Algorithmic Enhancements to Polynomial Matrix Factorisations

PhD Thesis

Fraser Kenneth Coutts

Centre for Signal and Image Processing
Electronic and Electrical Engineering
University of Strathclyde, Glasgow

May 29, 2019

Abstract

In broadband array processing applications, an extension of the eigenvalue decomposition (EVD) to parahermitian Laurent polynomial matrices — named the polynomial matrix EVD (PEVD) — has proven to be a useful tool for the decomposition of space-time covariance matrices and their associated cross-spectral density matrices. Existing PEVD methods typically operate in the time domain and utilise iterative frameworks established by the second-order sequential best rotation (SBR2) or sequential matrix diagonalisation (SMD) algorithms. However, motivated by recent discoveries that establish the existence of an analytic PEVD — which is rarely recovered by SBR2 or SMD — alternative algorithms that better meet analyticity by operating in the discrete Fourier transform (DFT)-domain have received increasing attention.

While offering promising results in applications including broadband MIMO and beamforming, the PEVD has seen limited deployment in hardware due to its high computational complexity. If the PEVD is to be fully utilised, overcoming this bottleneck is paramount. This thesis therefore seeks to reduce the computational cost of iterative PEVD algorithms — with particular emphasis on SMD — through the development of several novel algorithmic improvements. While these are effective, the complexity of the optimised algorithms still grows rapidly with the spatial dimensions of the decomposition. Steps are therefore taken to convert the sequential form of SMD to a novel reduced dimensionality and partially parallelisable divide-and-conquer architecture. The resulting algorithms are shown to converge an order of magnitude faster than existing methods for large spatial dimensions, and are well-suited to application scenarios with many sensors.

Further in this thesis, an investigation into DFT-based algorithms highlights their potential to offer compact, analytic solutions to the PEVD. Subsequently, two novel DFT-based algorithms improve upon an existing method by reducing decomposition error and eliminating a priori knowledge requirements. Finally, an innovative strategy is shown to be capable of extracting a minimum-order solution to the PEVD.

Acknowledgements

I would like to begin by extending my sincere thanks to my supervisors, Professor Stephan Weiss and Professor Stephen Marshall, who have kindly provided me with their expert guidance, encouragement, and key insights throughout my time at the University of Strathclyde. For their valuable advice and input to my research, I would also like to thank Dr Keith Thompson, Professor Ian Proudler, Dr Jamie Corr, Dr Jennifer Pestana, Professor John McWhirter, and Dr Paul Murray. I am also extremely grateful to the Carnegie Trust, who funded my research.

Many thanks to Dr Louise Crockett and Dr Gaetano Di Caterina for trusting me with their classes, and to Christine Bryson for her excellent administrative skills. For their conversation and thoughts of the day, I would like to thank Dani, Kenny, Shawn, Connor, Paul, David, Vianney, and the other Fraser.

I would like to extend my deepest gratitude to my parents, Karen and Kenny, for always believing in me, for motivating me, and for being fantastic sources of inspiration. At last, I'm no longer a student! Thanks should also go to Lauren, James, my grandparents, and my extended family, for their encouragement.

For her invaluable support, input, and Oxford comma checking prowess during the completion of this research, I would like to offer my profound gratitude to Elizabeth.

Fraser Kenneth Coutts

Glasgow, UK

May 29, 2019

Contents

List of Figures

List of Tables

List of Publications

Publications Relating to this Research as First Author

F. K. Coutts, J. Corr, K. Thompson, S. Weiss, I. K. Proudler, and J. G. McWhirter, "Memory and Complexity Reduction in Parahermitian Matrix Manipulations of PEVD Algorithms," in Proceedings of the 24th European Signal Processing Conference, August 2016, pp. 1633–1637.

F. K. Coutts, J. Corr, K. Thompson, S. Weiss, I. K. Proudler, and J. G. McWhirter, "Complexity and Search Space Reduction in Cyclic-by-Row PEVD Algorithms," in Proceedings of the 50th Asilomar Conference on Signals, Systems and Computers, November 2016, pp. 1349–1353.

F. K. Coutts, J. Corr, K. Thompson, S. Weiss, I. K. Proudler, and J. G. McWhirter, "Multiple Shift QR Decomposition for Polynomial Matrices," in Proceedings of the 11th IMA International Conference on Mathematics in Signal Processing, December 2016.

F. K. Coutts, K. Thompson, S. Weiss, and I. K. Proudler, "Analysing the Performance of Divide-and-Conquer Sequential Matrix Diagonalisation for Large Broadband Sensor Arrays," in Proceedings of the 2017 IEEE International Workshop on Signal Processing Systems, October 2017.

F. K. Coutts, J. Corr, K. Thompson, I. K. Proudler, and S. Weiss, "Divide-and-Conquer

Sequential Matrix Diagonalisation for Parahermitian Matrices," in Proceedings of the 2017 Sensor Signal Processing for Defence Conference, December 2017.

F. K. Coutts, K. Thompson, S. Weiss, and I. K. Proudler, "Impact of Fast-Converging PEVD algorithms on Broadband AoA Estimation," in Proceedings of the 2017 Sensor Signal Processing for Defence Conference, December 2017.

F. K. Coutts, K. Thompson, I. K. Proudler, and S. Weiss, "A Comparison of Iterative and DFT-Based Polynomial Matrix Eigenvalue Decompositions," in Proceedings of the 7th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, December 2017.

F. K. Coutts, K. Thompson, I. K. Proudler, and S. Weiss, "Restricted Update Sequential Matrix Diagonalisation for Parahermitian Matrices," in Proceedings of the 7th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, December 2017.

F. K. Coutts, K. Thompson, J. Pestana, I. K. Proudler, and S. Weiss, "Enforcing Eigenvector Smoothness for a Compact DFT-Based Polynomial Eigenvalue Decomposition," in Proceedings of the 10th IEEE Sensor Array and Multichannel Signal Processing Workshop, July 2018, pp. 159–163.

F. K. Coutts, K. Thompson, I. K. Proudler, and S. Weiss, "An Iterative DFT-Based Approach to the Polynomial Matrix Eigenvalue Decomposition," in Proceedings of the 52nd Asilomar Conference on Signals, Systems and Computers, October 2018, pp. 1011–1015.

F. K. Coutts, I. K. Proudler, and S. Weiss, "Efficient Implementation of Iterative Polynomial Matrix EVD Algorithms Exploiting Structural Redundancy and Parallelisation," in IEEE Transactions on Circuits and Systems I, submitted March 2019.

Publications Relating to this Research as Co-author

S. Weiss, S. Bendoukha, A. Alzin, F. K. Coutts, I. K. Proudler, and J. Chambers, "MVDR Broadband Beamforming Using Polynomial Matrix Techniques," in Proceedings of the 23rd European Signal Processing Conference, September 2015, pp. 839–843.

A. Alzin, F. K. Coutts, J. Corr, S. Weiss, I. K. Proudler, and J. A. Chambers, "Adaptive Broadband Beamforming with Arbitrary Array Geometry," in Proceedings of the 2nd IET International Conference on Intelligent Signal Processing, December 2015.

A. Alzin, F. K. Coutts, J. Corr, S. Weiss, I. K. Proudler, and J. A. Chambers, "Polynomial Matrix Formulation-Based Capon Beamformer," in Proceedings of the 11th IMA International Conference on Mathematics in Signal Processing, December 2016.

C. Delaosa, F. K. Coutts, J. Pestana, and S. Weiss, "Impact of Space-Time Covariance Estimation Errors on a Parahermitian Matrix EVD," in Proceedings of the 10th IEEE Sensor Array and Multichannel Signal Processing Workshop, July 2018, pp. 164–168.

S. Weiss, J. Pestana, I. K. Proudler, and F. K. Coutts, "Corrections to "On the Existence and Uniqueness of the Eigenvalue Decomposition of a Parahermitian Matrix"," in IEEE Transactions on Signal Processing, vol. 66, no. 23, pp. 6325–6327, December 2018.

S. Weiss, I. K. Proudler, F. K. Coutts, and J. Pestana, "Iterative Approximation of Analytic Eigenvalues of a Parahermitian Matrix EVD," in Proceedings of the 44th International Conference on Acoustics, Speech, and Signal Processing, May 2019.

Publications Relating to Other Research

F. K. Coutts, S. Marshall, and P. Murray, "Human Detection and Tracking Through Temporal Feature Recognition," in Proceedings of the 22nd European Signal Processing Conference, September 2014, pp. 2180–2184.

D. Gaglione, C. Clemente, F. K. Coutts, G. Li, and J. J. Soraghan, "Model-Based Sparse Recovery Method for Automatic Classification of Helicopters," in Proceedings of the IEEE Radar Conference, May 2015, pp. 1161–1165.

F. K. Coutts, D. Gaglione, C. Clemente, G. Li, I. K. Proudler, and J. J. Soraghan, "Label Consistent K-SVD for Sparse Micro-Doppler Classification," in Proceedings of the IEEE International Conference on Digital Signal Processing, July 2015, pp. 90–94.

C. G. Manich, T. Kelman, F. K. Coutts, B. Qiu, P. Murray, C. Gonzalez-Longo, and S. Marshall, "Exploring the Use of Image Processing to Survey and Quantitatively Assess Historic Buildings," in Proceedings of the 10th International Conference on Structural Analysis of Historical Constructions, September 2016.

A. Polak, F. K. Coutts, P. Murray, and S. Marshall, "Use of Hyperspectral Imaging for Cake Moisture and Hardness Prediction," in IET Image Processing, March 2019.

Abbreviations and Mathematical Symbols

Abbreviations

AEVD	approximate eigenvalue decomposition
AoA	angle of arrival
BC	by columns
CbR	cyclic-by-row
CRST	compensated row-shift truncation
CSD	cross-spectral density
DaC	divide-and-conquer
DC-SMD	divide-and-conquer sequential matrix diagonalisation
DFT	discrete Fourier transform
EPGR	elementary polynomial Givens rotation
EVD	eigenvalue decomposition
FFT	fast Fourier transform
FIR	finite impulse response
GCD	greatest common divisor
HRSMD	half-matrix restricted update sequential matrix diagonalisation
HRSMS	half-matrix restricted update sequential matrix segmentation
HSMD	half-matrix sequential matrix diagonalisation
HSMS	half-matrix sequential matrix segmentation
IDFT	inverse discrete Fourier transform

IFB	independent frequency bin
IFFT	inverse fast Fourier transform
MAC	multiply-accumulate
MIMO	multiple-input multiple-output
MS	multiple shift
MSE	mean square error
MSME	multiple shift maximum element
MUSIC	multiple signal classification
PEVD	polynomial matrix eigenvalue decomposition
PQRD	polynomial matrix QR decomposition
PSD	power spectral density
PSMD	parallel-sequential matrix diagonalisation
PSVD	polynomial matrix singular value decomposition
RS	reduced search space
RSMD	restricted update sequential matrix diagonalisation
RST	row-shift truncation
SBR2	second-order sequential best rotation
SMD	sequential matrix diagonalisation
SMS	sequential matrix segmentation
SSP-MUSIC	spatio-spectral polynomial multiple signal classification
SVD	singular value decomposition

Mathematical Symbols

$\mathbf{A}$	Matrix of coefficients
$\mathbf{a}$	Vector of coefficients
a, A	Scalar
$\mathbf{A}[\tau]$	Matrix of coefficients dependent on a discrete time variable τ
$\mathbf{a}[\tau]$	Vector of coefficients dependent on a discrete time variable τ
$a[\tau]$, $A[\tau]$	Scalar dependent on a discrete time variable τ
$\mathbf{A}[k]$	Matrix of coefficients dependent on a discrete frequency variable k
$\mathbf{a}[k]$	Vector of coefficients dependent on a discrete frequency variable k
$\mathsf{a}[k]$, $\mathsf{A}[k]$	Scalar dependent on a discrete frequency variable k
$\boldsymbol{A}(z)$	Matrix of polynomials dependent on a continuous variable z
$\boldsymbol{a}(z)$	Vector of polynomials dependent on a continuous variable z
$a(z)$, $A(z)$	Polynomial dependent on a continuous variable z
$\boldsymbol{A}(\mathrm{e}^{\mathrm{j}\Omega})$	Polynomial matrix $\boldsymbol{A}(z)$ evaluated on the unit circle with $z = \mathrm{e}^{\mathrm{j}\Omega}$
$\boldsymbol{a}(\mathrm{e}^{\mathrm{j}\Omega})$	Polynomial vector $\boldsymbol{a}(z)$ evaluated on the unit circle with $z = \mathrm{e}^{\mathrm{j}\Omega}$
$a(\mathrm{e}^{\mathrm{j}\Omega})$, $A(\mathrm{e}^{\mathrm{j}\Omega})$	Polynomial $a(z)$, $A(z)$ evaluated on the unit circle with $z = \mathrm{e}^{\mathrm{j}\Omega}$
$\mathbf{R}[\tau]$	Space-time covariance matrix
$\boldsymbol{R}(z)$	Cross-spectral density matrix, parahermitian matrix
$\boldsymbol{Q}(z), \boldsymbol{F}(z)$	Polynomial eigenvectors, paraunitary matrix
$\boldsymbol{D}(z)$	Polynomial eigenvalues, diagonal matrix
$\boldsymbol{B}(z)$	Block diagonal matrix
$\boldsymbol{S}^{(i)}(z)$	Parahermitian matrix dependent on iteration variable i
$\boldsymbol{H}^{(i)}(z)$	Paraunitary matrix dependent on iteration variable i
$\boldsymbol{\Lambda}^{(i)}(z)$	Paraunitary shifting matrix dependent on iteration variable i
$\mathbf{Q}^{(i)}$	Unitary rotation matrix dependent on iteration variable i
$E_{\mathrm{diag}}, E_{\mathrm{diag}}^{(i)}$	Diagonalisation metric (dependent on iteration variable i)
$I_{\max}$	Maximum number of algorithm iterations
K	DFT length, number of frequency bins
L	Length (i.e., order + 1) of parahermitian matrix

M	Spatial dimension, number of time series, number of sensors
O_Q	Order of ground truth polynomial eigenvectors
O_D	Order of ground truth polynomial eigenvalues
P	Spatial dimension, parahermitian matrix 'division' parameter
T	Maximum lag of parahermitian matrix
Δ_{DR}	Dynamic range
ϵ	Threshold used to cease algorithm iterations
η	Paraunitarity error
θ	Phase angle
ϑ	Angle of arrival, azimuth
κ	Threshold used to cease algorithm iterations
λ_{res}	Eigenvalue resolution
μ	Polynomial matrix truncation parameter
$\chi^{(n)}$	Smoothness metric computed up the nth derivative
Ω	Continuous frequency
Ω_k	Discrete frequency dependent on a discrete frequency variable k
$\mathbf{I}_M$	$M \times M$ identity matrix
$\mathbf{0}_M$	$M \times M$ matrix of zeroes
$\circ\!\!-\!\!\bullet$	Forward z-transform
$\bullet\!\!-\!\!\circ$	Reverse z-transform
$\{\cdot\}^{\mathrm{T}}$	Transpose operator
$\{\cdot\}^{\mathrm{H}}$	Hermitian transpose operator
$\tilde{\{\cdot\}}$	Parahermitian conjugate operator
$\mathcal{E}\{\cdot\}$	Expectation operator
$\|\cdot\|_{\mathrm{F}}$	Frobenius norm operator
$\overline{\{\cdot\}}$	Operator to obtain half-matrix form of parahermitian matrix
$L\{\cdot\}$	Computes the length (i.e., order + 1) of a polynomial matrix
$*$	Convolution operator
$\circledast$	Circular convolution operator

Chapter 1

Introduction

1.1 Polynomial Matrix Formulations

The eigenvalue decomposition (EVD) is a useful tool for many narrowband problems involving Hermitian instantaneous covariance matrices [1, 2]. In broadband array processing or multichannel time series applications, an instantaneous covariance matrix is not sufficient to measure correlation of signals across time delays. Instead, a space-time covariance matrix — which is more suited to the capture of broadband information — is used, which comprises the auto- and cross-correlation sequences obtained from multiple time series. Its z-transform, the cross-spectral density (CSD) matrix, is a Laurent polynomial matrix in $z \in \mathbb{C}$, and inherits the symmetries of the auto- and cross-correlation sequences, such that it exhibits parahermitian symmetry [3, 4].

A polynomial matrix eigenvalue decomposition (PEVD) has been defined as an extension of the EVD to parahermitian polynomial matrices in [5, 6]. The PEVD uses finite impulse response (FIR) paraunitary matrices [7] to approximately diagonalise and typically spectrally majorise [8] a CSD matrix and its associated space-time covariance matrix. Recent work in [9, 10] provides conditions for the existence and uniqueness of eigenvalues and eigenvectors of a PEVD, such that these can be represented by a power or Laurent series that is absolutely convergent, permitting a direct realisation in the time domain. Further research in [11] studies the impact of estimation errors in the sample space-time covariance matrix on its PEVD.

Once broadband multichannel problems have been expressed using polynomial matrix formulations, solutions to these problems can be obtained via the PEVD. For example, the PEVD has been successfully used in broadband multiple-input multiple-output (MIMO) precoding and equalisation using linear [12–14] and non-linear [15, 16] approaches, broadband angle of arrival estimation [17–20], broadband beamforming [21–24], optimal subband coding [8, 25], joint source-channel coding [26, 27], blind source separation [28] via a polynomial matrix generalised EVD [29], and scene discovery [30]. Several broadband MIMO problems [31–39] utilise the broadband singular value decomposition (SVD). Such a decomposition can be computed using two PEVDs, multiple polynomial matrix QR decompositions [40–43] or directly via a polynomial matrix SVD [44].

Existing PEVD algorithms include second-order sequential best rotation (SBR2) [6], sequential matrix diagonalisation (SMD) [45], and various evolutions of both algorithm families [25, 46–50]. Different from the fixed order time domain schemes in [51, 52] and discrete Fourier transform (DFT)-based approaches in [53, 54], SBR2 and SMD have proven convergence. Both algorithms employ iterative time domain schemes to approximately diagonalise a parahermitian matrix, and encourage spectral majorisation such that the power spectral densities (PSDs) of the resulting eigenvalues are ordered at all frequencies [8]; indeed, SBR2 has been shown to converge to this solution [55]. A DFT-based PEVD formulation, which transforms the problem into a pointwise-in-frequency standard matrix decomposition, is provided in [53]. This algorithm has been shown to perform well for finite order problems [56], but requires an a priori estimate of the length of the paraunitary matrix in the decomposition. Such DFT-based PEVD methods have received renewed interest in recent years [54], due to their ability to return either a spectrally majorised decomposition, or attempt to approximate maximally smooth, analytic eigenvalues, which are necessary for good eigenvector estimation [9,10].

While offering promising results, the PEVD has seen limited deployment in hardware. A parallel form of SBR2 whose performance has little dependency on the size of the input parahermitian matrix has been designed and implemented on an FPGA [57–59], but the SMD algorithm, which can achieve superior levels of diagonal-

isation [45], has been restricted to software applications due to its high computational complexity requirements and non-parallelisable architecture. Efforts to reduce the algorithmic cost of iterative PEVD algorithms, including SMD, have mostly been focussed on the trimming of polynomial matrices to curb growth in order [6, 60–62], which translates directly into a growth of computational complexity. By applying a row-shift truncation scheme for paraunitary matrices in [62–64], the polynomial order can be reduced with little loss to paraunitarity of the eigenvectors. These efforts, alongside a low-cost cyclic-by-row numerical approximation of the EVD [49] and methods that operate over reduced parameter sets [48, 50] have nonetheless not been able to reduce computational cost sufficiently to invite a hardware realisation. Given the many applications where these polynomial matrix techniques have the potential to make a great impact, overcoming the bottleneck of computational complexity in the implementation of PEVD algorithms is paramount.

1.2 Objective of Research

The fundamental aim of this research is to take note of existing methods for computing a PEVD, before developing improved and novel algorithms capable of achieving a comparable or superior decomposition through alternative means. Within this aim, the first — and most expansive — objective is to reduce the computational complexity of PEVD algorithms. Of particular interest in this regard are SMD-based algorithms, which have shown great promise in software simulations [45, 48–50] but are yet to be implemented in hardware. If the cost of this family of algorithms is adequately reduced, future work can create solutions for broadband signal processing applications that offer superior performance to existing implementations [57–59].

As demonstrated in previous research [65, 66], the computational cost of PEVD algorithms strongly depends on the dimensions of the parahermitian matrix in the decomposition. Thus, a second objective is to investigate dimensionality reduction techniques for the PEVD to create a technique that can be applicable to complex and high-dimensional data sets, which might be created by, for example, large broadband sensor arrays.

Finally, given the potential for DFT-based PEVD algorithms to approximate an analytic decomposition [53, 54, 56], which would more closely satisfy the conditions for a potentially minimum-order, absolutely convergent Laurent series solution than iterative time domain algorithms [9, 10], a third objective relates to an investigation into DFT-based approaches to the PEVD. This objective extends to the creation of novel DFT-based PEVD algorithms with superior performance to existing methods.

1.3 Organisation of Thesis and Original Contributions

To set the scene for the contributions described in subsequent chapters, an overview of the motivation behind the use of polynomial matrix formulations in broadband signal processing is provided in Chapter 2. This chapter also introduces the PEVD and gives a brief summary of the existing PEVD algorithms in the literature.

To satisfy the first of the objectives listed above, Chapter 3 discusses a number of novel methods to lower the computational cost of existing iterative algorithms related to the PEVD. The proposed methods conquer unexplored areas of PEVD algorithm improvement by optimising matrix manipulations in software, exploiting parahermitian matrix symmetry [67], minimising algorithmic redundancy [68], and reducing the search and update space of iterative PEVD algorithms [68, 69]. Since repeated application of the polynomial matrix QR decomposition (PQRD) to a parahermitian matrix can be used to form a PEVD, improvements are also made to an existing PQRD algorithm [70], such that PEVD implementations that rely on the PQRD can also benefit. Furthermore, in research encompassed by the first two objectives, improvements to an existing polynomial matrix truncation strategy yield a method capable of reducing polynomial matrix order without significantly impacting on decomposition mean square error.

While each of the methods discussed in Chapter 3 decrease the implementation costs of various PEVD approaches, the complexity of the algorithms grows rapidly with the spatial dimensions of the parahermitian matrix, such that even the improved PEVD algorithms are not well-suited for applications involving large broadband arrays. Chapter 4 addresses this problem — and the second objective of this research — by taking additional steps to convert the sequential form of existing PEVD algorithms

to a reduced dimensionality, partially parallelisable divide-and-conquer (DaC) architecture [71,72]. In the proposed DaC approach, a large parahermitian matrix is segmented into multiple independent parahermitian matrices of smaller spatial dimensions, which are subsequently diagonalised independently. Through these efforts, PEVD algorithms are created that exhibit convergence speeds an order of magnitude faster than existing methods for parahermitian matrices with large spatial dimensions [64]. In contrast to the current state-of-the-art approaches, the developed algorithms are shown to be well-suited to deployment in application scenarios [20].

Of particular interest are DFT-based PEVD algorithms, which offer potentially analytic and subsequently minimum-order solutions to the PEVD. In the process of completing the third objective of this research, Chapter 5 briefly introduces the concept of DFT-based PEVD algorithms before comparing an example of such an algorithm with an iterative time-based PEVD algorithm [56]. A subsequent section then details the formation of a novel DFT-based algorithm capable of outperforming an existing method while requiring less a priori knowledge regarding the order of the paraunitary matrices in the decomposition [73]. An extension of this algorithm, which uses an iterative approach to remove all a priori knowledge requirements, is then found to perform well relative to existing iterative PEVD algorithms [74].

Finally, Chapter 6 summarises the major novel contributions in this thesis, and proposes a number of future directions for further research in this area.

Chapter 2

Background

Following an introduction to the notations and definitions used throughout this thesis in Section 2.1, Section 2.2 provides a brief overview of the motivation behind the use of polynomial matrix formulations in broadband signal processing. Subsequently, Section 2.3 establishes the context for the contributions described in subsequent chapters by defining the polynomial matrix eigenvalue decomposition and listing the existing algorithms designed to compute such a decomposition. Finally, Section 2.4 defines the software and hardware utilised for the majority of the simulations in this thesis.

2.1 Notations and Definitions

In this thesis, upper- and lower-case boldface variables, such as $\mathbf{A}$ and $\mathbf{a}$, refer to matrix- and vector-valued quantities, respectively. The mth element in a vector or diagonal matrix is represented by a lower-case variable with subscript m, e.g., a_m, and the element shared by the mth row and nth column of a matrix is similarly represented with subscript m, n — such as in $a_{m,n}$. Let $\mathbf{I}_M$ and $\mathbf{0}_M$ denote identity and zero matrices of dimension $M \times M$, respectively. In addition, $\mathbb{Z}$ is the set of integers, $\mathbb{N}$ is the subset of positive integers, $\mathbb{R}$ is the field of real numbers, and $\mathbb{C}$ is the field of complex numbers. A dependency on a continuous and discrete variable is indicated via parentheses and square brackets, respectively; examples of this are $\mathbf{A}(t)$, $t \in \mathbb{R}$, and $\mathbf{a}[n]$, $n \in \mathbb{Z}$. Polynomial quantities, such as $\boldsymbol{A}(z)$, are denoted by their dependency on z and italic font. To facilitate a distinction between the time and frequency domain, quantities

dependent on a discrete frequency variable are denoted by sans-serif notation; examples of this are $\mathsf{a}[k]$, $\mathsf{\mathbf{a}}[k]$, and $\mathsf{\mathbf{A}}[k]$. The expectation operator is denoted as $\mathcal{E}\{\cdot\}$, $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm, and $\{\cdot\}^{\mathrm{H}}$ indicates a Hermitian transpose. When applied to a polynomial matrix, the latter is taken to mean the Hermitian transpose of all polynomial coefficient matrices and z, i.e., $\boldsymbol{R}^{\mathrm{H}}(z) = \boldsymbol{R}_*^{\mathrm{T}}(z^*)$, where $\boldsymbol{R}_*(z)$ denotes conjugation of the coefficients of $\boldsymbol{R}(z)$ without conjugating z and $\{\cdot\}^{\mathrm{T}}$ denotes the transpose. The parahermitian conjugate operator $\tilde{\{\cdot\}}$ implies a Hermitian transpose of each coefficient matrix and a replacement of z by z^{-1}, such that $\tilde{\boldsymbol{R}}(z) = \boldsymbol{R}_*^{\mathrm{T}}(z^{-1}) = \boldsymbol{R}^{\mathrm{H}}(1/z^*)$ [4]; i.e., a Hermitian transposition and time reversal of the corresponding time domain quantity. In this thesis, the order of a Laurent polynomial matrix $\boldsymbol{A}(z) = \sum_{n=a}^{b} \mathbf{A}_n z^n$ — for $a, b \in \mathbb{Z}$, $a \leq b$, $\mathbf{A}_n \in \mathbb{C}^{M\times M}$, $\mathbf{A}_a \neq \mathbf{0}_M$, and $\mathbf{A}_b \neq \mathbf{0}_M$ — is $b-a$ [7]. The length of a Laurent polynomial matrix is defined as its order plus one.

2.2 Polynomial Matrix Formulations in Broadband Signal Processing

2.2.1 Introduction

The Hermitian instantaneous covariance matrix $\mathbf{R} = \mathcal{E}\{\mathbf{x}[n]\mathbf{x}^{\mathrm{H}}[n]\}$, which captures the correlation and phase information present in a zero mean, multichannel signal $\mathbf{x}[n] \in \mathbb{C}^M$, is the subject of matrix decompositions at the centre of many optimal and robust solutions to narrowband array processing problems [1,2]. In particular, the eigenvalue decomposition (EVD) of a positive semi-definite matrix $\mathbf{R}$, which uses a unitary matrix $\mathbf{Q}$ and diagonal matrix $\mathbf{D}$ to decompose $\mathbf{R}$ such that

$$\mathbf{R} = \mathbf{Q}\mathbf{D}\mathbf{Q}^{\mathrm{H}}\,, \tag{2.1}$$

has proven to be a useful tool in such scenarios. For example, it is at the heart of the Karhunen–Loève transform for optimal data compaction [75], the widely utilised principal component analysis approach to dimensionality reduction [76], and the multiple signal classification (MUSIC) algorithm for the estimation of source frequency and

location [77].

The above EVD applications are well suited to narrowband signal processing, where matrices only consist of complex gain factors, or correlations are sufficiently defined by instantaneous covariance matrices. This model only considers the instantaneous mixing of signals, and is not necessarily suitable for all applications. For example, in the case of a broadband sensor array, the propagation of signals from sources to sensors cannot be modelled by a scalar mixing matrix; i.e., information relating to the angle of arrival is embedded in the relative time delay of each signal rather than a simple phase shift. A matrix of finite impulse response (FIR) filters is required instead. If each filter is represented as a polynomial in $z \in \mathbb{C}$, the propagation model takes the form of a polynomial mixing (or convolutive mixing [78]) matrix. Convolutive mixing can also be used to model the effects of multipath propagation, which constitutes an important factor in many areas of sensor array signal processing.

2.2.2 Space-Time Covariance and Cross-Spectral Density Matrices

In the case of a broadband sensor array or convolutively mixed signals, the sensor outputs will generally be correlated with one another across explicit delays instead of phase shifts. If the second order statistics of broadband signals is to be captured directly, the relative delays must be carried forward — ideally through space-time covariance matrices $\mathbf{R}[\tau]$, where each entry is not just a single correlation coefficient as in $\mathbf{R}$, but an entire auto- or cross-correlation sequence with a discrete lag parameter τ. A space-time covariance matrix

$$\mathbf{R}[\tau] = \mathcal{E}\{\mathbf{x}[n]\mathbf{x}^{\mathrm{H}}[n-\tau]\} \tag{2.2}$$

can be constructed from a vector-valued time series $\mathbf{x}[n] \in \mathbb{C}^M$, which depends on the discrete time index n and is assumed to be zero mean. A visualisation of an array collecting broadband data is provided in Figure 2.1. Auto-correlation functions of the M measurements in $\mathbf{x}[n]$ reside along the main diagonal of $\mathbf{R}[\tau]$, while cross-correlation terms between the different entries of $\mathbf{x}[n]$ form the off-diagonal terms, such that $\mathbf{R}[\tau] = \mathbf{R}^{\mathrm{H}}[-\tau]$.

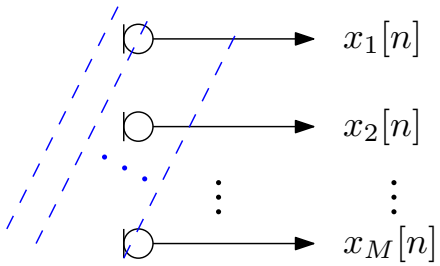

Figure 2.1: Visualisation of an M element broadband linear array collecting a vector-valued time series $\mathbf{x}[n] \in \mathbb{C}^M$.

The z-transform of $\mathbf{R}[\tau] \in \mathbb{C}^{M\times M}$, known as a cross-spectral density (CSD) matrix and denoted $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$, has Laurent polynomial elements in $z \in \mathbb{C}$ and therefore takes the form of a Laurent polynomial matrix[1] [4, 79]:

$$\boldsymbol{R}(z) = \sum_{\tau=-T}^{T} \mathbf{R}[\tau] z^{-\tau} \,. \tag{2.3}$$

Here, T is the maximum lag of $\mathbf{R}[\tau]$; i.e., $\mathbf{R}[\tau] = \mathbf{0} \;\forall\; |\tau| > T$. Throughout this thesis, the relationship between time domain and transform domain quantities is abbreviated as $\boldsymbol{R}(z) \bullet\!\!-\!\!\circ\, \mathbf{R}[\tau]$. Since $\mathbf{R}[\tau] = \mathbf{R}^{\mathrm{H}}[-\tau]$, $\boldsymbol{R}(z)$ is a parahermitian matrix, such that $\boldsymbol{R}(z) = \tilde{\boldsymbol{R}}(z)$ [4].

The space-time covariance matrix and corresponding CSD matrix can no longer be decorrelated using the EVD, which only measures and removes instantaneous spatial correlation; i.e., it eliminates correlation between pairs of signals sampled at the same instant in time. Instead, it is necessary to impose decorrelation over a suitably chosen range of relative time delays. This is referred to as strong decorrelation or total decorrelation [8], and a matrix of suitably chosen filters is required to achieve it.

A relatively naive approach to decorrelating the broadband sensor signals is to use the independent frequency bin (IFB) method, which splits the broadband spectrum into a number of narrow frequency bands via the discrete Fourier transform (DFT). The narrowband data in each band is then processed using the EVD. This scheme is also used to achieve spatial multiplexing in wireless communications [80]. However, drawbacks

[1] In this thesis, usage of the term 'polynomial' generally refers to a Laurent polynomial.

with this method are that the relatively small but important correlations between frequency bands are ignored, thus limiting the degree to which strong decorrelation can be achieved. It can also lead to a lack of phase coherence across the bands, since in each band the eigenvectors can experience an arbitrary phase shift and permutation (alongside the eigenvalues) relative to neighboring bands. These features of the IFB technique for space-time adaptive processing have previously been observed in phased array radar applications [81].

2.3 Polynomial Matrix Eigenvalue Decomposition

2.3.1 Definition

Research in [6] generalises the EVD to the case of broadband sensor arrays and convolutive mixing by requiring the strong decorrelation to be implemented using an FIR paraunitary polynomial matrix [7]. A paraunitary polynomial matrix represents a multichannel all-pass filter and, accordingly, it preserves the total signal power at every frequency [4]. In order to achieve strong decorrelation, the paraunitary matrix seeks to diagonalise a parahermitian polynomial matrix by means of a generalised similarity transformation in what has become known as a polynomial matrix eigenvalue decomposition (PEVD).

The PEVD [6] uses a paraunitary matrix $\boldsymbol{F}(z)$ or $\boldsymbol{Q}(z)$ to approximately diagonalise a parahermitian CSD matrix $\boldsymbol{R}(z)$ such that

$$\boldsymbol{R}(z) \approx \tilde{\boldsymbol{F}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z) = \boldsymbol{Q}(z)\boldsymbol{D}(z)\tilde{\boldsymbol{Q}}(z) \,, \tag{2.4}$$

where $\boldsymbol{D}(z) \approx \text{diag}\{d_1(z),\ d_2(z),\ \ldots,\ d_M(z)\}$ approximates a diagonal matrix and is typically spectrally majorised [8] with power spectral densities (PSDs) $d_m(\mathrm{e}^{\mathrm{j}\Omega}) \geq d_{m+1}(\mathrm{e}^{\mathrm{j}\Omega})\ \forall\ \Omega$, $m = 1 \ldots (M-1)$, where $d_m(\mathrm{e}^{\mathrm{j}\Omega}) = d_m(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega}}$. The diagonal of $\boldsymbol{D}(z)$ contains approximate polynomial eigenvalues, and the rows and columns of $\boldsymbol{F}(z)$ and $\boldsymbol{Q}(z)$, respectively, are approximate polynomial eigenvectors such that $\boldsymbol{F}(z) = \tilde{\boldsymbol{Q}}(z)$. While the definition of the PEVD in [6] generates polynomial eigenvectors of the form of $\boldsymbol{F}(z)$, the PEVD formulations of [9, 10, 53, 54] utilise the form of $\boldsymbol{Q}(z)$. Both forms

are used in this thesis; thus, different variables are used for clarity. The paraunitary property of the eigenvectors ensures that

$$\boldsymbol{F}(z)\tilde{\boldsymbol{F}}(z) = \tilde{\boldsymbol{F}}(z)\boldsymbol{F}(z) = \boldsymbol{Q}(z)\tilde{\boldsymbol{Q}}(z) = \tilde{\boldsymbol{Q}}(z)\boldsymbol{Q}(z) = \mathbf{I}_M \ , \tag{2.5}$$

where $\mathbf{I}_M$ is an $M \times M$ identity matrix. Note that the decomposition in (2.4) is unique up to permutations and arbitrary all-pass filters applied to the eigenvectors.

Equation (2.4) has only approximate equality, as the PEVD of a finite order polynomial matrix is generally transcendental; i.e. it is not of finite order. However, the approximation error can be shown to become arbitrarily small if the order of the approximation is selected sufficiently large [9]. A finite order approximation will therefore lead to only approximate diagonality of $\boldsymbol{D}(z)$ in (2.4). Similarly, a finite order approximation of $\boldsymbol{F}(z)$ or $\boldsymbol{Q}(z)$ through trimming will result in only approximate equality in (2.5). A paraunitary matrix can be implemented as a lossless FIR filter bank that conserves energy; as a result, the terms paraunitary matrix and paraunitary filter are used interchangeably throughout this thesis.

Recent research in [9, 10] provides conditions for the existence and uniqueness of eigenvalues and eigenvectors of a PEVD, such that these can be represented by a power or Laurent series that is absolutely convergent, permitting a direct realisation in the time domain. Further research in [11] studies the impact of estimation errors in the sample space-time covariance matrix on its PEVD.

2.3.2 Existing PEVD Algorithms

Iterative Time Domain Algorithms

For the calculation of the PEVD, only very limited ideal cases permit an exact decomposition. In general, PEVD algorithms have to rely on iterative approaches. The algorithms in this section operate in the time domain; i.e., they directly manipulate polynomial coefficients in an effort to diagonalise a parahermitian matrix. As such, through the course of algorithm iterations, for iteration parameter i, the diagonalisation metric $E_{\text{diag}}^{(i)}$ defined in Section A.3 and [45] is reduced.

An iterative gradient-based method to diagonalise a parahermitian matrix by means of paraunitary factorisation is presented in [82], but is limited to 2×2 parahermitian matrices with a specific structure found in subband coding. A fixed order approximate EVD (AEVD) algorithm for parahermitian matrices, operating in the time domain, was proposed in [52]. It applies a succession of first-order elementary paraunitary filter stages but does not always compute a good approximation and does not have proven convergence.

Second-order sequential best rotation (SBR2) [6], which has been proven to converge to a good approximate PEVD [6, 25, 83], is the most well-established PEVD algorithm. Following its implementation on an FPGA in [57–59], the algorithm has been further developed to employ a multiple shift strategy (MS-SBR2) in [46]. Additional research in [55] has confirmed that SBR2 converges to a spectrally majorised solution. In every iteration, SBR2 eliminates the 'dominant' off-diagonal element with maximum magnitude of a parahermitian matrix by means of a paraunitary operation. The paraunitary operation is not order-constrained, as in the AEVD [52], and applies a delay such that the dominant off-diagonal element is transferred onto lag zero of the space-time covariance matrix, $\mathbf{R}[0]$. A Jacobi rotation [2] then eliminates that element and transfers its energy onto the main diagonal.

In performing a delay operation, SBR2-based algorithms move an entire row and column of the CSD matrix into the lag zero matrix, where the Jacobi rotation will only eliminate the maximum element. The elimination of only a single element at each iteration typically leads to slow diagonalisation performance over algorithm iterations in practice, which the MS-SBR2 algorithm does not improve upon [66]. A second family of algorithms based on the sequential matrix diagonalisation (SMD) algorithm [45] advance this idea by transferring a greater quantity of energy to the diagonal of the lag zero matrix at each iteration through the use of a direct EVD. Application of the eigenvectors generated through an EVD of the lag zero matrix — following the shifting of energy to lag zero — guarantees that all shifted energy is transferred to the diagonal. Within a reasonable number of iterations, SMD is able to achieve levels of diagonalisation that cannot be obtained by SBR2 [45]. Particular focus is given to SMD in this

thesis for the reasons above, and those described later in Section 3.2.1, where SMD is found to be the most computationally efficient iterative PEVD algorithm. Further detail on the operation of SMD is therefore provided in Section A.1, as knowledge of this algorithm may aid the reader's understanding of subsequent chapters.

Although SMD has been shown to be able to achieve parahermitian matrix diagonalisation using lower order paraunitary matrices — which are beneficial for application purposes [17–19, 21–23, 25] — in [45], a developed row-shift truncation strategy in [62] is able to reduce the order of the paraunitary matrices from SBR2 but not SMD, such that SBR2 is preferable for a low order decomposition [63].

While the original SMD algorithm transfers the energy of an entire row and column pair [45], versions of SMD have been created that transfer the energy of multiple dominant elements to the diagonal of the lag zero matrix at each iteration. The first of these, which comes closest to matching the performance of an idealised maximum energy SMD algorithm in [84], is denoted multiple shift maximum element SMD (MSME-SMD) [47], with causality-constrained (C-MSME-SMD) [48] and reduced search space (RS-MSME-SMD) [50] versions found in the respective papers. Of these multiple shift approaches, the RS-MSME-SMD algorithm was identified as offering the fastest diagonalisation speed of all SMD-based algorithms, while producing shorter polynomial matrices than other multiple shift strategies [50].

Following the development of a 'cyclic-by-row' (CbR) approximation to the EVD step in SMD in [49], which is able to outperform SMD with respect to diagonalisation speed without sacrificing performance elsewhere, low-cost CbR versions have been created for all SMD-based algorithms. Using the CbR PEVD algorithms, the computational cost to reach a specific level of diagonalisation is reduced below even what is required for SBR2 [49]. Although multiple shift SMD algorithms were found to outperform SMD in terms of diagonalisation speed in [47, 48, 50], the equivalent CbR algorithms were found to be slower than a CbR implementation of SMD due to the high cost of their search step [65, 66]. Therefore, when the research in this thesis was initiated, the cyclic-by-row SMD (SMDCbR) algorithm was considered to be the state-of-the-art iterative PEVD algorithm.

DFT-Based Frequency Domain Algorithms

A DFT-based PEVD formulation is performed in [85]; however, the order of the paraunitary filter banks must be strictly limited. A second example, which transforms the problem into a pointwise-in-frequency standard matrix decomposition and has been shown to perform well for finite order problems [56], is provided in [53], but requires an a priori estimate of the length of the paraunitary matrix in the decomposition. This algorithm is summarised in Section A.2 for future reference. Such DFT-based PEVD methods have received renewed interest in recent years [54], due to their ability to return either a spectrally majorised decomposition, or attempt to approximate maximally smooth, analytic eigenvalues, which are necessary for good eigenvector estimation [9, 10].

Given their particular relevance in Chapter 5 of this thesis, additional information regarding the implementation of DFT-based algorithms is provided in Section 5.1.

2.3.3 Implementations of PEVD Algorithms

The PEVD has been successfully used in broadband multiple-input multiple-output (MIMO) precoding and equalisation using linear [12–14] and non-linear [15, 16] approaches, broadband angle of arrival estimation [17–20], broadband beamforming [21–23], optimal subband coding [8, 25], joint source-channel coding [27], and scene discovery [30].

A number of successful efforts to reduce the algorithmic cost of iterative PEVD algorithms have focussed on the trimming of polynomial matrices to curb growth in order [6, 60–62], which translates directly into a growth of computational complexity. However, while offering promising results, the PEVD has seen limited deployment in hardware. A parallel form of SBR2 whose performance has little dependency on the size of the input parahermitian matrix has been designed and implemented on an FPGA [57–59], but the SMD algorithm and its cyclic-by-row approximation, which can achieve superior levels of diagonalisation [45, 49], have been restricted to software applications — perhaps due to their dramatic increase in computational complexity with parahermitian matrix spatial dimension [65,66] and non-parallelisable architecture.

For the interested reader, versions of the original SMD and SBR2 algorithms can be found in the PEVD toolbox [86].

2.4 Simulation Software and Hardware Platform

Unless otherwise stated, all simulations in this thesis are performed within MATLAB® R2014a under Ubuntu® 16.04 on an MSI® GE60-2OE with Intel® Core™ i7-4700MQ 2.40 GHz × 8 cores, NVIDIA® GeForce® GTX 765M, and 8 GB RAM.

Chapter 3

Computational Advances for the Iterative PEVD

3.1 Introduction

As evidenced by the contents of Section 2.3.2, a number of iterative PEVD algorithms have been developed. Despite this, the PEVD has seen limited deployment for application purposes due to the high computational cost of these algorithms. This is especially true for scenarios involving a large number of broadband sensors, which require the decomposition of parahermitian matrices with a large spatial dimension, M. For example, as M is increased, the plot of Figure 3.1 from [65] demonstrates the increasingly poor diagonalisation versus execution time performance of the recent multiple shift maximum element SMD (MSME-SMD) algorithm. Here, the diagonalisation metric $E_{\text{diag}}^{(i)}$ defined in Section A.3 and [45] is used. For the case of a parahermitian matrix with spatial dimensions of 20×20, which in practice might be generated using data from 20 sensors, MSME-SMD requires hundreds of seconds to achieve even a moderate level of diagonalisation. Clearly, for application purposes, the required execution time of PEVD algorithms must be reduced.

This chapter therefore details a number of novel methods to lower the computational cost of existing iterative algorithms related to the PEVD. These methods conquer unexplored areas of PEVD algorithm improvement by optimising matrix manipulations

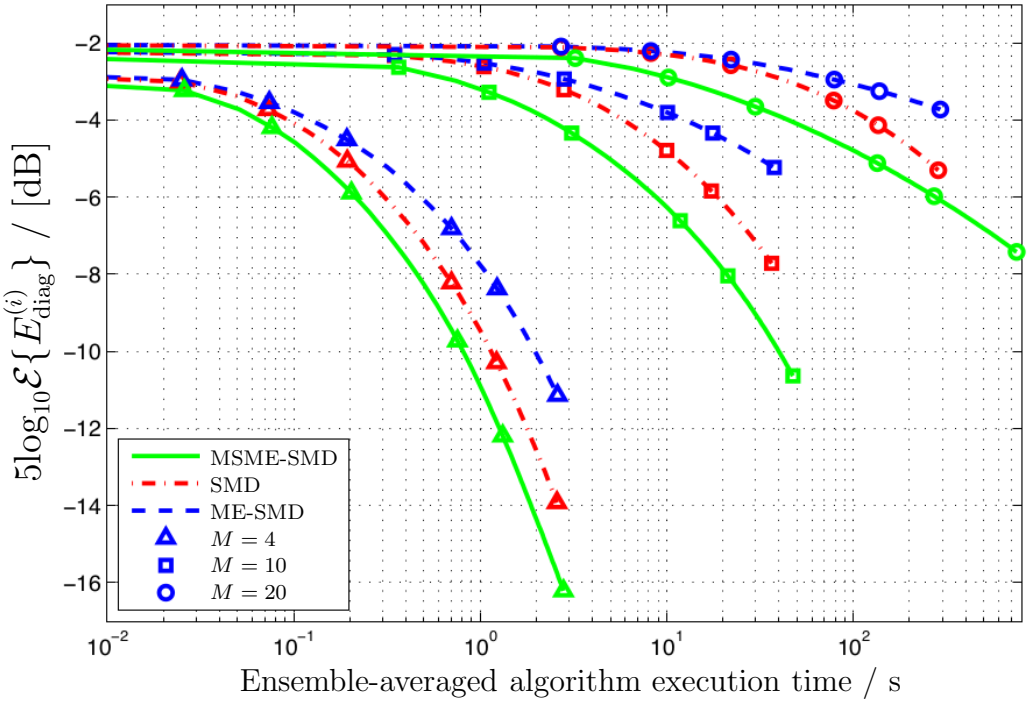

Figure 3.1: MSME-SMD, SMD [45], and maximum element SMD (ME-SMD) [45] diagonalisation speed for increasing spatial dimension M [65].

in software in Section 3.2, exploiting parahermitian matrix symmetry in Section 3.3, minimising algorithmic redundancy in Section 3.4, and reducing the search and update space of iterative PEVD algorithms in Sections 3.5 and 3.6. Repeated application of the polynomial matrix QR decomposition (PQRD) to a parahermitian matrix in a manner analogous to the QR algorithm [87] can be used to form a PEVD. A similar application of the PQRD has been explored for the polynomial matrix singular value decomposition (PSVD) in [43]. Improvements are therefore made to an existing PQRD algorithm in Section 3.7, such that both PEVD and PSVD implementations that rely on the PQRD can also benefit. Finally, improvements to an existing polynomial matrix truncation strategy yield benefits in Section 3.8, such that polynomial matrix length can be reduced without significantly impacting on decomposition mean square error.

3.2 Optimising Matrix Manipulations to Increase Algorithmic Efficiency

For application purposes, PEVD algorithms inevitably have to be implemented in software and subsequently deployed to hardware. A number of algorithmic improvements have been made to increase the performance of several iterative PEVD algorithm architectures [46–50, 88]; however, specifics regarding the algorithms' efficient implementation in software have not been addressed. This section therefore discusses a number of novel, significant ways to increase the algorithmic efficiency of iterative PEVD algorithms and polynomial matrix implementations without impacting accuracy. Methods to compute the product of a matrix and polynomial matrix are discussed in Section 3.2.1. Furthermore, efficient techniques to compute the product of two polynomial matrices are detailed in Section 3.2.2. The optimal point at which to integrate polynomial matrix truncation into iterative PEVD algorithms is explored in Section 3.2.3. Conclusions for this section are drawn in Section 3.2.4.

Elements of the work on polynomial matrix truncation in this section can be found published in the proceedings of the 24th European Signal Processing Conference in a paper titled 'Memory and Complexity Reduction in Parahermitian Matrix Manipulations of PEVD Algorithms' [67].

3.2.1 Product of a Matrix and Polynomial Matrix

In a number of iterative PEVD algorithms [45, 47, 48, 50], the product of a matrix and polynomial matrix is computed at least once per iteration. An example of this product can be seen in (A.5); here, it is required to transfer off-diagonal energy in the lag zero matrix of a parahermitian matrix onto the diagonal. Given that a significant number of iterations are typically required for convergence of a PEVD algorithm, this matrix multiplication step quickly forms a significant portion of the computational cost of such algorithms [45].

Matrix Multiplication Methods

Existing implementations of the algorithms in [45, 47, 48, 50] in MATLAB® typically make use of a polynomial matrix convolution function denoted PolyMatConv(·) from [86], whereby $\boldsymbol{S}(z) = \text{PolyMatConv}(\boldsymbol{Q}(z), \boldsymbol{R}(z))$ computes the product $\boldsymbol{S}(z) = \boldsymbol{Q}(z)\boldsymbol{R}(z)$, where $\boldsymbol{S}(z), \boldsymbol{R}(z), \boldsymbol{Q}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$. This function is implemented such that the polynomial element in the mth row and nth column of $\boldsymbol{S}(z)$ is found via

$$s_{m,n}(z) \bullet\!\!-\!\!\circ\, s_{m,n}[\tau] = q_{m,1}[\tau] * r_{1,n}[\tau] + q_{m,2}[\tau] * r_{2,n}[\tau] + \ldots + q_{m,M}[\tau] * r_{M,n}[\tau] \ . \quad (3.1)$$

If $\boldsymbol{Q}(z)$ is instead replaced with a simple matrix of coefficients $\mathbf{Q}$ as in (A.5), the implementation in (3.1) $\forall\ m, n$ — which involves a large number (M^3) of convolution operations — is not efficient. An alternative implementation for a parahermitian $\boldsymbol{R}(z)$ can instead compute

$$\mathbf{S}[\tau] = \mathbf{Q}\mathbf{R}[\tau] \ \ , \ \ \forall\, \tau \ , \quad (3.2)$$

which requires L $M \times M$ matrix multiplications, where $L = 2T + 1$ is the length of $\mathbf{R}[\tau]$ and T is its maximum lag. Given that MATLAB® is optimised for matrix multiplication [89], (3.2) is more efficient than (3.1) in this case. In essence, a choice is made to interpret $\boldsymbol{R}(z)$ as a single polynomial with matrix valued coefficients in (3.2), rather than a matrix of polynomials as in (3.1).

The formulation in (3.2) is highly parallelisable, as matrix multiplications for each τ are independent. This work therefore proposes the following novel approach, which enables the simultaneous computation of all lags of $\mathbf{S}[\tau]$. In this approach, by placing coefficient matrices of $\mathbf{R}[\tau]$ side-by-side in a horizontally concatenated matrix $\mathbf{R}_h \in \mathbb{C}^{M\times ML}$,

$$\mathbf{R}_h = [\,\mathbf{R}[-T]\ \mathbf{R}[-T+1]\ \cdots\ \mathbf{R}[T]\,] \ , \quad (3.3)$$

one can compute

$$\mathbf{S}_h = [\,\mathbf{S}[-T]\ \mathbf{S}[-T+1]\ \cdots\ \mathbf{S}[T]\,] = \mathbf{Q}\mathbf{R}_h \ , \quad (3.4)$$

which can be rearranged with little effort to obtain $\mathbf{S}[\tau]$.

The product in (A.5) also requires application of $\mathbf{Q}^{\text{H}}$ from the right — i.e., a formu-

lation where $\boldsymbol{S}'(z) = \boldsymbol{S}(z)\mathbf{Q}^{\mathrm{H}}$. Equations (3.1) and (3.2) can be very easily modified to accommodate this; however, the approach of (3.4) must be considered further, as the dimensionalities of $\mathbf{S}_h \in \mathbb{C}^{M\times ML}$ and $\mathbf{Q}^{\mathrm{H}} \in \mathbb{C}^{M\times M}$ are incompatible. Instead, one can compute $\mathbf{S}'_v \in \mathbb{C}^{ML\times M}$ using a vertically concatenated matrix $\mathbf{S}_v \in \mathbb{C}^{ML\times M}$:

$$\mathbf{S}'_v = \begin{bmatrix} \mathbf{S}'[-T] \\ \mathbf{S}'[-T+1] \\ \vdots \\ \mathbf{S}'[T] \end{bmatrix} = \mathbf{S}_v\mathbf{Q}^{\mathrm{H}} = \begin{bmatrix} \mathbf{S}[-T] \\ \mathbf{S}[-T+1] \\ \vdots \\ \mathbf{S}[T] \end{bmatrix} \mathbf{Q}^{\mathrm{H}}\,, \tag{3.5}$$

which can again be rearranged to obtain $\mathbf{S}'[\tau]$. A combination of (3.4) and (3.5) therefore enables computation of the product $\mathbf{Q}\boldsymbol{R}(z)\mathbf{Q}^{\mathrm{H}}$ using a mixture of horizontally and vertically concatenated matrices.

Matrix Multiplication Performance Comparison

To compare the computational costs of each of the three matrix multiplication methods, the simulations below record the average time taken for 10^3 instances of each method to compute $\boldsymbol{S}'(z) = \mathbf{Q}\boldsymbol{R}(z)\mathbf{Q}^{\mathrm{H}}$ in MATLAB® for $M \in \{5; 10; 20\}$ and $T \in \{50; 100\}$, with parahermitian matrix $\boldsymbol{R}(z) = \boldsymbol{A}(z)\tilde{\boldsymbol{A}}(z)$ of length $L = 2T + 1$. Matrices $\boldsymbol{A}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ and $\mathbf{Q} \in \mathbb{C}^{M\times M}$ contain coefficients drawn from a randomised complex Gaussian source with unit variance and zero mean.

In Table 3.1, it can be seen that the combined approach utilising (3.4) and (3.5) is faster than other methods for $T = 50$, while the PolyMatConv$(\cdot)$ function that is widespread in existing PEVD algorithms is by far the slowest. The latter experiences a significant slowdown as M increases, while the remaining methods decrease in speed more gradually.

Similar results can be found in Table 3.2 for $T = 100$, where the increase in temporal dimension has universally increased execution time requirements. Note that the approach using PolyMatConv$(\cdot)$ is less affected than the other methods, but is still significantly slower.

From these results, it can be concluded that the proposed two-dimensional concate-

Table 3.1: Average time taken for each matrix multiplication method with $T = 50$.

Method	Average execution time / s		
	$M = 5$	$M = 10$	$M = 20$
(3.1)	2.655×10^{-2}	2.086×10^{-1}	1.811
(3.2)	5.432×10^{-4}	8.582×10^{-4}	1.862×10^{-3}
(3.4) & (3.5)	1.178×10^{-4}	4.816×10^{-4}	9.986×10^{-4}

Table 3.2: Average time taken for each matrix multiplication method with $T = 100$.

Method	Average execution time / s		
	$M = 5$	$M = 10$	$M = 20$
(3.1)	2.786×10^{-2}	2.207×10^{-1}	1.890
(3.2)	1.038×10^{-3}	1.687×10^{-3}	3.598×10^{-3}
(3.4) & (3.5)	4.466×10^{-4}	6.249×10^{-4}	1.632×10^{-3}

nated matrix multiplication approach is most suitable for implementation purposes. Each algorithm in [45, 47, 48, 50] can be easily modified to adopt this strategy without affecting PEVD accuracy. For example, convergence speed results for various forms of SMD adapted to incorporate each matrix multiplication strategy are shown in Figure 3.2. In this plot, SMD represents the standard SMD algorithm [45] from [86] — which uses the PolyMatConv$(\cdot)$ function — implemented in MATLAB®; SMD† and SMD‡ have been adapted to use the approach of (3.2) and the combined approach of (3.4) and (3.5), respectively; and SMD* is a further modified form of SMD‡ that maintains only concatenated two-dimensional matrices throughout, and recovers the true three-dimensional representation upon completion. Note that all versions of SMD produce identical PEVDs. Simulations were performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M \times M}$ obtained from the randomised source model in Appendix B [45] with $M = 5$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$. During 200 iterations of each version of SMD — which uses the polynomial matrix truncation scheme of Appendix C — truncation parameters of $\mu = \mu_t = 10^{-12}$ and a stopping threshold of $\epsilon = 0$ were used. At each iteration of the algorithms, the elapsed simulation time and the diagonalisation metric $E_{\text{diag}}^{(i)}$ from Section A.3 — which is an indicator of PEVD algorithm convergence [45] — were recorded.

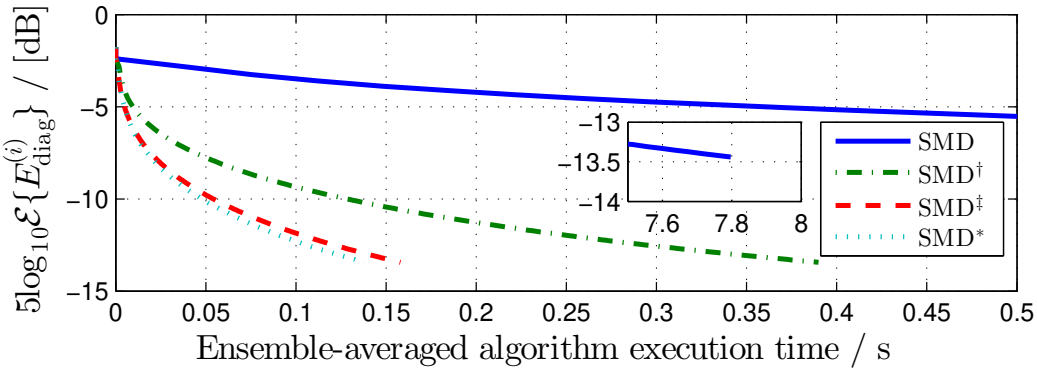

Figure 3.2: Matrix multiplication speed comparison using SMD algorithm.

In Figure 3.2, it can be observed that SMD‡, and the closely related SMD*, are quickest to converge, while the version of the algorithm available from [86] is significantly slower. Given sufficient development time, the purely two-dimensional implementation of SMD* can be transported to lower-level — potentially faster — software languages, such as C, with relative ease.

Extension of Matrix Multiplication Strategy to Existing Algorithms

It has been noted in the literature that while PEVD algorithms in the SMD family [45, 47–50] offer greater diagonalisation of a parahermitian matrix per iteration than alternative methods — such as those in the SBR2 family [6, 46] — their execution time requirements are significantly higher, and can be of the order of hundreds of seconds as in Figure 3.1. Fortunately, if these algorithms are modified to use the same matrix multiplication strategy as SMD‡, their speed can be dramatically increased. To convey this, all existing state-of-the-art PEVD algorithms are compared in a single simulation, which was performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M \in 5; 10$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$. The scenario inputs $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ to instances of the MSME-SMD‡ [47], restricted search MSME-SMD (RS-MSME-SMD‡) [50], SMD‡, cyclic-by-row SMD (SMDCbR) [49], SBR2 [6], and multiple shift SBR2 (MS-SBR2) [46] algorithms, which are each allowed a total of 200 iterations. The notation ‡ indicates that an algorithm has been modified to use the combined matrix multiplication

approach of (3.4) and (3.5). Each algorithm utilised the truncation strategy of Appendix C with polynomial matrix truncation parameters of $\mu = \mu_t = 10^{-6}$. At each iteration of the algorithms, the elapsed simulation time and diagonalisation metric $E_{\text{diag}}^{(i)}$ from Section A.3 were recorded.

Figure 3.3 shows the diagonalisation speed of each algorithm. It can be seen that within 200 iterations, the multiple shift SMD strategies achieve the highest level of diagonalisation. As has been noted in [66], when optimised, the SBR2 algorithm outperforms its multiple shift variant in terms of speed, but offers the lowest diagonalisation per iteration of any PEVD algorithm. However, its simplicity has led to SBR2 being used for implementation purposes [57–59]. The cyclic-by-row implementation of SMD is suited for implementations where direct EVD computation is not possible; it achieves a similar level of diagonalisation to SMD‡, but is significantly slower. Indeed, SMD‡ achieves a respectable level of diagonalisation within the 200 iterations, and offers the fastest performance overall.

If the results of Figures 3.1 and 3.3 are compared, it is clear that by optimising the matrix multiplication strategy used by the most computationally costly component of the SMD-based algorithms [45], significant performance gains have been made. Importantly, these gains have arisen without impacting on the numerical accuracy of the PEVDs produced. Since the SMD‡ algorithm offers the fastest — i.e., lowest complexity — PEVD, it is considered to be the state-of-the-art PEVD algorithm; it is therefore the predominant benchmark algorithm used throughout this thesis for comparison purposes. Of course, there are other aspects to the PEVD that can not be measured by diagonalisation speed alone. However, for brevity, the reader is referred to the literature for information regarding the efficacy of each algorithm with respect to other PEVD performance metrics.

Note that, when comparing algorithm modifications in subsequent sections, the matrix multiplication strategy in all SMD-based algorithms matches the procedure in SMD‡. If desired, any of these algorithms could be further modified to utilise a purely two-dimensional implementation.

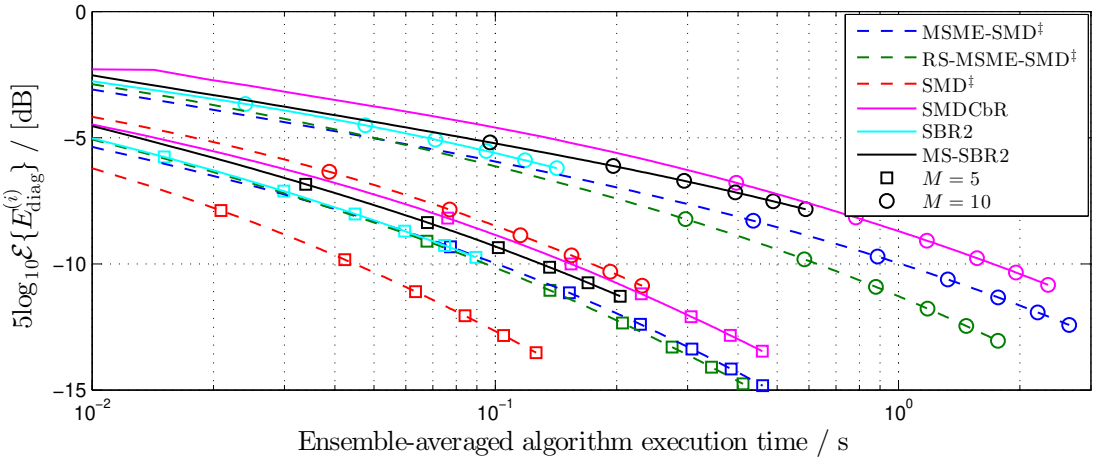

Figure 3.3: Diagonalisation speed comparison of state-of-the-art iterative PEVD algorithms when using optimised matrix multiplication strategies.

3.2.2 Product of Two Polynomial Matrices

The function PolyMatConv(·) [86] described in Section 3.2.1 employs convolution in the time domain to find the product of two polynomial matrices. For polynomial matrices of spatial dimension $M \times M$, this requires the convolution of M time series per polynomial element, and therefore M^3 convolutions for the computation of all elements. Linear convolution of a series of length L_1 and a series of length L_2 in the discrete time domain is equivalent to multiplication of their Fourier coefficients in the discrete frequency domain, provided that the number of frequency bins used is equal to $K = L_1 + L_2 - 1$. In the field of digital signal processing, it is well established that the linear convolution of two time series can be computed significantly faster in the frequency domain if the fast Fourier transform (FFT) is utilised [90]. In this section, this principle is extended to the computation of the product of two polynomial matrices, resulting in a decrease in the required computation time.

Moving to the Frequency Domain

The product $\boldsymbol{S}(z) = \boldsymbol{Q}(z)\boldsymbol{R}(z)$ computed by $\boldsymbol{S}(z) = \text{PolyMatConv}(\boldsymbol{Q}(z), \boldsymbol{R}(z))$, where $\boldsymbol{S}(z) : \mathbb{C} \to \mathbb{C}^{M\times N}$, $\boldsymbol{Q}(z) : \mathbb{C} \to \mathbb{C}^{M\times P}$, and $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{P\times N}$, is found via

$$s_{m,n}(z) \bullet\!\!-\!\!\circ\, s_{m,n}[\tau] = \sum_{p=1}^{P} q_{m,p}[\tau] * r_{p,n}[\tau] \; . \tag{3.6}$$

Here, $\boldsymbol{S}(z)$ has length $L_S = L_Q + L_R - 1$, where L_Q and L_R are the lengths of $\boldsymbol{Q}(z)$ and $\boldsymbol{R}(z)$, respectively.

The same product can be computed in the frequency domain if $\boldsymbol{Q}(z)$ and $\boldsymbol{R}(z)$ are evaluated at $K = L_S$ points on the unit circle ($z = \mathrm{e}^{\mathrm{j}\Omega_k}$, $\Omega_k = 2\pi k/K$) using the FFT. Here, $\mathbf{S}[k] = \boldsymbol{S}(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega_k}}$ is obtained via matrix multiplication:

$$\mathbf{S}[k] = \mathbf{Q}[k]\mathbf{R}[k] \; , \; k = 0 \ldots K-1 \; . \tag{3.7}$$

The inverse FFT (IFFT) then recovers $\boldsymbol{S}(z)$.

In MATLAB® — which is often the software of choice for the implementation of DSP algorithms — FFT, IFFT, and matrix multiplication operations are highly optimised [89,91–93]. An alternative implementation of PolyMatConv(·) that computes the product of two polynomial matrices in the frequency domain as described above is therefore well-suited to this platform. Such an implementation is introduced in this work, and is named PolyMatConvFreq(·). A particular third-party MATLAB® library named *MTIMESX* [94] is optimised for computing the product of multi-dimensional matrices, and can therefore facilitate fast computation of $\mathbf{S}[k]$.

Results and Discussion

Below, the computational cost of PolyMatConv(·) is compared with the costs of two frequency-based implementations. The first of these, encapsulated in a function denoted $\text{PolyMatConvFreq}_1(\cdot)$ employs the matrix multiplication routines native to MATLAB®, while the second, denoted $\text{PolyMatConvFreq}_2(\cdot)$ uses the *MTIMESX* library.

To compare the computational costs of each of the three methods, in the simulations

Table 3.3: Average time taken for methods to compute the product of polynomial matrices with $T = 50$.

Method	Average execution time / s		
	$M = 5$	$M = 10$	$M = 20$
PolyMatConv$(\cdot)$	1.389×10^{-2}	1.070×10^{-1}	8.705×10^{-1}
PolyMatConvFreq$_1(\cdot)$	9.438×10^{-4}	2.125×10^{-3}	3.365×10^{-3}
PolyMatConvFreq$_2(\cdot)$	7.064×10^{-4}	1.672×10^{-3}	2.598×10^{-3}

Table 3.4: Average time taken for methods to compute the product of polynomial matrices with $T = 100$.

Method	Average execution time / s		
	$M = 5$	$M = 10$	$M = 20$
PolyMatConv$(\cdot)$	1.504×10^{-2}	1.183×10^{-1}	9.800×10^{-1}
PolyMatConvFreq$_1(\cdot)$	1.587×10^{-3}	2.344×10^{-3}	6.352×10^{-3}
PolyMatConvFreq$_2(\cdot)$	8.272×10^{-4}	1.462×10^{-3}	4.779×10^{-3}

below, the average time taken for 10^3 instances of each method to compute $\boldsymbol{S}(z) = \boldsymbol{Q}(z)\boldsymbol{R}(z)$ in MATLAB® is calculated for $M \in \{5; 10; 20\}$, $M = P$, and $T \in \{50; 100\}$. Matrices $\boldsymbol{Q}(z) : \mathbb{C} \to \mathbb{C}^{M \times P}$ and $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{P \times M}$ are of order T and contain coefficients drawn from a randomised complex Gaussian source with unit variance and zero mean.

In Table 3.3, it can be seen that the frequency-based approaches are faster than the PolyMatConv$(\cdot)$ function, which is widespread in existing PEVD implementations, for $T = 50$. The latter experiences a significant slowdown as M increases, while the remaining methods decrease in speed more gradually.

Similar results can be found in Table 3.4 for $T = 100$, where the increase in temporal dimension has almost universally increased execution time requirements. Surprisingly, the PolyMatConvFreq$_2(\cdot)$ method actually becomes slightly faster for $M = 10$ with this larger T; while the reason why this occurs is not known, a reasonable guess could assume that the *MTIMESX* library is particularly well-optimised for this specific dimensionality of three-dimensional matrices.

From these results, it can be concluded that the proposed PolyMatConvFreq$_2(\cdot)$ frequency-based approach to computing the product of two polynomial matrices is

most suitable for implementation purposes. Any polynomial matrix implementation in MATLAB® can be easily modified to adopt this strategy without affecting accuracy.

