a significantly higher signal reduction factor.

An evaluation versus the temporal sample length, i.e., the filter length, was also conducted. In this experiment, the input SDNR was 0 dB, the input SSNR was 30 dB, and the number of microphones was $M = 3$. The temporal filter length was then varied to measure its influence on the filter performances. The results obtained are depicted in Fig. 5.7. If the filter length is increased, all filters provide an increase in output SNR in the considered interval, except between filter lengths of 80 and 100, where the Wiener and MVDR filters have nearly the same output SNR. The distortion for the maximum SNR filter also decreases for an increasing filter length, whereas the Wiener and MVDR filter has close to zero distortion for all filter lengths.

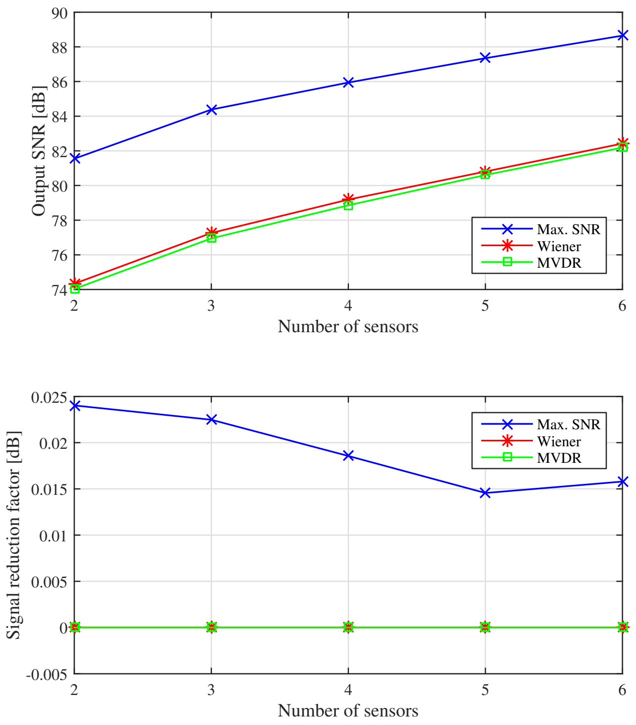


Fig. 5.8 Evaluation of the performance of the maximum SNR, Wiener, and MVDR filters versus the number of microphones, M.

This suggests, that the filter length should be between 80 and 100 for this particular simulation setup.

The following evaluation then considers the filter performance versus the number of microphones. This evaluation was done with an input SDNR of 0 dB, an input SSNR of 30 dB, and a filter length of $L = 60$. The results in Fig. 5.8 were then obtained using this simulation setup. We see that all filters have an increasing output SNR for an increasing number of sensors. Increasing the number of sensors from 2 to 6 results in an increase in output SNR of more than 6 dB. We observe that the maximum SNR filter has a significantly higher output SNR (approximately 7 dB) than the Wiener and MVDR filters for all numbers of sensors. However, as we can see from the signal reduction factor results, this comes at the cost of a higher distortion. The Wiener and MVDR filters, on the other hand, have small signal reduction factors close to 0 dB for all numbers of sensors.

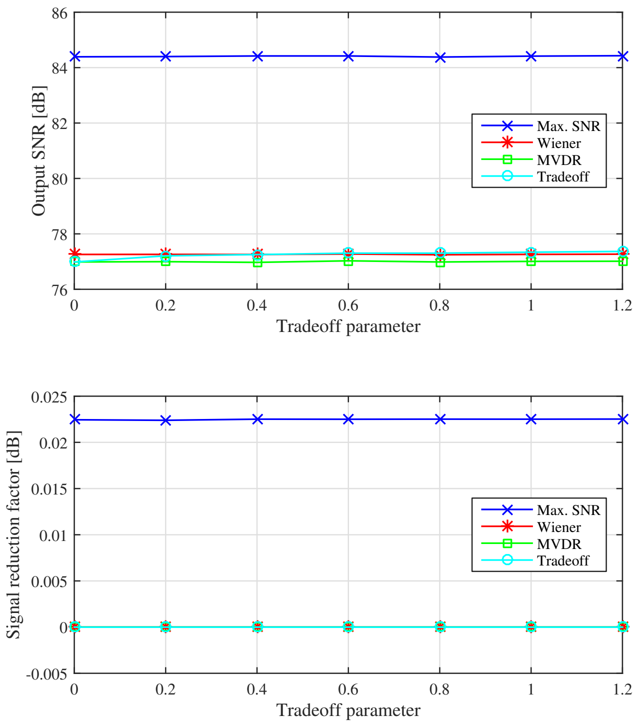


Fig. 5.9 Evaluation of the performance of the maximum SNR, Wiener, MVDR, and tradeoff filters versus the tradeoff parameter, μ.

In the final two evaluations, we evaluated the tradeoff filter for different choices of the tradeoff parameters. The general tradeoff filter has two tradeoff parameters: the assumed signal subspace rank, Q, and the Lagrange multiplier, μ, which we often refer to just as the tradeoff parameter. To investigate the influence of these parameters on the filtering performance, we considered a scenario with an input SSNR of 30 dB, and input SDNR of 0 dB, $M = 3$ microphones, and a filter length of $L = 60$. First, we then fixed Q to be equal to the filter length, and varied the tradeoff parameter, μ. This resulted in the measured performances depicted in Fig. 5.9. These results confirm that by tuning μ, when $Q = L$, the tradeoff filter can have different performances. As mentioned in connection with the filter derivations, the tradeoff filter yields the same performance as the MVDR filter when $\mu = 0$. When μ increases, the output SNR also increases, and when $\mu = 1$, the output SNR is equal to that of the Wiener filter. Moreover, if we let $\mu > 1$, we see that the output SNR is greater than the output SNRs of both the MVDR and Wiener

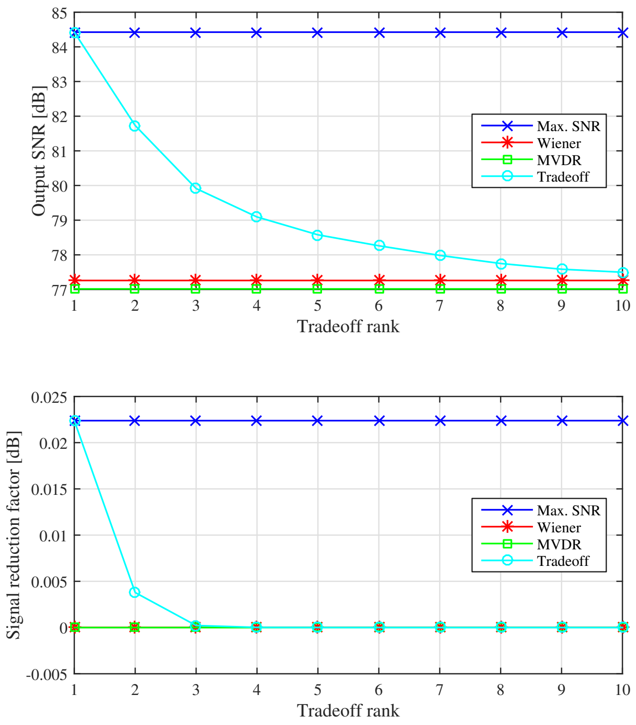


Fig. 5.10 Evaluation of the performance of the maximum SNR, Wiener, MVDR, and tradeoff filters versus the assumed signal subspace rank, Q

filter. However, when we increase the output SNR by increasing μ, we also increase the amount of distortion, as we can see from the signal reduction factor plot. We then investigated the influence of choosing different Q's on the filter performance by fixing μ to 1 and varying Q. This resulted in the plots in Fig. 5.10. We can see that the filter performance changes much more dramatically versus different Q's compared to when changing μ. When $Q = 1$, the tradeoff filter resembles the maximum SNR filter, and when we increase Q, the output SNR and signal reduction factors of the tradeoff filter then decays toward those of the MVDR and Wiener filters.

References

1. J. Benesty, J. Chen, and Y. Huang, *Microphone Array Signal Processing*. Berlin, Germany: Springer-Verlag, 2008.

2. M. Brandstein and D. B. Ward, Eds., *Microphone Arrays: Signal Processing Techniques and Applications*. Berlin, Germany: Springer-Verlag, 2001.
3. J. Benesty and J. Chen, *Optimal Time-domain Noise Reduction Filters–A Theoretical Study*. Springer Briefs in Electrical and Computer Engineering, 2011.
4. J. N. Franklin, *Matrix Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1968.
5. E. A. P. Habets, “Room impulse response generator,” Tech. Rep., Technische Universiteit Eindhoven, 2010, Ver. 2.0.20100920.
6. H.-G. Hirsch, and D. Pearce, “The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions,” in *Proc. ISCA Tutorial and Research Workshop ASR2000*, 2000.
7. E. A. P. Habets and S. Gannot, “Generating sensor signals in isotropic noise fields,” *J. Acoust. Soc. Am.*, vol. 122, pp. 3464–3470, Dec. 2007.
8. J. S. Sohn, N. S. Kim, and W. Sung, “A statistical model-based voice activity detection,” *IEEE Signal Process. Letters*, vol. 6, pp. 1–3, Jan. 1999.
9. R. Martin, “Noise power spectral density estimation based on optimal smoothing and minimum statistics,” *IEEE Trans. Speech Audio Process.*, vol. 9, pp. 504–512, Jul. 2001.
10. T. Gerkmann and R. Hendriks, “Unbiased MMSE-based noise power estimation with low complexity and low tracking delay,” *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, pp. 1383–1393, May 2012.
11. E. J. Diethorn, “Subband noise reduction methods for speech enhancement,” in *Audio Signal Processing for Next-Generation Multimedia Communication Systems*, Y. Huang and J.Benesty, eds., Boston, MA, USA: Kluwer, 2004, Chapter 4, pp. 91–115.

5.A MATLAB Code

5.A.1 Main Scripts

Listing 5.1 Script for evaluating the filter performances versus the noise forgetting factor.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 60;
setup.forgetNoiGrid = 0.6:0.05:0.9;
setup.forgetSig = 0.8;

setup.filtStrings = {'wiener','mvdr','maxSnr'};

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.forgetNoiGrid),
    setup.forgetNoi = setup.forgetNoiGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = vsTimeDomEnhanceMultChanSignals(...
        signals,setup);

    performance(idx,1) = vsTimeDomMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

%%
display('Measuring performance...');
for idx = 1:length(setup.forgetNoiGrid),
    iSnrMean(1,idx) = performance(idx,1).noiseReduction.iSnr.mean;
    oSnrMaxSnrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.mean;
    oSnrWienerMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.mean;
    oSnrMvdrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.mean;

    dsdMaxSnrMean(1,idx) = ...
    performance(idx,1).signalDistortion.dsd.maxSnr.mean;
    dsdWienerMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.mean;
    dsdMvdrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.mean;
end

for idx = 1:length(setup.forgetNoiGrid),
    iSnrOverTime(:,idx) = performance(idx,1).noiseReduction.iSnr.overTime;
    oSnrMaxSnrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.overTime;
    oSnrWienerOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.overTime;
    oSnrMvdrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.overTime;

    dsdMaxSnrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.overTime;
    dsdWienerOverTime(:,idx) = ...
    performance(idx,1).signalDistortion.dsd.wiener.overTime;
    dsdMvdrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.overTime;
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
close all;
figure(1);
plot(10*log10(mean(iSnrMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrMean,3)));
plot(10*log10(mean(oSnrWienerMean,3)));
plot(10*log10(mean(oSnrMvdrMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrMean,3)));
hold on;
plot(10*log10(mean(dsdWienerMean,3)));
plot(10*log10(mean(dsdMvdrMean,3).'),'g');
hold off;
```

Listing 5.2 Script for evaluating the filter performances versus the signal forgetting factor.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 60;
setup.forgetNoi = 0.8;
setup.forgetSigGrid = 0.6:0.05:0.9;

setup.filtStrings = {'wiener','mvdr','maxSnr'};

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.forgetSigGrid),
    setup.forgetSig = setup.forgetSigGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = vsTimeDomEnhanceMultChanSignals(...
        signals,setup);

    performance(idx,1) = vsTimeDomMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

%%
display('Measuring performance...');
for idx = 1:length(setup.forgetSigGrid),
    iSnrMean(1,idx) = performance(idx,1).noiseReduction.iSnr.mean;
    oSnrMaxSnrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.mean;
    oSnrWienerMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.mean;
    oSnrMvdrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.mean;

    dsdMaxSnrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.mean;
    dsdWienerMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.mean;
    dsdMvdrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.mean;
end

for idx = 1:length(setup.forgetSigGrid),
    iSnrOverTime(:,idx) = performance(idx,1).noiseReduction.iSnr.overTime;
    oSnrMaxSnrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.overTime;
    oSnrWienerOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.overTime;
    oSnrMvdrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.overTime;

    dsdMaxSnrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.overTime;
    dsdWienerOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.overTime;
    dsdMvdrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.overTime;
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
close all;
figure(1);
plot(10*log10(mean(iSnrMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrMean,3)));
plot(10*log10(mean(oSnrWienerMean,3)));
plot(10*log10(mean(oSnrMvdrMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrMean,3)));
hold on;
plot(10*log10(mean(dsdWienerMean,3)));
plot(10*log10(mean(dsdMvdrMean,3).'),'g');
hold off;
```

Listing 5.3 Script for evaluating the filter performances versus the input signal-to-diffuse-noise ratio.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnrGrid = -10:5:10;
setup.ssnr = 30;

setup.nWin = 80;
setup.forgetNoi = 0.8;
setup.forgetSig = 0.8;

setup.filtStrings = {'wiener','mvdr','maxSnr'};

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.sdnrGrid),
    setup.sdnr = setup.sdnrGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = vsTimeDomEnhanceMultChanSignals(...
        signals,setup);

    performance(idx,1) = vsTimeDomMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

%%
display('Measuring performance...');
for idx = 1:length(setup.sdnrGrid),
    iSnrMean(1,idx) = performance(idx,1).noiseReduction.iSnr.mean;
    oSnrMaxSnrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.mean;
    oSnrWienerMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.mean;
    oSnrMvdrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.mean;

    dsdMaxSnrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.mean;
    dsdWienerMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.mean;
    dsdMvdrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.mean;
end

for idx = 1:length(setup.sdnrGrid),
    iSnrOverTime(:,idx) = performance(idx,1).noiseReduction.iSnr.overTime;
    oSnrMaxSnrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.overTime;
    oSnrWienerOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.overTime;
    oSnrMvdrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.overTime;

    dsdMaxSnrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.overTime;
    dsdWienerOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.overTime;
    dsdMvdrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.overTime;
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
close all;
figure(1);
plot(10*log10(mean(iSnrMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrMean,3)));
plot(10*log10(mean(oSnrWienerMean,3)));
plot(10*log10(mean(oSnrMvdrMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrMean,3)));
hold on;
plot(10*log10(mean(dsdWienerMean,3)));
plot(10*log10(mean(dsdMvdrMean,3).'),'g');
hold off;

figure(3);
plot(10*log10(mean(iSnrOverTime,3)),'k');
```

```
hold on;
plot(10*log10(mean(oSnrMaxSnrOverTime,3)));
plot(10*log10(mean(oSnrWienerOverTime,3)));
plot(10*log10(mean(oSnrMvdrOverTime,3).'),'g');
hold off;

figure(4);
plot(10*log10(mean(dsdMaxSnrOverTime,3)));
hold on;
plot(10*log10(mean(dsdWienerOverTime,3)));
plot(10*log10(mean(dsdMvdrOverTime,3).'),'g');
hold off;
```

Listing 5.4 Script for evaluating the filter performances versus the filter length.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWinGrid = 20:20:100;
setup.forgetNoi = 0.8;
setup.forgetSig = 0.8;

setup.filtStrings = {'wiener','mvdr','maxSnr'};

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.nWinGrid),
    setup.nWin = setup.nWinGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = vsTimeDomEnhanceMultChanSignals(...
        signals,setup);

    performance(idx,1) = vsTimeDomMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

%%
display('Measuring performance...');
for idx = 1:length(setup.nWinGrid),
    iSnrMean(1,idx) = performance(idx,1).noiseReduction.iSnr.mean;
    oSnrMaxSnrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.mean;
    oSnrWienerMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.mean;
    oSnrMvdrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.mean;

    dsdMaxSnrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.mean;
    dsdWienerMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.mean;
    dsdMvdrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.mean;
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
close all;
figure(1);
plot(10*log10(mean(iSnrMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrMean,3)));
plot(10*log10(mean(oSnrWienerMean,3)));
plot(10*log10(mean(oSnrMvdrMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrMean,3)));
hold on;
plot(10*log10(mean(dsdWienerMean,3)));
plot(10*log10(mean(dsdMvdrMean,3).'),'g');
hold off;
```

Listing 5.5 Script for evaluating the filter performances versus the number of microphones.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensorsGrid = 2:6;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 60;
setup.forgetNoi = 0.8;
setup.forgetSig = 0.8;

setup.filtStrings = {'wiener','mvdr','maxSnr'};

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.nSensorsGrid),
    setup.nSensors = setup.nSensorsGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = vsTimeDomEnhanceMultChanSignals(...
        signals,setup);

    performance(idx,1) = vsTimeDomMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

```

```
%%
display('Measuring performance...');
for idx = 1:length(setup.nSensorsGrid),
    iSnrMean(1,idx) = performance(idx,1).noiseReduction.iSnr.mean;
    oSnrMaxSnrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.mean;
    oSnrWienerMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.mean;
    oSnrMvdrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.mean;

    dsdMaxSnrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.mean;
    dsdWienerMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.mean;
    dsdMvdrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.mean;
end

for idx = 1:length(setup.nSensorsGrid),
    iSnrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.iSnr.overTime;
    oSnrMaxSnrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.overTime;
    oSnrWienerOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.overTime;
    oSnrMvdrOverTime(:,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.overTime;

    dsdMaxSnrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.overTime;
    dsdWienerOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.overTime;
    dsdMvdrOverTime(:,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.overTime;
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
close all;
figure(1);
plot(10*log10(mean(iSnrMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrMean,3)));
plot(10*log10(mean(oSnrWienerMean,3)));
plot(10*log10(mean(oSnrMvdrMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrMean,3)));
hold on;
plot(10*log10(mean(dsdWienerMean,3)));
plot(10*log10(mean(dsdMvdrMean,3).'),'g');
hold off;
```

Listing 5.6 Script for evaluating the filter performances versus the tradeoff parameter, μ.

```
clc;clear all;close all;

```

```
addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 60;
setup.forgetNoi = 0.8;
setup.forgetSig = 0.8;

setup.filtStrings = {'wiener','mvdr','maxSnr','trOff'};
setup.trOff.signalRanks = setup.nWin;
setup.trOff.muGrid = 0:0.2:1.2;

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.trOff.muGrid),
    setup.trOff.mu = setup.trOff.muGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = vsTimeDomEnhanceMultChanSignals(...
        signals,setup);

    performance(idx,1) = vsTimeDomMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

%%
display('Measuring performance...');
for idx = 1:length(setup.trOff.muGrid),
    iSnrMean(1,idx) = performance(idx,1).noiseReduction.iSnr.mean;
    oSnrMaxSnrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.mean;
    oSnrWienerMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.mean;
    oSnrMvdrMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.mvdr.mean;
    oSnrTrOffMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.trOff.mean;

    dsdMaxSnrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.mean;
    dsdWienerMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.mean;
    dsdMvdrMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.mvdr.mean;
    dsdTrOffMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.trOff.mean;
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
close all;
figure(1);
plot(10*log10(mean(iSnrMean,3)),'k');
```

```
hold on;
plot(10*log10(mean(oSnrMaxSnrMean,3)));
plot(10*log10(mean(oSnrWienerMean,3)));
plot(10*log10(mean(oSnrMvdrMean,3).'),'g');
plot(10*log10(mean(oSnrTrOffMean,3).'),'m');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrMean,3)));
hold on;
plot(10*log10(mean(dsdWienerMean,3)));
plot(10*log10(mean(dsdMvdrMean,3).'),'g');
plot(10*log10(mean(dsdTrOffMean,3).'),'m');
hold off;
```

Listing 5.7 Script for evaluating the filter performances versus the tradeoff parameter, Q.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 60;
setup.forgetNoi = 0.8;
setup.forgetSig = 0.8;

setup.filtStrings = {'wiener','mvdr','maxSnr','trOff'};

setup.trOff.signalRanks = 1:1:10;
setup.trOff.mu = 1;

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
[signals,setup] = multichannelSignalGenerator(setup);

[simulationData,setup] = vsTimeDomEnhanceMultChanSignals(...
    signals,setup);

performance(1,1) = vsTimeDomMultichannelMeasurePerformance(...
    simulationData,setup,1);

%%
display('Measuring performance...');
iSnrMean(1,1) = performance(1,1).noiseReduction.iSnr.mean;
oSnrMaxSnrMean(1,1) = performance(1,1).noiseReduction.oSnr.maxSnr.mean;
oSnrWienerMean(1,1) = performance(1,1).noiseReduction.oSnr.wiener.mean;
oSnrMvdrMean(1,1) = performance(1,1).noiseReduction.oSnr.mvdr.mean;

dsdMaxSnrMean(1,1) = performance(1,1).signalDistortion.dsd.maxSnr.mean;
dsdWienerMean(1,1) = performance(1,1).signalDistortion.dsd.wiener.mean;
dsdMvdrMean(1,1) = performance(1,1).signalDistortion.dsd.mvdr.mean;
for idx = 1:length(setup.trOff.signalRanks),
    oSnrTrOffMean(1,idx) = ...
        performance(1,1).noiseReduction.oSnr.trOff.mean(:,:,idx);

    dsdTrOffMean(1,idx) = ...
        performance(1,1).signalDistortion.dsd.trOff.mean(:,:,idx);
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
close all;
figure(1);
plot(10*log10(mean(iSnrMean,3))*...
    ones(1,length(setup.trOff.signalRanks)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrMean,3))*...
    ones(1,length(setup.trOff.signalRanks)));
plot(10*log10(mean(oSnrWienerMean,3))*...
    ones(1,length(setup.trOff.signalRanks)));
plot(10*log10(mean(oSnrMvdrMean,3).'*...
    ones(1,length(setup.trOff.signalRanks))),'g');
plot(10*log10(mean(oSnrTrOffMean,3).'),'m');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrMean,3))*...
    ones(1,length(setup.trOff.signalRanks)));
hold on;
plot(10*log10(mean(dsdWienerMean,3))*...
    ones(1,length(setup.trOff.signalRanks)));
plot(10*log10(mean(dsdMvdrMean,3).')*...
    ones(1,length(setup.trOff.signalRanks)),'g');
plot(10*log10(mean(dsdTrOffMean,3).'),'m');
hold off;
```