3.2.3 Implementation of Truncation Within PEVD Algorithms

Truncation of outer lags of the iteratively updated polynomial matrices internal to PEVD algorithms is typically required to constrain computational complexity and memory costs [6, 45, 60]. For example, in the SMD algorithm described in Section A.1, this usually involves truncation of the parahermitian and paraunitary matrices at the end of each algorithm iteration. If the product of a unitary matrix $\mathbf{Q}$ and a polynomial matrix $\boldsymbol{R}(z)$ produces a polynomial matrix $\boldsymbol{S}(z) = \mathbf{Q}\boldsymbol{R}(z)$, the total energy in lag τ of $\mathbf{S}[\tau] \circ\!\!-\!\!\bullet\, \boldsymbol{S}(z)$, $\|\mathbf{S}[\tau]\|_{\mathrm{F}}^2$, is the same as the total energy in $\mathbf{R}[\tau]$; i.e., $\|\mathbf{S}[\tau]\|_{\mathrm{F}}^2 = \|\mathbf{R}[\tau]\|_{\mathrm{F}}^2$. As a result, for the example of SMD, truncation can actually be performed on $\boldsymbol{S}^{(i)\prime}(z)$ and $\boldsymbol{H}^{(i)\prime}(z)$ prior to the computationally expensive rotation to create $\boldsymbol{S}^{(i)}(z)$ and $\boldsymbol{H}^{(i)}(z)$ in (A.5). This section therefore proposes trimming of the parahermitian and paraunitary matrices prior to such rotation stages in a given iteration of any PEVD algorithm, such that no rotation operations are executed on terms that will subsequently be discarded. It is shown, for the case of the SMD algorithm, that this novel approach to truncation can reduce algorithm execution time without affecting PEVD performance.

Results and Discussion

Efforts to test the benefit of the novel truncation strategy described above led to the execution of the simulations below, which are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M = \{5; 10\}$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\mathrm{DR}} = 30$. Two versions of the SMD algorithm are tested: the first performs truncation after rotation, and the second — denoted SMD§ — performs truncation before rotation. For 200 iterations of the algorithms, polynomial matrix truncation parameters of $\mu = \mu_t = 10^{-6}$ and a stopping threshold of $\epsilon = 0$ are used, and the diagonalisation metric $E_{\mathrm{diag}}^{(i)}$ from Section A.3 and parahermitian matrix length are recorded.

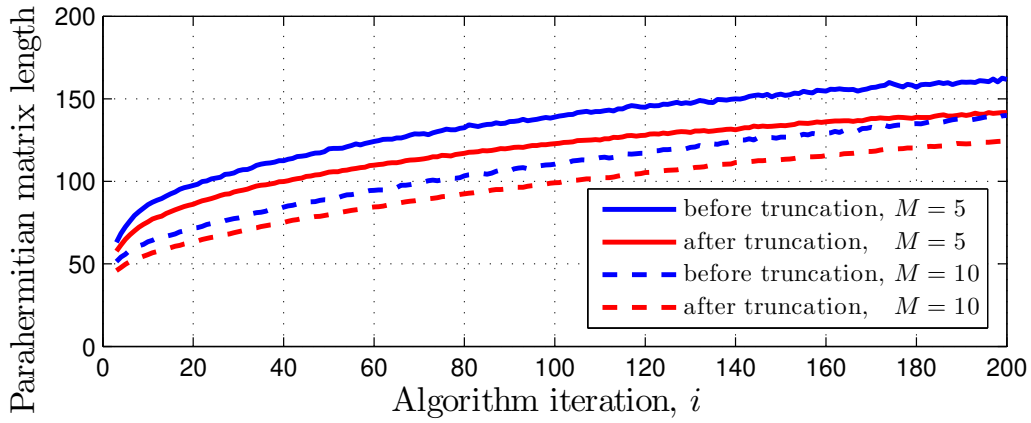

Figure 3.4: Parahermitian matrix length before and after truncation versus algorithm iteration for SMD for $M \in \{5;\ 10\}$.

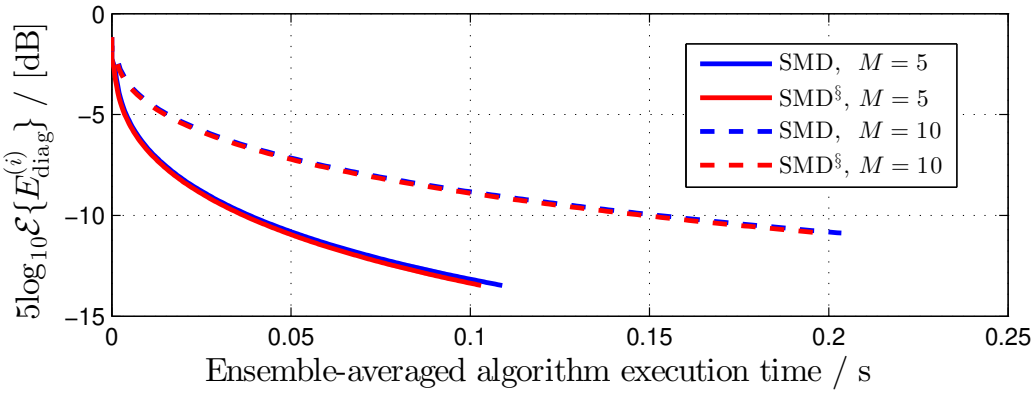

Figure 3.5: Diagonalisation metric versus algorithm execution time for SMD and SMD§ for $M \in \{5;\ 10\}$.

The impact of parahermitian matrix truncation according to Appendix C on the length of the parahermitian matrix internal to SMD can be seen in Figure 3.4. Note, the first two SMD algorithm iterations have been omitted for clarity. It is clear that the truncation process trims a significant number of lags of the parahermitian matrix at each iteration; thus, a significant number of matrix multiplication operations can been avoided if the truncation step is moved to before the rotation stage of (A.5).

The plot of Figure 3.5 confirms the assumption made above by demonstrating that the modified SMD§ converges slightly faster than the original SMD algorithm for both values of M. While this performance increase is not large, the modifications required to obtain it are minimal, and can be extended to any iterative PEVD algorithm.

Note that, when comparing algorithm modifications directly in subsequent sections, the implementation of truncation in any modified algorithms matches the truncation procedure in the original algorithm to avoid bias in any algorithm performance results.

3.2.4 Summary

This section has discussed a number of novel, significant ways to increase the algorithmic efficiency of iterative PEVD algorithms and polynomial matrix implementations. An efficient, two-dimensional approach to computing the product of a matrix and polynomial matrix has been introduced; when integrated with existing SMD-based PEVD algorithms, this has been shown to dramatically increase their speed. In fact, the speed of the improved SMD algorithm has been increased to the point where SMD can now be considered to be the most efficient, state-of-the-art iterative PEVD algorithm. Furthermore, it has been shown that the execution time of polynomial matrix implementations can be reduced by evaluating the product of two polynomial matrices in the frequency domain. Finally, the optimal point at which to integrate polynomial matrix truncation into iterative PEVD algorithms has been explored.

Importantly, the algorithmic efficiency improvements discussed in this section can be extended to any number of PEVD-based implementations without loss of accuracy.

3.3 Parahermitian Matrix Symmetry and the Half-Matrix Approach

This section addresses savings — in terms of both computational complexity and memory use — that exploit the natural symmetry of the parahermitian structure of the matrix being decomposed in PEVD algorithms, such that only one half of its elements are stored and processed. A demonstration of how this approach can be implemented within existing PEVD algorithms is shown through the modification of the SMD algorithm [45] described in Section A.1. The modified form of SMD is named half-matrix SMD (HSMD).

Based on the half-matrix representation of a parahermitian matrix defined in

Section 3.3.1, Section 3.3.2 provides an overview of the HSMD algorithm, Section 3.3.3 outlines the parameter search of HSMD, and Section 3.3.4 details the implementation of column- and row-shifts in HSMD. The resource requirements and performances of SMD and HSMD are compared in Section 3.3.5 and Section 3.3.6, respectively. Subsequently, conclusions are provided in Section 3.3.7.

Elements of the work in this section can be found published in the proceedings of the 24th European Signal Processing Conference in a paper titled 'Memory and Complexity Reduction in Parahermitian Matrix Manipulations of PEVD Algorithms' [67].

3.3.1 Half-Matrix Representation of a Parahermitian Matrix

By partitioning a parahermitian matrix $\boldsymbol{R}(z)$, it is possible to write

$$\boldsymbol{R}(z) = \boldsymbol{R}^{(-)}(z) + \mathbf{R}[0] + \boldsymbol{R}^{(+)}(z) \,, \tag{3.8}$$

where $\mathbf{R}[0]$ is the lag zero matrix of $\boldsymbol{R}(z)$, $\boldsymbol{R}^{(+)}(z)$ contains terms for positive lag ($\tau > 0$) elements only, and $\boldsymbol{R}^{(-)}(z) = \tilde{\boldsymbol{R}}^{(+)}(z)$. It is therefore sufficient to record half of $\boldsymbol{R}(z)$, which here without loss of generality is $\mathbf{R}[0] + \boldsymbol{R}^{(+)}(z)$. For the remainder of this thesis, the notation $\overline{\{\cdot\}}$ is used to represent the recorded half of a parahermitian matrix; i.e., $\overline{\boldsymbol{R}}(z) = \mathbf{R}[0] + \boldsymbol{R}^{(+)}(z)$ and $\overline{\boldsymbol{R}}(z) \bullet\!\!-\!\!\circ\, \overline{\mathbf{R}}[\tau]$, where

$$\overline{\mathbf{R}}[\tau] = \begin{cases} \mathbf{R}[\tau], & 0 \leq \tau \leq T \\ \mathbf{0}, & \text{otherwise} \end{cases} \,. \tag{3.9}$$

If one possesses knowledge of $\overline{\boldsymbol{R}}(z)$, one has all the information required to obtain $\boldsymbol{R}(z)$, $\overline{\mathbf{R}}[\tau]$, and $\mathbf{R}[\tau]$. Given this relationship, $\overline{\boldsymbol{R}}(z)$ is therefore referred to as a parahermitian matrix throughout the remainder of this thesis for brevity. In addition, $\overline{\boldsymbol{R}}(z)$ and $\overline{\mathbf{R}}[\tau]$ are referred to as the 'half-matrix' versions of the 'full-matrix' representations $\boldsymbol{R}(z)$ and $\mathbf{R}[\tau]$, respectively. As an example, Figure 3.6 demonstrates both half- and full-matrix representations of a 5×5 matrix with maximum lag $T = 3$.

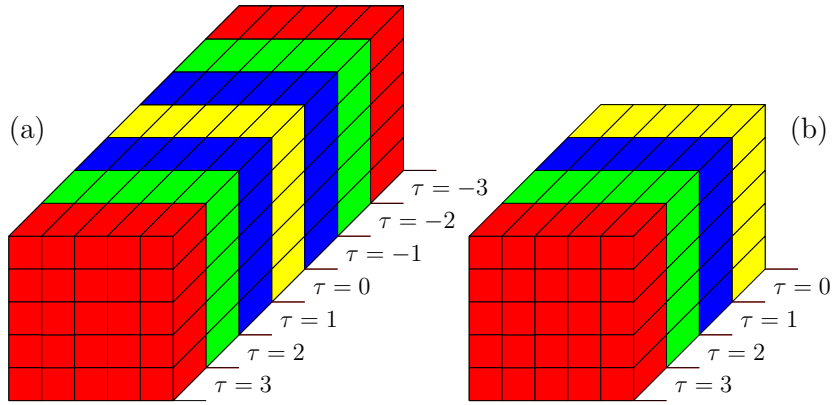

Figure 3.6: (a) Full-matrix and (b) half-matrix representation of a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{5\times5}$ for $T = 3$.

3.3.2 Half-Matrix Sequential Matrix Diagonalisation Algorithm

While SMD — which is detailed in Section A.1 — seeks to diagonalise $\boldsymbol{R}(z)$, HSMD diagonalises $\overline{\boldsymbol{R}}(z)$ by iteratively updating $\overline{\boldsymbol{S}}^{(i)}(z)$. Using a half-matrix representation naturally reduces computational costs and memory requirements; however, there are two main disadvantages to discarding $\boldsymbol{S}^{(i)(-)}(z)$. As a first disadvantage, the parameter search space of (A.6) is not guaranteed to identify the column with maximum off-diagonal energy, as columns for $\tau < 0$ are ignored. Secondly, columns and rows cannot be shifted in the direction of positive τ without the introduction of invalid polynomial coefficients. Sections 3.3.3 and 3.3.4 describe how these problems are tackled such that HSMD is able to mimic the results of SMD exactly.

The HSMD algorithm is functionally very similar to SMD; thus, only the differences between HSMD and SMD are described in the sections below, while pseudocode for HSMD is provided in Algorithm 1 for reference.

3.3.3 Modified Parameter Search in HSMD

To find the correct shift parameters for the HSMD algorithm, (A.6) can be used directly but with a restriction of the search space for column norms to $\tau \geq 0$, such that $\tau^{(i)} \geq 0$ is also imposed as a constraint. This requires only half the search space of the standard SMD approach, but neglects to search column norms for negative time lags, hence yielding a solution that is equivalent to the causally-constrained SMD algorithm [48].

Input: $\overline{\boldsymbol{R}}(z)$, $I_{\max}$, ϵ, μ, μ_t
Output: $\overline{\boldsymbol{D}}(z)$, $\boldsymbol{F}(z)$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise $\overline{\mathbf{R}}[0] \in \mathbb{C}^{M\times M}$
$\overline{\boldsymbol{S}}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\overline{\boldsymbol{R}}(z)\mathbf{Q}^{(0)\mathrm{H}}$; $\boldsymbol{H}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$; $i \leftarrow 0$; stop $\leftarrow 0$
do
  $i \leftarrow i+1$
  Find $\{k^{(i)}, \tau^{(i)}\}$ from (3.12); generate $\boldsymbol{\Lambda}^{(i)}(z)$ from (3.13)
  $\overline{\boldsymbol{S}}^{(i)\prime}(z) \leftarrow \mathrm{shift}_{\mathrm{HSMD}}(\,\overline{\boldsymbol{S}}^{(i-1)}(z),\, k^{(i)},\, \tau^{(i)}, \boldsymbol{\Lambda}^{(i)}(z), T^{(i-1)}, M\,)$
  $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
  Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\overline{\mathbf{S}}^{(i)\prime}[0]$
  $\overline{\boldsymbol{S}}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\overline{\boldsymbol{S}}^{(i)\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$
  $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime}(z)$
  if $i > I_{\max}$ or (3.14) satisfied **then**
    stop $\leftarrow 1$;
  end
  Truncate $\boldsymbol{H}^{(i)}(z)$ using threshold μ_t according to Appendix C
  Truncate $\overline{\boldsymbol{S}}^{(i)}(z)$ using threshold μ according to Appendix C
while stop $= 0$
$\boldsymbol{F}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$
$\overline{\boldsymbol{D}}(z) \leftarrow \overline{\boldsymbol{S}}^{(i)}(z)$

Algorithm 1: HSMD algorithm

If column norms for negative lags values $\tau < 0$ are to be included in the search, then due to the parahermitian structure of $\boldsymbol{S}^{(i-1)}(z)$, searching the unobtainable column norms of $\overline{\mathbf{S}}^{(i-1)}[\tau]$ for $\tau < 0$ is equivalent to searching row norms for $\tau \geq 0$. If a modified row norm for the kth row is defined as

$$\|\hat{\bar{\mathbf{s}}}^{(i-1)}_{(\mathrm{r}),k}[\tau]\|_2 = \sqrt{\textstyle\sum_{m=1,m\neq k}^{M} |\bar{s}^{(i-1)}_{k,m}[\tau]|^2}\,. \tag{3.10}$$

and a modified column norm for the kth column is defined as

$$\|\hat{\bar{\mathbf{s}}}^{(i-1)}_{k}[\tau]\|_2 = \sqrt{\textstyle\sum_{m=1,m\neq k}^{M} |\bar{s}^{(i-1)}_{m,k}[\tau]|^2}\,, \tag{3.11}$$

then the modified parameter search is

$$\{k^{(i)}, \tau^{(i)}\} = \arg\max_{k,\tau} \left\{ \|\hat{\bar{\mathbf{s}}}^{(i-1)}_{k}[\tau]\|_2\,,\ \|\hat{\bar{\mathbf{s}}}^{(i-1)}_{(\mathrm{r}),k}[-\tau]\|_2 \right\}\,. \tag{3.12}$$

Here, $\bar{s}_{m,k}^{(i-1)}[\tau]$ is the element in the mth row and kth column of $\bar{\mathbf{S}}^{(i-1)}[\tau]$. If (3.12) returns $\tau^{(i)} > 0$, then the $k^{(i)}$th column of $\overline{\boldsymbol{S}}^{(i-1)}(z)$ is to be shifted by $\tau^{(i)}$ lags towards lag zero. If $\tau^{(i)} < 0$, it is the $k^{(i)}$th row that requires shifting by $-\tau^{(i)}$ towards lag zero.

3.3.4 Modification of Shift Strategy in HSMD

The delay step in the SMD algorithm, which is implemented via equation (A.4) in Section A.1, can be performed with the reduced parahermitian matrix representation in the ith iteration by shifting either the $k^{(i)}$th column or row — whichever has the greater modified norm according to (3.11) or (3.10) — by $|\tau^{(i)}|$ coefficients to lag zero. To preserve the half-matrix representation, elements that are shifted beyond lag zero — i.e., outside the recorded half-matrix — have to be stored as their parahermitian conjugate and appended onto the $k^{(i)}$th row (for $\tau^{(i)} > 0$) or column (for $\tau^{(i)} < 0$) of the shifted matrix at lag zero. The concatenated row or column is then shifted by $|\tau^{(i)}|$ elements towards increasing τ. Note that the combination of row- and column-shifts effectively shifts the polynomial in the $k^{(i)}$th position along the diagonal of $\overline{\boldsymbol{S}}^{(i-1)}(z)$ in opposite directions, such that this polynomial remains unaffected. An efficient implementation of HSMD can therefore exclude this element from shifting operations.

An efficient example of the shift operation is depicted in Figure 3.7 for the case of $\overline{\boldsymbol{S}}^{(i-1)}(z) : \mathbb{C} \to \mathbb{C}^{5\times 5}$ with parameters $k^{(i)} = 2$ and $\tau^{(i)} = -3$. Owing to the negative sign of $\tau^{(i)}$, it is here the 2nd row that has to be shifted first, followed by the 2nd column shifted in the opposite direction. Another example of the shift operation is depicted in Figure 3.8 with parameters $k^{(i)} = 5$ and $\tau^{(i)} = 3$. Owing to the positive sign of $\tau^{(i)}$, it is here the 5th column that has to be shifted first, followed by the 5th row shifted in the opposite direction.

While the product in (A.2) is used to implement row- and column-shift operations in SMD, a function $\text{shift}_{\text{HSMD}}(\cdot)$ — which is described in Algorithm 2 — implements the delays encapsulated in the matrix $\boldsymbol{\Lambda}^{(i)}(z)$ for a half-matrix representation. Here, paraunitary delay matrix

$$\boldsymbol{\Lambda}^{(i)}(z) = \text{diag}\{\underbrace{1 \ \ldots \ 1}_{k^{(i)}-1} \ z^{-\tau^{(i)}} \ \underbrace{1 \ \ldots \ 1}_{M-k^{(i)}}\} \tag{3.13}$$

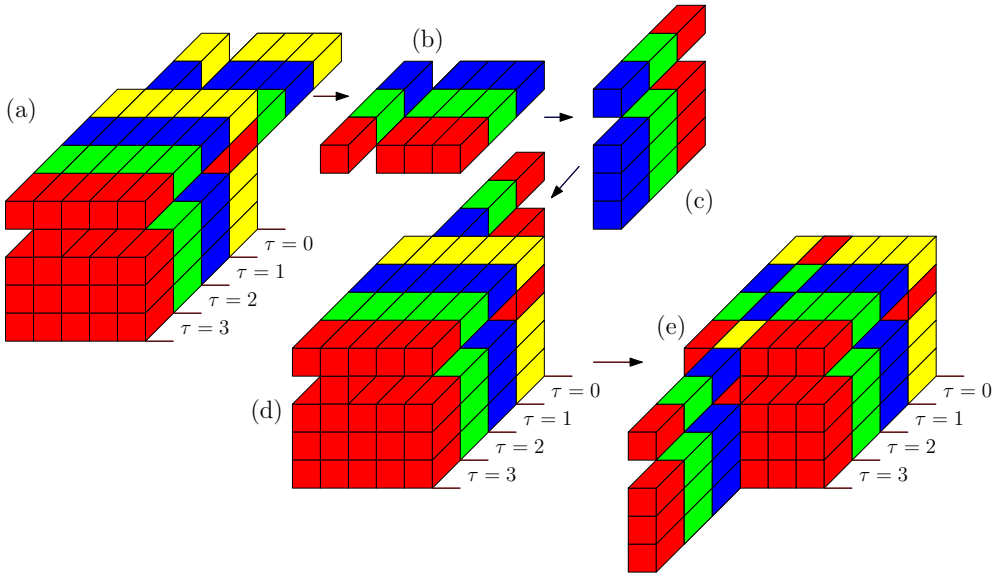

Figure 3.7: Example for a parahermitian matrix where in the ith iteration a row norm is maximum: (a) the row is shifted, with non-diagonal elements in the $k^{(i)}$th row past lag zero (b) extracted and (c) parahermitian conjugated. (d) These elements are appended to the $k^{(i)}$th column at lag zero and (e) shifted in the opposite direction with all off-diagonal column elements.

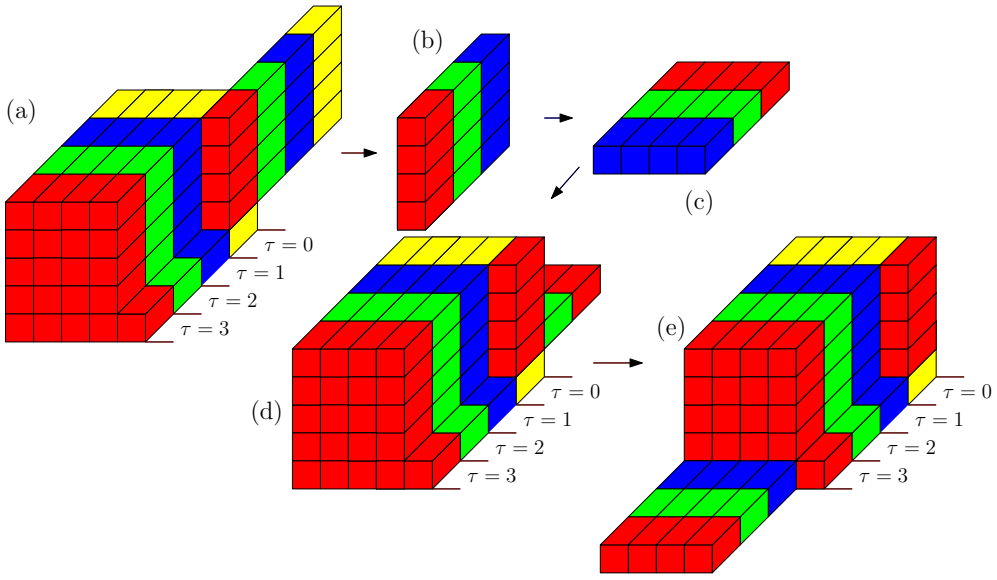

Figure 3.8: Example for a parahermitian matrix where in the ith iteration a column norm is maximum: (a) the column is shifted, with non-diagonal elements in the $k^{(i)}$th column past lag zero (b) extracted and (c) parahermitian conjugated. (d) These elements are appended to the $k^{(i)}$th row at lag zero and (e) shifted in the opposite direction with all off-diagonal row elements.

Input: $\overline{\boldsymbol{S}}(z)$, k_s, τ_s, $\boldsymbol{\Lambda}(z)$, T, M
Output: $\overline{\boldsymbol{S}}'(z)$
$\boldsymbol{\Gamma}(z) \leftarrow \begin{bmatrix} \boldsymbol{\gamma}_{1,1}(z) & \cdots & \boldsymbol{\gamma}_{1,M}(z) \\ \vdots & \ddots & \vdots \\ \boldsymbol{\gamma}_{M,1}(z) & \cdots & \boldsymbol{\gamma}_{M,M}(z) \end{bmatrix}$
if $\tau_s > 0$ **then**
  $\boldsymbol{L}(z) \leftarrow \overline{\boldsymbol{S}}(z)\tilde{\boldsymbol{\Lambda}}(z)$
  $\boldsymbol{\gamma}_{m,k}(z) \leftarrow \begin{cases} \sum_{\tau=-\tau_s+1}^{0} \mathbf{L}_{m,k}[\tau] z^{-\tau}, & k = k_s,\ m \neq k_s \\ 0, & \text{otherwise} \end{cases}$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z) + z^{\tau_s}\tilde{\boldsymbol{\Gamma}}(z)$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{\Lambda}(z)\boldsymbol{L}(z)$
else if $\tau_s < 0$ **then**
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{\Lambda}(z)\overline{\boldsymbol{S}}(z)$
  $\boldsymbol{\gamma}_{m,k}(z) \leftarrow \begin{cases} \sum_{\tau=\tau_s+1}^{0} \mathbf{L}_{m,k}[\tau] z^{-\tau}, & m = k_s,\ k \neq k_s \\ 0, & \text{otherwise} \end{cases}$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z) + z^{-\tau_s}\tilde{\boldsymbol{\Gamma}}(z)$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z)\tilde{\boldsymbol{\Lambda}}(z)$
else
  $\boldsymbol{L}(z) \leftarrow \overline{\boldsymbol{S}}(z)$
end
$\overline{\boldsymbol{S}}'(z) \leftarrow \sum_{\tau=0}^{T+|\tau_s|} \mathbf{L}[\tau] z^{-\tau}$

Algorithm 2: $\text{shift}_{\text{HSMD}}(\cdot)$ function

is selected based on the parameter set from (3.12), and $T^{(i-1)}$ is the maximum lag of $\overline{\mathbf{S}}^{(i-1)}[\tau]$.

Iterations of HSMD continue for a maximum of $I_{\max}$ steps, or until $\overline{\boldsymbol{S}}^{(I)}(z)$ is sufficiently diagonalised — for some I — with dominant off-diagonal column or row norm

$$\max_{k,\tau} \left\{ \|\hat{\tilde{\mathbf{s}}}_k^{(I)}[\tau]\|_2 \ , \ \|\hat{\tilde{\mathbf{s}}}_{(\mathrm{r}),k}^{(I)}[-\tau]\|_2 \right\} \leq \epsilon \quad , \tag{3.14}$$

where the value of ϵ is chosen to be arbitrarily small.

3.3.5 Complexity and Memory Reduction

SMD Algorithm Complexity and Memory Requirements

If at the ith iteration of SMD $\mathbf{S}^{(i-1)}[\tau] = \mathbf{0}\ \forall\ |\tau| > T^{(i-1)}$, the memory to store $\boldsymbol{S}^{(i-1)}(z) : \mathbb{C} \to \mathbb{C}^{M \times M}$ must hold $(2T^{(i-1)}+1)M^2$ coefficients. The maximum column

Table 3.5: Appproximate resource requirements of SMD and HSMD if $T^{(i)} \gg 1$.

Method	Complexity	Storage	Memory Moves
SMD	$4T^{(i)}M^3$	$2T^{(i)}M^2$	$4T^{(i-1)}(M-1)$
HSMD	$2T^{(i)}M^3$	$T^{(i)}M^2$	$2T^{(i-1)}(M-1)$

search requires the calculation of $(M-1)M(2T^{(i-1)}+1)$ multiply-accumulates (MACs) for the modified column norms according to (A.7).

During the ith iteration, the polynomial order growth leads to $T^{(i)} = T^{(i-1)} + |\tau^{(i)}|$, and the calculation of (A.4) is implemented as a combination of two block memory moves: one for the rows of $\mathbf{S}^{(i-1)}[\tau]$, and one for the columns. If diagonal elements are excluded, the number of coefficients of $\mathbf{S}^{(i-1)}[\tau]$ to be moved can therefore be approximated by $2(2T^{(i-1)}+1)(M-1) \approx 4T^{(i-1)}(M-1)$, assuming $T^{(i-1)}$ is large.

For (A.5), every matrix-valued coefficient in $\boldsymbol{S}^{(i)\prime}(z)$ must be left- and right-multiplied with a unitary matrix. Accounting for a multiplication of 2 $M \times M$ matrices by M^3 MACs [2, 95], a total of $(2(2T^{(i)}+1)M^3) \approx 4T^{(i)}M^3$ MACs arise to generate $\boldsymbol{S}^{(i)}(z)$ from $\boldsymbol{S}^{(i)\prime}(z)$. It is therefore the cost of this update step that dominates the computational cost of the ith iteration [45].

HSMD Algorithm Complexity and Memory Requirements

The memory required to store $\overline{\boldsymbol{S}}^{(i)}(z)$ at the ith iteration of HSMD is equal to $(T^{(i)}+1)M^2$ coefficients, and is therefore approximately half of what SMD needs. During the ith iteration, the first delay step of HSMD involves $2(T^{(i-1)}+1)(M-1)$ coefficients to be shifted in memory, as opposed to $2(2T^{(i-1)}+1)(M-1)$ for a full matrix representation. Therefore, the number of coefficient moves during the shift step is also halved using the proposed approach.

In terms of multiply-accumulates, the rotation operation with $\mathbf{Q}^{(i)}$ during the ith iteration requires $2M^3(T^{(i)}+1)$ MACs, saving approximately half of the operations executed during the standard approach outlined above. The various aspects of resource requirements for the maintenance of the parahermitian matrix are summarised in Table 3.5.

Of course, the paraunitary matrix must also be obtained during iterations, which

requires further resources; however, the requirements are the same for both SMD and HSMD, so this aspect is omitted for brevity.

3.3.6 Results and Discussion

Simulation Scenario

The simulations below are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M \in \{5;\ 10\}$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\mathrm{DR}} = 30$.

During PEVD algorithm iterations, a truncation parameter of $\mu = 10^{-15}$ and a stopping threshold of $\epsilon = 0$ are used. The SMD and HSMD implementations are run until the iteratively updated parahermitian matrix is diagonalised such that diagonalisation metric $5\log_{10} E_{\mathrm{diag}}^{(i)} = -15\,\mathrm{dB}$, and at every iteration step the metrics defined in Section A.3 are recorded together with the elapsed execution time.

Diagonalisation

The ensemble-averaged diagonalisation $E_{\mathrm{diag}}^{(i)}$ was calculated for both SMD and HSMD. While both algorithms are functionally identical and exhibit the same diagonalisation performance over algorithm iterations, the diagonalisation speed for both methods is shown in Figure 3.9. The curves demonstrate that for $M \in \{5;\ 10\}$, the lower complexity associated with the HSMD implementation translates to a faster diagonalisation than observed for the standard SMD realisation. Using a matrix with a larger spatial dimension of $M = 10$ versus $M = 5$ results in slower diagonalisation for both algorithms, but the same relative performance increase is still seen for the proposed reduced approach.

Simulated in MATLAB®, the results in Figure 3.9 are not as impressive as the computational savings suggested by Table 3.5. This is partly due to the requirement of both algorithms to maintain $\boldsymbol{H}^{(i)}(z)$. The MATLAB® profiler indicates that the execution time for matrix multiplications (i.e., the number of matrix multiplications) has been substantially reduced by the proposed method; however, while MATLAB® is optimised for the employed matrix multiplication strategy from Section 3.2.1, its

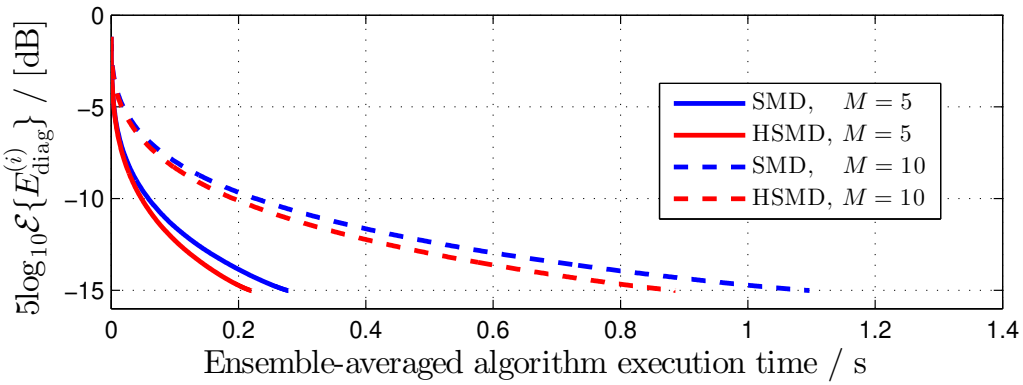

Figure 3.9: Diagonalisation metric versus algorithm execution time for SMD and HSMD for $M \in \{5;\ 10\}$.

shifting of memory is as not efficient and dominates the execution time. Despite this, Figure 3.9 indicates a 20% reduction in cost when using the proposed over the standard SMD implementation.

3.3.7 Summary

The symmetry in the parahermitian matrix has been exploited when calculating a polynomial matrix EVD, which has been exemplified here by a focus on the SMD algorithm. A reduced parahermitian matrix representation has been proposed that only records its causal part; this approach can produce the same accuracy of decomposition as a standard matrix representation in the SMD algorithm, but with increased efficiency with respect to memory use and computational complexity. Simulation results underline that the same diagonalisation performance can be achieved by both methods, but within a shorter execution time for the approach based on a reduced representation.

When designing PEVD implementations for real applications, the potential for the proposed techniques to reduce complexity and memory requirements therefore offers benefits without deficits with respect to important performance metrics such as the diagonalisation of the SMD algorithm. The reduced representation of parahermitian matrices proposed here can be extended to any PEVD algorithm in [6,46–50] by adapting the shift and rotation operations accordingly.

3.4 Increasing Efficiency within Cyclic-by-Row PEVD Implementations

This work addresses potential computational savings that can be applied to existing so-called 'cyclic-by-row' (CbR) approaches for the PEVD. An SMD cyclic-by-row (SMDCbR) approach [49] has been introduced as a low-cost variant of SMD, which approximates the EVD step inside the SMD algorithm by a cyclic-by-row implementation of the Jacobi algorithm [2]. In this work, a method to concatenate the Jacobi rotations in SMDCbR to reduce the computational cost of the algorithm is described, and thresholding to the rotation process to eliminate the rotation of near-zero elements is introduced. It is demonstrated that with the proposed techniques, computations can be reduced without significantly impacting on algorithm convergence. The proposed methods are advantageous for application purposes, where the SMDCbR algorithm's avoidance of a direct EVD computation may be useful.

Aspects of the iterative PEVD algorithm SMDCbR [49] are reviewed in Section 3.4.1, alongside an assessment of the main algorithmic cost. To reduce the complexity of the SMDCbR algorithm, Section 3.4.2 and Section 3.4.3 outline modifications to the rotation step of the algorithm to concatenate and threshold the Jacobi rotations, respectively. The performances of SMDCbR and its modified versions are compared in Section 3.4.4. Conclusions for this section are drawn in Section 3.4.5.

Elements of the work in this section can be found published in the proceedings of the 50th Asilomar Conference on Signals, Systems and Computers in a paper titled 'Complexity and Search Space Reduction in Cyclic-by-Row PEVD Algorithms' [68].

3.4.1 Cyclic-by-Row SMD Algorithm

Overview

The SMDCbR algorithm operates almost identically to SMD; thus, much of its description can be found in Section A.1. The only difference between SMDCbR and SMD arises during the rotation stage. In iteration i of SMD, this involves the computation of a unitary matrix $\mathbf{Q}^{(i)}$ from the EVD of lag zero, $\mathbf{S}^{(i)\prime}[0]$, and the subsequent computation

of $\mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$ and $\mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime}(z)$. At the ith iteration, SMDCbR instead calculates an approximation to the EVD of $\mathbf{S}^{(i)\prime}[0]$ using a sequence of $n = 1 \dots P$ Jacobi rotations $\mathbf{Q}^{(i,n)} \in \mathbb{C}^{M\times M}$. Each rotation takes the form of an $M \times M$ identity matrix, but with the four elements at the intersections of rows and columns $\{m^{(i,n)}, k^{(i,n)}\}$ defined such that

$$\mathbf{Q}^{(i,n)} = \begin{bmatrix} \mathbf{I}_{(1,n)} & & & & \\ & \cos\phi^{(i,n)} & \cdots & \mathrm{e}^{\mathrm{j}\theta^{(i,n)}}\sin\phi^{(i,n)} & \\ & \vdots & \mathbf{I}_{(2,n)} & \vdots & \\ & -\mathrm{e}^{-\mathrm{j}\theta^{(i,n)}}\sin\phi^{(i,n)} & \cdots & \cos\phi^{(i,n)} & \\ & & & & \mathbf{I}_{(3,n)} \end{bmatrix} . \tag{3.15}$$

The rotation angles $\phi^{(i,n)}$ and $\theta^{(i,n)}$ in (3.15) are determined by the target element at row $m^{(i,n)}$ and column $k^{(i,n)}$ in the slice [49]. The identity matrices $\mathbf{I}_{(j,n)}$, $j = 1, 2, 3$, have spatial dimensions of $(\min\{m^{(i,n)}, k^{(i,n)}\} - 1)$, $(|m^{(i,n)} - k^{(i,n)}| - 1)$, and $(M - \max\{m^{(i,n)}, k^{(i,n)}\} + 1)$, respectively. Each Jacobi rotation transfers the energy of two off-diagonal elements onto the diagonal via the product $\mathbf{Q}^{(i,n)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i,n)\mathrm{H}}$, which zeroes elements $\mathbf{S}^{(i)\prime}_{m^{(i,n)},k^{(i,n)}}[0]$ and $\mathbf{S}^{(i)\prime}_{k^{(i,n)},m^{(i,n)}}[0]$ and places their energy on the diagonal.

The process used in SMDCbR can be described as a cyclic-by-row implementation of the Jacobi algorithm [2]. The term CbR refers to the ordering of the rotations in a sequence like that of Figure 3.10, referred to as a Jacobi sweep. While in a standard EVD these might be repeated until off-diagonal elements are suppressed below a given threshold, SMDCbR employs only one Jacobi sweep at each iteration. This equates to a fixed number of $P = M(M-1)/2$ Jacobi rotations for a matrix of size $M \times M$ to provide the unitary $\mathbf{Q}^{(i)}$,

$$\mathbf{Q}^{(i)} = \mathbf{Q}^{(i,P)}\mathbf{Q}^{(i,P-1)}\cdots\mathbf{Q}^{(i,2)}\mathbf{Q}^{(i,1)} = \prod_{n=1}^{P}\mathbf{Q}^{(i,n)} . \tag{3.16}$$

Each rotation is applied in sequence to the parahermitian matrix, as computation of $\mathbf{Q}^{(i,2)}$ requires knowledge of lag zero of the product $\mathbf{Q}^{(i,1)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i,1)\mathrm{H}}$. That is,

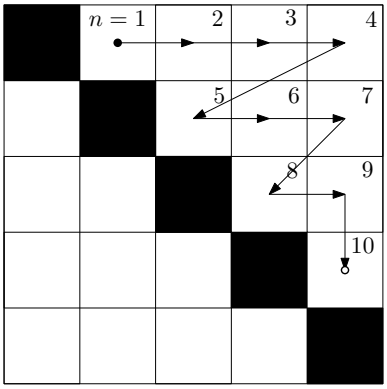

Figure 3.10: Values of n used for cyclic-by-row sequence of Jacobi rotations in one Jacobi sweep for a 5×5 matrix with start $\bullet$ and end point $\circ$.

$\mathbf{Q}^{(i,1)}$ is computed from $\mathbf{S}^{(i,1)\prime}[0] = \mathbf{S}^{(i)\prime}[0]$, $\mathbf{Q}^{(i,2)}$ is computed from $\mathbf{S}^{(i,2)\prime}[0]$ where $\boldsymbol{S}^{(i,2)\prime}(z) = \mathbf{Q}^{(i,1)}\boldsymbol{S}^{(i,1)\prime}(z)\mathbf{Q}^{(i,1)\mathrm{H}}$, and so on. Rotations are also applied in sequence to the paraunitary matrix, $\boldsymbol{H}^{(i)\prime}(z)$.

Algorithm Complexity

For each Jacobi rotation during iteration i of SMDCbR, of which there are $M(M-1)/2$, every matrix-valued coefficient in $\boldsymbol{S}^{(i)\prime}(z)$ must be left- and right-multiplied with a sparse unitary matrix. Accounting for the multiplication of a 4-sparse $M \times M$ matrix with a non-sparse $M \times M$ matrix by $4M$ MACs, a total of $4L\{\boldsymbol{S}^{(i)\prime}(z)\}M^2(M-1)$ MACs arise to generate $\boldsymbol{S}^{(i)}(z)$ from $\boldsymbol{S}^{(i)\prime}(z)$, where operator $L\{\cdot\}$ measures the length of a polynomial matrix. Every matrix-valued coefficient in $\boldsymbol{H}^{(i)\prime}(z)$ must also be left-multiplied with a sparse unitary matrix for each rotation. A total of $2L\{\boldsymbol{H}^{(i)\prime}(z)\}M^2(M-1)$ MACs arise to generate $\boldsymbol{H}^{(i)}(z)$ from $\boldsymbol{H}^{(i)\prime}(z)$.

The total number of MACs dedicated towards the rotation step of the algorithm at each iteration is therefore given by $4L\{\boldsymbol{S}^{(i)\prime}(z)\}M^2(M-1)+2L\{\boldsymbol{H}^{(i)\prime}(z)\}M^2(M-1) \approx 2M^3(2L\{\boldsymbol{S}^{(i)\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime}(z)\})$.

3.4.2 Concatenation of Rotations

In the approach detailed here, Jacobi rotations are performed on the lag zero matrix only, before a 'concatenated' unitary matrix $\mathbf{Q}^{(i)}$ is applied to every lag of the para-

hermitian and paraunitary matrices. This unitary matrix is equal to the product of all the sparse rotation matrices applied to lag zero, and is equivalent to (3.16). The direct application of sparse rotations to the polynomial matrices is therefore avoided, which results in a reduction in complexity.

Each rotation is still applied in sequence, but to lag zero of the parahermitian matrix only. That is, $\mathbf{Q}^{(i,1)}$ is computed from $\mathbf{S}^{(i,1)\prime}[0] = \mathbf{S}^{(i)\prime}[0]$, $\mathbf{Q}^{(i,2)}$ is computed from $\mathbf{S}^{(i,2)\prime}[0]$ where $\mathbf{S}^{(i,2)\prime}[0] = \mathbf{Q}^{(i,1)}\mathbf{S}^{(i,1)\prime}[0]\mathbf{Q}^{(i,1)\mathrm{H}}$, and so on. A record of the previous n rotations is maintained in a unitary matrix $\hat{\mathbf{Q}}^{(i,n)}$, such that $\hat{\mathbf{Q}}^{(i,1)} = \mathbf{Q}^{(i,1)}$, $\hat{\mathbf{Q}}^{(i,2)} = \mathbf{Q}^{(i,2)}\hat{\mathbf{Q}}^{(i,1)}$, $\hat{\mathbf{Q}}^{(i,3)} = \mathbf{Q}^{(i,3)}\hat{\mathbf{Q}}^{(i,2)}$, and so on. Matrix $\mathbf{Q}^{(i)} = \hat{\mathbf{Q}}^{(i,P)}$ is then applied to the parahermitian and paraunitary matrices in the same manner as in (A.5).

For each of the $M(M-1)/2$ sparse rotations, the lag zero and unitary matrices must be updated. Updating the lag zero matrix for each rotation involves left- and right-multiplication with a sparse unitary matrix, costing $8M$ MACs; thus, a full sweep requires $4M^2(M-1)$ MACs. Left-multiplying the unitary matrix for each rotation requires $4M$ MACs; thus, a full sweep encompasses $2M^2(M-1)$ MACs. Both steps combined therefore require $6M^2(M-1)$ MACs.

At each iteration, as in the SMD algorithm [45], every matrix-valued coefficient in $\boldsymbol{S}^{(i)\prime}(z)$ must be left- and right-multiplied with a non-sparse unitary matrix. Accounting for the multiplication of 2 $M \times M$ matrices by M^3 MACs [2, 95], a total of $2L\{\boldsymbol{S}^{(i)\prime}(z)\}M^3$ MACs arise to generate $\boldsymbol{S}^{(i)}(z)$ from $\boldsymbol{S}^{(i)\prime}(z)$. In addition, every matrix-valued coefficient in $\boldsymbol{H}^{(i)\prime}(z)$ must be left-multiplied with a non-sparse unitary matrix at each iteration; thus, $L\{\boldsymbol{H}^{(i)\prime}(z)\}M^3$ MACs are required to generate $\boldsymbol{H}^{(i)}(z)$ from $\boldsymbol{H}^{(i)\prime}(z)$.

The total number of MACs dedicated to the rotation stage per iteration is therefore $2L\{\boldsymbol{S}^{(i)\prime}(z)\}M^3 + L\{\boldsymbol{H}^{(i)\prime}(z)\}M^3 + 6M^2(M-1) \approx M^3(2L\{\boldsymbol{S}^{(i)\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime}(z)\})$ if $\min\{L\{\boldsymbol{S}^{(i)\prime}(z)\}, L\{\boldsymbol{H}^{(i)\prime}(z)\}\} \gg 6$. This is approximately equal to half of the MACs required for the standard SMDCbR algorithm.

3.4.3 Thresholding of Rotations

As the lag zero matrix is approximately diagonalised at each iteration of SMDCbR, the off-diagonal elements of $\mathbf{S}^{(i)\prime}[0]$ for $i \gg 0$ only contain significant values from the shifted dominant row and column found via the same parameter search as SMD in (A.6), with the remaining elements being approximately zero. As these elements possess very small values, there is little merit in applying a Jacobi rotation to transfer their energy onto the diagonal. By incorporating a threshold ν during execution of the cyclic-by-row Jacobi algorithm in SMDCbR, any element with an absolute value that falls below this threshold can be ignored.

It should be noted that ignoring Jacobi rotations in this way reduces the diagonalisation per iteration of the algorithm; however, if ν is kept sufficiently low, this approach can reduce computation time with only a minor impact on diagonalisation. Furthermore, the approach described in Section 3.4.2 does not benefit as significantly as the original SMDCbR algorithm would when employing a threshold for Jacobi rotations, as savings in the former would only be made during the lag zero and unitary matrix update step; that is, only $12M$ MACs would be avoided for each missed rotation. The latter would instead experience significant complexity reduction, as each skipped Jacobi rotation would equate to the avoidance of $8ML\{\boldsymbol{S}^{(i)\prime}(z)\} + 4ML\{\boldsymbol{H}^{(i)\prime}(z)\}$ MACs.

3.4.4 Results and Discussion

Simulation Scenario

The simulations below are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M \in \{3; 5; 7; 9; 11; 13; 15\}$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$.

The SMDCbR algorithm is below referred to as **standard**, and is compared to the following three proposed variations:

- **method 1**: SMDCbR with thresholding of Jacobi rotations;
- **method 2**: SMDCbR with concatenation of Jacobi rotations;
- **method 3**: SMDCbR with concatenation and thresholding of Jacobi rotations.

During iterations, polynomial matrix truncation parameters of $\mu = \mu_t = 10^{-6}$, a stopping threshold of $\epsilon = 10^{-6}$, and a Jacobi rotation threshold of $\nu \in \{0; 10^{-4}; 10^{-3}; 10^{-2}\}$ are used. The standard and proposed SMDCbR implementations are run over $I_{\max} = 200$ iterations, and at every iteration step the metrics defined in Section A.3 are recorded together with the elapsed execution time.

Diagonalisation

The ensemble-averaged diagonalisation, $E_{\text{diag}}^{(i)}$, was calculated for the standard and proposed implementations. The algorithms incorporating a threshold ν during diagonalisation (methods 1 and 3) are functionally different to those without this step, but very similar diagonalisation performance over algorithm iterations can be seen in Figure 3.11 for $M = 5$ and $\nu = 10^{-3}$. For the same parameters, the diagonalisation versus ensemble-averaged algorithm execution time for all methods is shown in Figure 3.12. The curves demonstrate that for $M = 5$, the lower complexity associated with the reduced implementations translates to a faster diagonalisation than observed for the standard realisation. An increase in performance is also evident for $M = 9$ and $\nu = 10^{-3}$ in Figure 3.13. In fact, the difference in diagonalisation performance between method 3 and method 1 has increased for this larger spatial dimension.

Thresholding of the Jacobi rotations in method 1 has a significant impact on its performance versus the standard algorithm; however, thresholding the same rotations in method 3 does not have such a high impact versus method 2. This is as a result of the thresholding in method 1 eliminating sparse rotations that would have been applied to all lags, while the thresholding in method 3 only eliminates sparse rotations from the lag zero and unitary matrix update step. Despite the small relative impact of thresholding in method 3, this algorithm performs marginally better than all other versions of the algorithm.

Figure 3.14 demonstrates that method 3 becomes more effective relative to method 1 as the spatial dimension M increases, for various values of threshold ν. It can be seen that the benefits of the proposed rotation concatenation approach exceed what could be expected from the calculated complexity reduction, owing to the well-suited nature

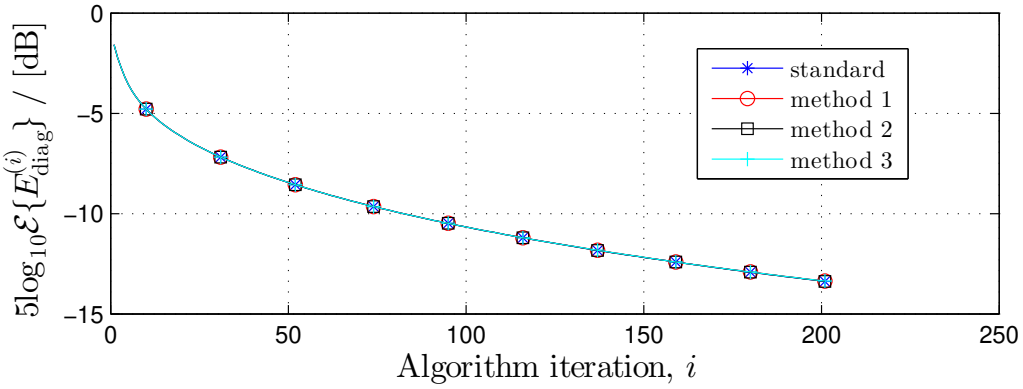

Figure 3.11: Diagonalisation metric versus algorithm iteration for the proposed and standard implementations for $M = 5$ and $\nu = 10^{-3}$.

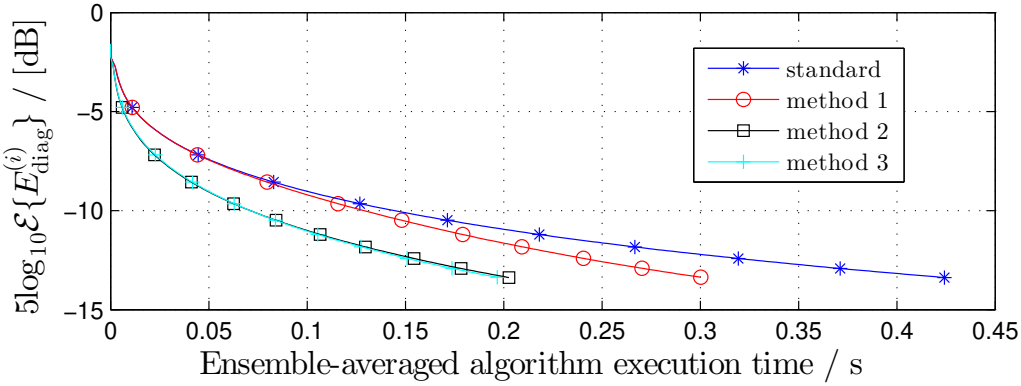

Figure 3.12: Diagonalisation metric versus algorithm execution time for the proposed and standard implementations for $M = 5$ and $\nu = 10^{-3}$.

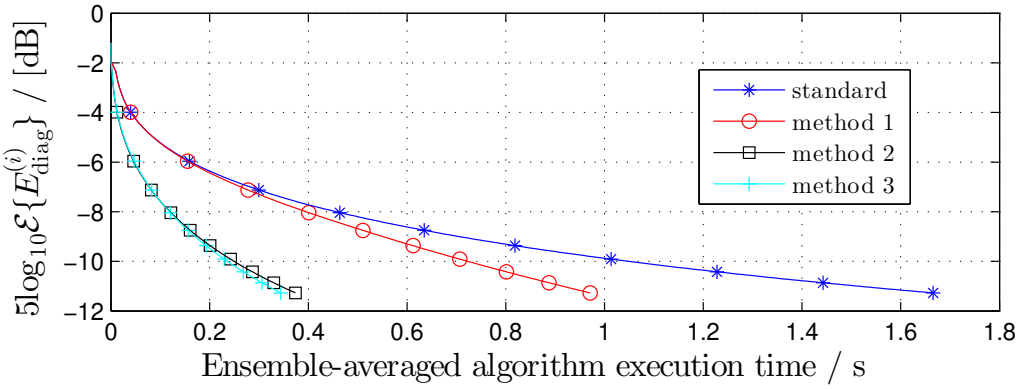

Figure 3.13: Diagonalisation metric versus algorithm execution time for the proposed and standard implementations for $M = 9$ and $\nu = 10^{-3}$.

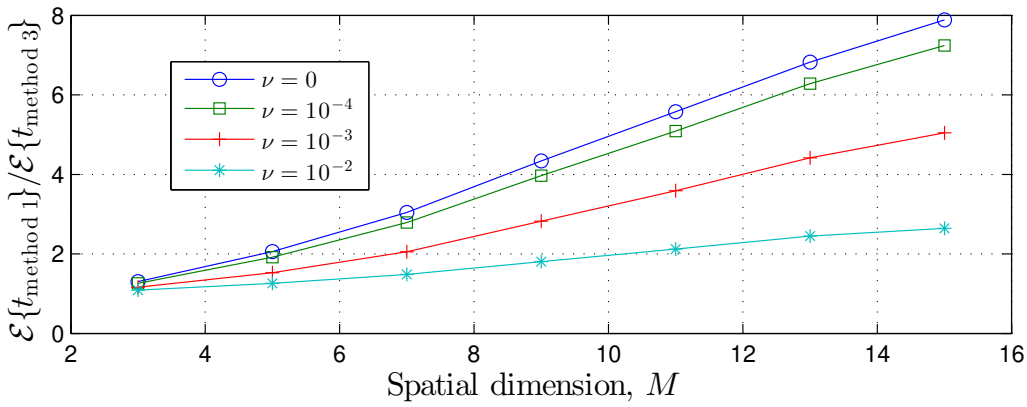

Figure 3.14: Ratio of method 1 to method 3 total execution time for different spatial dimension M.

of the utilised MATLAB® software for the matrix multiplication strategy introduced in Section 3.2.1. Increasing the value of ν decreases the diagonalisation per iteration of both methods, but also reduces the total execution time of method 1 such that it approaches the execution time of method 3. It should be noted that using a threshold of $\nu = 0$ in method 1 is equivalent to using the standard SMDCbR algorithm.

3.4.5 Summary

In this section, a series of steps to reduce the complexity of an existing cyclic-by-row SMD algorithm have been proposed. It has been shown through simulation that this reduction in complexity translates to an increase in the diagonalisation speed of the algorithm, with minimal impact on its convergence. The proposed methods are therefore advantageous for application purposes, where the SMDCbR algorithm's avoidance of a direct EVD computation may be useful. Of course, applying an approximate EVD at each iteration is less effective than a full EVD [49]; thus, if an efficient method for computing the EVD is present, the SMD algorithm will offer slightly superior convergence speed.

3.5 Restricting the Search Space of PEVD Algorithms

Research in [50, 88] has shown that restricting the search space of iterative PEVD algorithms to a subset of lags around lag zero of a parahermitian matrix can bring performance gains with little impact on algorithm convergence. In this section, a further cost reduction technique for all iterative PEVD algorithms is introduced, and is shown to improve the diagonalisation performance of the SMDCbR algorithm [49]. The proposed SMDCbR version is modified to limit the search of maximum off-diagonal energy in columns to a particular range of lags surrounding lag zero. It is shown that the size of this search segment can be determined by estimating the energy distribution in the parahermitian matrix prior to its decomposition.

The proposed technique to limit the search space of iterative PEVD algorithms is reviewed in Section 3.5.1. The performances of SMDCbR and a version of SMDCbR incorporating a restricted search space approach are then compared in Section 3.5.2. Subsequently, conclusions for this section are drawn in Section 3.5.3.

Elements of the work in this section can be found published in the proceedings of the 50th Asilomar Conference on Signals, Systems and Computers in a paper titled 'Complexity and Search Space Reduction in Cyclic-by-Row PEVD Algorithms' [68].

3.5.1 Limited Search Strategy

The search step of the SMDCbR algorithm — equivalent to that of SMD and described by (A.6) — evaluates modified column norms for all lags of the coefficient matrix $\mathbf{S}^{(i-1)}[\tau] \in \mathbb{C}^{M\times M}$. If at the ith iteration $\mathbf{S}^{(i-1)}[\tau] = \mathbf{0}\ \forall\ |\tau| > T^{(i-1)}$, for some maximum lag $T^{(i-1)} \in \mathbb{N}$, the search space encompasses $ML_S^{(i-1)}$ elements, where $L_S^{(i-1)} = (2T^{(i-1)} + 1)$ is the length of the parahermitian matrix.

The search step at each iteration of the modified SMDCbR algorithm uses the column norms of the off-diagonal elements for a reduced set of the lags of the coefficient matrix $\mathbf{S}^{(i-1)}[\tau]$. At the ith iteration, the search step is applied to a secondary matrix $\hat{\mathbf{S}}^{(i-1)}[\tau] = \mathbf{0}\ \forall\ |\tau| > \delta^{(i-1)}$, which is generated using the $L_{\hat{S}}^{(i-1)} = (2\delta^{(i-1)} + 1)$ centre lags of $\mathbf{S}^{(i-1)}[\tau]$; thus, the search space encompasses only $ML_{\hat{S}}^{(i-1)}$ elements. If this reduced search space can adequately contain most of the energy from the original

parahermitian matrix, then searching for a maximum within this search space can reduce computation time with little impact on algorithm performance.

A narrow distribution of energy around the lag zero can therefore lead to a large decrease in search space. As the algorithm progresses, the lags closest to lag zero become increasingly diagonalised, and therefore the average delay applied at each iteration increases; i.e., $|\tau^{(i)}|$ tends to increase for increasing i. This can be accounted for by gradually widening the search space around lag zero as the iteration number increases. To accommodate this widening of the search space, $\delta^{(i)}$ can be described as an increasing function of i.

The delay magnitude, $|\tau^{(i)}|$, at each iteration can be considered a measure of the concentration of energy in the parahermitian matrix. For example, if $|\tau^{(i)}|$ increases quickly for a small increase in i, this indicates that there is a low concentration of energy in the lags surrounding lag zero. In the following, X_i is the random variable that describes the $|\tau^{(i)}|$ seen in previous instances of a PEVD algorithm. In an application that requires a continuously updated estimate and PEVD of a parahermitian matrix $\boldsymbol{R}(z)$, the distribution of X_i can serve as a measure of the energy distribution in $\boldsymbol{R}(z)$ without the requirement of direct computation. Here, the search space $\delta^{(i)}$ in future instances of the PEVD algorithm can be controlled such that it encapsulates the majority of X_i in prior instances. Consider the simplistic example where X_i is distributed normally; in this case, choosing $\delta^{(i)} = \mathcal{E}\{X_i\} + 1.96\,\mathrm{std}\,\{X_i\}$ — where $\mathrm{std}\,\{\cdot\}$ computes the standard deviation — should allow for approximately 95% of $|\tau^{(i)}|$ in future instances of the PEVD algorithm to be unchanged relative to an instance of the algorithm without search space restriction. In practice, X_i is unlikely to be normally distributed; however, the above serves as a computationally cheap method for roughly approximating $\delta^{(i)}$.

3.5.2 Results and Discussion

Simulation Scenario

The simulations below are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with

$M = 5$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\mathrm{DR}} = 30$.

During iterations, polynomial matrix truncation parameters of $\mu = \mu_t = 10^{-6}$ and a stopping threshold of $\epsilon = 10^{-6}$ are used. A standard SMDCbR implementation and a version of SMDCbR utilising the proposed limited search strategy are each allowed to run for $I_{\max} = 200$ iterations. At every iteration step the metrics defined in Section A.3 are recorded together with the elapsed execution time, the absolute delay $|\tau^{(i)}|$ applied to a column in the parahermitian matrix, and the lengths of the parahermitian and paraunitary matrices, $L_S^{(i)}$ and $L_H^{(i)}$.

In this simulation scenario, the search space in the proposed implementation of SMDCbR has been reduced to approximate the 95% confidence interval of absolute delays applied in a previous ensemble of 10^3 instances of $\boldsymbol{R}(z)$, following the assumption that these values were normally distributed. Using this information, the search space reduction parameter evolution was identified as $\delta^{(i)} = 0.15i + 24$.

Diagonalisation

Figure 3.15 indicates the potential performance gain when limiting the search space to those lags deemed likely to contain high energy. As the algorithm's search step is a relatively inexpensive process compared with the shifting and rotation stages, the performance gain observed is small but significant. Time profiling in MATLAB® indicated that the use of the limited search strategy reduces the search time by 50.6% for the simulation used to generate Figure 3.15.

Impact on Order of Parahermitian Matrix

The impact of search space reduction on algorithm operation was investigated for the modified SMDCbR algorithm. Figures 3.16 and 3.17 show the ensemble-averaged absolute delay $|\tau^{(i)}|$ applied to a column in iteration i of SMDCbR and its modified version, and the length $L_S^{(i)}$ of the parahermitian matrix in both algorithms, respectively. From these figures, it is clear that the reduction in search space does not significantly impact the average delay applied — and thus the order of the parahermitian matrix — at each iteration of the proposed algorithm. In fact, since the proposed method typically

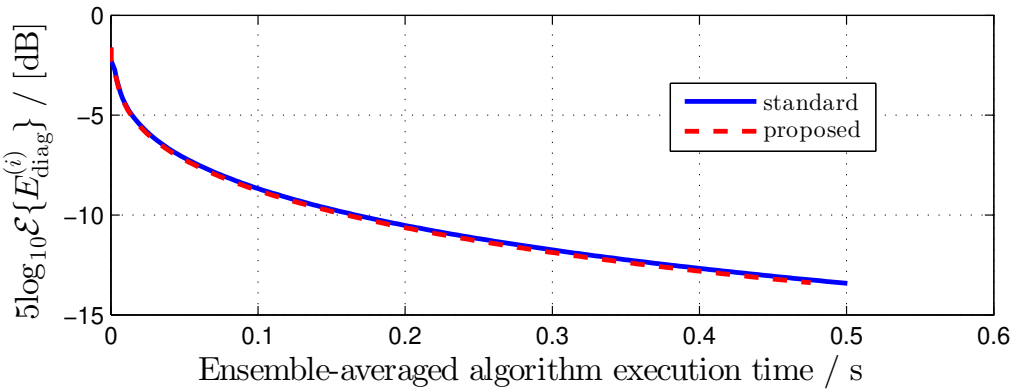

Figure 3.15: Diagonalisation metric versus algorithm execution time for the standard SMDCbR implementation and a version of SMDCbR utilising the proposed limited search strategy.

employs delays of smaller magnitude in this simulation scenario, it actually outputs a shorter parahermitian matrix. Interestingly, this does not translate to a shorter paraunitary matrix, as evidenced by Figure 3.18, which plots the ensemble-averaged paraunitary matrix length, $L_H^{(i)}$, over algorithm iterations.

3.5.3 Summary

The work in this section has demonstrated that a reduction in the search space can slightly improve the diagonalisation performance of an existing PEVD algorithm without significantly impacting convergence or the rate of growth of the parahermitian or paraunitary matrices. The technique proposed here can be extended to any PEVD algorithm in [6, 45–48]. However, given the relatively small performance gain achieved through the implementation of this method, it is likely that if the additional computations required to estimate the evolution of the search space reduction parameter $\delta^{(i)}$ are considered, the overall result is a small, or perhaps even negative, net reduction in computation time.

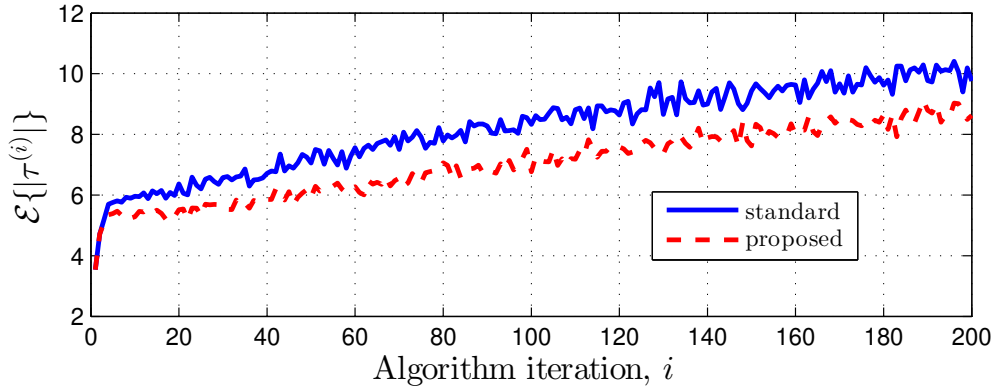

Figure 3.16: Absolute applied delay, $|\tau^{(i)}|$, versus iteration number for the proposed and standard implementations.

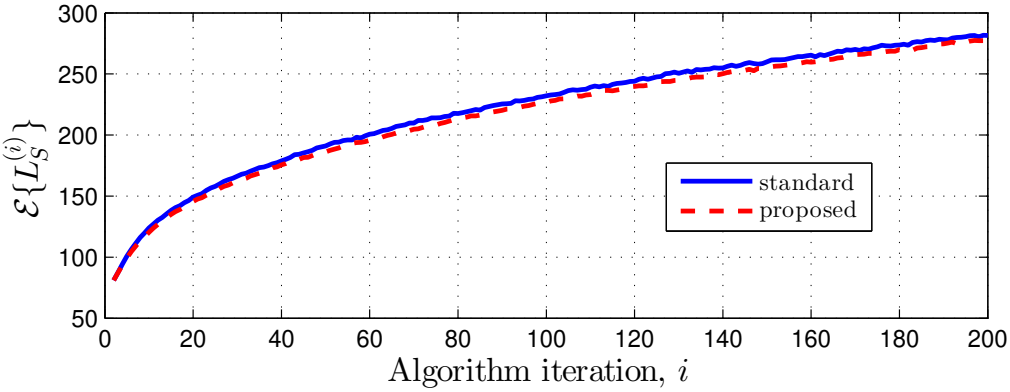

Figure 3.17: Length of parahermitian matrix $\boldsymbol{S}^{(i)}(z)$, $L_S^{(i)}$, versus iteration number for the proposed and standard implementations.

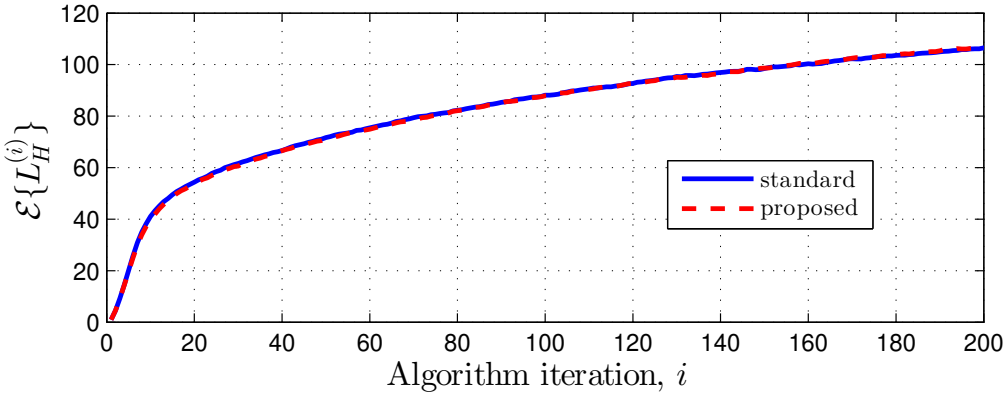

Figure 3.18: Length of paraunitary matrix $\boldsymbol{H}^{(i)}(z)$, $L_H^{(i)}$, versus iteration number for the proposed and standard implementations.

3.6 Restricting the Update Space of PEVD Algorithms

The research conducted in the previous section has bolstered the results of [50, 88] by confirming that restricting the search space of iterative PEVD algorithms can improve convergence speed. However, despite focussing on only a small portion of the parahermitian matrix during the search step, the entire matrix must still be updated at each iteration in these approaches. Using SMD as an example of an iterative PEVD algorithm, the work in this section expands upon this restricted search space idea by introducing a novel restricted update approach for the PEVD. The techniques involved are implemented within a restricted update SMD (RSMD) algorithm, which calculates the paraunitary matrix while restricting the search space of the algorithm and the portion of the parahermitian matrix that is updated at each iteration. As with the majority of iterative PEVD algorithms, the update step of SMD is its most computationally costly operation [45]; thus, a reduction in the complexity of this step is useful. It is demonstrated that by using the proposed RSMD algorithm instead of SMD, PEVD complexity and execution time can be significantly reduced with minimal impact on algorithm convergence.

An overview of the method is given in Section 3.6.1, with a more detailed description of the novel functionality in Section 3.6.2. Section 3.6.3 describes the algorithmic complexity of the approach. Simulation results comparing the performances of SMD and RSMD are presented in Section 3.6.4, and conclusions for this section are drawn in Section 3.6.5.