5.A.2 Functions

Listing 5.8 Function for enhancing noisy signals using the multichannel, variable span filters in the time domain.

```
function [data,setup] = vsTimeDomEnhanceMultChanSignals(signals,setup)

data.raw.sig = signals.clean;
data.raw.noi = signals.noise;
data.raw.obs = signals.observed;

noiCorr1 = eye(setup.nWin);
obsCorr1 = eye(setup.nWin);
sigCorr1 = eye(setup.nWin);

noiCorrAll = eye(setup.nWin*setup.nSensors);
sigMCorr = eye(setup.nWin*setup.nSensors);
sigICorr = eye(setup.nWin*setup.nSensors);

sigCorrAll1 = eye(setup.nWin*setup.nSensors,setup.nWin);

regulPar = 1e-6;
```

```

iter = 1;
frameNdx = 1:setup.nWin;
while frameNdx(end) <= size(data.raw.noi,1),

    noiBlock = data.raw.noi(frameNdx,:);
    sigBlock = data.raw.sig(frameNdx,:);
    obsBlock = data.raw.obs(frameNdx,:);

    noiCorr1 = (1-setup.forgetNoi)*noiCorr1 + ...
        (setup.forgetNoi)*(noiBlock(:,1)*noiBlock(:,1)');
    sigCorr1 = (1-setup.forgetSig)*sigCorr1 + ...
        (setup.forgetSig)*(sigBlock(:,1)*sigBlock(:,1)');

    if rank(sigCorr1)<setup.nWin,
        sigCorr1 = sigCorr1*(1-regulPar)+...
            (regulPar)*trace(sigCorr1)/(setup.nWin)*...
            eye(setup.nWin);
    end


    obsCorr1 = (1-setup.forgetSig)*obsCorr1 + ...
        (setup.forgetSig)*(obsBlock(:,1)*obsBlock(:,1)');

    noiCorrAll = (1-setup.forgetNoi)*noiCorrAll + ...
        (setup.forgetNoi)*(noiBlock(:)*noiBlock(:)');

    sigCorrAll1 = (1-setup.forgetSig)*sigCorrAll1 + ...
        (setup.forgetSig)*(sigBlock(:)*sigBlock(:,1)');

    gamSig = sigCorrAll1/sigCorr1;
    sigM = gamSig*sigBlock(:,1);
    sigI = sigBlock(:) - sigM;

    sigMCorr = (1-setup.forgetSig)*sigMCorr + ...
        (setup.forgetSig)*(sigM*sigM');
    sigICorr = (1-setup.forgetSig)*sigICorr + ...
        (setup.forgetSig)*(sigI*sigI');
    intNoiCorr = sigICorr + noiCorrAll;

    if rank(intNoiCorr)<setup.nWin*setup.nSensors,
        intNoiCorr = intNoiCorr*(1-regulPar)+...
            (regulPar)*trace(intNoiCorr)/(setup.nWin*setup.nSensors)*...
            eye(setup.nWin*setup.nSensors);
    end

    %GEVD
    [geigVec,geigVal] = jeig(sigMCorr,intNoiCorr,1);

    % filter signals
    data.raw.sigFrames(:,iter) = sigBlock(:,1);
    data.raw.obsFrames(:,iter) = obsBlock(:,1);
    data.raw.noiFrames(:,iter) = noiBlock(:,1);

    for iFiltStr=1:length(setup.filtStrings),
        switch char(setup.filtStrings(iFiltStr)),
            case 'maxSnr',
            % max snr filt
            HmaxSnr = sigCorr1*gamSig'*geigVec(:,1)*...
                (geigVal(1,1)\...
                geigVec(:,1)');
            % filtering
            data.maxSnr.sigFrames(:,iter) = HmaxSnr*sigBlock(:);
            data.maxSnr.obsFrames(:,iter) = HmaxSnr*obsBlock(:);
            data.maxSnr.noiFrames(:,iter) = HmaxSnr*noiBlock(:);

            case 'wiener',
            Hw = sigCorr1*gamSig'*geigVec(:,1:setup.nWin)*...
                ((eye(setup.nWin)+geigVal(1:setup.nWin,1:setup.nWin))\...
                geigVec(:,1:setup.nWin)');
            % filtering
            data.wiener.sigFrames(:,iter) = Hw*sigBlock(:);
            data.wiener.obsFrames(:,iter) = Hw*obsBlock(:);
            data.wiener.noiFrames(:,iter) = Hw*noiBlock(:);

            case 'mvdr',
            % mvdr
            geigValTmp = geigVal(1:setup.nWin,1:setup.nWin);
            if rank(geigValTmp)<setup.nWin,
                geigValTmp = geigValTmp*(1-regulPar)+...
                    (regulPar)*trace(geigValTmp)/(setup.nWin)*...
                    eye(setup.nWin);
            end
            Hmvdr = sigCorr1*gamSig'*geigVec(:,1:setup.nWin)*...
                ((geigValTmp)\...
                geigVec(:,1:setup.nWin)');
            % filtering
            data.mvdr.sigFrames(:,iter) = Hmvdr*sigBlock(:);
            data.mvdr.obsFrames(:,iter) = Hmvdr*obsBlock(:);
            data.mvdr.noiFrames(:,iter) = Hmvdr*noiBlock(:);

            case 'trOff',
            % trade off
            for iRanks = 1:length(setup.trOff.signalRanks),
                geigValTmp = geigVal(1:setup.trOff.signalRanks(iRanks),...
                1:setup.trOff.signalRanks(iRanks));
                if rank(geigValTmp)<setup.nWin,
                    geigValTmp = geigValTmp*(1-regulPar)+...
                        (regulPar)*trace(geigValTmp)/...
                        setup.trOff.signalRanks(iRanks)*...
                        eye(setup.trOff.signalRanks(iRanks));
                end
                HtrOff(:,:,iRanks) = sigCorr1*gamSig'*...
                    geigVec(:,1:setup.trOff.signalRanks(iRanks))*...
                ((setup.trOff.mu*eye(setup.trOff.signalRanks(iRanks))...
                +geigValTmp)\...
                geigVec(:,1:setup.trOff.signalRanks(iRanks))');
            end

            % filtering
            for iRanks = 1:length(setup.trOff.signalRanks),
                data.trOff.sigFrames(:,iter,iRanks) = ...
                    HtrOff(:,:,iRanks)*sigBlock(:);
                data.trOff.obsFrames(:,iter,iRanks) = ...
                    HtrOff(:,:,iRanks)*obsBlock(:);
                data.trOff.noiFrames(:,iter,iRanks) = ...
                    HtrOff(:,:,iRanks)*noiBlock(:);
            end
        end
    end

    frameNdx = frameNdx + setup.nWin/2;
    iter = iter + 1;
end

for iFiltStr=1:length(setup.filtStrings),
    switch char(setup.filtStrings(iFiltStr)),
        case 'maxSnr',
        data.maxSnr.sig = mergeSignalFrames(data.maxSnr.sigFrames);
        data.maxSnr.obs = mergeSignalFrames(data.maxSnr.obsFrames);
        data.maxSnr.noi = mergeSignalFrames(data.maxSnr.noiFrames);
        case 'wiener',
        data.wiener.sig = mergeSignalFrames(data.wiener.sigFrames);
        data.wiener.obs = mergeSignalFrames(data.wiener.obsFrames);
        data.wiener.noi = mergeSignalFrames(data.wiener.noiFrames);
        case 'mvdr',
        data.mvdr.sig = mergeSignalFrames(data.mvdr.sigFrames);
        data.mvdr.obs = mergeSignalFrames(data.mvdr.obsFrames);
        data.mvdr.noi = mergeSignalFrames(data.mvdr.noiFrames);
        case 'trOff',
        for iRanks = 1:length(setup.trOff.signalRanks),
            data.trOff.sig(:,iRanks) = mergeSignalFrames(...
                data.trOff.sigFrames(:,:,iRanks));
            data.trOff.obs(:,iRanks) = mergeSignalFrames(...
                data.trOff.obsFrames(:,:,iRanks));
            data.trOff.noi(:,iRanks) = mergeSignalFrames(...
                data.trOff.noiFrames(:,:,iRanks));
        end
    end
end
end

function signal = mergeSignalFrames(signalFrames)
frameLength = size(signalFrames,1);
nFrames = size(signalFrames,2);
signal = zeros((nFrames+1)*frameLength/2,1);

window = hanning(frameLength,'periodic');

prevFrameEnd = zeros(frameLength,1);
ndx = 1:frameLength;
for ii = 1:nFrames,

    thisFrame = prevFrameEnd + window.*signalFrames(:,ii);
    signal(ndx,1) = thisFrame;

    prevFrameEnd = [thisFrame(frameLength/2+1:end,1);...
        zeros(frameLength/2,1);];
    ndx = ndx + frameLength/2;
end
end
```

Listing 5.9 Function for measuring the performance of multichannel, variable span filters in the time domain.

```
function [performance] = vsTimeDomMultichannelMeasurePerformance(...
    data,setup,flagFromSignals)

nBlockSkip = 10;

if flagFromSignals,

    % raw signal powers
    [performance.power.raw.sigPow,performance.power.raw.sigPowMean] = ...
    calculatePowers(data.raw.sigFrames,nBlockSkip);

    % raw noise powers
    [performance.power.raw.noiPow,performance.power.raw.noiPowMean] = ...
    calculatePowers(data.raw.noiFrames,nBlockSkip);

    [performance.noiseReduction.iSnr.overTime,...
        performance.noiseReduction.iSnr.mean] ...
        = measurePerformance(performance,'raw');

    for iFiltStr=1:length(setup.filtStrings),
        switch char(setup.filtStrings(iFiltStr)),
            case 'maxSnr',
            % signal and noise powers (max snr)
            [performance.power.maxSnr.sigPow,...
                performance.power.maxSnr.sigPowMean] = ...
                calculatePowers(data.maxSnr.sigFrames,nBlockSkip);

            [performance.power.maxSnr.noiPow,...
                performance.power.maxSnr.noiPowMean] = ...
                calculatePowers(data.maxSnr.noiFrames,nBlockSkip);

            [performance.noiseReduction.oSnr.maxSnr.overTime,...
                performance.noiseReduction.oSnr.maxSnr.mean,...
                performance.signalDistortion.dsd.maxSnr.overTime,...
                performance.signalDistortion.dsd.maxSnr.mean] ...
                = measurePerformance(performance,...
                char(setup.filtStrings(iFiltStr)));
            case 'wiener',
            % signal and noise powers (wiener)
            [performance.power.wiener.sigPow,...
                performance.power.wiener.sigPowMean] = ...
                calculatePowers(data.wiener.sigFrames,nBlockSkip);

            [performance.power.wiener.noiPow,...
                performance.power.wiener.noiPowMean] = ...
                calculatePowers(data.wiener.noiFrames,nBlockSkip);

            [performance.noiseReduction.oSnr.wiener.overTime,...
                performance.noiseReduction.oSnr.wiener.mean,...
                performance.signalDistortion.dsd.wiener.overTime,...
                performance.signalDistortion.dsd.wiener.mean] ...
                = measurePerformance(performance,char(...
                setup.filtStrings(iFiltStr)));
            case 'mvdr',
            % signal and noise powers (mvdr)
            [performance.power.mvdr.sigPow,...
                performance.power.mvdr.sigPowMean] = ...
                calculatePowers(data.mvdr.sigFrames,nBlockSkip);

            [performance.power.mvdr.noiPow,...
                performance.power.mvdr.noiPowMean] = ...
                calculatePowers(data.mvdr.noiFrames,nBlockSkip);

            [performance.noiseReduction.oSnr.mvdr.overTime,...
                performance.noiseReduction.oSnr.mvdr.mean,...
                performance.signalDistortion.dsd.mvdr.overTime,...
                performance.signalDistortion.dsd.mvdr.mean] ...
                = measurePerformance(performance,char(...
                setup.filtStrings(iFiltStr)));
            case 'trOff',
            % signal and noise powers (tr off)
            for iRank = 1:size(data.trOff.sigFrames,3);
                [performance.power.trOff.sigPow(:,:,iRank),...
                    performance.power.trOff.sigPowMean(:,:,iRank)] = ...
                    calculatePowers(data.trOff.sigFrames(:,:,iRank),...
                    nBlockSkip);

                [performance.power.trOff.noiPow(:,:,iRank),...
                    performance.power.trOff.noiPowMean(:,:,iRank)] = ...
                    calculatePowers(data.trOff.noiFrames(:,:,iRank),...
                    nBlockSkip);

        [performance.noiseReduction.oSnr.trOff.overTime(:,:,iRank),...
            performance.noiseReduction.oSnr.trOff.mean(:,:,iRank),...
            performance.signalDistortion.dsd.trOff.overTime(:,:,iRank),...
            performance.signalDistortion.dsd.trOff.mean(:,:,iRank)] ...
            = measurePerformance(performance,char(...
            setup.filtStrings(iFiltStr)),iRank);
            end
        end
```

```
    end
end

end

function [pow,powMean] = ...
    calculatePowers(data,nSkip)

[winLen,nFrames] = size(data);
for iFrames = 1:nFrames,
    pow(1,iFrames) = norm(data(:,iFrames))/winLen;
end

powMean = mean(pow(nSkip+1:end));
end

function [snrOverTime,snrMean,dsdOverTime,dsdMean] ...
    = measurePerformance(performance,filtStr,iRank)

if nargin < 3,
    iRank = 1;
end

snrOverTime = eval(['performance.power.',filtStr,'.sigPow(:,:,iRank)'])...
    ./eval(['performance.power.',filtStr,'.noiPow(:,:,iRank)']);

snrMean = eval(['performance.power.',filtStr,'.sigPowMean(:,:,iRank)'])...
    ./eval(['performance.power.',filtStr,'.noiPowMean(:,:,iRank)']);

dsdOverTime = performance.power.raw.sigPow...
    ./eval(['performance.power.',filtStr,'.sigPow(:,:,iRank)']);

dsdMean = performance.power.raw.sigPowMean...
    ./eval(['performance.power.',filtStr,'.sigPowMean(:,:,iRank)']);

end
```

Chapter 6
Multichannel Signal Enhancement in the STFT Domain

In Chapter 4, we exploited the temporal (and spectral) information from a single sensor signal to derive different variable span (VS) linear filters for noise reduction in the STFT domain. In this chapter, we exploit the spatial information available from signals picked up by a determined number of microphones at different positions in the acoustics space in order to mitigate the noise effect. The processing is performed in the STFT domain.

6.1 Signal Model and Problem Formulation

We consider the conventional signal model with M sensors explained in Chapter 5 [1], [2], i.e.,

$$\begin{aligned} y_m(t) &= g_m(t) * s(t) + v_m(t) \\ &= x_m(t) + v_m(t),\ m = 1, 2, \ldots, M. \end{aligned} \tag{6.1}$$

Using the STFT, (6.1) can be rewritten in the time-frequency domain as [3]

$$Y_m(k, n) = X_m(k, n) + V_m(k, n),\ m = 1, 2, \ldots, M, \tag{6.2}$$

where the zero-mean complex random variables $Y_m(k, n)$, $X_m(k, n)$, and $V_m(k, n)$ are the STFTs of $y_m(t)$, $x_m(t)$, and $v_m(t)$, respectively, at frequency bin $k \in \{0, 1, \ldots, K - 1\}$ and time frame n. It is more convenient to write the M STFT-domain sensor signals in a vector notation:

$$\begin{aligned} \mathbf{y}(k, n) &= \begin{bmatrix} Y_1(k, n)\ Y_2(k, n) \cdots\ Y_M(k, n) \end{bmatrix}^T \\ &= \mathbf{x}(k, n) + \mathbf{v}(k, n), \end{aligned} \tag{6.3}$$

J. Benesty et al., *Signal Enhancement with Variable Span Linear Filters*,
Springer Topics in Signal Processing 7, DOI 10.1007/978-981-287-739-0_6

where $\mathbf{x}(k,n)$ and $\mathbf{v}(k,n)$ are defined similarly to $\mathbf{y}(k,n)$. Since $X_m(k,n)$ and $V_m(k,n)$ are uncorrelated by assumption, we deduce that the correlation matrix of $\mathbf{y}(k,n)$ is

$$\begin{aligned}\mathbf{\Phi}_{\mathbf{y}}(k,n) &= E\left[\mathbf{y}(k,n)\mathbf{y}^H(k,n)\right] \\ &= \mathbf{\Phi}_{\mathbf{x}}(k,n) + \mathbf{\Phi}_{\mathbf{v}}(k,n),\end{aligned} \tag{6.4}$$

where $\mathbf{\Phi}_{\mathbf{x}}(k,n)$ and $\mathbf{\Phi}_{\mathbf{v}}(k,n)$ are the correlation matrices of $\mathbf{x}(k,n)$ and $\mathbf{v}(k,n)$, respectively.

Very often, in the speech enhancement problem with microphone arrays, it is assumed that

$$X_m(k,n) = G_m(k)S(k,n),\ m = 1,2,\ldots,M, \tag{6.5}$$

where $G_m(k)$ and $S(k,n)$ are the STFTs of $g_m(t)$ and $s(t)$, respectively. When the equality in (6.5) holds, it is easy to check that the rank of $\mathbf{\Phi}_{\mathbf{x}}(k,n)$ is equal to 1. Moreover, most of the noise reduction algorithms derived in the literature are based on (6.5), which is a valid expression only when the analysis window of the STFT is infinitely long. However, for some very good reasons, this length is always taken rather short [4]. As a result, the rank of $\mathbf{\Phi}_{\mathbf{x}}(k,n)$ is no longer equal to 1 but rather equal to a positive integer between 1 and M. Therefore, it is of great interest to study noise reduction algorithms in this context as most of them will be different from the conventional ones.

In this chapter, the interframe correlation is taken into account in order to improve filtering since speech signals are correlated at successive time frames with the STFT [3]. Considering N of these successive frames, we can rewrite the observations as

$$\begin{aligned}\underline{\mathbf{y}}(k,n) &= \left[\,\mathbf{y}^T(k,n)\ \mathbf{y}^T(k,n-1)\ \cdots\ \mathbf{y}^T(k,n-N+1)\,\right]^T \\ &= \underline{\mathbf{x}}(k,n) + \underline{\mathbf{v}}(k,n),\end{aligned} \tag{6.6}$$

where $\underline{\mathbf{x}}(k,n)$ and $\underline{\mathbf{v}}(k,n)$ resemble $\underline{\mathbf{y}}(k,n)$ of length MN. The correlation matrix of $\underline{\mathbf{y}}(k,n)$ is then

$$\begin{aligned}\mathbf{\Phi}_{\underline{\mathbf{y}}}(k,n) &= E\left[\underline{\mathbf{y}}(k,n)\underline{\mathbf{y}}^H(k,n)\right] \\ &= \mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n) + \mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n),\end{aligned} \tag{6.7}$$

where $\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)$ and $\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)$ are the correlation matrices of $\underline{\mathbf{x}}(k,n)$ and $\underline{\mathbf{v}}(k,n)$, respectively, whose ranks are assumed to be equal to $P < MN$ and MN.

Sensor 1 is chosen as the reference. Therefore, the multichannel signal enhancement problem in the STFT domain considered in this study is one of recovering $X_1(k,n)$ from $\underline{\mathbf{y}}(k,n)$ the best way we can.

The two Hermitian matrices $\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)$ and $\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)$ can be jointly diagonalized as follows [5]:

$$\mathbf{B}^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\mathbf{B}(k,n) = \mathbf{\Lambda}(k,n), \tag{6.8}$$

$$\mathbf{B}^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\mathbf{B}(k,n) = \mathbf{I}_{MN}, \tag{6.9}$$

where $\mathbf{B}(k,n)$ is a full-rank square matrix (of size $MN \times MN$), $\mathbf{\Lambda}(k,n)$ is a diagonal matrix whose main elements are real and nonnegative, and $\mathbf{I}_{MN}$ is the $MN \times MN$ identity matrix. Furthermore, $\mathbf{\Lambda}(k,n)$ and $\mathbf{B}(k,n)$ are the eigenvalue and eigenvector matrices, respectively, of $\mathbf{\Phi}_{\underline{\mathbf{v}}}^{-1}(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)$, i.e.,

$$\mathbf{\Phi}_{\underline{\mathbf{v}}}^{-1}(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\mathbf{B}(k,n) = \mathbf{B}(k,n)\mathbf{\Lambda}(k,n). \tag{6.10}$$

Since the rank of the matrix $\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)$ is assumed to be equal to P, the eigenvalues of $\mathbf{\Phi}_{\underline{\mathbf{v}}}^{-1}(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)$ can be ordered as $\lambda_1(k,n) \geq \lambda_2(k,n) \geq \cdots \geq \lambda_P(k,n) > \lambda_{P+1}(k,n) = \cdots = \lambda_{MN}(k,n) = 0$. In other words, the first P and last $MN - P$ eigenvalues of the matrix product $\mathbf{\Phi}_{\underline{\mathbf{v}}}^{-1}(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)$ are positive and exactly zero, respectively. We also denote by $\underline{\mathbf{b}}_1(k,n), \underline{\mathbf{b}}_2(k,n), \ldots, \underline{\mathbf{b}}_{MN}(k,n)$, the corresponding eigenvectors. Therefore, the noisy signal correlation matrix can also be diagonalized as

$$\mathbf{B}^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{y}}}(k,n)\mathbf{B}(k,n) = \mathbf{\Lambda}(k,n) + \mathbf{I}_{MN}. \tag{6.11}$$

We can interpret the joint diagonalization as a particular spatiotemporal filterbank decomposition with MN subbands, where the noise is whitened and equalized in all subbands. From (6.11), we can also observe that the spatiotemporal SNR in the ith subband is equal to $\lambda_i(k,n)$.

We cloture this section with the definition of the subband input SNR:

$$\text{iSNR}(k,n) = \frac{\phi_{X_1}(k,n)}{\phi_{V_1}(k,n)}, \tag{6.12}$$

where $\phi_{X_1}(k,n) = E\left[|X_1(k,n)|^2\right]$ and $\phi_{V_1}(k,n) = E\left[|V_1(k,n)|^2\right]$ are the variances of $X_1(k,n)$ and $V_1(k,n)$, respectively.

6.2 Direct Approach for Signal Enhancement

In the direct approach, we estimate the desired signal, $X_1(k,n)$, directly from the observation signal vector, $\underline{\mathbf{y}}(k,n)$, through a filtering operation, i.e.,

$$Z(k,n) = \underline{\mathbf{h}}^H(k,n)\underline{\mathbf{y}}(k,n), \tag{6.13}$$

where $Z(k,n)$ is the estimate of $X_1(k,n)$,

$$\underline{\mathbf{h}}(k,n) = \left[\, \mathbf{h}^T(k,n)\; \mathbf{h}^T(k,n-1) \cdots \mathbf{h}^T(k,n-N+1) \,\right]^T \tag{6.14}$$

is a long complex-valued filter of length MN, and $\mathbf{h}(k, n-i)$ is a filter of length M containing all the complex gains applied to the sensor outputs at frequency bin k and time frame $n-i$. It is always possible to write $\underline{\mathbf{h}}(k,n)$ in a basis formed from the vectors $\underline{\mathbf{b}}_i(k,n),\ i = 1, 2, \ldots, MN$, i.e.,

$$\underline{\mathbf{h}}(k,n) = \mathbf{B}(k,n)\underline{\mathbf{a}}(k,n), \tag{6.15}$$

where the components of

$$\underline{\mathbf{a}}(k,n) = \begin{bmatrix} A_1(k,n) \ A_2(k,n) \ \cdots \ A_{MN}(k,n) \end{bmatrix}^T \tag{6.16}$$

are the coordinates of $\underline{\mathbf{h}}(k,n)$ in the new basis. Now, instead of estimating the coefficients of $\underline{\mathbf{h}}(k,n)$ as in conventional approaches, we can estimate, equivalently, the coordinates $A_i(k,n),\ i = 1, 2, \ldots, MN$. Substituting (6.15) into (6.13), we get

$$Z(k,n) = \underline{\mathbf{a}}^H(k,n)\mathbf{B}^H(k,n)\underline{\mathbf{x}}(k,n) + \underline{\mathbf{a}}^H(k,n)\mathbf{B}^H(k,n)\underline{\mathbf{v}}(k,n). \tag{6.17}$$

We deduce that the variance of $Z(k,n)$ is

$$\phi_Z(k,n) = \underline{\mathbf{a}}^H(k,n)\boldsymbol{\Lambda}(k,n)\underline{\mathbf{a}}(k,n) + \underline{\mathbf{a}}^H(k,n)\underline{\mathbf{a}}(k,n). \tag{6.18}$$

Let us decompose the vector $\underline{\mathbf{a}}(k,n)$ into two subvectors:

$$\underline{\mathbf{a}}(k,n) = \begin{bmatrix} \mathbf{a}'^T(k,n) \ \mathbf{a}''^T(k,n) \end{bmatrix}^T, \tag{6.19}$$

where $\mathbf{a}'(k,n)$ is a vector of length P containing the first P coefficients of $\underline{\mathbf{a}}(k,n)$ and $\mathbf{a}''(k,n)$ is a vector of length $MN-P$ containing the last $MN-P$ coefficients of $\underline{\mathbf{a}}(k,n)$. In the same way, we have

$$\mathbf{B}(k,n) = \begin{bmatrix} \mathbf{B}'(k,n) \ \mathbf{B}''(k,n) \end{bmatrix}, \tag{6.20}$$

$$\boldsymbol{\Lambda}'(k,n) = \mathrm{diag}\left[\lambda_1(k,n), \lambda_2(k,n), \ldots, \lambda_P(k,n)\right], \tag{6.21}$$

where $\mathbf{B}'(k,n)$ is a matrix of size $MN \times P$ containing the first P columns of $\mathbf{B}(k,n)$ and $\mathbf{B}''(k,n)$ is a matrix of size $MN \times (MN-P)$ containing the last $MN-P$ columns of $\mathbf{B}(k,n)$. It is clear from (6.18) that $\underline{\mathbf{a}}^H(k,n)\underline{\mathbf{a}}(k,n) = \mathbf{a}'^H(k,n)\mathbf{a}'(k,n) + \mathbf{a}''^H(k,n)\mathbf{a}''(k,n)$ represents the residual noise. Many optimal noise reduction filters with at most P constraints will lead to $\mathbf{a}''(k,n) = \mathbf{0}_{(MN-P)\times 1}$ since there is no speech in the directions $\mathbf{B}''(k,n)$. Therefore, we can simplify our problem and force $\mathbf{a}''(k,n) = \mathbf{0}_{(MN-P)\times 1}$, so that (6.17) becomes

$$\begin{aligned} Z(k,n) &= \mathbf{a}'^H(k,n)\mathbf{B}'^H(k,n)\underline{\mathbf{x}}(k,n) + \mathbf{a}'^H(k,n)\mathbf{B}^H(k,n)\underline{\mathbf{v}}(k,n) \\ &= X_{\mathrm{fd}}(k,n) + V_{\mathrm{rn}}(k,n), \end{aligned} \tag{6.22}$$

where $X_{\text{fd}}(k,n)$ is the filtered desired signal and $V_{\text{rn}}(k,n)$ is the residual noise. Now, only $\mathbf{a}'(k,n)$ needs to be determined. The variance of $Z(k,n)$ is then

$$\phi_Z(k,n) = \mathbf{a}'^H(k,n)\mathbf{\Lambda}'(k,n)\mathbf{a}'(k,n) + \mathbf{a}'^H(k,n)\mathbf{a}'(k,n). \tag{6.23}$$

From (6.23), it is easy to find that the subband output SNR is

$$\begin{aligned} \text{oSNR}\left[\mathbf{a}'(k,n)\right] &= \frac{\mathbf{a}'^H(k,n)\mathbf{\Lambda}'(k,n)\mathbf{a}'(k,n)}{\mathbf{a}'^H(k,n)\mathbf{a}'(k,n)} \\ &= \frac{\sum_{p=1}^{P}\lambda_p(k,n)\left|A_p(k,n)\right|^2}{\sum_{p=1}^{P}\left|A_p(k,n)\right|^2} \end{aligned} \tag{6.24}$$

and it can be shown that

$$\text{oSNR}\left[\mathbf{a}'(k,n)\right] \le \lambda_1(k,n). \tag{6.25}$$

Another important measure is the subband noise reduction factor, which is defined as

$$\xi_{\text{nr}}\left[\mathbf{a}'(k,n)\right] = \frac{\phi_{V_1}(k,n)}{\sum_{p=1}^{P}\left|A_p(k,n)\right|^2}. \tag{6.26}$$

The noise reduction factor should be lower bounded by 1; otherwise, the filter $\underline{\mathbf{h}}(k,n)$ amplifies the noise.

In practice, most multichannel noise reduction filters distort the desired signal. In order to quantify the level of this distortion, we define the subband desired signal reduction factor as

$$\xi_{\text{sr}}\left[\mathbf{a}'(k,n)\right] = \frac{\phi_{X_1}(k,n)}{\sum_{p=1}^{P}\lambda_p(k,n)\left|A_p(k,n)\right|^2}. \tag{6.27}$$

In order to have no distortion, we must have $\xi_{\text{sr}}\left[\mathbf{a}'(k,n)\right] = 1$. Distortion occurs when $\xi_{\text{sr}}\left[\mathbf{a}'(k,n)\right] > 1$.

It is clear from (6.24) that the subband output SNR is maximized if and only if $A_1(k,n) \neq 0$ and $A_2(k,n) = \cdots = A_P(k,n) = 0$. As a consequence, the maximum SNR filter is

$$\underline{\mathbf{h}}_{\text{max}}(k,n) = A_1(k,n)\underline{\mathbf{b}}_1(k,n), \tag{6.28}$$

where $A_1(k,n) \neq 0$ is an arbitrary complex number, whose optimal value needs to be found.

The MSE corresponding to distortion is defined as

$$J_{\text{ds}}\left[\mathbf{a}'(k,n)\right] = E\left[\left|X_1(k,n) - \mathbf{a}'^H(k,n)\mathbf{B}'^H(k,n)\underline{\mathbf{x}}(k,n)\right|^2\right]. \tag{6.29}$$

Substituting (6.28) into (6.29) and minimizing the resulting expression with respect to $A_1(k,n)$, we find the maximum SNR filter with minimum distortion:

$$\underline{\mathbf{h}}_{\max}(k,n) = \frac{\underline{\mathbf{b}}_1(k,n)\underline{\mathbf{b}}_1^H(k,n)}{\lambda_1(k,n)} \mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}, \tag{6.30}$$

where $\underline{\mathbf{i}}$ is the first column of $\mathbf{I}_{MN}$. It can be verified that

$$\text{oSNR}\left[\underline{\mathbf{h}}_{\max}(k,n)\right] = \lambda_1(k,n) \tag{6.31}$$

and

$$\text{oSNR}\left[\underline{\mathbf{h}}(k,n)\right] \leq \text{oSNR}\left[\underline{\mathbf{h}}_{\max}(k,n)\right], \ \forall \underline{\mathbf{h}}(k,n). \tag{6.32}$$

The MVDR filter (see also Section 6.3) is obtained by minimizing $J_{\text{ds}}\left[\mathbf{a}'(k,n)\right]$. We get

$$\begin{aligned}\mathbf{a}'_{\text{MVDR}}(k,n) &= \left[\mathbf{B}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\mathbf{B}'(k,n)\right]^{-1}\mathbf{B}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}} \\ &= \mathbf{\Lambda}'^{-1}(k,n)\mathbf{B}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}.\end{aligned} \tag{6.33}$$

Therefore, the MVDR filter is

$$\begin{aligned}\underline{\mathbf{h}}_{\text{MVDR}}(k,n) &= \mathbf{B}'(k,n)\mathbf{a}'_{\text{MVDR}}(k,n) \\ &= \sum_{p=1}^{P} \frac{\underline{\mathbf{b}}_p(k,n)\underline{\mathbf{b}}_p^H(k,n)}{\lambda_p(k,n)} \mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}.\end{aligned} \tag{6.34}$$

From the obvious relationship between the maximum SNR and MVDR filters, we propose a class of minimum distortion (MD) filters:

$$\underline{\mathbf{h}}_{\text{MD},Q}(k,n) = \sum_{q=1}^{Q} \frac{\underline{\mathbf{b}}_q(k,n)\underline{\mathbf{b}}_q^H(k,n)}{\lambda_q(k,n)} \mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}, \tag{6.35}$$

where $1 \leq Q \leq P$. We observe that for $Q = 1$ and $Q = P$, we obtain $\underline{\mathbf{h}}_{\text{MD},1}(k,n) = \underline{\mathbf{h}}_{\max}(k,n)$ and $\underline{\mathbf{h}}_{\text{MD},P}(k,n) = \underline{\mathbf{h}}_{\text{MVDR}}(k,n)$, respectively. We should have

$$\text{oSNR}\left[\underline{\mathbf{h}}_{\text{MD},1}(k,n)\right] \geq \text{oSNR}\left[\underline{\mathbf{h}}_{\text{MD},2}(k,n)\right] \geq \cdots \geq \text{oSNR}\left[\underline{\mathbf{h}}_{\text{MD},P}(k,n)\right] \tag{6.36}$$

and

$$\xi_{\text{sr}}\left[\underline{\mathbf{h}}_{\text{MD},1}(k,n)\right] \geq \xi_{\text{sr}}\left[\underline{\mathbf{h}}_{\text{MD},2}(k,n)\right] \geq \cdots \geq \xi_{\text{sr}}\left[\underline{\mathbf{h}}_{\text{MD},P}(k,n)\right]. \tag{6.37}$$

We define the error signal between the estimated and desired signals at frequency bin k as

$$\mathcal{E}(k,n) = Z(k,n) - X_1(k,n). \tag{6.38}$$

Then, the MSE is

$$\begin{aligned} J\left[\mathbf{a}'(k,n)\right] &= E\left[\left|\mathcal{E}(k,n)\right|^2\right] \\ &= \phi_{X_1}(k,n) - \underline{\mathbf{i}}^T \mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\mathbf{B}'(k,n)\mathbf{a}'(k,n) \\ &\quad - \mathbf{a}'^H(k,n)\mathbf{B}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}} \\ &\quad + \mathbf{a}'^H(k,n)\left[\mathbf{\Lambda}'(k,n) + \mathbf{I}_{MN}\right]\mathbf{a}'(k,n) \\ &= J_{\mathrm{ds}}\left[\mathbf{a}'(k,n)\right] + J_{\mathrm{rs}}\left[\mathbf{a}'(k,n)\right], \end{aligned} \tag{6.39}$$

where

$$J_{\mathrm{rs}}\left[\mathbf{a}'(k,n)\right] = \mathbf{a}'^H(k,n)\mathbf{a}'(k,n) \tag{6.40}$$

is the MSE of the residual noise. The minimization of $J\left[\mathbf{a}'(k,n)\right]$ leads to

$$\mathbf{a}'_{\mathrm{W}}(k,n) = \left[\mathbf{\Lambda}'(k,n) + \mathbf{I}_{MN}\right]^{-1} \mathbf{B}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}. \tag{6.41}$$

We deduce that the Wiener filter confined in the desired signal-plus-noise subspace is[1]

$$\begin{aligned} \underline{\mathbf{h}}_{\mathrm{W}}(k,n) &= \mathbf{B}'(k,n)\mathbf{a}'_{\mathrm{W}}(k,n) \\ &= \sum_{p=1}^{P} \frac{\underline{\mathbf{b}}_p(k,n)\underline{\mathbf{b}}_p^H(k,n)}{1 + \lambda_p(k,n)} \mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}. \end{aligned} \tag{6.42}$$

We can see that the MVDR and Wiener filters are very close to each other; they only differ by the weighting function, which strongly depends on the spatiotemporal subband SNR. In the first case, it is equal to $\lambda_p^{-1}(k,n)$ while in the second case it is equal to $\left[1 + \lambda_p(k,n)\right]^{-1}$. The MVDR filter will always extract the desired speech from all directions while it will be more attenuated with the Wiener filter. We should have

$$\mathrm{oSNR}\left[\underline{\mathbf{h}}_{\mathrm{W}}(k,n)\right] \geq \mathrm{oSNR}\left[\underline{\mathbf{h}}_{\mathrm{MVDR}}(k,n)\right] \tag{6.43}$$

and

$$\xi_{\mathrm{sr}}\left[\underline{\mathbf{h}}_{\mathrm{W}}(k,n)\right] \geq \xi_{\mathrm{sr}}\left[\underline{\mathbf{h}}_{\mathrm{MVDR}}(k,n)\right]. \tag{6.44}$$

Another interesting approach that can compromise between noise reduction and desired signal distortion is the tradeoff filter obtained by

[1] This Wiener filter is different from the one proposed in [3] within the same context.

Table 6.1 Optimal linear filters for multichannel signal enhancement in the STFT domain.

Maximum SNR:	$\underline{\mathbf{h}}_{\max}(k,n) = \frac{\underline{\mathbf{b}}_1(k,n)\underline{\mathbf{b}}_1^H(k,n)}{\lambda_1(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}$
Wiener:	$\underline{\mathbf{h}}_{\mathrm{W}}(k,n) = \sum_{p=1}^{P} \frac{\underline{\mathbf{b}}_p(k,n)\underline{\mathbf{b}}_p^H(k,n)}{1+\lambda_p(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}$
MVDR:	$\underline{\mathbf{h}}_{\mathrm{MVDR}}(k,n) = \sum_{p=1}^{P} \frac{\underline{\mathbf{b}}_p(k,n)\underline{\mathbf{b}}_p^H(k,n)}{\lambda_p(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}$
MD,Q:	$\underline{\mathbf{h}}_{\mathrm{MD},Q}(k,n) = \sum_{q=1}^{Q} \frac{\underline{\mathbf{b}}_q(k,n)\underline{\mathbf{b}}_q^H(k,n)}{\lambda_q(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}$
Tradeoff:	$\underline{\mathbf{h}}_{\mathrm{T},\mu}(k,n) = \sum_{p=1}^{P} \frac{\underline{\mathbf{b}}_p(k,n)\underline{\mathbf{b}}_p^H(k,n)}{\mu+\lambda_p(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}$
General Tradeoff:	$\underline{\mathbf{h}}_{\mu,Q}(k,n) = \sum_{q=1}^{Q} \frac{\underline{\mathbf{b}}_q(k,n)\underline{\mathbf{b}}_q^H(k,n)}{\mu+\lambda_q(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}$

$$\min_{\mathbf{a}'(k,n)} J_{\mathrm{ds}}\left[\mathbf{a}'(k,n)\right] \quad \text{subject to} \quad J_{\mathrm{rs}}\left[\mathbf{a}'(k,n)\right] = \beta\phi_{V_1}(k,n), \tag{6.45}$$

where $0 \leq \beta \leq 1$, to ensure that filtering achieves some degree of noise reduction. We easily find that the optimal filter is

$$\underline{\mathbf{h}}_{\mathrm{T},\mu}(k,n) = \sum_{p=1}^{P} \frac{\underline{\mathbf{b}}_p(k,n)\underline{\mathbf{b}}_p^H(k,n)}{\mu+\lambda_p(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}, \tag{6.46}$$

where $\mu \geq 0$ is a Lagrange multiplier. Clearly, for $\mu = 0$ and $\mu = 1$, we get the MVDR and Wiener filters, respectively.

From all what we have seen so far, we can propose a very general tradeoff noise reduction filter:

$$\underline{\mathbf{h}}_{\mu,Q}(k,n) = \sum_{q=1}^{Q} \frac{\underline{\mathbf{b}}_q(k,n)\underline{\mathbf{b}}_q^H(k,n)}{\mu+\lambda_q(k,n)}\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{i}}. \tag{6.47}$$

This form encompasses all known optimal filters. Indeed, it is clear that

- $\underline{\mathbf{h}}_{0,1}(k,n) = \underline{\mathbf{h}}_{\max}(k,n)$,
- $\underline{\mathbf{h}}_{1,P}(k,n) = \underline{\mathbf{h}}_{\mathrm{W}}(k,n)$,
- $\underline{\mathbf{h}}_{0,P}(k,n) = \underline{\mathbf{h}}_{\mathrm{MVDR}}(k,n)$,
- $\underline{\mathbf{h}}_{0,Q}(k,n) = \underline{\mathbf{h}}_{\mathrm{MD},Q}(k,n)$,
- $\underline{\mathbf{h}}_{\mu,P}(k,n) = \underline{\mathbf{h}}_{\mathrm{T},\mu}(k,n)$.

In Table 6.1, we summarize all optimal filters studied in this section.

6.3 Indirect Approach for Signal Enhancement

The indirect approach resembles the general sidelobe canceler (GSC) structure [6]. It consists of two successive stages. First, we apply the filter, $\underline{\mathbf{h}}'(k,n)$ of length MN, to the observation signal vector, $\underline{\mathbf{y}}(k,n)$. The filter output is then

$$Z'(k,n) = \underline{\mathbf{h}}'^H(k,n)\underline{\mathbf{x}}(k,n) + \underline{\mathbf{h}}'^H(k,n)\underline{\mathbf{v}}(k,n). \tag{6.48}$$

The subband output SNR is

$$\text{oSNR}_{\widehat{V}_1}\left[\underline{\mathbf{h}}'(k,n)\right] = \frac{\underline{\mathbf{h}}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)\underline{\mathbf{h}}'(k,n)}{\underline{\mathbf{h}}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\underline{\mathbf{h}}'(k,n)}. \tag{6.49}$$

Then, we find $\underline{\mathbf{h}}'(k,n)$ that minimizes $\text{oSNR}_{\widehat{V}_1}\left[\underline{\mathbf{h}}'(k,n)\right]$. It is easy to see that the solution is

$$\underline{\mathbf{h}}'(k,n) = \mathbf{B}''(k,n)\mathbf{a}''(k,n), \tag{6.50}$$

where $\mathbf{B}''(k,n)$ and $\mathbf{a}''(k,n)$ are defined in the previous section. With (6.50), $\text{oSNR}_{\widehat{V}_1}\left[\underline{\mathbf{h}}'(k,n)\right] = 0$ and $Z'(k,n)$ can be seen as the estimate of the noise, $V_1(k,n)$.