Elements of the work in this section can be found published in the proceedings of the 7th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing in a paper titled 'Restricted Update Sequential Matrix Diagonalisation for Parahermitian Matrices' [69].

3.6.1 Restricted Update SMD Algorithm

Similarly to SMD, which is described in Section A.1, the RSMD algorithm approximates a PEVD by iteratively diagonalising a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ over $i = 0 \dots \hat{I}$ iteration steps. Over the course of these iterations, the update space

contracts piecewise strictly monotonically. The term 'update space' here refers to the number of lags in the parahermitian matrix that are updated at each iteration of the algorithm. This restriction limits the number of search operations, but also reduces the computations required to update the increasingly diagonalised parahermitian matrix. The update space contracts until order zero is reached and the update space only includes the lag zero matrix. After this, in a so-called 'regeneration' step, the calculated paraunitary matrix is applied to the input matrix to construct the full-sized parahermitian factor. The update space is then maximised and thereafter again contracts monotonically over the following iterations. The maximum β of index $\alpha = 0, 1, \ldots, \beta$ — where α counts the number of regenerations — is not known a priori.

Following the αth regeneration step, in the ith iteration of RSMD, $\boldsymbol{S}^{(i)}(z) = \boldsymbol{R}_{(\alpha)}(z)$. Note, $\boldsymbol{R}_{(0)}(z) = \mathbf{Q}^{(0)}\boldsymbol{R}(z)\mathbf{Q}^{(0)\mathrm{H}}$, where $\mathbf{Q}^{(0)}$ diagonalises coefficient matrix $\mathbf{R}[0]$. The restricted update stage of RSMD restricts the search and update steps typically used in the SMD algorithm to only consider an iteratively decreasing selection of lags of $\boldsymbol{S}^{(i)}(z)$ around lag zero at the ith iteration. When the update space of $\boldsymbol{S}^{(i)}(z)$ reaches zero, the restricted update stage has produced a paraunitary matrix $\boldsymbol{F}_{(\alpha)}(z)$ such that matrix $\boldsymbol{R}_{(\alpha+1)}(z) = \boldsymbol{F}_{(\alpha)}(z)\boldsymbol{R}_{(\alpha)}(z)\tilde{\boldsymbol{F}}_{(\alpha)}(z)$ — which is generated during the matrix regeneration stage — is more diagonal that $\boldsymbol{R}_{(\alpha)}(z)$.

If the total number of algorithm iterations i exceeds a user-defined $I_{\max}$, or if the stopping criterion from SMD, (A.8), is satisfied, the RSMD algorithm ends with $\boldsymbol{D}(z) = \boldsymbol{R}_{(\alpha+1)}(z)$ and $\boldsymbol{F}(z) = \boldsymbol{F}'_{(\alpha+1)}(z)$ following the truncation of each matrix according to Appendix C. Here, $\boldsymbol{F}'_{(\alpha+1)}(z)$ is a concatenation of the paraunitary matrices generated for indices $0 \ldots \alpha$ and initial matrix $\mathbf{Q}^{(0)}$:

$$\boldsymbol{F}'_{(\alpha+1)}(z) = \boldsymbol{F}_{(\alpha)}(z) \cdots \boldsymbol{F}_{(0)}(z)\mathbf{Q}^{(0)} = \left(\prod_{x=0}^{\alpha} \boldsymbol{F}_{(\alpha-x)}(z) \right) \mathbf{Q}^{(0)} \, .$$

Algorithm 3 gives the pseudocode for RSMD. Output matrices $\boldsymbol{F}(z)$ and $\boldsymbol{D}(z)$ contain polynomial eigenvectors and eigenvalues, respectively. More detail of the restricted update component of the algorithm's operation is provided in Section 3.6.2. Proof of convergence of SMD — which focusses on the monotonic increase of on-diagonal en-

Input: $\boldsymbol{R}(z)$, μ, μ_t, ϵ, $I_{\max}$
Output: $\boldsymbol{D}(z)$, $\boldsymbol{F}(z)$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise $\mathbf{R}[0]$
$\boldsymbol{R}_{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\boldsymbol{R}(z)\mathbf{Q}^{(0)\mathrm{H}}$; $\boldsymbol{F}'_{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$; $\alpha \leftarrow 0$; $i \leftarrow 0$; stop $\leftarrow 0$
while stop $= 0$ **do**
  $[\boldsymbol{F}_{(\alpha)}(z), i, \text{stop}] \leftarrow \text{restrictedUpdate}(\boldsymbol{R}_{(\alpha)}(z), \mu_t, \epsilon, I_{\max}, i)$
  Regenerate matrix:
  $\boldsymbol{F}'_{(\alpha+1)}(z) \leftarrow \boldsymbol{F}_{(\alpha)}(z)\boldsymbol{F}'_{(\alpha)}(z)$
  $\boldsymbol{R}_{(\alpha+1)}(z) \leftarrow \boldsymbol{F}_{(\alpha)}(z)\boldsymbol{R}_{(\alpha)}(z)\tilde{\boldsymbol{F}}_{(\alpha)}(z)$
  Truncate $\boldsymbol{F}'_{(\alpha+1)}(z)$ using threshold μ_t according to Appendix C
  Truncate $\boldsymbol{R}_{(\alpha+1)}(z)$ using threshold μ according to Appendix C
  $\alpha \leftarrow \alpha + 1$
end
$\boldsymbol{F}(z) \leftarrow \boldsymbol{F}'_{(\alpha)}(z)$; $\boldsymbol{D}(z) \leftarrow \boldsymbol{R}_{(\alpha)}(z)$

Algorithm 3: RSMD Algorithm

ergy enforced by the algorithm — has been established in [45]; this proof also holds for RSMD.

3.6.2 Restricted Update Step

The restricted update step of RSMD functions similarly to the update step of SMD in Section A.1; however, the key difference is that RSMD increasingly restricts the number of lags of $\boldsymbol{S}^{(i)}(z)$ that are updated at each iteration i. Algorithm 4 provides pseudocode for the restrictedUpdate$(\cdot)$ function, whose operation is discussed below.

From a parahermitian matrix $\boldsymbol{R}_{(\alpha)}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ input to the restrictedUpdate$(\cdot)$ function for index α, during the ith iteration of RSMD, a matrix $\boldsymbol{S}^{(i-1)}(z) = \boldsymbol{R}_{(\alpha)}(z)$ with maximum lag $T^{(i-1)}$ is formed. As in the standard SMD algorithm, the $k^{(i)}$th column and row with maximum energy are found and shifted by $|\tau^{(i)}|$ to lag zero using a delay matrix $\boldsymbol{\Lambda}^{(i)}(z)$ to produce $\boldsymbol{S}^{(i)\prime}(z)$.

A unitary matrix $\mathbf{Q}^{(i)}$ is generated from an EVD of lag zero $\mathbf{S}^{(i)\prime}[0]$, but is only applied to an 'update region' matrix $\boldsymbol{S}^{(i)\prime\prime}(z)$, which contains the central $(2(T^{(i-1)} - |\tau^{(i)}|) + 1)$ lags of $\boldsymbol{S}^{(i)\prime}(z)$. Thus, matrix $\boldsymbol{S}^{(i)}(z) = \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$ is formed, which has maximum lag $T^{(i)} = T^{(i-1)} - |\tau^{(i)}|$.

The coefficients of $\mathbf{S}^{(i)}[\tau]$ at lags $|\tau| > T^{(i)}$, which are zero by definition — and not

Input: $\boldsymbol{R}_{(\alpha)}(z)$, μ_t, ϵ, $I_{\max}$, i
Output: $\boldsymbol{F}_{(\alpha)}(z)$, i, stop
$\boldsymbol{S}^{(i)}(z) \leftarrow \boldsymbol{R}_{(\alpha)}(z)$; $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{I}_{M\times M}$; stop $\leftarrow 0$
do
  $i \leftarrow i+1$
  Find $\{k^{(i)}, \tau^{(i)}\}$ from (A.6); generate $\boldsymbol{\Lambda}^{(i)}(z)$ from (A.3)
  $\boldsymbol{S}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{\Lambda}}^{(i)}(z)$
  $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
  Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\mathbf{S}^{(i)\prime}[0]$
  $T^{(i-1)}$ is maximum lag of $\boldsymbol{S}^{(i-1)}(z)$
  $\boldsymbol{S}^{(i)\prime\prime}(z) \leftarrow \sum_{\tau=-T^{(i-1)}+|\tau^{(i)}|}^{T^{(i-1)}-|\tau^{(i)}|} \mathbf{S}^{(i)\prime}[\tau]z^{-\tau}$
  $\boldsymbol{S}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$
  $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime}(z)$
  Truncate $\boldsymbol{H}^{(i)}(z)$ using threshold μ_t according to Appendix C
  if $i > I_{\max}$ or (A.8) satisfied **then**
    stop $\leftarrow 1$
  end
while stop $= 0$ and $(T^{(i-1)} - |\tau^{(i)}|) > 0$
$\boldsymbol{F}_{(\alpha)}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$

Algorithm 4: restrictedUpdate($\cdot$) Function

obtained from the transformation of $\boldsymbol{S}^{(i-1)}(z)$ — must be kept outside of the update region in the next iteration, $\boldsymbol{S}^{(i+1)\prime\prime}(z)$, if the accuracy of the decomposition is to be maintained. To guarantee that these coefficients are excluded, the update region must shrink by the maximum possible distance that the coefficients can travel towards lag zero, $|\tau^{(i+1)}|$. That is, $\boldsymbol{S}^{(i+1)\prime\prime}(z)$ should only contain the central $2(T^{(i)} - |\tau^{(i+1)}|) + 1$ lags of $\boldsymbol{S}^{(i+1)\prime}(z)$.

Iterations of this process continue in the same manner until the end of some iteration $I_{(\alpha)}$, when the maximum lag of matrix $\boldsymbol{S}^{(I_{(\alpha)})}(z) = \mathbf{Q}^{(I_{(\alpha)})}\boldsymbol{S}^{(I_{(\alpha)})\prime\prime}(z)\mathbf{Q}^{(I_{(\alpha)})\mathrm{H}}$ is $T^{(I_{(\alpha)}-1)} - |\tau^{(I_{(\alpha)})}| = 0$, or when $\boldsymbol{S}^{(I_{(\alpha)})}(z)$ is sufficiently diagonalised with dominant off-diagonal column norm below a threshold ϵ as in (A.8). Alternatively, the restricted update process ends if the total number of iterations of RSMD exceeds some user-defined value, $I_{\max}$.

Figure 3.19 demonstrates the restricted update step for $M = 5$ and $T^{(i-1)} = 3$. As

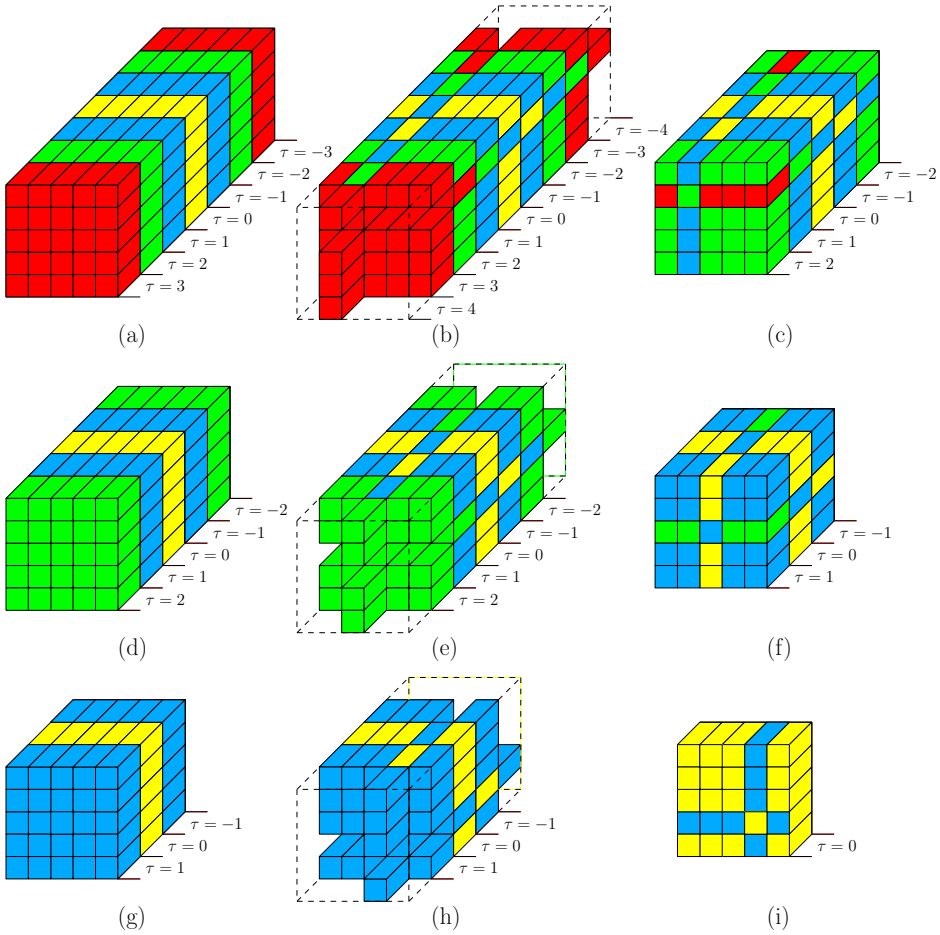

Figure 3.19: (a) Original matrix $\boldsymbol{S}^{(i-1)}(z) = \boldsymbol{R}_{(\alpha)}(z) : \mathbb{C} \to \mathbb{C}^{5\times 5}$ with maximum lag $T^{(i-1)} = 3$ is input to restrictedUpdate$(\cdot)$; (b) shifting of row and column energy to lag zero ($k^{(i)} = 2$, $\tau^{(i)} = -1$); (c) central matrix with maximum lag $(T^{(i-1)} - |\tau^{(i)}|) = 2$, $\boldsymbol{S}^{(i)\prime\prime}(z)$, is extracted. (d) $\boldsymbol{S}^{(i)}(z) = \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$; (e) $k^{(i+1)} = 3$, $\tau^{(i+1)} = -1$; (f) $\boldsymbol{S}^{(i+1)\prime\prime}(z)$ extracted. (g) $\boldsymbol{S}^{(i+1)}(z)$; (h) $k^{(i+2)} = 4$, $\tau^{(i+2)} = -1$; (i) $\boldsymbol{S}^{(i+2)\prime\prime}(z)$ is extracted.

can be seen, after three iterations, the maximum lag of the matrix in Figure 3.19(i) is equal to zero; thus, $I_{(\alpha)} = 3$.

Note that $\boldsymbol{S}^{(i)}(z)$ will typically have fewer lags than the equivalent matrix in the ith iteration of the traditional SMD algorithm; thus, the search to identify the $k^{(i)}$th column and row in the proposed approach may produce an inferior result to the search in SMD. However, it is demonstrated that this does not significantly affect algorithm convergence in Section 3.6.4.

3.6.3 Complexity Reduction

Following the methodology of the complexity calculations for SMD in Section A.1.2, at iteration i of restrictedUpdate$(\cdot)$ within RSMD, the number of MACs required to generate $\boldsymbol{S}^{(i)}(z) = \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$ can be approximated by $2L\{\boldsymbol{S}^{(i)\prime\prime}(z)\}M^3$. Here, operator $L\{\cdot\}$ computes the length of a polynomial matrix and the multiplication of 2 $M \times M$ matrices is assumed to require M^3 MACs [2, 95]. To update $\boldsymbol{H}^{(i)}(z)$, $L\{\boldsymbol{H}^{(i)\prime}(z)\}M^3$ MACs are required. Note that $L\{\boldsymbol{H}^{(i)}(z)\}$ is reset to one following matrix regeneration. Assuming $\epsilon = 0$, the cumulative complexity of restrictedUpdate$(\cdot)$ is therefore approximately

$$C_{\mathrm{RU}}(I_{\max}) = M^3 \sum_{i=1}^{I_{\max}} (2L\{\boldsymbol{S}^{(i)\prime\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime}(z)\}) \; . \tag{3.17}$$

During matrix regeneration, $\boldsymbol{F}'_{(\alpha+1)}(z) = \boldsymbol{F}_{(\alpha}(z)\boldsymbol{F}'_{(\alpha)}(z)$ and $\boldsymbol{R}_{(\alpha+1)}(z) = \boldsymbol{F}_{(\alpha)}(z)\boldsymbol{R}_{(\alpha)}(z)\tilde{\boldsymbol{F}}_{(\alpha)}(z)$ are computed. If these are computed in the frequency domain as recommended in Section 3.2.2 — and if the MACs required to compute the FFT and IFFT are ignored — the former requires approximately $(L\{\boldsymbol{F}_{(\alpha)}(z)\} + L\{\boldsymbol{F}'_{(\alpha)}(z)\} - 1)M^3$ MACs, and the latter requires approximately $(2L\{\boldsymbol{F}_{(\alpha)}(z)\} + L\{\boldsymbol{R}_{(\alpha)}(z)\} - 2)M^3$ MACs; thus, the cumulative complexity of matrix regeneration for β total regenerations in RSMD is approximately

$$C_{\mathrm{MR}}(\beta) = M^3 \sum_{\alpha=0}^{\beta-1} (3L\{\boldsymbol{F}_{(\alpha)}(z)\} + L\{\boldsymbol{F}'_{(\alpha)}(z)\} + L\{\boldsymbol{R}_{(\boldsymbol{\alpha})}(z)\} - 3) \; .$$

The total cumulative complexity of RSMD can therefore be approximated as

$$C_{\mathrm{RSMD}}(I_{\max}, \beta) = C_{\mathrm{RU}}(I_{\max}) + C_{\mathrm{MR}}(\beta) \; . \tag{3.18}$$

If the savings made during the restricted update step are larger than the overheads added by the matrix regeneration step — i.e., if $(C_{\mathrm{SMD}}(I_{\max}) - C_{\mathrm{RU}}(I_{\max})) > C_{\mathrm{MR}}(\beta))$, where $C_{\mathrm{SMD}}(I_{\max})$ is obtained from (A.11) — the total cumulative complexity of RSMD will be lower than SMD.

3.6.4 Results and Discussion

Simulation Scenario

The simulations below are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M \in \{10;\ 20\}$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\mathrm{DR}} = 30$.

Each algorithm is executed for a maximum of $I_{\max} = 200$ iterations with a stopping threshold of $\epsilon = 0$ and polynomial matrix truncation parameters of $\mu = \mu_t = 10^{-6}$. At every iteration of both implementations, the diagonalisation metric defined in Section A.3 is recorded alongside the elapsed execution time and the estimated cumulative complexities from the equations of Sections A.1.2 and 3.6.3. The length of $\boldsymbol{F}(z)$ is recorded upon each algorithm's completion.

Diagonalisation

The ensemble-averaged diagonalisation was calculated for the standard SMD and proposed RSMD implementations. The diagonalisation performance versus cumulative complexity and time for both methods are shown in Figures 3.20 and 3.21, respectively. The curves of Figure 3.20 demonstrate that for $M \in \{10;\ 20\}$, the proposed implementation operates with a lower estimated cumulative complexity than the standard realisation, and is able to achieve a similar degree of diagonalisation — indicating that convergence is not affected by the use of a restricted update procedure. In addition, Figure 3.21 shows that the lower complexity associated with the proposed approach translates to a faster diagonalisation than observed for SMD. Note that the translation from cumulative complexity to execution time is not linear, as other processes — such as shifting operations — contribute to the algorithm run-time that are not accounted for in Figure 3.20.

Paraunitary Filter Length

The ensemble-averaged paraunitary filter lengths were calculated for both algorithms. For $M = 10$, $\boldsymbol{F}(z)$ from SMD and RSMD was of length 84.4 and 87.1, respectively.

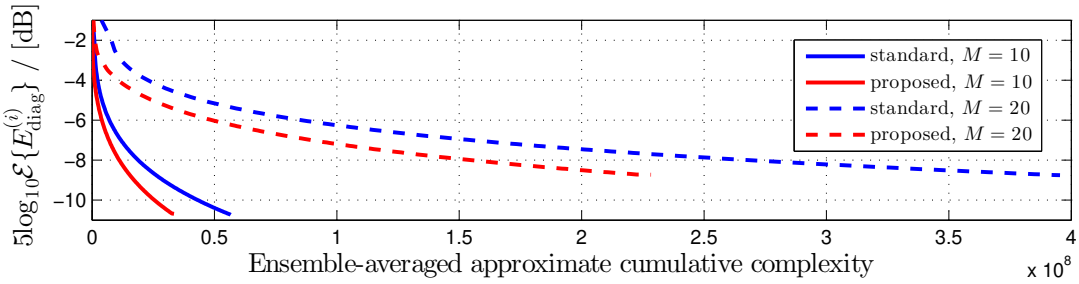

Figure 3.20: Diagonalisation metric versus cumulative algorithm complexity for the proposed and standard implementations for $M \in \{10;\ 20\}$.

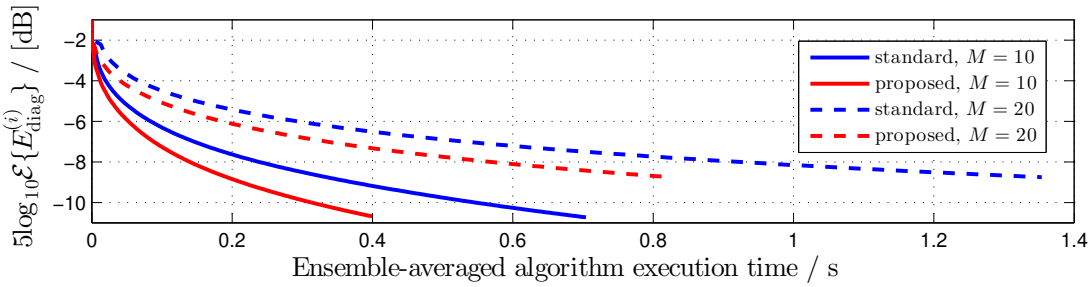

Figure 3.21: Diagonalisation metric versus algorithm execution time for the proposed and standard implementations for $M \in \{10;\ 20\}$.

Similarly for $M = 20$, lengths of 70.9 and 73.0 were observed for SMD and RSMD. The paraunitary filters generated by RSMD were therefore slightly longer than those from SMD. While the exact reason for this discrepancy is not known, a reasonable guess could assume that it arises as an accumulation of slight differences between the shifting and truncation operations used to generate the paraunitary matrices in each algorithm.

3.6.5 Summary

In this section, a novel restricted update sequential matrix diagonalisation algorithm has been introduced. This algorithm can produce the same quality of decomposition as SMD, but with decreased computational complexity. Simulation results underline that the same diagonalisation performance can be achieved by both methods, but within a shorter execution time for RSMD. While RSMD gives a slight increase in paraunitary filter length, it is not significant enough to negate the performance gains made elsewhere.

The proposed RSMD algorithm therefore has the potential to reduce time and complexity requirements in application scenarios involving the PEVD. In addition, the restricted update approach proposed here can be extended to any iterative PEVD algorithm in [6,46–50] by adapting the restricted update and matrix regeneration steps accordingly.

3.7 Multiple Shift Approach to the Polynomial Matrix QR Decomposition

In recent years, several algorithms for the iterative calculation of a polynomial matrix QR decomposition (PQRD) have been introduced [40–42]. The PQRD is a generalisation of the ordinary QR decomposition and uses paraunitary operations to upper-triangularise a polynomial matrix. Analogously to the QR decomposition, which can be used to generate an EVD of a Hermitian matrix via the QR algorithm [87], the PQRD can be used to find the PEVD of a parahermitian matrix. The PQRD has also been used to generate a polynomial matrix SVD algorithm in [43].

Applying a multiple shift approach to the SMD algorithm for the PEVD of parahermitian matrices has been proven to be beneficial in [46, 47]. This section addresses a multiple shift strategy that can be applied to existing PQRD algorithms. As an example, the polynomial matrix QR decomposition by columns (PQRD-BC) algorithm, defined as an improvement to the PQRD by steps algorithm from [40, 41] in [42], is modified to incorporate these techniques. It is demonstrated that with the proposed strategy, the computation time of the resulting multiple shift PQRD-BC (MS-PQRD-BC) algorithm is lower than PQRD-BC.

An overview of the proposed approach is given in Section 3.7.1. Simulation results comparing the performances of PQRD-BC and MS-PQRD-BC are presented in Section 3.7.2, and conclusions for this section are drawn in Section 3.7.3.

Elements of the work in this section can be found published in the proceedings of the 11th IMA International Conference on Mathematics in Signal Processing in a paper titled ‘Multiple Shift QR Decomposition for Polynomial Matrices’ [70].

3.7.1 Multiple Shift Strategy

The MS-PQRD-BC algorithm approximates the PQRD using a series of elementary paraunitary operations to iteratively upper-triangularise a polynomial matrix $\boldsymbol{A}(z) : \mathbb{C} \rightarrow \mathbb{C}^{P\times Q}$. The algorithm calculates a paraunitary matrix $\boldsymbol{Q}(z) \in \mathbb{C}^{P\times P}$ such that

$$\boldsymbol{Q}(z)\boldsymbol{A}(z) \approx \boldsymbol{R}(z) \, , \tag{3.19}$$

where $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{P\times Q}$ is an upper-triangular polynomial matrix. The series of paraunitary operations used to compute the polynomial matrix $\boldsymbol{Q}(z)$ in (3.19) can be broken down into elementary delay and rotation matrices similar to those described by [4]. These matrices can be used to formulate an elementary polynomial Givens rotation (EPGR).

Elementary Polynomial Givens Rotation

An EPGR is a polynomial matrix that can be applied to a polynomial matrix $\boldsymbol{A}(z) : \mathbb{C} \rightarrow \mathbb{C}^{P\times Q}$ to zero one coefficient of a polynomial element. For example, if the polynomial coefficient in the jth row and kth column of $\mathbf{A}[\tau]$, $a_{j,k}[\tau]$, for $j > k$ is to be made equal to zero, the required EPGR takes the form of

$$\boldsymbol{G}^{(j,k,\tau,\alpha,\theta,\phi)}(z) = \begin{bmatrix} \mathbf{I}_{k-1} & & & & \\ & \mathrm{e}^{\mathrm{j}\alpha}\cos\theta & \cdots & \mathrm{e}^{\mathrm{j}\phi}z^{\tau}\sin\theta & \\ & \vdots & \mathbf{I}_{j-k-1} & \vdots & \\ & -\mathrm{e}^{-\mathrm{j}\phi}\sin\theta & \cdots & \mathrm{e}^{-\mathrm{j}\alpha}z^{\tau}\cos\theta & \\ & & & & \mathbf{I}_{P-j} \end{bmatrix} , \tag{3.20}$$

where $\mathbf{I}_{k-1}$ denotes a $(k-1)\times(k-1)$ identity matrix. The rotation angles θ, α, and ϕ are chosen such that

$$\theta = \begin{cases} \tan^{-1}\left(\dfrac{|a_{j,k}[\tau]|}{|a_{k,k}[0]|}\right), & \text{if } a_{k,k}[0] \neq 0 \\ \pi/2, & \text{if } a_{k,k}[0] = 0 \end{cases} \tag{3.21}$$

$$\alpha = -\arg(a_{k,k}[0]) \;\text{ and }\; \phi = -\arg(a_{j,k}[\tau])\;; \tag{3.22}$$

thus resulting in $a'_{j,k}[0] = 0$, where $\boldsymbol{A}'(z) = \boldsymbol{G}^{(j,k,\tau,\alpha,\theta,\phi)}(z)\boldsymbol{A}(z)$. Furthermore, following the application of the EPGR, the coefficient $a'_{k,k}[0]$ is real and has increased in magnitude such that $|a'_{k,k}[0]|^2 = |a_{k,k}[0]|^2 + |a_{j,k}[\tau]|^2$.

MS-PQRD-BC Algorithm

Using the work of [46,47] as inspiration, this section describes a multiple shift approach to the PQRD-BC algorithm. By applying a number of row shifts at each iteration of the developed MS-PQRD-BC algorithm, the largest polynomial coefficient in each row beneath the diagonal in a single column can be transferred to lag zero at once. A series of Givens rotations can then be applied to approximately zero each shifted coefficient. This process can be repeated for each column of the matrix until a similar stopping criterion to PQRD-BC is reached.

The MS-PQRD-BC algorithm operates as a series of ordered column-steps where at each step, all coefficients relating to all polynomial elements beneath the diagonal in one column of the matrix are driven sufficiently small by applying a series of EPGR matrices interspersed with paraunitary delay matrices. The term 'column-step' is used, as the columns of $\boldsymbol{A}(z) \in \mathbb{C}^{P\times Q}$ are visited in sequence according to the ordering $k = 1, 2, \ldots, \min\{(P-1), Q\}$.

Noting that $\boldsymbol{A}^{(0,1)}(z) = \boldsymbol{A}(z)$, at iteration i, $i = 1 \ldots I_{(k)}$, of column-step k the search step of MS-PQRD-BC locates $n = (P-k)$ coefficients with maximum magnitude from the n rows beneath the diagonal in column k; i.e., a dominant coefficient is found for each row according to

$$\boldsymbol{\tau}^{(i,k)} = \underset{\tau_1,\ldots,\tau_n}{\arg\max} \left\{ |a^{(i-1,k)}_{(k+1),k}[\tau_1]| + \cdots + |a^{(i-1,k)}_{P,k}[\tau_n]| \right\} \quad , \tag{3.23}$$

where $\boldsymbol{\tau}^{(i,k)} = [\tau_1^{(i,k)}, \ldots, \tau_n^{(i,k)}]^{\mathrm{T}}$ contains the lags of the dominant coefficients for rows $j = k+1 \ldots P$ and $a^{(i-1,k)}_{j,k}[\tau]$ represents the element in the jth row and kth column of the coefficient matrix $\mathbf{A}^{(i-1,k)}[\tau]$.

Following the identification of $\boldsymbol{\tau}^{(i,k)}$ in (3.23), a delay matrix $\boldsymbol{\Lambda}^{(i,k)}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$,

which takes the form

$$\mathbf{\Lambda}^{(i,k)}(z) = \begin{bmatrix} \mathbf{I}_k & & & \\ & z^{\tau_1^{(i,k)}} & & \\ & & \ddots & \\ & & & z^{\tau_n^{(i,k)}} \end{bmatrix}, \tag{3.24}$$

can be applied to $\boldsymbol{A}^{(i-1,k)}(z)$ to induce a $\tau_m^{(i,k)}$-fold delay to the mth row beneath the diagonal in column k, where $m = 1 \dots n$. Lag zero of the resulting matrix $\boldsymbol{A}^{(i-1,k)\prime}(z) = \mathbf{\Lambda}^{(i,k)}(z)\boldsymbol{A}^{(i-1,k)}(z)$ will contain the dominant coefficients from column k.

EPGRs are then applied to $\mathbf{A}^{(i-1,k)\prime}[0]$ in sequence to drive elements $a_{(k+1),k}^{(i-1,k)\prime}[0] \dots a_{P,k}^{(i-1,k)\prime}[0]$ to zero according to

$$\begin{aligned} \mathbf{A}^{(i-1,k)\prime\prime}[0] &= \left(\mathbf{G}^{(k+1,k,\gamma_1)} \cdots \mathbf{G}^{(P,k,\gamma_n)}\right) \mathbf{A}^{(i-1,k)\prime}[0] \\ &= \mathbf{H}^{(i,k)} \mathbf{A}^{(i-1,k)\prime}[0] \,, \end{aligned} \tag{3.25}$$

where the elements beneath the diagonal in the kth column of $\mathbf{A}^{(i-1,k)\prime\prime}[0]$ are zero, and $\mathbf{G}^{(j,k,\gamma)}$ represents the EPGR required to zero element $a_{j,k}^{(i-1,k)\prime}[0]$ of $\mathbf{A}^{(i-1,k)\prime}[0]$. Vector $\boldsymbol{\gamma} = [\alpha, \theta, \phi]^{\mathrm{T}}$ contains the calculated rotation angles from (3.21) and (3.22) for the targeted element. Note that $\mathbf{G}^{(j,k,\gamma)}$ is an EPGR without shift parameter τ, and is therefore a simple unitary matrix.

The matrix $\mathbf{H}^{(i,k)}$ in (3.25) is the product of EPGRs required to drive the dominant coefficients in iteration i of MS-PQRD-BC to zero. This unitary matrix is applied to all lags of $\boldsymbol{A}^{(i-1,k)\prime}(z)$ to generate $\boldsymbol{A}^{(i-1,k)\prime\prime}(z) = \mathbf{H}^{(i,k)}\boldsymbol{A}^{(i-1,k)\prime}(z)$.

For stability reasons [42, 43], the final step of an iteration of MS-PQRD-BC is to apply an inverse delay matrix $\tilde{\mathbf{\Lambda}}^{(i,k)}(z)$ to produce the overall transformation

$$\begin{aligned} \boldsymbol{A}^{(i,k)}(z) &= \tilde{\mathbf{\Lambda}}^{(i,k)}(z)\mathbf{H}^{(i,k)}\mathbf{\Lambda}^{(i,k)}(z)\boldsymbol{A}^{(i-1,k)}(z) \\ &= \boldsymbol{Q}^{(i,k)}(z)\boldsymbol{A}^{(i-1,k)}(z) \,. \end{aligned} \tag{3.26}$$

This iterative process is repeated for a maximum of $I_{(k)}$ iterations or until all co-

efficients associated with polynomial elements beneath the diagonal in the kth column of the matrix are sufficiently small in magnitude and therefore satisfy the stopping condition

$$|a_{j,k}^{(i,k)}[\tau]| < \epsilon \tag{3.27}$$

for $j > k$ and $\forall\ \tau \in \mathbb{Z}$ where $\epsilon > 0$ is a prespecified small value. The overall transformation performed in the kth column-step of the algorithm is of the form

$$\boldsymbol{A}^{(k)}(z) = \boldsymbol{Q}^{(k)}(z)\boldsymbol{A}(z) \tag{3.28}$$

where $\boldsymbol{Q}^{(k)}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$ is the paraunitary product of all EPGRs and delay matrices applied during iterations $i = 1 \dots I_{(k)}$ within column-steps $1 \dots k$:

$$\boldsymbol{Q}^{(k)}(z) = \prod_{i=1}^{I_{(k)}} \boldsymbol{Q}^{(i,k)}(z) \cdots \prod_{i=1}^{I_{(1)}} \boldsymbol{Q}^{(i,1)}(z)\ , \tag{3.29}$$

where

$$\prod_{i=1}^{I_{(k)}} \boldsymbol{Q}^{(i,k)}(z) = \boldsymbol{Q}^{(I_{(k)},k)}(z)\boldsymbol{Q}^{(I_{(k)}-1,k)}(z) \cdots \boldsymbol{Q}^{(2,k)}(z)\boldsymbol{Q}^{(1,k)}(z) \tag{3.30}$$

is the product of all EPGRs and delay matrices applied during iterations $i = 1 \dots I_{(k)}$ and $I_{(k)}$ is the maximum number of iterations within column-step k, with $I_{(k)} = \lceil \frac{I}{P-k} \rceil$ for some $I \in \mathbb{N}$.

Following the kth column-step, all coefficients relating to all polynomial elements beneath the diagonal in the kth column of the matrix $\boldsymbol{A}(z)$ will be sufficiently small. After $\ell = \min\{(P-1), Q\}$ column-steps of the algorithm, all columns of the matrix have been visited, and the transformation is of the form

$$\hat{\boldsymbol{R}}(z) = \boldsymbol{Q}(z)\boldsymbol{A}(z) \tag{3.31}$$

where $\boldsymbol{Q}(z) = \boldsymbol{Q}^{(\ell)}(z)$ is the paraunitary product of all EPGRs and delay matrices applied during column-steps $1 \dots \ell$ and $\hat{\boldsymbol{R}}(z)$ is approximately upper-triangular.

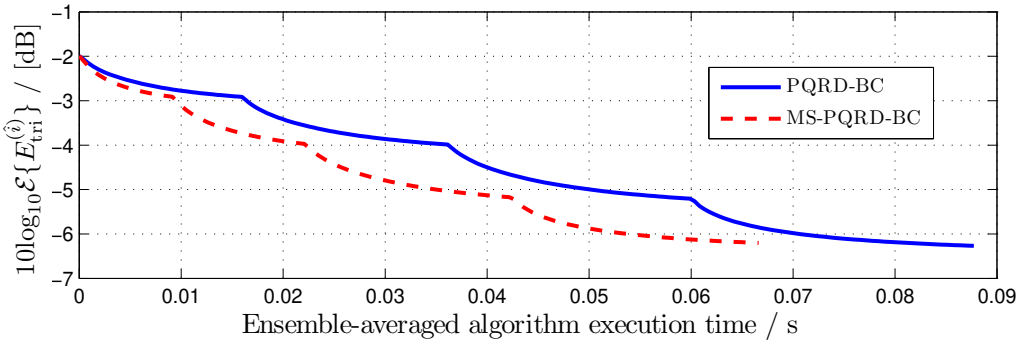

Figure 3.22: Upper-triangularisation metric versus algorithm execution time for the proposed and standard implementations for $\boldsymbol{A}(z) : \mathbb{C} \to \mathbb{C}^{5\times5}$ with polynomial order 14.

3.7.2 Results and Discussion

Simulation Scenario

Simulations are performed over an ensemble of 10^3 instantiations of $\boldsymbol{A}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$, $M \in \{5; 7; 9; 11; 13\}$, with a polynomial order of 14, $\epsilon = 10^{-6}$, and $I = 50$. The polynomial coefficients of $\boldsymbol{A}(z)$ are drawn from a complex Gaussian source with unit variance and zero mean.

A suitable normalised upper-triangularisation metric $E_{\text{tri}}^{(\hat{i})}$ defined in Section A.3 is used for evaluating the standard PQRD-BC and proposed MS-PQRD-BC implementations, which divides the lower-triangular energy in $\boldsymbol{A}(z)$ at the $\hat{i}$th algorithm iteration — ignoring column-steps — by the total energy.

Upper-Triangularisation

The ensemble-averaged upper-triangularisation versus algorithm execution time for both methods with $M = 5$ is shown in Figure 3.22. The curves demonstrate that the multiple shift implementation obtains a faster upper-triangularisation than the standard realisation. The stepped characteristic of both curves is a natural outcome of the column-step approach of both algorithms; each step corresponds to the minimisation of below-diagonal energy in a single column. For the 5×5 matrix, four columns are targeted; thus, four steps are seen in the curves.

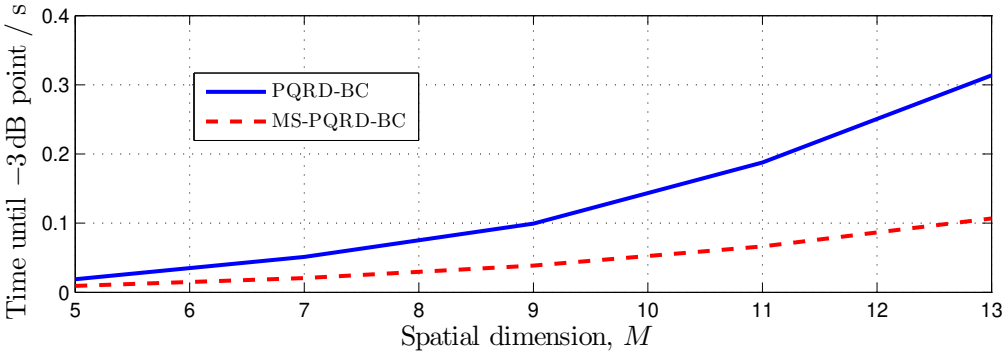

Figure 3.23: Algorithm execution time required to achieve $10\log_{10} E_{\text{tri}}^{(\hat{i})} = -3\,\text{dB}$ for the proposed and standard implementations for $M \in \{5; 7; 9; 11; 13\}$.

The ensemble-averaged time taken for both algorithms to attain an upper-triangularisation of $10\log_{10} E_{\text{tri}}^{(\hat{i})} = -3\,\text{dB}$ is plotted versus spatial dimension M in Figure 3.23. The widening gap between the two curves indicates that MS-PQRD-BC increasingly outperforms PQRD-BC for increasing M.

3.7.3 Summary

In this work, a multiple shift approach for an existing polynomial QR decomposition algorithm has been proposed. It has been shown through simulation that the implementation of this approach translates to an increase in upper-triangularisation speed of the created MS-PQRD-BC algorithm. Through an extension of the QR algorithm [87] to the case of polynomial matrices, successive applications of the PQRD can generate a PEVD. The improvements made to an existing PQRD algorithm in this section can therefore be extended to the PEVD; however, the standalone PEVD algorithms of Section 2.3.2 are likely to converge more quickly than an approach using multiple instances of the PQRD. This is especially true if the PEVD algorithm improvements discussed in this chapter are utilised.

3.8 Compensated Row-Shift Truncation of Paraunitary Matrices

A number of applications [17–19, 21–23, 25] utilise the PEVD's paraunitary matrix of polynomial eigenvectors, which can be considered to be a lossless FIR filter bank. Given that low implementation costs are desirable, minimising the order of this matrix is of importance.

As an extension of the traditional polynomial matrix truncation strategies in [60,61], work in [62, 63] has shown that by employing a row-shift truncation (RST) scheme for paraunitary matrices from the SBR2 [6] PEVD algorithm, filter order can be reduced with little loss to paraunitarity of the eigenvectors. This strategy requires a diagonal $\boldsymbol{D}(z)$ to be effective without introducing decomposition error. In this section, a novel variation of row-shift truncation titled compensated row-shift truncation (CRST) is introduced; this method is able to reduce paraunitary matrix order without significantly affecting decomposition error for even non-diagonal $\boldsymbol{D}(z)$.

An overview of the proposed approach is given in Section 3.8.1. Simulation results comparing the performances of traditional, row-shift, and compensated row-shift truncation when applied after SBR2 algorithm completion are presented in Section 3.8.2. Further performance comparisons when using each truncation strategy at each iteration of the SBR2 algorithm are given in Section 3.8.3. Conclusions for this section are drawn in Section 3.8.4.

3.8.1 Compensated Row-Shift Truncation Strategy

The RST method [62, 63] exploits ambiguity in the paraunitary matrices [62, 96]. This arises as a generalisation of a phase ambiguity inherent to eigenvectors from a standard EVD [2], which in the polynomial case extends to arbitrary phase responses or all-pass filters. The simplest manifestation of such filters can form an integer number of unit delays. For a diagonal $\boldsymbol{D}(z)$, this ambiguity permits eigenvectors $\boldsymbol{F}(z)$ to be replaced by a lower order $\hat{\boldsymbol{F}}(z)$, where $\hat{\boldsymbol{F}}(z) = \boldsymbol{\Gamma}(z)\boldsymbol{F}(z)$ and $\boldsymbol{\Gamma}(z)$ is a paraunitary diagonal

matrix. In this case, since diagonal matrices commute,

$$\begin{aligned}\boldsymbol{R}(z) \approx \tilde{\boldsymbol{F}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z) &= \tilde{\hat{\boldsymbol{F}}}(z)\boldsymbol{\Gamma}(z)\boldsymbol{D}(z)\tilde{\boldsymbol{\Gamma}}(z)\hat{\boldsymbol{F}}(z) \\ &= \tilde{\hat{\boldsymbol{F}}}(z)\boldsymbol{D}(z)\hat{\boldsymbol{F}}(z)\ , \end{aligned} \tag{3.32}$$

and $\boldsymbol{D}(z)$ in unaffected. If, however, $\boldsymbol{D}(z)$ is not diagonal — as is often the case for a reasonable number of PEVD algorithm iterations when spatial dimension M is large — $\boldsymbol{\Gamma}(z)$ does not cancel as in (3.32). In a novel variation of row-shift truncation from [62] titled compensated row-shift truncation, this matrix $\boldsymbol{\Gamma}(z)$ is incorporated into the parahermitian matrix to avoid negatively impacting the decomposition error. Since one is typically only interested in the approximate polynomial eigenvalues stored on the diagonal of $\boldsymbol{D}(z)$, the augmented parahermitian matrix $\hat{\boldsymbol{D}}(z) = \boldsymbol{\Gamma}(z)\boldsymbol{D}(z)\tilde{\boldsymbol{\Gamma}}(z)$ is used, which has the same approximate polynomial eigenvalues. The decomposition accuracy can now be maintained while using the lower order $\hat{\boldsymbol{F}}(z)$:

$$\boldsymbol{R}(z) \approx \tilde{\hat{\boldsymbol{F}}}(z)\hat{\boldsymbol{D}}(z)\hat{\boldsymbol{F}}(z)\ . \tag{3.33}$$

While $\hat{\boldsymbol{D}}(z)$ has a higher polynomial order than $\boldsymbol{D}(z)$, the order of the paraunitary matrix is typically more important for application purposes [17–19, 21–23, 25].

From the original RST approach of [62], $\boldsymbol{\Gamma}(z)$ takes the form

$$\boldsymbol{\Gamma}(z) = \operatorname{diag}\{z^{\tau_1},\ z^{\tau_2},\ \ldots,\ z^{\tau_M}\}\ . \tag{3.34}$$

The delay matrix $\boldsymbol{\Gamma}(z)$ has the effect of shifting the mth row of the paraunitary matrix $\boldsymbol{F}(z)$ by τ_m. These row shifts can be used to align the first polynomial coefficients in each row of the paraunitary matrix following the independent truncation of each row via the process below.

The matrix $\boldsymbol{F}(z)$ can be subdivided into its M row vectors $\boldsymbol{f}_m(z) : \mathbb{C} \to \mathbb{C}^{1\times M}$,

$m = 1 \dots M$,

$$\boldsymbol{F}(z) = \begin{bmatrix} \boldsymbol{f}_1(z) \\ \vdots \\ \boldsymbol{f}_M(z) \end{bmatrix} . \tag{3.35}$$

Each row — which has minimum lag $T_{1,m}$ and maximum lag $T_{2,m}$ — is then truncated individually according to $f_{\text{trim}}(\mathbf{f}_m[\tau], \mu)$ from (C.1) in Appendix C. The row shifts, τ_m, in (3.34) are then set equal to $(T_{1,m} + T_{3,m}(\mu))$, $m = 1 \dots M$, such that the minimum lag of each shifted row $\hat{\boldsymbol{f}}_m(z)$ is zero. Here, $T_{3,m}(\mu)$ is the $T_3(\mu)$ obtained via (C.1) in Appendix C for the mth row. Following row-shift truncation, each row of $\hat{\boldsymbol{F}}(z)$ has order $T_m(\mu)$, and the order of the paraunitary matrix is $\max\limits_{m=1\dots M}\{T_m(\mu)\}$, where $T_m(\mu) = T_{2,m} - (T_{3,m}(\mu) + T_{4,m}(\mu))$ and $T_{4,m}(\mu)$ is the $T_4(\mu)$ obtained via (C.1).

When applying CRST to a matrix $\boldsymbol{F}(z)$, one therefore obtains

$$[\hat{\boldsymbol{F}}(z), \hat{\boldsymbol{D}}(z)] \leftarrow f_{\text{crst}}(\boldsymbol{F}(z), \boldsymbol{D}(z), \mu) , \tag{3.36}$$

where

$$\hat{\mathbf{f}}_m[\tau] \circ\!\!-\!\!\bullet\, \hat{\boldsymbol{f}}_m(z) = z^{\tau_m} \sum_{\tau=T_{1,m}}^{T_{2,m}} f_{\text{trim}}(\mathbf{f}_m[\tau], \mu) z^{-\tau} . \tag{3.37}$$

3.8.2 Truncating After Algorithm Completion

Simulation Scenario

The simulations below are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M \times M}$ obtained from the randomised source model in Appendix B [45] with $M = 5$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$.

Three instances of SBR2 are implemented; the first uses the standard SBR2 paraunitary matrix truncation strategy of Appendix C with threshold $\mu_{\text{trad}} = 10^{-6}$, the second uses the RST strategy of [62] with $\mu_{\text{RST}} = 10^{-6}$, and the third uses CRST with $\mu_{\text{CRST}} = 10^{-6}$. Each instance of SBR2 is executed for a maximum of $I_{\max} \in \{1; 2; \dots; 200\}$ iterations with a stopping threshold of $\epsilon = 0$ and parahermitian matrix truncation parameter of $\mu = 10^{-12}$. Parahermitian matrix truncation is executed at the end of each algorithm iteration, and immediately following CRST, but

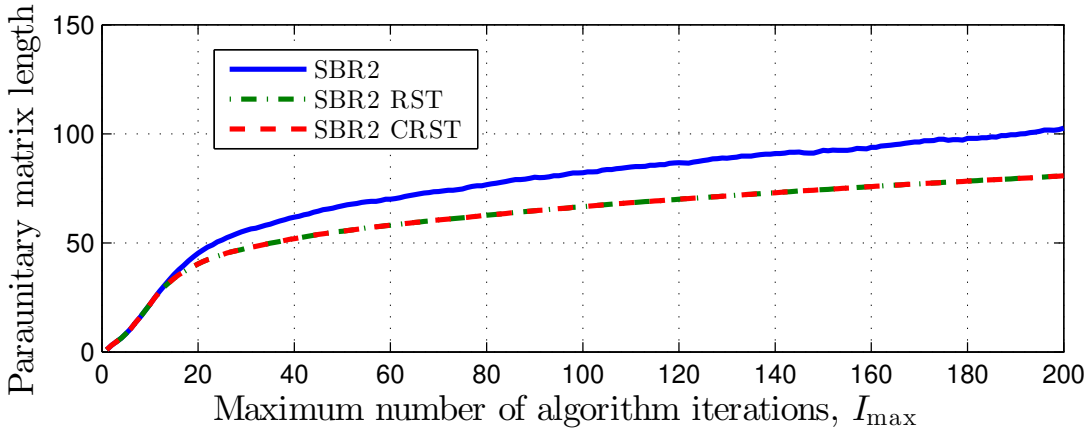

Figure 3.24: Paraunitary matrix length versus maximum number of algorithm iterations for SBR2 with paraunitary matrix truncation after algorithm completion.

paraunitary matrix truncation is only executed after $I_{\max}$ iterations have passed, i.e., after completion of each instance of SBR2. Following $I_{\max}$ iterations of each implementation, the reconstruction error and paraunitarity error metrics defined in Section A.3 are recorded alongside the lengths of $\boldsymbol{F}(z)$ and $\boldsymbol{D}(z)$.

Paraunitary Matrix Length

The ensemble-averaged output paraunitary matrix lengths were calculated for each implementation of SBR2, and can be seen plotted against $I_{\max}$ in Figure 3.24. As expected, the CRST method performs identical paraunitary matrix truncation to the RST method in this scenario. Both CRST and RST offer superior truncation to the traditional truncation method implemented by default in SBR2.

Decomposition MSE

The ensemble-averaged decomposition MSEs were calculated for each implementation of SBR2, and can be seen plotted against $I_{\max}$ in Figure 3.25. As described in Section 3.8.1, the RST method introduces decomposition error if the parahermitian matrix is not diagonal. This is the case for a low number of algorithm iterations, but as the maximum number of allowed iterations increases, the parahermitian matrix output by SBR2 is more diagonal, and the error introduced by RST decreases slightly. While the

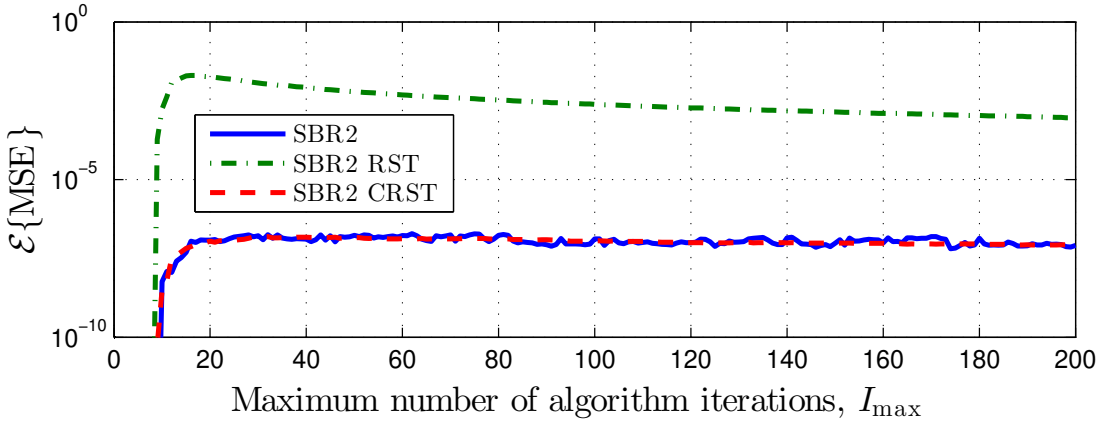

Figure 3.25: MSE versus maximum number of algorithm iterations for SBR2 with paraunitary matrix truncation after algorithm completion.

CRST method performs identical paraunitary matrix truncation to the RST method, by 'compensating' the delays applied to the paraunitary matrix, a level of decomposition MSE similar to that of the traditional truncation method is obtained.

Paraunitarity Error

The ensemble-averaged paraunitarity errors were calculated for each implementation of SBR2, and can be seen plotted against $I_{\max}$ in Figure 3.26. As the CRST method performs identical paraunitary matrix truncation to the RST method, the resulting paraunitarity error is the same. The traditional truncation method implemented by default in SBR2 offers slightly superior paraunitarity error at the cost of higher paraunitary matrix order.

Parahermitian Matrix Length

The ensemble-averaged output parahermitian matrix lengths were calculated for each implementation of SBR2, and can be seen plotted against $I_{\max}$ in Figure 3.27. Surprisingly, despite applying delays to the parahermitian matrix to counteract the delays applied to the paraunitary matrix, the CRST method is able to reduce output parahermitian matrix length. Combined with the results above, this outcome is significant, as the proposed CRST approach seems to be able to reduce paraunitary and parahermi-

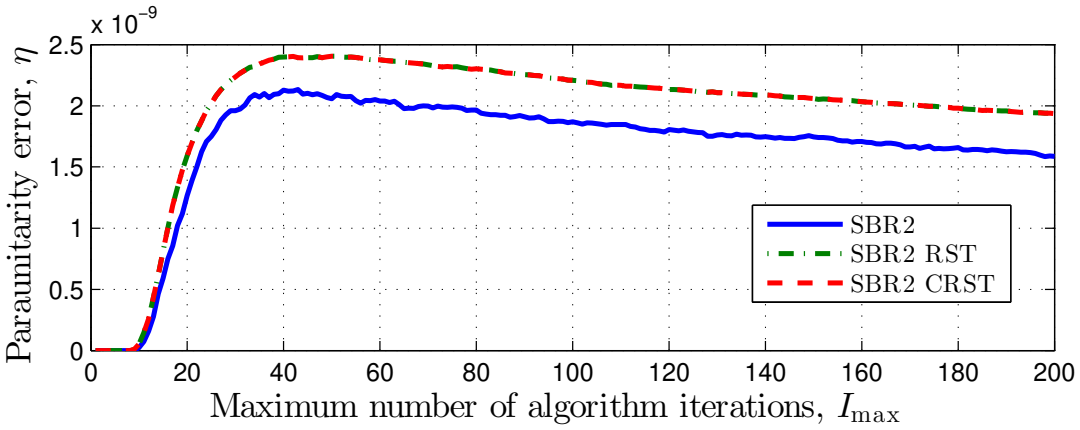

Figure 3.26: Paraunitarity error η versus maximum number of algorithm iterations for SBR2 with paraunitary matrix truncation after algorithm completion.

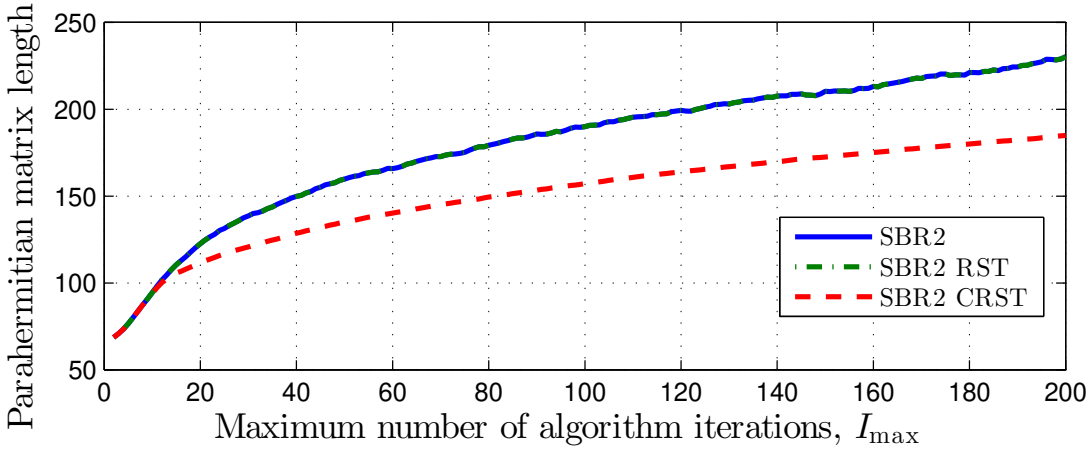

Figure 3.27: Parahermitian matrix length versus maximum number of algorithm iterations for SBR2 with paraunitary matrix truncation after algorithm completion.

tian matrix length with only a minor impact on decomposition MSE and paraunitarity error. Note, the traditional and RST methods of paraunitary matrix truncation do not have any impact on parahermitian matrix length.

3.8.3 Truncating at Each Algorithm Iteration

Simulation Scenario

The simulations below are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{M \times M}$ obtained from the randomised source model in Appendix B [45] with

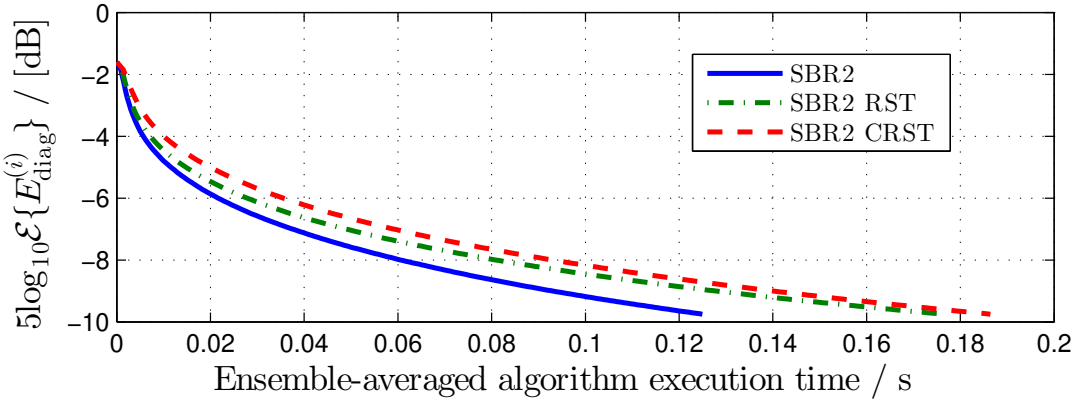

Figure 3.28: Diagonalisation metric versus algorithm execution time for SBR2 with paraunitary matrix truncation at each iteration.

$M = 5$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$.

Three instances of SBR2 are implemented; the first uses the standard SBR2 paraunitary matrix truncation strategy of Appendix C with threshold $\mu_{\text{trad}} = 10^{-6}$, the second uses RST with $\mu_{\text{RST}} = 10^{-6}$, and the third uses CRST with $\mu_{\text{CRST}} = 10^{-6}$. Each instance of SBR2 is executed for a maximum of $I_{\max} = 200$ iterations with a stopping threshold of $\epsilon = 0$ and parahermitian matrix truncation parameter of $\mu = 10^{-12}$. Both parahermitian and paraunitary matrix truncation are implemented at the end of each algorithm iteration. At each iteration of the implementations, the reconstruction error, paraunitarity error, and diagonalisation metrics defined in Section A.3 are recorded alongside the lengths of the internal polynomial matrices and the elapsed execution time.

Diagonalisation

The ensemble-averaged diagonalisation was calculated for each implementation of SBR2, and can be seen plotted against ensemble-averaged algorithm execution time in Figure 3.28. Here it can be seen that the higher complexity associated with the proposed CRST strategy translates to a slightly slower diagonalisation than observed for SBR2 using the RST strategy. Both CRST and RST decrease diagonalisation speed relative to an instance of SBR2 utilising standard paraunitary matrix truncation.

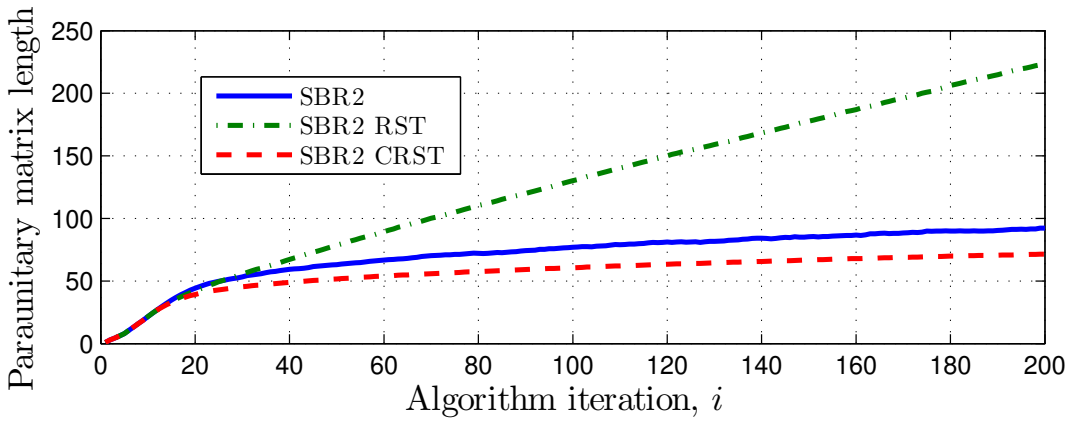

Figure 3.29: Paraunitary matrix length versus algorithm iteration for SBR2 with paraunitary matrix truncation at each iteration.

Paraunitary Matrix Length

The ensemble-averaged output paraunitary matrix lengths were calculated for each implementation of SBR2, and can be seen plotted against algorithm iteration in Figure 3.29. As an unexpected side-effect of the error RST introduces into the decomposition, the paraunitary matrix is not well truncated at each iteration, and instead increases in length with algorithm iteration in an approximately linear relationship. For this scenario, it can be concluded that RST does not provide adequate truncation of the paraunitary matrix. In the same figure, it can be observed that CRST obtains shorter paraunitary matrices and therefore offers superior truncation to the traditional truncation method implemented by default in SBR2.

Decomposition MSE

The ensemble-averaged decomposition MSEs were calculated for each implementation of SBR2, and can be seen plotted against algorithm iteration in Figure 3.30. The RST method again introduces significant decomposition error, as the parahermitian matrix is not diagonal; indeed, the error introduced is even higher than in Figure 3.25. This error stays relatively constant with increasing algorithm iteration, and is too high for an accurate reconstruction of the original parahermitian matrix to be obtained. The CRST method again obtains a similar level of decomposition MSE to that of the traditional truncation method.

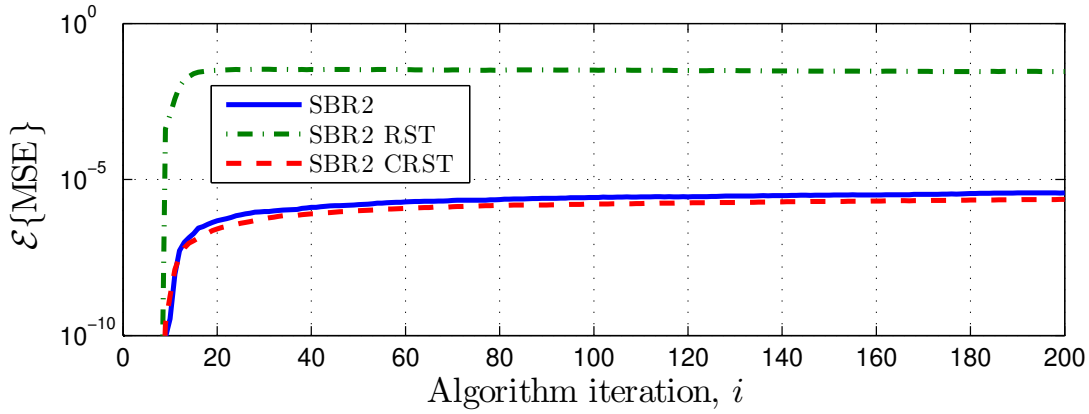

Figure 3.30: MSE versus algorithm iteration for SBR2 with paraunitary matrix truncation at each iteration.

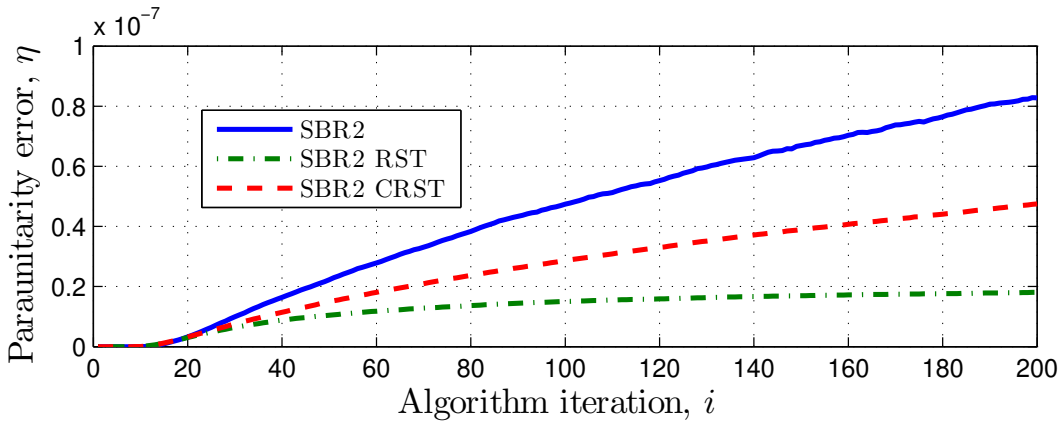

Figure 3.31: Paraunitarity error η versus algorithm iteration for SBR2 with paraunitary matrix truncation at each iteration.

Paraunitarity Error

The ensemble-averaged paraunitarity errors were calculated for each implementation of SBR2, and can be seen plotted against algorithm iteration in Figure 3.31. Interestingly, despite offering poorer truncation and decomposition MSE performance, the RST method has the lowest paraunitarity error. This is likely a result of the RST method's inability to adequately truncate — and therefore introduce error into — the paraunitary matrix. The CRST method offers superior paraunitarity error performance to the traditional method while obtaining lower order paraunitary matrices.

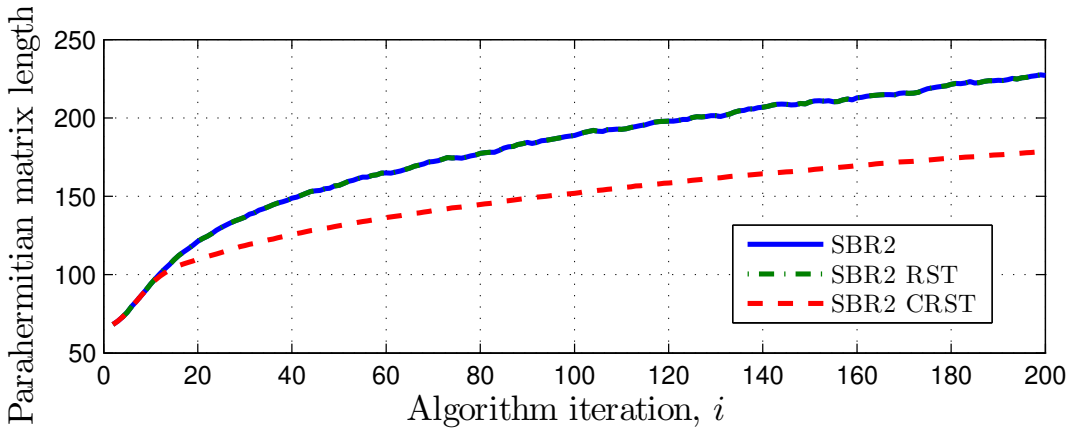

Figure 3.32: Parahermitian matrix length versus algorithm iteration for SBR2 with paraunitary matrix truncation at each iteration.

Parahermitian Matrix Length

The ensemble-averaged output parahermitian matrix lengths were calculated for each implementation of SBR2, and can be seen plotted against algorithm iteration in Figure 3.32. As in Figure 3.27, the CRST method is able to reduce parahermitian matrix length. The proposed CRST approach reliably reduces paraunitarity error and paraunitary and parahermitian matrix length with only a minor impact on decomposition MSE and diagonalisation speed. Note, the traditional and RST methods of paraunitary matrix truncation again have no impact on parahermitian matrix length.

3.8.4 Summary

A novel variation of row-shift truncation (RST) titled compensated row-shift truncation (CRST) has been introduced. When used on the paraunitary matrices output by the SBR2 PEVD algorithm, it has been shown that this method is able to outperform traditional truncation and RST by reducing paraunitary matrix order without significantly affecting paraunitarity error or decomposition error for non-diagonal $\boldsymbol{D}(z)$. Furthermore, simulation results have demonstrated that unlike RST, CRST can be implemented at each iteration of the SBR2 algorithm to reduce paraunitary matrix length and paraunitarity error relative to traditional truncation, with only a slight decrease to diagonalisation speed. An unforeseen advantage of the CRST method is its ability

to reduce the length of the polynomial eigenvalues in the parahermitian matrix output by SBR2. Given the results in this section, it can be concluded that CRST is a viable alternative to traditional paraunitary matrix truncation methods.