In the second stage, we estimate the desired signal, $X_1(k,n)$, as

$$\begin{aligned}\widehat{X}_1(k,n) &= Y_1(k,n) - Z'(k,n) \\ &= X_1(k,n) + V_1(k,n) - \mathbf{a}''^H(k,n)\mathbf{B}''^H(k,n)\underline{\mathbf{v}}(k,n).\end{aligned} \tag{6.51}$$

From the previous expression, we observe that the desired signal is not distorted no matter the choice of $\mathbf{a}''(k,n)$. Also from (6.51), we define the MSE of the residual noise and the subband output SNR as, respectively,

$$\begin{aligned}J_{\text{rs}}\left[\mathbf{a}''(k,n)\right] &= E\left[\left|V_1(k,n) - \mathbf{a}''^H(k,n)\mathbf{B}''^H(k,n)\underline{\mathbf{v}}(k,n)\right|^2\right] \\ &= \phi_{V_1}(k,n) - \underline{\mathbf{i}}^T\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\mathbf{B}''(k,n)\mathbf{a}''(k,n) \\ &\quad - \mathbf{a}''^H(k,n)\mathbf{B}''^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\underline{\mathbf{i}} + \mathbf{a}''^H(k,n)\mathbf{a}''(k,n)\end{aligned} \tag{6.52}$$

and

$$\text{oSNR}\left[\mathbf{a}''(k,n)\right] = \frac{\phi_{X_1}(k,n)}{J_{\text{rs}}\left[\mathbf{a}''(k,n)\right]}. \tag{6.53}$$

Obviously, maximizing the subband output SNR is equivalent to minimizing the residual noise. The minimization of $J_{\text{rs}}\left[\mathbf{a}''(k,n)\right]$ leads to the MVDR filter:

$$\mathbf{a}''_{\text{MVDR}}(k,n) = \mathbf{B}''^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\underline{\mathbf{i}}. \tag{6.54}$$

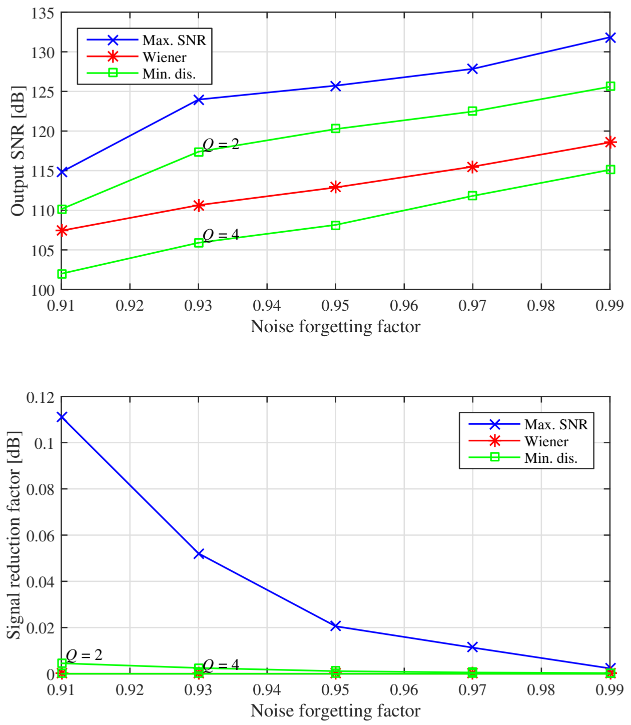


Fig. 6.1 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the noise forgetting factor, $\xi_{\rm n}$.

We deduce that the residual noise and subband output SNR with the MVDR filter are, respectively,

$$\begin{aligned} J_{\rm rs}\left[\mathbf{a}''_{\rm MVDR}(k,n)\right] &= \phi_{V_1}(k,n) - \underline{\mathbf{i}}^T \mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\mathbf{B}''(k,n)\mathbf{B}''^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\underline{\mathbf{i}} \\ &= \underline{\mathbf{i}}^T \mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\mathbf{B}'(k,n)\mathbf{B}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\underline{\mathbf{i}} \end{aligned} \tag{6.55}$$

and

$$\text{oSNR}\left[\mathbf{a}''_{\rm MVDR}(k,n)\right] = \frac{\phi_{X_1}(k,n)}{\underline{\mathbf{i}}^T \mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\mathbf{B}'(k,n)\mathbf{B}'^H(k,n)\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)\underline{\mathbf{i}}} \geq \text{iSNR}(k,n). \tag{6.56}$$

Of course, we can do much more with this indirect approach like in Chapters 2 and 3, but we leave this to the reader.

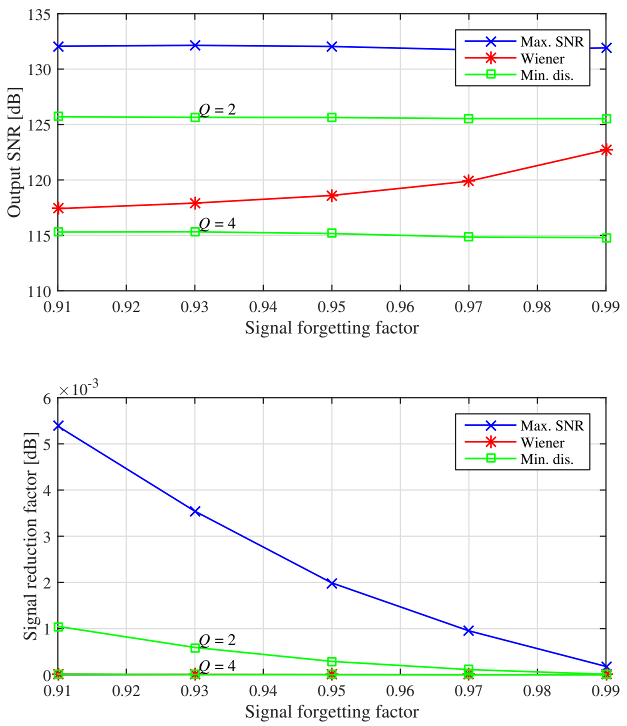
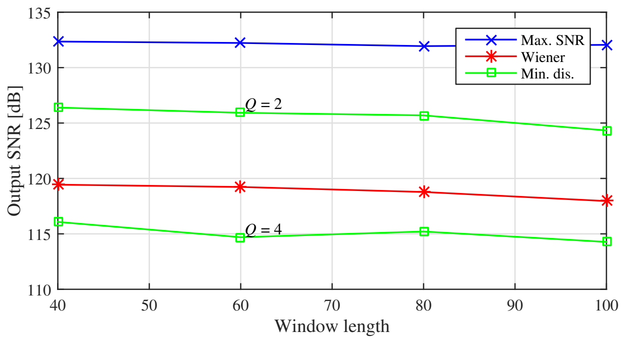


Fig. 6.2 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the signal forgetting factor, ξ_{n}.

6.4 Experimental Results

In this section, we then present the evaluation of the multichannel, variable span and STFT-based enhancement filters presented in Section 6.2. For these evaluations, we considered the enhancement of different reverberant speech signals contaminated by diffuse babble noise and white sensor noise. The desired signals and noise types as well as the room setup was the same as in the previous chapter (see Section 5.5 for a more detailed description). Using signal and noise mixtures, we then conducted evaluations of the filters in terms of their output SNRs and signal reduction factors. In each of the evaluations, the performance of the filters were measured over time and averaged, hence, the depicted results in the remainder of the chapter are the time-averaged performance measures for different simulation settings.

In order to design the optimal variable span based filters for multichannel enhancement in the STFT domain, we need knowledge about the signal cor-



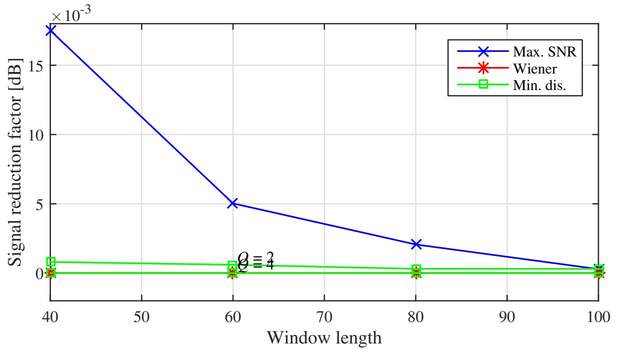
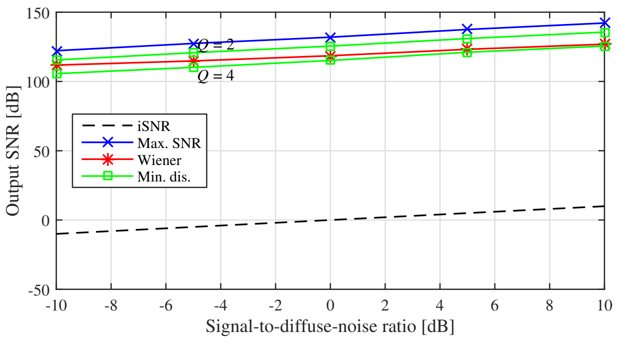


Fig. 6.3 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the STFT window length.

relation, $\mathbf{\Phi}_{\underline{\mathbf{x}}}(k,n)$, and noise correlation, $\mathbf{\Phi}_{\underline{\mathbf{v}}}(k,n)$, matrices. The focus herein is not on noise or signal estimation, but rather on the relative performance of the presented filter designs, so the necessary statistics are estimated directly from the desired signal and the noise signal. To estimate the statistics in practice, techniques such as VAD, minimum statistics or sequential methods could be pursued [7], [8], [9], [10]. The statistics estimation directly from the speech and noise signals, respectively, was conducted recursively using the following general equation for approximating the correlation matrix, $\mathbf{\Phi}_{\underline{\mathbf{a}}}(k,n)$, of a vector $\underline{\mathbf{a}}(k,n)$:

$$\widehat{\mathbf{\Phi}}_{\underline{\mathbf{a}}}(k,n) = (1-\xi)\widehat{\mathbf{\Phi}}_{\underline{\mathbf{a}}}(k,n-1) + \xi\underline{\mathbf{a}}(k,n)\underline{\mathbf{a}}^H(k,n), \tag{6.57}$$

where ξ is the forgetting factor, and $\widehat{\mathbf{\Phi}}_{\underline{\mathbf{a}}}(k,n)$ denotes an estimate of $\mathbf{\Phi}_{\underline{\mathbf{a}}}(k,n)$ at frequency bin, k, and time instance n. In Section 6.A and Appendix A, the MATLAB code used for evaluating the filters is found.



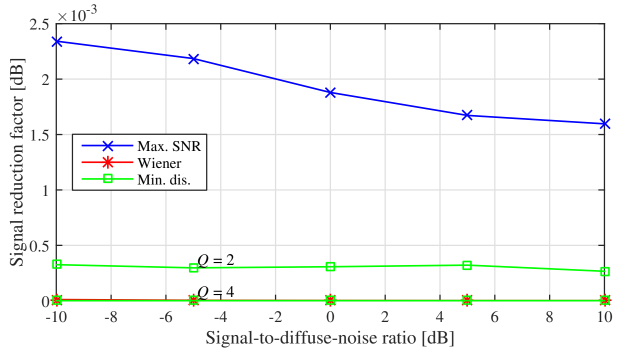


Fig. 6.4 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the input SDNR.

In the evaluations, we used two forgetting factors. A signal forgetting factor, ξ_{s}, for the recursive update of the signal correlation matrix, and a noise forgetting factor, ξ_{n}, for the noise correlation matrix estimation. Therefore, we first investigated the influence of these forgetting factors on the filter performances. For this evaluation, we considered a scenario where the input signal-to-diffuse-noise ratio (SDNR) was 0 dB, the input signal-to-sensor-noise ratio (SSNR) was 30 dB, the number of microphones was $M = 3$, and the temporal filter length was $N = 4$. Moreover, the STFT was computed from blocks of 80 samples using a rectangular window and an FFT length of 128. The blocks were overlapping by 50 %, and, after enhancement, the blocks were therefore combined using overlap-add with Hanning windows. The choices of 1) using a rectangular window, 2) using an STFT window length of 80, 3) using an FFT length of 128, and 4) using an overlap-add procedure to obtain the final output were also applied in the remaining evaluations in this chapter.

With the simulation setup described above, we then evaluated the filter performances, by fixing the signal forgetting factor to 0.95, and varying the noise forgetting factor. This resulted in the plot in Fig. 6.1. The noise forgetting factor should clearly be chosen relatively high, since the output SNR is increasing for an increasing noise forgetting factor. This is the trend for all of the filters in the evaluation (i.e., the maximum SNR, Wiener, and minimum distortion[2] filters). This is further motivated by the fact, that the signal reduction factors of the filters also decrease for an increasing noise forgetting factor. Using the same simulation setup, we then fixed the noise forgetting factor to 0.99 and varied the signal forgetting factor to obtain the results in Fig. 6.2. In this case, the output SNR is, generally, slightly decreasing for the maximum SNR and minimum distortion filters when the signal forgetting factor increases. This trend is exactly opposite to that of the Wiener filter, which shows an increasing noise reduction performance. Regarding distortion, the signal reduction factors decrease for all the filters when the signal forgetting factor grows, and the Wiener and minimum distortion ($Q = 4$) filters are nearly distortionless. Based on these two evaluations versus the forgetting factors, and since we use the same forgetting factors for all filter designs, we chose $\xi_{\mathrm{s}} = 0.95$ and $\xi_{\mathrm{n}} = 0.99$ for the remaining evaluations in this chapter.

We also evaluated the filter performances versus the window length used in computation of the STFT. This was done for an input SDNR of 0 dB and an input SSNR of 30 dB with $M = 3$ microphones and a temporal filter length of $N = 4$. The outcome of this evaluation is depicted in Fig. 6.3. The output SNR does not change much versus the window length, but it is generally slightly decreasing for all the filters when the window length increases. There is less distortion (i.e., the signal reduction factor decreases), on the other hand, when the window length is increasing. The window length should therefore be chosen as a compromise between noise reduction and signal distortion.

In the next evaluation, we investigated the filter performances versus the input SDNR using the following simulation setup: the input SSNR was 30 dB, the number of microphones was $M = 3$, and the temporal filter length was $N = 4$. Using this setup, the filters were then evaluated versus the input SDNR, yielding the results in Fig. 6.4. The output SNR of all filters increase with an increase in the input SDNR. However, the gain between input SDNR and output SNR decreases for the Wiener filter for a growing input SDNR. The distortion of the filters remain almost constant for different diffuse noise levels, except for the maximum SNR filter, which shows a clear decrease in signal reduction factor for an increasing input SDNR.

The filter performances also depend on the temporal filter length, and this dependence was considered in the following evaluation. The setup for this evaluation was as follows: the input SDNR and SSNR were 0 dB and 30 dB,

[2] Note that the performance of the minimum distortion filter was investigated for different assumed signal subspace ranks, Q, in all evaluations.

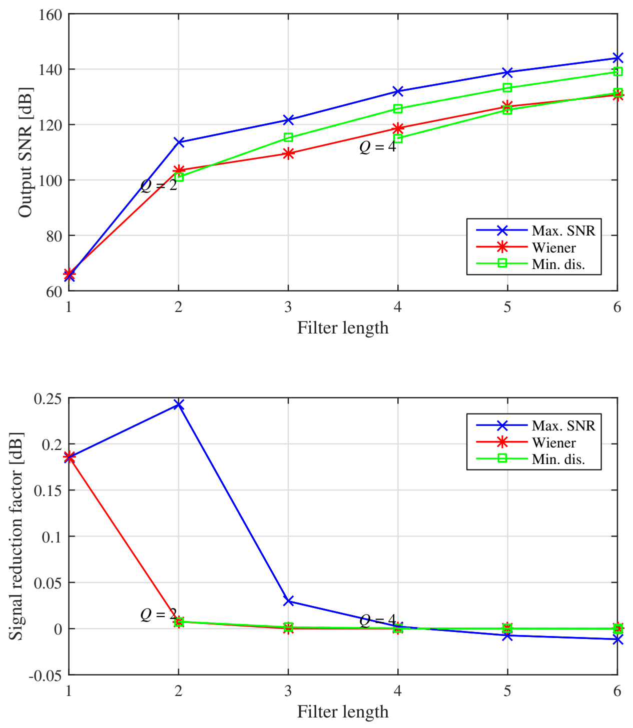
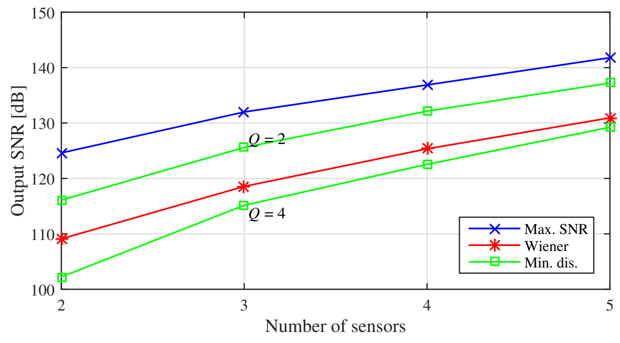


Fig. 6.5 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the temporal filter length, N.

respectively, and the number of microphone was $M = 3$, while the temporal filter length was varied. This resulted in the plots in Fig. 6.5. We can see that the filter length has a significant influence on both the noise reduction and signal distortion performance of all filter designs. In general, all filters have a higher output SNR when the filter length is longer. Moreover, the filters also have a decreasing signal reduction factor for an increasing filter length except between filter lengths of 1 and 2 for the maximum SNR filter.

We then considered an evaluation of the filter performances versus different numbers of sensors. This evaluation was conducted using a scenario with an input SDNR of 0 dB, and input SSNR of 30 dB, and a temporal filter length of $N = 4$. The number of sensors was then varied to obtain the results in Fig. 6.6. Clearly, the noise reduction performance of all filters depend strongly on the number of sensors. The output SNRs of all filters increase when the number of sensors increase. More specifically, by increasing the number of sensors from 2 to 5, an increase in output SNR of 15–20 dB can be obtained for all



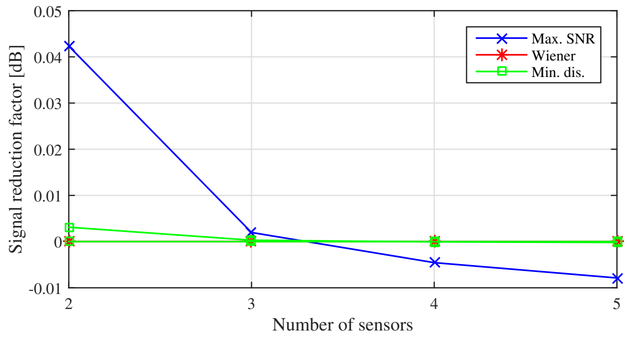
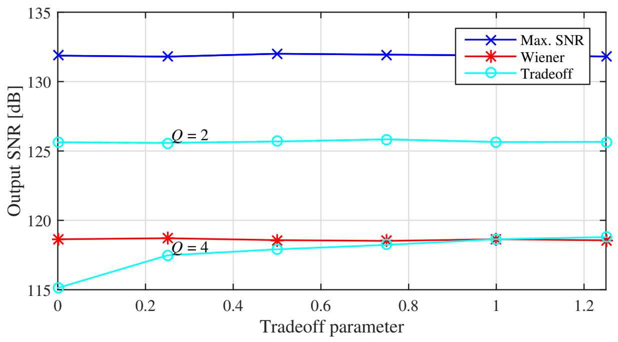


Fig. 6.6 Evaluation of the performance of the maximum SNR, Wiener, and minimum distortion filters versus the number of sensors, M.

filters. Moreover, the signal reduction factor decreases when the number of sensors is increasing. The minimum distortion and Wiener filters have small amounts of distortion, i.e., a signal reduction factor close to 0 dB, for all numbers of sensors.

Finally, we investigated the performance of the tradeoff filter for different tuning parameters. The general tradeoff filter has two tuning parameters: the tradeoff parameters, μ, and the assumed signal subspace rank, Q. To investigate the influence of those on the filter performance, we considered a scenario with input SDNRs and SSNRs of 0 dB and 30 dB, respectively, $M = 3$ microphones, and a temporal filter length of $N = 4$. The tradeoff filters were then evaluated for different μ's and Q's, yielding the results in Fig. 6.7. It is clear to see that the tradeoff filter strongly depends on the assumed signal subspace rank, Q. If this parameter is decreased, we get a great increase in output SNR, but it comes at the cost of a higher signal reduction factor. Moreover, for a fixed Q, we can also obtain an increase



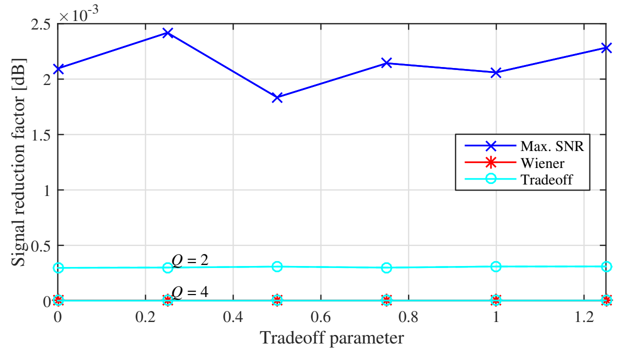


Fig. 6.7 Evaluation of the performance of the maximum SNR, Wiener, and tradeoff filters versus the assumed signal subspace rank, Q, and the tradeoff parameter, μ.

in output SNR by increasing the tradeoff parameter μ. The increase in the signal reduction factor for such a change is only small, at least in the scenario considered here. If we take the example with $Q = 4$, the output SNR of the tradeoff filter is approximately 4 dB lower than the Wiener filter's for $\mu = 0$, whereas it slightly outperforms the Wiener filter, in terms of output SNR, for $\mu > 1$.

References

1. J. Benesty, J. Chen, and Y. Huang, *Microphone Array Signal Processing*. Berlin, Germany: Springer-Verlag, 2008.
2. M. Brandstein and D. B. Ward, Eds., *Microphone Arrays: Signal Processing Techniques and Applications*. Berlin, Germany: Springer-Verlag, 2001.

3. J. Benesty, J. Chen, and E. Habets, *Speech Enhancement in the STFT Domain.* Springer Briefs in Electrical and Computer Engineering, 2011.
4. Y. Avargel and I. Cohen, "System identification in the short-time Fourier transform domain with crossband filtering," *IEEE Trans. Audio, Speech, Language Process.*, vol. 15, pp. 1305–1319, May 2007.
5. J. N. Franklin, *Matrix Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1968.
6. L. J. Griffiths and C. W. Jim, "An alternative approach to linearly constrained adaptive beamforming," *IEEE Trans. Antennas & Propagation*, vol. AP-30, pp. 27–34, Jan. 1982.
7. J. S. Sohn, N. S. Kim, and W. Sung, "A statistical model-based voice activity detection," *IEEE Signal Process. Letters*, vol. 6, pp. 1–3, Jan. 1999.
8. R. Martin, "Noise power spectral density estimation based on optimal smoothing and minimum statistics," *IEEE Trans. Speech Audio Process.*, vol. 9, pp. 504–512, Jul. 2001.
9. T. Gerkmann and R. Hendriks, "Unbiased MMSE-based noise power estimation with low complexity and low tracking delay," *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, pp. 1383–1393, May 2012.
10. E. J. Diethorn, "Subband noise reduction methods for speech enhancement," in *Audio Signal Processing for Next-Generation Multimedia Communication Systems*, Y. Huang and J.Benesty, eds., Boston, MA, USA: Kluwer, 2004, Chapter 4, pp. 91–115.

6.A MATLAB Code

6.A.1 Main Scripts

Listing 6.1 Script for evaluating the filter performances versus the noise forgetting factor.

```matlab
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 80;
setup.nFft = 2^nextpow2(setup.nWin);
setup.nFilt = 4;
setup.forgetNoiGrid = 0.1:0.1:0.9;
setup.forgetSig = 0.95;

setup.filtStrings = {'wiener','minDis','maxSnr'};
setup.minDis.signalRanks = [2,4,6];

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
```

```
for idx = 1:length(setup.forgetNoiGrid),
    setup.forgetNoi = setup.forgetNoiGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = stftEnhanceMultChanSignals(signals,setup);

    performance(idx,1) = stftMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

display('Measuring performance...');
for idx = 1:length(setup.forgetNoiGrid),
    iSnrFbMean(1,idx) = performance(idx,1).noiseReduction.iSnr.fbMean;
    oSnrMaxSnrFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.fbMean;
    oSnrWienerFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.fbMean;
    oSnrMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).noiseReduction.oSnr.minDis.fbMean);

    dsdMaxSnrFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.fbMean;
    dsdWienerFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.fbMean;
    dsdMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).signalDistortion.dsd.minDis.fbMean);
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(dsdWienerFbMean,3)));
plot(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;
```

Listing 6.2 Script for evaluating the filter performances versus the signal forgetting factor.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 80;
setup.nFft = 2^nextpow2(setup.nWin);
setup.nFilt = 4;
setup.forgetNoi = 0.99;
setup.forgetSigGrid = 0.1:0.1:0.9;

setup.filtStrings = {'wiener','minDis','maxSnr'};
setup.minDis.signalRanks = [2,4,6];

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.forgetSigGrid),
    setup.forgetSig = setup.forgetSigGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = stftEnhanceMultChanSignals(signals,setup);

    performance(idx,1) = stftMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

display('Measuring performance...');
for idx = 1:length(setup.forgetSigGrid),
    iSnrFbMean(1,idx) = performance(idx,1).noiseReduction.iSnr.fbMean;
    oSnrMaxSnrFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.fbMean;
    oSnrWienerFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.fbMean;
    oSnrMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).noiseReduction.oSnr.minDis.fbMean);

    dsdMaxSnrFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.fbMean;
    dsdWienerFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.fbMean;
    dsdMinDisFbMean(:,idx) = ...
        squeeze(performance(idx,1).signalDistortion.dsd.minDis.fbMean);
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(dsdWienerFbMean,3)));
plot(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;
```

Listing 6.3 Script for evaluating the filter performances versus the window length.

```
clc;clear all;close all;
```

```

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWinGrid = 40:20:100;
setup.nFilt = 4;
setup.forgetNoi = 0.99;
setup.forgetSig = 0.95;

setup.filtStrings = {'wiener','minDis','maxSnr'};
setup.minDis.signalRanks = [2,4,6];

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.nWinGrid),
    setup.nWin = setup.nWinGrid(idx);
    setup.nFft = 2^nextpow2(setup.nWin);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = stftEnhanceMultChanSignals(signals,setup);

    performance(idx,1) = stftMultichannelMeasurePerformance(...
        simulationData,setup,1);
end
%%
display('Measuring performance...');
for idx = 1:length(setup.nWinGrid),
    iSnrFbMean(1,idx) = performance(idx,1).noiseReduction.iSnr.fbMean;
    oSnrMaxSnrFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.fbMean;
    oSnrWienerFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.fbMean;
    oSnrMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).noiseReduction.oSnr.minDis.fbMean);

    dsdMaxSnrFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.fbMean;
    dsdWienerFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.fbMean;
    dsdMinDisFbMean(:,idx) = ...
        squeeze(performance(idx,1).signalDistortion.dsd.minDis.fbMean);
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%%
figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;
```

```

figure(2);
plot(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(dsdWienerFbMean,3)));
plot(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;
```

Listing 6.4 Script for evaluating the filter performances versus the input signal-to-diffuse-noise ratio.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnrGrid = -10:5:10;
setup.ssnr = 30;

setup.nWin = 80;
setup.nFft = 2^nextpow2(setup.nWin);
setup.nFilt = 4;
setup.forgetNoi = 0.99;
setup.forgetSig = 0.95;

setup.filtStrings = {'wiener','minDis','maxSnr'};
setup.minDis.signalRanks = [2,4,6];

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.sdnrGrid),
    setup.sdnr = setup.sdnrGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = stftEnhanceMultChanSignals(signals,setup);

    performance(idx,1) = stftMultichannelMeasurePerformance(...
        simulationData,setup,1);
end

display('Measuring performance...');
for idx = 1:length(setup.sdnrGrid),
    iSnrFbMean(1,idx) = performance(idx,1).noiseReduction.iSnr.fbMean;
    oSnrMaxSnrFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.fbMean;
    oSnrWienerFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.fbMean;
    oSnrMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).noiseReduction.oSnr.minDis.fbMean);

    dsdMaxSnrFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.fbMean;
    dsdWienerFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.fbMean;
    dsdMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).signalDistortion.dsd.minDis.fbMean);
```

```
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plot
figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(dsdWienerFbMean,3)));
plot(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;
```

Listing 6.5 Script for evaluating the filter performances versus the filter length.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 80;
setup.nFft = 2^nextpow2(setup.nWin);
setup.nFiltGrid = 1:6;
setup.minDis.signalRankMax = 6;
setup.forgetNoi = 0.99;
setup.forgetSig = 0.95;

setup.filtStrings = {'wiener','minDis','maxSnr'};

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.nFiltGrid),
    setup.nFilt = setup.nFiltGrid(idx);
    setup.minDis.signalRanks = 1:min([...
        setup.nFilt*setup.nSensors,setup.minDis.signalRankMax]);

    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = stftEnhanceMultChanSignals(signals,setup);

    performance(idx,1) = stftMultichannelMeasurePerformance(...
        simulationData,setup,1);
end
%%
```

```
display('Measuring performance...');
for idx = 1:length(setup.nFiltGrid),
    iSnrFbMean(1,idx) = performance(idx,1).noiseReduction.iSnr.fbMean;
    oSnrMaxSnrFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.fbMean;
    oSnrWienerFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.fbMean;

    dsdMaxSnrFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.fbMean;
    dsdWienerFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.fbMean;

    for nn = 1:idx,
        oSnrMinDisFbMean(nn,idx) = ...
            (performance(idx,1).noiseReduction.oSnr.minDis.fbMean(nn));

        dsdMinDisFbMean(nn,idx) = ...
            (performance(idx,1).signalDistortion.dsd.minDis.fbMean(nn));
    end
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plots
close all;
figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(dsdWienerFbMean,3)));
plot(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;
```

Listing 6.6 Script for evaluating the filter performances versus the number of microphones.

```
clc;clear all;close all;

addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensorsGrid = 2:5;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 80;
setup.nFft = 2^nextpow2(setup.nWin);
setup.nFilt = 4;
setup.forgetNoi = 0.99;
setup.forgetSig = 0.95;

setup.filtStrings = {'wiener','minDis','maxSnr'};
setup.minDis.signalRanks = [2,4,6];

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.nSensorsGrid),
    setup.nSensors = setup.nSensorsGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = stftEnhanceMultChanSignals(signals,setup);

    performance(idx,1) = stftMultichannelMeasurePerformance(...
        simulationData,setup,1);
end
%%
display('Measuring performance...');
for idx = 1:length(setup.nSensorsGrid),
    iSnrFbMean(1,idx) = performance(idx,1).noiseReduction.iSnr.fbMean;
    oSnrMaxSnrFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.fbMean;
    oSnrWienerFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.fbMean;
    oSnrMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).noiseReduction.oSnr.minDis.fbMean);

    dsdMaxSnrFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.fbMean;
    dsdWienerFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.fbMean;
    dsdMinDisFbMean(:,idx) = squeeze(...
        performance(idx,1).signalDistortion.dsd.minDis.fbMean);
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%% plots
figure(1);
plot(10*log10(mean(iSnrFbMean,3)),'k');
hold on;
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrMinDisFbMean,3).'),'g');
hold off;

figure(2);
plot(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(dsdWienerFbMean,3)));
plot(10*log10(mean(dsdMinDisFbMean,3).'),'g');
hold off;
```

Listing 6.7 Script for evaluating the filter performances versus the tradeoff parameter, μ.

```
clc;clear all;close all;

```

```
addpath([cd,'\..\nonstationaryMultichanNoiseGenerator']);
addpath([cd,'\..\functions\']);

setup.sensorDistance = 0.1;
setup.speedOfSound = 343;
setup.nSensors = 3;
setup.noiseField = 'spherical';
setup.reverbTime = 0.2;

setup.sdnr = 0;
setup.ssnr = 30;

setup.nWin = 80;
setup.nFft = 2^nextpow2(setup.nWin);
setup.nFilt = 4;
setup.forgetNoi = 0.99;
setup.forgetSig = 0.95;

setup.filtStrings = {'wiener','trOff','maxSnr'};
setup.trOff.muGrid = 0:0.25:1.25;
setup.trOff.signalRanks = [2,4,6];

display(['Running script: ',mfilename]);
display(' ');

display('Enhancing...');
for idx = 1:length(setup.trOff.muGrid),
    setup.trOff.mu = setup.trOff.muGrid(idx);
    [signals,setup] = multichannelSignalGenerator(setup);

    [simulationData,setup] = stftEnhanceMultChanSignals(signals,setup);

    performance(idx,1) = stftMultichannelMeasurePerformance(...
        simulationData,setup,1);
end
%%
display('Measuring performance...');
for idx = 1:length(setup.trOff.muGrid),
    iSnrFbMean(1,idx) = performance(idx,1).noiseReduction.iSnr.fbMean;
    oSnrMaxSnrFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.maxSnr.fbMean;
    oSnrWienerFbMean(1,idx) = ...
        performance(idx,1).noiseReduction.oSnr.wiener.fbMean;
    oSnrTrOffFbMean(:,idx) = ...
        squeeze(performance(idx,1).noiseReduction.oSnr.trOff.fbMean);

    dsdMaxSnrFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.maxSnr.fbMean;
    dsdWienerFbMean(1,idx) = ...
        performance(idx,1).signalDistortion.dsd.wiener.fbMean;
    dsdTrOffFbMean(:,idx) = ...
        squeeze(performance(idx,1).signalDistortion.dsd.trOff.fbMean);
end

%% save
% dateString = datestr(now,30);
%
% save([mfilename,'_',dateString,'.mat']);

%%
close all
figure(1);
plot(10*log10(mean(oSnrMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(oSnrWienerFbMean,3)));
plot(10*log10(mean(oSnrTrOffFbMean,3).'),'g');
hold off;
```

```

figure(2);
plot(10*log10(mean(dsdMaxSnrFbMean,3)));
hold on;
plot(10*log10(mean(dsdWienerFbMean,3)));
plot(10*log10(mean(dsdTrOffFbMean,3).'),'g');
hold off;
```

6.A.2 Functions

Listing 6.8 Function for enhancing noisy signals using the multichannel, variable span filters in the STFT domain.

```
function [data,setup] = stftEnhanceMultChanSignals(signals,setup)

if isfield(setup,'noiseOrSignalStat')==0,
    setup.noiseOrSignalStat = 'signal';
end

data.raw.sig = signals.clean;
data.raw.noi = signals.noise;
data.raw.obs = signals.observed;

for iSens = 1:setup.nSensors,
    % apply stft
    [data.raw.sigStft(:,:,iSens),freqGrid,timeGrid,...
        data.raw.sigBlocks(:,:,iSens)] ...
        = stftBatch(signals.clean(:,iSens),...
    setup.nWin,setup.nFft,setup.sampFreq);
    [data.raw.noiStft(:,:,iSens),~,~,data.raw.noiBlocks(:,:,iSens)] ...
        = stftBatch(signals.noise(:,iSens),setup.nWin,setup.nFft,...
        setup.sampFreq);
    [data.raw.obsStft(:,:,iSens),~,~,data.raw.obsBlocks(:,:,iSens)] ...
        = stftBatch(signals.observed(:,iSens),setup.nWin,setup.nFft,...
        setup.sampFreq);
end
[setup.nFreqs,setup.nFrames,~] = size(data.raw.sigStft);


noiCorr = repmat(eye(setup.nFilt*setup.nSensors),1,1,setup.nFreqs);
obsCorr = repmat(eye(setup.nFilt*setup.nSensors),1,1,setup.nFreqs);
sigCorr = repmat(eye(setup.nFilt*setup.nSensors),1,1,setup.nFreqs);
for iFrame = 1:setup.nFrames,

    for iFreq=1:setup.nFreqs,
        noiBlock = zeros(setup.nFilt*setup.nSensors,1);
        sigBlock = zeros(setup.nFilt*setup.nSensors,1);
        obsBlock = zeros(setup.nFilt*setup.nSensors,1);
        for iSens = 1:setup.nSensors,
            if iFrame<setup.nFilt,
                noiBlock((1:setup.nFilt)+(iSens-1)*setup.nFilt,1) ...
                    = [data.raw.noiStft(iFreq,iFrame:-1:1,iSens).';...
                    zeros(setup.nFilt-iFrame,1)];
                sigBlock((1:setup.nFilt)+(iSens-1)*setup.nFilt,1) ...
                    = [data.raw.sigStft(iFreq,iFrame:-1:1,iSens).';...
                    zeros(setup.nFilt-iFrame,1)];
                obsBlock((1:setup.nFilt)+(iSens-1)*setup.nFilt,1) ...
                    = [data.raw.obsStft(iFreq,iFrame:-1:1,iSens).';...
                    zeros(setup.nFilt-iFrame,1)];
            else
```

```
                noiBlock((1:setup.nFilt)+(iSens-1)*setup.nFilt,1) ...
                    = data.raw.noiStft(iFreq,iFrame:-1:iFrame-...
                    setup.nFilt+1,iSens).';
                sigBlock((1:setup.nFilt)+(iSens-1)*setup.nFilt,1) ...
                    = data.raw.sigStft(iFreq,iFrame:-1:iFrame-...
                    setup.nFilt+1,iSens).';
                obsBlock((1:setup.nFilt)+(iSens-1)*setup.nFilt,1) ...
                    = data.raw.obsStft(iFreq,iFrame:-1:iFrame-...
                    setup.nFilt+1,iSens).';
            end
        end

        noiCorr(:,:,iFreq) = (1-setup.forgetNoi)*noiCorr(:,:,iFreq)...
            + (setup.forgetNoi)*(noiBlock*noiBlock');
        obsCorr(:,:,iFreq) = (1-setup.forgetSig)*obsCorr(:,:,iFreq)...
            + (setup.forgetSig)*(obsBlock*obsBlock');

        switch setup.noiseOrSignalStat,
            case 'signal',
                sigCorr(:,:,iFreq) = (1-setup.forgetSig)*...
                    sigCorr(:,:,iFreq) + (setup.forgetSig)*...
                    (sigBlock*sigBlock');
            case 'noise',
                sigCorr(:,:,iFreq) = obsCorr(:,:,iFreq) - ...
                    noiCorr(:,:,iFreq);
        end

        for iFilt = 1:setup.nSensors*setup.nFilt,
            noiCorr(iFilt,iFilt,iFreq)=real(noiCorr(iFilt,iFilt,iFreq));
            obsCorr(iFilt,iFilt,iFreq)=real(obsCorr(iFilt,iFilt,iFreq));
            sigCorr(iFilt,iFilt,iFreq)=real(sigCorr(iFilt,iFilt,iFreq));
        end


        regulPar = 1e-10;
        if rank(noiCorr(:,:,iFreq))<setup.nSensors*setup.nFilt,
            noiCorr(:,:,iFreq) = noiCorr(:,:,iFreq)*(1-regulPar)+...
                (regulPar)*trace(noiCorr(:,:,iFreq))/(setup.nFilt*...
                setup.nSensors)*...
                eye(setup.nFilt*setup.nSensors);
        end
        if rank(obsCorr(:,:,iFreq))<setup.nSensors*setup.nFilt,
            obsCorr(:,:,iFreq) = obsCorr(:,:,iFreq)*(1-regulPar)+...
                (regulPar)*trace(obsCorr(:,:,iFreq))/(setup.nFilt*...
                setup.nSensors)*...
                eye(setup.nFilt*setup.nSensors);
        end

        % joint diagonalization
        [geigVec(:,:,iFreq),geigVal(:,:,iFreq)] = ...
            jeig(sigCorr(:,:,iFreq),...
            noiCorr(:,:,iFreq),1);

        for iFiltStr=1:length(setup.filtStrings),
            switch char(setup.filtStrings(iFiltStr)),
                case 'maxSnr',
                % max snr filt
                hMaxSnr(:,iFreq) = (geigVec(:,1,iFreq)*...
                    geigVec(:,1,iFreq)')/geigVal(1,1,iFreq)*...
                    sigCorr(:,1,iFreq);
                if norm(hMaxSnr(:,iFreq))==0,
                    hMaxSnr(:,iFreq) = eye(setup.nFilt*setup.nSensors,1);
                end
                data.maxSnr.sigStft(iFreq,iFrame) = ...
                    hMaxSnr(:,iFreq)'*sigBlock;
                data.maxSnr.noiStft(iFreq,iFrame) = ...
                    hMaxSnr(:,iFreq)'*noiBlock;
                data.maxSnr.obsStft(iFreq,iFrame) = ...
                    hMaxSnr(:,iFreq)'*obsBlock;

                case 'wiener',
                % wiener filt
                blbSubW = zeros(setup.nFilt*setup.nSensors);
                if isfield(setup,'wiener')==0,
                    setup.wiener.signalRanks = setup.nFilt;
                end
                iterRanks = 1;
                for iRanks = 1:max(setup.wiener.signalRanks),
                    blbSubW = blbSubW + (geigVec(:,iRanks,iFreq)...
                        *geigVec(:,iRanks,iFreq)')...
                        /(1+geigVal(iRanks,iRanks,iFreq));
                    if sum(iRanks==setup.wiener.signalRanks),
                        hWiener(:,iFreq,iterRanks) = blbSubW...
                            *sigCorr(:,1,iFreq);
                        if norm(hWiener(:,iFreq,iterRanks))==0,
                            hWiener(:,iFreq,iterRanks) = ...
                                eye(setup.nFilt*setup.nSensors,1);
                        end
                        iterRanks = iterRanks + 1;
                    end
                end

                for iRanks=1:length(setup.wiener.signalRanks),
                    data.wiener.sigStft(iFreq,iFrame,iRanks) = ...
                        hWiener(:,iFreq,iRanks)'*sigBlock;
                    data.wiener.noiStft(iFreq,iFrame,iRanks) = ...
                        hWiener(:,iFreq,iRanks)'*noiBlock;
                    data.wiener.obsStft(iFreq,iFrame,iRanks) = ...
                        hWiener(:,iFreq,iRanks)'*obsBlock;
                end