3.9 Conclusions

This chapter has detailed a number of novel methods to lower the computational cost of existing iterative algorithms related to the PEVD. The first section discussed a number of novel, significant ways to increase the algorithmic efficiency of iterative PEVD algorithms and polynomial matrix implementations without impacting accuracy. Most significantly, an efficient, two-dimensional approach to computing the product of a matrix and polynomial matrix was introduced. When utilising this approach, the SMD algorithm was identified as the most computationally efficient, state-of-the-art iterative PEVD algorithm.

The symmetry in the parahermitian matrix was then exploited to reduce computational costs and memory requirements when calculating a PEVD. The reduced, half-matrix representation — which can be utilised in any PEVD algorithm — records only the causal part of a parahermitian matrix, and can be used to produce the same accuracy of decomposition as a standard, full-matrix representation.

The subsequent section detailed a series of steps to reduce the complexity of an existing cyclic-by-row SMD algorithm, and used simulations to demonstrate that the obtained reduction in complexity translates to an increase in the diagonalisation speed of the algorithm — with minimal impact on its convergence. While the cyclic-by-row SMD algorithm's avoidance of a direct EVD computation may be useful in applications, applying an approximate EVD at each iteration is less effective than a full EVD [49]. If an efficient method for computing the EVD is present, the SMD algorithm will offer superior convergence speed.

Further research demonstrated that a reduction in the search space can slightly improve the diagonalisation performance of an existing PEVD algorithm without significantly impacting convergence or the rate of growth of the parahermitian or paraunitary matrices. The proposed technique can be extended to any iterative PEVD algorithm;

however, given the relatively small performance gains achieved through the implementation of this method, it is likely that the additional effort required to estimate the search space reduction negates any gains made.

A more promising restricted update approach to the PEVD was then introduced. A version of the SMD algorithm modified to incorporate this method was found to produce the same quality of decomposition as SMD, but with decreased computational complexity and execution time requirements. Importantly, the restricted update approach can be extended to any iterative PEVD algorithm to increase diagonalisation speed.

With the knowledge that successive applications of the polynomial matrix QR decomposition can generate a PEVD, improvements were then made to an existing polynomial matrix QR decomposition algorithm. The resulting multiple shift approach demonstrated superior triangularisation speed to the original method; however, existing PEVD algorithms are likely to be more efficient than using multiple instances of the PQRD.

Finally, a novel compensated row-shift truncation approach was introduced as an alternative to existing truncation methods. When used on the paraunitary matrices output by the SBR2 PEVD algorithm, it was demonstrated that this approach is able to outperform existing truncation strategies by reducing paraunitary matrix order without significantly affecting paraunitarity error or decomposition error for non-diagonal $\boldsymbol{D}(z)$. Furthermore, simulation results demonstrated that the developed method can potentially be used to reduce the lengths of the polynomial matrices internal to PEVD algorithms with only a slight decrease to diagonalisation speed.

While each of the methods discussed in this chapter decrease the implementation costs of various PEVD approaches, the complexity of the algorithms grows rapidly with the spatial dimensions of the parahermitian matrix, such that even the improved PEVD algorithms are not well-suited for applications involving large broadband arrays. The next chapter addresses this problem by taking additional steps to convert the sequential form of existing PEVD algorithms to a partially parallelisable divide-and-conquer architecture.

Chapter 4

Divide-and-Conquer Strategy for PEVD Algorithms

4.1 Introduction

While the contents of Chapter 3 addressed computational advances applicable to a number of the existing iterative PEVD algorithms discussed in Section 2.3.2, the complexities of the improved algorithms are still highly dependent on the spatial dimension of the parahermitian matrix. As discussed in the previous chapter, parahermitian matrices with a large spatial dimension M — which dictates the number of rows and columns in the matrix — can occur in scenarios involving a large number of broadband sensors. In addition, due to the current reliance of PEVD algorithms on sequential processing — i.e., requiring iteration i to be fully complete prior to iteration $(i+1)$ — the explosion in the availability of parallel processing capable hardware [97–101] has not yet been fully exploited during the implementation of PEVD algorithms. This chapter therefore introduces a novel methodology for the PEVD that takes inspiration from existing so-called divide-and-conquer (DaC) solutions to eigenproblems to 'divide' the PEVD of a parahermitian matrix with large spatial dimension into a number of components involving smaller spatial dimensions, before 'conquering' each element separately — and potentially simultaneously.

Below, Section 4.2 provides a brief introduction to DaC methods and their applica-

tion to eigenproblems, before Section 4.3 describes how DaC strategies can be extended to the PEVD by incorporating a transformation of the input parahermitian matrix to block diagonal form. Section 4.4 then details a novel algorithm, named sequential matrix segmentation (SMS), capable of carrying out such a transformation. Based on the framework of Section 4.3, a further novel algorithm in Section 4.5 utilises SMS to enforce the transformation of a parahermitian matrix with large spatial dimension to block diagonal form, before employing an existing PEVD algorithm to diagonalise each block independently. This algorithm is suitably named divide-and-conquer SMD (DC-SMD), owing to its use of the SMD algorithm [45] for diagonalisation purposes. In a final effort to produce the most computationally efficient PEVD algorithm, Section 4.6 incorporates several of the computational schemes discussed in Chapter 3 within DC-SMD and introduces parallelisation to the 'conquer' stage of the resulting algorithm, which is named parallel-SMD (PSMD). Broadband source model and angle of arrival estimation simulation results are used to compare the performances of the developed DaC approaches with existing PEVD algorithms in Section 4.7, and conclusions for this chapter are provided in Section 4.8.

Elements of the work in this chapter can be found published in the proceedings of the 2017 IEEE International Workshop on Signal Processing Systems [64], and the 2017 IEEE Sensor Signal Processing for Defence Conference [20, 71]. A further paper has been submitted to IEEE Transactions on Circuits and Systems I [72].

4.2 Divide-and-Conquer as a Methodology

In the realm of algorithm design, DaC strategies typically decompose a large problem to be solved into a number of smaller, independent subproblems; the solutions to these subproblems are then combined to form an overall solution to the original problem [102]. Such approaches can be extended to any number of large scale signal processing problems, and can be found in efficient sorting [103] and fast Fourier transform algorithms [93], for example. Specifically in relation to eigenproblems, research in [104] introduces a DaC method for the symmetric tridiagonal eigenproblem, while [105] extends this approach to create a fully parallel algorithm for the symmet-

ric eigenvalue problem. Furthermore, in [106], a low-complexity version of the DaC approach in [104] is presented.

While the PEVD does not explicitly involve tridiagonal matrices, there is clearly precedent for the utilisation of DaC algorithms to obtain eigenvalues. The computational complexity of current PEVD algorithms is strongly influenced by the spatial dimensions of the parahermitian matrix; devising a similar strategy to above that reduces the dimensionality of the problem is therefore of great interest. Furthermore, if such a strategy can be developed, the parallelised and low-complexity algorithms of [105] and [106], respectively, suggest that a low-complexity, parallelised DaC PEVD algorithm may be obtainable.

4.3 Extending the Divide-and-Conquer Methodology to the PEVD

4.3.1 Problem Formulation

The existing iterative PEVD algorithms discussed in Section 2.3.2 all take a purely sequential approach. That is, the computations in iteration $i+1$ of existing algorithms cannot be commenced before finalising the computations in iteration i. This is problematic for the development of a parallelised PEVD algorithm. An implicit assumption within these algorithms is that the entirety of the parahermitian matrix must be updated at each iteration to avoid the introduction of error to the decomposition; however, this is not always the case. For example, consider the parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$, which can be thought of as four matrices arranged as shown:

$$\boldsymbol{R}(z) = \begin{bmatrix} \boldsymbol{R}_{11}(z) & \boldsymbol{R}_{12}(z) \\ \boldsymbol{R}_{21}(z) & \boldsymbol{R}_{22}(z) \end{bmatrix} . \tag{4.1}$$

Here, $\boldsymbol{R}(z)$ is represented using parahermitian submatrices $\boldsymbol{R}_{11}(z) : \mathbb{C} \to \mathbb{C}^{(M-P)\times(M-P)}$ and $\boldsymbol{R}_{22}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$, and non-parahermitian submatrices $\boldsymbol{R}_{21}(z) : \mathbb{C} \to \mathbb{C}^{P\times(M-P)}$ and $\boldsymbol{R}_{12}(z) : \mathbb{C} \to \mathbb{C}^{(M-P)\times P}$, with $\boldsymbol{R}_{21}(z) = \tilde{\boldsymbol{R}}_{12}(z)$ by definition. If $\boldsymbol{R}_{21}(z) = \boldsymbol{0}$, $\boldsymbol{R}(z)$ is block diagonal, and submatrices $\boldsymbol{R}_{11}(z)$ and $\boldsymbol{R}_{22}(z)$ can each ex-

perience paraunitary operations without influencing the other; i.e., $\boldsymbol{R}_{11}(z)$ and $\boldsymbol{R}_{22}(z)$ are independent parahermitian matrices. In this scenario, the PEVD of $\boldsymbol{R}(z)$ can be computed by inputting $\boldsymbol{R}_{11}(z)$ and $\boldsymbol{R}_{22}(z)$ to two separate instances of a PEVD algorithm. If $\boldsymbol{F}_{11}(z)$ and $\boldsymbol{D}_{11}(z)$, and $\boldsymbol{F}_{22}(z)$ and $\boldsymbol{D}_{22}(z)$ are the polynomial eigenvectors and eigenvalues of $\boldsymbol{R}_{11}(z)$ and $\boldsymbol{R}_{22}(z)$ obtained from two PEVDs, respectively,

$$\boldsymbol{F}(z) = \begin{bmatrix} \boldsymbol{F}_{11}(z) & \mathbf{0} \\ \mathbf{0} & \boldsymbol{F}_{22}(z) \end{bmatrix} \tag{4.2}$$

and

$$\boldsymbol{D}(z) = \begin{bmatrix} \boldsymbol{D}_{11}(z) & \mathbf{0} \\ \mathbf{0} & \boldsymbol{D}_{22}(z) \end{bmatrix} \tag{4.3}$$

are polynomial eigenvectors and eigenvalues of $\boldsymbol{R}(z)$. Analogously to the divide-and-conquer methodologies discussed previously, the natural 'divided' structure of $\boldsymbol{R}(z)$ has facilitated the independent 'conquering' of the two constituent matrices in this scenario.

As an extension of the above, consider the case that $\boldsymbol{R}_{11}(z)$ is itself block diagonal, and can be represented by two independent parahermitian matrices of even smaller spatial dimension. Matrix $\boldsymbol{R}(z)$ is now 'divided' into three components, which can again be 'conquered' independently.

If $\mathbf{R}[\tau]$ is a space-time covariance matrix computed according to (2.2), the resulting $\boldsymbol{R}(z)$ can be block diagonal if two or more groups of time series in $\mathbf{x}[n] \in \mathbb{C}^M$ exist that have zero correlation with each other for all τ. Of course, naturally obtaining a block diagonal $\boldsymbol{R}(z)$ in an application scenario is unlikely. It is therefore important to find some means to enforce such a block diagonal structure for a general case parahermitian matrix $\boldsymbol{R}(z)$ without introducing significant error to the decomposition. This approach would ideally use paraunitary operations, as they preserve energy and are therefore reversible. Furthermore, if the ultimate goal is a PEVD of $\boldsymbol{R}(z)$, the overall similarity transformation to achieve diagonalisation should be paraunitary, and therefore has to consist of paraunitary operations.

If they exist, the sequence of paraunitary operations that transform $\boldsymbol{R}(z)$ into a block diagonal $\boldsymbol{B}(z)$ — composed of N blocks — can be combined into a single parau-

nitary matrix $\boldsymbol{T}(z)$ such that

$$\boldsymbol{B}(z) = \boldsymbol{T}(z)\boldsymbol{R}(z)\tilde{\boldsymbol{T}}(z) \approx \begin{bmatrix} \boldsymbol{B}_{11}(z) & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{22}(z) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \boldsymbol{B}_{NN}(z) \end{bmatrix} . \tag{4.4}$$

As in equation (2.4), (4.4) has only approximate equality, as it is likely that the decomposition to obtain a block diagonal matrix from a finite order polynomial matrix is not of finite order. However, as found with the PEVD [9], it is assumed that the approximation error will become arbitrarily small if the order of the approximation is selected to be sufficiently large. As above, the PEVDs of each $\boldsymbol{B}_{nn}(z)$, $n = 1 \dots N$, can be conducted in parallel to produce polynomial eigenvectors

$$\hat{\boldsymbol{F}}(z) = \begin{bmatrix} \hat{\boldsymbol{F}}_{11}(z) & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \hat{\boldsymbol{F}}_{22}(z) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \hat{\boldsymbol{F}}_{NN}(z) \end{bmatrix} \tag{4.5}$$

and polynomial eigenvalues

$$\boldsymbol{D}(z) = \begin{bmatrix} \boldsymbol{D}_{11}(z) & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{D}_{22}(z) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \boldsymbol{D}_{NN}(z) \end{bmatrix} , \tag{4.6}$$

such that $\boldsymbol{B}_{nn}(z) \approx \tilde{\hat{\boldsymbol{F}}}_{nn}(z)\boldsymbol{D}_{nn}(z)\hat{\boldsymbol{F}}_{nn}(z)$ and

$$\boldsymbol{R}(z) \approx \tilde{\boldsymbol{T}}(z)\tilde{\hat{\boldsymbol{F}}}(z)\boldsymbol{D}(z)\hat{\boldsymbol{F}}(z)\boldsymbol{T}(z) = \tilde{\boldsymbol{F}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z) , \tag{4.7}$$

where $\boldsymbol{F}(z) = \hat{\boldsymbol{F}}(z)\boldsymbol{T}(z)$. Matrix $\boldsymbol{F}(z)$, which is the product of two paraunitary matrices and is therefore paraunitary by construction, decomposes $\boldsymbol{R}(z)$ to a diagonal

matrix $\boldsymbol{D}(z)$ containing polynomial eigenvalues, and can therefore be considered to contain polynomial eigenvectors.

4.3.2 Block Diagonalising a Parahermitian Matrix

The paraunitary matrix $\boldsymbol{T}(z)$ that 'divides' a parahermitian matrix into block diagonal form cannot be immediately ascertained from observation of $\boldsymbol{R}(z)$. However, using an approach akin to iterative PEVD algorithms — which seek to iteratively diagonalise a parahermitian matrix via paraunitary operations — an algorithm can be developed that uses paraunitary operations to iteratively block diagonalise a parahermitian matrix. One approach, which is described below, is to block diagonalise the parahermitian matrix in stages.

For some user-specified P, with $P < M$, the first stage of the iterative process to block diagonalise a parahermitian matrix $\boldsymbol{B}^{(0)}(z) = \boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ seeks to obtain a paraunitary matrix $\boldsymbol{T}^{(1)}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ such that

$$\boldsymbol{B}^{(1)}(z) = \boldsymbol{T}^{(1)}(z)\boldsymbol{B}^{(0)}(z)\tilde{\boldsymbol{T}}^{(1)}(z) \approx \begin{bmatrix} \boldsymbol{B}_{11}^{(1)}(z) & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{22}^{(1)}(z) \end{bmatrix} \tag{4.8}$$

is block diagonal with two parahermitian submatrices $\boldsymbol{B}_{11}^{(1)}(z) : \mathbb{C} \to \mathbb{C}^{(M-P)\times(M-P)}$ and $\boldsymbol{B}_{22}^{(1)}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$. Note that, as in equation (4.4), this transformation is unlikely to be achievable using finite order polynomial matrices; thus, (4.8) has only approximate equality.

Provided that $P < (M - P)$, the second stage seeks to obtain a further paraunitary matrix

$$\boldsymbol{T}^{(2)}(z) = \begin{bmatrix} \hat{\boldsymbol{T}}^{(2)}(z) & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_P \end{bmatrix}, \tag{4.9}$$

that operates on $\boldsymbol{B}_{11}^{(1)}(z)$ such that

$$\boldsymbol{B}^{(2)}(z) = \boldsymbol{T}^{(2)}(z)\boldsymbol{B}^{(1)}(z)\tilde{\boldsymbol{T}}^{(2)}(z) \approx \begin{bmatrix} \boldsymbol{B}_{11}^{(2)}(z) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{22}^{(2)}(z) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{B}_{22}^{(1)}(z) \end{bmatrix} \tag{4.10}$$

is block diagonal with two parahermitian submatrices $\boldsymbol{B}_{11}^{(2)}(z) : \mathbb{C} \to \mathbb{C}^{(M-2P)\times(M-2P)}$ and $\boldsymbol{B}_{22}^{(2)}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$, plus parahermitian submatrix $\boldsymbol{B}_{22}^{(1)}(z)$ from before. Here, $\hat{\boldsymbol{T}}^{(2)}(z) : \mathbb{C} \to \mathbb{C}^{(M-P)\times(M-P)}$ has transformed $\boldsymbol{B}_{11}^{(1)}(z)$ into block diagonal form, while $\boldsymbol{B}_{22}^{(1)}(z)$ is influenced by identity matrix $\mathbf{I}_P$ and is therefore unaffected.

The above process repeats a total of β times, i.e., until $P \geq (M - \beta P)$ and some $\boldsymbol{B}^{(\beta)}(z)$ has been obtained with parahermitian submatrices $\boldsymbol{B}_{11}^{(\beta)}(z) : \mathbb{C} \to \mathbb{C}^{(M-\beta P)\times(M-\beta P)}$ and $\boldsymbol{B}_{22}^{(\beta)}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$, alongside a 'dividing' matrix

$$\boldsymbol{T}^{(\beta)}(z) = \begin{bmatrix} \hat{\boldsymbol{T}}^{(\beta)}(z) & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{(\beta-1)P} \end{bmatrix} . \tag{4.11}$$

The final block diagonal parahermitian matrix can be written as

$$\boldsymbol{B}(z) = \boldsymbol{T}(z)\boldsymbol{R}(z)\tilde{\boldsymbol{T}}(z) \approx \begin{bmatrix} \boldsymbol{B}_{11}^{(\beta)}(z) & \mathbf{0} & \dots & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{22}^{(\beta)}(z) & & & \vdots \\ \vdots & & \boldsymbol{B}_{22}^{(\beta-1)}(z) & & \vdots \\ \vdots & & & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \dots & \boldsymbol{B}_{22}^{(1)}(z) \end{bmatrix} . \tag{4.12}$$

Note that matrices $\boldsymbol{B}_{22}^{(b)}(z)$, $b = 1 \dots \beta$, have spatial dimensions of $P \times P$, while $\boldsymbol{B}_{11}^{(\beta)}(z)$ is of spatial dimension $(M - \beta P) \times (M - \beta P)$. The overall paraunitary 'dividing' matrix $\boldsymbol{T}(z)$ that transforms $\boldsymbol{R}(z)$ to block diagonal form can be found by computing the product of all $\boldsymbol{T}^{(b)}(z)$, $b = 1 \dots \beta$, according to

$$\begin{aligned} \boldsymbol{T}(z) &= \boldsymbol{T}^{(\beta)}(z)\boldsymbol{T}^{(\beta-1)}(z) \cdots \boldsymbol{T}^{(1)}(z) \\ &= \begin{bmatrix} \hat{\boldsymbol{T}}^{(\beta)}(z) & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{(\beta-1)P} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{T}}^{(\beta-1)}(z) & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{(\beta-2)P} \end{bmatrix} \cdots \boldsymbol{T}^{(1)}(z) , \end{aligned} \tag{4.13}$$

where the insertion of identity matrices has ensured an overall spatial dimension of $M \times M$.

Once $\boldsymbol{R}(z)$ has been transformed to block diagonal form, each parahermitian submatrix can be diagonalised independently through the use of a PEVD algorithm.

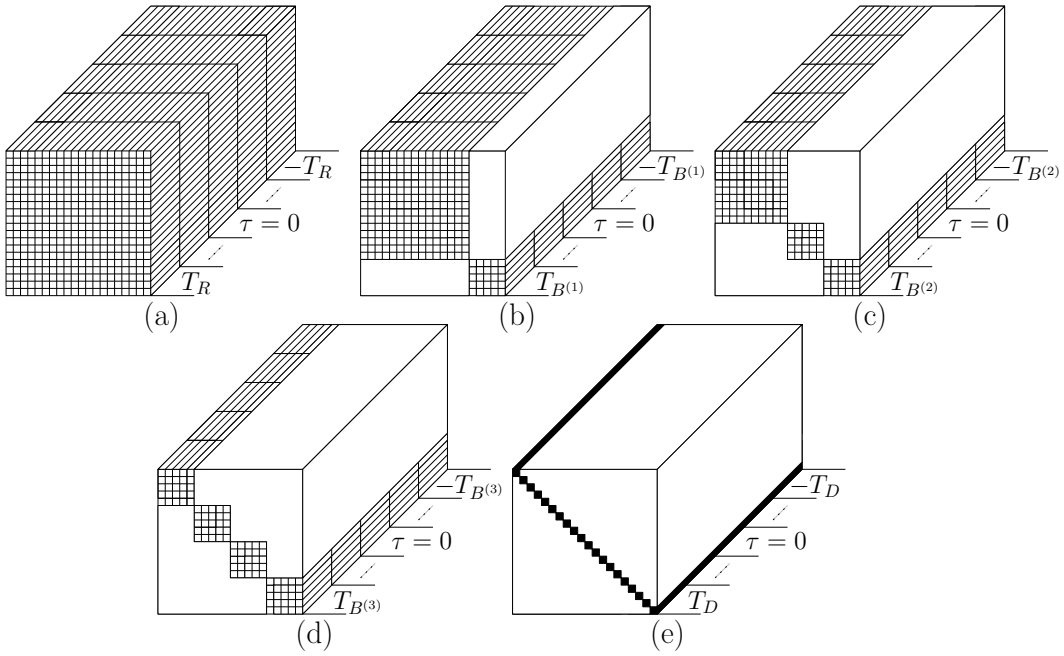

Figure 4.1: Steps taken to block diagonalise a parahermitian matrix with $M = 20$ and $P = 5$. (a) Original matrix $\mathbf{R}[\tau] \in \mathbb{C}^{20\times20}$; (b) block diagonal $\mathbf{B}^{(1)}[\tau]$ with parahermitian submatrices $\mathbf{B}^{(1)}_{11}[\tau] \in \mathbb{C}^{15\times15}$ and $\mathbf{B}^{(1)}_{22}[\tau] \in \mathbb{C}^{5\times5}$; (c) block diagonal $\mathbf{B}^{(2)}[\tau]$ with parahermitian submatrices $\mathbf{B}^{(2)}_{11}[\tau] \in \mathbb{C}^{10\times10}$, $\mathbf{B}^{(2)}_{22}[\tau] \in \mathbb{C}^{5\times5}$, and $\mathbf{B}^{(1)}_{22}[\tau] \in \mathbb{C}^{5\times5}$; (d) block diagonal $\mathbf{B}^{(3)}[\tau]$ with parahermitian submatrices $\mathbf{B}^{(3)}_{11}[\tau] \in \mathbb{C}^{5\times5}$, $\mathbf{B}^{(3)}_{22}[\tau] \in \mathbb{C}^{5\times5}$, $\mathbf{B}^{(2)}_{22}[\tau] \in \mathbb{C}^{5\times5}$, and $\mathbf{B}^{(1)}_{22}[\tau] \in \mathbb{C}^{5\times5}$; and (e) diagonal $\mathbf{D}[\tau]$, which contains polynomial eigenvalues obtained from independent diagonalisation of each submatrix. T_R, $T_{B^{(1)}}$, $T_{B^{(2)}}$, $T_{B^{(3)}}$, and T_D are the maximum lags for matrices $\mathbf{R}[\tau]$, $\mathbf{B}^{(1)}[\tau]$, $\mathbf{B}^{(2)}[\tau]$, $\mathbf{B}^{(3)}[\tau]$, and $\mathbf{D}[\tau]$, respectively. Blank regions indicate regions containing only zeroes.

The diagram of Figure 4.1 illustrates the use of the above iterative block diagonalisation process for the diagonalisation of a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{20\times20}$ for $P = 5$.

4.4 Sequential Matrix Segmentation Algorithm

The sequential matrix segmentation (SMS) algorithm is a novel variant of SMD [45] designed to approximately block diagonalise a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$. The algorithm outputs are a block diagonal matrix $\boldsymbol{B}(z)$ with independent parahermitian submatrices $\boldsymbol{B}_{11}(z) : \mathbb{C} \to \mathbb{C}^{(M-P)\times(M-P)}$ and $\boldsymbol{B}_{22}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$, and a

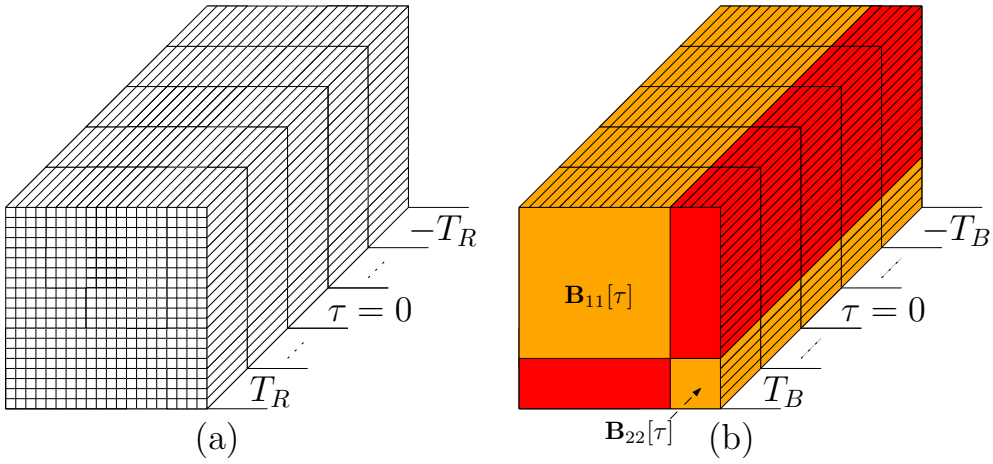

Figure 4.2: (a) Original matrix $\mathbf{R}[\tau] \in \mathbb{C}^{20\times 20}$ with regions to be driven to zero in SMS, and (b) block diagonal result for $P = 5$. T_R and T_B are the maximum lags for the original and block diagonal matrices, respectively.

paraunitary matrix $\boldsymbol{T}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ that satisfies

$$\boldsymbol{B}(z) \approx \boldsymbol{T}(z)\boldsymbol{R}(z)\tilde{\boldsymbol{T}}(z) \; . \tag{4.14}$$

The parameter P, which determines the dimensions of the matrices produced during block diagonalisation, must satisfy $P < M$.

Figure 4.2 illustrates the block diagonalisation process of SMS for $M = 20$ and $P = 5$. Note that this could be the first of three 'division' steps during the block diagonalisation of $\boldsymbol{R}(z)$ as in Figure 4.1(b).

Below, Section 4.4.1 provides an overview of the SMS algorithm, while proof of convergence is given in Section 4.4.2 and a brief derivation of algorithm complexity is supplied in Section 4.4.3. Some results that confirm the SMS algorithm's ability to block diagonalise a parahermitian matrix are shown in Section 4.4.4.

4.4.1 Algorithm Overview

The SMS algorithm is initialised and operates in a similar manner to the SMD algorithm in Section A.1, but with a few key differences. Instead of iteratively shifting single row-column pairs in an effort to diagonalise a parahermitian matrix $\boldsymbol{S}^{(i)}(z)$, SMS iteratively minimises the energy in select regions of $\boldsymbol{S}^{(i)}(z)$ in an attempt to block diagonalise the

matrix over $\hat{I}$ iterations; i.e., for $i = 1 \dots \hat{I}$. If

$$\boldsymbol{S}^{(i)}(z) = \begin{bmatrix} \boldsymbol{S}_{11}^{(i)}(z) & \boldsymbol{S}_{12}^{(i)}(z) \\ \boldsymbol{S}_{21}^{(i)}(z) & \boldsymbol{S}_{22}^{(i)}(z) \end{bmatrix} \tag{4.15}$$

is represented using four polynomial matrices as shown, the regions to experience energy reduction are denoted $\boldsymbol{S}_{12}^{(i)}(z) : \mathbb{C} \to \mathbb{C}^{(M-P)\times P}$ and $\boldsymbol{S}_{21}^{(i)}(z) : \mathbb{C} \to \mathbb{C}^{P\times(M-P)}$, and are located to the top-right and bottom-left of $\boldsymbol{S}^{(i)}(z)$, such that $\boldsymbol{S}_{12}^{(i)}(z) = \tilde{\boldsymbol{S}}_{21}^{(i)}(z)$. These regions are highlighted in red for the example of Figure 4.2. The energy from these regions is iteratively transferred to parahermitian submatrices $\boldsymbol{S}_{11}^{(i)}(z) : \mathbb{C} \to \mathbb{C}^{(M-P)\times(M-P)}$ and $\boldsymbol{S}_{22}^{(i)}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$ on the diagonal of $\boldsymbol{S}^{(i)}(z)$; this results in $\boldsymbol{S}^{(i)}(z)$ more closely approximating a block diagonal matrix as i increases. That is, in an ideal scenario, SMS enables the following:

$$\lim_{i\to\infty} \sum_{\tau} \left\| \mathbf{S}_{12}^{(i)}[\tau] \right\|_{\mathrm{F}}^2 = \lim_{i\to\infty} \sum_{\tau} \left\| \mathbf{S}_{21}^{(i)}[\tau] \right\|_{\mathrm{F}}^2 = 0 \,; \tag{4.16}$$

however, in reality this may require an infeasible number of iterations. A discussion on SMS convergence in practice is provided in Section 4.4.2.

Alongside the parameter P, each instance of SMS is provided with a parameter $I_{\max}$ — which defines the maximum possible number of algorithm iterations — a stopping threshold κ, and truncation parameters μ and μ_t.

To achieve matrix block diagonalisation, the SMS algorithm uses a series of elementary paraunitary operations to iteratively minimise the energy in $\boldsymbol{S}_{12}^{(i)}(z)$ and $\boldsymbol{S}_{21}^{(i)}(z)$. Each elementary paraunitary operation consists of two steps: first, a delay step is used to move the bottom-left and top-right regions with the largest energy to lag zero; then, an EVD diagonalises the lag zero matrix, transferring the shifted energy onto the diagonal.

Upon initialisation, the algorithm diagonalises the lag zero coefficient matrix $\mathbf{R}[0]$ by means of its modal matrix $\mathbf{Q}^{(0)}$, which is obtained from the ordered EVD of $\mathbf{R}[0]$, such that $\boldsymbol{S}^{(0)}(z) = \mathbf{Q}^{(0)}\boldsymbol{R}(z)\mathbf{Q}^{(0)\mathrm{H}}$. The unitary $\mathbf{Q}^{(0)}$ is applied to all coefficient matrices $\mathbf{R}[\tau]\ \forall\ \tau$, and initialises $\boldsymbol{H}^{(0)}(z) = \mathbf{Q}^{(0)}$.

The SMS algorithm then computes

$$\begin{aligned}\boldsymbol{S}^{(i)}(z) &= \boldsymbol{U}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{U}}^{(i)}(z)\\ \boldsymbol{H}^{(i)}(z) &= \boldsymbol{U}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)\end{aligned} \tag{4.17}$$

in the ith step, $i = 1, 2, \ldots \hat{I}$, in which

$$\boldsymbol{U}^{(i)}(z) = \mathbf{Q}^{(i)}\boldsymbol{\Lambda}^{(i)}(z)\ . \tag{4.18}$$

The product in (4.18) consists of a paraunitary delay matrix

$$\boldsymbol{\Lambda}^{(i)}(z) = \mathrm{diag}\{\underbrace{1\ \ldots\ 1}_{M-P}\ \underbrace{z^{\tau^{(i)}}\ \ldots\ z^{\tau^{(i)}}}_{P}\} \tag{4.19}$$

and a unitary matrix $\mathbf{Q}^{(i)}$, such that $\boldsymbol{U}^{(i)}(z)$ in (4.18) is paraunitary. For subsequent discussion, it is convenient to define intermediate variables $\boldsymbol{S}^{(i)\prime}(z)$ and $\boldsymbol{H}^{(i)\prime}(z)$ where

$$\begin{aligned}\boldsymbol{S}^{(i)\prime}(z) &= \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{\Lambda}}^{(i)}(z)\\ \boldsymbol{H}^{(i)\prime}(z) &= \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)\quad .\end{aligned} \tag{4.20}$$

Matrix $\boldsymbol{\Lambda}^{(i)}(z)$ is selected based on the lag position of the dominant region in the bottom-left of $\boldsymbol{S}^{(i-1)}(z) \bullet\!\!-\!\!\circ\, \mathbf{S}^{(i-1)}[\tau]$, which is identified by

$$\tau^{(i)} = \arg\max_{\tau} \left\| \mathbf{S}_{21}^{(i-1)}[\tau] \right\|_{\mathrm{F}}^2\ , \tag{4.21}$$

where

$$\left\| \mathbf{S}_{21}^{(i-1)}[\tau] \right\|_{\mathrm{F}}^2 = \sum_{m=M-P+1}^{M} \sum_{k=1}^{M-P} \left| s_{m,k}^{(i-1)}[\tau] \right|^2 \quad . \tag{4.22}$$

Here, $s_{m,k}^{(i-1)}[\tau]$ represents the element in the mth row and kth column of the coefficient matrix $\mathbf{S}^{(i-1)}[\tau]$.

The shifting process in (4.20) moves the dominant bottom-left region $\mathbf{S}_{21}^{(i-1)}[\tau^{(i)}]$ and dominant top-right region $\mathbf{S}_{12}^{(i-1)}[-\tau^{(i)}] = \mathbf{S}_{21}^{(i-1)\mathrm{H}}[\tau^{(i)}]$ in $\mathbf{S}^{(i-1)}[\tau]$ into the lag zero coefficient matrix $\mathbf{S}^{(i)\prime}[0]$. The energy in the shifted regions is then transferred onto the

diagonal of $\mathbf{S}^{(i)\prime}[0]$ by a unitary matrix $\mathbf{Q}^{(i)}$ — which diagonalises $\mathbf{S}^{(i)\prime}[0]$ by means of an ordered EVD — in

$$\begin{aligned}\boldsymbol{S}^{(i)}(z) &= \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i)\mathrm{H}} \\ \boldsymbol{H}^{(i)}(z) &= \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime}(z) \quad .\end{aligned} \tag{4.23}$$

The use of an ordered EVD ensures that the eigenvalues in $\mathbf{S}^{(i)}[0]$ — and therefore the polynomial eigenvalues of $\boldsymbol{S}^{(i)}(z)$ — are sorted in descending order of power. This results in spectral majorisation, such that the polynomial eigenvalues of $\boldsymbol{B}_{11}(z)$ are always greater in power that the polynomial eigenvalues of $\boldsymbol{B}_{22}(z)$.

Note that the application of $\boldsymbol{\Lambda}^{(i)}(z)$ via similarity transform shifts the bottom-right $P\times P$ region of $\boldsymbol{S}^{(i-1)}(z)$ in opposite directions, such that this region remains unaffected. An efficient implementation of this similarity transform can therefore exclude this region from shifting operations.

The orders of $\boldsymbol{S}^{(i)}(z)$ and $\boldsymbol{H}^{(i)}(z)$ increase at each iteration; to constrain computational complexity, truncation according to Appendix C is typically required. Parameters μ and μ_t are used for parahermitian and paraunitary matrix truncation, respectively.

After $I_{\max}$ iterations — for a user-defined $I_{\max}$ — or when the energy in $\boldsymbol{S}_{21}^{(i)}(z) = \tilde{\boldsymbol{S}}_{12}^{(i)}(z)$ has been sufficiently minimised such that

$$\left\|\mathbf{S}_{21}^{(I)}[\tau]\right\|_{\mathrm{F}}^2 \leq \kappa \sum_{\tau} \|\mathbf{R}[\tau]\|_{\mathrm{F}}^2 \tag{4.24}$$

at some iteration I — where κ is chosen to be arbitrarily small — the SMS algorithm returns matrices $\boldsymbol{B}(z)$ and $\boldsymbol{T}(z)$. The latter is constructed from the concatenation of the elementary paraunitary matrices:

$$\boldsymbol{T}(z) = \boldsymbol{H}^{(\hat{I})}(z) = \boldsymbol{U}^{(\hat{I})}(z)\cdots\boldsymbol{U}^{(0)}(z) = \prod_{i=0}^{\hat{I}} \boldsymbol{U}^{(\hat{I}-i)}(z) \;, \tag{4.25}$$

where $\hat{I} = \min\{I_{\max}, I\}$. The parahermitian submatrices of $\boldsymbol{B}(z)$, $\boldsymbol{B}_{11}(z)$ and $\boldsymbol{B}_{22}(z)$, are the top-left $(M-P)\times(M-P)$ and the bottom-right $P\times P$ blocks of $\boldsymbol{S}^{(\hat{I})}(z)$,

Input: $\boldsymbol{R}(z)$, $I_{\max}$, P, μ, μ_t, κ
Output: $\boldsymbol{B}(z)$, $\boldsymbol{T}(z)$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise $\mathbf{R}[0] \in \mathbb{C}^{M\times M}$
$\boldsymbol{S}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\boldsymbol{R}(z)\mathbf{Q}^{(0)\mathrm{H}}$; $\boldsymbol{H}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$; $i \leftarrow 0$; stop $\leftarrow 0$
do
  $i \leftarrow i+1$
  Find $\tau^{(i)}$ from (4.21); generate $\boldsymbol{\Lambda}^{(i)}(z)$ from (4.19)
  $\boldsymbol{S}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{\Lambda}}^{(i)}(z)$
  $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
  Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\mathbf{S}^{(i)\prime}[0]$
  $\boldsymbol{S}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$
  $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime}(z)$
  if $i > I_{\max}$ or (4.24) satisfied **then**
    stop $\leftarrow 1$;
  end
  Truncate $\boldsymbol{H}^{(i)}(z)$ using threshold μ_t according to Appendix C
  Truncate $\boldsymbol{S}^{(i)}(z)$ using threshold μ according to Appendix C
while stop $= 0$
$\boldsymbol{T}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$
$\boldsymbol{B}_{11}(z)$ is top-left $(M-P)\times(M-P)$ block of $\boldsymbol{S}^{(i)}(z)$
$\boldsymbol{B}_{22}(z)$ is bottom-right $P \times P$ block of $\boldsymbol{S}^{(i)}(z)$

Algorithm 5: SMS algorithm

$\boldsymbol{S}_{11}^{(\hat{I})}(z)$ and $\boldsymbol{S}_{22}^{(\hat{I})}(z)$, respectively.

Note that while the relationship $\boldsymbol{R}(z) = \tilde{\boldsymbol{T}}(z)\boldsymbol{S}^{(\hat{I})}(z)\boldsymbol{T}(z)$ is true if no truncation is used — owing to the paraunitary nature of $\boldsymbol{T}(z)$ — discarding $\boldsymbol{S}_{12}^{(\hat{I})}(z)$ and $\boldsymbol{S}_{21}^{(\hat{I})}(z)$ upon algorithm completion results in only approximate equality for $\boldsymbol{R}(z) \approx \tilde{\boldsymbol{T}}(z)\boldsymbol{B}(z)\boldsymbol{T}(z)$. As $\hat{I}$ increases, the energy in these discarded regions tends to zero; thus, the mean square reconstruction error of the decomposition produced by SMS also tends to zero.

The above steps of SMS are summarised in Algorithm 5.

4.4.2 Algorithm Convergence

To show that the SMS algorithm outlined above converges to an approximate block diagonalisation (in the absence of polynomial matrix truncation) of an input paraher-mitian matrix $\boldsymbol{R}(z)$, the following theorem — which is very similar to the theorem for convergence of the SMD algorithm in [45] — is used:

Theorem 1 (Convergence of the SMS algorithm)**.** *With a sufficiently large number of iterations, the sequential matrix segmentation algorithm approximately block diagonalises* $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ *and decreases the energy in the bottom-left* $P \times (M-P)$ *and top-right* $(M-P)\times P$ *regions to an arbitrarily low threshold* $\rho > 0$ *for any* P *that fulfils* $P < M$*.*

Proof: To prove Theorem 1, a number of norms need to be defined:

$$\mathcal{N}_1\left\{\boldsymbol{S}^{(i)}(z)\right\} \triangleq \sum_{m=1}^{M}\left|s^{(i)}_{m,m}[0]\right|^2 \tag{4.26}$$

$$\mathcal{N}_2\left\{\boldsymbol{S}^{(i)}(z)\right\} \triangleq \left\|\mathbf{S}^{(i)}[0]\right\|_{\mathrm{F}}^2 \tag{4.27}$$

$$\mathcal{N}_3\left\{\boldsymbol{S}^{(i)}(z)\right\} \triangleq \mathcal{N}_2\left\{\boldsymbol{S}^{(i)}(z)\right\} - \mathcal{N}_1\left\{\boldsymbol{S}^{(i)}(z)\right\} \tag{4.28}$$

$$\mathcal{N}_4\left\{\boldsymbol{S}^{(i)}(z)\right\} \triangleq \sum_{\tau}\left\|\mathbf{S}^{(i)}[\tau]\right\|_{\mathrm{F}}^2 . \tag{4.29}$$

Note that $\mathcal{N}_1\{\cdot\}$ is invariant under a delay matrix as in (4.20), i.e.,

$$\begin{aligned}\mathcal{N}_1\left\{\boldsymbol{S}^{(i)\prime}(z)\right\} &= \mathcal{N}_1\left\{\boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{\Lambda}}^{(i)}(z)\right\}\\ &= \mathcal{N}_1\left\{\boldsymbol{S}^{(i-1)}(z)\right\} ,\end{aligned} \tag{4.30}$$

and that $\mathcal{N}_2\{\cdot\}$ is invariant under a unitary operation, i.e.,

$$\begin{aligned}\mathcal{N}_2\left\{\boldsymbol{S}^{(i)}(z)\right\} &= \mathcal{N}_2\left\{\mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}\right\}\\ &= \mathcal{N}_2\left\{\boldsymbol{S}^{(i)\prime}(z)\right\} .\end{aligned} \tag{4.31}$$

Furthermore, $\mathcal{N}_4\{\cdot\}$ is invariant under the application of a paraunitary $\boldsymbol{U}^{(i)}(z)$ such that

$$\begin{aligned}\mathcal{N}_4\left\{\boldsymbol{S}^{(i)}(z)\right\} &= \mathcal{N}_4\left\{\boldsymbol{U}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{U}}^{(i)}(z)\right\}\\ &= \mathcal{N}_4\left\{\boldsymbol{S}^{(i-1)}(z)\right\} .\end{aligned} \tag{4.32}$$

The squared norm of the bottom-left $P \times (M-P)$ region of $\mathbf{S}^{(i-1)}[\tau^{(i)}]$ is given by

$$\zeta^{(i)} = \left\| \mathbf{S}_{21}^{(i-1)}[\tau^{(i)}] \right\|_{\mathrm{F}}^2 \ . \tag{4.33}$$

With step (4.20), this energy is transferred onto both the bottom-left $P \times (M-P)$ and top-right $(M-P) \times P$ regions of the lag zero slice $\mathbf{S}^{(i)\prime}[0]$, such that its total off-diagonal energy is

$$\mathcal{N}_3 \left\{ \boldsymbol{S}^{(i)\prime}(z) \right\} = 2\zeta^{(i)} \ . \tag{4.34}$$

In the subsequent rotation step with $\mathbf{Q}^{(i)}$, this energy is transferred onto the main diagonal such that $\mathcal{N}_3 \left\{ \boldsymbol{S}^{(i)}(z) \right\} = 0$ and therefore

$$\begin{aligned} \mathcal{N}_1 \left\{ \boldsymbol{S}^{(i)}(z) \right\} &= \mathcal{N}_1 \left\{ \boldsymbol{S}^{(i)\prime}(z) \right\} + 2\zeta^{(i)} \\ &= \mathcal{N}_1 \left\{ \boldsymbol{S}^{(i-1)}(z) \right\} + 2\zeta^{(i)} \ , \end{aligned} \tag{4.35}$$

exploiting (4.30), while the overall energy $\mathcal{N}_4 \left\{ \boldsymbol{S}^{(i)}(z) \right\}$ remains constant. Due to (4.35), $\mathcal{N}_1 \left\{ \boldsymbol{S}^{(i)}(z) \right\}$ increases monotonically with iteration index i. Since $\mathcal{N}_4 \left\{ \boldsymbol{S}^{(i)}(z) \right\}$ is invariant over iterations due to (4.32) and forms an upper bound

$$\mathcal{N}_1 \left\{ \boldsymbol{S}^{(i)}(z) \right\} \leq \mathcal{N}_4 \left\{ \boldsymbol{S}^{(i)}(z) \right\} \quad \forall\, i \ , \tag{4.36}$$

$\mathcal{N}_1 \left\{ \boldsymbol{S}^{(i)}(z) \right\}$ must have a supremum $\mathcal{S}$,

$$\mathcal{S} = \sup_i \mathcal{N}_1 \left\{ \boldsymbol{S}^{(i)}(z) \right\} \ . \tag{4.37}$$

It follows that for any $\rho > 0$, there must be an iteration number I for which $\left| \mathcal{S} - \mathcal{N}_1 \left\{ \boldsymbol{S}^{(I)}(z) \right\} \right| < \rho$, and so the increase $2\zeta^{(I+i)}$, $i > 0$, at any subsequent stage must satisfy

$$2\zeta^{(I+i)} \leq \left| \mathcal{S} - \mathcal{N}_1 \left\{ \boldsymbol{S}^{(I)}(z) \right\} \right| < \rho \ . \tag{4.38}$$

Hence, for any $\rho > 0$, there must be an iteration I by which $\zeta^{(I)}$ is bounded by ρ. □

4.4.3 Algorithm Complexity

At the ith iteration, the length of $\boldsymbol{S}^{(i)\prime}(z)$ is equal to $L\{\boldsymbol{S}^{(i)\prime}(z)\}$, where $L\{\cdot\}$ computes the length of a polynomial matrix. For (4.23), every matrix-valued coefficient in $\boldsymbol{S}^{(i)\prime}(z)$ must be left- and right-multiplied with a unitary matrix. Accounting for a multiplication of 2 $M \times M$ matrices by M^3 multiply-accumulates (MACs) [2, 95], a total of $2L\{\boldsymbol{S}^{(i)\prime}(z)\}M^3$ MACs arise to generate $\boldsymbol{S}^{(i)}(z)$. Every matrix-valued coefficient in $\boldsymbol{H}^{(i)\prime}(z)$ must also be left-multiplied with a unitary matrix; thus, a total of $L\{\boldsymbol{H}^{(i)\prime}(z)\}M^3$ MACs arise to generate $\boldsymbol{H}^{(i)}(z)$. If $\kappa = 0$, the cumulative complexity of the SMS algorithm over $I_{\max}$ iterations can therefore be approximated as

$$C_{\text{SMS}}(I_{\max}) = M^3 \sum_{i=0}^{I_{\max}} (2L\{\boldsymbol{S}^{(i)\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime}(z)\}) \,, \tag{4.39}$$

which is equivalent to the approximate complexity of the SMD algorithm in Section A.1.2.

4.4.4 Results and Discussion

Block Diagonalisation Performance Metric

Iterative PEVD algorithms progressively minimise off-diagonal energy and therefore use the metric $E^{(i)}_{\text{diag}}$ — defined in Section A.3 and [45] — to measure diagonalisation performance; this metric divides the off-diagonal energy in the parahermitian matrix at the ith iteration by the total energy. The SMS algorithm seeks to iteratively block diagonalise a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ by progressively minimising the energy in the bottom-left $P \times (M-P)$ and top-right $(M-P) \times P$ regions. A suitable metric for block diagonalisation, derived from $E^{(i)}_{\text{diag}}$, is therefore

$$E^{(i)}_{\text{b.diag}} = \frac{2\sum_\tau \left\| \mathbf{S}^{(i)}_{21}[\tau] \right\|^2_{\text{F}}}{\sum_\tau \|\mathbf{R}[\tau]\|^2_{\text{F}}} \,, \tag{4.40}$$

which divides the sum of the energy in the bottom-left $P\times(M-P)$ and top-right $(M-P)\times P$ regions of the iteratively updated matrix by the total energy. Here, $\left\| \mathbf{S}^{(i)}_{21}[\tau] \right\|^2_{\text{F}}$

can be found using (4.22). Computation of $E_{\text{b.diag}}^{(i)}$ generates squared covariance terms; therefore a logarithmic notation of $5\log_{10} E_{\text{b.diag}}^{(i)}$ is employed during testing.

Simulation Scenarios

The simulations below have been performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M \in \{10;\ 30\}$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$.

An initial simulation scenario is designed to measure the block diagonalisation performance of the novel SMS algorithm relative to the SMD algorithm. During 200 iterations of SMD and SMS, truncation parameters of $\mu = \mu_t = 10^{-6}$ and stopping thresholds of $\epsilon = \kappa = 0$ are used. SMS is executed with $P = 5$. At every iteration step, the metrics defined in Section A.3 are recorded together with the elapsed execution time and $E_{\text{b.diag}}^{(i)}$.

A second simulation scenario is designed to measure the error introduced to the decomposition produced by SMS by discarding non-zero $\boldsymbol{S}_{12}^{(\hat{I})}(z)$ and $\boldsymbol{S}_{21}^{(\hat{I})}(z)$ upon algorithm completion. To ensure no error is introduced by polynomial matrix truncation, truncation parameters of $\mu = \mu_t = 0$ are used. Each instance of SMS is executed for $I_{\max} \in \{1; 2; \ldots; 4000\}$ iterations ($\kappa = 0$) with $P = 5$. Following $\hat{I} = I_{\max}$ iterations of each instance of SMS, the decomposition MSE metric defined in Section A.3 is recorded alongside $E_{\text{b.diag}}^{(I_{\max})}$.

Performance of SMS Relative to SMD

Out of interest, the ensemble-averaged diagonalisation $E_{\text{diag}}^{(i)}$ was calculated for both SMD and SMS for the first simulation scenario for $M = 10$, and can be seen plotted against algorithm iteration in Figure 4.3. Clearly, SMD outperforms SMS with respect to parahermitian matrix diagonalisation; however, this is to be expected, as SMS only enforces block diagonalisation.

For $M = 10$, Figure 4.4 confirms that SMS offers far superior block diagonalisation per algorithm iteration to SMD. As a result of the different search strategy used by SMS, both the iteratively updated paraunitary and parahermitian matrices, $\boldsymbol{H}^{(i)}(z)$ and

$\boldsymbol{S}^{(i)}(z)$, are shorter in SMS than in SMD, as shown by Figures 4.5 and 4.6, respectively. As evidenced by Figure 4.7, shorter polynomial matrices directly translate to improved algorithm speed.

Figure 4.8 shows that for constant P, the performance gap between SMS and SMD for $M = 30$ has increased relative to the gap in Figure 4.7. This indicates that SMS becomes more effective as the ratio M/P increases, as SMS is able to focus on minimising the energy in an increasingly small percentage of the parahermitian matrix.

Decomposition Error Introduced by SMS

In the absence of truncation, the SMD algorithm — on which SMS is based — offers a PEVD with zero reconstruction error. However, even for $\mu = \mu_t = 0$, the decomposition offered by SMS is typically not without error, as true block diagonalisation is not normally achieved. For the second simulation scenario, Figure 4.9 demonstrates that for $M \in \{10; 30\}$, as the maximum number of SMS iterations increases, the decomposition MSE decreases. This decrease is rapid in the range $I_{\max} \in [0, 500]$, but becomes notably slower in the range $I_{\max} \in [3500, 4000]$. For $M = 30$, which has a larger ratio M/P, the relative area of the regions whose energy is minimised is smaller; thus, the energy contained within these regions has less impact on the decomposition MSE.

Given that both the block diagonalisation metric and decomposition MSE effectively measure the energy in $\boldsymbol{S}_{12}^{(\hat{I})}(z)$ upon completion of the SMS algorithm, it is perhaps intuitive that an approximately linear relationship — as observed in Figure 4.10 — exists between $5\log_{10}\mathcal{E}\{E_{\text{b.diag}}^{(I_{\max})}\}$ and $10\log_{10}\mathcal{E}\{\text{MSE}\}$.

Values of $I_{\max}$ for application purposes must therefore be given strong consideration, as direct links exist between $I_{\max}$, decomposition MSE, and block diagonalisation performance.

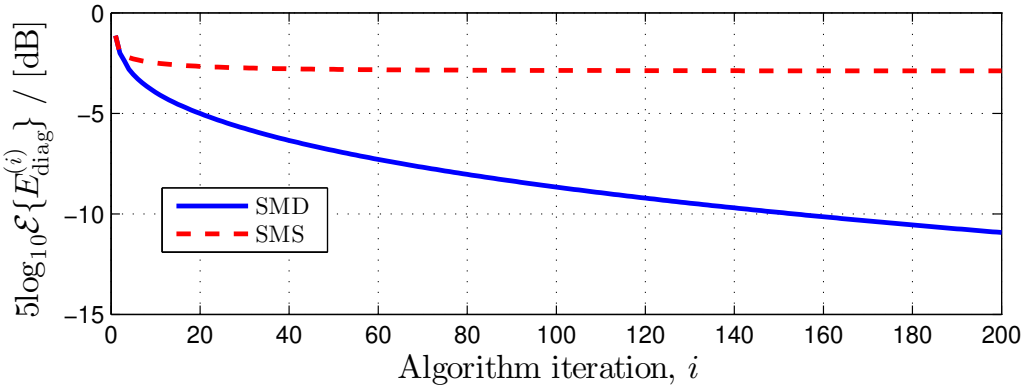

Figure 4.3: Diagonalisation metric versus algorithm iteration for SMD and SMS for $M = 10$.

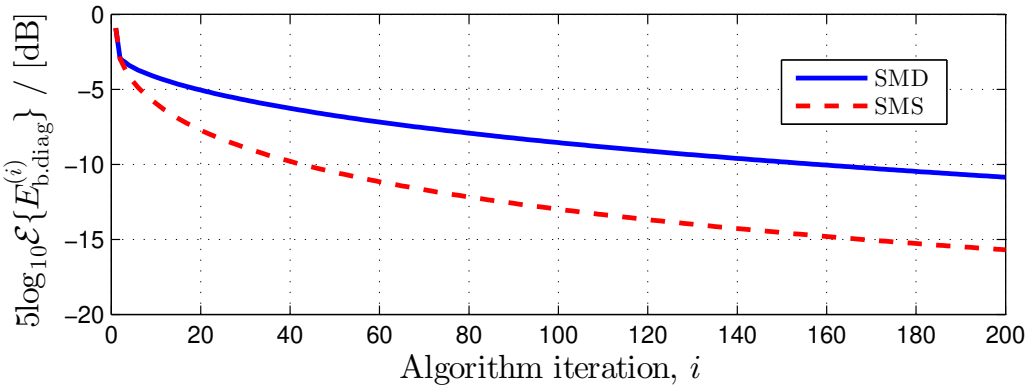

Figure 4.4: Block diagonalisation metric versus algorithm iteration for SMD and SMS for $M = 10$.

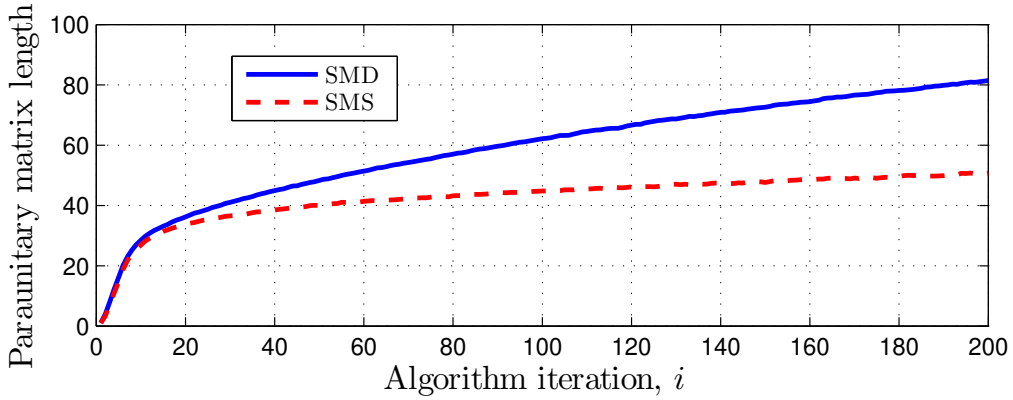

Figure 4.5: Paraunitary matrix length versus algorithm iteration for SMD and SMS for $M = 10$.

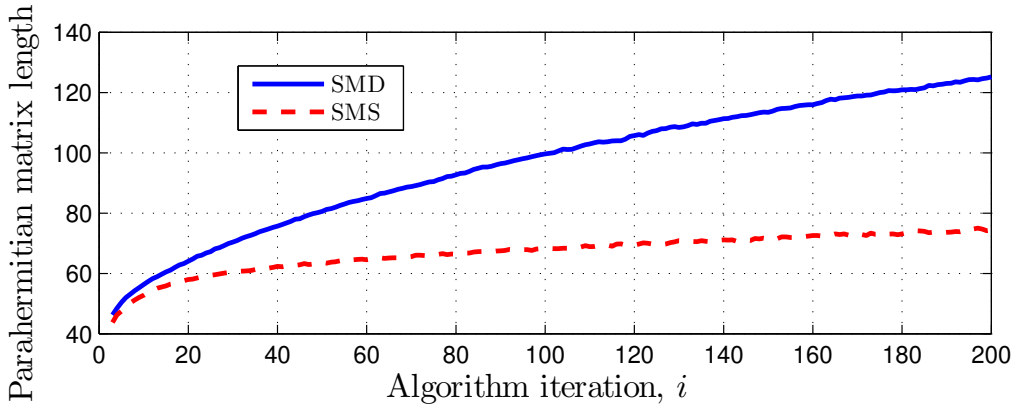

Figure 4.6: Parahermitian matrix $\boldsymbol{S}^{(i)}(z)$ length versus algorithm iteration for SMD and SMS for $M = 10$.

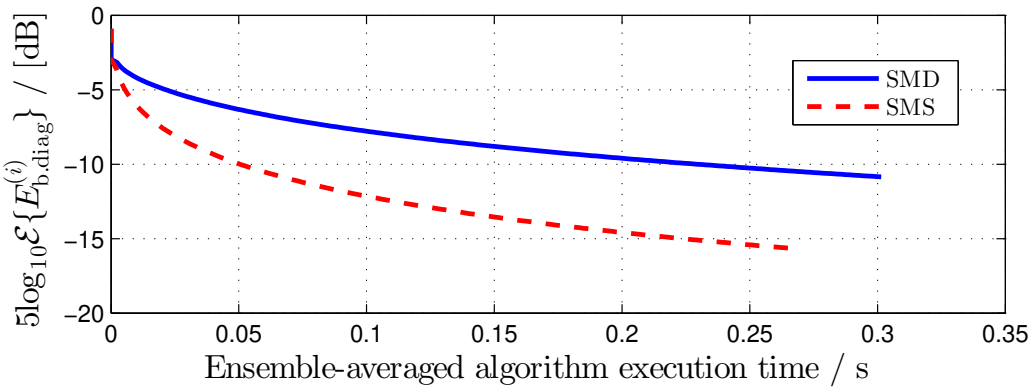

Figure 4.7: Block diagonalisation metric versus algorithm execution time for SMD and SMS for $M = 10$.

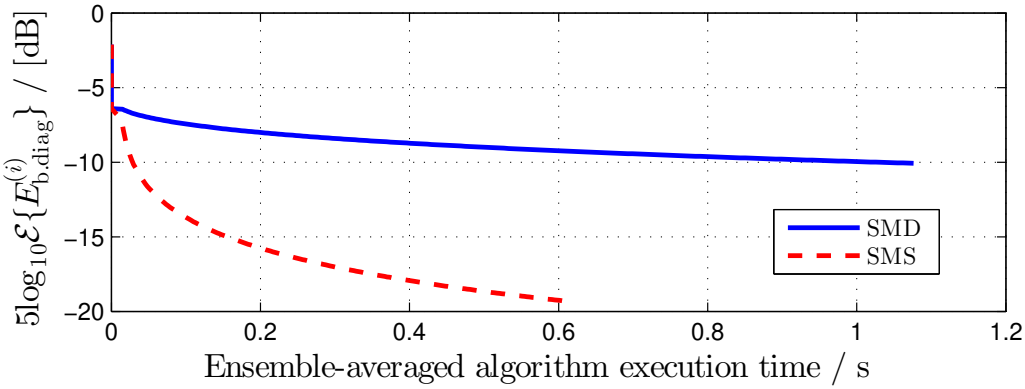

Figure 4.8: Block diagonalisation metric versus algorithm execution time for SMD and SMS for $M = 30$.

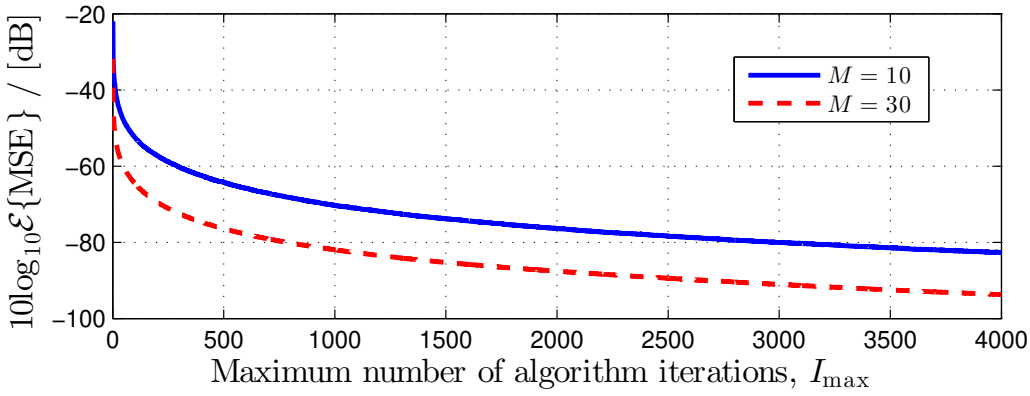

Figure 4.9: Maximum number of algorithm iterations versus decomposition MSE for SMS for $M \in \{10; 30\}$.

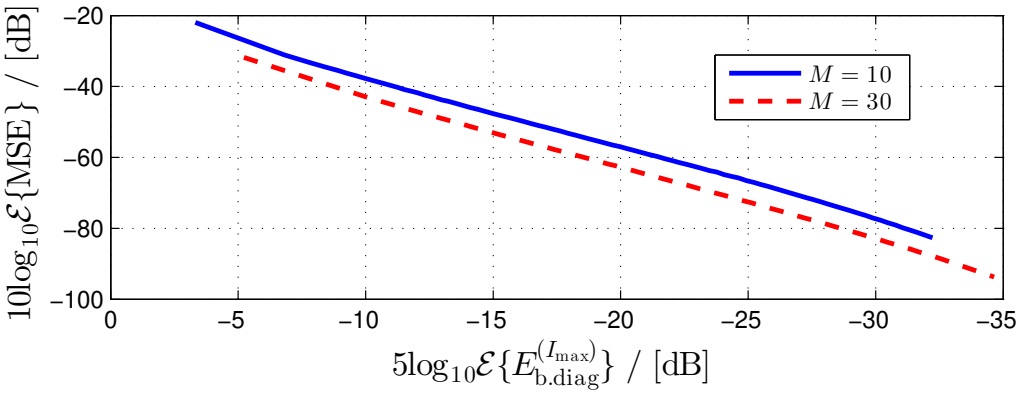

Figure 4.10: Block diagonalisation metric versus decomposition MSE for SMS for $M \in \{10; 30\}$.

4.5 Divide-and-Conquer Sequential Matrix Diagonalisation PEVD Algorithm

Inspired by the development of divide-and-conquer solutions to eigenproblems in [104–106], this section outlines the components of a novel divide-and-conquer sequential matrix diagonalisation (DC-SMD) PEVD algorithm — which is summarised in Section 4.5.1. The 'divide' stage of this algorithm, which utilises the iterative block diagonalisation approaches of Sections 4.3 and 4.4, is described in Section 4.5.2, and Section 4.5.3 details the 'conquer' stage. Some comments on algorithm convergence, decomposition error, and computational complexity are provided in Sections 4.5.4, 4.5.5, and 4.5.6, respectively.

4.5.1 Algorithm Overview

The DC-SMD algorithm diagonalises a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ via a number of paraunitary operations. The algorithm outputs an approximately diagonal matrix $\boldsymbol{D}(z)$, which contains approximate polynomial eigenvalues, and an approximately paraunitary $\boldsymbol{F}(z)$, which contains the corresponding approximate polynomial eigenvectors, such that (2.4) is satisfied.

While all existing iterative PEVD algorithms, such as SMD — described in Section A.1 — attempt to diagonalise an entire $M \times M$ parahermitian matrix at once, the DC-SMD algorithm first transforms the matrix into block diagonal form in a 'divide' stage before diagonalising — or 'conquering' — each of the smaller, now independent, matrices on the diagonal separately. For example, a matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{20\times 20}$ might be 'divided' into four 5×5 parahermitian matrices, each of which can be diagonalised independently. Figure 4.11, which is a condensed form of Figure 4.1, shows the state of the parahermitian matrix at each stage of the process for this example.

If matrix $\boldsymbol{R}(z)$ is of large spatial dimension, the SMS algorithm from Section 4.4 is repeatedly used to 'divide' the matrix into multiple independent parahermitian matrices. Using the block diagonalisation approach of Section 4.3.2, this function generates a paraunitary matrix $\boldsymbol{T}^{(b)}(z)$ that 'divides' the top-left $(M-(b-1)P)\times(M-(b-1)P)$

block of an input matrix $\boldsymbol{B}^{(b-1)}(z)$ into two independent parahermitian matrices, $\boldsymbol{B}_{11}^{(b)}(z)$ and $\boldsymbol{B}_{22}^{(b)}(z)$, of smaller spatial dimension. $\boldsymbol{B}_{11}^{(b)}(z)$ is then subject to further division if it still has sufficiently large spatial dimension. Note that $\boldsymbol{B}^{(0)}(z) = \boldsymbol{R}(z)$. Following $\beta = \left\lceil \frac{M-\hat{M}}{P} \right\rceil$ 'division' steps, for $b = 1 \dots \beta$ and user-defined $\hat{M}$ and P, a block diagonal $\boldsymbol{B}(z)$ has been obtained as in (4.12). In the alternative notation of (4.4), $\boldsymbol{B}(z)$ comprises $N = (\beta+1)$ blocks, which are denoted $\boldsymbol{B}_{nn}(z)$, $n = 1 \dots N$. The matrices $\boldsymbol{T}^{(b)}(z)$ — which SMS generates to 'divide' each $\boldsymbol{B}^{(b)}(z)$ — are concatenated to form an overall dividing matrix $\boldsymbol{T}(z)$. It is therefore possible to approximately reconstruct $\boldsymbol{R}(z)$ from the product $\tilde{\boldsymbol{T}}(z)\boldsymbol{B}(z)\boldsymbol{T}(z)$.

As in the strategy of Section 4.3.1, each block $\boldsymbol{B}_{nn}(z)$, $n = 1 \dots N$, on the diagonal of matrix $\boldsymbol{B}(z)$ is then diagonalised in sequence through the use of a PEVD algorithm; in this case, SMD is used. The diagonalised outputs, $\boldsymbol{D}_{nn}(z)$, are placed on the diagonal of matrix $\boldsymbol{D}(z)$, and the corresponding paraunitary matrices, $\hat{\boldsymbol{F}}_{nn}(z)$, are stored on the diagonal of matrix $\hat{\boldsymbol{F}}(z)$. Matrix $\boldsymbol{B}(z)$ can be approximately reconstructed from $\tilde{\hat{\boldsymbol{F}}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)$; by extension, it is possible to approximately reconstruct $\boldsymbol{R}(z)$ from the product $\tilde{\boldsymbol{T}}(z)\tilde{\hat{\boldsymbol{F}}}(z)\boldsymbol{D}(z)\hat{\boldsymbol{F}}(z)\boldsymbol{T}(z) = \tilde{\boldsymbol{F}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)$, where $\boldsymbol{F}(z) = \hat{\boldsymbol{F}}(z)\boldsymbol{T}(z)$.

Algorithm 6 summarises the above steps of DC-SMD in more detail. Of the parameters passed to DC-SMD, μ and μ_t are truncation parameters, and κ and ϵ are the stopping thresholds for SMS and SMD, which are allowed a maximum of I_D and I_C iterations. Matrices of spatial dimension greater than $\hat{M} \times \hat{M}$ will be subject to the 'dividing' process. Parameter P is an input to SMS as in Section 4.4. Matrices $\mathbf{I}_M$ and $\mathbf{0}_M$ are identity and zero matrices of spatial dimensions $M \times M$, respectively.

4.5.2 'Dividing' the Parahermitian Matrix

If $\boldsymbol{R}(z)$ is measured to have spatial dimension $M > \hat{M}$, the 'divide' stage of DC-SMD comes into effect. This stage repeatedly applies the SMS algorithm of Section 4.4 to 'divide' $\boldsymbol{R}(z)$ into multiple independent parahermitian matrices, and can therefore be considered to operate in a recursive fashion. In the first recursion, the matrix $\boldsymbol{B}^{(0)}(z)$ input to SMS is equal to $\boldsymbol{R}(z)$ and $M' = M$. The output matrix $\boldsymbol{B}_{22}^{(1)}(z)$ is stored and subsequently diagonalised during the 'conquer' stage. If the second output matrix

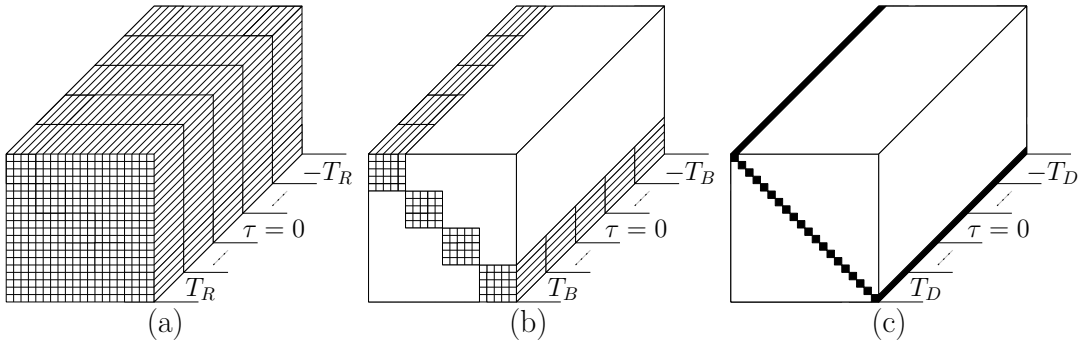

Figure 4.11: (a) Original matrix $\mathbf{R}[\tau] \in \mathbb{C}^{20\times 20}$, (b) block diagonal $\mathbf{B}[\tau]$, and (c) diagonalised output $\mathbf{D}[\tau]$. T_R, T_B, and T_D are the maximum lags for matrices $\mathbf{R}[\tau]$, $\mathbf{B}[\tau]$, and $\mathbf{D}[\tau]$, respectively.