                case 'minDis',
                % minimum dist filt
                blbSub = zeros(setup.nFilt*setup.nSensors);
                iterRanks = 1;
                for iRanks = 1:max(setup.minDis.signalRanks),
                    blbSub = blbSub + (geigVec(:,iRanks,iFreq)*...
                        geigVec(:,iRanks,iFreq)')/...
                        geigVal(iRanks,iRanks,iFreq);
                    if sum(iRanks==setup.minDis.signalRanks),
                        hMinDis(:,iFreq,iterRanks) = blbSub*...
                            sigCorr(:,1,iFreq);
                        if norm(hMinDis(:,iFreq,iterRanks))==0,
                            hMinDis(:,iFreq,iterRanks) = ...
                                eye(setup.nFilt*setup.nSensors,1);
                        end
                        iterRanks = iterRanks + 1;
                    end
                end
                for iRanks=1:length(setup.minDis.signalRanks),
                    data.minDis.sigStft(iFreq,iFrame,iRanks) = ...
                        hMinDis(:,iFreq,iRanks)'*sigBlock;
                    data.minDis.noiStft(iFreq,iFrame,iRanks) = ...
                        hMinDis(:,iFreq,iRanks)'*noiBlock;
                    data.minDis.obsStft(iFreq,iFrame,iRanks) = ...
                        hMinDis(:,iFreq,iRanks)'*obsBlock;
                end
                case 'trOff',
                % trade off filt
                blbSub = zeros(setup.nFilt*setup.nSensors);
                iterRanks = 1;
                for iRanks = 1:max(setup.trOff.signalRanks),
                    blbSub = blbSub + (geigVec(:,iRanks,iFreq)*...
                        geigVec(:,iRanks,iFreq)')/(setup.trOff.mu+...
                        geigVal(iRanks,iRanks,iFreq));
                    if sum(iRanks==setup.trOff.signalRanks),
                        hTrOff(:,iFreq,iterRanks) = blbSub*...
                            sigCorr(:,1,iFreq);
                        if norm(hTrOff(:,iFreq,iterRanks))==0,
                            hTrOff(:,iFreq,iterRanks) = ...
                                eye(setup.nFilt*setup.nSensors,1);
                        end
                        iterRanks = iterRanks + 1;
                    end
                end
                for iRanks=1:length(setup.trOff.signalRanks),
                    data.trOff.sigStft(iFreq,iFrame,iRanks) = ...
                        hTrOff(:,iFreq,iRanks)'*sigBlock;
                    data.trOff.noiStft(iFreq,iFrame,iRanks) = ...
                        hTrOff(:,iFreq,iRanks)'*noiBlock;
                    data.trOff.obsStft(iFreq,iFrame,iRanks) = ...
                        hTrOff(:,iFreq,iRanks)'*obsBlock;
                    data.trOff.sigPow(iFreq,iFrame,iRanks) = ...
                        real(hTrOff(:,iFreq,iRanks)'*...
                        sigCorr(:,:,iFreq)*hTrOff(:,iFreq,iRanks));
                    data.trOff.noiPow(iFreq,iFrame,iRanks) = ...
                        real(hTrOff(:,iFreq,iRanks)'*...
                        noiCorr(:,:,iFreq)*hTrOff(:,iFreq,iRanks));
                    data.trOff.obsPow(iFreq,iFrame,iRanks) = ...
                        real(hTrOff(:,iFreq,iRanks)'*...
                        obsCorr(:,:,iFreq)*hTrOff(:,iFreq,iRanks));
                end
            end
        end

    end
end

for iFiltStr=1:length(setup.filtStrings),
    switch char(setup.filtStrings(iFiltStr)),
        case 'maxSnr',
        data.maxSnr.sig = stftInvBatch(data.maxSnr.sigStft,...
            setup.nWin,setup.nFft);
        data.maxSnr.noi = stftInvBatch(data.maxSnr.noiStft,...
            setup.nWin,setup.nFft);
        data.maxSnr.obs = stftInvBatch(data.maxSnr.obsStft,...
            setup.nWin,setup.nFft);
        case 'wiener',
        for iRanks=1:length(setup.wiener.signalRanks),
            data.wiener.sig(:,iRanks) = stftInvBatch(...
                data.wiener.sigStft(:,:,iRanks),setup.nWin,setup.nFft);
            data.wiener.noi(:,iRanks) = stftInvBatch(...
                data.wiener.noiStft(:,:,iRanks),setup.nWin,setup.nFft);
            data.wiener.obs(:,iRanks) = stftInvBatch(...
                data.wiener.obsStft(:,:,iRanks),setup.nWin,setup.nFft);
        end
        case 'minDis',
        for iRanks=1:length(setup.minDis.signalRanks),
            data.minDis.sig(:,iRanks) = stftInvBatch(...
                data.minDis.sigStft(:,:,iRanks),setup.nWin,setup.nFft);
            data.minDis.noi(:,iRanks) = stftInvBatch(...
                data.minDis.noiStft(:,:,iRanks),setup.nWin,setup.nFft);
            data.minDis.obs(:,iRanks) = stftInvBatch(...
                data.minDis.obsStft(:,:,iRanks),setup.nWin,setup.nFft);
        end
        case 'trOff',
        for iRanks=1:length(setup.trOff.signalRanks),
            data.trOff.sig(:,iRanks) = stftInvBatch(...
                data.trOff.sigStft(:,:,iRanks),setup.nWin,setup.nFft);
            data.trOff.noi(:,iRanks) = stftInvBatch(...
                data.trOff.noiStft(:,:,iRanks),setup.nWin,setup.nFft);
            data.trOff.obs(:,iRanks) = stftInvBatch(...
                data.trOff.obsStft(:,:,iRanks),setup.nWin,setup.nFft);
        end
    end
end

% save setup
setup.stftFreqGrid = freqGrid;
setup.stftTimeGrid = timeGrid;
```

Listing 6.9 Function for measuring the performance of multichannel, variable span filters in the STFT domain.

```
function [performance] = stftMultichannelMeasurePerformance(data,...
    setup,flagFromSignals)

[nFreqs,nFrames] = size(data.raw.sigStft);
nWin = setup.nWin;
nBlockSkip = 5;
filtStrings = setup.filtStrings;


if flagFromSignals,
    % raw signal powers (channel 1)
    [performance.power.raw.sigPowNb,performance.power.raw.sigPowNbMean,...
        performance.power.raw.sigPowFb,...
        performance.power.raw.sigPowFbMean] = ...
        calculatePowers(data.raw.sigStft(:,:,1),nFreqs,nWin,nBlockSkip);

    % raw noise powers (channel 1)
    [performance.power.raw.noiPowNb,performance.power.raw.noiPowNbMean,...
        performance.power.raw.noiPowFb,...
        performance.power.raw.noiPowFbMean] = ...
        calculatePowers(data.raw.noiStft(:,:,1),nFreqs,nWin,nBlockSkip);

    [performance.noiseReduction.iSnr.nb,...
        performance.noiseReduction.iSnr.fb,...
        performance.noiseReduction.iSnr.nbMean, ...
        performance.noiseReduction.iSnr.fbMean] ...
        = measurePerformance(performance,'raw');

    for iFiltStr=1:length(filtStrings),
        switch char(filtStrings(iFiltStr)),
            case 'maxSnr',
            % signal and noise powers (max snr)
            [performance.power.maxSnr.sigPowNb,...
                performance.power.maxSnr.sigPowNbMean,...
                performance.power.maxSnr.sigPowFb,...
                performance.power.maxSnr.sigPowFbMean] = ...
                calculatePowers(data.maxSnr.sigStft,nFreqs,...
                nWin,nBlockSkip);

            [performance.power.maxSnr.noiPowNb,...
                performance.power.maxSnr.noiPowNbMean,...
                performance.power.maxSnr.noiPowFb,...
                performance.power.maxSnr.noiPowFbMean] = ...
                calculatePowers(data.maxSnr.noiStft,nFreqs,...
                nWin,nBlockSkip);

            [performance.noiseReduction.oSnr.maxSnr.nb,...
                performance.noiseReduction.oSnr.maxSnr.fb,...
                performance.noiseReduction.oSnr.maxSnr.nbMean,...
                performance.noiseReduction.oSnr.maxSnr.fbMean,...
                performance.signalDistortion.dsd.maxSnr.nb,...
                performance.signalDistortion.dsd.maxSnr.fb,...
                performance.signalDistortion.dsd.maxSnr.nbMean,...
                performance.signalDistortion.dsd.maxSnr.fbMean]...
                = measurePerformance(performance,char(...
                filtStrings(iFiltStr)));

            case 'wiener',
            [performance.power.wiener.sigPowNb,...
                performance.power.wiener.sigPowNbMean,...
                performance.power.wiener.sigPowFb,...
                performance.power.wiener.sigPowFbMean] = ...
                calculatePowers(data.wiener.sigStft,nFreqs,nWin,...
                nBlockSkip);

            [performance.power.wiener.noiPowNb,...
                performance.power.wiener.noiPowNbMean,...
                performance.power.wiener.noiPowFb,...
                performance.power.wiener.noiPowFbMean] = ...
                calculatePowers(data.wiener.noiStft,nFreqs,nWin,...
                nBlockSkip);

            [performance.noiseReduction.oSnr.wiener.nb,...
                performance.noiseReduction.oSnr.wiener.fb,...
                performance.noiseReduction.oSnr.wiener.nbMean,...
                performance.noiseReduction.oSnr.wiener.fbMean,...
                performance.signalDistortion.dsd.wiener.nb,...
                performance.signalDistortion.dsd.wiener.fb,...
                performance.signalDistortion.dsd.wiener.nbMean,...
                performance.signalDistortion.dsd.wiener.fbMean]...
                = measurePerformance(performance,char(...
                filtStrings(iFiltStr)));
            case 'minDis',
                for iRank = 1:size(data.minDis.sigStft,3),
                    [performance.power.minDis.sigPowNb(:,:,iRank),...
                    performance.power.minDis.sigPowNbMean(:,:,iRank),...
                    performance.power.minDis.sigPowFb(:,:,iRank),...
                    performance.power.minDis.sigPowFbMean(:,:,iRank)] = ...
                    calculatePowers(data.minDis.sigStft(:,:,iRank),...
                    nFreqs,nWin,nBlockSkip);

                    [performance.power.minDis.noiPowNb(:,:,iRank), ...
                    performance.power.minDis.noiPowNbMean(:,:,iRank), ...
                    performance.power.minDis.noiPowFb(:,:,iRank), ...
                    performance.power.minDis.noiPowFbMean(:,:,iRank)] = ...
                    calculatePowers(data.minDis.noiStft(:,:,iRank),...
                    nFreqs,nWin,nBlockSkip);

            [performance.noiseReduction.oSnr.minDis.nb(:,:,iRank),...
            performance.noiseReduction.oSnr.minDis.fb(:,:,iRank),...
            performance.noiseReduction.oSnr.minDis.nbMean(:,:,iRank),...
            performance.noiseReduction.oSnr.minDis.fbMean(:,:,iRank),...
            performance.signalDistortion.dsd.minDis.nb(:,:,iRank),...
            performance.signalDistortion.dsd.minDis.fb(:,:,iRank),...
            performance.signalDistortion.dsd.minDis.nbMean(:,:,iRank),...
            performance.signalDistortion.dsd.minDis.fbMean(:,:,iRank)]...
            = measurePerformance(performance,char(...
            filtStrings(iFiltStr)),iRank);
                end
            case 'trOff',
                for iRank = 1:size(data.trOff.sigStft,3),
                    [performance.power.trOff.sigPowNb(:,:,iRank),...
                    performance.power.trOff.sigPowNbMean(:,:,iRank),...
                    performance.power.trOff.sigPowFb(:,:,iRank),...
                    performance.power.trOff.sigPowFbMean(:,:,iRank)] = ...
                    calculatePowers(data.trOff.sigStft(:,:,iRank),...
                    nFreqs,nWin,nBlockSkip);

```

```
                        [performance.power.trOff.noiPowNb(:,:,iRank), ...
                        performance.power.trOff.noiPowNbMean(:,:,iRank), ...
                        performance.power.trOff.noiPowFb(:,:,iRank), ...
                        performance.power.trOff.noiPowFbMean(:,:,iRank)] = ...
                        calculatePowers(data.trOff.noiStft(:,:,iRank),...
                        nFreqs,nWin,nBlockSkip);

                [performance.noiseReduction.oSnr.trOff.nb(:,:,iRank),...
                performance.noiseReduction.oSnr.trOff.fb(:,:,iRank),...
                performance.noiseReduction.oSnr.trOff.nbMean(:,:,iRank),...
                performance.noiseReduction.oSnr.trOff.fbMean(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.nb(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.fb(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.nbMean(:,:,iRank),...
                performance.signalDistortion.dsd.trOff.fbMean(:,:,iRank)]...
                = measurePerformance(performance,char(...
                filtStrings(iFiltStr)),iRank);
                    end
            end
        end
    end

    end

    %%
    function [powNb,powNbMean,powFb,powFbMean] = calculatePowers(stftData,...
        fftLen,winLen,nSkip)

    powNb = abs([stftData(:,:);conj(flipud(stftData(2:end-1,:)))]).^2/...
        fftLen/winLen;
    powNbMean = mean(powNb(:,nSkip+1:end),2);
    powFb = sum(powNb,1);
    powFbMean = mean(powFb(1,nSkip+1:end));

    end

    function [snrNb,snrFb,snrNbMean,snrFbMean,dsdNb,dsdFb,dsdNbMean,dsdFbMean]..
        = measurePerformance(performance,filtStr,iRank)

    if nargin < 3,
        iRank = 1;
    end

    snrNb = eval(['performance.power.',filtStr,'.sigPowNb(:,:,iRank)'])...
        ./eval(['performance.power.',filtStr,'.noiPowNb(:,:,iRank)']);
    snrFb = eval(['performance.power.',filtStr,'.sigPowFb(:,:,iRank)'])...
        ./eval(['performance.power.',filtStr,'.noiPowFb(:,:,iRank)']);

    snrNbMean = eval(['performance.power.',filtStr,...
        '.sigPowNbMean(:,:,iRank)'])...
        ./eval(['performance.power.',filtStr,'.noiPowNbMean(:,:,iRank)']);
    snrFbMean = eval(['performance.power.',filtStr,...
        '.sigPowFbMean(:,:,iRank)'])...
        ./eval(['performance.power.',filtStr,'.noiPowFbMean(:,:,iRank)']);

    dsdNb = performance.power.raw.sigPowNb...
        ./eval(['performance.power.',filtStr,'.sigPowNb(:,:,iRank)']);
    dsdFb = performance.power.raw.sigPowFb...
        ./eval(['performance.power.',filtStr,'.sigPowFb(:,:,iRank)']);

    dsdNbMean = performance.power.raw.sigPowNbMean...
        ./eval(['performance.power.',filtStr,'.sigPowNbMean(:,:,iRank)']);
    dsdFbMean = performance.power.raw.sigPowFbMean...
        ./eval(['performance.power.',filtStr,'.sigPowFbMean(:,:,iRank)']);