$\boldsymbol{B}_{11}^{(1)}(z)$ is of a spatial dimension greater than $\hat{M}\times\hat{M}$, the second recursion of the 'divide' stage uses $\boldsymbol{B}_{11}^{(1)}(z)$ as the input to SMS, and M' is set equal to $M - P$. Recursions continue in this fashion until $(M' - P) \leq \hat{M}$.

All matrices input to the 'conquer' stage should have spatial dimensions of at most $\hat{M} \times \hat{M}$, therefore P, which determines the dimensions of the smaller matrix produced during each 'divide' step, is forced to satisfy $P \leq \hat{M}$.

4.5.3 'Conquering' the Independent Matrices

At this stage of DC-SMD, $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ has been 'divided' into multiple independent parahermitian matrices, which are stored as blocks on the diagonal of $\boldsymbol{B}(z)$. Each matrix can now be diagonalised individually through the use of a PEVD algorithm; here, the SMD algorithm — which is described in detail in Section A.1 — is chosen. Upon completion, the SMD algorithm returns matrices $\hat{\boldsymbol{F}}_{nn}(z)$ and $\boldsymbol{D}_{nn}(z)$, which contain the polynomial eigenvectors and eigenvalues for input matrix $\boldsymbol{B}_{nn}(z)$, respectively. At iteration n of this stage, $\boldsymbol{B}_{nn}(z)$ contains the nth block of $\boldsymbol{B}(z)$ from the top-left.

4.5.4 Algorithm Convergence

The SMD and SMS algorithms have been shown to converge in [45] and Section 4.4.2, respectively. Given that SMD and SMS form the only active, iterative components of

Input: $\boldsymbol{R}(z)$, μ, μ_t, κ, ϵ, I_D, I_C, $\hat{M}$, P
Output: $\boldsymbol{D}(z)$, $\boldsymbol{F}(z)$
Determine if input matrix is large:
if $M > \hat{M}$ **then**
  Large matrix — divide-and-conquer:
  $M' \leftarrow M$; $\boldsymbol{B}^{(0)}(z) \leftarrow \boldsymbol{R}(z)$; $\boldsymbol{T}(z) \leftarrow \mathbf{I}_M$; $\boldsymbol{B}(z), \hat{\boldsymbol{F}}(z), \boldsymbol{D}(z) \leftarrow \mathbf{0}_M$; $b \leftarrow 0$
  'Divide' matrix:
  while $M' > \hat{M}$ **do**
    $b \leftarrow b + 1$
    $[\boldsymbol{B}^{(b)}(z), \hat{\boldsymbol{T}}^{(b)}(z)] \leftarrow \text{SMS}(\boldsymbol{B}^{(b-1)}(z), I_D, P, \mu, \mu_t, \kappa)$
    $\boldsymbol{T}^{(b)}(z) \leftarrow \text{diag}\left\{\hat{\boldsymbol{T}}^{(b)}(z), \mathbf{I}_{M-M'}\right\}$
    Store $\boldsymbol{B}_{22}^{(b)}(z)$ on diagonal of $\boldsymbol{B}(z)$ in bth $P \times P$ block from bottom-right
    $\boldsymbol{T}(z) \leftarrow \boldsymbol{T}^{(b)}(z)\boldsymbol{T}(z)$; $\boldsymbol{B}^{(b)}(z) \leftarrow \boldsymbol{B}_{11}^{(b)}(z)$; $M' \leftarrow M' - P$
  end
  Store $\boldsymbol{B}^{(b)}(z)$ on diagonal of $\boldsymbol{B}(z)$ in top-left $M' \times M'$ block
  'Conquer' independent matrices:
  for $n \leftarrow 1$ **to** $(b+1)$ **do**
    $\boldsymbol{B}_{nn}(z)$ is nth block of $\boldsymbol{B}(z)$ from top-left
    $[\boldsymbol{D}_{nn}(z), \hat{\boldsymbol{F}}_{nn}(z)] \leftarrow \text{SMD}(\boldsymbol{B}_{nn}(z), I_C, \epsilon, \mu, \mu_t)$
    Store $(\boldsymbol{D}_{nn}(z), \hat{\boldsymbol{F}}_{nn}(z))$ in nth block of $(\boldsymbol{D}(z), \hat{\boldsymbol{F}}(z))$ from top-left
  end
  $\boldsymbol{F}(z) \leftarrow \hat{\boldsymbol{F}}(z)\boldsymbol{T}(z)$
else
  Small matrix — perform 'conquer' stage only:
  $[\boldsymbol{D}(z), \boldsymbol{F}(z)] \leftarrow \text{SMD}(\boldsymbol{R}(z), I_C, \epsilon, \mu, \mu_t)$
end

Algorithm 6: DC-SMD Algorithm

DC-SMD, it can therefore be concluded that DC-SMD must also converge for a suitable combination of I_D and I_C.

4.5.5 Impact of Algorithm Parameters on Decomposition Error

Imperfect transformation of the parahermitian matrix to block diagonal form can be observed if SMS is not executed with a sufficient number of iterations; i.e., if I_D is too low or κ is too high. A result of this is that matrices $\boldsymbol{S}_{21}^{(\hat{I})}(z)$ and $\boldsymbol{S}_{12}^{(\hat{I})}(z)$ internal to SMS contain non-zero energy when iterations end. Given that these matrices are discarded upon completion of an instance of SMS, an error can be introduced to the PEVD produced by the DC-SMD algorithm. To reduce this error, which worsens the approximation given by (2.4), the parameter I_D can be increased, or κ can be decreased;

however, both of these changes will reduce the speed and increase the complexity of the algorithm, as more effort will be contributed to the 'divide' stage.

Parahermitian and paraunitary matrix truncation (for non-zero truncation parameters μ and μ_t) both introduce an error to the resulting PEVD. Larger truncation parameter values introduce a higher degree of error. Any error from truncation both worsens the approximation given by (2.4) and weakens the paraunitary property of eigenvectors $\boldsymbol{F}(z)$; i.e., equality in (2.5) is no longer guaranteed if truncation is employed during generation of $\boldsymbol{F}(z)$. This problem is not unique to DC-SMD, however, as all iterative PEVD algorithms typically employ polynomial matrix truncation to constrain computational complexity and memory requirements [6, 60–62].

4.5.6 Algorithm Complexity

The majority of the computational cost of the DC-SMD algorithm arises from multiple instances of SMS and SMD. The instantaneous complexity of DC-SMD therefore varies as the algorithm progresses, due to the changing spatial dimensions of the matrices being processed by SMS and SMD. The main cost of these internal algorithms can be attributed to the matrix multiplication steps in (4.23) and (A.5). As mentioned previously, multiplying two $\mathcal{M}\times\mathcal{M}$ matrices together requires approximately $\mathcal{M}^3$ MAC operations [2, 95]. Here, $\mathcal{M}$ captures the different spatial dimensions used throughout DC-SMD and satisfies $\min\{M-\beta P, P\} \leq \mathcal{M} \leq M$. If the number of MACs required to complete an operation is used as an estimation of computational complexity, the computational complexity of a single matrix multiplication step in either algorithm at iteration i for $\boldsymbol{S}^{(i)\prime}(z) : \mathbb{C} \to \mathbb{C}^{\mathcal{M}\times\mathcal{M}}$ and $\boldsymbol{H}^{(i)\prime}(z) : \mathbb{C} \to \mathbb{C}^{\mathcal{M}\times\mathcal{M}}$ can be approximated as

$$C^{(i)}_{\mathrm{DC-SMD},\mathcal{M}} = \mathcal{M}^3(2L\{\boldsymbol{S}^{(i)\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime}(z)\}) \,, \tag{4.41}$$

where operator $L\{\cdot\}$ computes the length of a polynomial matrix.

From the description of DC-SMD in Algorithm 6, it can be seen that an $M \times M$ matrix is only ever processed in the first recursion of the 'divide' stage; at all other points in the DC-SMD algorithm, the processed matrices are of lower spatial dimension. For example, typically all but one of the parahermitian matrices in the 'conquer'

stage are of spatial dimension $P \times P$. Given that the complexity is approximately proportional to the cube of the spatial dimension, significantly lower complexity will be observed beyond the first recursion of DC-SMD. Simulations demonstrating how this lower complexity, divide-and-conquer approach translates to lower algorithm execution times are provided in Section 4.7.

4.6 Parallel-Sequential Matrix Diagonalisation PEVD Algorithm

Motivated by the development of the DC-SMD algorithm in Section 4.5, here a low complexity, partially parallelisable DaC approach for the PEVD — titled parallel-sequential matrix diagonalisation (PSMD) — that improves upon the DC-SMD algorithm is described. This algorithm achieves novelty by being the only PEVD algorithm to combine the complexity reduction techniques of Sections 3.2, 3.3, 3.6, and 3.8. Following a sequential, matrix 'divide' stage, which segments a large parahermitian matrix into multiple independent parahermitian matrices, a parallelised 'conquer' stage is used to diagonalise each independent matrix simultaneously. Both the 'divide' and 'conquer' stages make use of the algorithmic optimisations in Section 3.2 and the half-matrix and restricted update algorithmic improvements from Sections 3.3 and 3.6, respectively, to minimise algorithm complexity. The final stage of the algorithm employs the compensated row-shift truncation scheme of Section 3.8 to reduce the polynomial order of the paraunitary matrix.

The PSMD algorithm is summarised in Section 4.6.1. The implementation of complexity reduction techniques within the 'divide' and 'conquer' stages of this algorithm is described in Sections 4.6.2 and 4.6.3. Some comments on algorithm convergence, decomposition error, and computational complexity are provided in Sections 4.6.4, 4.6.5, and 4.6.6, respectively.

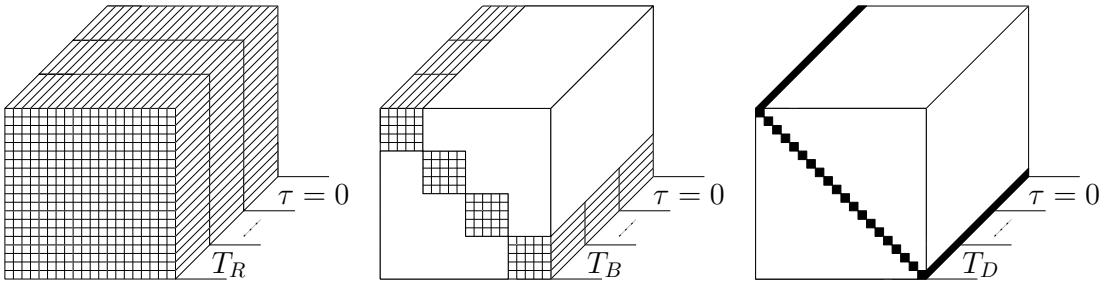

Figure 4.12: (a) Original matrix $\overline{\mathbf{R}}[\tau] \in \mathbb{C}^{20\times 20}$, (b) block diagonal $\overline{\mathbf{B}}[\tau]$, and (c) diagonalised output $\overline{\mathbf{D}}[\tau]$. T_R, T_B, and T_D are the maximum lags for matrices $\overline{\mathbf{R}}[\tau]$, $\overline{\mathbf{B}}[\tau]$, and $\overline{\mathbf{D}}[\tau]$, respectively.

4.6.1 Algorithm Overview

The PSMD algorithm diagonalises a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ via a number of paraunitary operations. The algorithm returns an approximately diagonal matrix $\boldsymbol{D}(z)$, which contains the approximate eigenvalues, and an approximately paraunitary matrix $\boldsymbol{F}(z)$, which contains the corresponding approximate eigenvectors, such that (2.4) is satisfied.

Similarly to DC-SMD, the PSMD algorithm first transforms an input parahermitian matrix into block diagonal form in a 'divide' stage before diagonalising or 'conquering' each of the smaller, now independent, matrices on the diagonal separately. The 'divide' stage is a sequential process, while the 'conquer' stage is parallelised. For example, a matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{20\times 20}$ might be 'divided' into four 5×5 parahermitian matrices, each of which can be diagonalised independently and simultaneously. Figure 4.12 shows the state of the parahermitian matrix at each stage of the process for this example. As in Section 3.3, here the natural symmetry of the parahermitian matrix structure is exploited and only one half of its elements are stored; i.e., the algorithm diagonalises $\overline{\boldsymbol{R}}(z)$. Given the demonstration of its ability to reduce the complexity of PEVD algorithms in Section 3.6, the restricted update method is employed in the 'divide' and 'conquer' stages of PSMD. This method restricts both the search and update spaces of the algorithms used in each stage of PSMD.

If matrix $\boldsymbol{R}(z)$ is of large spatial dimension, a novel half-matrix, restricted update version of the SMS algorithm from Section 4.4 — denoted HRSMS — is repeatedly used

to 'divide' the matrix into multiple independent parahermitian matrices. Using the block diagonalisation approach of Section 4.3.2, this function generates a paraunitary matrix $\boldsymbol{T}^{(b)}(z)$ that 'divides' the top-left $(M-(b-1)P)\times(M-(b-1)P)$ block of an input matrix $\overline{\boldsymbol{B}}^{(b-1)}(z)$ into two independent parahermitian matrices, $\overline{\boldsymbol{B}}^{(b)}_{11}(z)$ and $\overline{\boldsymbol{B}}^{(b)}_{22}(z)$, of smaller spatial dimension. $\overline{\boldsymbol{B}}^{(b)}_{11}(z)$ is then subject to further division if it still has sufficiently large spatial dimension. Note that $\overline{\boldsymbol{B}}^{(0)}(z) = \overline{\boldsymbol{R}}(z)$. Following $\beta = \left\lceil \frac{M-\hat{M}}{P} \right\rceil$ 'division' steps, $b = 1 \ldots \beta$, a block diagonal $\overline{\boldsymbol{B}}(z)$ has been obtained as in (4.12). In the alternative notation of (4.4), $\overline{\boldsymbol{B}}(z)$ comprises $N = (\beta+1)$ blocks, which are denoted $\overline{\boldsymbol{B}}_{nn}(z)$, $n = 1 \ldots N$. The matrices $\boldsymbol{T}^{(b)}(z)$ — which HRSMS generates to 'divide' each $\overline{\boldsymbol{B}}^{(b)}(z)$ — are concatenated to form an overall dividing matrix $\boldsymbol{T}(z)$. With a full-matrix representation, it is therefore possible to approximately reconstruct $\boldsymbol{R}(z)$ from the product $\tilde{\boldsymbol{T}}(z)\boldsymbol{B}(z)\boldsymbol{T}(z)$.

Using the strategy of Section 4.3.1, each block $\overline{\boldsymbol{B}}_{nn}(z)$, $n = 1 \ldots N$, on the diagonal of matrix $\overline{\boldsymbol{B}}(z)$ is then diagonalised in parallel through the use of a novel half-matrix, restricted update version of the SMD algorithm named HRSMD. The diagonalised outputs, $\overline{\boldsymbol{D}}_{nn}(z)$, are placed on the diagonal of matrix $\overline{\boldsymbol{D}}(z)$, and the corresponding paraunitary matrices, $\hat{\boldsymbol{F}}_{nn}(z)$, are stored on the diagonal of matrix $\hat{\boldsymbol{F}}(z)$. Full-matrix $\boldsymbol{B}(z)$ can be approximately reconstructed from $\tilde{\hat{\boldsymbol{F}}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)$; by extension, it is possible to approximately reconstruct $\boldsymbol{R}(z)$ from the product $\tilde{\boldsymbol{T}}(z)\tilde{\hat{\boldsymbol{F}}}(z)\boldsymbol{D}(z)\hat{\boldsymbol{F}}(z)\boldsymbol{T}(z) = \tilde{\boldsymbol{F}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)$, where $\boldsymbol{F}(z) = \hat{\boldsymbol{F}}(z)\boldsymbol{T}(z)$.

The polynomial matrix truncation scheme of Appendix C is implemented within HRSMS and HRSMD. While the paraunitary matrix compensated row-shift truncation (CRST) scheme of Section 3.8 has outperformed traditional truncation strategies when paired with the SBR2 PEVD algorithm, it is more computationally costly than the method of Appendix C. Furthermore, a row-shift truncation (RST) scheme has been shown not to provide an increase in truncation performance when implemented within the SMD algorithm [63], which serves as the foundation of HRSMS and HRSMD. However, the RST scheme has been found to be effective when truncating the output paraunitary matrix of a DaC PEVD scheme in [64]. Similarly, the CRST scheme is employed to truncate the final paraunitary matrix in PSMD. Subsequent use of function

$f_{\text{trim}}(\cdot)$, which implements the truncation scheme of Appendix C, truncates the final parahermitian matrix.

The optimisations for polynomial matrix multiplication and truncation discussed in Section 3.2 are employed throughout PSMD; however, specifics regarding their implementation on each occasion are omitted for brevity.

Algorithm 7 summarises the above steps of PSMD in more detail. Of the parameters input to PSMD, μ, μ_t, and μ_s are truncation parameters, and κ and ϵ are stopping thresholds for HRSMS and HRSMD, which are allowed a maximum of I_D and I_C iterations. Matrices of spatial dimension greater than $\hat{M} \times \hat{M}$ will be subject to the 'dividing' process. Parameter P is an input to HRSMS, and is used in the same capacity as for SMS in Section 4.4. Matrices $\mathbf{I}_M$ and $\mathbf{0}_M$ are identity and zero matrices of spatial dimensions $M \times M$, respectively.

4.6.2 'Dividing' the Parahermitian Matrix

If $\boldsymbol{R}(z)$ is measured to have spatial dimension $M > \hat{M}$, the 'divide' stage of PSMD comes into effect. This stage functions in the same manner as the 'divide' stage of DC-SMD in Section 4.5.2, but maintains only half-matrix representations for all parahermitian matrices and uses HRSMS instead of SMS.

The restricted update method of Section 3.6 is employed in HRSMS; this facilitates calculation of the paraunitary matrix while restricting the search space of the algorithm and the portion of the parahermitian matrix that is updated at each iteration. This restriction limits both the number of search operations and the computations required to update the increasingly block diagonalised parahermitian matrix. Over the course of algorithm iterations, the update space contracts piecewise strictly monotonically. That is, the update space contracts until order zero is reached; after this, in a so-called regeneration step, the calculated paraunitary matrix is applied to the input matrix to construct the full-sized parahermitian factor. The update space is then maximised and thereafter again contracts monotonically over the following iterations.

Input: $\boldsymbol{R}(z)$, μ, μ_t, μ_s κ, ϵ, I_D, I_C, $\hat{M}$, P
Output: $\boldsymbol{D}(z)$, $\boldsymbol{F}(z)$
Determine if input matrix is large:
if $M > \hat{M}$ **then**
  Large matrix — divide-and-conquer:
  $M' \leftarrow M$; $\overline{\boldsymbol{B}}^{(0)}(z) \leftarrow \overline{\boldsymbol{R}}(z)$; $\boldsymbol{T}(z) \leftarrow \mathbf{I}_M$; $\overline{\boldsymbol{B}}(z), \hat{\boldsymbol{F}}(z), \overline{\boldsymbol{D}}(z) \leftarrow \mathbf{0}_M$; $b \leftarrow 0$
  'Divide' matrix:
  while $M' > \hat{M}$ **do**
    $b \leftarrow b + 1$
    $[\overline{\boldsymbol{B}}^{(b)}(z), \hat{\boldsymbol{T}}^{(b)}(z)] \leftarrow \text{HRSMS}(\overline{\boldsymbol{B}}^{(b-1)}(z), I_D, P, \mu, \mu_t, \kappa)$
    $\boldsymbol{T}^{(b)}(z) \leftarrow \text{diag}\left\{\hat{\boldsymbol{T}}^{(b)}(z), \mathbf{I}_{M-M'}\right\}$
    Store $\overline{\boldsymbol{B}}^{(b)}_{22}(z)$ on diagonal of $\overline{\boldsymbol{B}}(z)$ in bth $P \times P$ block from bottom-right
    $\boldsymbol{T}(z) \leftarrow \boldsymbol{T}^{(b)}(z)\boldsymbol{T}(z)$; $\overline{\boldsymbol{B}}^{(b)}(z) \leftarrow \overline{\boldsymbol{B}}^{(b)}_{11}(z)$; $M' \leftarrow M' - P$
  end
  Store $\overline{\boldsymbol{B}}^{(b)}(z)$ on diagonal of $\overline{\boldsymbol{B}}(z)$ in top-left $M' \times M'$ block
  'Conquer' independent matrices in parallel:
  for $n \leftarrow 1$ **to** $(b+1)$ **do**
    $\overline{\boldsymbol{B}}_{nn}(z)$ is nth block of $\overline{\boldsymbol{B}}(z)$ from top-left
    $[\overline{\boldsymbol{D}}_{nn}(z), \hat{\boldsymbol{F}}_{nn}(z)] \leftarrow \text{HRSMD}(\overline{\boldsymbol{B}}_{nn}(z), I_C, \epsilon, \mu, \mu_t)$
    Store $(\overline{\boldsymbol{D}}_{nn}(z), \hat{\boldsymbol{F}}_{nn}(z))$ in nth block of $(\overline{\boldsymbol{D}}(z), \hat{\boldsymbol{F}}(z))$ from top-left
  end
  $\boldsymbol{F}(z) \leftarrow \hat{\boldsymbol{F}}(z)\boldsymbol{T}(z)$
else
  Small matrix — perform 'conquer' stage only:
  $[\overline{\boldsymbol{D}}(z), \boldsymbol{F}(z)] \leftarrow \text{HRSMD}(\overline{\boldsymbol{R}}(z), I_C, \epsilon, \mu, \mu_t)$
end
Apply compensated row-shift truncation:
$[\boldsymbol{F}(z), \boldsymbol{D}(z)] \leftarrow f_{\text{crst}}(\boldsymbol{F}(z), \boldsymbol{D}(z), \mu_s)$
$\mathbf{D}[\tau] \leftarrow f_{\text{trim}}(\mathbf{D}[\tau], \mu)$

Algorithm 7: PSMD Algorithm

HRSMS Algorithm

As the HRSMS algorithm functions very similarly to the SMS algorithm described in Section 4.4, only the main differences between the two will be described below.

Upon initialisation, the HRSMS algorithm diagonalises the lag zero coefficient matrix $\overline{\mathbf{R}}[0]$ of the half-matrix representation of parahermitian matrix $\boldsymbol{R}(z)$ by means of its modal matrix $\mathbf{Q}^{(0)}$, which is obtained from the ordered EVD of $\overline{\mathbf{R}}[0]$, such that $\overline{\boldsymbol{S}}^{(0)}(z) = \mathbf{Q}^{(0)}\overline{\boldsymbol{R}}(z)\mathbf{Q}^{(0)\text{H}}$. The unitary $\mathbf{Q}^{(0)}$ is applied to all coefficient matrices $\overline{\mathbf{R}}[\tau]$ $\forall\, \tau \geq 0$, and initialises $\boldsymbol{H}^{(0)}(z) = \mathbf{Q}^{(0)}$.

Although it actually operates on $\overline{\boldsymbol{S}}^{(i)}(z)$, the HRSMS algorithm effectively mimics

Input: $\overline{\boldsymbol{S}}(z)$, τ_s, $\boldsymbol{\Lambda}(z)$, T, M, P
Output: $\overline{\boldsymbol{S}}'(z)$
$\boldsymbol{\Gamma}(z) \leftarrow \begin{bmatrix} \gamma_{1,1}(z) & \dots & \gamma_{1,M}(z) \\ \vdots & \ddots & \vdots \\ \gamma_{M,1}(z) & \dots & \gamma_{M,M}(z) \end{bmatrix}$
if $\tau_s > 0$ **then**
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{\Lambda}(z)\overline{\boldsymbol{S}}(z)$
  $\gamma_{m,k}(z) \leftarrow \begin{cases} \sum_{\tau=-\tau_s+1}^{0} \mathbf{L}_{m,k}[\tau] z^{-\tau}, & k < (M-P+1) \leq m \\ 0, & \text{otherwise} \end{cases}$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z) + z^{\tau_s}\tilde{\boldsymbol{\Gamma}}(z)$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z)\tilde{\boldsymbol{\Lambda}}(z)$
else if $\tau_s < 0$ **then**
  $\boldsymbol{L}(z) \leftarrow \overline{\boldsymbol{S}}(z)\tilde{\boldsymbol{\Lambda}}(z)$
  $\gamma_{m,k}(z) \leftarrow \begin{cases} \sum_{\tau=\tau_s+1}^{0} \mathbf{L}_{m,k}[\tau] z^{-\tau}, & m < (M-P+1) \leq k \\ 0, & \text{otherwise} \end{cases}$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z) + z^{-\tau_s}\tilde{\boldsymbol{\Gamma}}(z)$
  $\boldsymbol{L}(z) \leftarrow \boldsymbol{\Lambda}(z)\boldsymbol{L}(z)$
else
  $\boldsymbol{L}(z) \leftarrow \overline{\boldsymbol{S}}(z)$
end
$\overline{\boldsymbol{S}}'(z) \leftarrow \sum_{\tau=0}^{T+|\tau_s|} \mathbf{L}[\tau] z^{-\tau}$

Algorithm 8: $\text{shift}_{\text{HSMS}}(\cdot)$ function

SMS by computing (4.17) in the ith step, $i = 1, 2, \dots \hat{I}$. Paraunitary delay matrix $\boldsymbol{\Lambda}^{(i)}(z)$ is as defined in (4.19); however, it cannot be directly used to shift energy in $\overline{\boldsymbol{S}}^{(i)}(z)$ due to the latter's half-matrix form. Instead, the intermediate variables $\overline{\boldsymbol{S}}^{(i)\prime}(z)$ and $\boldsymbol{H}^{(i)\prime}(z)$ are obtained according to

$$\begin{aligned} \overline{\boldsymbol{S}}^{(i)\prime}(z) &= \text{shift}_{\text{HSMS}}(\, \overline{\boldsymbol{S}}^{(i-1)}(z),\, \tau^{(i)}, \boldsymbol{\Lambda}^{(i)}(z), T^{(i-1)}, M, P\,) \\ \boldsymbol{H}^{(i)\prime}(z) &= \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z) \quad , \end{aligned} \tag{4.42}$$

where $\text{shift}_{\text{HSMS}}(\cdot)$ — which is very similar to $\text{shift}_{\text{HSMD}}(\cdot)$ from Section 3.3, and is described in Algorithm 8 — implements the delays encapsulated in the matrix $\boldsymbol{\Lambda}^{(i)}(z)$ for a half-matrix representation and $T^{(i-1)}$ is the maximum lag of $\overline{\mathbf{S}}^{(i-1)}[\tau]$. The matrix $\boldsymbol{\Lambda}^{(i)}(z)$ is selected based on the lag position of the dominant region in

$\overline{\boldsymbol{S}}^{(i-1)}(z) \bullet\!\!-\!\!\circ\, \overline{\mathbf{S}}^{(i-1)}[\tau]$, which is identified by

$$\tau^{(i)} = \arg\max_{\tau} \left\{ \left\| \overline{\mathbf{S}}_{21}^{(i-1)}[\tau] \right\|_{\mathrm{F}}^2 \,,\; \left\| \overline{\mathbf{S}}_{12}^{(i-1)}[-\tau] \right\|_{\mathrm{F}}^2 \right\} \,, \tag{4.43}$$

where

$$\left\| \overline{\mathbf{S}}_{21}^{(i-1)}[\tau] \right\|_{\mathrm{F}}^2 = \sum_{m=M-P+1}^{M} \sum_{k=1}^{M-P} \left| \bar{s}_{m,k}^{(i-1)}[\tau] \right|^2 \quad , \tag{4.44}$$

and

$$\left\| \overline{\mathbf{S}}_{12}^{(i-1)}[\tau] \right\|_{\mathrm{F}}^2 = \sum_{m=1}^{M-P} \sum_{k=M-P+1}^{M} \left| \bar{s}_{m,k}^{(i-1)}[\tau] \right|^2 \quad . \tag{4.45}$$

Here, $\bar{s}_{m,k}^{(i-1)}[\tau]$ represents the element in the mth row and kth column of the coefficient matrix $\overline{\mathbf{S}}^{(i-1)}[\tau]$.

According to the $\mathrm{shift}_{\mathrm{HSMS}}(\cdot)$ function: if (4.43) returns $\tau^{(i)} > 0$, then the bottom-left $P \times (M-P)$ region of $\overline{\boldsymbol{S}}^{(i-1)}(z)$ is to be shifted by $\tau^{(i)}$ lags towards lag zero. If $\tau^{(i)} < 0$, it is the top-right $(M-P) \times P$ region that requires shifting by $-\tau^{(i)}$ lags towards lag zero. To preserve the half-matrix representation, elements that are shifted beyond lag zero — i.e., outside the recorded half-matrix — have to be stored as their parahermitian conjugate and appended onto the bottom-left $P \times (M-P)$ (for $\tau^{(i)} < 0$) or top-right $(M-P) \times P$ (for $\tau^{(i)} > 0$) region of the shifted matrix at lag zero. The concatenated region is then shifted by $|\tau^{(i)}|$ lags towards increasing τ. Note that the $\mathrm{shift}_{\mathrm{HSMS}}(\cdot)$ function shifts the bottom-right $P \times P$ region of $\overline{\boldsymbol{S}}^{(i-1)}(z)$ in opposite directions, such that this region remains unaffected. An efficient implementation of $\mathrm{shift}_{\mathrm{HSMS}}(\cdot)$ can therefore exclude this region from shifting operations.

An efficient example of the shift operation is depicted in Figure 4.13 for the case of $\overline{\boldsymbol{S}}^{(i-1)}(z) : \mathbb{C} \to \mathbb{C}^{5\times 5}$ with parameters $\tau^{(i)} = -3$, $T^{(i-1)} = 3$, and $P = 2$. Owing to the negative sign of $\tau^{(i)}$, it is here the top-right $(M-P) \times P$ region that has to be shifted first, followed by the bottom-left $P \times (M-P)$ region, which is shifted in the opposite direction.

The shifting process in (4.42) moves the dominant bottom-left region $\overline{\mathbf{S}}_{21}^{(i-1)}[\tau^{(i)}]$ or dominant top-right region $\overline{\mathbf{S}}_{12}^{(i-1)}[-\tau^{(i)}]$ in $\overline{\mathbf{S}}^{(i-1)}[\tau]$ into the lag zero coefficient matrix $\overline{\mathbf{S}}^{(i)\prime}[0]$. Note that the $\mathrm{shift}_{\mathrm{HSMS}}(\cdot)$ function ensures that the Hermitian symmetry of

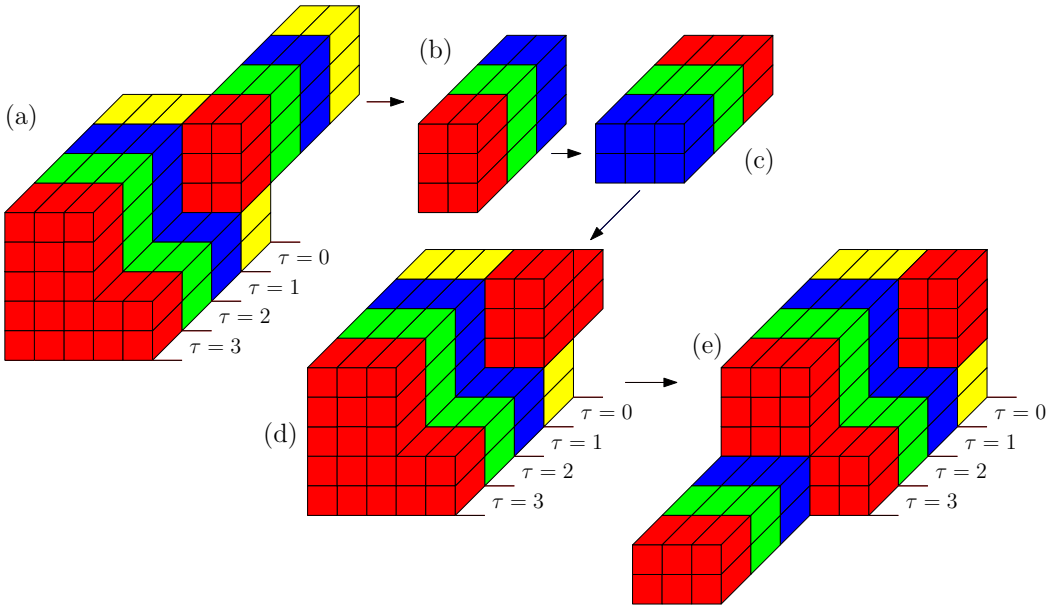

Figure 4.13: Example for a matrix where in the ith iteration the Frobenius norm of a region in the top-right of the matrix is maximum: (a) the region is shifted, with elements in the region past lag zero (b) extracted and (c) parahermitian conjugated; (d) these elements are appended to the bottom-left region at lag zero and (e) shifted in the opposite direction.

$\bar{\mathbf{S}}^{(i)\prime}[0]$ is maintained such that the relationship $\bar{\mathbf{S}}^{(i)\prime}[0] = \bar{\mathbf{S}}^{(i)\prime\mathrm{H}}[0]$ holds; i.e., the shifted bottom-left or top-right region also exists as its Hermitian transpose in the top-right or bottom-left region of $\bar{\mathbf{S}}^{(i)\prime}[0]$.

Following shifting, in accordance with the restricted update scheme of Section 3.6, a matrix

$$\bar{\boldsymbol{S}}^{(i)\prime\prime}(z) = \sum_{\tau=0}^{T^{(i-1)}-|\tau^{(i)}|} \bar{\mathbf{S}}^{(i)\prime}[\tau] z^{-\tau} \tag{4.46}$$

is obtained, which is of lower order than $\bar{\boldsymbol{S}}^{(i)\prime}(z)$, and is therefore less computationally costly to update in the subsequent rotation step. Applying (4.46) at each iteration enforces a monotonic contraction of the update space of the algorithm. Truncation of $\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)$ at each iteration can therefore be avoided, as its order is not increasing. As a result of this, the search space of (4.43) is also limited, which negatively impacts the convergence of HRSMS, as the same $\tau^{(i)}$ as an unrestricted version of the algorithm may not be identified. Despite this, the proof of convergence for SMS in Section 4.4.2

— which focusses on the monotonic increase of on-diagonal energy enforced by the algorithm — still holds for HRSMS.

The order of the paraunitary matrix $\boldsymbol{H}^{(i)\prime}(z)$ does increase at each iteration; to constrain computational complexity, a truncated paraunitary matrix

$$\mathbf{H}^{(i)\prime\prime}[\tau] = f_{\text{trim}}(\mathbf{H}^{(i)\prime}[\tau], \mu_t) \tag{4.47}$$

is obtained using the $f_{\text{trim}}(\cdot)$ function from Appendix C.

The energy in the shifted regions is then transferred onto the diagonal of $\bar{\mathbf{S}}^{(i)\prime\prime}[0]$ by a unitary matrix $\mathbf{Q}^{(i)}$ — which diagonalises $\bar{\mathbf{S}}^{(i)\prime\prime}[0]$ by means of an ordered EVD — in

$$\begin{aligned} \overline{\boldsymbol{S}}^{(i)}(z) &= \mathbf{Q}^{(i)}\,\overline{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}} \\ \boldsymbol{H}^{(i)}(z) &= \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime\prime}(z) \quad . \end{aligned} \tag{4.48}$$

If at this point the order of $\overline{\boldsymbol{S}}^{(i)}(z)$ is zero, a regenerated parahermitian matrix

$$\boldsymbol{S}^{(i)}(z) \leftarrow \boldsymbol{H}^{(i)}(z)\boldsymbol{R}(z)\tilde{\boldsymbol{H}}^{(i)}(z) \tag{4.49}$$

is obtained; this is then truncated to minimise future computational complexity:

$$\mathbf{S}^{(i)}[\tau] \leftarrow f_{\text{trim}}(\mathbf{S}^{(i)}[\tau], \mu)\ . \tag{4.50}$$

Note that obtaining the regenerated matrix requires the use of a full-matrix representation. Following regeneration, algorithm iterations continue with a half-matrix representation.

Figure 4.14 demonstrates the progression of several iterations of the HRSMS algorithm for $M = 5$, $T^{(i-1)} = 3$, and $P = 2$. As can be seen, after three iterations, the maximum lag of the matrix in Figure 4.14(i) is equal to zero; at this point, parahermitian matrix regeneration must occur.

After a user-defined $I_{\max}$ iterations, or when the energy in $\overline{\boldsymbol{S}}^{(i)}_{21}(z)$ and $\overline{\boldsymbol{S}}^{(i)}_{12}(z)$ has

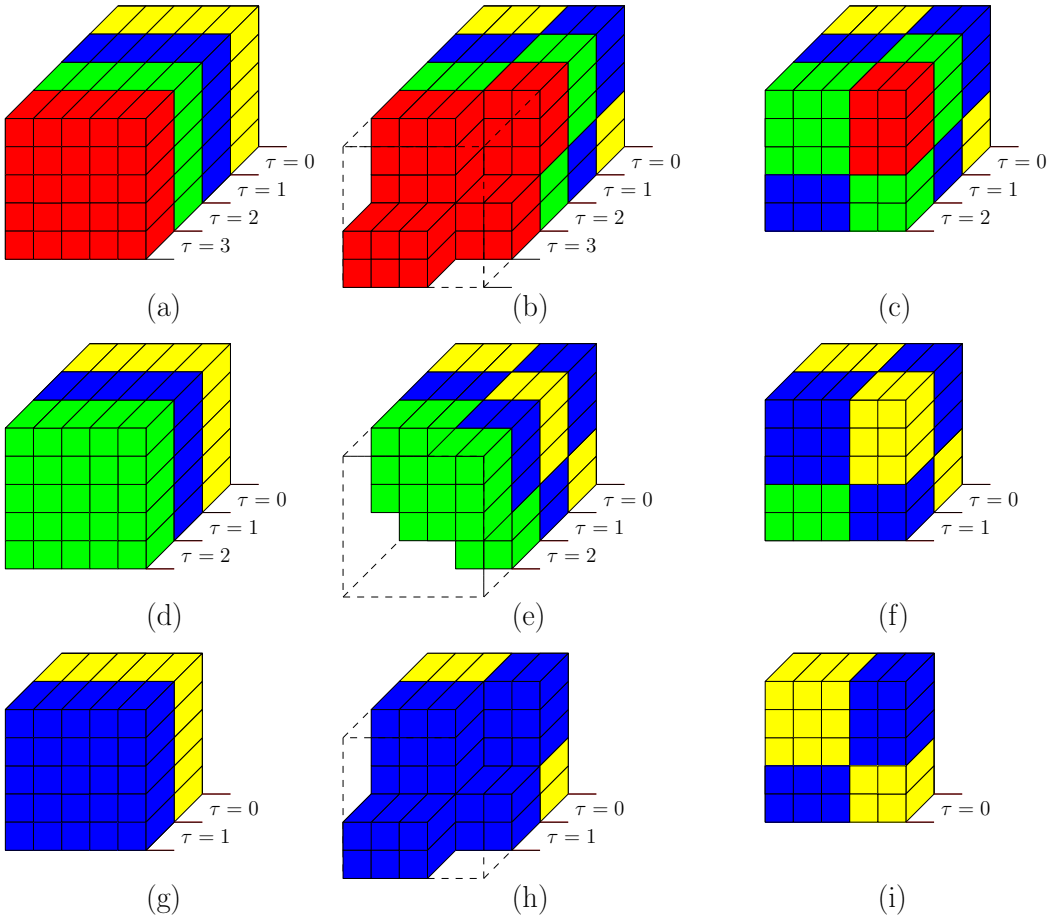

Figure 4.14: (a) Matrix $\overline{\boldsymbol{S}}^{(i-1)}(z) : \mathbb{C} \to \mathbb{C}^{5\times 5}$ with maximum lag $T^{(i-1)} = 3$ and $P = 2$; (b) shifting of region with maximum energy to lag zero ($\tau^{(i)} = -1$); (c) central matrix with maximum lag ($T^{(i-1)} - |\tau^{(i)}|$) $= 2$, $\overline{\boldsymbol{S}}^{(i)\prime\prime}(z)$, is extracted. (d) $\overline{\boldsymbol{S}}^{(i)}(z) = \mathbf{Q}^{(i)}\overline{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$; (e) $\tau^{(i+1)} = 1$; (f) $\overline{\boldsymbol{S}}^{(i+1)\prime\prime}(z)$ extracted. (g) $\overline{\boldsymbol{S}}^{(i+1)}(z)$; (h) $\tau^{(i+2)} = -1$; (i) $\overline{\boldsymbol{S}}^{(i+2)\prime\prime}(z)$ is extracted.

been sufficiently minimised such that

$$\max_{\tau}\left\{\left\|\overline{\mathbf{S}}_{21}^{(I)}[\tau]\right\|_{\mathrm{F}}^{2}\ ,\ \left\|\overline{\mathbf{S}}_{12}^{(I)}[-\tau]\right\|_{\mathrm{F}}^{2}\right\} \leq \kappa \sum_{\tau}\|\mathbf{R}[\tau]\|_{\mathrm{F}}^{2} \tag{4.51}$$

at some iteration I — where κ is chosen to be arbitrarily small — the HRSMS algorithm returns matrices $\overline{\boldsymbol{B}}(z)$ and $\boldsymbol{T}(z)$. The latter is constructed from the concatenation of the elementary paraunitary matrices as in (4.25). The parahermitian submatrices of $\overline{\boldsymbol{B}}(z)$, $\overline{\boldsymbol{B}}_{11}(z)$ and $\overline{\boldsymbol{B}}_{22}(z)$, are the top-left $(M-P)\times(M-P)$ and the bottom-right $P \times P$ blocks of $\overline{\boldsymbol{S}}^{(\hat{I})}(z)$, $\overline{\boldsymbol{S}}_{11}^{(\hat{I})}(z)$ and $\overline{\boldsymbol{S}}_{22}^{(\hat{I})}(z)$, respectively, where $\hat{I} = \min\{I_{\max}, I\}$.

Input: $\overline{\boldsymbol{R}}(z)$, $I_{\max}$, P, μ, μ_t, κ
Output: $\overline{\boldsymbol{B}}$, $\boldsymbol{T}(z)$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise $\overline{\mathbf{R}}[0] \in \mathbb{C}^{M\times M}$
$\overline{\boldsymbol{S}}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\overline{\boldsymbol{R}}(z)\mathbf{Q}^{(0)\mathrm{H}}$; $\boldsymbol{H}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$; $i \leftarrow 0$; stop $\leftarrow 0$
do
  $i \leftarrow i+1$
  Find $\tau^{(i)}$ from (4.43); generate $\boldsymbol{\Lambda}^{(i)}(z)$ from (4.19)
  $\overline{\boldsymbol{S}}^{(i)\prime}(z) \leftarrow \mathrm{shift}_{\mathrm{HSMS}}(\,\overline{\boldsymbol{S}}^{(i-1)}(z), \tau^{(i)}, \boldsymbol{\Lambda}^{(i)}(z), T^{(i-1)}, M, P\,)$
  $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
  $\overline{\boldsymbol{S}}^{(i)\prime\prime}(z) \leftarrow \sum_{\tau=0}^{T^{(i-1)}-|\tau^{(i)}|} \overline{\mathbf{S}}^{(i)\prime}[\tau]z^{-\tau}$
  $\mathbf{H}^{(i)\prime\prime}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{H}^{(i)\prime}[\tau], \mu_t)$
  Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\overline{\mathbf{S}}^{(i)\prime\prime}[0]$
  $\overline{\boldsymbol{S}}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\overline{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$
  $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime\prime}(z)$
  if $T^{(i)} = 0$ **or** $i > I_{\max}$ **or** (4.51) satisfied **then**
    $\boldsymbol{S}^{(i)}(z) \leftarrow \boldsymbol{H}^{(i)}(z)\boldsymbol{R}(z)\tilde{\boldsymbol{H}}^{(i)}(z)$
    $\mathbf{S}^{(i)}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{S}^{(i)}[\tau], \mu)$
  end
  if $i > I_{\max}$ or (4.51) satisfied **then**
    stop $\leftarrow 1$
  end
while stop $= 0$
$\boldsymbol{T}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$
$\overline{\boldsymbol{B}}_{11}(z)$ is top-left $(M-P)\times(M-P)$ block of $\overline{\boldsymbol{S}}^{(i)}(z)$
$\overline{\boldsymbol{B}}_{22}(z)$ is bottom-right $P \times P$ block of $\overline{\boldsymbol{S}}^{(i)}(z)$

Algorithm 9: HRSMS algorithm

The above steps of HRSMS are summarised in Algorithm 9.

4.6.3 'Conquering' the Independent Matrices

At this stage of PSMD, $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ has been 'divided' into multiple independent parahermitian matrices, which are stored as blocks on the diagonal of $\overline{\boldsymbol{B}}(z)$. Each matrix can now be diagonalised individually through the use of a PEVD algorithm; here, HRSMD is chosen. Upon completion, the HRSMD algorithm returns matrices $\hat{\boldsymbol{F}}_{nn}(z)$ and $\overline{\boldsymbol{D}}_{nn}(z)$, which contain the polynomial eigenvectors and eigenvalues for input matrix $\overline{\boldsymbol{B}}_{nn}(z)$, respectively. At iteration n of this stage, $\overline{\boldsymbol{B}}_{nn}(z)$ contains the nth block of $\overline{\boldsymbol{B}}(z)$ from the top-left.

Input: $\overline{\boldsymbol{R}}(z)$, $I_{\max}$, ϵ, μ, μ_t
Output: $\overline{\boldsymbol{D}}(z)$, $\boldsymbol{F}(z)$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise $\overline{\mathbf{R}}[0] \in \mathbb{C}^{M\times M}$
$\overline{\boldsymbol{S}}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\overline{\boldsymbol{R}}(z)\mathbf{Q}^{(0)\mathrm{H}}$; $\boldsymbol{H}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$; $i \leftarrow 0$; stop $\leftarrow 0$
do
  $i \leftarrow i+1$
  Find $\{k^{(i)}, \tau^{(i)}\}$ from (3.12); generate $\boldsymbol{\Lambda}^{(i)}(z)$ from (3.13)
  $\overline{\boldsymbol{S}}^{(i)\prime}(z) \leftarrow \mathrm{shift}_{\mathrm{HSMD}}(\,\overline{\boldsymbol{S}}^{(i-1)}(z),\, k^{(i)},\, \tau^{(i)}, \boldsymbol{\Lambda}^{(i)}(z), T^{(i-1)}, M\,)$
  $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
  $\overline{\boldsymbol{S}}^{(i)\prime\prime}(z) \leftarrow \sum_{\tau=0}^{T^{(i-1)}-|\tau^{(i)}|} \overline{\mathbf{S}}^{(i)\prime}[\tau]z^{-\tau}$
  $\mathbf{H}^{(i)\prime\prime}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{H}^{(i)\prime}[\tau], \mu_t)$
  Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\overline{\mathbf{S}}^{(i)\prime\prime}[0]$
  $\overline{\boldsymbol{S}}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\overline{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$
  $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime\prime}(z)$
  if $T^{(i)} = 0$ **or** $i > I_{\max}$ **or** (3.14) satisfied **then**
    $\boldsymbol{S}^{(i)}(z) \leftarrow \boldsymbol{H}^{(i)}(z)\boldsymbol{R}(z)\tilde{\boldsymbol{H}}^{(i)}(z)$
    $\mathbf{S}^{(i)}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{S}^{(i)}[\tau], \mu)$
  end
  if $i > I_{\max}$ or (3.14) satisfied **then**
    stop $\leftarrow 1$;
  end
while stop $= 0$
$\boldsymbol{F}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$
$\overline{\boldsymbol{D}}(z) \leftarrow \overline{\boldsymbol{S}}^{(i)}(z)$

Algorithm 10: HRSMD algorithm

With relative ease, the HRSMD algorithm can be derived from the definitions of half-matrix SMD and $\mathrm{shift}_{\mathrm{HSMD}}(\cdot)$ in Section 3.3, restricted update SMD in Section 3.6, HRSMS in Section 4.6.2, and $f_{\mathrm{trim}}(\cdot)$ in Appendix C. For this reason, only the pseudocode of Algorithm 10 is provided to describe the implementation of HRSMD.

For the parallel implementation of the 'conquer' stage in MATLAB®, the Parallel Computing Toolbox™ can be used. This software package allows the processing required by multiple instances of the HRSMD algorithm to be spread across a maximum of 768 CUDA® cores in the NVIDIA® GeForce® GTX 765M graphics card present on the simulation platform defined in Section 2.4. Note that, to within machine precision, a version of PSMD that does not utilise parallelisation will have identical PEVD performance to a parallelised version, though the former will typically require a longer

execution time.

4.6.4 Algorithm Convergence

The SMD and SMS algorithms have been shown to converge in [45] and Section 4.4.2, respectively. Half-matrix versions of each are functionally identical and therefore also converge. Furthermore, employing a restricted update strategy for either algorithm does not impact the proofs of convergence provided in [45] and Section 4.4.2. Given that HRSMD and HRSMS form the only active, iterative components of PSMD, it can therefore be concluded that PSMD must also converge for a suitable combination of I_D and I_C.

4.6.5 Impact of Algorithm Parameters on Decomposition Error

The comments made in Section 4.5.5 with respect to the introduction of error to the decomposition produced by the DC-SMD algorithm can also be extended to PSMD.

4.6.6 Algorithm Complexity

The majority of the computational cost of the PSMD algorithm arises from multiple instances of HRSMS and HRSMD. The instantaneous complexity of PSMD therefore varies as the algorithm progresses, due to the changing spatial dimensions of the matrices being processed by HRSMS and HRSMD, and the parallel implementation of HRSMD. The main cost of these internal algorithms can be attributed to the matrix multiplication step in (4.48) shared by both algorithms. Using $\mathcal{M}$ to capture the different spatial dimensions used throughout PSMD — with $\min\{M-\beta P, P\} \leq \mathcal{M} \leq M$ — and the same assumptions as in Section 4.5.6, the computational complexity of a single matrix multiplication step in either algorithm at iteration i for $\overline{\boldsymbol{S}}^{(i)\prime\prime}(z) : \mathbb{C} \to \mathbb{C}^{\mathcal{M}\times\mathcal{M}}$ and $\boldsymbol{H}^{(i)\prime\prime}(z) : \mathbb{C} \to \mathbb{C}^{\mathcal{M}\times\mathcal{M}}$ can be approximated as

$$C^{(i)}_{\text{PSMD},\mathcal{M}} = \mathcal{M}^3(2L\{\overline{\boldsymbol{S}}^{(i)\prime\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime\prime}(z)\}) \,. \tag{4.52}$$

Note that while the process of matrix regeneration in HRSMS and HRSMD does contribute towards computational complexity, the results of Section 3.6.4 have demonstrated that savings made during the aforementioned matrix multiplication steps outweigh the cost of matrix regeneration over the course of algorithm iterations. The complexity of matrix regeneration is therefore considered to be negligible.

When compared with (4.41), which estimates the complexity of an equivalent step in DC-SMD, (4.52) will typically return a lower value. This disparity arises as a result of the length of the restricted, half-matrix form of $\overline{\boldsymbol{S}}^{(i)\prime\prime}(z)$ being lower than the length of the full-matrix form of $\boldsymbol{S}^{(i)\prime}(z)$ in DC-SMD.

In PSMD, an $M \times M$ matrix is only ever processed in the first recursion of the 'divide' stage; at all other points in the algorithm, the processed matrices are of lower spatial dimension. Given that the complexity is approximately proportional to the cube of the spatial dimension, significantly lower complexity will be observed beyond the first recursion of PSMD. Simulation results that confirm the low complexity of PSMD are provided in the subsequent section.

4.7 Simulations and Results

This section demonstrates that by employing the DaC techniques proposed in previous sections, PEVD complexity and therefore run-time can be significantly reduced when compared with existing iterative PEVD methods. Through the use of the broadband source model of Appendix B [45] in Sections 4.7.1 and 4.7.2, it is possible to evaluate the performance of DaC methods for the decomposition of parahermitian matrices of increasingly large spatial dimension — corresponding to the acquisition of data from an increasingly large number of broadband sensors. Furthermore, to confirm the efficacy of the DaC methods for application purposes, the broadband angle of arrival estimation simulation scenario of Section 4.7.3 couples a number of PEVD algorithms with the spatio-spectral polynomial multiple signal classification (SSP-MUSIC) algorithm from [18].

4.7.1 Source Model Simulation Scenario 1

This scenario provides an introduction to the performance of DaC methods, which do not converge in the same manner as existing PEVD algorithms. A relatively large spatial dimension, M, is used to demonstrate the failings of existing methods and the suitability of DC-SMD and PSMD for applications involving large arrays of sensors.

Simulations for this scenario are performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M \times M}$ obtained from the randomised source model in Appendix B [45] with $M = 30$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$. The performances of the existing SBR2 [6] and SMD [45] PEVD algorithms are compared with the novel DC-SMD and PSMD algorithms. To demonstrate the flexibility of the PSMD algorithm, three variants are tested; these are denoted PSMD^1, PSMD^2, and PSMD^3. SBR2 and SMD are allowed to run for 1800 and 1400 iterations, respectively, with polynomial matrix truncation parameters $\mu_{\text{SBR2}} = \mu_{\text{SMD}} = 10^{-6}$. DC-SMD and PSMD^1 are provided with parameters $\mu = \mu_t = \mu_s = 10^{-12}$, $\kappa = \epsilon = 0$, $I_D = 100$, $I_C = 200$, $\hat{M} = 8$, and $P = 8$. The remaining two variants of PSMD use identical parameters bar some small differences: PSMD^2 employs more significant polynomial matrix truncation and is supplied with $\mu = \mu_t = \mu_s = 10^{-6}$, and PSMD^3 is supplied with $I_D = 400$. At every iteration step of each algorithm, the performance metrics defined in Section A.3 are recorded together with the elapsed execution time.

The multiple shift maximum element SMD (MSME-SMD) of [47] and its derivatives in [48, 50] have been shown to outperform SMD in terms of diagonalisation per algorithm iteration. However, the work of [49] demonstrates that if the update step of each method is replaced with a lower-cost cyclic-by-row approximation, the multiple shift class of algorithms is inferior to SMD with respect to diagonalisation per unit of time. Given that the update steps of all SMD-based algorithms — including those incorporating multiple shift strategies — have been updated to use the efficient matrix multiplication approach of Section 3.2.1, the SMD algorithm now behaves as its approximation in [49] and outperforms all multiple shift methods. This degradation of multiple shift algorithm performance is due to the methods' reliance on a more costly search step, which has a complexity proportional to M^3 [65]. For high M — as in the

simulation scenario here — the cost of this search step prohibits the algorithms' implementation. The multiple shift algorithms are therefore omitted from testing. Note that the search step of SMD has a complexity proportional to only M^2 [65].

A cyclic-by-row approximation to SMD from [49] was to be tested for the above scenario, but was found to take in excess of 500 seconds to complete the same number of iterations as SMD — when using identical algorithm parameters — for the decomposition of one instance of $\boldsymbol{R}(z)$. Thus, this algorithm was excluded from testing.

Diagonalisation

The ensemble-averaged diagonalisation metric at each iteration for each of the tested PEVD algorithms is plotted against the ensemble-averaged elapsed system time at each iteration in Figure 4.15. The curves demonstrate that the proposed implementation achieves a similar degree of diagonalisation to most of the other algorithms, but in a shorter time. The SBR2 algorithm exhibits relatively low diagonalisation with respect to time, and would require a great deal of additional simulation time to attain a diagonalisation performance similar to the other algorithms. By utilising a restricted update approach, PSMD^1 has sacrificed a small amount of diagonalisation performance to decrease algorithm run-time versus the DC-SMD algorithm. Increased levels of truncation within PSMD^2 have decreased algorithm run-time but have also decreased diagonalisation performance slightly. The increase in I_D within PSMD^3 has increased the run-time of the 'divide' stage and marginally improved diagonalisation.

The 'stepped' characteristics of the curves for the DaC strategies of DC-SMD and PSMD are a result of the algorithms' two-stage implementation. The 'divide' steps of the algorithms exhibit low diagonalisation for a large increase in execution time. In the 'conquer' steps, high diagonalisation is seen for a small increase in execution time. Note that the data points in the curve corresponding to the parallelised 'conquer' stage of PSMD could not be acquired directly, as the processing for this stage was distributed across a number of CUDA® cores. The missing data points were instead obtained using knowledge of the time and diagonalisation metric values at the start and end points of the curve, and a ground truth convergence curve for the 'conquer' stage of a

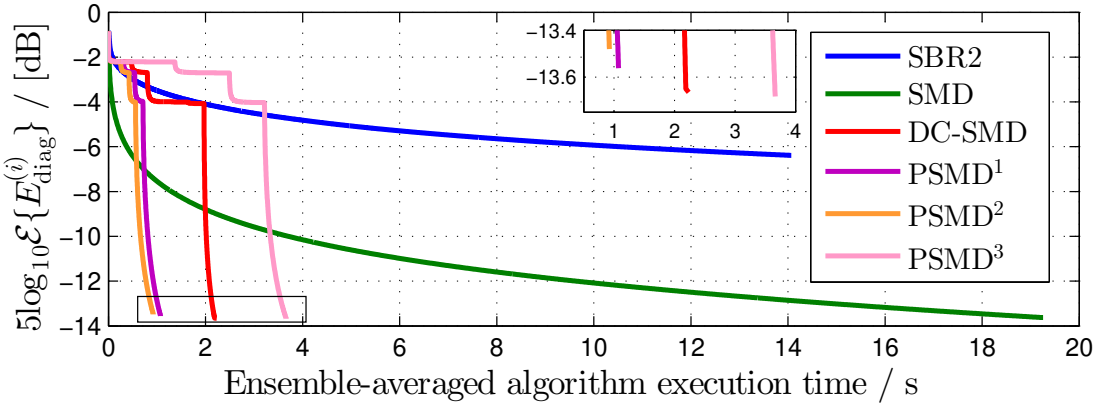

Figure 4.15: Execution time of DC-SMD, PSMD^1, PSMD^2, and PSMD^3 relative to SBR2 [6] and SMD [45] for the decomposition of a 30×30 parahermitian matrix.

non-parallelised version of PSMD.

From Figure 4.15, the average run-time for the PSMD^1 algorithm for the given simulation scenario is 1.075 seconds. If the MATLAB® Parallel Computing Toolbox™ is not used to parallelise the 'conquer' step of PSMD^1 by spreading four instances of HRSMD across a maximum of 768 CUDA® cores, the average run-time increases by 38.1% to 1.485 seconds. The performance of PSMD^1 is otherwise identical. In this case, the use of parallelisation has dramatically reduced the run-time of the 'conquer' stage to the point where it is almost negligible when compared with the run-time of the 'divide' stage. Unfortunately, as the 'divide' stage has to process matrices with larger spatial dimensions, it tends to be slower, and ultimately provides a relatively high lower bound for the overall run-time.

Decomposition Mean Square Error

The ensemble-averaged mean square reconstruction error for each algorithm can be seen in column two of Table 4.1. In agreement with the comments of Section 4.5.5, the DaC methods can be seen to introduce error to the PEVD, and produce a decomposition with higher MSE than SBR2 and SMD. By decreasing truncation levels, the MSE of all PEVD algorithms can be reduced at the expense of longer algorithm run-time and paraunitary matrices of higher order. Conversely, the higher truncation within

Table 4.1: Performance metric comparison for DaC and traditional PEVD methods. Metrics used are: decomposition mean square error, MSE; length of paraunitary matrix $\boldsymbol{F}(z)$, L_F; eigenvalue resolution, λ_{res}; and paraunitarity error, η.

Method	MSE	L_F	λ_{res}	η
SBR2 [6]	1.577×10^{-6}	149.5	1.1305	2.910×10^{-8}
SMD [45]	3.514×10^{-6}	165.5	0.0773	6.579×10^{-8}
DC-SMD	6.785×10^{-6}	360.5	0.0644	1.226×10^{-14}
PSMD^1	6.918×10^{-6}	279.0	0.0658	4.401×10^{-15}
PSMD^2	8.346×10^{-6}	155.6	0.0661	1.303×10^{-8}
PSMD^3	7.618×10^{-7}	307.6	0.0245	1.307×10^{-14}

PSMD^2 has resulted in marginally higher MSE. To reduce the MSE of DC-SMD and PSMD in this scenario, I_D can be increased or κ can be decreased; however, this will reduce the speed of each algorithm, as more effort will be contributed to the 'divide' stage. This can be observed in the results of PSMD^3, which has the lowest MSE of any of the tested algorithms.

Paraunitary Matrix Length

Paraunitary matrix length, L_F, refers to the number of lags for which the energy in $\mathbf{F}[\tau]$ is non-zero; i.e., the number of coefficient matrices required to implement $\boldsymbol{F}(z)$ as a finite impulse response filter bank. From column three of Table 4.1, it can be observed that a disadvantage of DaC strategies is their tendency to produce longer paraunitary matrices. However, it can be seen that the use of CRST in PSMD^1 has successfully reduced L_F relative to DC-SMD. Using higher levels of paraunitary matrix truncation in any algorithm would reduce paraunitary matrix length and algorithm run-time at the expense of higher MSE and paraunitarity error; this relationship is observed in the results of PSMD^2, which is able to provide significantly shorter paraunitary matrices than PSMD^1. Indeed, the matrices produced are actually shorter than those given by SMD. Increasing I_D in PSMD^3 has resulted in an increase in L_F.

Polynomial Eigenvalue Resolution

The ensemble-averaged eigenvalue resolution, λ_{res}, was calculated for each algorithm and can be seen in column four of Table 4.1. A lower λ_{res} indicates a lower error between a spectrally majorised version of the ground truth polynomial eigenvalues and those obtained from a PEVD. From this, it can be observed that the DaC approaches to the PEVD offer superior eigenvalue resolution versus SMD, despite the fact that all algorithms bar SBR2 achieve similar levels of diagonalisation. The slightly inferior diagonalisation performance of PSMD^1 relative to DC-SMD has translated to marginally higher λ_{res}. The poor diagonalisation performance of SBR2 has resulted in significantly worse resolution of the eigenvalues. Paired with its degraded diagonalisation performance, PSMD^2 has slightly higher λ_{res} than PSMD^1. While PSMD^3 achieves a similar level of diagonalisation to PSMD^1, the additional effort contributed towards the 'divide' stage has dramatically improved λ_{res}. For applications where resolution of the eigenvalues is of critical importance, it is clear that care must be taken to ensure that sufficient effort is spent block diagonalising an input parahermitian matrix $\boldsymbol{R}(z)$ prior to the 'conquer' stage in PSMD.

Experimental results indicate that SMD prioritises resolution of isolated eigenvalues with high power, and requires a high number of iterations to satisfactorily resolve closely spaced eigenvalues with low power, while the DaC methods attempt to resolve the eigenvalues more equally. This property of SMD has also been observed in [45] and can perhaps be explained by research in [11], which has noted that isolated eigenvalues are more easily estimated than closely spaced eigenvalues. Examples of eigenvalue resolution are demonstrated by Figure 4.16, which shows the on-diagonal PSDs of $\boldsymbol{D}(z)|_{z=e^{\mathrm{j}\Omega}}$ from SMD and PSMD^1 when applied to a single instance of the simulation scenario. For simplicity, only the first and last four of the 30 eigenvalues are shown, with a spectrally majorised version of the ground truth shown with dotted lines. The spectral majorisation of the polynomial eigenvalues output by iterative PEVD algorithms has been discussed in Section 2.3.2. A comparison of Figure 4.16(a) and (b) indicates that SMD offers slightly better resolution of the first four eigenvalues, while Figure 4.16(c) and (d) show that PSMD is more able to resolve the last four eigen-

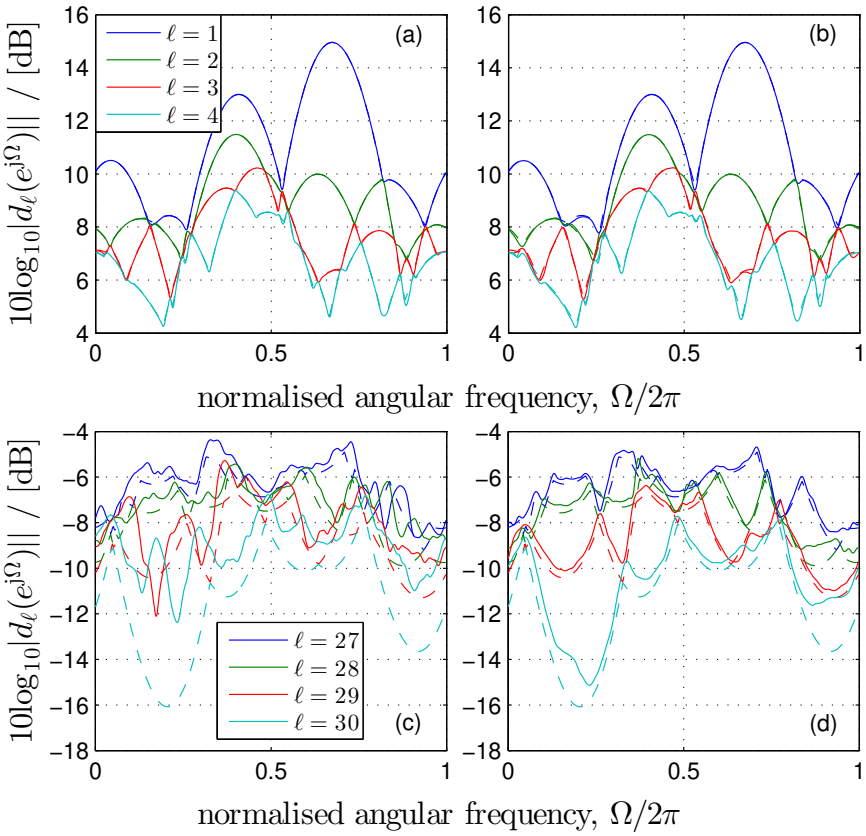

Figure 4.16: PSDs of the (a,b) first and (c,d) last four on-diagonal polynomials of $\boldsymbol{D}(z)$ obtained from (a,c) SMD and (b,d) PSMD^1 when applied to a single instance of the specified scenario, with ideal PSDs underlaid with dotted lines.

values. More accurately resolving eigenvalues of low power may be advantageous in some applications; for example, when attempting to estimate the noise-only subspace in broadband angle of arrival estimation scenarios.

Paraunitarity Error

The ensemble-averaged paraunitarity error for each algorithm can be seen in column five of Table 4.1. A lower η indicates that the product $\boldsymbol{F}(z)\tilde{\boldsymbol{F}}(z)$ closely approximates an identity matrix. Owing to their short run-time, low levels of truncation can be used for the DaC algorithms; this directly translates to low paraunitarity error. Conversely, high truncation is typically required to allow SBR2 and SMD to provide feasible run-times, resulting in higher η. The use of larger truncation parameters in PSMD^2 has

resulted in a significant increase in paraunitarity error, such that η is only slightly lower for PSMD^2 than SMD. Increasing I_D in PSMD^3 has slightly increased η, as more iterations of each 'divide' step — and therefore more truncation operations — are completed.

4.7.2 Source Model Simulation Scenario 2

This scenario provides additional evidence to confirm the ability of DaC methods to outperform existing PEVD algorithms as the spatial dimension of the input paraherMitian matrix increases. Simulations for this scenario are performed over an ensemble of 10^2 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $O_D = 118$, $O_Q = 60$, and $\Delta_{\text{DR}} = 30$.

The performance of the existing SMD [45] PEVD algorithm is compared with the novel PSMD algorithm. For performance metrics other than execution time, the performance of PSMD is representative of the performance of DC-SMD, so the latter is omitted from simulations for brevity. Using a polynomial matrix truncation parameter of $\mu_{\text{SMD}} = 10^{-6}$, SMD is executed until an input matrix is sufficiently diagonalised such that the off-diagonal energy in the output matrix equals one-tenth of the total energy in the matrix. PSMD is executed with input parameters $\mu_s \in \{0;\ 10^{-6};\ 10^{-9};\ 10^{-12}\}$, $\mu = \mu_t = 10^{-12}$, $\kappa \in \{2.5\times10^{-6}; 10^{-5}; 5\times10^{-5}; 10^{-4}\}$, $\epsilon = 0$, $I_D = 1000$, $I_C = 200$, $P = 10$, and $\hat{M} = 10$. These PSMD input parameters are deliberately chosen such that the algorithm achieves a diagonalisation level similar to that of SMD for all M. Note that the level of diagonalisation obtained by an instance of PSMD is mostly determined by the choice of 'conquer' stage parameters I_C and ϵ, as the amount of effort devoted to the 'divide' stage does not significantly impact diagonalisation performance — as seen in Figure 4.15.

At every iteration step of both algorithms, the performance metrics defined in Section A.3 are recorded together with the elapsed execution time.

Diagonalisation

The ensemble-averaged diagonalisation was calculated for the SMD and PSMD implementations. By evaluating the execution times required for both algorithms to achieve a diagonalisation level of $5\log_{10} E_{\text{diag}}^{(i)} \approx -10\,\text{dB}$, it is possible to directly compare the performance of both algorithms. Figure 4.17 uses the ratio of the execution times to demonstrate algorithm performance for various spatial dimensions, where

$$t_{\text{ratio}}(M,\kappa) = \frac{\mathcal{E}\{t_{\text{SMD}}(-10\,\text{dB}, M)\}}{\mathcal{E}\{t_{\text{PSMD}}(-10\,\text{dB}, M, \kappa)\}} \tag{4.53}$$

and $t_{\text{PSMD}}(-10\,\text{dB}, M, \kappa)$ denotes the time taken for one instance of PSMD to achieve a diagonalisation level of approximately $-10\,\text{dB}$ for some combination of M and κ. The value of $t_{\text{SMD}}(-10\,\text{dB}, M)$ similarly denotes the time taken for SMD.

From Figure 4.17, it is clear that $t_{\text{ratio}}(M,\kappa)$ increases with increasing spatial dimension M for all κ; i.e., the use of PSMD over SMD becomes more important the larger the matrix to be factorised. Indeed, $t_{\text{ratio}}(M, \kappa = 10^{-4})$ reaches a value of 64.4 for $M = 70$, signifying that PSMD is 64.4 times faster than SMD on average for this dimensionality and κ.

In Figure 4.17, it can be seen that the PSMD algorithm run-time increases with decreasing κ. Smaller values of κ lead to additional time being spent in the 'divide' stage of the algorithm and a more accurate block diagonalisation of the parahermitian matrix. Note that even for $\kappa = 2.5 \times 10^{-6}$, the PSMD algorithm is still faster than SMD.

Decomposition Mean Square Error

The ensemble-averaged decomposition mean square error was calculated for each algorithm, according to Section A.3. Figure 4.18 shows the results for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5 \times 10^{-6}; 10^{-5}; 5 \times 10^{-5}; 10^{-4}\}$, and $\mu_s = 0$. This plot confirms the results of Section 4.7.1 by indicating that the increased diagonalisation speed of PSMD typically comes at the cost of a higher decomposition error; however, for $\kappa = 2.5 \times 10^{-6}$, the decomposition MSE has fallen below that of SMD.

Of course, as evidenced by Figure 4.17, this lower threshold κ reduces the speed of the algorithm, as more effort is contributed to the 'divide' stage; however, PSMD is still faster than SMD for this choice of κ. Note that the relative difference in average MSE remains reasonably constant for increasing M. However, for $M = 10$, PSMD does not employ a 'divide' stage; without the requirement of a transformation of the input parahermitian matrix to block diagonal form, the low levels of truncation in PSMD lead to low MSE. The decomposition MSE of SMD can be decreased by decreasing polynomial matrix truncation parameter μ_{SMD} at the expense of even greater execution time requirements.

For PSMD, the decomposition MSE was recorded before and after the utilisation of CRST to estimate the method's impact. Figure 4.19 shows the results for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\mu_s \in \{0; 10^{-6}; 10^{-9}; 10^{-12}\}$, and $\kappa = 10^{-4}$; from this, it is clear that — as seen in Section 3.8.2 — the CRST method has little impact on reconstruction error.

Paraunitary Matrix Length

The ensemble-averaged paraunitary matrix length was calculated for each algorithm. Figure 4.20 shows the results for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$ and $\kappa \in \{2.5 \times 10^{-6}; 10^{-5}; 5 \times 10^{-5}; 10^{-4}\}$ without the use of CRST ($\mu_s = 0$). This plot indicates that the increased diagonalisation speed of PSMD typically comes at the cost of a higher paraunitary matrix length for all M. It can also be seen that decreasing threshold κ increases the length of the paraunitary matrix output by the algorithm. Note that the paraunitary matrix length increases with M for all algorithms. While PSMD is functionally very similar to SMD for the case of $M = 10$, as no 'division' occurs, the lower truncation parameter used in PSMD results in less significant truncation of the polynomial matrices internal to the algorithm and therefore a longer paraunitary matrix at the output.

For PSMD, the paraunitary matrix length was recorded before and after the utilisation of CRST to estimate the method's impact. Figure 4.21 shows the results for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\mu_s \in \{0; 10^{-6}; 10^{-9}; 10^{-12}\}$, and $\kappa = 10^{-4}$. It can

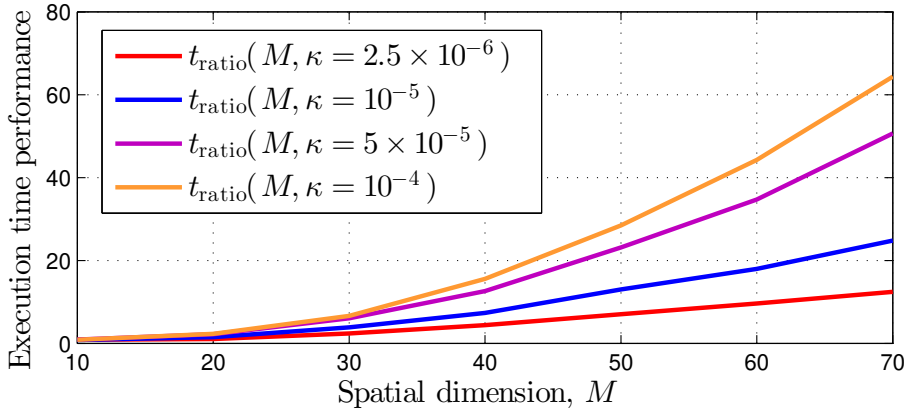

Figure 4.17: Ratio of SMD to PSMD algorithm execution time required to achieve $5\log_{10} E_{\text{diag}} \approx -10\,\text{dB}$ for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5\times10^{-6}; 10^{-5}; 5\times10^{-5}; 10^{-4}\}$, and $\mu_s = 0$.

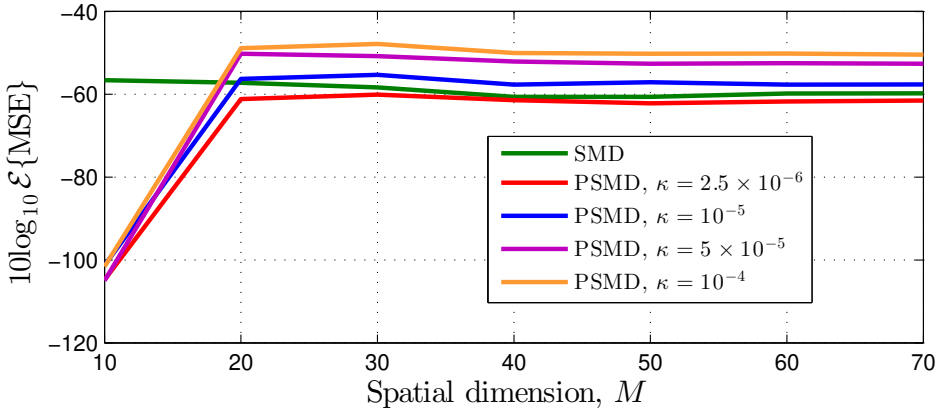

Figure 4.18: Decomposition mean square error versus spatial dimension M for SMD and PSMD for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5\times10^{-6}; 10^{-5}; 5\times10^{-5}; 10^{-4}\}$, and $\mu_s = 0$.

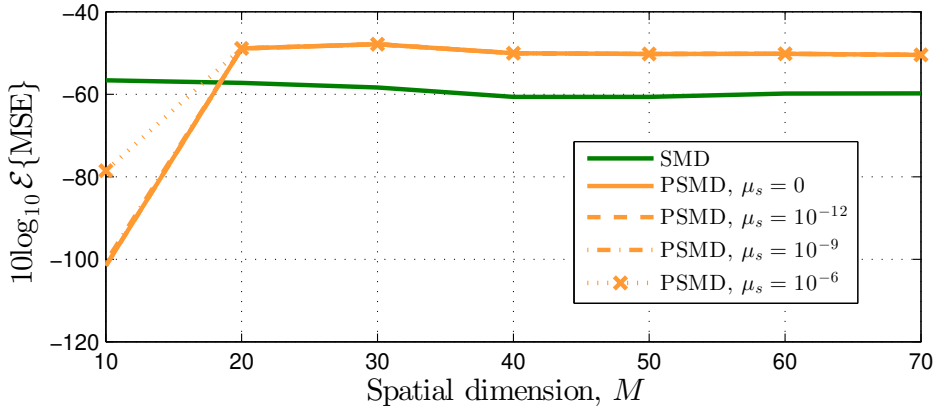

Figure 4.19: Decomposition mean square error versus spatial dimension M for SMD and PSMD for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\mu_s \in \{0; 10^{-6};\ 10^{-9};\ 10^{-12}\}$, and $\kappa = 10^{-4}$.

be seen from this graph that while the average paraunitary matrix length is still larger for PSMD than SMD for all M, the use of CRST has successfully reduced paraunitary matrix length. Increasing μ_s for the CRST method of truncation barely affected decomposition error in Figure 4.19; however, in Figure 4.21, a significant decrease in paraunitary matrix length is observed as μ_s is increased.

While larger paraunitary matrices are disadvantageous for application purposes, the increased performance of PSMD in other areas may be of greater importance.

Note that — as was found for a different row-shift truncation strategy in [63] — CRST was found to have minimal impact when applied to the paraunitary matrices generated by SMD.

Polynomial Eigenvalue Resolution

The ensemble-averaged eigenvalue resolution, λ_{res}, was calculated for each algorithm. Figure 4.22 shows the results for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5 \times 10^{-6}; 10^{-5}; 5 \times 10^{-5}; 10^{-4}\}$, and $\mu_s = 0$. This plot indicates that while PSMD tends to introduce a greater degree of error to the decomposition, it is more able to resolve the polynomial eigenvalues for all M. As κ is decreased, the decomposition MSE decreases and the energy that would be thrown away during the 'divide' stage for larger κ is instead transferred to the polynomial eigenvalues, and λ_{res} decreases as a result. While SMD is functionally very similar to PSMD for $M = 10$ — since PSMD does not use a 'divide' stage for $M \leq \hat{M}$ — the increased level of polynomial matrix truncation in the former has resulted in significantly poorer eigenvalue resolution for this dimensionality.

Interestingly, λ_{res} for the SMD algorithm decreases with M, becoming similar to that obtained for PSMD with $\kappa = 10^{-4}$. However, the time required for SMD to achieve this level of performance — which is of the order of 150 seconds — suggests that PSMD is more suitable if low eigenvalue resolution is required within a reasonable time frame.

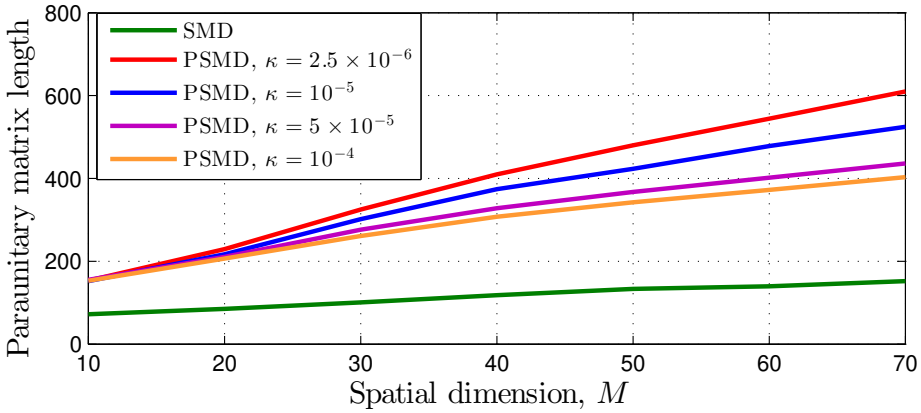

Figure 4.20: Paraunitary matrix length versus spatial dimension M for SMD and PSMD for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5 \times 10^{-6}; 10^{-5}; 5 \times 10^{-5}; 10^{-4}\}$, and $\mu_s = 0$.

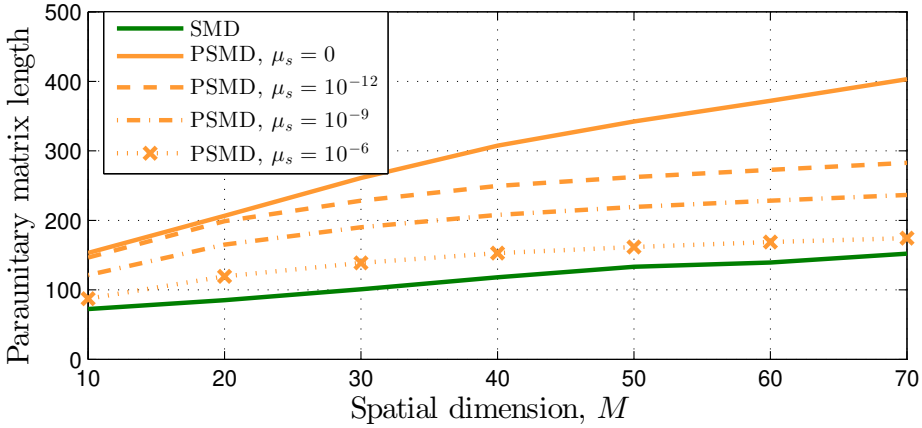

Figure 4.21: Paraunitary matrix length versus spatial dimension M for SMD and PSMD for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\mu_s \in \{0; 10^{-6};\ 10^{-9};\ 10^{-12}\}$, and $\kappa = 10^{-4}$.

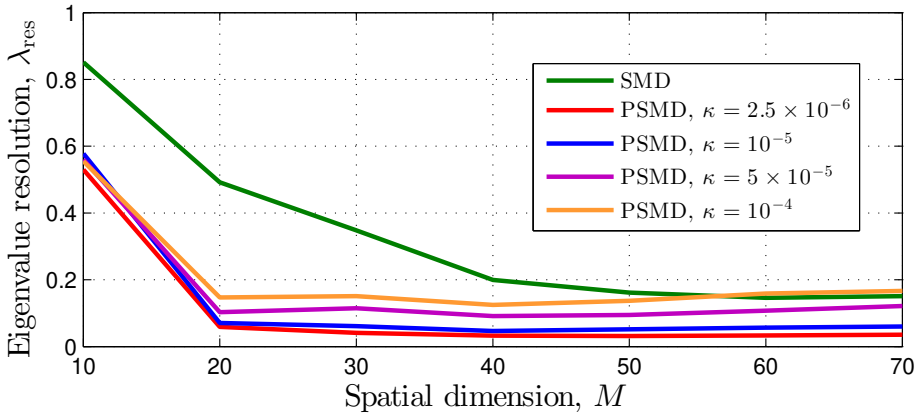

Figure 4.22: Eigenvalue resolution versus spatial dimension M for SMD and PSMD for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5 \times 10^{-6}; 10^{-5}; 5 \times 10^{-5}; 10^{-4}\}$, and $\mu_s = 0$.

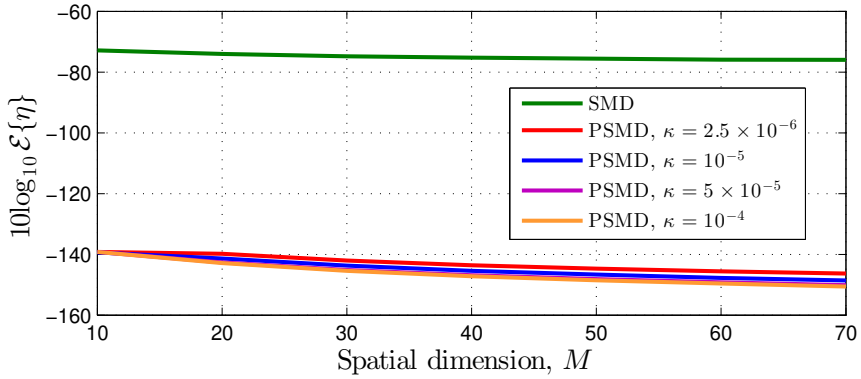

Figure 4.23: Paraunitarity error versus spatial dimension M for SMD and PSMD for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5 \times 10^{-6}; 10^{-5}; 5 \times 10^{-5}; 10^{-4}\}$, and $\mu_s = 0$.

Paraunitarity Error

The ensemble-averaged decomposition paraunitarity error, η, was calculated for each algorithm, according to Section A.3. Figure 4.23 shows the results for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\kappa \in \{2.5 \times 10^{-6}; 10^{-5}; 5 \times 10^{-5}; 10^{-4}\}$, and $\mu_s = 0$. The increased diagonalisation speed of PSMD means that lower levels of polynomial matrix truncation can be used and the resulting η is low. SMD requires high truncation to converge within a reasonable time frame — especially when M is large; thus the resulting η is significantly higher.

For PSMD, the decomposition η was recorded before and after the utilisation of CRST to estimate the method's impact. Figure 4.24 shows the results for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\mu_s \in \{0; 10^{-6}; 10^{-9}; 10^{-12}\}$, and $\kappa = 10^{-4}$. Increasing μ_s has a clear detrimental impact on η; however, this increase may be worthwhile when considering the paraunitary matrix length reduction offered by CRST in Figure 4.21. Even for $\mu_s = 10^{-6}$, the decomposition paraunitarity error of SMD is still higher than PSMD for all M.

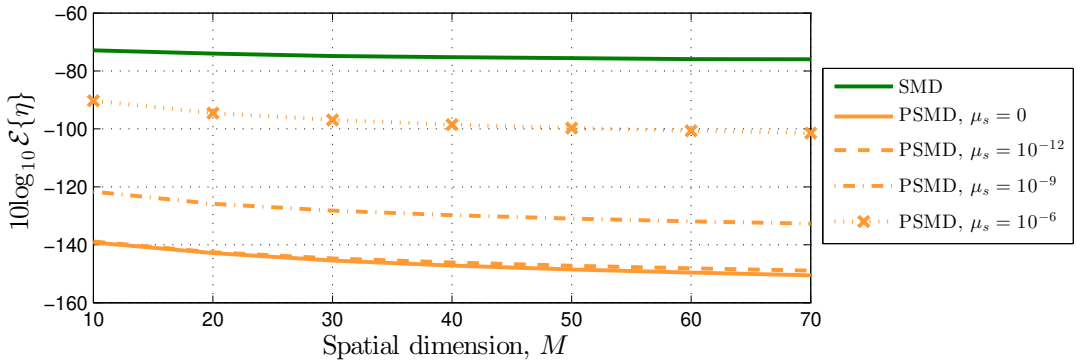

Figure 4.24: Paraunitarity error η versus spatial dimension M for SMD and PSMD for $M \in \{10;\ 20;\ 30;\ 40;\ 50;\ 60;\ 70\}$, $\mu_s \in \{0; 10^{-6};\ 10^{-9};\ 10^{-12}\}$, and $\kappa = 10^{-4}$.

4.7.3 Broadband Angle of Arrival Estimation Simulation Scenario

Problem Formulation

For broadband angle of arrival (AoA) estimation, powerful narrowband methods such as the multiple signal classification (MUSIC) algorithm [77] are not directly applicable. In [18], the PEVD is used to generalise MUSIC to the case of polynomial space-time covariance matrices, resulting in the development of the spatio-spectral polynomial MUSIC (SSP-MUSIC) algorithm. A comparison in [17] of SSP-MUSIC with an auto-focussing coherent signal subspace AoA estimation approach [107] has found that SSP-MUSIC provides lower AoA estimation performance, but has the advantage of not relying on a priori spectral information of the sources. Further work in [19] has shown that the accuracy of SSP-MUSIC depends strongly on the efficacy of the PEVD algorithm used.

For a parahermitian matrix $\boldsymbol{R}(z)$ with polynomial eigenvalues $\boldsymbol{D}(z) = \operatorname{diag}\{d_1(z), d_2(z), \ldots, d_M(z)\}$, thresholding the polynomial eigenvalues — e.g., extracting eigenvalues with energy $\sum_\tau |d_i[\tau]|^2 > \lambda$ for some λ — reveals the number of independent broadband sources R contributing to $\boldsymbol{R}(z)$. This permits a distinction between signal-plus-noise and noise-only subspaces $\boldsymbol{F}_s(z) : \mathbb{C} \to \mathbb{C}^{R\times M}$ and

$\boldsymbol{F}_n(z) : \mathbb{C} \to \mathbb{C}^{(M-R)\times M}$,

$$\boldsymbol{R}(z) \approx \begin{bmatrix} \tilde{\boldsymbol{F}}_s(z) & \tilde{\boldsymbol{F}}_n(z) \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_s(z) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D}_n(z) \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_s(z) \\ \boldsymbol{F}_n(z) \end{bmatrix}, \tag{4.54}$$

where $R < M$, $\boldsymbol{D}_s(z) : \mathbb{C} \to \mathbb{C}^{R\times R}$ and $\boldsymbol{D}_n(z) : \mathbb{C} \to \mathbb{C}^{(M-R)\times(M-R)}$. The SSP-MUSIC algorithm seeks to scan the noise-only subspace $\boldsymbol{F}_n(z)$ of $\boldsymbol{R}(z)$, which is spanned by eigenvectors corresponding to eigenvalues close to the noise floor. The steering vectors of sources that contribute to $\boldsymbol{R}(z)$ will define the signal-plus-noise subspace $\boldsymbol{F}_s(z)$ and therefore lie in the nullspace of $\boldsymbol{F}_n(z)$. More detail on how the SSP-MUSIC algorithm operates can be found in Appendix D.

Note that the structure of (4.54) is very similar to the block diagonal decomposition of a parahermitian matrix in (4.4); i.e., consider the form

$$\boldsymbol{R}(z) \approx \begin{bmatrix} \tilde{\boldsymbol{T}}_s(z) & \tilde{\boldsymbol{T}}_n(z) \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_s(z) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_n(z) \end{bmatrix} \begin{bmatrix} \boldsymbol{T}_s(z) \\ \boldsymbol{T}_n(z) \end{bmatrix}, \tag{4.55}$$

where paraunitary matrix $\tilde{\boldsymbol{T}}(z) = \begin{bmatrix} \tilde{\boldsymbol{T}}_s(z) & \tilde{\boldsymbol{T}}_n(z) \end{bmatrix}$ does not contain the eigenvectors of $\boldsymbol{R}(z)$, but instead contains the orthonormal basis that transforms $\boldsymbol{R}(z)$ into the block diagonal $\boldsymbol{B}(z) = \text{diag}\{\boldsymbol{B}_s(z), \boldsymbol{B}_n(z)\}$. If an algorithm that enforces spectral majorisation of the underlying polynomial eigenvalues is used, such as SMS in Section 4.4, the eigenvalues of $\boldsymbol{B}_s(z) : \mathbb{C} \to \mathbb{C}^{R\times R}$ should be those corresponding to signal-plus-noise, and the eigenvalues of $\boldsymbol{B}_n(z) : \mathbb{C} \to \mathbb{C}^{(M-R)\times(M-R)}$ should be those corresponding to noise. Since $\boldsymbol{B}_s(z)$ and $\boldsymbol{B}_n(z)$ are independent parahermitian matrices and $\boldsymbol{T}(z)$ is paraunitary, the signal-plus-noise subspace $\boldsymbol{T}_s(z)$, which also contains the steering vectors of sources that contribute to $\boldsymbol{R}(z)$, lies in the nullspace of $\boldsymbol{T}_n(z)$. The SSP-MUSIC algorithm can therefore instead scan the noise-only subspace $\boldsymbol{T}_n(z)$ to identify the presence of source steering vectors in $\boldsymbol{R}(z)$. The results of Section 4.4.4 have shown that the iterative block diagonalisation of a parahermitian matrix is possible. If, in practice, the block diagonalisation of some $\boldsymbol{R}(z)$ is less computationally expensive or faster than its diagonalisation through the use of the PEVD, then a decomposition of

the form in (4.55) may be preferable for applications involving AoA estimation. Note that eigenvalues $\boldsymbol{D}_s(z)$ and $\boldsymbol{D}_n(z)$ can be obtained from $\boldsymbol{B}_s(z)$ and $\boldsymbol{B}_n(z)$ if desired; however, this is not always required for the purposes of AoA estimation.

For a parahermitian matrix with large spatial dimensions, which traditional PEVD algorithms would typically struggle to diagonalise, PSMD can be used (with $I_C = 0$) to transform the matrix into block diagonal form. Through smart choice of input parameters $\hat{M}$ and P, the signal-plus-noise subspace can be isolated, and the noise-only subspace can be used in SSP-MUSIC. For example, if 10 sources are known to contribute to a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{20\times 20}$, then PSMD can be executed using parameters $\hat{M} = 10$, $P = 5$, and $I_C = 0$ such that the first and last 10 rows of output matrix $\boldsymbol{F}(z)$ correspond to the signal-plus-noise and noise-only subspaces, respectively. Given the relatively low cost of the 'conquer' stage of the PSMD algorithm, and the low likelihood that a block diagonal form will perfectly separate signal-plus-noise and noise-only subspaces, it is typically worthwhile to provide PSMD with at least a small value for I_C. Using a relatively large stopping threshold ϵ for the 'conquer' stage would then prevent the PSMD from spending time diagonalising parahermitian matrices containing only noise.

Simulation Scenario

In this simulation scenario, the AoA estimation performance that PSMD offers when paired with SSP-MUSIC is compared with the performance that SBR2 and SMD provide in a broadband AoA estimation scenario. Again, DC-SMD is omitted for brevity. Performance of each PEVD algorithm is measured in terms of AoA estimation accuracy and with the metrics of Section A.3. AoA estimation accuracy is measured through a visual assessment of metric $P_{\text{SSP}}(\vartheta, \Omega) \in \mathbb{R}_{>0}$ output by the SSP-MUSIC algorithm in [18], which is evaluated at a range of angles of arrival, ϑ, and frequencies, Ω. A high value of $P_{\text{SSP}}(\vartheta, \Omega)$ indicates the presence of a source at a specific angle of arrival and frequency, while a low value indicates that no source is present.

In the simulations below, an $M = 20$ element array is illuminated by six broadband sources active over a frequency range $\Omega_j \in [0.1\pi, 0.9\pi]$, $j = 1 \ldots 6$, and different angles

of arrival $\vartheta_j \in \{\pm 22.5°; \pm 45°; \pm 63°\}$. For simplicity, the elevation angles of the sources are equal to zero. The array signals are corrupted by uncorrelated independent and identically distributed complex Gaussian noise at a signal-to-noise ratio of 20 dB. To exclude error sources other than inaccuracies in the subspace identification, the source data is modelled as a sum of closely spaced sinusoids, with randomised phases, of length 64000 samples, for which highly accurate narrowband steering vectors can be used. Space-time covariance matrix $\mathbf{R}[\tau]$ is estimated for $|\tau| \leq 20$. The broadband steering vectors that the SSP-MUSIC algorithm uses to scan the noise-only subspace are based on fractional delay filters constructed from truncated sinc functions [108,109].

One instance of PSMD, denoted PSMD^1, is executed with input parameters $\mu = \mu_t = 10^{-12}$, $\mu_s = 10^{-6}$, $\kappa = 5 \times 10^{-6}$, $\epsilon = 10^{-2}$, $I_D = 1000$, $I_C = 0$, $\hat{M} = 6$, and $P = 7$. For the above simulation scenario, this choice of P and $\hat{M}$ will exactly isolate the signal-plus-noise subspace through block diagonalisation of the parahermitian matrix; thus, no iterations of the 'conquer' stage are necessary. A second instance of PSMD, denoted PSMD^2, is executed with input parameters $\mu = \mu_t = 10^{-12}$, $\mu_s = 10^{-6}$, $\kappa = 10^{-5}$, $\epsilon = 10^{-2}$, $I_D = 1000$, $I_C = 250$, and $\hat{M} = P = 10$. This choice of algorithm parameters is more suitable for an unknown AoA estimation scenario. For the above scenario, the block diagonalisation process will not adequately isolate the signal-plus-noise subspace, but the allocation of 250 iterations to each component of the 'conquer' stage will facilitate the identification of the noise-only subspace. A relatively high ϵ will prevent the algorithm from spending time diagonalising parahermitian matrices containing only noise. Both instances of PSMD are provided with a high value of I_D, such that convergence parameter κ determines the length of each 'divide' step.

During iterations of SMD and SBR2, convergence parameters of $\epsilon_{\text{SMD}} = \epsilon_{\text{SBR2}} = 0$ and polynomial matrix truncation parameters of $\mu_{\text{SMD}} = \mu_{\text{SBR2}} = 10^{-6}$ are used. SMD and SBR2 are run until their execution times match the time taken for the completion of each instance of PSMD, which is designed to be approximately one second. At every iteration step of each PEVD algorithm, the performance metrics defined in Section A.3 are recorded together with the elapsed execution time.

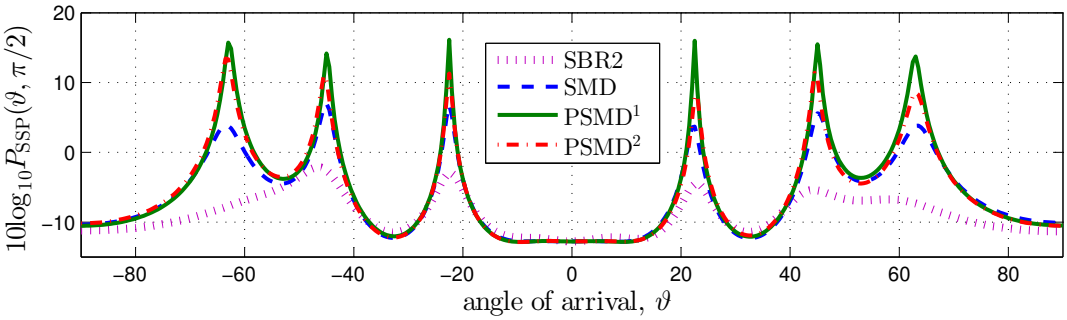

Figure 4.25: Performance of SSP-MUSIC at $\Omega = \pi/2$ based on PSMD^1, PSMD^2, SBR2, and SMD for a scenario with six independent broadband sources.

PEVD Algorithm Comparison

The plot of Figure 4.25 shows the SSP-MUSIC performance at a frequency of $\Omega = \pi/2$; from this, it can be observed that PSMD^1 offers superior localisation of the sources. Since PSMD^1 does not require any computational effort to be spent on the diagonalisation of parahermitian matrices, additional time can be allocated to the block diagonalisation process. The result is a superior isolation of the signal-plus-noise subspace, and therefore a better identification of the orthonormal basis corresponding to noise used by the SSP-MUSIC algorithm. Of course, the parameters of PSMD^1 have been specifically chosen for this simulation scenario, which is not realistic. The more general purpose approach of PSMD^2 is still able to significantly outperform SBR2 and SMD for the localisation of sources at this frequency.

The results of Figure 4.26 reiterate the above by demonstrating that the PSMD^1 algorithm is capable of outperforming all other methods at most frequencies when each is paired with SSP-MUSIC. Clearly, for time-limited AoA estimation scenarios with large M, divide-and conquer methods can produce superior results.

Table 4.2 compares the metrics attributed to each decomposition for this simulation scenario. It can be observed that while SMD and PSMD^2 obtain superior diagonalisation to PSMD^1, this does not translate to better AoA estimation performance. Both instances of PSMD result in higher decomposition MSE than SMD and SBR2; however, the instances of PSMD — which utilise lower levels of truncation — provide slightly lower paraunitarity error. PSMD^1 outputs the shortest paraunitary matrix,

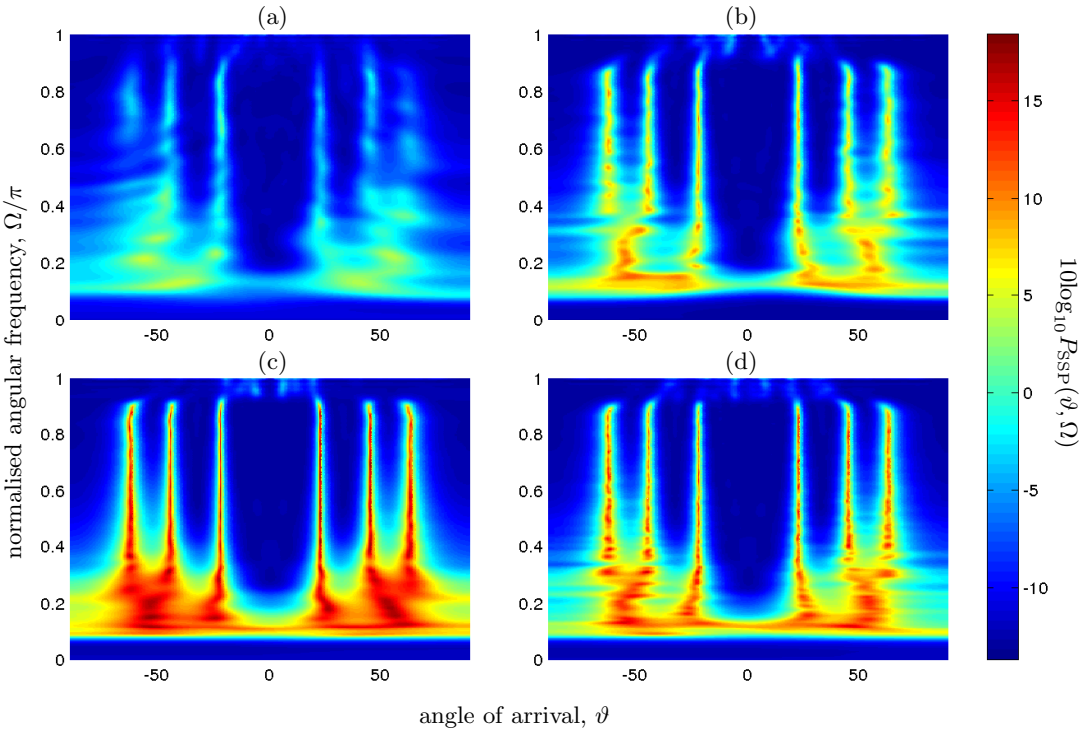

Figure 4.26: Performance of SSP-MUSIC based on: (a) SBR2; (b) SMD; (c) PSMD with $\kappa = 5 \times 10^{-6}$, $\hat{M} = 6$, $P = 7$, and $I_C = 0$; and (d) PSMD with $\kappa = 10^{-5}$, $\hat{M} = P = 10$, and $I_C = 250$ for a scenario with six independent broadband sources.

but PSMD^2, which utilises both 'divide' and 'conquer' stages, produces a paraunitary matrix of greater length than the other methods.

4.8 Conclusions

This chapter has proposed an alternative, divide-and-conquer (DaC) strategy for computing the polynomial matrix EVD of a parahermitian matrix. By embracing this methodology, the historically sequential nature of PEVD algorithms has been converted to a partially parallelisable one. Following the introduction of a structured approach to the block diagonalisation of a parahermitian matrix, a novel iterative algorithm named sequential matrix segmentation (SMS) has been shown to be capable of facilitating this goal.

Two further novel algorithms developed in this chapter — named divide-and-conquer sequential matrix diagonalisation (DC-SMD) and parallel-sequential matrix diagonali-

Table 4.2: Performance metric comparison for PSMD[1], PSMD[2], SBR2, and SMD in AoA estimation scenario. Metrics used: final algorithm diagonalisation, E_{diag}; length of paraunitary matrix $\boldsymbol{F}(z)$, L_F; decomposition mean square error, MSE; and η.

Algorithm	$5\log_{10} E_{\text{diag}}$	L_F	MSE	η
PSMD[1]	-7.76 dB	140	1.58×10^{-6}	2.94×10^{-10}
PSMD[2]	-12.2 dB	285	1.44×10^{-6}	1.59×10^{-10}
SBR2 [6]	-5.61 dB	148	2.01×10^{-9}	2.77×10^{-9}
SMD [45]	-9.49 dB	184	9.86×10^{-9}	2.00×10^{-8}

sation (PSMD) — make use of SMS within a DaC approach to the PEVD, and have been shown to offer several advantages over existing iterative PEVD algorithms. When compared with the SBR2 [6] and SMD [45] algorithms, PSMD offers a significant decrease in run-time and paraunitarity error — and provides superior eigenvalue resolution — but typically results in higher mean square reconstruction error and paraunitary matrix length. However, the latter can be reduced at the expense of higher paraunitarity error. Simulation results have also demonstrated that when paired with the SSP-MUSIC broadband AoA estimation algorithm, PSMD offers significant performance gains over traditional PEVD algorithms at the expense of increased paraunitary matrix length and decomposition error.

Further simulation results have demonstrated that the low algorithmic complexity and parallelised nature of PSMD results in lower algorithm run-time than DC-SMD, with the advantage of decreasing the paraunitary matrix length. The range of input parameters of both algorithms imbues them with a large degree of operational flexibility. An investigation into the performance trade-offs of PSMD has shown that through careful choice of algorithm input parameters, a balance can be obtained between decomposition MSE, algorithm execution time, matrix paraunitarity, paraunitary matrix length, and eigenvalue resolution. The presence of these trade-offs is important for the implementation of PSMD in broadband multichannel applications.

When designing PEVD implementations for such applications — particularly those involving a large number of sensors — the potential for the proposed DaC approach to increase diagonalisation speed while reducing complexity requirements offers benefits.

In addition, the parallelisable nature of PSMD, which has been exploited here to reduce algorithm run-time, is well suited to hardware implementation. For applications involving broadband angle of arrival estimation, the short run-time of PSMD will decrease the time between estimations of source locations and bandwidths; similarly, use of PSMD will allow for signal of interest and interferer locations and bandwidths to be updated more quickly in broadband beamforming applications. Furthermore, the low paraunitarity error of PSMD, which facilitates the implementation of near-lossless filter banks, is advantageous for communications applications.

Chapter 5

DFT-Based Alternatives for PEVD Algorithms

5.1 Introduction

The previous chapters have dealt with a number of iterative PEVD algorithms that manipulate polynomial coefficients directly to diagonalise a parahermitian matrix. Such algorithms — represented in [6, 45–50] — form a significant proportion of the available methods for computing the PEVD. This chapter instead investigates the less-explored area of PEVD algorithms based on the discrete Fourier transform (DFT). Following the introduction of a capable DFT-based PEVD (and PSVD) algorithm in [53], there has been little research in this field; indeed, only very brief relative comparisons between iterative and DFT-based methods had been conducted prior to the research discussed in this chapter. While the algorithm of [53] has in almost all cases been demonstrated to produce a compact, accurate PEVD, it requires a priori knowledge of the order of the polynomial eigenvectors in the decomposition, and employs an eigenvalue reordering scheme that is susceptible to errors in the presence of eigenvalues with algebraic multiplicities greater than one.

For future reference, the general structure of a DFT-based PEVD algorithm is provided below. If the PEVD equation of (2.4) is evaluated at K discrete points on

the unit circle, the following formulation can be obtained:

$$\mathbf{R}[k] = \mathbf{Q}[k]\mathbf{D}[k]\mathbf{Q}^{\mathrm{H}}[k]\,,\ k = 0 \dots K-1\,, \tag{5.1}$$

where $\mathbf{R}[k] = \boldsymbol{R}(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega_k}}$ and $\Omega_k = 2\pi k/K$. In (5.1), $\mathbf{Q}[k]$ is a unitary matrix containing eigenvectors in its columns, $\mathbf{D}[k]$ is a diagonal matrix containing eigenvalues, and $\mathbf{R}[k]$ is obtained from the K-point DFT of $\mathbf{R}[\tau] \in \mathbb{C}^{M\times M}$,

$$\mathbf{R}[k] = \boldsymbol{R}(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega_k}} = \sum\nolimits_{\tau} \mathbf{R}[\tau]\mathrm{e}^{-\mathrm{j}\Omega_k\tau}\,,\ k = 0 \dots K-1\,, \tag{5.2}$$

where $K \geq L$ and L is the length of $\boldsymbol{R}(z)$. Since each frequency bin k contains an independent Hermitian matrix $\mathbf{R}[k]$, an approximate PEVD is therefore obtained via K independent EVDs.

Noting that the decompositions in (2.4) and (5.1) are only unique up to permutations and phase shifts [9], an advantage of DFT-based approaches is the option to rearrange the eigenvalues and eigenvectors at each frequency bin if desired. If the eigenvalues in $\mathbf{D}[k]$ are arranged in descending order, approximate spectral majorisation of the resulting polynomial eigenvalues occurs [53]. An analytic decomposition, whereby both eigenvalues and eigenvectors are continuous in frequency, typically produces a more compact decomposition, but requires additional effort. Spectral majorisation is preferable to an analytic decomposition in only a limited number of applications, e.g., when the extraction of multiple-input multiple-output (MIMO) subchannels of ordered quality [110] matters. Figure 5.1 shows the PSDs of analytic and spectrally majorised eigenvalues for an example parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{2\times 2}$.

A disadvantage of DFT-based algorithms involves the inherent lack of phase coherence between independent frequency bins [81]. Each eigenvector at each frequency bin can be influenced by an arbitrary scalar phase angle and still be valid. Discontinuities in phase between adjacent frequency bins can arise due to this ambiguity in eigenvector phase. To obtain a short paraunitary matrix $\boldsymbol{Q}(z)$, these discontinuities must be smoothed by enforcing phase coherence between frequency bins [9]. In [53], this is achieved through the use of a phase alignment function, described in Section A.2.3,

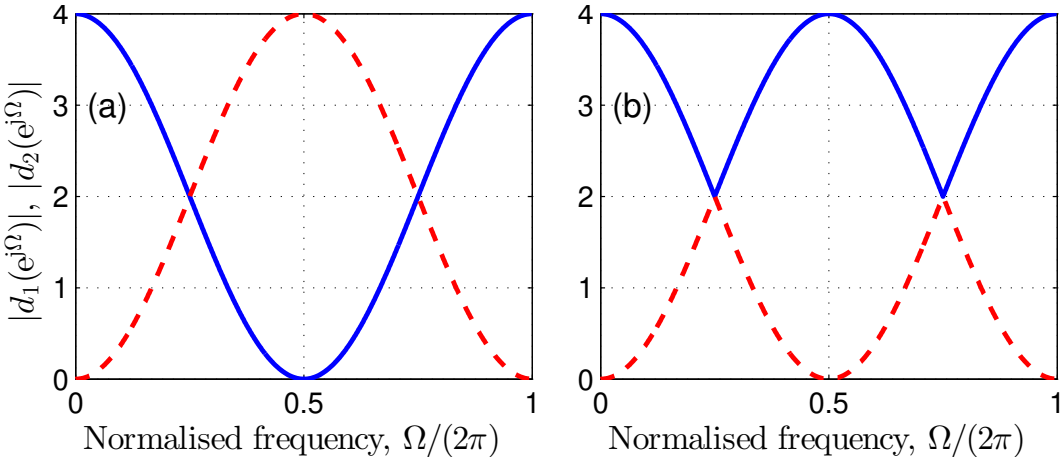

Figure 5.1: Example for $\boldsymbol{D}(z) : \mathbb{C} \to \mathbb{C}^{2\times 2}$ with (a) analytic and (b) spectrally majorised eigenvalues.

which uses Powell's 'dogleg' algorithm [111,112] to solve an unconstrained optimisation problem.

Following the permutation (if desired) and phase alignment of $\mathbf{Q}[k]$, $\mathbf{Q}[\tau]$ is computed via the inverse DFT (IDFT) as

$$\mathbf{Q}[\tau] = \frac{1}{K}\sum_{k=0}^{K-1} \mathbf{Q}[k] e^{j\Omega_k \tau}\,,\ \tau = 0 \ldots K-1\,, \tag{5.3}$$

and $\mathbf{D}[\tau]$ is found in a similar fashion or by extracting the diagonal elements of the product $\tilde{\boldsymbol{Q}}(z)\boldsymbol{R}(z)\boldsymbol{Q}(z)$. Matrix $\boldsymbol{Q}(z)$ therefore contains polynomial eigenvectors in its columns, and $\boldsymbol{D}(z)$ contains polynomial eigenvalues on its diagonal.

While analytic polynomial eigenvalues and eigenvectors have been shown to exist as absolutely convergent Laurent series in [9], there is currently no way of knowing the length of the series a priori. When converting $\mathbf{D}[k]$ and $\mathbf{Q}[k]$ to the lag domain, the order of the IDFT restricts the series' length to K. For an insufficient length K, this can result in time domain aliasing, whereby energy from ignored high order polynomial coefficients is wrapped around an internal period of length K. This introduces an error into the decomposition, and effectively limits the range of applications in which DFT-based PEVD algorithms can be used to those of known order; i.e., a suitably high K must be known a priori.

Algorithm 11 summarises the above steps of a generic DFT-based PEVD algorithm.

Input: $\boldsymbol{R}(z)$, K, υ
Output: $\boldsymbol{D}(z)$, $\boldsymbol{Q}(z)$
for $k \leftarrow K-1, \ldots, 1, 0$ **do**
  Compute $\mathbf{R}[k]$ according to DFT in (5.2)
  Compute $\mathbf{Q}[k]$ and $\mathbf{D}[k]$ via ordered EVD of $\mathbf{R}[k]$
end
if $\upsilon = 1$ **then**
  Permute $\mathbf{Q}[k]$ and $\mathbf{D}[k]$ to approximate an analytic decomposition
end
Use phase alignment strategy to promote coherence between frequency bins in $\mathbf{Q}[k]$
Obtain $\mathbf{Q}[\tau]$ from IDFT of $\mathbf{Q}[k]$
Obtain $\mathbf{D}[\tau]$ from IDFT of $\mathbf{D}[k]$ or via $\boldsymbol{D}(z) \leftarrow \operatorname{diag}\Big\{\operatorname{diag}\Big\{\tilde{\boldsymbol{Q}}(z)\boldsymbol{R}(z)\boldsymbol{Q}(z)\Big\}\Big\}$

Algorithm 11: General structure of a DFT-based PEVD algorithm

Here, a user-input parameter υ determines if the algorithm approximates an analytic decomposition.

Below, Section 5.2 provides a qualitative and quantitative comparison of the DFT-based PEVD algorithm of [53] with the well-established SMD algorithm of [45]. Using knowledge gained during this analysis, Section 5.3 introduces a novel DFT-based algorithm that requires less a priori information than the algorithm of [53], while still providing an accurate PEVD. This algorithm also utilises a modified eigenvalue reordering scheme that is less susceptible to errors in the presence of eigenvalues with an algebraic multiplicity greater than one at a particular frequency bin. In an effort to remove the requirement of any a priori information and increase the applicability of the DFT-based PEVD, Section 5.4 formulates a second novel algorithm that uses an iterative DFT-based approach to the PEVD that is able to decompose any parahermitian matrix without prior knowledge of K. Section 5.5 demonstrates that DFT-based approaches can facilitate the acquisition of the minimum-order solution of a PEVD, which has maximally short polynomial eigenvectors. Such a solution, which is not currently obtainable for any of the existing iterative PEVD algorithms, would be ideal for implementation purposes, where limitations are often placed on paraunitary matrix length for storage and complexity reasons. Conclusions for this chapter are provided in Section 5.6.

5.2 Comparison of Iterative and DFT-Based PEVDs

This section compares the decomposition accuracies of two fundamentally different methods capable of computing an approximate PEVD. The first of these — sequential matrix diagonalisation (SMD) [45] — iteratively decomposes a parahermitian matrix by directly manipulating polynomial coefficients and is described in Section A.1, while a second algorithm introduced in [53] and summarised in Section A.2 computes a DFT-based decomposition using the formulation in (5.1).

Following a comparison of the algorithmic complexities of each decomposition in Section 5.2.1, the accuracies of the polynomial eigenvalues generated by each is discussed in Section 5.2.2. The paraunitarity error induced by each algorithm is then reviewed in Section 5.2.3, and simulations in Section 5.2.4 are used to compare the performance of each strategy with respect to the metrics of Section A.3. From the comparisons conducted in these sections, Section 5.2.5 is able to provide concluding remarks that indicate the type of broadband multichannel problems that are better suited to each algorithm.

Elements of the work in this section can be found in a paper titled 'A Comparison of Iterative and DFT-Based Polynomial Matrix Eigenvalue Decompositions' [56] published in the proceedings of the 7th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, and in a paper titled "Correction to 'On the Uniqueness and Existence of the Eigenvalue Decomposition of a Parahermitian Matrix'" [10] published in IEEE Transactions on Signal Processing.

5.2.1 Algorithm Complexities

At the ith iteration of SMD, every matrix-valued coefficient in $\boldsymbol{S}^{(i)\prime}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ must be left- and right-multiplied with unitary matrix $\mathbf{Q}^{(i)} \in \mathbb{C}^{M\times M}$; similarly $\boldsymbol{H}^{(i)\prime}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ is left-multiplied with $\mathbf{Q}^{(i)}$. A total of $2L\{\boldsymbol{S}^{(i)\prime}(z)\}$ and $L\{\boldsymbol{H}^{(i)\prime}(z)\}$ matrix multiplications are therefore required to update $\boldsymbol{S}^{(i)\prime}(z)$ and $\boldsymbol{H}^{(i)\prime}(z)$, respectively, where operator $L\{\cdot\}$ measures the length of a polynomial matrix. While potentially lower complexity matrix multiplication methods have been derived with complexities of, e.g., $\mathcal{O}(M^{2.807})$ [2, 113] and $\mathcal{O}(M^{2.3728639})$ [114] for the multiplication of two

square matrices with spatial dimension M, these often require very large M to outperform more traditional methods [95]. A simpler, naive assumption estimates that the complexity of the multiplication of two $M \times M$ matrices is of order $\mathcal{O}(M^3)$ [2,95]; thus, it can be approximated that the complexity of one SMD iteration is of order $\mathcal{O}(M^3(2L\{\boldsymbol{S}^{(i)\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime}(z)\})) \approx \mathcal{O}(M^3 L)$ — if it is also assumed that the lengths of these internal matrices are ultimately proportional to the length, L, of the input parahermitian matrix $\boldsymbol{R}(z)$. The update step dominates the complexity of SMD [45]; thus, for a constant number of algorithm iterations, the algorithm complexity is of order $\mathcal{O}(M^3 L)$.

In the DFT-based algorithm, for a constant number of optimisation steps, the complexity of each of the M instantiations of the phase alignment method is $\mathcal{O}(K^3)$ due to matrix inversion [53,95]; thus, the total complexity of the phase alignment step is of order $\mathcal{O}(MK^3)$. Given that K is bounded from below by L in (A.15) for constant N, $\mathcal{O}(MK^3)$ can be expressed as $\mathcal{O}(ML^3)$. The computation of the frequency domain representation of $\boldsymbol{R}(z)$, the execution of K EVDs, and the reordering of the eigenvalues and eigenvectors are of lower complexity than this step; thus, the total complexity of the algorithm is approximately $\mathcal{O}(ML^3)$.

5.2.2 Approximation of Eigenvalues

The SMD algorithm iteratively diagonalises a parahermitian matrix $\boldsymbol{R}(z)$; thus, the approximation of the polynomial eigenvalues becomes better with each algorithm iteration. Almost exact diagonalisation of $\boldsymbol{R}(z)$ typically requires a large number of iterations; this can be problematic, as the parahermitian matrix grows in order at each iteration of SMD [45]. Truncation of the outer lags of the matrix containing lower energy can help mitigate this growth, but risks the introduction of error into the polynomial eigenvalues and resulting PEVD. The polynomial eigenvalues produced by SMD are approximately spectrally majorised [45], and cannot be reordered. As mentioned in Section 5.1, this is only beneficial for a subset of applications.

Matrix $\boldsymbol{D}(z)$ is computed via the product $\tilde{\boldsymbol{Q}}(z)\boldsymbol{R}(z)\boldsymbol{Q}(z)$ and is set to be exactly diagonal in the final step of the DFT-based approach [53]; thus, diagonalisation metric

$E_{\text{diag}} = 0$ for all instances of the algorithm. However, for insufficiently accurate eigenvectors that do not completely diagonalise $\boldsymbol{R}(z)$, directly setting off-diagonal elements equal to zero in this way negatively impacts the decomposition MSE. The eigenvalue approximation ultimately depends upon the accuracy of the eigenvectors, which typically increases for increasing parameters N and K, where the latter is dictated by $K \geq 2N + L - 2$. Here, N is a user-defined estimate of the output paraunitary matrix length, and is not known a priori. The DFT-based algorithm naturally produces an approximately spectrally majorised decomposition, but as described in Section A.2.2, the eigenvalues can be reordered to approximate an analytic decomposition. The latter avoids discontinuities at the intersection of eigenvalues in the frequency domain, and typically leads to a more compact (lower order) decomposition.

5.2.3 Paraunitarity of Polynomial Eigenvectors

The eigenvectors $\boldsymbol{Q}(z) = \tilde{\boldsymbol{F}}(z)$ generated by SMD are strictly paraunitary, as they are created as the product of a series of elementary paraunitary matrices. While this is advantageous for some applications, some loss of paraunitarity may be acceptable if other performance gains are made. For example, the truncation strategy of Appendix C — which is used to truncate the paraunitary matrices within the SMD update step — introduces a trade-off between paraunitarity error, η, and diagonalisation, E_{diag}, for a given paraunitary matrix length; i.e., a larger truncation value, μ, sacrifices paraunitarity to reduce the paraunitary matrix order required to achieve a certain diagonalisation.

The eigenvectors generated by the DFT-based PEVD are only approximately paraunitary [53]. For increasing N, the approximation typically improves; thus, to achieve a desired level of paraunitarity in an application, an adequate value of N must be determined through experimentation. The required value of N is likely to be lower if an analytic decomposition is sought, as discontinuities at eigenvalue intersections are avoided. To represent such discontinuities requires infinitely long polynomials, which do not fit well into the fixed order model of the DFT-based algorithm, as energy from ignored high order polynomial coefficients may corrupt the extracted coefficients from lags $\tau = 0 \ldots N - 1$ via time domain aliasing.

5.2.4 Model Examples and Results

Finite Order Example

Consider the parahermitian matrix

$$\boldsymbol{R}(z) = \begin{bmatrix} .5z^2 + 3 + .5z^{-2} & -.5z^2 + .5z^{-2} \\ .5z^2 - .5z^{-2} & -.5z^2 + 1 - .5z^{-2} \end{bmatrix}, \tag{5.4}$$

which has an exact finite order analytic decomposition with

$$\tilde{\boldsymbol{Q}}(z) = \frac{1}{2}\begin{bmatrix} z+1 & -z+1 \\ -z+1 & z+1 \end{bmatrix} \quad \boldsymbol{D}(z) = \begin{bmatrix} z+2+z^{-1} & 0 \\ 0 & -z+2-z^{-1} \end{bmatrix}$$

where $\boldsymbol{Q}(z)$ contains eigenvectors in its columns, and $\boldsymbol{D}(z)$ contains analytic eigenvalues on its diagonal. When evaluated on the unit circle, these eigenvalues match the power spectral densities given in Figure 5.1(a).

The metrics of Section A.3 were calculated for the decomposition of (5.4) by the DFT-based and SMD algorithms, and can be seen in Table 5.1. Here, L_Q is the length of the computed paraunitarity matrix. For the DFT-based algorithm, both majorised and analytic decompositions were generated to approximate the solutions given in Figure 5.1(b) and (a), respectively. SMD paraunitary matrix truncation parameters of $\mu_1 = 10^{-16}$ and $\mu_2 = 10^{-8}$ were used to demonstrate the trade-off between paraunitarity error and diagonalisation for the algorithm. Both instances of SMD were run until the parahermitian matrix was approximately diagonalised with $E_{\text{diag}} = 10^{-6}$. For the majorised DFT-based decomposition, N was set equal to 165 ($K = 333$) to allow comparison with SMD when utilising μ_2. The phase alignment function used in the DFT-based algorithm was allowed a maximum of 50 optimisation steps.

Given its ability to approximate an analytic decomposition, the DFT-based approach is able to almost perfectly approximate the finite order $\boldsymbol{Q}(z)$ and $\boldsymbol{D}(z)$ for $N = 3$, $K = 9$. In contrast, the SMD algorithm is able to produce a spectrally majorised, approximately diagonal $\boldsymbol{D}(z)$, but the eigenvector matrices are of significantly higher order for both μ_1 and μ_2. The spectrally majorised DFT-based decomposition

Table 5.1: Mean square error, paraunitarity error, diagonalisation, and paraunitary matrix length comparison for finite order example.

Method	MSE	η	E_{diag}	L_Q
DFT-based, analytic	7.1×10^{-9}	4.9×10^{-9}	0	3
DFT-based, majorised	8.8×10^{-7}	2.4×10^{-3}	0	165
SMD, μ_1	2.6×10^{-25}	1.2×10^{-16}	1×10^{-6}	689
SMD, μ_2	1.7×10^{-10}	4.8×10^{-8}	1×10^{-6}	165

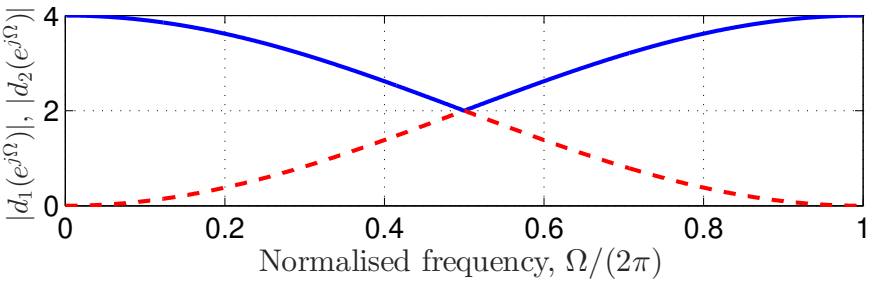

Figure 5.2: PSDs of ground truth eigenvalues in non-finite order example.

has significantly higher MSE and η for the same L_Q as SMD with μ_2. By utilising a higher truncation within SMD, it can been seen that MSE and η have increased, but L_Q has decreased.

Non-finite Order Example

As a second example, consider the parahermitian matrix

$$\boldsymbol{R}(z) = \begin{bmatrix} 2 & z^{-1}+1 \\ z+1 & 2 \end{bmatrix}. \tag{5.5}$$

The eigenvectors

$$\boldsymbol{Q}(z) = \frac{1}{\sqrt{2}} \begin{bmatrix} (z^{-1}+1)(z^{-1}+2+z)^{-1/2} & z^{-1} \\ 1 & -(z^{-1}+1)(z^{-1}+2+z)^{-1/2} \end{bmatrix}$$

and eigenvalues (which are visualised in Figure 5.2)

Table 5.2: Mean square error, paraunitarity error, diagonalisation, and paraunitary matrix length comparison for non-finite order example.

Method	MSE	η	E_{diag}	L_Q
DFT-based, analytic	2.1×10^{-5}	2.3×10^{-3}	0	83
DFT-based, majorised	1.4×10^{-6}	2.2×10^{-3}	0	83
SMD, μ_1	4.4×10^{-25}	2.5×10^{-16}	1×10^{-6}	345
SMD, μ_2	2.9×10^{-10}	9.5×10^{-8}	1×10^{-6}	83

$$\boldsymbol{D}(z) = \begin{bmatrix} 2 + (z^{-1} + 2 + z)^{1/2} & 0 \\ 0 & 2 - (z^{-1} + 2 + z)^{1/2} \end{bmatrix}$$

of $\boldsymbol{R}(z)$ are neither of finite order nor rational. An analytic EVD does not exist for this matrix [9, 10]; to decompose $\boldsymbol{R}(z)$ via an exact PEVD would require polynomial matrices of infinite length for both analytic and majorised decompositions. Note that the ground truth polynomial eigenvalues are actually spectrally majorised in this example.

The metrics of Section A.3 were calculated for the decomposition of (5.5) by the DFT-based and SMD algorithms, and can be seen in Table 5.2. Again, SMD truncation parameters of $\mu_1 = 10^{-16}$ and $\mu_2 = 10^{-8}$ were used, and both instances of SMD were run until $E_{\text{diag}} = 10^{-6}$. For both DFT-based decompositions, N was set equal to 83 ($K = 167$) to allow comparison with SMD when utilising μ_2. The phase alignment function used in the DFT-based algorithm was allowed a maximum of 50 optimisation steps.

The values of MSE and η for both DFT-based PEVDs are significantly higher for this example, while the eigenvectors generated by SMD are of lower order than in Table 5.1. This indicates that the DFT-based approach may suffer for similarly complex problems, while SMD is relatively unaffected. For this example, there is actually a slight disadvantage to approximating an analytic decomposition; this is an indication that the eigenvalue and eigenvector reordering scheme has failed at $\Omega = \pi$, where an eigenvalue with algebraic multiplicity of two is present. Using a higher truncation within SMD has again increased MSE and η, but L_Q has decreased.

5.2.5 Summary

This section has compared the decomposition accuracies of two recent PEVD algorithms. The iterative SMD algorithm has been shown to exhibit significantly lower decomposition mean square error and paraunitarity error than a DFT-based approach; however, SMD does not achieve exact diagonalisation, and its enforcement of spectral majorisation can lead to high polynomial eigenvector orders unless truncation is employed. The ability of the DFT-based method to approximate an analytic decomposition can produce extremely compact eigenvectors, but the algorithm's reliance on a fixed eigenvector order can introduce significant paraunitarity error for decompositions where the ground truth is of infinite order.

From an analysis of both algorithms' complexities, it can be determined that SMD becomes significantly more complex for increasing spatial dimension M, while the DFT-based approach becomes significantly more complex for increasing parahermitian matrix length L. Typically, $L > M$ for a parahermitian matrix input to a PEVD algorithm; thus, in general, SMD is likely to offer a lower complexity solution. A further advantage of SMD in this regard is that it can be executed for any number of iterations and therefore its complexity is flexible and can be adjusted to suit different applications. The DFT-based algorithm offers less flexibility in terms of PEVD complexity; however, if desired, the number of optimisation steps in the phase alignment function can be reduced at the expense of increased eigenvector error.

When designing PEVD implementations for real applications, both of the algorithms described in this section could be extremely useful. As a relatively stable algorithm, with typically low decomposition error, potentially paraunitary eigenvectors, and customisable diagonalisation and eigenvector length, SMD can be deployed in any scenario with reasonably low M. For problems of fixed, finite order, or situations in which an analytic decomposition is preferable or paraunitarity is not required, the DFT-based approach can be used to great effect, provided that L is not too large. However, N not being known a priori is a disadvantage of this method for application purposes.

5.3 Development of a Novel DFT-Based PEVD Algorithm

The DFT-based PEVD formulation of [53] — which is summarised in Section A.2 — has been shown to perform well for finite order problems in the previous section, but requires an a priori estimate of the length of the paraunitary matrix $\boldsymbol{Q}(z)$ in the decomposition. In this section, a novel DFT-based PEVD algorithm that can compute an accurate PEVD without requiring an estimate of the paraunitary matrix length is presented. This algorithm utilises a modified eigenvalue reordering scheme that is less susceptible to errors in the presence of eigenvalues with an algebraic multiplicity greater than one. It has been demonstrated in [9] that the lowest order approximation to (2.4) is possible if one attempts to approximate analytic eigenvectors which — on the unit circle — are characterised by being infinitely differentiable. Here, a cost function that measures the power in the derivatives of a function based on the Fourier coefficients of their discrete samples on the unit circle is utilised. This permits an approach that maximises eigenvector smoothness for a low order decomposition.

Below, Section 5.3.1 and Section 5.3.2 will introduce the smoothness metric and the proposed algorithm, respectively, before Section 5.3.3 details the novel method for eigenvalue and eigenvector reordering. Section 5.3.4 then describes how the smoothness metric is used to align the phase of the eigenvectors, and Section 5.3.5 briefly outlines the complexity of the algorithm. A comparison of the algorithm's performance relative to the method in [53] is presented in Section 5.3.6. Section 5.3.7 then provides a model example to demonstrate the advantages of the proposed eigenvalue and eigenvector reordering approach. Conclusions for this section are drawn in Section 5.3.8.

Elements of the work in this section can be found published in the proceedings of the 10th IEEE Sensor Array and Multichannel Signal Processing Workshop in a paper titled 'Enforcing Eigenvector Smoothness for a Compact DFT-Based Polynomial Eigenvalue Decomposition' [73].

5.3.1 Smoothness Metric

A PEVD produces eigenvalues $d_m(\mathrm{e}^{\mathrm{j}\Omega})$ and eigenvectors $\boldsymbol{q}_m(\mathrm{e}^{\mathrm{j}\Omega})$, where $d_m(\mathrm{e}^{\mathrm{j}\Omega})$ is the mth diagonal element of $\boldsymbol{D}(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega}}$ and $\boldsymbol{q}_m(\mathrm{e}^{\mathrm{j}\Omega})$ is the mth column of $\boldsymbol{Q}(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega}}$. For

the eigenvectors $\boldsymbol{q}_m(\mathrm{e}^{\mathrm{j}\Omega})$ to be compact in the time domain, they must be maximally smooth on the unit circle; tied to this is the requirement for analytic eigenvectors to be infinitely differentiable [9]. In this section, a novel metric to measure the smoothness of eigenvectors is proposed.

If K is sufficiently large and odd, the samples $\mathsf{F}[k] = F(\mathrm{e}^{\mathrm{j}\Omega_k})$, $\Omega_k = 2\pi k/K$, $k = 0 \ldots (K-1)$, of a function $F(\mathrm{e}^{\mathrm{j}\Omega})$ permit its reconstruction via the Dirichlet kernel $P(\mathrm{e}^{\mathrm{j}\Omega})$:

$$F(\mathrm{e}^{\mathrm{j}\Omega}) = \sum_{k=0}^{K-1} \mathsf{F}[k] P(\mathrm{e}^{\mathrm{j}(\Omega-\Omega_k)}) \,, \tag{5.6}$$

where

$$P(\mathrm{e}^{\mathrm{j}\Omega}) = \frac{1}{K} \cdot \frac{\sin(\frac{K}{2}\Omega)}{\sin(\frac{1}{2}\Omega)} = \frac{1}{K} \sum_{\ell=0}^{K-1} \mathrm{e}^{-\mathrm{j}\Omega(\ell - \frac{K-1}{2})} \,. \tag{5.7}$$

Thus,

$$F(\mathrm{e}^{\mathrm{j}\Omega}) = \frac{1}{K} \sum_{k=0}^{K-1} \mathsf{F}[k] \sum_{\ell=0}^{K-1} \mathrm{e}^{-\mathrm{j}(\Omega-\Omega_k)(\ell - \frac{K-1}{2})} \,. \tag{5.8}$$

The pth derivative of $F(\mathrm{e}^{\mathrm{j}\Omega})$ can be written as

$$\frac{\mathrm{d}^p}{\mathrm{d}\Omega^p} F(\mathrm{e}^{\mathrm{j}\Omega}) = \frac{1}{K} \sum_{k=0}^{K-1} \mathsf{F}[k] \sum_{\ell=-(K-1)/2}^{(K-1)/2} (-\mathrm{j}\ell)^p \mathrm{e}^{-\mathrm{j}(\Omega-\Omega_k)\ell} \,. \tag{5.9}$$

Of interest is the power contained within the pth derivative of $F(\mathrm{e}^{\mathrm{j}\Omega})$,

$$\chi_{(p)} = \frac{1}{2\pi} \int_{-\pi}^{\pi} \left| \frac{\mathrm{d}^p}{\mathrm{d}\Omega^p} F(\mathrm{e}^{\mathrm{j}\Omega}) \right|^2 \mathrm{d}\Omega \,, \tag{5.10}$$

which provides some measure of the smoothness of $F(\mathrm{e}^{\mathrm{j}\Omega})$.

Note that due to orthogonality of the complex exponential terms and integration over an integer number of fundamental periods, for a Fourier series with arbitrary coefficients a_ℓ,

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} \left| \sum_\ell a_\ell \mathrm{e}^{\mathrm{j}\Omega\ell} \right|^2 \mathrm{d}\Omega = \sum_\ell \frac{1}{2\pi} \int_{-\pi}^{\pi} \left| a_\ell \mathrm{e}^{\mathrm{j}\Omega\ell} \right|^2 \mathrm{d}\Omega = \sum_\ell |a_\ell|^2 \,. \tag{5.11}$$

Therefore, (5.10) can be written as

$$\chi_{(p)} = \frac{1}{K^2} \sum_{\ell=-(K-1)/2}^{(K-1)/2} \left| (-\mathrm{j}\ell)^p \sum_{k=0}^{K-1} \mathsf{F}[k] \mathrm{e}^{\mathrm{j}\Omega_k \ell} \right|^2 \tag{5.12}$$

$$= \frac{1}{K^2} \sum_{\ell=-(K-1)/2}^{(K-1)/2} \ell^{2p} \mathbf{f}^{\mathrm{H}} \mathbf{C}_\ell \mathbf{f} = \mathbf{f}^{\mathrm{H}} \mathbf{C}_{(p)} \mathbf{f} \,, \tag{5.13}$$

where $\mathbf{f} = [\mathsf{F}[0], \mathsf{F}[1], \cdots, \mathsf{F}[K-1]]^{\mathrm{T}}$ contains K frequency samples $\mathsf{F}[k] = F(\mathrm{e}^{\mathrm{j}\Omega_k})$, $k = 0 \ldots (K-1)$,

$$\mathbf{C}_\ell = \begin{bmatrix} 1 & \mathrm{e}^{\mathrm{j}\frac{2\pi}{K}\ell} & \cdots & \mathrm{e}^{\mathrm{j}\frac{2\pi}{K}(K-1)\ell} \\ \mathrm{e}^{-\mathrm{j}\frac{2\pi}{K}\ell} & 1 & & \vdots \\ \vdots & & \ddots & \vdots \\ \mathrm{e}^{-\mathrm{j}\frac{2\pi}{K}(K-1)\ell} & \cdots & \cdots & 1 \end{bmatrix} , \tag{5.14}$$

and

$$\mathbf{C}_{(p)} = \frac{1}{K^2} \sum_{\ell=-(K-1)/2}^{(K-1)/2} \ell^{2p} \mathbf{C}_\ell \,. \tag{5.15}$$

To consider smoothness up to the Pth derivative, the metric

$$\chi^{(P)} = \sum_{p=1}^{P} \chi_{(p)} = \mathbf{f}^{\mathrm{H}} \left(\sum_{p=1}^{P} \mathbf{C}_{(p)} \right) \mathbf{f}$$

$$= \mathbf{f}^{\mathrm{H}} \left(\frac{1}{K^2} \sum_{\ell=-(K-1)/2}^{(K-1)/2} \mathbf{C}_\ell \sum_{p=1}^{P} \ell^{2p} \right) \mathbf{f} = \mathbf{f}^{\mathrm{H}} \mathbf{C}^{(P)} \mathbf{f} \tag{5.16}$$

can be formed, which calculates the sum of all powers of derivatives of $\mathbf{f}$ up to and including the Pth derivative. For even K, the derivation is similar to above and produces

$$\chi^{(P)} = \mathbf{f}^{\mathrm{H}} \left(\frac{1}{K^2} \sum_{\ell=-\frac{K}{2}}^{\frac{K}{2}-1} \mathbf{C}_\ell \sum_{p=1}^{P} \ell^{2p} \right) \mathbf{f} = \mathbf{f}^{\mathrm{H}} \mathbf{C}^{(P)} \mathbf{f} \,. \tag{5.17}$$

Thus, the metric can be evaluated via a simple weighted inner product [2].