    end
```

Chapter 7
Binaural Signal Enhancement in the Time Domain

In binaural signal enhancement, two signals need to be extracted from the sensor array as the phase information is also of importance. Since we deal with two real signals, it is more convenient to artificially form a complex signal that can carry both the amplitude and phase. This process naturally leads to the use of the widely linear filtering technique. In this context, we show again how the concept of variable span (VS) linear filtering can be applied in the time domain.

7.1 Signal Model and Problem Formulation

We consider the signal model in which an array consisting of $2M$ sensors[1] capture a source signal convolved with acoustic impulse responses in some noise field. The signal received at the ith sensor is then expressed as [1]

$$\begin{aligned} y_{\mathrm{re},i}(t) &= g_{\mathrm{re},i}(t) * s(t) + v_{\mathrm{re},i}(t) \\ &= x_{\mathrm{re},i}(t) + v_{\mathrm{re},i}(t),\ i = 1, 2, \ldots, 2M, \end{aligned} \tag{7.1}$$

where the subscript "re" stands for real, t is the discrete-time index, $g_{\mathrm{re},i}(t)$ is the acoustic impulse response from the unknown desired source, $s(t)$, location to the ith sensor, $*$ stands for linear convolution, and $v_{\mathrm{re},i}(t)$ is the additive noise at sensor i. We assume that the impulse responses are time invariant. We also assume that the signals $x_{\mathrm{re},i}(t) = g_{\mathrm{re},i}(t) * s(t)$ and $v_{\mathrm{re},i}(t)$ are uncorrelated, zero mean, real, broadband, and stationary.

Since we want to deal with binaural signals, it is more convenient to work in the complex domain in order that the original (binaural) problem is transformed into the conventional (monaural) noise reduction processing with a sensor array [2]. Indeed, from the $2M$ real-valued sensor signals given in (7.1),

[1] The generalization to an odd number of sensors is straightforward.

we can form M complex-valued sensor signals as

$$\begin{aligned} y_m(t) &= y_{\mathrm{re},m}(t) + \jmath y_{\mathrm{re},M+m}(t) \\ &= x_m(t) + v_m(t),\ m = 1, 2, \ldots, M, \end{aligned} \tag{7.2}$$

where $\jmath = \sqrt{-1}$ is the imaginary unit,

$$x_m(t) = x_{\mathrm{re},m}(t) + \jmath x_{\mathrm{re},M+m}(t),\ m = 1, 2, \ldots, M \tag{7.3}$$

is the complex convolved desired source signal, and

$$v_m(t) = v_{\mathrm{re},m}(t) + \jmath v_{\mathrm{re},M+m}(t),\ m = 1, 2, \ldots, M \tag{7.4}$$

is the complex additive noise.

In the rest, it is convenient to work with blocks of L successive time samples, i.e.,

$$\mathbf{y}_m(t) = \mathbf{x}_m(t) + \mathbf{v}_m(t),\ m = 1, 2, \ldots, M, \tag{7.5}$$

where

$$\mathbf{y}_m(t) = \begin{bmatrix} y_m(t)\ y_m(t-1) \cdots y_m(t-L+1) \end{bmatrix}^T \tag{7.6}$$

is a vector of length L, and $\mathbf{x}_m(t)$ and $\mathbf{v}_m(t)$ are defined in a similar way to $\mathbf{y}_m(t)$. Concatenating all vectors $\mathbf{y}_m(t),\ m = 1, 2, \ldots, M$ together, we get the vector of length ML:

$$\begin{aligned} \underline{\mathbf{y}}(t) &= \begin{bmatrix} \mathbf{y}_1^T(t)\ \mathbf{y}_2^T(t) \cdots \mathbf{y}_M^T(t) \end{bmatrix}^T \\ &= \underline{\mathbf{x}}(t) + \underline{\mathbf{v}}(t), \end{aligned} \tag{7.7}$$

where $\underline{\mathbf{x}}(t)$ and $\underline{\mathbf{v}}(t)$ are also concatenated vectors of $\mathbf{x}_m(t)$ and $\mathbf{v}_m(t)$, respectively. We deduce that the $ML \times ML$ correlation matrix of $\underline{\mathbf{y}}(t)$ is

$$\begin{aligned} \mathbf{R}_{\underline{\mathbf{y}}} &= E\left[\underline{\mathbf{y}}(t)\underline{\mathbf{y}}^H(t)\right] \\ &= \mathbf{R}_{\underline{\mathbf{x}}} + \mathbf{R}_{\underline{\mathbf{v}}}, \end{aligned} \tag{7.8}$$

where $\mathbf{R}_{\underline{\mathbf{x}}} = E\left[\underline{\mathbf{x}}(t)\underline{\mathbf{x}}^H(t)\right]$ and $\mathbf{R}_{\underline{\mathbf{v}}} = E\left[\underline{\mathbf{v}}(t)\underline{\mathbf{v}}^H(t)\right]$ are the correlation matrices of $\underline{\mathbf{x}}(t)$ and $\underline{\mathbf{v}}(t)$, respectively.

As we can notice from the model given in (7.2), we deal with complex random variables. A very important statistical characteristic of a complex random variable (CRV) is the so-called circularity property or lack of it (noncircularity) [3], [4]. A zero-mean CRV, z, is circular if and only if the only nonnull moments and cumulants are the moments and cumulants constructed with the same power in z and z^* [5], [6], where the superscript * denotes complex conjugation. In particular, z is said to be a second-order circular CRV (CCRV) if its so-called pseudo-variance [3] is equal to zero, i.e.,

$E\left(z^2\right) = 0$, while its variance is nonnull, i.e., $E\left(|z|^2\right) \neq 0$. This means that the second-order behavior of a CCRV is well described by its variance. If the pseudo-variance $E\left(z^2\right)$ is not equal to 0, the CRV z is then noncircular. A good measure of the second-order circularity is the circularity quotient [3] defined as the ratio between the pseudo-variance and the variance, i.e.,

$$\gamma_z = \frac{E\left(z^2\right)}{E\left(|z|^2\right)}. \tag{7.9}$$

This measure coincides with the coherence function between z and z^*. Therefore, it is obvious that $0 \leq |\gamma_z| \leq 1$. If $\gamma_z = 0$ then z is a second-order CCRV; otherwise, z is noncircular and a large value of $|\gamma_z|$ indicates that the CRV z is highly noncircular.

Now, let us examine whether the complex convolved desired source signal, $x_m(t)$, is second-order circular or not. We have

$$\begin{aligned} \gamma_{x_m} &= \frac{E\left[x_m^2(t)\right]}{E\left[|x_m(t)|^2\right]} \qquad (7.10)\\ &= \frac{E\left[x_{\text{re},m}^2(t)\right] - E\left[x_{\text{re},M+m}^2(t)\right] + 2\jmath E\left[x_{\text{re},m}(t)x_{\text{re},M+m}(t)\right]}{\sigma_{x_m}^2}, \end{aligned}$$

where $\sigma_{x_m}^2 = E\left[|x_m(t)|^2\right]$ is the variance of $x_m(t)$. One can check from (7.10) that the CRV $x_m(t)$ is second-order circular (i.e., $\gamma_{x_m} = 0$) if and only if

$$E\left[x_{\text{re},m}^2(t)\right] = E\left[x_{\text{re},M+m}^2(t)\right] \quad \text{and} \quad E\left[x_{\text{re},m}(t)x_{\text{re},M+m}(t)\right] = 0. \tag{7.11}$$

Since the signals $x_{\text{re},m}(t)$ and $x_{\text{re},M+m}(t)$ come from the same source, they are in general correlated. As a result, the second condition in (7.11) should not be true. Therefore, we can safely state that the complex convolved desired source signal, $x_m(t)$, is noncircular, and so is the complex sensor signal, $y_m(t)$. If we assume that the noise terms at the sensors are uncorrelated and have the same power then $\gamma_{v_m} = 0$ [i.e., $v_m(t)$ is a second-order CCRV].

Since we deal with noncircular CRVs as demonstrated above, the vector $\underline{\mathbf{y}}^*(t)$ should also be included as part of the observations as required by the widely linear (WL) estimation theory [4], [7]. Therefore, we define the augmented observation vector of length $2ML$ as

$$\begin{aligned} \underline{\widetilde{\mathbf{y}}}(t) &= \begin{bmatrix} \underline{\mathbf{y}}(t) \\ \underline{\mathbf{y}}^*(t) \end{bmatrix} \\ &= \underline{\widetilde{\mathbf{x}}}(t) + \underline{\widetilde{\mathbf{v}}}(t), \end{aligned} \tag{7.12}$$

where $\underline{\widetilde{\mathbf{x}}}(t)$ and $\underline{\widetilde{\mathbf{v}}}(t)$ are defined similarly to $\underline{\widetilde{\mathbf{y}}}(t)$.

Now, our aim is to recover the complex desired signal vector, $\mathbf{x}_1(t)$, from the augmented complex observation vector, $\underline{\widetilde{\mathbf{y}}}(t)$, the best way we can. It is clear then that we have two objectives. The first one is to attenuate the contribution of the noise terms as much as possible. The second objective is to preserve the spatial information included in $\mathbf{x}_1(t)$, so that with the enhanced signals, along with our binaural hearing process, we will still be able to localize the source $s(t)$.

Since $\mathbf{x}_1(t)$ is the desired signal vector, we need to extract it from $\underline{\widetilde{\mathbf{x}}}(t)$. Specifically, the vector $\underline{\widetilde{\mathbf{x}}}(t)$ is decomposed into the following form:

$$\begin{aligned}\underline{\widetilde{\mathbf{x}}}(t) &= \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}^{-1}\mathbf{x}_1(t) + \underline{\widetilde{\mathbf{x}}}_{\text{i}}(t) \\ &= \mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{x}_1(t) + \underline{\widetilde{\mathbf{x}}}_{\text{i}}(t),\end{aligned} \tag{7.13}$$

where

$$\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} = \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}^{-1}, \tag{7.14}$$

$\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} = E\left[\underline{\widetilde{\mathbf{x}}}(t)\mathbf{x}_1^H(t)\right]$ is the cross-correlation matrix of size $2ML \times L$ between $\underline{\widetilde{\mathbf{x}}}(t)$ and $\mathbf{x}_1(t)$, $\mathbf{R}_{\mathbf{x}_1} = E\left[\mathbf{x}_1(t)\mathbf{x}_1^H(t)\right]$ is the $L \times L$ correlation matrix of $\mathbf{x}_1(t)$, and $\underline{\widetilde{\mathbf{x}}}_{\text{i}}(t)$ is the interference signal vector. It is easy to check that $\underline{\widetilde{\mathbf{x}}}_{\text{d}}(t) = \mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{x}_1(t)$ and $\underline{\widetilde{\mathbf{x}}}_{\text{i}}(t)$ are orthogonal, i.e.,

$$E\left[\underline{\widetilde{\mathbf{x}}}_{\text{d}}(t)\underline{\widetilde{\mathbf{x}}}_{\text{i}}^H(t)\right] = \mathbf{0}_{2ML \times 2ML}. \tag{7.15}$$

Using (7.13), we can rewrite (7.12) as

$$\begin{aligned}\underline{\widetilde{\mathbf{y}}}(t) &= \mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{x}_1(t) + \underline{\widetilde{\mathbf{x}}}_{\text{i}}(t) + \underline{\widetilde{\mathbf{v}}}(t) \\ &= \underline{\widetilde{\mathbf{x}}}_{\text{d}}(t) + \underline{\widetilde{\mathbf{x}}}_{\text{i}}(t) + \underline{\widetilde{\mathbf{v}}}(t)\end{aligned} \tag{7.16}$$

and the correlation matrix (of size $2ML \times 2ML$) of $\underline{\widetilde{\mathbf{y}}}(t)$ from the previous expression is

$$\begin{aligned}\mathbf{R}_{\underline{\widetilde{\mathbf{y}}}} &= \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{d}}} + \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{i}}} + \mathbf{R}_{\underline{\widetilde{\mathbf{v}}}} \\ &= \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{d}}} + \mathbf{R}_{\text{in}},\end{aligned} \tag{7.17}$$

where $\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{d}}} = \mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}^H = \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}^{-1}\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}^H$ and $\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{i}}}$ are the correlation matrices of $\underline{\widetilde{\mathbf{x}}}_{\text{d}}(t)$ and $\underline{\widetilde{\mathbf{x}}}_{\text{i}}(t)$, respectively, and

$$\mathbf{R}_{\text{in}} = \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{i}}} + \mathbf{R}_{\underline{\widetilde{\mathbf{v}}}} \tag{7.18}$$

is the interference-plus-noise correlation matrix. It is clear that the rank of $\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{d}}}$ is L, while the rank of $\mathbf{R}_{\text{in}}$ is assumed to be equal to $2ML$.

Using the well-known joint diagonalization technique [8], the two Hermitian matrices $\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\text{d}}}$ and $\mathbf{R}_{\text{in}}$ can be jointly diagonalized as follows:

$$\mathbf{B}^H \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}} \mathbf{B} = \mathbf{\Lambda}, \tag{7.19}$$

$$\mathbf{B}^H \mathbf{R}_{\mathrm{in}} \mathbf{B} = \mathbf{I}_{2ML}, \tag{7.20}$$

where $\mathbf{B}$ is a full-rank square matrix (of size $2ML \times 2ML$), $\mathbf{\Lambda}$ is a diagonal matrix whose main elements are real and nonnegative, and $\mathbf{I}_{2ML}$ is the $2ML \times 2ML$ identity matrix. Furthermore, $\mathbf{\Lambda}$ and $\mathbf{B}$ are the eigenvalue and eigenvector matrices, respectively, of $\mathbf{R}_{\mathrm{in}}^{-1} \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}}$, i.e.,

$$\mathbf{R}_{\mathrm{in}}^{-1} \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}} \mathbf{B} = \mathbf{B}\mathbf{\Lambda}. \tag{7.21}$$

Since the rank of the matrix $\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}}$ is equal to L, the eigenvalues of $\mathbf{R}_{\mathrm{in}}^{-1} \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}}$ can be ordered as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_L > \lambda_{L+1} = \cdots = \lambda_{2ML} = 0$. In other words, the last $2ML - L$ eigenvalues of the matrix product $\mathbf{R}_{\mathrm{in}}^{-1} \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}}$ are exactly zero, while its first L eigenvalues are positive, with λ_1 being the maximum eigenvalue. We also denote by $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{2ML}$, the corresponding eigenvectors. Therefore, the noisy signal correlation matrix can also be diagonalized as

$$\mathbf{B}^H \mathbf{R}_{\underline{\widetilde{\mathbf{y}}}} \mathbf{B} = \mathbf{\Lambda} + \mathbf{I}_{2ML}. \tag{7.22}$$

We can decompose the matrix $\mathbf{B}$ as

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}' \ \mathbf{B}'' \end{bmatrix}, \tag{7.23}$$

where

$$\mathbf{B}' = \begin{bmatrix} \mathbf{b}_1 \ \mathbf{b}_2 \ \cdots \ \mathbf{b}_L \end{bmatrix} \tag{7.24}$$

is a $2ML \times L$ matrix that spans the desired signal-plus-noise subspace and

$$\mathbf{B}'' = \begin{bmatrix} \mathbf{b}_{L+1} \ \mathbf{b}_{L+2} \ \cdots \ \mathbf{b}_{2ML} \end{bmatrix} \tag{7.25}$$

is a $2ML \times (2ML - L)$ matrix that spans the noise subspace. As a result,

$$\mathbf{B}'^H \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}} \mathbf{B}' = \mathbf{\Lambda}', \tag{7.26}$$

$$\mathbf{B}'^H \mathbf{R}_{\mathrm{in}} \mathbf{B}' = \mathbf{I}_L, \tag{7.27}$$

$$\mathbf{B}'^H \mathbf{R}_{\underline{\widetilde{\mathbf{y}}}} \mathbf{B}' = \mathbf{\Lambda}' + \mathbf{I}_L, \tag{7.28}$$

where $\mathbf{\Lambda}'$ is an $L \times L$ diagonal matrix containing the (nonnull) positive eigenvalues of $\mathbf{R}_{\mathrm{in}}^{-1} \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}}$ and $\mathbf{I}_L$ is the $L \times L$ identity matrix. Matrices $\mathbf{B}'$ and $\mathbf{\Lambda}'$ will be very useful to manipulate in the rest of the chapter.

7.2 Widely Linear Filtering with a Rectangular Matrix

Binaural noise reduction with the widely linear filtering technique is performed by applying a linear transformation to the augmented observation vector $\underline{\widetilde{\mathbf{y}}}(t)$. We get

$$\begin{aligned}\mathbf{z}(t) &= \mathbf{H}\underline{\widetilde{\mathbf{y}}}(t) \\ &= \mathbf{x}_{\text{fd}}(t) + \mathbf{x}_{\text{ri}}(t) + \mathbf{v}_{\text{rn}}(t),\end{aligned} \tag{7.29}$$

where $\mathbf{z}(t)$ is the estimate of $\mathbf{x}_1(t)$,

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1^H \\ \mathbf{h}_2^H \\ \vdots \\ \mathbf{h}_L^H \end{bmatrix} \tag{7.30}$$

is a rectangular filtering matrix of size $L \times 2ML$, $\mathbf{h}_l,\ l = 1, 2, \ldots, L$ are filters of length $2ML$,

$$\mathbf{x}_{\text{fd}}(t) = \mathbf{H}\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{x}_1(t) \tag{7.31}$$

is the filtered desired signal,

$$\mathbf{x}_{\text{ri}}(t) = \mathbf{H}\underline{\widetilde{\mathbf{x}}}_{\text{i}}(t) \tag{7.32}$$

is the residual interference, and

$$\mathbf{v}_{\text{rn}}(t) = \mathbf{H}\underline{\widetilde{\mathbf{v}}}(t) \tag{7.33}$$

is the residual noise.

It is always possible to write $\mathbf{h}_l$ in a basis formed from the vectors $\mathbf{b}_i,\ i = 1, 2, \ldots, L$ that span the desired signal-plus-noise subspace, i.e.,

$$\mathbf{h}_l = \sum_{i=1}^{L} a_{li}\mathbf{b}_i = \mathbf{B}'\mathbf{a}_l, \tag{7.34}$$

where $a_{li},\ i = 1, 2, \ldots, L$ are the components of the vector $\mathbf{a}_l$ of length L. Notice that we completely ignore the noise-only subspace as many optimal filtering matrices will do the same. We will see that this choice is reasonable and will lead to interesting filtering matrices for noise reduction. Therefore, the filtering matrix can be expressed as

$$\mathbf{H} = \mathbf{A}\mathbf{B}'^H, \tag{7.35}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^H \\ \mathbf{a}_2^H \\ \vdots \\ \mathbf{a}_L^H \end{bmatrix} \tag{7.36}$$

is an $L \times L$ matrix. Now, instead of estimating $\mathbf{H}$ (of size $L \times 2ML$) as in conventional approaches, we estimate $\mathbf{A}$.

The correlation matrix of $\mathbf{z}(t)$ is then

$$\mathbf{R}_{\mathbf{z}} = \mathbf{R}_{\mathbf{x}_{\mathrm{fd}}} + \mathbf{R}_{\mathbf{x}_{\mathrm{ri}}} + \mathbf{R}_{\mathbf{v}_{\mathrm{rn}}}, \tag{7.37}$$

where

$$\begin{aligned} \mathbf{R}_{\mathbf{x}_{\mathrm{fd}}} &= \mathbf{A}\mathbf{B}'^H \mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}} \mathbf{B}' \mathbf{A}^H && (7.38) \\ &= \mathbf{A}\mathbf{\Lambda}'\mathbf{A}^H, \\ \mathbf{R}_{\mathbf{x}_{\mathrm{ri}}} + \mathbf{R}_{\mathbf{v}_{\mathrm{rn}}} &= \mathbf{A}\mathbf{B}'^H \mathbf{R}_{\mathrm{in}} \mathbf{B}' \mathbf{A}^H && (7.39) \\ &= \mathbf{A}\mathbf{A}^H. \end{aligned}$$

7.3 Performance Measures

We explain the performance measures in the context of binaural noise reduction with the complex sensor 1 as the reference. We start by deriving measures related to noise reduction. In the second subsection, we discuss the evaluation of desired signal distortion. Finally, in the last subsection, we present the MSE criterion.

7.3.1 Noise Reduction

Since sensor 1 is the reference, the input SNR is computed from the first L components of $\underline{\widetilde{\mathbf{y}}}(t)$ as defined in (7.16). We easily find that

$$\mathrm{iSNR} = \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}, \tag{7.40}$$

where $\mathbf{R}_{\mathbf{v}_1}$ is the correlation matrix of $\mathbf{v}_1(t)$.

The output SNR is obtained from (7.37). It is given by

$$\begin{aligned}\mathrm{oSNR}\left(\mathbf{A}\right) &= \frac{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}}\mathbf{H}^H\right)}{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\mathrm{in}}\mathbf{H}^H\right)} \\ &= \frac{\mathrm{tr}\left(\mathbf{A}\boldsymbol{\Lambda}'\mathbf{A}^H\right)}{\mathrm{tr}\left(\mathbf{A}\mathbf{A}^H\right)}.\end{aligned} \tag{7.41}$$

Then, the main objective of binaural signal enhancement is to find an appropriate $\mathbf{A}$ that makes the output SNR greater than the input SNR. Consequently, the quality of the complex noisy signal may be enhanced. It can be checked that

$$\mathrm{oSNR}\left(\mathbf{A}\right) \leq \max_l \frac{\mathbf{a}_l^H\boldsymbol{\Lambda}'\mathbf{a}_l}{\mathbf{a}_l^H\mathbf{a}_l}. \tag{7.42}$$

As a result,

$$\mathrm{oSNR}\left(\mathbf{A}\right) \leq \lambda_1. \tag{7.43}$$

This shows how the output SNR is upper bounded.

The noise reduction factor is defined as

$$\begin{aligned}\xi_{\mathrm{nr}}\left(\mathbf{A}\right) &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\mathrm{in}}\mathbf{H}^H\right)} \\ &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}{\mathrm{tr}\left(\mathbf{A}\mathbf{A}^H\right)}.\end{aligned} \tag{7.44}$$

For optimal filtering matrices, we should have $\xi_{\mathrm{nr}}\left(\mathbf{A}\right) \geq 1$. The noise reduction factor is not upper bounded and can go to infinity if we allow infinite distortion.

7.3.2 Desired Signal Distortion

The distortion of the desired signal vector can be measured with the desired signal reduction factor:

$$\begin{aligned}\xi_{\mathrm{sr}}\left(\mathbf{A}\right) &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}{\mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\underline{\widetilde{\mathbf{x}}}_{\mathrm{d}}}\mathbf{H}^H\right)} \\ &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}{\mathrm{tr}\left(\mathbf{A}\boldsymbol{\Lambda}'\mathbf{A}^H\right)}.\end{aligned} \tag{7.45}$$

For optimal filtering matrices, we should have $\xi_{\mathrm{sr}}\left(\mathbf{A}\right) \geq 1$. In the distortionless case, we have $\xi_{\mathrm{sr}}\left(\mathbf{A}\right) = 1$. Hence, a rectangular filtering matrix that does not affect the desired signal requires the constraint:

$$\begin{aligned}\mathbf{H}\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} &= \mathbf{A}\mathbf{B}'^H\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} \\ &= \mathbf{I}_L.\end{aligned} \tag{7.46}$$

It is obvious that we always have

$$\frac{\text{oSNR}\left(\mathbf{A}\right)}{\text{iSNR}} = \frac{\xi_{\text{nr}}\left(\mathbf{A}\right)}{\xi_{\text{sr}}\left(\mathbf{A}\right)}. \tag{7.47}$$

The distortion can also be measured with the desired signal distortion index:

$$\begin{aligned}\upsilon_{\text{sd}}\left(\mathbf{A}\right) &= \frac{E\left\{\left[\mathbf{x}_{\text{fd}}(t) - \mathbf{x}_1(t)\right]^H\left[\mathbf{x}_{\text{fd}}(t) - \mathbf{x}_1(t)\right]\right\}}{\text{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)} \\ &= \frac{\text{tr}\left[\left(\mathbf{H}\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{R}_{\mathbf{x}_1}\left(\mathbf{H}\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} - \mathbf{I}_L\right)^H\right]}{\text{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)} \\ &= \frac{\text{tr}\left[\left(\mathbf{A}\mathbf{B}'^H\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{R}_{\mathbf{x}_1}\left(\mathbf{A}\mathbf{B}'^H\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} - \mathbf{I}_L\right)^H\right]}{\text{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)}.\end{aligned} \tag{7.48}$$

For optimal rectangular filtering matrices, we should have

$$0 \leq \upsilon_{\text{sd}}\left(\mathbf{A}\right) \leq 1 \tag{7.49}$$

and a value of $\upsilon_{\text{sd}}\left(\mathbf{A}\right)$ close to 0 is preferred.

7.3.3 MSE Criterion

It is clear that the error signal vector between the estimated and desired signals is

$$\begin{aligned}\mathbf{e}(t) &= \mathbf{z}(t) - \mathbf{x}_1(t) \\ &= \mathbf{H}\underline{\widetilde{\mathbf{y}}}(t) - \mathbf{x}_1(t) \\ &= \mathbf{e}_{\text{ds}}(t) + \mathbf{e}_{\text{rs}}(t),\end{aligned} \tag{7.50}$$

where

$$\begin{aligned}\mathbf{e}_{\text{ds}}(t) &= \mathbf{x}_{\text{fd}}(t) - \mathbf{x}_1(t) \\ &= \left(\mathbf{H}\mathbf{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{x}_1(t)\end{aligned} \tag{7.51}$$

represents the signal distortion and

$$\begin{aligned}\mathbf{e}_{\mathrm{rs}}(t) &= \mathbf{x}_{\mathrm{ri}}(t) + \mathbf{v}_{\mathrm{rn}}(t) \\ &= \mathbf{H}\underline{\widetilde{\mathbf{x}}}_{\mathrm{i}}(t) + \mathbf{H}\underline{\widetilde{\mathbf{v}}}(t)\end{aligned} \tag{7.52}$$

is the residual interference-plus-noise. We deduce that the MSE criterion is

$$\begin{aligned}J(\mathbf{A}) &= \mathrm{tr}\left\{E\left[\mathbf{e}(t)\mathbf{e}^H(t)\right]\right\} \\ &= \mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right) + \mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\underline{\widetilde{\mathbf{y}}}}\mathbf{H}^H\right) - \mathrm{tr}\left(\mathbf{H}\boldsymbol{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\right) - \mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\boldsymbol{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}^H\mathbf{H}^H\right) \\ &= \mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right) + \mathrm{tr}\left[\mathbf{A}\left(\boldsymbol{\Lambda}' + \mathbf{I}_L\right)\mathbf{A}^H\right] - \mathrm{tr}\left(\mathbf{A}\mathbf{B}'^H\boldsymbol{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\right) \\ &\quad - \mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\boldsymbol{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}^H\mathbf{B}'\mathbf{A}^H\right).\end{aligned} \tag{7.53}$$

Since $E\left[\mathbf{e}_{\mathrm{ds}}(t)\mathbf{e}_{\mathrm{rs}}^H(t)\right] = \mathbf{0}_{L\times L}$, $J(\mathbf{A})$ can also be expressed as

$$\begin{aligned}J(\mathbf{A}) &= \mathrm{tr}\left\{E\left[\mathbf{e}_{\mathrm{ds}}(t)\mathbf{e}_{\mathrm{ds}}^H(t)\right]\right\} + \mathrm{tr}\left\{E\left[\mathbf{e}_{\mathrm{rs}}(t)\mathbf{e}_{\mathrm{rs}}^H(t)\right]\right\} \\ &= J_{\mathrm{ds}}(\mathbf{A}) + J_{\mathrm{rs}}(\mathbf{A}),\end{aligned} \tag{7.54}$$

where

$$\begin{aligned}J_{\mathrm{ds}}(\mathbf{A}) &= \mathrm{tr}\left[\left(\mathbf{H}\boldsymbol{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} - \mathbf{I}_L\right)\mathbf{R}_{\mathbf{x}_1}\left(\mathbf{H}\boldsymbol{\Gamma}_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} - \mathbf{I}_L\right)^H\right] \\ &= \mathrm{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right)\upsilon_{\mathrm{sd}}(\mathbf{A})\end{aligned} \tag{7.55}$$

and

$$\begin{aligned}J_{\mathrm{rs}}(\mathbf{A}) &= \mathrm{tr}\left(\mathbf{H}\mathbf{R}_{\mathrm{in}}\mathbf{H}^H\right) \\ &= \frac{\mathrm{tr}\left(\mathbf{R}_{\mathbf{v}_1}\right)}{\xi_{\mathrm{nr}}(\mathbf{A})}.\end{aligned} \tag{7.56}$$

Finally, we have

$$\begin{aligned}\frac{J_{\mathrm{ds}}(\mathbf{A})}{J_{\mathrm{rs}}(\mathbf{A})} &= \mathrm{iSNR} \times \xi_{\mathrm{nr}}(\mathbf{A}) \times \upsilon_{\mathrm{sd}}(\mathbf{A}) \\ &= \mathrm{oSNR}(\mathbf{A}) \times \xi_{\mathrm{sr}}(\mathbf{A}) \times \upsilon_{\mathrm{sd}}(\mathbf{A}).\end{aligned} \tag{7.57}$$

This shows how the MSEs are related to the most fundamental performance measures.

7.4 Optimal Rectangular Linear Filtering Matrices

In this section, we derive the most important rectangular filtering matrices for binaural noise reduction in the time domain with a sensor array. We will see how these optimal matrices are very closely related thanks to the joint diagonalization formulation.

7.4.1 Maximum SNR

From Subsection 7.3.1, we know that the output SNR is upper bounded by λ_1, which we can consider as the maximum possible output SNR. Then, it is easy to verify that with

$$\mathbf{A}_{\max} = \begin{bmatrix} a_{11}\mathbf{i}^T \\ a_{21}\mathbf{i}^T \\ \vdots \\ a_{L1}\mathbf{i}^T \end{bmatrix}, \tag{7.58}$$

where $a_{l1},\ l = 1, 2, \ldots, L$ are arbitrary complex numbers with at least one of them different from 0 and $\mathbf{i}$ is the first column of the $L \times L$ identity matrix, we have

$$\text{oSNR}\left(\mathbf{A}_{\max}\right) = \lambda_1. \tag{7.59}$$

As a consequence,

$$\begin{aligned} \mathbf{H}_{\max} &= \mathbf{A}_{\max}\mathbf{B}'^H \qquad (7.60) \\ &= \begin{bmatrix} a_{11}\mathbf{b}_1^H \\ a_{21}\mathbf{b}_1^H \\ \vdots \\ a_{L1}\mathbf{b}_1^H \end{bmatrix} \end{aligned}$$

is considered to be the maximum SNR filtering matrix. Clearly,

$$\text{oSNR}\left(\mathbf{H}_{\max}\right) \geq \text{iSNR} \tag{7.61}$$

and

$$0 \leq \text{oSNR}\left(\mathbf{H}\right) \leq \text{oSNR}\left(\mathbf{H}_{\max}\right), \forall \mathbf{H}. \tag{7.62}$$

The choice of the values of $a_{l1},\ l = 1, 2, \ldots, L$ is extremely important in practice. A poor choice of these values leads to high distortions of the desired signal. Therefore, the a_{l1}'s should be found in such a way that distortion is minimized. Substituting (7.58) into the the distortion-based MSE, we get

$$\begin{aligned} J_{\text{ds}}\left(\mathbf{H}_{\max}\right) = \text{tr}\left(\mathbf{R}_{\mathbf{x}_1}\right) + \lambda_1 \sum_{l=1}^{L} \left|a_{l1}\right|^2 - \sum_{l=1}^{L} a_{l1}\mathbf{i}^T\mathbf{B}'^H\mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}\mathbf{R}_{\mathbf{x}_1}\mathbf{i}_{l,L} \\ - \sum_{l=1}^{L} a_{l1}^*\mathbf{i}_{l,L}^T\mathbf{R}_{\mathbf{x}_1}\mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H\mathbf{B}'\mathbf{i}, \end{aligned} \tag{7.63}$$

where $\mathbf{i}_{l,L}$ is the lth column of $\mathbf{I}_L$, and minimizing the previous expression with respect to the a_{l1}^*'s, we find

$$a_{l1} = \mathbf{i}_{l,L}^T \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H \frac{\mathbf{b}_1}{\lambda_1},\ l = 1, 2, \ldots, L. \tag{7.64}$$

Plugging these optimal values in (7.60), we obtain the optimal maximum SNR filtering matrix with minimum desired signal distortion:

$$\mathbf{H}_{\max} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H \frac{\mathbf{b}_1 \mathbf{b}_1^H}{\lambda_1}. \tag{7.65}$$

7.4.2 Wiener

If we differentiate the MSE criterion, $J(\mathbf{A})$, with respect to $\mathbf{A}$ and equate the result to zero, we find

$$\mathbf{A}_{\mathrm{W}} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H \mathbf{B}' \left(\mathbf{\Lambda}' + \mathbf{I}_L\right)^{-1}. \tag{7.66}$$

We deduce that the Wiener filtering matrix for the estimation of the vector $\mathbf{x}_1(t)$, which is confined in the desired signal-plus-noise subspace[2], is

$$\mathbf{H}_{\mathrm{W}} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^H}{1 + \lambda_l}. \tag{7.67}$$

From the proposed formulation, we see clearly how $\mathbf{H}_{\mathrm{W}}$ and $\mathbf{H}_{\max}$ are related. Besides a (slight) different weighting factor, $\mathbf{H}_{\mathrm{W}}$ considers all directions where the desired signal is present, while $\mathbf{H}_{\max}$ relies only on the direction where the maximum of the desired signal energy is present.

Property 7.1. The output SNR with the Wiener filtering matrix is always greater than or equal to the input SNR, i.e., $\mathrm{oSNR}(\mathbf{H}_{\mathrm{W}}) \geq \mathrm{iSNR}$.

Obviously, we have

$$\mathrm{oSNR}(\mathbf{H}_{\mathrm{W}}) \leq \mathrm{oSNR}(\mathbf{H}_{\max}) \tag{7.68}$$

and, in general,

$$\upsilon_{\mathrm{sd}}(\mathbf{H}_{\mathrm{W}}) \leq \upsilon_{\mathrm{sd}}(\mathbf{H}_{\max}). \tag{7.69}$$

[2] This Wiener filtering matrix is different from the one given in [2] within the same context.

7.4.3 MVDR

The MVDR filtering matrix is obtained directly from the constraint (7.46). Since $\mathbf{B}'^H \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}$ is a full-rank square matrix, we deduce that

$$\mathbf{A}_{\mathrm{MVDR}} = \left(\mathbf{B}'^H \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1} \right)^{-1} . \tag{7.70}$$

As a result, the MVDR filtering matrix is

$$\mathbf{H}_{\mathrm{MVDR}} = \left(\mathbf{B}'^H \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1} \right)^{-1} \mathbf{B}'^H . \tag{7.71}$$

From (7.26), we find that

$$\left(\mathbf{B}'^H \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1} \right)^{-1} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H \mathbf{B}' \mathbf{\Lambda}'^{-1}, \tag{7.72}$$

suggesting that we can formulate the MVDR as

$$\mathbf{H}_{\mathrm{MVDR}} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^H}{\lambda_l} . \tag{7.73}$$

It is worth comparing $\mathbf{H}_{\mathrm{MVDR}}$ with $\mathbf{H}_{\max}$ and $\mathbf{H}_{\mathrm{W}}$.

Property 7.2. The output SNR with the MVDR filtering matrix is always greater than or equal to the input SNR, i.e., $\mathrm{oSNR}\left(\mathbf{H}_{\mathrm{MVDR}}\right) \geq \mathrm{iSNR}$.

We have

$$\mathrm{oSNR}\left(\mathbf{H}_{\mathrm{MVDR}}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\mathrm{W}}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\max}\right) \tag{7.74}$$

and, obviously, with the MVDR filtering matrix, we have no distortion, i.e.,

$$\xi_{\mathrm{sr}}\left(\mathbf{H}_{\mathrm{MVDR}}\right) = 1, \tag{7.75}$$

$$\upsilon_{\mathrm{sd}}\left(\mathbf{H}_{\mathrm{MVDR}}\right) = 0. \tag{7.76}$$

From the obvious relationship between the MVDR and maximum SNR filtering matrices, we can deduce a whole class of minimum distortion filtering matrices:

$$\mathbf{H}_{\mathrm{MD},Q} = \mathbf{R}_{\mathbf{x}_1} \mathbf{\Gamma}_{\underline{\tilde{\mathbf{x}}}\mathbf{x}_1}^H \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\lambda_q}, \tag{7.77}$$

where $1 \leq Q \leq L$. We observe that $\mathbf{H}_{\mathrm{MD},1} = \mathbf{H}_{\max}$ and $\mathbf{H}_{\mathrm{MD},L} = \mathbf{H}_{\mathrm{MVDR}}$. Also, we have

$$\mathrm{oSNR}\left(\mathbf{H}_{\mathrm{MD},L}\right) \leq \mathrm{oSNR}\left(\mathbf{H}_{\mathrm{MD},L-1}\right) \leq \cdots \leq \mathrm{oSNR}\left(\mathbf{H}_{\mathrm{MD},1}\right) = \lambda_1 \tag{7.78}$$

and

$$0 = \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MD},L}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MD},L-1}\right) \leq \cdots \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MD},1}\right). \tag{7.79}$$

7.4.4 Tradeoff

By minimizing the desired signal distortion index with the constraint that the noise reduction factor is equal to a positive value that is greater than 1, we get the tradeoff filtering matrix:

$$\mathbf{H}_{\text{T},\mu} = \mathbf{R}_{\mathbf{x}_1}\boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\sum_{l=1}^{L}\frac{\mathbf{b}_l\mathbf{b}_l^H}{\mu+\lambda_l}, \tag{7.80}$$

where $\mu \geq 0$ is a Lagrange multiplier. We observe that $\mathbf{H}_{\text{T},0} = \mathbf{H}_{\text{MVDR}}$ and $\mathbf{H}_{\text{T},1} = \mathbf{H}_{\text{W}}$.

Property 7.3. The output SNR with the tradeoff filtering matrix is always greater than or equal to the input SNR, i.e., $\text{oSNR}\left(\mathbf{H}_{\text{T},\mu}\right) \geq \text{iSNR},\ \forall\mu \geq 0$.

We should have for $\mu \geq 1$,

$$\text{oSNR}\left(\mathbf{H}_{\text{MVDR}}\right) \leq \text{oSNR}\left(\mathbf{H}_{\text{W}}\right) \leq \text{oSNR}\left(\mathbf{H}_{\text{T},\mu}\right) \leq \text{oSNR}\left(\mathbf{H}_{\max}\right), \tag{7.81}$$

$$0 = \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MVDR}}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{W}}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{T},\mu}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\max}\right), \tag{7.82}$$

and for $\mu \leq 1$,

$$\text{oSNR}\left(\mathbf{H}_{\text{MVDR}}\right) \leq \text{oSNR}\left(\mathbf{H}_{\text{T},\mu}\right) \leq \text{oSNR}\left(\mathbf{H}_{\text{W}}\right) \leq \text{oSNR}\left(\mathbf{H}_{\max}\right), \tag{7.83}$$

$$0 = \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{MVDR}}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{T},\mu}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\text{W}}\right) \leq \upsilon_{\text{sd}}\left(\mathbf{H}_{\max}\right). \tag{7.84}$$

From all what we have seen so far, we can propose a very general noise reduction filtering matrix:

$$\mathbf{H}_{\mu,Q} = \mathbf{R}_{\mathbf{x}_1}\boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1}\sum_{q=1}^{Q}\frac{\mathbf{b}_q\mathbf{b}_q^H}{\mu+\lambda_q}. \tag{7.85}$$

This form encompasses all known optimal filtering matrices. Indeed, it is clear that

- $\mathbf{H}_{0,1} = \mathbf{H}_{\max}$,
- $\mathbf{H}_{1,L} = \mathbf{H}_{\text{W}}$,
- $\mathbf{H}_{0,L} = \mathbf{H}_{\text{MVDR}}$,
- $\mathbf{H}_{0,Q} = \mathbf{H}_{\text{MD},Q}$,
- $\mathbf{H}_{\mu,L} = \mathbf{H}_{\text{T},\mu}$.

In Table 7.1, we summarize all optimal filtering matrices derived in this chapter.

Table 7.1 Optimal linear filtering matrices for binaural signal enhancement in the time domain.

Maximum SNR:	$\mathbf{H}_{\max} = \mathbf{R}_{\mathbf{x}_1} \boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} \frac{\mathbf{b}_1 \mathbf{b}_1^H}{\lambda_1}$
Wiener:	$\mathbf{H}_{\mathrm{W}} = \mathbf{R}_{\mathbf{x}_1} \boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^H}{1 + \lambda_l}$
MVDR:	$\mathbf{H}_{\mathrm{MVDR}} = \mathbf{R}_{\mathbf{x}_1} \boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^H}{\lambda_l}$
MD,Q:	$\mathbf{H}_{\mathrm{MD},Q} = \mathbf{R}_{\mathbf{x}_1} \boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\lambda_q}$
Tradeoff:	$\mathbf{H}_{\mathrm{T},\mu} = \mathbf{R}_{\mathbf{x}_1} \boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} \sum_{l=1}^{L} \frac{\mathbf{b}_l \mathbf{b}_l^H}{\mu + \lambda_l}$
General Tradeoff:	$\mathbf{H}_{\mu,Q} = \mathbf{R}_{\mathbf{x}_1} \boldsymbol{\Gamma}^H_{\underline{\widetilde{\mathbf{x}}}\mathbf{x}_1} \sum_{q=1}^{Q} \frac{\mathbf{b}_q \mathbf{b}_q^H}{\mu + \lambda_q}$

References

1. J. Benesty, J. Chen, and Y. Huang, *Microphone Array Signal Processing*. Berlin, Germany: Springer-Verlag, 2008.
2. J. Benesty, J. Chen, and Y. Huang, "Binaural noise reduction in the time domain with a stereo setup," *IEEE Trans. Audio, Speech, Language Process.*, vol. 19, pp. 2260–2272, Nov. 2011.
3. E. Ollila, "On the circularity of a complex random variable," *IEEE Signal Process. Lett.*, vol. 15, pp. 841–844, 2008.
4. D. P. Mandic and S. L. Goh, *Complex Valued Nonlinear Adaptive Filters: Noncircularity, Widely Linear and Neural Models*. Wiley, 2009.
5. P. O. Amblard, M. Gaeta, and J. L. Lacoume, "Statistics for complex variables and signals–Part I: variables," *Signal Process.*, vol. 53, pp. 1–13, 1996.
6. P. O. Amblard, M. Gaeta, and J. L. Lacoume, "Statistics for complex variables and signals–Part II: signals," *Signal Process.*, vol. 53, pp. 15–25, 1996.
7. B. Picinbono and P. Chevalier, "Widely linear estimation with complex data," *IEEE Trans. Signal Process.*, vol. 43, pp. 2030–2033, Aug. 1995.
8. J. N. Franklin, *Matrix Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1968.

Appendix A
Auxiliary MATLAB Functions

Listing A.1 Function for computing the joint diagonalization.