5.3.2 Algorithm Overview

The PEVD approach presented here uses a DFT-based scheme with a similar structure to Algorithm 11 to obtain a solution to the PEVD equation of (2.4). Section 5.3.3 discusses the method used within this scheme to approximate an analytic decomposition, which utilises a novel, modified version of the method in [53].

As discussed in Section 5.1, discontinuities in phase between adjacent frequency bins can arise due to ambiguity in eigenvector phase. When assessed via the metric of Section 5.3.1, eigenvectors containing such phase discontinuities are not smooth and return high values of $\chi^{(P)}$. For a short paraunitary matrix $\boldsymbol{Q}(z)$, these discontinuities must be smoothed [9] and $\chi^{(P)}$ decreased. This is achieved through the use of a phase alignment function, described in Section 5.3.4, which uses Powell's 'dogleg' algorithm [111, 112] to maximise eigenvector smoothness. Contrary to the phase alignment approach of [53], this function does not require a restriction of the paraunitary matrix length to an a priori estimate.

Following the reordering (if desired) and phase alignment of $\mathbf{Q}[k]$, $\mathbf{Q}[\tau]$ and $\mathbf{D}[\tau]$ are computed via the IDFT as in (5.3).

5.3.3 Novel Method for Reordering the Eigenvectors and Eigenvalues

In an analytic decomposition, the eigenvalues — and their eigenvectors — are arranged such that discontinuities between adjacent frequency bins are minimised. Even for an arrangement that provides continuous eigenvalues, such discontinuities can occur when the eigenvalues intersect at some frequencies. The DFT-based PEVD method of [53] uses the eigenvalue and eigenvector reordering approach described in Section A.2.2, which is of relatively low complexity, but relies entirely on the eigenvectors. Since in and near an eigenvalue with J-fold algebraic multiplicity the eigenvectors are ill-defined within a J-dimensional subspace [2], the reliance of this method on eigenvectors only is not ideal. A scheme that uses the smoothness of the eigenvalues for reordering purposes has been shown to be robust in the presence of multiplicities in [54]; however, this method is currently prohibitively expensive to implement.

For implementation in the proposed DFT-based PEVD algorithm, the following

modification is made to the function in Section A.2.2, which has a small impact on complexity, but improves the accuracy of the reordering procedure around a frequency bin with eigenvalues that exhibit an algebraic multiplicity greater than one. The reordering approach described in Section A.2.2 occurs as normal for $k = 1, 2, \ldots, K-1$, unless (for $k > 1$) an eigenvalue $\mathsf{d}_m[k-1]$ in $\mathbf{D}[k-1] = \text{diag}\{\mathsf{d}_1[k-1], \ldots, \mathsf{d}_M[k-1]\}$ is identified that satisfies the criterion

$$|\mathsf{d}_m[k-1] - \mathsf{d}_n[k-1]| < \lambda \tag{5.18}$$

for some eigenvalues $\mathsf{d}_n[k-1]$, $n \in N_m \subseteq \{1 \ldots M\} \backslash \{m\}$, and arbitrarily low threshold λ. Here, the eigenvectors $\mathbf{q}_m[k-1]$ and $\mathbf{q}_n[k-1]$ correspond to repeated eigenvalues and are therefore ill-defined; thus (A.18) may not identify the best ordering of eigenvectors and eigenvalues in the kth frequency bin. For a continuous PSD, the repeated eigenvalue $\mathsf{d}_{\hat{n}}[k-1]$, $\hat{n} \in \{m\} \cup N_m$, that follows $\mathsf{d}_{\hat{n}}[k-2]$ is arbitrary, while the ordering of $\mathsf{d}_{\hat{n}}[k-1]$ and $\mathbf{q}_{\hat{n}}[k-1]$ is ambiguous since the eigenvectors are ill-defined. Therefore, to maintain the ordering established prior to the eigenvalue intersection point, each eigenvector $\mathbf{q}_{\hat{n}}[k-1]$ is set equal to its immediate predecessor $\mathbf{q}_{\hat{n}}[k-2]$ for reordering purposes only. Following the reordering of the kth frequency bin according to Section A.2.2, eigenvectors $\mathbf{q}_{\hat{n}}[k-1]$ are returned to their previous values.

5.3.4 Adjusting the Phase of the Eigenvectors to Promote Phase Alignment

Employing a method to align the phase of eigenvectors in adjacent frequency bins is vital for a compact, low order decomposition; i.e., for a short output paraunitary matrix that introduces little error to the PEVD. Phase alignment can be achieved by finding the phase changes required for each eigenvector $\mathbf{q}_m[k]$ $\forall$ m, k to be maximally smooth according to metric $\chi^{(P)}$. The phase of the mth eigenvector at frequency bin k can be adjusted by an angle $\theta[k]$ according to $\mathbf{q}_m[k] \leftarrow \mathrm{e}^{\mathrm{j}\theta[k]}\mathbf{q}_m[k]$. Below, an objective function is defined which, when minimised via an unconstrained non-linear optimisation algorithm, can be used to determine the optimal $\theta[k]$ $\forall$ k to maximise the mth eigenvector's smoothness.

Objective Function Formulation

The smoothness metric defined for odd K in (5.16) and even K in (5.17) is able to measure the smoothness of a single function. The goal here is to compute a vector of phases $\boldsymbol{\theta} = [\,\theta[0], \theta[1], \cdots, \theta[K-1]\,]^{\mathrm{T}} \in \mathbb{R}^{K\times 1}$ such that all elements of the mth eigenvector $\mathbf{q}_m[k]\ \forall\ k$ are maximally smooth. The objective function therefore measures the smoothness of all elements of $\mathbf{q}_m[k]$ and takes the form

$$f(\boldsymbol{\theta}) = \mathbb{R}\left\{ \sum_{n=1}^{M} \mathbf{f}_n^{\mathrm{H}} \mathbf{C}^{(P)} \mathbf{f}_n \right\}\ , \tag{5.19}$$

where $\mathbf{f}_n^{\mathrm{H}} = \mathbf{v}_n \mathrm{diag}\left\{ \left[\, \mathrm{e}^{\mathrm{j}\theta[0]}, \cdots, \mathrm{e}^{\mathrm{j}\theta[K-1]} \,\right] \right\}$ and $\mathbf{v}_n = [\,\mathsf{q}_{m,n}[0], \cdots, \mathsf{q}_{m,n}[K-1]\,]$, where $\mathsf{q}_{m,n}[k]$ denotes the nth element (row) of eigenvector $\mathbf{q}_m[k]$. Only the real component of the sum in the above equation is taken, as the imaginary component should be zero.

The above equation can be rearranged to form

$$f(\boldsymbol{\theta}) = \mathbb{R}\left\{ \boldsymbol{u}^{\mathrm{H}} \left(\sum_{n=1}^{M} \mathrm{diag}\{\mathbf{v}_n\} \mathbf{C}^{(P)} \mathrm{diag}\{\mathbf{v}_n^{\mathrm{H}}\} \right) \boldsymbol{u} \right\}\ , \tag{5.20}$$

where $\boldsymbol{u}^{\mathrm{H}} = \left[\, \mathrm{e}^{\mathrm{j}\theta[0]}, \cdots, \mathrm{e}^{\mathrm{j}\theta[K-1]} \,\right]$. Setting $\boldsymbol{\Gamma} = \sum_{n=1}^{M} \mathrm{diag}\{\mathbf{v}_n\} \mathbf{C}^{(P)} \mathrm{diag}\{\mathbf{v}_n^{\mathrm{H}}\}$, for $\Gamma \in \mathbb{C}^{K\times K}$, allows for the following compact representation:

$$f(\boldsymbol{\theta}) = \mathbb{R}\{\boldsymbol{u}^{\mathrm{H}} \boldsymbol{\Gamma} \boldsymbol{u}\}\ . \tag{5.21}$$

One can therefore obtain $\boldsymbol{\theta}$ by solving

$$\boldsymbol{\theta} = \underset{\boldsymbol{\theta}' \in \mathbb{R}^{K\times 1}}{\arg\min}\ f(\boldsymbol{\theta}')\ . \tag{5.22}$$

Minimising the Objective Function

Given its simplicity and relatively low cost [112], Powell's iterative 'dogleg' trust region strategy [111] is employed for the unconstrained minimisation of (5.21). In iteration j, a trust region strategy uses the gradient vector $\mathbf{g}$ and approximate Hessian matrix $\hat{\mathbf{H}}$

of the objective function to construct a second-order Taylor series expansion:

$$m_j(\boldsymbol{\varphi}^{(j)}) = f(\boldsymbol{\theta}^{(j)}) + \boldsymbol{\varphi}^{(j)\mathrm{T}}\mathbf{g} + \boldsymbol{\varphi}^{(j)\mathrm{T}}\hat{\mathbf{H}}\boldsymbol{\varphi}^{(j)} \,. \tag{5.23}$$

This model approximates $f(\boldsymbol{\theta}^{(j)} + \boldsymbol{\varphi}^{(j)})$ within a trusted region of radius $\Delta^{(j)}$ around the current point $\boldsymbol{\theta}^{(j)}$ for some step $\boldsymbol{\varphi}^{(j)}$, which is identified according to

$$\boldsymbol{\varphi}^{(j)} \leftarrow \underset{\left\{\boldsymbol{\varphi}^{(j)} \in \mathbb{R}^{K\times 1} \,\middle|\, \|\boldsymbol{\varphi}^{(j)}\| < \Delta^{(j)}\right\}}{\arg\min} m_j(\boldsymbol{\varphi}^{(j)}) \,. \tag{5.24}$$

The selected method first finds the minimiser of (5.23) along the steepest descent direction:

$$\boldsymbol{\varphi}_c^{(j)} = -\frac{\mathbf{g}^{\mathrm{T}}\mathbf{g}}{\mathbf{g}^{\mathrm{T}}\hat{\mathbf{H}}\mathbf{g}}\mathbf{g} \,. \tag{5.25}$$

If this point lies outside of the trusted region, the intersection of the line from the origin to $\boldsymbol{\varphi}_c^{(j)}$ with the trust region boundary is used for $\boldsymbol{\varphi}^{(j)}$. Otherwise, the quasi-Newton point is found:

$$\boldsymbol{\varphi}_{qn}^{(j)} = -\hat{\mathbf{H}}^{-1}\mathbf{g} \,. \tag{5.26}$$

If this point lies within the trust region, $\boldsymbol{\varphi}^{(j)} \leftarrow \boldsymbol{\varphi}_{qn}^{(j)}$, and the method moves to the next iteration. Otherwise, a solution is found at the intersection of the trust region boundary with the line from $\boldsymbol{\varphi}_c^{(j)}$ to $\boldsymbol{\varphi}_{qn}^{(j)}$.

A parameter ρ, which compares the predicted reduction in the objective function with the actual reduction, is used to measure the strength of the approximation given by (5.23):

$$\rho = \frac{f(\boldsymbol{\theta}^{(j)}) - f(\boldsymbol{\theta}^{(j)} + \boldsymbol{\varphi}^{(j)})}{m_j(\mathbf{0}) - m_j(\boldsymbol{\varphi}^{(j)})} \,. \tag{5.27}$$

In practice, if $\rho > 0.75$, the approximation is good and the trust region is expanded such that $\Delta^{(j+1)} = \min\left\{2\Delta^{(j)}, \Delta_{\max}\right\}$, where $\Delta_{\max}$ is a user-defined maximum. If $\rho < 0.25$, the approximation is poor and the trust region is decreased in size such that $\Delta^{(j+1)} = \Delta^{(j)}/4$. Furthermore, if $\rho > 0$, $\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)} + \boldsymbol{\varphi}^{(j)}$; otherwise, $\boldsymbol{\theta}^{(j+1)} = \boldsymbol{\theta}^{(j)}$.

The employed unconstrained minimisation strategy requires an initial guess of $\boldsymbol{\theta}$; here, the initial guess is chosen to be a vector of zeroes for simplicity. Note that

the initial trust region size, $\Delta^{(0)}$, is user-defined; however, choosing $\Delta^{(0)} \leftarrow \Delta_{\max}$ is typically valid.

Gradient Vector and Hessian Matrix Formulation

If $\theta_i = \theta[i]$, u_i is the ith element of $\boldsymbol{u}$, and γ_{ik} is the element shared by the ith row and kth column of $\boldsymbol{\Gamma}$, the objective function in (5.21) can also be written as

$$f(\boldsymbol{\theta}) = \sum_{i=0}^{K-1}\sum_{k=0}^{K-1} u_i^* \gamma_{ik} u_k = \sum_{i=0}^{K-1}\sum_{k=0}^{K-1} \gamma_{ik} \mathrm{e}^{\mathrm{j}(\theta_i - \theta_k)} \; . \tag{5.28}$$

Taking the derivative with respect to θ_ℓ yields

$$\begin{aligned}\frac{\partial f}{\partial \theta_\ell} &= -\mathrm{j} \sum_{i=0, i\neq\ell}^{K-1} \gamma_{i\ell} \mathrm{e}^{\mathrm{j}(\theta_i - \theta_\ell)} + \mathrm{j} \sum_{k=0, k\neq\ell}^{K-1} \gamma_{\ell k} \mathrm{e}^{\mathrm{j}(\theta_\ell - \theta_k)} \\ &= -\mathrm{j} \sum_{i=0}^{K-1} \gamma_{i\ell} \mathrm{e}^{\mathrm{j}(\theta_i - \theta_\ell)} + \mathrm{j} \sum_{k=0}^{K-1} \gamma_{k\ell}^* \mathrm{e}^{-\mathrm{j}(\theta_k - \theta_\ell)} \; . \end{aligned} \tag{5.29}$$

Note that $\gamma_{\ell k} = \gamma_{k\ell}^*$, as $\boldsymbol{\Gamma}$ is Hermitian by construction. Converting to matrix notation, the gradient vector is

$$\begin{aligned}\mathbf{g} = \frac{\partial f}{\partial \boldsymbol{\theta}} &= -\mathrm{j}\mathrm{diag}\left\{\boldsymbol{u}\right\} \boldsymbol{\Gamma}^{\mathrm{T}} \boldsymbol{u}^* + \mathrm{j}\mathrm{diag}\left\{\boldsymbol{u}^{\mathrm{H}}\right\} \boldsymbol{\Gamma}^{\mathrm{H}} \boldsymbol{u} \\ &= 2\mathbb{R}\left\{\mathrm{j}\mathrm{diag}\left\{\boldsymbol{u}^{\mathrm{H}}\right\} \boldsymbol{\Gamma}^{\mathrm{H}} \boldsymbol{u}\right\} = -2\mathbb{I}\left\{\mathrm{diag}\left\{\boldsymbol{u}^{\mathrm{H}}\right\} \boldsymbol{\Gamma} \boldsymbol{u}\right\} \; . \end{aligned} \tag{5.30}$$

Taking the derivative of (5.29) with respect to θ_p, with θ_ℓ now fixed, one acquires the equation for the off-diagonal elements of the Hessian matrix:

$$\begin{aligned}\frac{\partial f}{\partial \theta_\ell \partial \theta_p} &= -\mathrm{j}^2 \gamma_{p\ell} \mathrm{e}^{\mathrm{j}(\theta_p - \theta_\ell)} - \mathrm{j}^2 \gamma_{p\ell}^* \mathrm{e}^{-\mathrm{j}(\theta_p - \theta_\ell)} \\ &= \gamma_{p\ell} \mathrm{e}^{\mathrm{j}(\theta_p - \theta_\ell)} + \gamma_{p\ell}^* \mathrm{e}^{-\mathrm{j}(\theta_p - \theta_\ell)} \; . \end{aligned} \tag{5.31}$$

So the off-diagonal (OD) terms of the Hessian matrix are:

$$\begin{aligned}\mathbf{H}_{\text{OD}} &= \text{diag}\left\{\boldsymbol{u}\right\}\boldsymbol{\Gamma}^{\text{T}}\text{diag}\left\{\boldsymbol{u}^{*}\right\} + \text{diag}\left\{\boldsymbol{u}^{\text{H}}\right\}\boldsymbol{\Gamma}^{\text{H}}\text{diag}\left\{\boldsymbol{u}\right\} \\ &= 2\mathbb{R}\left\{\text{diag}\left\{\boldsymbol{u}^{\text{H}}\right\}\boldsymbol{\Gamma}\text{diag}\left\{\boldsymbol{u}\right\}\right\}\ . \end{aligned} \tag{5.32}$$

Taking the derivative of (5.29) with respect to θ_ℓ, one acquires the equation for the diagonal elements of the Hessian matrix:

$$\frac{\partial^2 f}{\partial \theta_\ell^2} = -\sum_{i=0, i\neq\ell}^{K-1} \gamma_{i\ell}\text{e}^{\text{j}(\theta_i-\theta_\ell)} - \sum_{k=0, k\neq\ell}^{K-1} \gamma_{\ell k}\text{e}^{\text{j}(\theta_\ell-\theta_k)}\ . \tag{5.33}$$

In matrix form, this equates to

$$\begin{aligned}\mathbf{H}_{\text{D}} &= \text{diag}\left\{-\text{diag}\left\{\boldsymbol{u}\right\}\hat{\boldsymbol{\Gamma}}^{\text{T}}\boldsymbol{u}^{*} - \text{diag}\left\{\boldsymbol{u}^{\text{H}}\right\}\hat{\boldsymbol{\Gamma}}^{\text{H}}\boldsymbol{u}\right\} \\ &= -2\mathbb{R}\left\{\text{diag}\left\{\text{diag}\left\{\boldsymbol{u}\right\}\hat{\boldsymbol{\Gamma}}^{\text{T}}\boldsymbol{u}^{*}\right\}\right\}\ , \end{aligned} \tag{5.34}$$

where $\hat{\boldsymbol{\Gamma}} = \boldsymbol{\Gamma} - \text{diag}\{\text{diag}\{\boldsymbol{\Gamma}\}\}$ contains zeroes on its diagonal. The Hessian matrix is then given by

$$\begin{aligned}\mathbf{H} &= \mathbf{H}_{\text{OD}} - \text{diag}\{\text{diag}\{\mathbf{H}_{\text{OD}}\}\} + \mathbf{H}_{\text{D}} \\ &= 2\mathbb{R}\left\{\text{diag}\left\{\boldsymbol{u}^{\text{H}}\right\}\boldsymbol{\Gamma}\text{diag}\left\{\boldsymbol{u}\right\}\right\} \\ &\quad - \text{diag}\big\{\text{diag}\big\{2\mathbb{R}\left\{\text{diag}\left\{\boldsymbol{u}^{\text{H}}\right\}\boldsymbol{\Gamma}\text{diag}\left\{\boldsymbol{u}\right\}\right\}\big\}\big\} \\ &\quad - 2\mathbb{R}\left\{\text{diag}\left\{\text{diag}\left\{\boldsymbol{u}\right\}\hat{\boldsymbol{\Gamma}}^{\text{T}}\boldsymbol{u}^{*}\right\}\right\} \\ &= 2\mathbb{R}\left\{\text{diag}\left\{\boldsymbol{u}^{\text{H}}\right\}\boldsymbol{\Gamma}\text{diag}\left\{\boldsymbol{u}\right\}\right\} \\ &\quad - 2\mathbb{R}\left\{\text{diag}\left\{\text{diag}\{\boldsymbol{\Gamma}\} + \text{diag}\left\{\boldsymbol{u}\right\}\hat{\boldsymbol{\Gamma}}^{\text{T}}\boldsymbol{u}^{*}\right\}\right\} \\ &= 2\mathbb{R}\left\{\text{diag}\left\{\boldsymbol{u}^{\text{H}}\right\}\boldsymbol{\Gamma}\text{diag}\left\{\boldsymbol{u}\right\}\right\} - 2\mathbb{R}\left\{\text{diag}\left\{\text{diag}\left\{\boldsymbol{u}\right\}\boldsymbol{\Gamma}^{\text{T}}\boldsymbol{u}^{*}\right\}\right\}, \end{aligned} \tag{5.35}$$

since $\text{diag}\big\{\text{diag}\big\{\boldsymbol{u}^{\text{H}}\big\}\boldsymbol{\Gamma}\text{diag}\{\boldsymbol{u}\}\big\} = \text{diag}\{\boldsymbol{\Gamma}\}$.

The second part of (5.35) is often less significant when compared to the first part;

e.g., simulations typically find that

$$\frac{\left\|2\mathbb{R}\left\{\operatorname{diag}\left\{\operatorname{diag}\left\{\boldsymbol{u}\right\}\boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{u}^{*}\right\}\right\}\right\|_{\mathrm{F}}}{\left\|2\mathbb{R}\left\{\operatorname{diag}\left\{\boldsymbol{u}^{\mathrm{H}}\right\}\boldsymbol{\Gamma}\operatorname{diag}\left\{\boldsymbol{u}\right\}\right\}\right\|_{\mathrm{F}}} \approx 10^{-3}\ . \tag{5.36}$$

In numerical experiments, (5.35) is not always positive definite, which is a requirement for this method of minimisation [112]; however, $2\mathbb{R}\left\{\operatorname{diag}\left\{\boldsymbol{u}^{\mathrm{H}}\right\}\boldsymbol{\Gamma}\operatorname{diag}\left\{\boldsymbol{u}\right\}\right\}$ is positive semi-definite, as $\mathbf{C}^{(P)}$ and $\boldsymbol{\Gamma}$ are positive semi-definite. The following approximation to the Hessian matrix is proposed, which is guaranteed to be positive definite and eliminates unnecessary computation:

$$\hat{\mathbf{H}} = 2\mathbb{R}\left\{\operatorname{diag}\left\{\boldsymbol{u}^{\mathrm{H}}\right\}\boldsymbol{\Gamma}\operatorname{diag}\left\{\boldsymbol{u}\right\}\right\} + \alpha\mathbf{I}_K\ , \tag{5.37}$$

where α is very small and $\mathbf{I}_K$ is a $K \times K$ identity matrix.

5.3.5 Algorithm Complexity

For a constant number of optimisation steps, the complexity of the phase alignment step for a single eigenvector is $\mathcal{O}(K^3)$ due to matrix inversion [53, 95]; thus, the total complexity of the phase alignment step for M eigenvectors is of order $\mathcal{O}(MK^3)$. The computation of the frequency domain representation of $\boldsymbol{R}(z)$, the execution of K EVDs, and the reordering of the eigenvalues and eigenvectors are of lower complexity than this step; thus, the total complexity of the proposed algorithm is approximately $\mathcal{O}(MK^3)$.

5.3.6 Source Model Simulation Scenarios and Results

The simulations below have been performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M = 5$, $O_D = 18$, $O_Q = 10$, and $\Delta_{\mathrm{DR}} = 50$.

The proposed algorithm considers smoothness up to the third derivative; i.e., the metric $\chi^{(3)}$ is used. A maximum of 50 minimisation steps according to Section 5.3.4 are allowed for both algorithms, and a parameter of $\alpha = 10^{-14}$ is used. Each algorithm is instructed to approximate an analytic decomposition.

Simulation Scenario 1

The method in [53], henceforth referred to as the 'existing' algorithm, requires knowledge of the length of $\boldsymbol{Q}(z)$, L_Q, prior to execution, and typically chooses a number of frequency bins $K_e = 2L_Q + L - 2$, where L is the length of the input matrix $\boldsymbol{R}(z)$. The method proposed in this section, referred to as the 'proposed' algorithm, does not require an input value of L_Q, but instead outputs $\boldsymbol{Q}(z)$ with length equal to the number of frequency bins K_p used in the decomposition. Note that outer lags of $\mathbf{Q}[\tau]$ may contain negligible energy and can be trimmed using a threshold μ_t in a process detailed in Appendix C. The first scenario provides the existing method with values of $L_Q \in \{5;\ 10\}$, and uses the same generated values of $K_e = K_p \in \{47;\ 57\}$ for both algorithms.

Simulation Scenario 2

To test the case where both algorithms produce $\boldsymbol{Q}(z)$ of the same length, the second scenario provides the proposed method with values of $K_p \in \{47;\ 57\}$, and provides the existing method with $L_Q = K_p$ such that $K_e \in \{131;\ 151\}$.

Results and Discussion

The ensemble-averaged mean square reconstruction error, paraunitarity error, and L_Q were calculated for both of the algorithms and simulation scenarios, and can be seen in Table 5.3. The table demonstrates that the proposed approach is able to provide extremely low decomposition MSE and paraunitarity error, η. Furthermore, the existing method is not capable of achieving such performance even when using significantly more frequency bins and generating paraunitary matrices of the same length. The algorithmic complexity of both algorithms is approximately $\mathcal{O}(MK^3)$ [53]; thus the choice of DFT length K is extremely significant. Also shown is the impact of truncation on performance: if paraunitary matrix length is of critical importance, then MSE and η can be sacrificed to generate shorter matrices. Typically, increasing K is shown to improve MSE and η at the expense of higher L_Q.

Table 5.3: Average mean square error, paraunitarity error, and a posteriori paraunitary matrix length comparison.

Method	MSE	η	L_Q
existing[1], $K_e = 47$	0.2139	0.03696	5
existing[1], $K_e = 57$	6.923×10^{-6}	2.319×10^{-6}	10
existing[2], $K_e = 131$	1.251×10^{-9}	8.596×10^{-10}	47
existing[2], $K_e = 151$	5.854×10^{-9}	3.296×10^{-9}	57
proposed[1,2], $K_p = 47$	9.648×10^{-18}	1.011×10^{-15}	47
proposed[1,2], $K_p = 57$	1.197×10^{-22}	6.179×10^{-19}	57
proposed[1,2,‡], $K_p = 47$	2.341×10^{-10}	8.116×10^{-11}	23.35
proposed[1,2,‡], $K_p = 57$	3.278×10^{-10}	8.211×10^{-11}	23.22

[1] Simulation scenario 1 [2] Simulation scenario 2

[‡] $\mathbf{Q}[\tau]$ truncated using $\mu_t = 10^{-10}$ and scheme from Appendix C

5.3.7 Model Example with Repeated Eigenvalues

As mentioned in Section 5.3.3, the eigenvalue and eigenvector reordering strategy in [53] can encounter problems if, at some frequency bin k, there exists an eigenvalue with an algebraic multiplicity greater than one. Such an eigenvalue is present in $\boldsymbol{D}(z) = \operatorname{diag}\{d_1(z), d_2(z), d_3(z)\}$,

$$\begin{aligned} d_1(z) &= -\mathrm{j}0.25z^{-1} + 1 + \mathrm{j}0.25z \\ d_2(z) &= 0.25z^{-2} + 0.5 + 0.25z^2 \\ d_3(z) &= -0.25z^{-1} + 0.5 - 0.25z \quad , \end{aligned} \tag{5.38}$$

which — when evaluated at $z = \mathrm{e}^{\mathrm{j}\pi}$ — possesses the eigenvalue $d_1(\mathrm{e}^{\mathrm{j}\pi}) = d_2(\mathrm{e}^{\mathrm{j}\pi}) = d_3(\mathrm{e}^{\mathrm{j}\pi}) = 1$ with an algebraic multiplicity of 3. The crossing of the eigenvalues at this point is illustrated in Figure 5.3(a). Figure 5.3(b) and Figure 5.3(c) show the PSDs of the resulting eigenvalues from an instance of the proposed algorithm when using the reordering approach of [53] and the novel method of Section 5.3.3, respectively, for $K = 92$. Note that K has been chosen carefully for this example, to ensure that the repeated eigenvalue coincides with a frequency bin exactly; as such, this example is somewhat unrealistic. Random paraunitary eigenvectors $\boldsymbol{Q}(z)$ of order 4 — generated from the

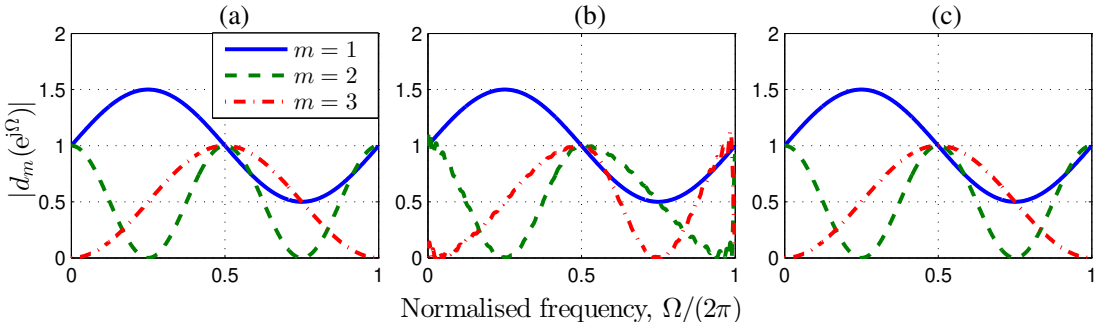

Figure 5.3: PSDs of (a) ground truth eigenvalues with algebraic multiplicity of three at $\Omega = \pi$, and eigenvalues obtained for reordering schemes of (b) Section A.2.2 and (c) Section 5.3.3.

concatenation of arbitrary paraunitary first order components according to the scheme in [4] — were used to generate input parahermitian matrix $\boldsymbol{R}(z) = \boldsymbol{Q}(z)\boldsymbol{D}(z)\tilde{\boldsymbol{Q}}(z)$.

In Figure 5.3(b), it can be observed that the second and third eigenvalues are identified incorrectly, resulting in discontinuities at $\Omega = 0$ and $\Omega = 2\pi$. Consequently, error is introduced to this decomposition, as K is insufficiently large to provide the large time domain support required to describe these discontinuities; i.e., error is present in the polynomial eigenvalues, which impacts on the decomposition MSE. The modified reordering approach has successfully recovered the original eigenvalues without error.

Despite the successful reordering of the eigenvalues to a continuous form, the eigenvectors are still ill-defined in the presence of repeated eigenvalues. To demonstrate the problems this can cause, Figure 5.4(a) shows the ground truth PSDs of the first eigenvector, while Figure 5.4(b) shows the PSDs of the equivalent eigenvector output by the proposed algorithm. Clearly, the ill-defined nature of eigenvector $\boldsymbol{q}_1(e^{j\pi})$ has resulted in $\boldsymbol{q}_1(e^{j\Omega})$ being discontinuous around $\Omega = \pi$. As for the case above with discontinuous eigenvalues, such a discontinuity introduces error to the decomposition; however, this time the error is present in the output polynomial eigenvectors and impacts on both decomposition MSE and paraunitarity error.

5.3.8 Summary

In this section, a novel frequency-based algorithm capable of computing a compact PEVD has been introduced. This algorithm makes use of a newly developed metric for

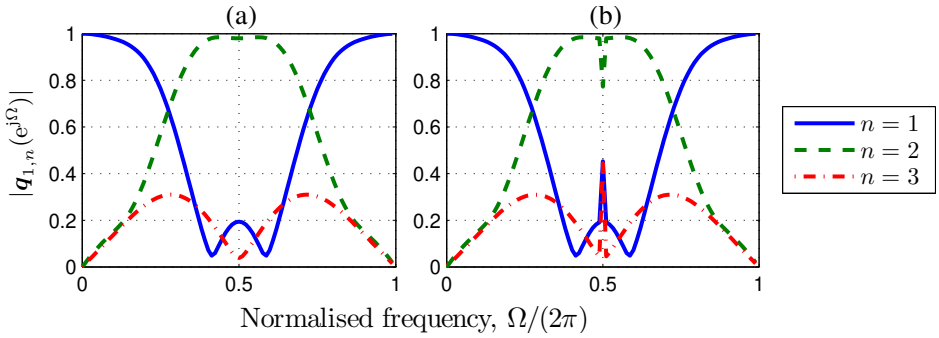

Figure 5.4: PSDs of (a) ground truth first eigenvector and (b) first eigenvector obtained from proposed method.

measuring the smoothness of a function evaluated on the unit circle. By minimising this metric for the eigenvectors produced by the algorithm, the phase responses of the eigenvectors have been successfully modified and their compactness in the time domain has been enforced as a result.

Simulation results have demonstrated that the proposed algorithm offers superior performance to an existing frequency-based PEVD algorithm, with the advantage of not requiring a priori information regarding the paraunitary matrix length. Furthermore, the introduction of a modified eigenvalue and eigenvector reordering scheme has been shown to enable the extraction of a ground truth $\boldsymbol{D}(z)$ in the presence of eigenvalues with algebraic multiplicity greater than one. However, error can still be introduced to the resulting PEVD via the polynomial eigenvectors, which are required to describe discontinuities in the frequency domain due to the ill-defined nature of the eigenvectors corresponding to repeated eigenvalues.

When designing PEVD implementations for real applications, the algorithm described in this section could be extremely useful, provided that K is not prohibitively large — as algorithm complexity is approximately proportional to K^3. For an input parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{M \times M}$, existing iterative methods often have complexities proportional to M^3, while the proposed algorithm has a complexity proportional to M; thus, the latter can offer advantages when decomposing parahermitian matrices with large spatial dimensions.

5.4 An Order-Iterated Novel DFT-Based PEVD Algorithm

The novel algorithm of the previous section, which does not require a priori information regarding the paraunitary matrix order, improved upon the algorithm of [53] in terms of PEVD performance. However, the algorithm still requires a priori knowledge of the DFT length, K, to be used in the decomposition. To address this problem, this section details a novel iterative frequency-based PEVD algorithm — based on the algorithm in Section 5.3 — which can compute an accurate PEVD without requiring paraunitary matrix order or decomposition length information. Instead, the decomposition length is increased within the set $K \in \{2^j \,|\, j \in \mathbb{N}\}$ until the decomposition error falls below a user-specified threshold. By restricting the decomposition length to powers of two, the computations in a given algorithm iteration can be carried forward to subsequent iterations to reduce redundancy.

Below, Section 5.4.1 will introduce the proposed algorithm and Section 5.4.2 will briefly outline its complexity. Using the simulation scenarios of Section 5.4.3, a comparison of the algorithm's performance relative to existing iterative PEVD methods is presented in Section 5.4.4, and conclusions are drawn in Section 5.4.5.

Elements of the work in this section can be found published in the proceedings of the 52nd Asilomar Conference on Signals, Systems and Computers in a paper titled 'An Iterative DFT-based Approach to the Polynomial Matrix Eigenvalue Decomposition' [74], and in the proceedings of the 44th International Conference on Acoustics, Speech, and Signal Processing in a paper titled 'Iterative Approximation of Analytic Eigenvalues of a Parahermitian Matrix EVD' [54].

5.4.1 Algorithm Overview

The PEVD approach presented here uses an iterative DFT-based scheme with a similar underlying structure to Algorithm 11 to obtain a solution to the PEVD equation

of (2.4). In this iterative approach, equation (5.1) is adapted to

$$\mathbf{R}[k] = \mathbf{Q}[k]\mathbf{D}[k]\mathbf{Q}^{\mathrm{H}}[k]\,,\ k = 0 \dots K_i - 1\,, \tag{5.39}$$

which uses an increasing number of frequency bins, K_i, until the decomposition MSE falls below a user-defined threshold. Here, K_i is restricted to powers of two such that $K_i \in \{2^{\lceil \log_2 L \rceil + i}\}$, where L is the length of $\boldsymbol{R}(z)$, and $i = 0, 1, \dots, I_{\max} - 1$ records the current iteration. Parameter $I_{\max}$ is a user-defined maximum number of iterations for the algorithm. As in other DFT-based PEVD algorithms, $\mathbf{R}[k]$ is obtained from the K_i-point DFT of $\mathbf{R}[\tau]$,

$$\mathbf{R}[k] = \boldsymbol{R}(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega_k}} = \sum\nolimits_{\tau} \mathbf{R}[\tau]\mathrm{e}^{-\mathrm{j}\Omega_k \tau}\,,\ k = 0 \dots K_i - 1\,, \tag{5.40}$$

where $\Omega_k = 2\pi k / K_i$. Note that in iteration i, $\mathbf{R}[k]$ in (5.40) for $k = 0, 1, \dots, K_i - 1$ is identical to $\mathbf{R}[k]$ for $k = 0, 2, 4, \dots, K_{i+1} - 2$ in the $(i+1)$th iteration. By extension, $\mathbf{Q}[k]$ and $\mathbf{D}[k]$ from (5.39) are also the same for these k. The calculation of half of the eigenvectors and eigenvalues can therefore be avoided at each iteration beyond the first. Similarly, the phase alignment step can exploit this redundancy to aid optimisation.

The eigenvalue and eigenvector reordering scheme of Section 5.3.3 is employed in this algorithm to facilitate the approximation of an analytic decomposition. If $i > 0$, $\mathbf{Q}[k]$ for $k = 0, 2, 4, \dots, K_i - 2$ can be carried through from the previous iteration; however, the eigenvector ordering in these matrices can change, as the inner products of eigenvectors in subsequent frequency bins have changed. The process of Section 5.3.3 must therefore be completed as normal for $k = 1, 2, \dots, K_i - 1$ at each algorithm iteration.

Phase alignment of eigenvectors in adjacent frequency bins is vital for a compact, low order decomposition. The phase alignment strategy of Section 5.3.4 is also used here; however, a slight modification is made to aid the optimisation process by exploiting the iterative nature of the algorithm. The employed unconstrained minimisation strategy requires an initial guess of $\boldsymbol{\theta}$. For $i = 0$, the initial guess is chosen to be a vector of zeroes; however, in subsequent iterations of the algorithm, values of $\theta[k]$ for $k = 0, 2, 4, \dots, K_i - 2$ can be carried through from the previous iteration and used to form

Input: $\boldsymbol{R}(z)$, ϵ, $I_{\max}$, υ
Output: $\boldsymbol{D}(z)$, $\boldsymbol{Q}(z)$
$i \leftarrow 0$; L is the length of $\boldsymbol{R}(z)$
do
  $K_i \leftarrow 2^{\lceil \log_2 L \rceil + i}$
  for $k \leftarrow K_i - 1, \ldots, 1, 0$ **do**
    if $i > 0$ **and** $k \bmod 2 = 0$ **then**
      $(\mathbf{Q}[k], \mathbf{D}[k]) \leftarrow (\mathbf{Q}[k/2], \mathbf{D}[k/2])$
    else
      Compute $\mathbf{R}[k]$ according to DFT in (5.40)
      Compute $\mathbf{Q}[k]$ and $\mathbf{D}[k]$ via ordered EVD of $\mathbf{R}[k]$
    end
  end
  if $\upsilon = 1$ **then**
    Permute $\mathbf{Q}[k]$ and $\mathbf{D}[k]$ to approximate an analytic decomposition according to Section 5.3.3
  end
  Use phase alignment strategy of Section 5.3.4 to maximise smoothness of $\mathbf{Q}[k]$
  Obtain $(\mathbf{Q}[\tau], \mathbf{D}[\tau])$ from IDFT of $(\mathbf{Q}[k], \mathbf{D}[k])$
  Find MSE from (A.28) with $\hat{\boldsymbol{R}}(z) \leftarrow \boldsymbol{Q}(z)\boldsymbol{D}(z)\tilde{\boldsymbol{Q}}(z)$
  $i \leftarrow i + 1$
while MSE $> \epsilon$ and $i < I_{\max}$

Algorithm 12: Proposed iterative DFT-based PEVD algorithm

the initial guess alongside zeroes for $k = 1, 3, 5, \ldots, K_i - 1$.

Following the permutation (if desired) and phase alignment of $\mathbf{Q}[k]$, $\mathbf{Q}[\tau]$ is computed via the IDFT as

$$\mathbf{Q}[\tau] = \frac{1}{K_i} \sum\nolimits_{k=0}^{K_i-1} \mathbf{Q}[k] \mathrm{e}^{\mathrm{j}\Omega_k \tau} \,,\ \tau = 0 \ldots K_i - 1 \,, \tag{5.41}$$

and $\mathbf{D}[\tau]$ is found in a similar fashion. If the mean square reconstruction error of the decomposition is above a user-defined threshold ϵ, algorithm iterations continue; otherwise, the algorithm ends. The MSE is computed according to (A.28) in Section A.3.

Algorithm 12 summarises the above steps of the proposed method. Here, a user-input parameter υ determines if the algorithm approximates an analytic decomposition.

5.4.2 Algorithm Complexity

For a constant number of optimisation steps, the complexity of the phase alignment step for a single eigenvector in iteration i is $\mathcal{O}(K_i^3)$ due to the execution of matrix inversion in the optimisation algorithm [53,95]; thus, the total complexity of the phase alignment step for M eigenvectors is of order $\mathcal{O}(MK_i^3)$. All other components of the algorithm have lower complexity; thus, the total complexity of one iteration of the proposed algorithm is approximately $\mathcal{O}(MK_i^3)$. If I iterations of the algorithm are completed before it ceases operations, the overall complexity of the proposed algorithm is determined by the complexity of the inversion of the largest matrix and is approximately $\mathcal{O}(MK_{I-1}^3)$. For fixed I, K_{I-1} is proportional to L; thus, the complexity can be estimated to be of order $\mathcal{O}(ML^3)$.

5.4.3 Simulation Scenarios

To benchmark the proposed approach, its performance is compared against the iterative coefficient domain methods of SMD [45] and SBR2 [6]. For the proposed method, an analytic (permuted) decomposition is approximated ($\upsilon = 1$), and eigenvector smoothness is measured using $\chi^{(3)}$; i.e., up to the third derivative. To cease iterations, a decomposition MSE threshold of $\epsilon = 10^{-8}$ and $I_{\max} = 5$ are used. A maximum of 50 minimisation steps according to Section 5.3.4 were allowed and a parameter of $\alpha = 10^{-14}$ is employed. Polynomial matrix truncation parameters of $\mu_1 = 10^{-16}$ and $\mu_2 = 10^{-8}$ are used for SMD and a truncation parameter of μ_2 is used for SBR2. Iterations of the algorithms other than the one proposed cease if $E_{\text{diag}} \leq 10^{-6}$. Note that E_{diag} for the proposed method is always zero, as $\mathbf{D}[\tau]$ is diagonal.

Simulation Scenario 1

This scenario utilises the above PEVD algorithms to decompose an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $M = 5$, $O_D = 18$, $O_Q = 10$, and $\Delta_{\text{DR}} = 50$. Following the completion of each algorithm, the metrics defined in Section A.3 are recorded alongside the lengths of the output paraunitary matrices, L_Q.

Simulation Scenario 2

This scenario utilises the above algorithms to decompose the theoretical parahermitian matrix

$$\boldsymbol{R}(z) = \begin{bmatrix} 2 & z^{-1}+1 \\ z+1 & 2 \end{bmatrix}, \tag{5.42}$$

which has eigenvectors and eigenvalues (which are visualised in Figure 5.2) that are neither of finite order nor rational. An analytic EVD does not exist for this matrix [9, 10]; to decompose $\boldsymbol{R}(z)$ via an exact PEVD would require polynomial matrices of infinite length. Execution time measurements are averaged over 10^3 instances of the simulation scenario.

Simulation Scenario 3

For broadband angle of arrival (AoA) estimation, powerful narrowband methods such as the multiple signal classification (MUSIC) algorithm [77] are not directly applicable. In [18], the PEVD is used to generalise MUSIC to the case of polynomial matrices, resulting in the development of the spatio-spectral polynomial MUSIC (SSP-MUSIC) algorithm. Work in [19] demonstrates that the accuracy of SSP-MUSIC depends strongly on the efficacy of the PEVD algorithm used. Here, the AoA estimation performances that the above algorithms offer when paired with SSP-MUSIC are compared. For brevity, the reader is referred to Appendix D and [18, 19] for details regarding the implementation of SSP-MUSIC. Performance of each PEVD algorithm is measured in terms of AoA estimation accuracy; this is determined through a visual assessment of metric $P_{\text{SSP}}(\vartheta, \Omega) \in \mathbb{R}_{>0}$ output by the SSP-MUSIC algorithm in [18], which is evaluated at a range of angles of arrival, ϑ, and frequencies, Ω. A high value of $P_{\text{SSP}}(\vartheta, \Omega)$ indicates the presence of a source at a specific angle of arrival and frequency, while a low value indicates that no source is present.

In this simulation scenario, an $M = 6$ element array collecting broadband data in a vector $\mathbf{x}[n] \in \mathbb{C}^6$ is illuminated by three broadband sources characterised as follows:

- source 1 — located at $\vartheta_1 = -45°$, active over a frequency range $\Omega_1' \in [0.5\pi, 0.9\pi]$;

- source 2 — located at $\vartheta_2 = -30°$, active over a frequency range $\Omega_2' \in [0.1\pi, 0.9\pi]$;
- source 3 — located at $\vartheta_3 = -10°$, active over a frequency range $\Omega_3' \in [0.4\pi, 0.9\pi]$.

For simplicity, the elevation angles of the sources are equal to zero. The array signals are corrupted by uncorrelated independent and identically distributed complex Gaussian noise at an SNR of 20 dB. To exclude error sources other than inaccuracies in the subspace identification, the source data is modelled as a sum of closely spaced sinusoids with randomised phases and lengths of 96000 samples, for which highly accurate narrowband steering vectors can be used to simulate incidence on the array. The broadband steering vectors that the SSP-MUSIC algorithm uses to scan the noise-only subspace are based on fractional delay filters constructed from truncated sinc functions [108, 109]. Space-time covariance matrix $\mathbf{R}[\tau] = \mathcal{E}\{\mathbf{x}[n]\mathbf{x}^{\mathrm{H}}[n-\tau]\}$ is estimated for $|\tau| \leq 20$. Metric $P_{\mathrm{SSP}}(\vartheta, \Omega)$ is averaged over 10^3 instances of the scenario. In each instance, SMD and SBR2 are allowed the same execution time as the proposed method.

5.4.4 Results and Discussion

Simulation Scenario 1

It is clear from Table 5.4 that the proposed method outperforms SMD and SBR2 with respect to all performance metrics. The good performance of the proposed algorithm for this scenario is to be expected, as DFT-based PEVD algorithms were observed to perform well for finite order problems in Sections 5.2.4 and 5.3.6. The tendency of iterative time-based PEVD algorithms to encourage spectral majorisation of the eigenvalues results in slow convergence and paraunitary matrices of high order. The proposed method can instead opt to approximate an analytic decomposition, and converges quickly to a low order solution.

Simulation Scenario 2

In Table 5.5, the proposed method does not perform as well for the non-finite matrix in (5.42). As discussed in Section 5.1, limiting the decomposition to a finite number of frequency bins (and subsequently the same number of polynomial coefficients) can result

Table 5.4: Simulation Scenario 1: MSE, paraunitarity error, diagonalisation, execution time, and paraunitary matrix length comparison.

Method	MSE	η	E_{diag}	Time / s	L_Q
proposed	5.750×10^{-29}	2.887×10^{-22}	0	0.08854	64
SMD, μ_1	2.791×10^{-16}	1.702×10^{-16}	10^{-6}	29.47	1296
SMD, μ_2	9.321×10^{-7}	3.847×10^{-8}	10^{-6}	11.34	357.9
SBR2	1.815×10^{-6}	3.303×10^{-8}	10^{-6}	37.64	600.0

Table 5.5: Simulation Scenario 2: MSE, paraunitarity error, diagonalisation, execution time, and paraunitary matrix length comparison.

Method	MSE	η	E_{diag}	Time / s	L_Q
proposed	7.077×10^{-9}	1.381×10^{-4}	0	0.1196	64
SMD, μ_1	4.362×10^{-25}	2.466×10^{-16}	10^{-6}	0.6256	345
SMD, μ_2	2.909×10^{-10}	9.546×10^{-8}	10^{-6}	0.1995	83
SBR2	2.909×10^{-10}	9.546×10^{-8}	10^{-6}	0.1724	83

in time domain aliasing. The paraunitarity error of the proposed method is very high, indicating that the majority of the decomposition error arises in the eigenvectors. Given that the ground truth eigenvalues are spectrally majorised, the iterative time-based PEVD strategies can converge quickly to a relatively low order solution if moderate truncation is employed.

Simulation Scenario 3

Obtained for Scenario 3, the results of Figure 5.5 demonstrate that the proposed algorithm is capable of outperforming SMD and SBR2 at most frequencies when each is paired with SSP-MUSIC. However, the proposed method is not as capable when identifying the absence of sources for frequencies in the intervals $\Omega \in \{[0, 0.1\pi); (0.9\pi, \pi]\}$. The presence of noise in the decomposition results in poor phase alignment for those eigenvectors that correspond to the noise-only subspace. Furthermore, the ill-defined eigenvectors associated with noise-only cause the eigenvector and eigenvalue reordering scheme to fail for this subspace. Both of these aspects result in increased error in the eigenvectors and poorer isolation of the source frequency ranges. The plot of Figure 5.6

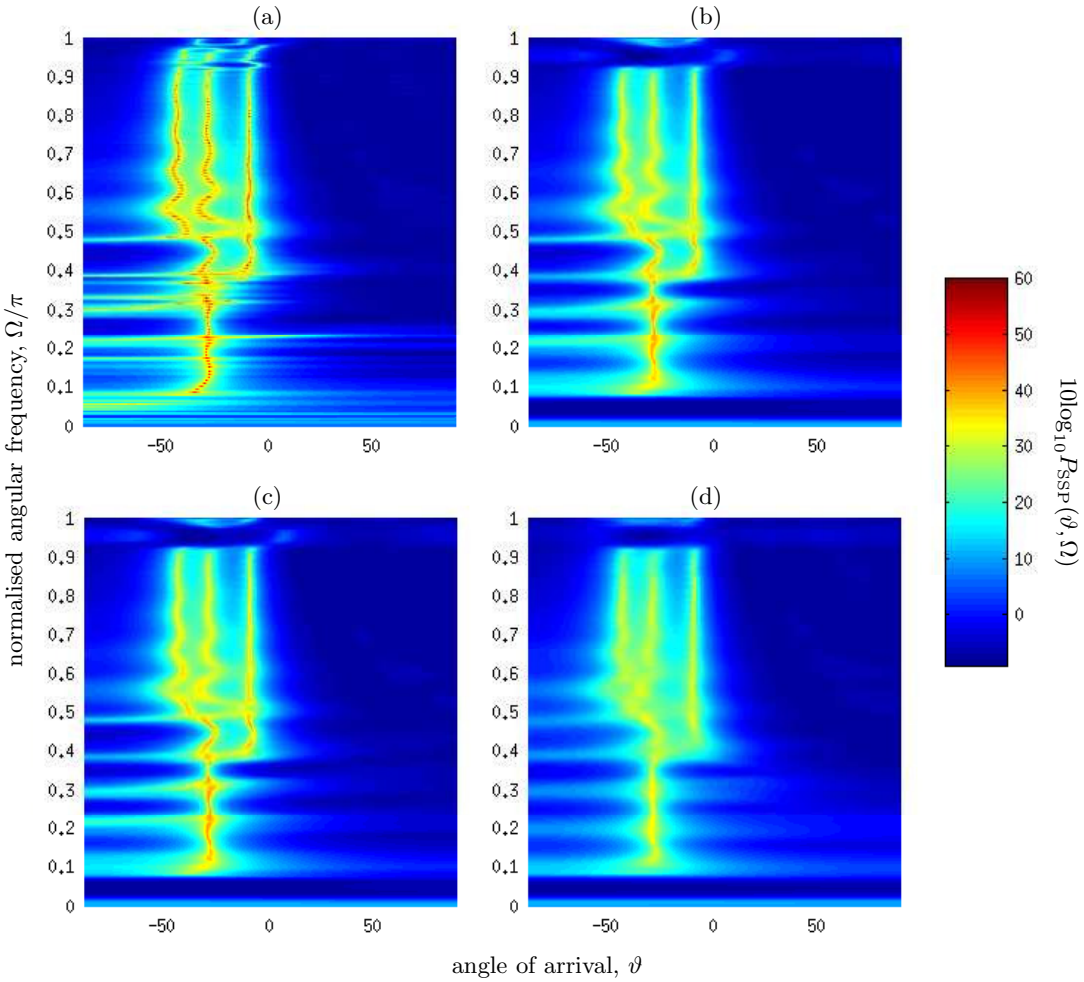

Figure 5.5: Performance of SSP-MUSIC based on (a) proposed method, (b) SMD, μ_1, (c) SMD, μ_2, and (d) SBR2 for simulation scenario 3.

shows SSP-MUSIC performance at a frequency of $\Omega = 2\pi/3$, where all sources are active; from this, it can be seen that the proposed method offers superior angle of arrival localisation of the sources at this frequency.

5.4.5 Summary

In this section, a novel iterative frequency-based algorithm capable of computing a compact and accurate PEVD has been introduced. Simulation results have demonstrated that when decomposing empirically constructed matrices, the proposed algorithm can offer superior performance to existing iterative PEVD algorithms. While the proposed

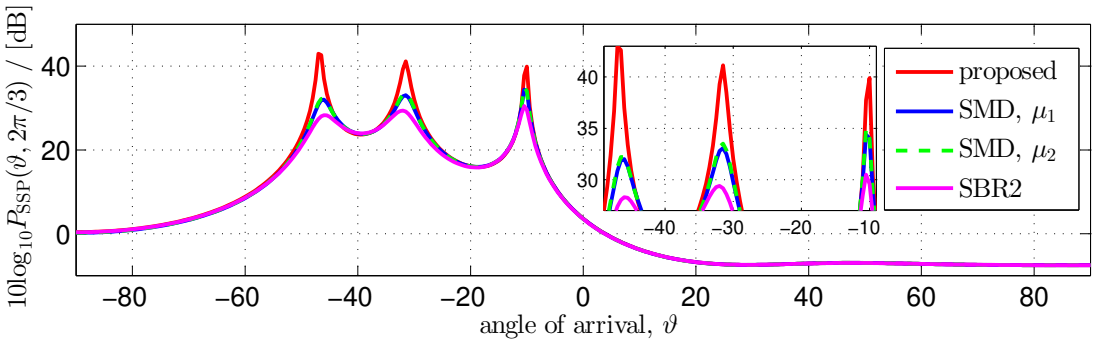

Figure 5.6: Performance of SSP-MUSIC at $\Omega = 2\pi/3$ based on proposed method, SMD, and SBR2 for simulation scenario 3.

method does not perform as well for a difficult, non-finite order decomposition, an AoA estimation simulation scenario has demonstrated the superior source angle of arrival localisation enabled by the developed algorithm. The evidence presented suggests that the method is suitable for application-based scenarios — particularly those with a high number of sensors, as algorithm complexity only grows linearly with spatial dimension M.

5.5 Eigenvector Ambiguity and Approximating a Minimum-Order Solution

Using the DFT-based PEVD approaches of Sections 5.3 or 5.4, it is possible to approximate analytic eigenvectors $\boldsymbol{Q}(z)$ and eigenvalues $\boldsymbol{D}(z)$ that satisfy

$$\boldsymbol{R}(z) \approx \boldsymbol{Q}(z)\boldsymbol{D}(z)\tilde{\boldsymbol{Q}}(z) \,. \tag{5.43}$$

In a simulation scenario, a parahermitian matrix $\boldsymbol{R}(z)$ input to the algorithm can be constructed using a randomised source model — such as the one described in Appendix B [45] — with ground truth eigenvectors $\boldsymbol{V}(z)$ and eigenvalues $\boldsymbol{\Lambda}(z)$ according to

$$\boldsymbol{R}(z) = \boldsymbol{V}(z)\boldsymbol{\Lambda}(z)\tilde{\boldsymbol{V}}(z) \,. \tag{5.44}$$

Typically, $\boldsymbol{D}(z)$ is a very good approximation to $\boldsymbol{\Lambda}(z)$ and is of comparable order; however, $\boldsymbol{Q}(z)$ typically has greater order than $\boldsymbol{V}(z)$ even after moderate truncation, as seen in the results of Section 5.3.6.

With the knowledge that each eigenvector in $\boldsymbol{V}(z)$ or $\boldsymbol{Q}(z)$ may be influenced by an arbitrary all-pass filter and still be valid, this section investigates a method to find some diagonal matrix $\boldsymbol{U}(z)$ with all-pass filters on its diagonal that satisfies

$$\boldsymbol{Q}(z) = \boldsymbol{H}(z)\boldsymbol{U}(z)\,, \tag{5.45}$$

where $\boldsymbol{H}(z)$ is the paraunitary matrix that can fulfil the same purpose as $\boldsymbol{Q}(z)$ in (5.43) with minimum-order; thus, $\boldsymbol{H}(z)$ should be of order less than or equal to the order of $\boldsymbol{V}(z)$.

For the decomposition of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$, the above paraunitary matrices can be described according to $\boldsymbol{U}(z) = \mathrm{diag}\{u_1(z), u_2(z), \ldots, u_M(z)\}$,

$$\boldsymbol{Q}(z) = \begin{bmatrix} \boldsymbol{q}_1(z), & \boldsymbol{q}_2(z), & \ldots, & \boldsymbol{q}_M(z) \end{bmatrix} = \begin{bmatrix} q_{1,1}(z) & q_{1,2}(z) & \ldots & q_{1,M}(z) \\ q_{2,1}(z) & q_{2,2}(z) & \ldots & q_{2,M}(z) \\ \vdots & \vdots & \ddots & \vdots \\ q_{M,1}(z) & q_{M,2}(z) & \ldots & q_{M,M}(z) \end{bmatrix}, \tag{5.46}$$

and

$$\boldsymbol{H}(z) = \begin{bmatrix} \boldsymbol{h}_1(z), & \boldsymbol{h}_2(z), & \ldots, & \boldsymbol{h}_M(z) \end{bmatrix} = \begin{bmatrix} h_{1,1}(z) & h_{1,2}(z) & \ldots & h_{1,M}(z) \\ h_{2,1}(z) & h_{2,2}(z) & \ldots & h_{2,M}(z) \\ \vdots & \vdots & \ddots & \vdots \\ h_{M,1}(z) & h_{M,2}(z) & \ldots & h_{M,M}(z) \end{bmatrix}. \tag{5.47}$$

Here, $u_m(z)\tilde{u}_m(z) = 1\ \forall\ m$ and $\boldsymbol{q}_1(z) = [q_{1,1}(z), q_{2,1}(z), \ldots, q_{M,1}(z)]^{\mathrm{T}}$ is the first eigen-

vector in $\boldsymbol{Q}(z)$. For this eigenvector, the following series of equations exist:

$$\begin{aligned} q_{1,1}(z) &= h_{1,1}(z)u_1(z) \\ q_{2,1}(z) &= h_{2,1}(z)u_1(z) \\ &\vdots \\ q_{M,1}(z) &= h_{M,1}(z)u_1(z) \end{aligned} \quad . \tag{5.48}$$

If $\tilde{u}_1(z) = u_1^*(1/z^*)$ is defined as the all-pass filter

$$\tilde{u}_1(z) = \frac{\tilde{a}_1(z)}{a_1(z)} \, , \tag{5.49}$$

its parahermitian conjugate is

$$u_1(z) = \frac{a_1(z)}{\tilde{a}_1(z)} \tag{5.50}$$

and $u_1(z)\tilde{u}_1(z) = 1$ as desired.

Below, Section 5.5.1 discusses a method to approximate $\boldsymbol{U}(z)$ via the identification of the greatest common divisor of multiple polynomials, and Section 5.5.2 confirms the viability of this method.

5.5.1 Greatest Common Divisor of Multiple Polynomials

One can find a suitable polynomial for $a_1(z)$ by identifying the shared greatest common divisor (GCD) of pairs of polynomials in the set $\{q_{1,1}(z); \ldots; q_{M,1}(z)\}$. For this, the monic polynomial subtraction technique implemented by [115] can be used. It has been found through experimentation that this method is capable of computing approximately equivalent GCDs for all pairs of polynomials; $a_1(z)$ can then be assigned such that it is equal to the average of all GCDs.

Once $a_1(z)$ has been identified, an FIR approximation, $\tilde{\hat{a}}_1^{-1}(z)$, to $\tilde{a}_1^{-1}(z)$ can be obtained by evaluating $1/\tilde{a}_1(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega_k}}$, $\Omega_k = 2\pi k/K$, at K discrete points on the unit circle and computing the IDFT of the result. Subsequently, $\hat{u}_1(z) = a_1(z)\tilde{\hat{a}}_1^{-1}(z)$ is computed, and a shortened representation for the eigenvector $\boldsymbol{q}_1(z)$ is obtained:

$$\hat{\boldsymbol{h}}_1(z) = \boldsymbol{q}_1(z)\tilde{\hat{u}}_1(z) \, . \tag{5.51}$$

Repeating the above for all eigenvectors yields an overall lower order paraunitary matrix

$$\hat{\boldsymbol{H}}(z) = \begin{bmatrix} \hat{\boldsymbol{h}}_1(z), & \hat{\boldsymbol{h}}_2(z), & \ldots, & \hat{\boldsymbol{h}}_M(z) \end{bmatrix} \tag{5.52}$$

that serves as an approximation to the minimum-order solution and replaces $\boldsymbol{Q}(z)$ in (5.43).

5.5.2 Results and Discussion

To test the efficacy of the above method for attaining a minimum-order solution, three simulation scenarios are used. Each scenario utilises an instantiation of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained from the randomised source model in Appendix B [45] with $\Delta_{\text{DR}} = 30$. PEVDs for each scenario are obtained using the DFT-based PEVD approach of Section 5.3, which is instructed to approximate an analytic decomposition. The length, K, of each decomposition is selected to be suitably high, such that the decomposition MSE and paraunitarity error are negligible.

The first scenario uses a matrix $\boldsymbol{Q}(z)$ of order 15 obtained from the PEVD of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{3\times 3}$ of order 6, which is generated using a ground truth paraunitary matrix $\boldsymbol{V}(z)$ of order 2. A second simulation scenario is designed to test the method for the decomposition of $\boldsymbol{R}(z)$ of larger spatial dimension and slightly higher polynomial order. Here, a matrix $\boldsymbol{Q}(z)$ of order 63 is obtained from the PEVD of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{10\times 10}$ of order 38, which is generated using a ground truth paraunitary matrix $\boldsymbol{V}(z)$ of order 10. To test the method for the decomposition of $\boldsymbol{R}(z)$ of high polynomial order, a third simulation scenario uses a matrix $\boldsymbol{Q}(z)$ of order 127 obtained from the PEVD of $\boldsymbol{R}(z) : \mathbb{C} \to \mathbb{C}^{3\times 3}$ of order 118, which is generated using a ground truth paraunitary matrix $\boldsymbol{V}(z)$ of order 30.

For the first simulation scenario, Figure 5.7 compares the average power at each lag of $\mathbf{Q}[\tau]$, $\hat{\mathbf{H}}[\tau]$, and ground truth $\mathbf{V}[\tau]$. From this, it can be seen that by approximately cancelling the effects of paraunitary matrix $\boldsymbol{U}(z)$, a more compact matrix $\hat{\boldsymbol{H}}(z)$ can be obtained. A compact paraunitary matrix in this sense has its energy contained within a small number of contiguous lags. Unexpectedly, in lags with low power, the distribution of power is not uniform and roughly trends upwards towards the start and end lags of

the matrix. However, if lags with power below $-75\,\text{dB}$ are ignored, it can be observed that $\hat{\boldsymbol{H}}(z)$ has the same polynomial order as the ground truth $\boldsymbol{V}(z)$, indicating that either $\boldsymbol{V}(z)$ is close to the minimum-order solution, or that the approach of Section 5.5.1 is insufficient for its desired purpose. Nevertheless, the reduction in order of $\boldsymbol{Q}(z)$ is beneficial for application purposes. Importantly, it was found that using $\hat{\boldsymbol{H}}(z)$ instead of $\boldsymbol{Q}(z)$ resulted in negligible differences in decomposition MSE and paraunitarity error.

Obtained following the acquisition of $\hat{\boldsymbol{H}}(z)$ for the second simulation scenario, Figure 5.8 compares the average power at each lag of the different paraunitary matrices. From this, it would appear that increasing the spatial dimension does not significantly impact performance. Interestingly, in lags with low power, the distribution of power is more uniform than observed in Figure 5.7. Again, negligible differences in decomposition MSE and paraunitarity error were observed when using $\hat{\boldsymbol{H}}(z)$ instead of $\boldsymbol{Q}(z)$.

Similarly to above, the third simulation scenario provided Figure 5.9. From this, it would appear that increasing the polynomial order does significantly impact performance. Indeed, the computation of all GCDs was not always possible for this simulation scenario, and the results of Figure 5.9 can be considered the best-case scenario. An unsuccessful instance of the method finds at least one GCD of a pair of polynomials to be equal to 1. If this occurs for some m, as long as a non-zero number of GCDs are not equal to 1, the average of these GCDs is used to generate $a_m(z)$. If all GCDs are equal to 1, $\hat{\boldsymbol{H}}(z) = \boldsymbol{Q}(z)$ and no order reduction occurs. If lags with power below approximately $-75\,\text{dB}$ are ignored, it can be observed that $\hat{\boldsymbol{H}}(z)$ has the same polynomial order as the ground truth $\boldsymbol{V}(z)$; however, the ignored lags contain higher power than in Figures 5.7 and 5.8 — indicating that the average of the GCDs is not quite as accurate as in the other simulation scenarios. When computation of a reduced-order $\hat{\boldsymbol{H}}(z)$ was successful for this scenario, there was again negligible differences in decomposition MSE and paraunitarity error when using $\hat{\boldsymbol{H}}(z)$ instead of $\boldsymbol{Q}(z)$.

The accuracy of each of the polynomial coefficients of $\boldsymbol{Q}(z)$ is limited by the choice of numerical representation in the employed MATLAB® simulation software; i.e., the true polynomial coefficient values are quantised to the nearest possible double-precision floating-point value. In this scenario, matrix $\boldsymbol{Q}(z)$ has many low-valued polynomial

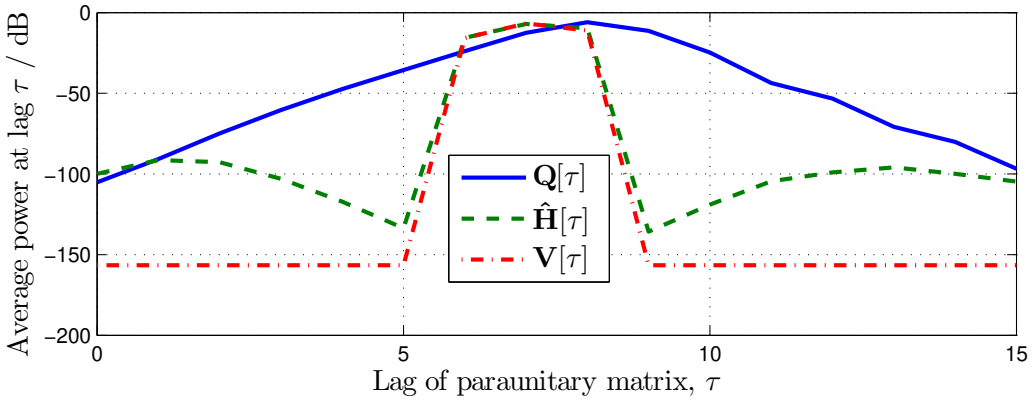

Figure 5.7: Average power at lag τ of original $\mathbf{Q}[\tau]$, shortened $\hat{\mathbf{H}}[\tau]$, and ground truth $\mathbf{V}[\tau]$ paraunitary matrices for $\boldsymbol{Q}(z) : \mathbb{C} \to \mathbb{C}^{3\times3}$ of order 15 and $\boldsymbol{V}(z)$ of order 2.

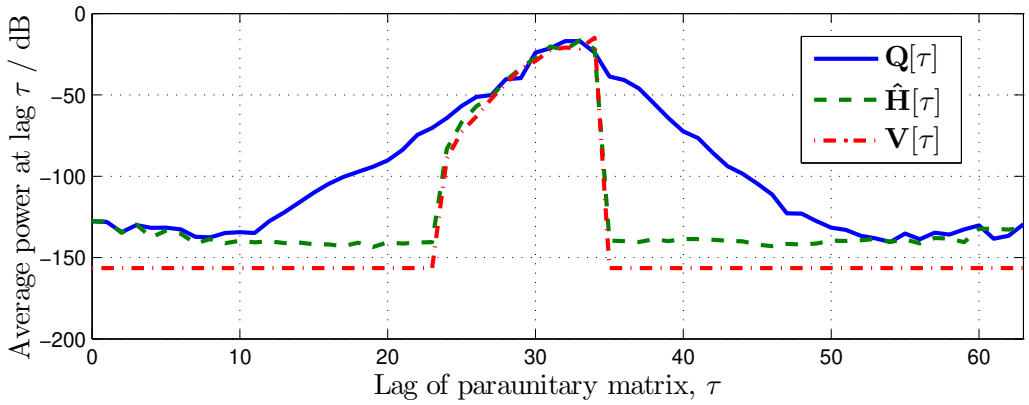

Figure 5.8: Average power at lag τ of original $\mathbf{Q}[\tau]$, shortened $\hat{\mathbf{H}}[\tau]$, and ground truth $\mathbf{V}[\tau]$ paraunitary matrices for $\boldsymbol{Q}(z) : \mathbb{C} \to \mathbb{C}^{10\times10}$ of order 63 and $\boldsymbol{V}(z)$ of order 10.