```
function [X,D]=jeig(A,B,srtstr);
L=chol(B,'lower');
G=inv(L);
C=G*A*G';
[Q,D]=schur(C);
X=G'*Q;

if srtstr,
    d = diag(D);
    [ds,is] = sort(d,'descend');

    D = diag(ds);
    X = X(:,is);
end
```

Listing A.2 Function for computing the STFT with a rectangular synthesis window.

```
function [stft,freq,time,blocks] = stftBatch(x,winLen,nFft,sampFreq)

% for periodic hann window
win = hann(winLen,'periodic');

% inits
n = 1:winLen;
iCol = 1;

nLowerSpec = ceil((1+nFft)/2);

% perform stft
while n(end) <= length(x);

    % obtain block and window
    xBlock = x(n);
    xwBlock = xBlock;
    blocks(:,iCol) = xwBlock;

    % fft
    X = fft(xwBlock,nFft);

    % stft matrix
    stft(:,iCol) = X(1:nLowerSpec);

    % update indices
    n = n + winLen/2;
    iCol = iCol + 1;
    iTime(iCol) = mean(n-1);
end

% calc time + freq vectors
freq = (0:nLowerSpec-1)*sampFreq/nFft;
time = iTime/sampFreq;
```

Listing A.3 Function for computing the inverse STFT with a Hanning synthesis window.

```
function [x] = stftInvBatch(stft,winLen,nFft,sampFreq)

% for periodic hann window
win = hann(winLen,'periodic');

% inits
n = 1:winLen;
iCol = 1;

nFrames = size(stft,2);
x = zeros(winLen+nFrames*winLen/2,1);

% perform inv stft
while iCol <= size(stft,2),
    % obtain block and window
    xBlock = ifft([stft(:,iCol);flipud(conj(stft(2:end-1,iCol)))],nFft);
    xBlock = xBlock(1:winLen);
    xwBlock = xBlock.*win;

    x(n) = x(n) + xwBlock;

    % update indices
    n = n + winLen/2;
    iCol = iCol + 1;
end
```

Listing A.4 Function for generating sensor coordinates for a uniform linear array.

```
function micPos = generateUlaCoords(nodePos,nMic,spacing,dir,height)

tmpX = (0:nMic-1)*spacing-(nMic-1)*spacing/2;
tmpY = zeros(1,nMic);
tmp = [tmpX;tmpY];

rotMat = [cosd(dir),-sind(dir);sind(dir),cosd(dir)];

tmpRot = rotMat*tmp;
micPos(1:2,:) = tmpRot+nodePos(1:2,1)*ones(1,nMic);
micPos(3,:) = height;

end
```

Listing A.5 Function for generating a multichannel signal with reverberation, diffuse babble noise, and white Gaussian sensor noise. The code for the function

`rir_generator()` is documented in [1] and is online available at http://www.audiolabs-erlangen.de/fau/professor/habets/software/rir-generator.

```matlab
function [mcSignals,setup] = multichannelSignalGenerator(setup)

addpath([cd,'\..\rirGen\']);

setup.nRirLength = 2048;
setup.hpFilterFlag = 1;
setup.reflectionOrder = -1;
setup.micType = 'omnidirectional';

setup.roomDim = [3;4;3];

srcHeight = 1.5;
arrayHeight = 1;

arrayCenter = [setup.roomDim(1:2)/2;1];

arrayToSrcDistInt = [0.75,1];

setup.srcPoint = [0.75;1;1.5];

setup.micPoints = generateUlaCoords(arrayCenter,setup.nSensors,...
    setup.sensorDistance,0,arrayHeight);

[cleanSignal,setup.sampFreq] = audioread(...
    '..\data\twoMaleTwoFemale20Seconds.wav');

if setup.reverbTime == 0,
    setup.reverbTime = 0.2;
    reflectionOrder = 0;
else
    reflectionOrder = -1;
end

rirMatrix = rir_generator(setup.speedOfSound,setup.sampFreq,...
    setup.micPoints',setup.srcPoint',setup.roomDim',...
    setup.reverbTime,setup.nRirLength,setup.micType,...
    setup.reflectionOrder,[],[],setup.hpFilterFlag);

for iSens = 1:setup.nSensors,
    tmpCleanSignal(:,iSens) = fftfilt(rirMatrix(iSens,:)',cleanSignal);
end
mcSignals.clean = tmpCleanSignal(setup.nRirLength:end,:);
setup.nSamples = length(mcSignals.clean);

mcSignals.clean = mcSignals.clean - ones(setup.nSamples,1)*...
    mean(mcSignals.clean);

cleanSignalPowerMeas = var(mcSignals.clean);


mcSignals.diffNoise = generateMultichanBabbleNoise(setup.nSamples,...
    setup.nSensors,setup.sensorDistance,...
    setup.speedOfSound,setup.noiseField);
diffNoisePowerMeas = var(mcSignals.diffNoise);
diffNoisePowerTrue = cleanSignalPowerMeas/10^(setup.sdnr/10);
mcSignals.diffNoise = mcSignals.diffNoise*...
    diag(sqrt(diffNoisePowerTrue)./sqrt(diffNoisePowerMeas));

mcSignals.sensNoise = randn(setup.nSamples,setup.nSensors);
sensNoisePowerMeas = var(mcSignals.sensNoise);
sensNoisePowerTrue = cleanSignalPowerMeas/10^(setup.ssnr/10);
mcSignals.sensNoise = mcSignals.sensNoise*...
    diag(sqrt(sensNoisePowerTrue)./sqrt(sensNoisePowerMeas));

```

```
mcSignals.noise = mcSignals.diffNoise + mcSignals.sensNoise;
mcSignals.observed = mcSignals.clean + mcSignals.noise;
```

Listing A.6 Function for generating a multichannel, diffuse, babble noise. This function is inspired by [2] and code available at http://www.audiolabs-erlangen.de/fau/professor/habets/software/noise-generators. The code for the function `mix_signals()` and the functions used therein is also available from the aforementioned URL.

```
function [ multichannelBabble ] = generateMultichanBabbleNoise(...
    nSamples,nSensors,sensorDistance,speedOfSound,noiseField)

nFft = 256;

[singleChannelData,samplingFreq] = audioread('babble_8kHz.wav');

if (nSamples*nSensors)>length(singleChannelData),
    error([...
    '[nSamples]x[nSensors] exceeds the length of the noise signal. ',...
    'Maximum length with ',num2str(nSensors),' sensors is: ',...
    num2str(floor(length(singleChannelData)/nSensors))]);
end

singleChannelData = singleChannelData - mean(singleChannelData);
babble = zeros(nSamples,nSensors);
for iSensors=1:nSensors
    babble(:,iSensors) = singleChannelData((iSensors-1)*nSamples+1:...
        iSensors*nSamples);
end

%% Generate matrix with desired spatial coherence
ww = 2*pi*samplingFreq*(0:nFft/2)/nFft;
desiredCoherence = zeros(nSensors,nSensors,nFft/2+1);
for p = 1:nSensors
    for q = 1:nSensors
        if p == q
            desiredCoherence(p,q,:) = ones(1,1,nFft/2+1);
        else
            switch lower(noiseField)
                case 'spherical'
                    desiredCoherence(p,q,:) = sinc(ww*abs(p-q)*...
                        sensorDistance/(speedOfSound*pi));

                case 'cylindrical'
                    desiredCoherence(p,q,:) = bessel(0,ww*abs(p-q)*...
                        sensorDistance/speedOfSound);

                otherwise
                    error('Unknown noise field.');
            end
        end
    end
end

%% Generate sensor signals with desired spatial coherence
multichannelBabble = mix_signals(babble,desiredCoherence,'cholesky');

end
```

References

1. E. A. P. Habets, "Room impulse response generator," Tech. Rep., Technische Universiteit Eindhoven, 2010, Ver. 2.0.20100920.
2. E. A. P. Habets, I. Cohen, and S. Gannot, "Generating nonstationary multisensor signals under a spatial coherence constraint," *J. Acoust. Soc. Am.*, vol. 124, pp. 2911–2917, Nov. 2008.

Index