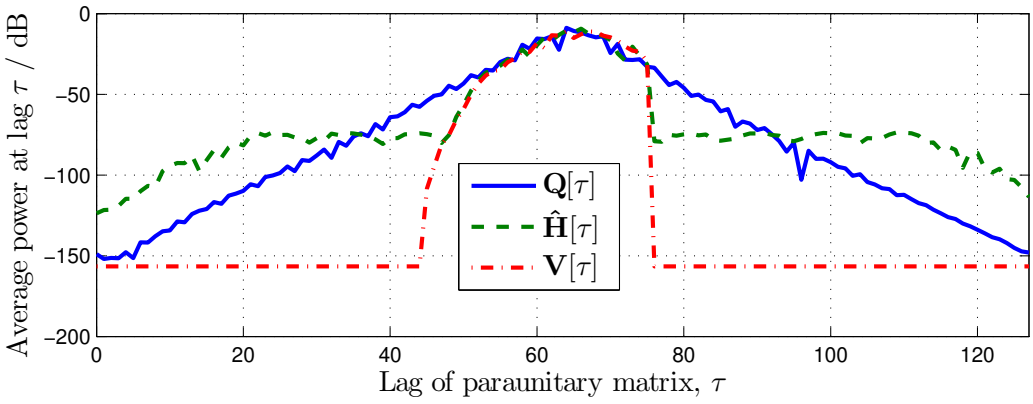

Figure 5.9: Average power at lag τ of original $\mathbf{Q}[\tau]$, shortened $\hat{\mathbf{H}}[\tau]$, and ground truth $\mathbf{V}[\tau]$ paraunitary matrices for $\boldsymbol{Q}(z) : \mathbb{C} \to \mathbb{C}^{3\times3}$ of order 127 and $\boldsymbol{V}(z)$ of order 30.

coefficients that can be easily corrupted by quantisation error. In the presence of too many coefficient values with a high degree of error, the monic polynomial subtraction technique can fail to identify the GCDs for all pairs of polynomials. If the polynomial coefficients of $\boldsymbol{Q}(z)$ were represented with higher accuracy, the reliability of the method might increase for paraunitary matrices with high polynomial order. Alternatively, a more robust method for computing the GCD might give better results.

5.5.3 Summary

This section has discussed a method to approximate a minimum-order solution to the PEVD via the identification of the greatest common divisor of multiple polynomials. The GCD identification approach utilised in this work — which employs the monic polynomial subtraction method of [115] — is well-behaved for short polynomials ($\boldsymbol{Q}(z)$ of order less than 100), but sometimes fails to find accurate GCDs for all pairs of longer polynomials. It might therefore be worthwhile to find a more robust method for computing the GCD of polynomials of high order. Despite the shortcomings of this approach for $\boldsymbol{Q}(z)$ of high polynomial order, it is clear that coupling a method capable of approximating a minimum-order solution with the paraunitary matrix output by a DFT-based PEVD algorithm can yield shorter paraunitary matrices without any disadvantages other than the time taken to approximate and utilise $\boldsymbol{U}(z)$. For applications where short paraunitary matrices are more important than increased algorithm execution time, the method described in this section could be useful.

5.6 Conclusions

In contrast to previous chapters, which discussed modified and novel coefficient domain PEVD algorithms, this chapter has investigated DFT-based alternatives for the PEVD. Following an introduction of the general structure and merits of DFT-based PEVD algorithms, a comparison of the characteristics of an example DFT-based algorithm with a coefficient domain algorithm was provided. Through this comparison, it was established that the DFT-based algorithm is particularly suited to problems of relatively low, finite order, or situations in which an analytic decomposition is preferable or only

approximate paraunitarity is required. In contrast, the SMD algorithm attained reliably low decomposition error and potentially paraunitary eigenvectors, and offered customisable diagonalisation and eigenvector length. Furthermore, it was concluded that the DFT-based approach offers a viable alternative to SMD for the decomposition of parahermitian matrices with large spatial dimensions. However, the former algorithm's reliance on an a priori estimate of paraunitary matrix length was considered to be disadvantageous for application purposes.

A subsequent section overcame this final point by developing a novel DFT-based PEVD algorithm that is able to perform well without requiring a priori knowledge of the paraunitary matrix length. By minimising a newly developed metric capable of measuring the smoothness of the eigenvectors produced by the algorithm, the phase responses of the eigenvectors were adjusted until phase alignment was achieved. The resulting polynomial eigenvectors were shown to be both compact and more accurate than those obtained by an existing DFT-based PEVD algorithm. Furthermore, the introduction of a modified eigenvalue and eigenvector reordering scheme was shown to enable the extraction of ground truth polynomial eigenvalues despite the presence of eigenvalues with algebraic multiplicity greater than one.

This algorithm formed the basis of a novel iterative DFT-based algorithm capable of adapting to any dimensionality of parahermitian matrix without an a priori estimate of the DFT length. Simulation results demonstrated that when decomposing empirically constructed matrices, the proposed algorithm can offer superior performance to existing iterative PEVD algorithms. Furthermore, an angle of arrival estimation simulation scenario demonstrated that the developed algorithm is capable of superior source angle of arrival localisation; however, it was noted that the presence of noise hinders the phase alignment and eigenvector and eigenvalue reordering elements of the DFT-based approach. The evidence presented throughout this chapter suggests that frequency-based PEVD methods are suitable for application-based scenarios — particularly those with a high number of sensors, as algorithm complexity only grows linearly with spatial dimension M.

Finally, a method to approximate a minimum-order solution to the PEVD was

introduced. If a DFT-based algorithm is able to approximate analytic polynomial eigenvectors of relatively low order, and if processing time is not of critical importance, applying this method to the polynomial eigenvectors can decrease their length without introducing error. However, it might be worthwhile to find a more robust method for reducing the order of polynomial eigenvectors of relatively high order.

Chapter 6

Conclusions

6.1 Summary of Contributions

By introducing a number of novel methods to lower the computational cost of existing iterative algorithms related to the PEVD, the research of Chapter 3 has successfully accomplished the first of the objectives listed in Section 1.2. The proposed methods in this chapter conquered unexplored areas of PEVD algorithm improvement by optimising matrix manipulations in software, exploiting parahermitian matrix symmetry [67], minimising algorithmic redundancy [68], and reducing the search and update space of iterative PEVD algorithms [68, 69]. Importantly, the significant performance gains made possible by the half-matrix parahermitian matrix representation, restricted update strategy, and matrix multiplication optimisations can be used simultaneously and extended to any number of iterative PEVD algorithms without impacting accuracy. Improvements were also made to an existing polynomial matrix QR decomposition algorithm [70] and a polynomial matrix truncation strategy. While the latter enforced a degree of polynomial matrix order reduction, and therefore made some steps towards satisfying the second objective in Section 1.2, each of the improved PEVD approaches still exhibited algorithmic complexity that grew rapidly with the spatial dimensions of the parahermitian matrix. Some form of spatial dimension reduction was therefore required if the algorithms were to be suitable for applications involving large broadband arrays, and by extension large parahermitian matrices.

Chapter 4 addressed the problem of spatial dimension reduction — and therefore the second objective of this research — by taking additional steps to convert the sequential form of existing PEVD algorithms to a reduced dimensionality, partially parallelisable divide-and-conquer (DaC) architecture. Simulations utilising the proposed DaC approach demonstrated that a recursive block diagonalisation strategy can be used to segment a large parahermitian matrix into multiple independent parahermitian matrices of smaller spatial dimensions. These matrices were subsequently diagonalised independently and in parallel using a PEVD algorithm that incorporated the most well-performing complexity reduction strategies of Chapter 3. The divide-and-conquer sequential matrix diagonalisation [71] and parallel-sequential matrix diagonalisation [72] algorithms created using this DaC framework exhibited convergence speeds an order of magnitude faster than existing methods for parahermitian matrices with large spatial dimensions [64]. In contrast to the current state-of-the-art approaches, the developed algorithms were shown to be well-suited to deployment in application scenarios [20].

In the process of completing the third objective of this research, Chapter 5 introduced the concept of DFT-based PEVD algorithms, which offer potentially analytic and subsequently minimum-order solutions to the PEVD. A comparison of such an algorithm with an iterative time-based PEVD algorithm established that DFT-based algorithms are particularly suited to the generation of approximately analytic decompositions of relatively low, finite order, provided that only approximate paraunitarity of the polynomial eigenvectors is required [56]. However, the DFT-based algorithm's reliance on an a priori estimate of the paraunitary matrix length was considered to be disadvantageous for application purposes. A novel DFT-based PEVD algorithm overcame this problem by utilising an alternative phase alignment strategy that relied on the smoothness of the eigenvectors and not on a priori knowledge of the paraunitary matrix length. The resulting polynomial eigenvectors were shown to be both compact and more accurate than those obtained from the previously tested existing DFT-based PEVD algorithm [73]. An extension of this algorithm, which uses an iterative approach to remove all a priori knowledge requirements, was then found to perform well relative to existing iterative PEVD algorithms [74]. The evidence presented throughout Chap-

ter 5 has indicated that DFT-based PEVD methods are suitable for application-based scenarios — particularly those with a high number of sensors, as their algorithmic complexity only grows linearly with parahermitian matrix spatial dimension. A method to approximate a minimum-order solution to the PEVD was then introduced. Applying this method to the polynomial eigenvectors from an analytic PEVD successfully decreased their length without introducing error; however, a more robust method would perhaps have more success when reducing the order of polynomial eigenvectors of relatively high order.

In brief, the objectives set out in Section 1.2 were to reduce the computational complexity of PEVD algorithms, to investigate dimensionality reduction of the parahermitian matrix, and to investigate DFT-based approaches to the PEVD. Given the above contributions, it can be concluded that these objectives have been met. Most importantly, the work in this thesis has demonstrated the flexibility of PEVD algorithms, and has investigated, illuminated, and improved upon various aspects of these algorithms to the point where they are now potentially suitable for real-time application purposes.

6.2 Future Work

While reductions in the computational complexity of iterative PEVD algorithms have been comprehensively discussed in this thesis, all implementations of these algorithms — excepting SBR2 [57–59] — are currently restricted to MATLAB®. Moving to a lower-level software language such as C would require an extended period of development time, but would increase algorithm speed and portability substantially. In addition, proof of convergence for the DaC algorithms of Chapter 4 would certainly be of interest. Tied to this is the potential for an investigation into the enforcement of spectral majorisation of the polynomial eigenvalues from the sequential matrix diagonalisation algorithm and by extension, the sequential matrix segmentation algorithm (SMS). It would be useful to identify if the enforcement of block diagonalisation of a parahermitian matrix via SMS aids or hinders spectral majorisation. Related to these sources of future work is the potential to detect which channels contributing to a space-

time covariance matrix are already weakly correlated; these channels could perhaps help to create a 'natural' block diagonalisation of the resulting parahermitian matrix. The 'divide' scheme of a DaC algorithm might then be able to suppress any remaining small or sparse coefficients with greater ease, and therefore reach the 'conquer' or subsequent 'divide' stages faster than a brute-force approach.

Future work in DFT-based PEVD algorithm development includes the utilisation of alternative optimisation strategies for phase alignment of the eigenvectors. Ideally, these would avoid the costly matrix inversion required by the novel algorithms discussed in this thesis. Currently, the DFT-based algorithms do not scale well in terms of parahermitian matrix order. For emerging DFT-based PEVD algorithms, lower-cost schemes should be considered. These could utilise smoothness metrics that are less costly than the one used in Section 5.3.1 and [54,73,74] — which has order K^3 [116,117] — and could perhaps take inspiration from the lost-cost interpolation schemes in [118], which are of order $K \log_2 K$. A final algorithmic source of future work could involve the identification of a more robust method for computing the greatest common divisor of multiple polynomials; this would aid the acquisition of minimum-order solutions to the PEVD despite the presence of polynomial eigenvectors of large order.

Of course, actually applying the new and improved PEVD algorithms discussed in this thesis in real-world application scenarios is perhaps the largest task for future work. For this, any of the applications detailed in Chapter 1 could be targeted. On a related note, the greatest common divisor extraction approach of [115] used in Section 5.5 could be extended to help with the scene recovery task in [30]. While this could facilitate the recovery of the magnitude of any transfer functions between an unknown source and an array of sensors, the phase information would have to be obtained from elsewhere. With this phase information, future work could continue the efforts of [119, 120] and attempt to estimate a room's geometry from its impulse response.

Appendix A

Existing PEVD Algorithms and Performance Metrics

A.1 Sequential Matrix Diagonalisation PEVD Algorithm

Within a reasonable number of iterations, the sequential matrix diagonalisation (SMD) polynomial matrix eigenvalue decomposition (PEVD) algorithm defined in [45] and its derivatives [47–50] are able to achieve superior levels of diagonalisation to other iterative PEVD methods. Given that Section 3.2.1 also identifies that SMD is the most computationally efficient iterative PEVD algorithm, particular focus is therefore given to SMD in this thesis. Further detail on the operation of SMD is therefore provided below, as knowledge of this algorithm may aid the reader's understanding of the novel contributions detailed in this thesis.

Below, Section A.1.1 provides a brief summary of the operations used in each iteration of SMD, and Section A.1.2 evaluates the computational complexity of the algorithm.

A.1.1 Algorithm Overview

The SMD algorithm approximates the PEVD using a series of elementary paraunitary operations to iteratively diagonalise a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{M \times M}$ and its associated coefficient matrix, $\mathbf{R}[\tau]$.

Upon initialisation, the algorithm diagonalises the lag zero coefficient matrix $\mathbf{R}[0]$ by means of its modal matrix $\mathbf{Q}^{(0)}$; i.e., $\boldsymbol{S}^{(0)}(z) = \mathbf{Q}^{(0)}\boldsymbol{R}(z)\mathbf{Q}^{(0)\mathrm{H}}$. The unitary $\mathbf{Q}^{(0)}$ — obtained from the EVD of the lag zero slice $\mathbf{R}[0]$ — is applied to all coefficient matrices $\mathbf{R}[\tau]\ \forall\ \tau$, and initialises $\boldsymbol{H}^{(0)}(z) = \mathbf{Q}^{(0)}$.

In the ith step, $i = 1, 2, \ldots \hat{I}$, the SMD algorithm computes

$$\begin{aligned} \boldsymbol{S}^{(i)}(z) &= \boldsymbol{U}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{U}}^{(i)}(z) \\ \boldsymbol{H}^{(i)}(z) &= \boldsymbol{U}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z) \quad , \end{aligned} \tag{A.1}$$

in which

$$\boldsymbol{U}^{(i)}(z) = \mathbf{Q}^{(i)}\boldsymbol{\Lambda}^{(i)}(z)\ . \tag{A.2}$$

The product in (A.2) consists of a paraunitary delay matrix

$$\boldsymbol{\Lambda}^{(i)}(z) = \mathrm{diag}\{\underbrace{1\ \ldots\ 1}_{k^{(i)}-1}\ z^{-\tau^{(i)}}\ \underbrace{1\ \ldots\ 1}_{M-k^{(i)}}\} \quad , \tag{A.3}$$

and a unitary matrix $\mathbf{Q}^{(i)}$, with the result that $\boldsymbol{U}^{(i)}(z)$ in (A.2) is paraunitary.

For subsequent discussion, it is convenient to define intermediate variables $\boldsymbol{S}^{(i)\prime}(z)$ and $\boldsymbol{H}^{(i)\prime}(z)$ as the outputs of a 'shifting' step, where

$$\begin{aligned} \boldsymbol{S}^{(i)\prime}(z) &= \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{\Lambda}}^{(i)}(z) \\ \boldsymbol{H}^{(i)\prime}(z) &= \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z) \quad . \end{aligned} \tag{A.4}$$

A further 'rotation' (or 'update') step obtains the updated polynomial matrices

$$\begin{aligned} \boldsymbol{S}^{(i)}(z) &= \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i)\mathrm{H}} \\ \boldsymbol{H}^{(i)}(z) &= \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime}(z) \quad . \end{aligned} \tag{A.5}$$

Matrices $\boldsymbol{\Lambda}^{(i)}(z)$ and $\mathbf{Q}^{(i)}$ are selected based on the position of the dominant off-diagonal

column in $\boldsymbol{S}^{(i-1)}(z) \bullet\!\!-\!\!\circ\, \mathbf{S}^{(i-1)}[\tau]$, as identified by the parameter set

$$\{k^{(i)}, \tau^{(i)}\} = \arg\max_{k,\tau} \|\hat{\mathbf{s}}_k^{(i-1)}[\tau]\|_2 \quad , \tag{A.6}$$

where

$$\|\hat{\mathbf{s}}_k^{(i-1)}[\tau]\|_2 = \sqrt{\sum_{m=1, m\neq k}^{M} |s_{m,k}^{(i-1)}[\tau]|^2} \tag{A.7}$$

and $s_{m,k}^{(i-1)}[\tau]$ represents the element in the mth row and kth column of the coefficient matrix $\mathbf{S}^{(i-1)}[\tau]$.

The 'shifting' process in (A.4) moves the dominant off-diagonal row and column into the lag zero coefficient matrix $\mathbf{S}^{(i)\prime}[0]$. The off-diagonal energy in the shifted row and column is then transferred onto the diagonal by the unitary matrix $\mathbf{Q}^{(i)}$ in the 'rotation' (or 'update') step of (A.5). This matrix $\mathbf{Q}^{(i)}$ contains the eigenvectors that diagonalise $\mathbf{S}^{(i)\prime}[0]$ by means of an ordered EVD.

Truncation of outer coefficients of $\boldsymbol{S}^{(i)}(z)$ and $\boldsymbol{H}^{(i)}(z)$ with small Frobenius norm is used to limit growth in order. The SMD algoithm uses the truncation approach detailed in Appendix C.

Iterations of SMD — which has been shown to converge in [45] — continue for a maximum of $I_{\max}$ steps, or until $\boldsymbol{S}^{(I)}(z)$ is sufficiently diagonalised with dominant off-diagonal column norm

$$\max_{k,\tau} \|\hat{\mathbf{s}}_k^{(I)}[\tau]\|_2 \leq \epsilon \quad , \tag{A.8}$$

where the value of ϵ is chosen to be arbitrarily small. On completion, SMD generates an approximate PEVD given by

$$\boldsymbol{D}(z) = \boldsymbol{S}^{(\hat{I})}(z) = \boldsymbol{F}(z)\boldsymbol{R}(z)\tilde{\boldsymbol{F}}(z) \; , \tag{A.9}$$

where the parahermitian matrix $\boldsymbol{D}(z)$ approximates a diagonal matrix of polynomial eigenvalues, $\hat{I} = \min\{I, I_{\max}\}$, and a paraunitary matrix $\boldsymbol{F}(z)$ contains approximate

Input: $\boldsymbol{R}(z)$, $I_{\max}$, ϵ, μ, μ_t
Output: $\boldsymbol{D}(z)$, $\boldsymbol{F}(z)$
If μ_t not specified, $\mu_t \leftarrow \mu$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise $\mathbf{R}[0] \in \mathbb{C}^{M\times M}$
$\boldsymbol{S}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\boldsymbol{R}(z)\mathbf{Q}^{(0)\mathrm{H}}$; $\boldsymbol{H}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$; $i \leftarrow 0$; stop $\leftarrow 0$
do
  $i \leftarrow i+1$
  Find $\{k^{(i)}, \tau^{(i)}\}$ from (A.6); generate $\boldsymbol{\Lambda}^{(i)}(z)$ from (A.3)
  $\boldsymbol{S}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\tilde{\boldsymbol{\Lambda}}^{(i)}(z)$
  $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
  Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\mathbf{S}^{(i)\prime}[0]$
  $\boldsymbol{S}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{S}^{(i)\prime}(z)\mathbf{Q}^{(i)\mathrm{H}}$
  $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime}(z)$
  if $i > I_{\max}$ or (A.8) satisfied **then**
    stop $\leftarrow 1$;
  end
  Truncate $\boldsymbol{H}^{(i)}(z)$ using threshold μ_t according to Appendix C
  Truncate $\boldsymbol{S}^{(i)}(z)$ using threshold μ according to Appendix C
while stop $= 0$
$\boldsymbol{F}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$
$\boldsymbol{D}(z) \leftarrow \boldsymbol{S}^{(i)}(z)$

Algorithm 13: SMD algorithm

polynomial eigenvectors and is a concatenation of the paraunitary matrices:

$$\boldsymbol{F}(z) = \boldsymbol{H}^{(\hat{I})}(z) = \boldsymbol{U}^{(\hat{I})}(z)\cdots\boldsymbol{U}^{(0)}(z) = \prod_{i=0}^{\hat{I}}\boldsymbol{U}^{(\hat{I}-i)}(z)\,. \tag{A.10}$$

For reference purposes, pseudocode for the SMD algorithm is provided in Algorithm 13.

A.1.2 Algorithm Complexity

At the ith iteration, the length of $\boldsymbol{S}^{(i)\prime}(z)$ is equal to $L\{\boldsymbol{S}^{(i)\prime}(z)\}$, where $L\{\cdot\}$ computes the length of a polynomial matrix. For (A.5), every matrix-valued coefficient in $\boldsymbol{S}^{(i)\prime}(z)$ must be left- and right-multiplied with a unitary matrix. Accounting for a multiplication of 2 $M \times M$ matrices by M^3 multiply-accumulates (MACs) [2, 95], a total of $2L\{\boldsymbol{S}^{(i)\prime}(z)\}M^3$ MACs arise to generate $\boldsymbol{S}^{(i)}(z)$. Every matrix-valued coefficient in $\boldsymbol{H}^{(i)\prime}(z)$ must also be left-multiplied with a unitary matrix; thus, a total of

$L\{\boldsymbol{H}^{(i)\prime}(z)\}M^3$ MACs arise to generate $\boldsymbol{H}^{(i)}(z)$. If $\epsilon = 0$, the cumulative complexity of the SMD algorithm over $I_{\max}$ iterations can therefore be approximated as

$$C_{\text{SMD}}(I_{\max}) = M^3 \sum_{i=0}^{I_{\max}} (2L\{\boldsymbol{S}^{(i)\prime}(z)\} + L\{\boldsymbol{H}^{(i)\prime}(z)\}) \; . \tag{A.11}$$

A.2 Existing DFT-Based Approach to the PEVD

A discrete Fourier transform (DFT)-based PEVD formulation, which transforms the problem into a pointwise in frequency standard matrix decomposition, is provided in [53]. The method can either return a spectrally majorised decomposition, or attempt to compute smooth, approximately analytic, eigenvalues. The inherent drawback of a lack of phase-coherence between independent frequency bins [81] is solved via a quadratic non-linear minimisation problem, which encourages phase alignment between adjacent bins. A comparison between the characteristics of this approach versus the SMD algorithm — which is summarised in Section A.1 — is given in Section 5.2; a description of the former is therefore provided below for reference purposes.

Section A.2.1 gives a brief overview of the DFT-based approach defined in [53], while Sections A.2.2 and A.2.3 describe methods used within the algorithm to approximate analytic eigenvalues and adjust eigenvector phase.

A.2.1 Algorithm Overview

The approach in [53] uses a decomposition of the form

$$\mathbf{R}[k] = \mathbf{Q}[k]\mathbf{D}[k]\mathbf{Q}^{\text{H}}[k] \, , \; k = 0, 1, \ldots, K-1 \; , \tag{A.12}$$

where $\mathbf{Q}[k]$ contains the eigenvectors and $\mathbf{D}[k]$ contains the eigenvalues obtained from the K-point DFT of $\mathbf{R}[\tau]$,

$$\mathbf{R}[k] = \boldsymbol{R}(z)|_{z=\text{e}^{\text{j}\Omega_k}} = \sum_{\tau} \mathbf{R}[\tau]\text{e}^{-\text{j}\Omega_k \tau} \; , \; k = 0, 1, \ldots, K-1 \; . \tag{A.13}$$

An approximate PEVD is therefore obtained via K EVDs that are pointwise in frequency.

The PEVD in (2.4) corresponds to a linear convolution in the coefficient domain; however, the DFT-based decomposition here corresponds to the circular convolution

$$\mathbf{R}[\,((\tau))_K\,] = \mathbf{Q}[\,((\tau))_K\,] \circledast \mathbf{D}[\,((\tau))_K\,] \circledast \mathbf{Q}^{\mathrm{H}}[\,-((\tau))_K\,]\,, \tag{A.14}$$

where $\circledast$ is the circular convolution operator, and $((\tau))_K$ denotes τ modulo K. For (A.14) to be equivalent to (2.4), the number of frequency bins must satisfy

$$K \geq (2N + L - 2)\,, \tag{A.15}$$

where $L = (2T + 1)$ is the length of input parahermitian matrix $\boldsymbol{R}(z)$, which has maximum lag T, and N is the assumed length of the paraunitary matrix. That is, $\mathbf{Q}[\tau] = \mathbf{0}$ for $\tau \geq N$ and $\tau < 0$. Typically, choosing $K = (2N + L - 2)$ is valid, as decomposition accuracy does not increase significantly for larger K [53], but algorithmic computational complexity does.

At each frequency bin, eigenvalues are typically arranged in descending order; this results in approximate spectral majorisation of the polynomial eigenvalues. Section A.2.2 discusses the rearrangement of eigenvalues to approximate an analytic decomposition.

Each eigenvector in a conventional EVD may be influenced by an arbitrary scalar phase angle and still be valid. This ambiguity in phase of each eigenvector can lead to discontinuities in phase between adjacent frequency bins. For a short paraunitary matrix $\boldsymbol{Q}(z)$, these discontinuities must be smoothed. This is achieved through the use of a phase alignment function, described in Section A.2.3, which uses the 'dogleg' algorithm [111] to solve an unconstrained optimisation problem.

Following phase alignment, $\mathbf{Q}[\tau]$ is computed as

$$\mathbf{Q}[\tau] = \frac{1}{K} \sum_{k=0}^{K-1} \mathbf{Q}[k] \mathrm{e}^{\mathrm{j}\Omega_k \tau}\,,\ \tau = 0, 1, \ldots, N-1\,, \tag{A.16}$$

and $\boldsymbol{D}(z)$ is a diagonal parahermitian matrix with diagonal elements equal to the diagonal elements of $\tilde{\boldsymbol{Q}}(z)\boldsymbol{R}(z)\boldsymbol{Q}(z)$. Any energy in lags $\tau = N \dots K-1$ of $\mathbf{Q}[\tau]$ is ignored.

A.2.2 Reordering the Eigenvectors and Eigenvalues to Approximate an Analytic Decomposition

If strong decorrelation is required for an application, but spectral majorisation is not, then an analytic decomposition may be preferable. In an analytic decomposition, the eigenvalues and their eigenvectors are arranged such that non-differentiabilities and discontinuities, respectively, between adjacent frequency bins are minimised. Even for an arrangement that provides continuous eigenvalues, such discontinuities can occur when the eigenvalues intersect at some frequencies.

To approximate an analytic decomposition, the eigenvectors in adjacent frequency bins are rearranged using the inner product

$$\mathsf{c}_{mn}[k] = \mathbf{q}_m^{\mathrm{H}}[k-1]\mathbf{q}_n[k] \,, \tag{A.17}$$

where, $\mathbf{q}_m[k]$ is the mth column of $\mathbf{Q}[k]$. For each eigenvector $\mathbf{q}_m[k-1]$, $m = 1 \dots M$, a subsequent eigenvector $\mathbf{q}_{m'}[k]$ is chosen from an initial set $S = \{1 \dots M\}$ of the columns of $\mathbf{Q}[k]$ such that

$$m' = \arg\max_{n \in S} \{|\mathsf{c}_{mn}[k]|\} \,. \tag{A.18}$$

Once m' is identified, it is removed from the set: $S \leftarrow S \backslash \{m'\}$, and the next eigenvector is chosen. The selected eigenvectors are combined in a rearranged matrix $\mathbf{Q}'[k] = [\mathbf{q}_{1'}[k], \cdots, \mathbf{q}_{M'}[k]]$, and $\mathbf{Q}[k]$ is set equal to $\mathbf{Q}'[k]$. The eigenvalues $\mathbf{D}[k]$ are rearranged according to the reordering of the eigenvectors.

A.2.3 Adjusting the Phase of the Eigenvectors via Unconstrained Optimisation

Phase alignment of eigenvectors in adjacent frequency bins is vital for a compact, low order decomposition. Thus, if such a decomposition is sought (e.g., here it is desired

that only lags $\tau = 0 \dots N-1$ are non-zero), then phase alignment can be achieved by finding the phase changes required for each eigenvector $\mathbf{q}_m[k]\ \forall\ m,k$ such that the resulting eigenvectors are of low order.

The phase of the mth eigenvector at frequency bin k can be adjusted by an angle $\theta[k]$ according to $\mathbf{q}_m[k] \leftarrow \mathrm{e}^{\mathrm{j}\theta[k]}\mathbf{q}_m[k]$. For the mth polynomial eigenvector $\mathbf{q}_m[\tau]$ to be compact, it is required to find angles $\boldsymbol{\theta} = [\theta[1] \dots \theta[K-1]]^{\mathrm{T}}$, that satisfy

$$\mathbf{q}_m[\tau] = \frac{1}{K}\sum_{k=0}^{K-1} \mathbf{q}_m[k]\mathrm{e}^{\mathrm{j}\theta[k]}\mathrm{e}^{\mathrm{j}\Omega_k\tau} = \mathbf{0}\ , \tag{A.19}$$

for $\tau = N \dots K-1$. Without loss of generality, let $\theta[0] = 0$. Note that $\boldsymbol{\theta}$ does not include $\theta[0]$. Equation (A.19) can be expressed as

$$\mathcal{F}_N\mathbf{x}(\boldsymbol{\theta}) + f_N = \mathbf{0}\ , \tag{A.20}$$

where $\mathbf{x}(\boldsymbol{\theta}) = [\mathrm{e}^{\mathrm{j}\theta[1]}, \mathrm{e}^{\mathrm{j}\theta[2]}, \dots, \mathrm{e}^{\mathrm{j}\theta[K-1]}]^{\mathrm{T}}$, $f_N = [\mathbf{q}_m^{\mathrm{T}}[0], \mathbf{q}_m^{\mathrm{T}}[0], \dots, \mathbf{q}_m^{\mathrm{T}}[0]]^{\mathrm{T}}$ is a $M(K-N)\times 1$ vector, and

$$\mathcal{F}_N = \begin{bmatrix} \mathbf{q}_m[1]w_K^{-N} & \mathbf{q}_m[2]w_K^{-2N} & \dots & \mathbf{q}_m[K-1]w_K^{-(K-1)N} \\ \mathbf{q}_m[1]w_K^{-(N+1)} & \mathbf{q}_m[2]w_K^{-2(N+1)} & \dots & \mathbf{q}_m[K-1]w_K^{-(K-1)(N+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{q}_m[1]w_K^{-(K-1)} & \mathbf{q}_m[2]w_K^{-2(K-1)} & \dots & \mathbf{q}_m[K-1]w_K^{-(K-1)^2} \end{bmatrix} \tag{A.21}$$

is a $M(K-N)\times(K-1)$ matrix, with $w_K = \mathrm{e}^{\mathrm{j}2\pi/K}$.

In general, there may exist no phase vector $\boldsymbol{\theta}$ which satisfies (A.20). However, by minimising the energy in the coefficients for $\tau = N, \dots, K-1$, some $\boldsymbol{\theta}$ can be obtained. The energy in these coefficients,

$$J(\boldsymbol{\theta}) = \|\mathcal{F}_N\mathbf{x}(\boldsymbol{\theta}) + f_N\|_2^2\ , \tag{A.22}$$

is therefore used as the objective of an unconstrained minimisation problem. In [53], it was found that the 'dogleg' trust-region strategy for unconstrained minimisation [111] was able to satisfactorily identify the $\boldsymbol{\theta}$ that minimised $J(\boldsymbol{\theta})$ for $m = 1 \dots M$.

Of course, a disadvantage of this method is that it requires an estimate of N prior to execution. If N is not sufficiently high, convergence of the 'dogleg' algorithm can be poor, resulting in insufficient eigenvector phase alignment and a PEVD with a high degree of decomposition error.

A.3 Performance Metrics

This section defines several metrics used throughout this thesis to evaluate the quality of a PEVD. Section A.3.1 defines the well-established diagonalisation metric from [45] and Section A.3.2 adapts this metric to the case of polynomial matrix upper-triangularisation. Subsequently, Section A.3.3 establishes a metric to measure the accuracy of the polynomial eigenvalues obtained from a PEVD, Section A.3.4 defines a metric for evaluating the decomposition mean square error of a PEVD, and Section A.3.5 describes a paraunitarity metric for the polynomial eigenvectors from a PEVD. Finally, Section A.3.6 defines an intuitive metric that records paraunitary matrix length.

A.3.1 Normalised Off-Diagonal Energy

Polynomial matrix eigenvalue decomposition algorithms iteratively minimise off-diagonal energy. To measure their performance, a suitable normalised metric $E_{\text{diag}}^{(i)}$ [45] is used, which divides the off-diagonal energy in the iteratively updated parahermitian matrix $\boldsymbol{S}^{(i)}(z) \bullet\!\!-\!\!\circ\, \mathbf{S}^{(i)}[\tau] \in \mathbb{C}^{M\times M}$ at the ith algorithm iteration by the total energy in the original parahermitian matrix $\boldsymbol{R}(z)$. The metric is defined as

$$E_{\text{diag}}^{(i)} = \frac{\sum_\tau \sum_{k=1}^{M} \|\hat{\mathbf{s}}_k^{(i)}[\tau]\|_2^2}{\sum_\tau \|\mathbf{R}[\tau]\|_{\text{F}}^2}\,, \tag{A.23}$$

with

$$\|\hat{\mathbf{s}}_k^{(i)}[\tau]\|_2^2 = \sum_{m=1,m\neq k}^{P} |s_{m,k}^{(i)}[\tau]|^2 \tag{A.24}$$

where $s_{m,k}^{(i)}[\tau]$ represents the element in the mth row and kth column of $\mathbf{S}^{(i)}[\tau]$.

Computation of $E_{\text{diag}}^{(i)}$ generates squared covariance terms; therefore a logarithmic notation of $5\log_{10} E_{\text{diag}}^{(i)}$ is employed. Note that diagonalisation metric E_{diag}, which

omits the iteration parameter i, measures the diagonalisation obtained by a PEVD algorithm following the completion of all iterations.

A.3.2 Normalised Below-Diagonal Energy

Polynomial matrix QR decomposition algorithms iteratively minimise lower-triangular energy. To measure their performance, a suitable normalised metric $E_{\text{tri}}^{(i)}$ is used, which divides the lower-triangular energy in the iteratively updated polynomial matrix $\boldsymbol{A}^{(i)}(z) \bullet\!\!-\!\!\circ\, \mathbf{A}^{(i)}[\tau] \in \mathbb{C}^{P\times Q}$ at the ith algorithm iteration by the total energy in the original matrix $\boldsymbol{A}(z)$. The metric is defined as

$$E_{\text{tri}}^{(i)} = \frac{\sum_{\tau} \sum_{k=1}^{\min\{(P-1),Q\}} \|\hat{\mathbf{a}}_k^{(i)}[\tau]\|_2^2}{\sum_{\tau} \|\mathbf{A}[\tau]\|_{\text{F}}^2} \,, \tag{A.25}$$

with

$$\|\hat{\mathbf{a}}_k^{(i)}[\tau]\|_2^2 = \sum_{j=k+1}^{P} |a_{j,k}^{(i)}[\tau]|^2 \,, \tag{A.26}$$

where $a_{j,k}^{(i)}[\tau]$ represents the element in the jth row and kth column of $\mathbf{A}^{(i)}[\tau]$.

Note that for demonstration purposes, the iteration parameter i describes the running total of iterations, and is not reset, e.g., at the beginning of each column step of the polynomial matrix QR decomposition by columns algorithm from [42].

Unlike the computation in (A.23), (A.25) does not necessarily square covariance terms, therefore a logarithmic metric of $10\log_{10} E_{\text{tri}}^{(i)}$ is used.

A.3.3 Eigenvalue Resolution

The eigenvalue resolution is defined as the mean normalised absolute error between the ground truth and measured eigenvalue PSDs:

$$\lambda_{\text{res}} = \frac{1}{MK} \sum_{m=1}^{M} \sum_{k=0}^{K-1} \left| \frac{\mathbf{D}_{m,m}[k] - \hat{\mathbf{W}}_{m,m}[k]}{\hat{\mathbf{W}}_{m,m}[k]} \right| \,, \tag{A.27}$$

where $\mathbf{D}[k]$ is obtained from the K-point DFT of eigenvalues $\mathbf{D}[\tau] \in \mathbb{C}^{M\times M}$ computed by a PEVD and $\hat{\mathbf{W}}[k]$ is found by appropriately associating values of the K-point DFT

of ground truth eigenvalues $\mathbf{W}[\tau]$. For example, if $\mathbf{D}[k]$ is obtained from an iterative time domain PEVD algorithm and therefore approximately spectrally majorised for all k, the eigenvalues in $\hat{\mathbf{W}}[k]$ will be ordered such that they are also spectrally majorised. A suitable K is identified as the smallest power of two greater than the lengths of $\boldsymbol{D}(z)$ and $\boldsymbol{W}(z)$. The normalisation in (A.27) will give emphasis to the correct extraction of small eigenvalues in the presence of stronger ones, and is therefore similar to the coding gain metric in [25].

A.3.4 Decomposition Mean Square Error

Denote the mean square reconstruction error for an approximate PEVD as

$$\text{MSE} = \frac{1}{M^2 L'} \sum\nolimits_{\tau} \|\mathbf{E}_R[\tau]\|_{\text{F}}^2 \ , \tag{A.28}$$

where $\mathbf{E}_R[\tau] = \hat{\mathbf{R}}[\tau] - \mathbf{R}[\tau]$, $\mathbf{R}[\tau] \in \mathbb{C}^{M\times M}$, $\hat{\boldsymbol{R}}(z) = \tilde{\boldsymbol{F}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)$, and L' is the length of $\boldsymbol{E}_R(z)$. Matrices $\boldsymbol{D}(z)$ and $\boldsymbol{F}(z) = \tilde{\boldsymbol{Q}}(z)$ are polynomial eigenvalues and eigenvectors computed by a PEVD.

A.3.5 Paraunitarity Error

The paraunitarity error for polynomial eigenvectors $\boldsymbol{F}(z) = \tilde{\boldsymbol{Q}}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ obtained by a PEVD is defined as

$$\eta = \frac{1}{M} \sum\nolimits_{\tau} \|\mathbf{E}_F[\tau] - \mathbf{I}_{\text{M}}[\tau]\|_{\text{F}}^2 \ , \tag{A.29}$$

where $\boldsymbol{E}_F(z) = \boldsymbol{F}(z)\tilde{\boldsymbol{F}}(z)$, $\mathbf{I}_{\text{M}}[0]$ is an $M \times M$ identity matrix, and $\mathbf{I}_{\text{M}}[\tau]$ for $\tau \neq 0$ is an $M \times M$ matrix of zeroes.

A.3.6 Paraunitary Filter Length

The paraunitary matrix $\boldsymbol{F}(z) = \tilde{\boldsymbol{Q}}(z)$ output by a PEVD algorithm can be implemented as a lossless bank of finite impulse response filters in signal processing applications; a useful metric for gauging the implementation cost of this matrix is its length, $L_F = L_Q$, which — via the definitions in Section 2.1 — directly relates to the order of the filters.

Appendix B

Broadband Randomised Source Model

A randomised source model from [45] generates a parahermitian matrix $\boldsymbol{R}(z) = \tilde{\boldsymbol{Q}}(z)\boldsymbol{D}(z)\boldsymbol{Q}(z)$, where $\boldsymbol{D}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ is a diagonal parahermitian matrix containing the PSDs of M independent sources. These sources are spectrally shaped by innovation filters such that $\boldsymbol{D}(z)$ has an order of O_D, with a restriction on the placement of zeros to limit the dynamic range of the PSDs to about Δ_{DR} dB. A random paraunitary matrix $\boldsymbol{Q}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$ of order O_Q performs a convolutive mixing of these sources, such that $\boldsymbol{R}(z)$ has a full polynomial rank and an order of $O_R = 2O_Q + O_D$. Assuming that $\boldsymbol{Q}(z)$ is of minimum order — i.e., it cannot be reduced to the product of a paraunitary matrix and a diagonal paraunitary matrix — an ideal PEVD algorithm will recover $\boldsymbol{D}(z)$ and $\boldsymbol{Q}(z)$ (or equivalently, $\boldsymbol{F}(z) = \tilde{\boldsymbol{Q}}(z)$) given $\boldsymbol{R}(z)$.

Appendix C

State-of-the-Art in Polynomial Matrix Truncation

The polynomial matrix truncation method from [61] is described here. This approach reduces the order of a polynomial matrix $\boldsymbol{Y}(z)$ — which has minimum lag T_1 and maximum lag T_2 — by removing the $T_3(\mu)$ leading and $T_4(\mu)$ trailing lags using a trim function

$$f_{\text{trim}}(\mathbf{Y}[\tau], \mu) = \begin{cases} \mathbf{Y}[\tau], & T_1 + T_3(\mu) \leq \tau \leq T_2 - T_4(\mu) \\ \mathbf{0}, & \text{otherwise} . \end{cases} \tag{C.1}$$

The amount of energy lost by removing the $T_3(\mu)$ leading and $T_4(\mu)$ trailing lags of $\mathbf{Y}[\tau]$ via the $f_{\text{trim}}(\cdot)$ operation is measured by

$$\gamma_{\text{trim}} = 1 - \frac{\sum_\tau \|f_{\text{trim}}(\mathbf{Y}[\tau], \mu)\|_{\text{F}}^2}{\sum_\tau \|\mathbf{Y}[\tau]\|_{\text{F}}^2} , \tag{C.2}$$

where $\|\cdot\|_{\text{F}}$ is the Frobenius norm. A parameter μ is used to provide an upper bound for γ_{trim}. The truncation procedure can be expressed as the constrained optimisation problem:

$$\text{maximise } \{T_3(\mu) + T_4(\mu)\} , \quad \text{s.t.} \quad \gamma_{\text{trim}} \leq \mu . \tag{C.3}$$

This is implemented by removing the outermost matrix coefficients of matrix $\boldsymbol{Y}(z)$ until γ_{trim} approaches μ from below. Note that if $\boldsymbol{Y}(z)$ is parahermitian, $T_1 = -T_2$ and $T_3(\mu) = T_4(\mu)$ due to symmetry.

Appendix D

Spatio-Spectral MUSIC Algorithm

The spatio-spectral polynomial MUSIC (SSP-MUSIC) algorithm [18] is an extension of narrowband MUSIC [77] to the broadband case. The idea of the SSP-MUSIC algorithm is to scan the noise-only subspace $\boldsymbol{Q}_n(z) = \tilde{\boldsymbol{F}}_n(z) = [\boldsymbol{q}_{R+1}(z) \ \ldots \ \boldsymbol{q}_M(z)] : \mathbb{C} \rightarrow \mathbb{C}^{M\times(M-R)}$, which is spanned by eigenvectors corresponding to eigenvalues close to the noise floor, $\boldsymbol{D}_n(z) \approx \sigma_v^2 \mathbf{I}_{M-R}$. The steering vectors of the R sources that contribute to $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{M\times M}$ will define the signal-plus-noise subspace $\boldsymbol{Q}_s(z) = \tilde{\boldsymbol{F}}_s(z)$ and therefore lie in the nullspace of its complement $\boldsymbol{Q}_n(z)$. As a result, the vector $\tilde{\boldsymbol{Q}}_n(\mathrm{e}^{\mathrm{j}\Omega})\boldsymbol{a}_{\vartheta,\varphi}(\mathrm{e}^{\mathrm{j}\Omega})$ has to be close to the origin for $\boldsymbol{a}_{\vartheta,\varphi}(\mathrm{e}^{\mathrm{j}\Omega})$ to be a steering vector of a contributing source at frequency Ω, where $\tilde{\boldsymbol{Q}}_n(\mathrm{e}^{\mathrm{j}\Omega}) = \tilde{\boldsymbol{Q}}_n(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega}}$ and $\boldsymbol{a}_{\vartheta,\varphi}(\mathrm{e}^{\mathrm{j}\Omega}) = \boldsymbol{a}_{\vartheta,\varphi}(z)|_{z=\mathrm{e}^{\mathrm{j}\Omega}}$. Thus, the SSP-MUSIC algorithm evaluates the reciprocal of the norm of this vector,

$$P_{\mathrm{SSP}}(\vartheta,\varphi,\Omega) = \frac{1}{\tilde{\boldsymbol{a}}_{\vartheta,\varphi}(z)\boldsymbol{Q}_n(z)\tilde{\boldsymbol{Q}}_n(z)\boldsymbol{a}_{\vartheta,\varphi}(z)}\Big|_{z=\mathrm{e}^{\mathrm{j}\Omega}}, \tag{D.1}$$

which is large when $\boldsymbol{a}_{\vartheta,\varphi}(\mathrm{e}^{\mathrm{j}\Omega})$ is a steering vector of a contributing source. In addition to estimating the spatial direction of sources in terms of azimuth, ϑ, and elevation, φ, $P_{\mathrm{SSP}}(\vartheta,\varphi,\Omega)$ can determine over which frequency range sources in the direction defined by the steering vector $\boldsymbol{a}_{\vartheta,\varphi}(z)$ are active. If φ is assumed or known to be zero, the notation $P_{\mathrm{SSP}}(\vartheta,\Omega)$ is used.

Bibliography

[1] G. W. Stewart, "The decompositional approach to matrix computation," *Computing in Science & Engineering*, vol. 2, no. 1, pp. 50–59, 2000.

[2] G. Golub and C. V. Loan, *Matrix computations*, 4th ed. Baltimore, Maryland: John Hopkins University Press, 2013.

[3] I. Gohberg, P. Lancaster, and L. Rodman, *Matrix Polynomials*. New York: Academic Press, 1982.

[4] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs: Prentice Hall, 1993.

[5] J. G. McWhirter and P. D. Baxter, "A Novel Technqiue for Broadband SVD," in *12th Annual Workshop on Adaptive Sensor Array Processing*, MIT Lincoln Labs, Cambridge, MA, 2004.

[6] J. G. McWhirter, P. D. Baxter, T. Cooper, S. Redif, and J. Foster, "An EVD Algorithm for Para-Hermitian Polynomial Matrices," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 2158–2169, May 2007.

[7] S. Icart and P. Comon, "Some properties of Laurent polynomial matrices," in *IMA International Conference on Signal Processing in Mathematics*, December 2012.

[8] P. P. Vaidyanathan, "Theory of optimal orthonormal subband coders," *IEEE Transactions on Signal Processing*, vol. 46, no. 6, pp. 1528–1543, June 1998.

[9] S. Weiss, J. Pestana, and I. K. Proudler, "On the existence and uniqueness of the eigenvalue decomposition of a parahermitian matrix," *IEEE Transactions on Signal Processing*, vol. 66, no. 10, pp. 2659–2672, May 2018.

[10] S. Weiss, J. Pestana, I. K. Proudler, and F. K. Coutts, "Corrections to 'On the existence and uniqueness of the eigenvalue decomposition of a parahermitian matrix'," *IEEE Transactions on Signal Processing*, vol. 66, no. 23, pp. 6325–6327, December 2018.

[11] C. Delaosa, F. K. Coutts, J. Pestana, and S. Weiss, "Impact of space-time covariance estimation errors on a parahermitian matrix EVD," in *IEEE Workshop on Sensor Array and Multichannel Signal Processing*, July 2018, pp. 164–168.

[12] C. H. Ta and S. Weiss, "A design of precoding and equalisation for broadband MIMO systems," in *Proc. Asilomar Conf. Signals, Systems and Computers*, November 2007, pp. 1616–1620.

[13] R. Brandt and M. Bengtsson, "Wideband MIMO channel diagonalization in the time domain," in *Proc. Int. Symp. Pers., Indoor, Mobile Radio Commun.*, September 2011, pp. 1958–1962.

[14] N. Moret, A. Tonello, and S. Weiss, "MIMO precoding for filter bank modulation systems based on PSVD," in *Proc. IEEE 73rd Veh. Technol. Conf.*, May 2011.

[15] C. H. Ta and S. Weiss, "A jointly optimal precoder and block decision feedback equaliser design with low redundancy," in *15th European Signal Processing Conference*, September 2007, pp. 489–492.

[16] J. Foster, J. G. McWhirter, S. Lambotharan, I. K. Proudler, M. Davies, and J. Chambers, "Polynomial matrix QR decomposition for the decoding of frequency selective multiple-input multiple-output communication channels," *IET Signal Processing*, vol. 6, no. 7, pp. 704–712, September 2012.

[17] S. Weiss, M. Alrmah, S. Lambotharan, J. McWhirter, and M. Kaveh, "Broadband angle of arrival estimation methods in a polynomial matrix decomposition frame-

work," in *IEEE 5th Int. Workshop Comp. Advances in Multi-Sensor Adaptive Process.*, December 2013, pp. 109–112.

[18] M. Alrmah, S. Weiss, and S. Lambotharan, "An extension of the MUSIC algorithm to broadband scenarios using polynomial eigenvalue decomposition," in *19th European Signal Processing Conference*, August 2011, pp. 629–633.

[19] M. Alrmah, J. Corr, A. Alzin, K. Thompson, and S. Weiss, "Polynomial subspace decomposition for broadband angle of arrival estimation," in *Sensor Signal Processing for Defence Conference*, September 2014.

[20] F. K. Coutts, K. Thompson, S. Weiss, and I. K. Proudler, "Impact of fast-converging PEVD algorithms on broadband AoA estimation," in *Sensor Signal Processing for Defence Conference*, December 2017.

[21] S. Redif, J. G. McWhirter, P. D. Baxter, and T. Cooper, "Robust broadband adaptive beamforming via polynomial eigenvalues," in *Proc. IEEE/MTS OCEANS*, September 2006.

[22] S. Weiss, S. Bendoukha, A. Alzin, F. K. Coutts, I. K. Proudler, and J. Chambers, "MVDR broadband beamforming using polynomial matrix techniques," in *23rd European Signal Processing Conference*, September 2015, pp. 839–843.

[23] A. Alzin, F. K. Coutts, J. Corr, S. Weiss, I. K. Proudler, and J. A. Chambers, "Adaptive broadband beamforming with arbitrary array geometry," in *IET/EURASIP ISP*, December 2015.

[24] ——, "Polynomial matrix formulation-based Capon beamformer," in *IMA International Conference on Signal Processing in Mathematics*, December 2016.

[25] S. Redif, J. McWhirter, and S. Weiss, "Design of FIR paraunitary filter banks for subband coding using a polynomial eigenvalue decomposition," *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5253–5264, November 2011.

[26] S. Weiss, "On the design of oversampled filter banks for channel coding," in *12th European Signal Processing Conference*, September 2004, pp. 885–888, invited paper.

[27] S. Weiss, S. Redif, T. Cooper, C. Liu, P. D. Baxter, and J. G. McWhirter, "Paraunitary oversampled filter bank design for channel coding," *EURASIP Journal of Applied Signal Processing*.

[28] S. Redif, "Convolutive blind signal separation via polynomial matrix generalised eigenvalue decomposition," *Electronics Letters*, vol. 53, no. 2, pp. 87–89, 2017.

[29] J. Corr, J. Pestana, S. Weiss, I. K. Proudler, S. Redif, and M. Moonen, "Investigation of a polynomial matrix generalised EVD for multi-channel Wiener filtering," in *Proc. Asilomar Conf. Signals, Systems and Computers*, November 2016, pp. 1354–1358.

[30] S. Weiss, N. J. Goddard, S. Somasundaram, I. K. Proudler, and P. A. Naylor, "Identification of broadband source-array responses from sensor second order statistics," in *Sensor Signal Processing for Defence Conference*, December 2017.

[31] C. H. Ta and S. Weiss, "Design of precoding and equalisation for broadband MIMO transmission," in *IEE/EURASIP Conference on DSPenabledRadio*, September 2005.

[32] W. Al-Hanafy, A. P. Millar, C. H. Ta, and S. Weiss, "Broadband SVD and non-linear precoding applied to broadband MIMO channels," in *Proc. Asilomar Conf. Signals, Systems and Computers*, October 2008, pp. 2053–2057.

[33] W. Al-Hanafy and S. Weiss, "Comparison of precoding methods for broadband MIMO systems," in *IEEE 3rd Int. Workshop Comp. Advances in Multi-Sensor Adaptive Process.*, 2009.

[34] S. Weiss, P. Yarr, W. Al-Hanafy, A. Millar, and C.-H. Ta, "An oversampled modulated filter bank transmultiplexer with precoding and equalisation," in *3rd Workshop on Power Line Communications*, October 2009.

[35] A. Sandmann, A. Ahrens, and S. Lochmann, "Resource allocation in SVD-assisted optical MIMO systems using polynomial matrix factorization," in *Proc. 16. ITG Symp. Photon. Netw.*, May 2015.

[36] A. Ahrens, A. Sandmann, E. Auer, and S. Lochmann, "Optimal power allocation in zero-forcing assisted PMSVD-based optical MIMO systems," in *Sensor Signal Processing for Defence Conference*, December 2017.

[37] A. A. Nagy and S. Weiss, "Channel equalisation of a MIMO FBMC/OQAM system using a polynomial matrix pseudo-inverse," in *10th IEEE Workshop on Sensor Array and Multichannel Signal Processing*, July 2018, pp. 568–572.

[38] X. Mestre and D. Gregoratti, "A parallel processing approach to filterbank multicarrier MIMO transmission under strong frequency selectivity," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2014, pp. 8078–8082.

[39] A. I. Prez-Neira, M. Caus, R. Zakaria, D. L. Ruyet, E. Kofidis, M. Haardt, X. Mestre, and Y. Cheng, "MIMO signal processing in offset-QAM based filter bank multicarrier systems," *IEEE Transactions on Signal Processing*, vol. 64, no. 21, pp. 5733–5762, November 2016.

[40] J. A. Foster, J. G. McWhirter, and J. A. Chambers, "An algorithm for computing the QR decomposition of a polynomial matrix," in *Proc. 15th Int. Conf. Digital Signal Process.*, July 2007, pp. 71–74.

[41] ——, "A polynomial matrix QR decomposition with application to MIMO channel equalisation," in *Proc. Asilomar Conf. Signals, Systems and Computers*, November 2007, pp. 1379–1383.

[42] ——, "A novel algorithm for calculating the QR decomposition of a polynomial matrix," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 2009, pp. 3177–3180.

[43] J. A. Foster, J. G. McWhirter, M. R. Davies, and J. A. Chambers, "An algorithm for calculating the QR and singular value decompositions of polynomial matrices," *IEEE Transactions on Signal Processing*, vol. 58, no. 3, pp. 1263–1274, March 2010.

[44] J. G. McWhirter, "An algorithm for polynomial matrix SVD based on generalised Kogbetliantz transformations," in *18th European Signal Processing Conference*, August 2010, pp. 457–461.

[45] S. Redif, S. Weiss, and J. McWhirter, "Sequential matrix diagonalization algorithms for polynomial EVD of parahermitian matrices," *IEEE Transactions on Signal Processing*, vol. 63, no. 1, pp. 81–89, January 2015.

[46] Z. Wang, J. G. McWhirter, J. Corr, and S. Weiss, "Multiple shift second order sequential best rotation algorithm for polynomial matrix EVD," in *23rd European Signal Processing Conference*, September 2015, pp. 844–848.

[47] J. Corr, K. Thompson, S. Weiss, J. McWhirter, S. Redif, and I. K. Proudler, "Multiple shift maximum element sequential matrix diagonalisation for parahermitian matrices," in *IEEE Workshop on Statistical Signal Processing*, June 2014, pp. 312–315.

[48] J. Corr, K. Thompson, S. Weiss, J. G. McWhirter, and I. K. Proudler, "Causality-Constrained multiple shift sequential matrix diagonalisation for parahermitian matrices," in *22nd European Signal Processing Conference*, September 2014, pp. 1277–1281.

[49] J. Corr, K. Thompson, S. Weiss, J. McWhirter, and I. K. Proudler, "Cyclic-by-row approximation of iterative polynomial EVD algorithms," in *Sensor Signal Processing for Defence Conference*, September 2014.

[50] J. Corr, K. Thompson, S. Weiss, I. K. Proudler, and J. G. McWhirter, "Reduced search space multiple shift maximum element sequential matrix diagonalisation algorithm," in *IET Int. Conf. Intelligent Sig. Process.*, December 2015.

[51] A. Tkacenko and P. P. Vaidyanathan, "Iterative greedy algorithm for solving the FIR paraunitary approximation problem," *IEEE Transactions on Signal Processing*, vol. 54, no. 1, pp. 146–160, January 2006.

[52] A. Tkacenko, "Approximate eigenvalue decomposition of para-Hermitian systems through successive FIR paraunitary transformations," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, March 2010, pp. 4074–4077.

[53] M. Tohidian, H. Amindavar, and A. M. Reza, "A DFT-based approximate eigenvalue and singular value decomposition of polynomial matrices," *EURASIP J. Adv. Signal Process.*, vol. 93, December 2013.

[54] S. Weiss, I. K. Proudler, F. K. Coutts, and J. Pestana, "Iterative approximation of analytic eigenvalues of a parahermitian matrix EVD," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2019.

[55] J. G. McWhirter and Z. Wang, "A novel insight to the SBR2 algorithm for diagonalising para-Hermitian matrices," in *IMA International Conference on Signal Processing in Mathematics*, December 2016.

[56] F. K. Coutts, K. Thompson, I. Proudler, and S. Weiss, "A comparison of iterative and DFT-based polynomial matrix eigenvalue decompositions," in *IEEE 7th Int. Workshop Comp. Advances in Multi-Sensor Adaptive Process.*, December 2017.

[57] S. Kasap and S. Redif, "FPGA-based design and implementation of an approximate polynomial matrix EVD algorithm," in *2012 International Conference on Field-Programmable Technology*, Dec 2012, pp. 135–140.

[58] ——, "FPGA implementation of a second-order convolutive blind signal separation algorithm," in *2013 21st Signal Processing and Communications Applications Conference*, April 2013.

[59] ——, "Novel field-programmable gate array architecture for computing the eigenvalue decomposition of para-hermitian polynomial matrices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 522–536, March 2014.

[60] J. Foster, J. G. McWhirter, and J. Chambers, "Limiting the order of polynomial matrices within the SBR2 algorithm," in *IMA International Conference on Signal Processing in Mathematics*, December 2006.

[61] C. H. Ta and S. Weiss, "Shortening the order of paraunitary matrices in SBR2 algorithm," in *Int. Conf. Information, Comm. and Sig. Process.*, December 2007.

[62] J. Corr, K. Thompson, S. Weiss, I. K. Proudler, and J. McWhirter, "Row-shift corrected truncation of paraunitary matrices for PEVD algorithms," in *23rd European Signal Processing Conference*, September 2015, pp. 849–853.

[63] ——, "Shortening of paraunitary matrices obtained by polynomial eigenvalue decomposition algorithms," in *Sensor Signal Processing for Defence Conference*, September 2015.

[64] F. K. Coutts, K. Thompson, S. Weiss, and I. K. Proudler, "Analysing the performance of divide-and-conquer sequential matrix diagonalisation for large broadband sensor arrays," in *IEEE International Workshop on Signal Processing Systems*, October 2017.

[65] J. Corr, K. Thompson, S. Weiss, J. McWhirter, and I. K. Proudler, "Performance trade-offs in sequential matrix diagonalisation search strategies," in *IEEE 6th Int. Workshop Comp. Advances in Multi-Sensor Adaptive Process.*, December 2015.

[66] J. Corr, *Advanced Algorithms for Polynomial Matrix Eigenvalue Decomposition.* Glasgow, UK: Ph.D. dissertation, Univ. of Strathclyde, 2017.

[67] F. K. Coutts, J. Corr, K. Thompson, S. Weiss, I. K. Proudler, and J. G. McWhirter, "Memory and complexity reduction in parahermitian matrix manip-

ulations of PEVD algorithms," in *24th European Signal Processing Conference*, August 2016, pp. 1633–1637.

[68] ——, "Complexity and search space reduction in cyclic-by-row PEVD algorithms," in *Proc. Asilomar Conf. Signals, Systems and Computers*, November 2016, pp. 1349–1353.

[69] F. K. Coutts, K. Thompson, I. Proudler, and S. Weiss, "Restricted update sequential matrix diagonalisation for parahermitian matrices," in *IEEE 7th Int. Workshop Comp. Advances in Multi-Sensor Adaptive Process.*, December 2017.

[70] F. K. Coutts, J. Corr, K. Thompson, S. Weiss, I. K. Proudler, and J. McWhirter, "Multiple shift QR decomposition for polynomial matrices," in *IMA International Conference on Signal Processing in Mathematics*, December 2016.

[71] F. K. Coutts, J. Corr, K. Thompson, I. K. Proudler, and S. Weiss, "Divide-and-conquer sequential matrix diagonalisation for parahermitian matrices," in *Sensor Signal Processing for Defence Conference*, December 2017.

[72] F. K. Coutts, I. K. Proudler, and S. Weiss, "Efficient implementation of iterative polynomial matrix EVD algorithms exploiting structural redundancy and parallelisation," *IEEE Transactions on Circuits and Systems I*, Submitted March 2019.

[73] F. K. Coutts, K. Thompson, J. Pestana, I. K. Proudler, and S. Weiss, "Enforcing eigenvector smoothness for a compact DFT-based polynomial eigenvalue decomposition," in *IEEE Workshop on Sensor Array and Multichannel Signal Processing*, July 2018, pp. 159–163.

[74] F. K. Coutts, K. Thompson, I. K. Proudler, and S. Weiss, "An iterative DFT-based approach to the polynomial matrix eigenvalue decomposition," in *Proc. Asilomar Conf. Signals, Systems and Computers*, October 2018, pp. 1011–1015.

[75] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs: Prentice Hall, 2002.

[76] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer, 2002.

[77] R. O. Schmidt, "Multiple emitter location and signal parameter estimation," *IEEE Transactions Ant. Prop.*, vol. 34, no. 3, pp. 276–280, Mar. 1986.

[78] K. J. Pope and R. E. Bogner, "Blind signal separation II: Linear, convolutive combinations," *Digital Signal Processing*, vol. 6, no. 1, pp. 17–28, January 1996.

[79] T. Kailath, *Linear Systems*. Englewood Cliffs: Prentice Hall, 1980.

[80] A. Paulraj, R. Nabar, and D. Gore, *Introduction to Space-Time Wireless Communications*. Cambridge, UK: Cambridge University Press, 2003.

[81] R. Klemm, *Space-time Adaptive Processing Principles and Applications*. Institution of Electrical Engineers, 1998.

[82] P. A. Regalia and D.-Y. Huang, "Attainable error bounds in multirate adaptive lossless FIR filters," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, May 1995, pp. 1460–1463.

[83] S. Redif, S. Weiss, and J. G. McWhirter, "An approximate polynomial matrix eigenvalue decomposition algorithm for para-Hermitian matrices," in *Proc. 11th IEEE Int. Symp. Signal Process. Inf. Technol.*, December 2011, pp. 421–425.

[84] J. Corr, K. Thompson, S. Weiss, J. G. McWhirter, and I. K. Proudler, "Maximum energy sequential matrix diagonalisation for parahermitian matrices," in *Proc. Asilomar Conf. Signals, Systems and Computers*, November 2014, pp. 470–474.

[85] R. H. Lambert, M. Joho, and H. Mathis, "Polynomial singular values for number of wideband source estimation and principal component analysis," in *Proc. Int. Conf. Independ. Compon. Analysis*, December 2001, pp. 379–383.

[86] "Polynomial matrix EVD toolbox." [Online]. Available: pevd-toolbox.eee.strath.ac.uk (Date last accessed May 29, 2019).

[87] J. G. F. Francis, "The QR transformation a unitary analogue to the LR transformation – part 1," *The Computer Journal*, vol. 4, no. 3, pp. 265–271, January 1961.

[88] Z. Wang, J. McWhirter, J. Corr, and S. Weiss, "Order-controlled multiple shift SBR2 algorithm for para-hermitian polynomial matrices," in *IEEE Workshop on Sensor Array and Multichannel Signal Processing*, July 2016.

[89] MathWorks®, "Matlab® vectorization." [Online]. Available: uk.mathworks.com/help/matlab/matlab_prog/vectorization.html (Date last accessed May 29, 2019).

[90] S. W. Smith, *The Scientist & Engineer's Guide to Digital Signal Processing*. California Technical Pub, 1997.

[91] MathWorks®, "Matlab® FFT functionality." [Online]. Available: https://uk.mathworks.com/help/matlab/ref/fft.html (Date last accessed May 29, 2019).

[92] M. Frigo and S. G. Johnson, "FFTW web page." [Online]. Available: http://www.fftw.org/ (Date last accessed May 29, 2019).

[93] ——, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, February 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[94] J. Tursa, "MTIMESX - fast matrix multiply with multi-dimensional support." [Online]. Available: https://uk.mathworks.com/matlabcentral/fileexchange/25977 (Date last accessed May 29, 2019).

[95] S. S. Skiena, *The Algorithm Design Manual*. London: Springer, 2008.

[96] A. Jafarian and J. McWhirter, "A novel method for multichannel spectral factorization," in *20th European Signal Processing Conference*, August 2012, pp. 1069–1073.

[97] C. Grozea, Z. Bankovic, and P. Laskov, "Facing the multicore-challenge." Berlin, Heidelberg: Springer-Verlag, 2010, ch. FPGA vs. Multi-core CPUs vs. GPUs: Hands-on Experience with a Sorting Application, pp. 105–117.

[98] B. Betkaoui, D. B. Thomas, and W. Luk, "Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing," in *2010 International Conference on Field-Programmable Technology*, December 2010, pp. 94–101.

[99] Victor Eijkhout with Robert van de Geijn and Edmond Chow, *Introduction to High Performance Scientific Computing.* lulu.com, 2011, http://www.tacc.utexas.edu/~eijkhout/istc/istc.html.

[100] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, "A performance and energy comparison of convolution on GPUs, FPGAs, and Multicore Processors," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 25:1–25:21, 2013.

[101] U. I. Minhas, S. Bayliss, and G. A. Constantinides, "GPU vs FPGA: A comparative analysis for non-standard precision," in *Reconfigurable Computing: Architectures, Tools, and Applications.* Springer International Publishing, 2014, pp. 298–305.

[102] G. Brassard and P. Bratley, *Fundamental of Algorithmics.* Prentice-Hall, 1996.

[103] C. A. R. Hoare, "Algorithm 64: Quicksort," *Commun. ACM*, vol. 4, no. 7, p. 321, Jul. 1961.

[104] J. J. M. Cuppen, "A divide and conquer method for the symmetric tridiagonal eigenproblem," *Numerische Mathematik*, vol. 36, no. 2, pp. 177–195, Jun 1980.

[105] J. J. Dongarra and D. C. Sorensen, "A fully parallel algorithm for the symmetric eigenvalue problem," *SIAM Journal on Scientific and Statistical Computing*, vol. 8, no. 2, pp. 139–154, 1987.

[106] D. Gill and E. Tadmor, "An $O(N^2)$ method for computing the eigensystem of $N \times N$ symmetric tridiagonal matrices by the divide and conquer approach," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 1, pp. 161–173, 1990.

[107] P. Pal and P. P. Vaidyanathan, "A novel autofocusing approach for estimating directions-of-arrival of wideband signals," in *Proc. Asilomar Conf. Signals, Systems and Computers*, November 2009, pp. 1663–1667.

[108] T. I. Laakso, V. Välimäki, M. Karjalainen, and U. K. Laine, "Splitting the Unit Delay," *IEEE Signal Processing Magazine*, vol. 13, no. 1, pp. 30–60, January 1996.

[109] J. Selva, "An efficient structure for the design of variable fractional delay filters based on the windowing method," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3770–3775, August 2008.

[110] C. H. Ta and S. Weiss, "A design of precoding and equalisation for broadband MIMO systems," in *Proc. 15th Int. Conf. Digital Signal Process.*, July 2007, pp. 571–574.

[111] M. J. D. Powell, "A new algorithm for unconstrained optimization," *Nonlinear programming*, pp. 31–65, 1970.

[112] J. Nocedal and S. Wright, *Numerical Optimization*. NY: Springer, 1999.

[113] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, no. 4, pp. 354–356, Aug. 1969.

[114] F. L. Gall, "Powers of tensors and fast matrix multiplication," in *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ACM, 2014, pp. 296–303.

[115] F. C. Chang, "Polynomial GCD derived through monic polynomial subtractions," *ISRN Applied Mathematics*, vol. 2011, 2011.

[116] S. Weiss and M. D. Macleod, "Maximally smooth dirichlet interpolation from complete and incomplete sample points on the unit circle," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2019.

[117] S. Weiss, I. K. Proudler, and M. D. Macleod, "Measuring smoothness of real-valued functions defined by sample points on the unit circle," in *Sensor Signal Processing for Defence Conference*, May 2019.

[118] J. Selva, "FFT interpolation from nonuniform samples lying in a regular grid," *IEEE Transactions on Signal Processing*, vol. 63, no. 11, pp. 2826–2834, June 2015.

[119] I. Dokmanić, Y. Lu, and M. Vetterli, "Can one hear the shape of a room: the 2-D polygonal case," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2011, pp. 321–324.

[120] A. H. Moore, M. Brookes, and P. A. Naylor, "Room geometry estimation from a single channel acoustic impulse response," in *21st European Signal Processing Conference*, September 2013.